Stateless Deterministic Multi-Party EdDSA Signatures with Low Communication

Qi Feng [*] Kang Yang [†] Kaiyi Zhang [‡] Xiao Wang [§]
Yu Yu [¶] Xiang Xie [‖] Debiao He [**]

February 28, 2024

Abstract

EdDSA, standardized by both IRTF and NIST, is a variant of the well-known Schnorr signature based on Edwards curves, and enjoys the benefit of statelessly and deterministically deriving nonces (i.e., it does not require reliable source of randomness or state continuity). Recently, NIST calls for multi-party threshold EdDSA signatures in one mode of deriving nonce statelessly and deterministically and verifying such derivation via zero-knowledge (ZK) proofs. Multi-party full-threshold EdDSA signatures in the dishonest-majority malicious setting have the advantage of strong security guarantee, and specially cover the two-party case. However, it is challenging to translate the stateless and deterministic benefit of EdDSA to the multi-party setting, as no fresh randomness is available for the protocol execution.

We present the notion of information-theoretic message authenticated codes (IT-MACs) over groups in the multi-verifier setting, and adopt the recent pseudorandom correlation function (PCF) to generate IT-MACs statelessly and deterministically. Furthermore, we generalize the two-party IT-MACs-based ZK protocol by Baum et al. (Crypto'21) into the multi-verifier setting, which may be of independent interest. Together with multi-verifier extended doubly-authenticated bits ($\mathsf{mv\text{-}edaBits}$) with errors, we design a multi-verifier zero-knowledge (MVZK) protocol to derive nonces statelessly and deterministically. Building upon the MVZK protocol, we propose a stateless deterministic multi-party EdDSA signature, tolerating all-but-one malicious corruptions. Compared to the state-of-the-art multi-party EdDSA signature by Garillot et al. (Crypto'21), we improve communication cost by a factor of $61\times$, at the cost of increasing computation cost by about $2.25\times$ and requiring three extra rounds.

1 Introduction

Edwards-curve digital signature algorithm (EdDSA) [BDL+11] is a highly efficient Schnorr-variant digital signature over twisted Edwards curves. It has been standardized by both NIST Federal Information Processing Standard (FIPS) for Digital Signatures [nis19] and IRTF Request for Comments 8032 [JL17] to facilitate its adoption in the Internet community. Unlike ECDSA or primitive Schnorr, EdDSA largely benefits from its deterministic signature structure, i.e., the nonce value is derived by $\boldsymbol{r} = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$ with a pseudorandom function PRF (standardized as $\mathsf{SHA512}$). Ignoring some encoding details, an EdDSA signature over a message is defined as $r = \sum_{i=1}^{\ell} \boldsymbol{r}[i] \cdot 2^{i-1} \mod q, R = r \cdot G$ and $\sigma = s \cdot \mathsf{H}(R, \mathsf{pk}, \mathsf{msg}) + r \mod q$.

[*]Wuhan University, fengqi.whu@whu.edu.cn
[†]State Key Laboratory of Cryptology, yangk@sklc.org
[‡]Shanghai Jiao Tong University, kzoacn@cs.sjtu.edu.cn
[§]Northwestern University, wangxiao@northwestern.edu
[¶]Shanghai Jiao Tong University & Shanghai Qi Zhi Institute, yuyu@yuyu.hk
[‖]PADO Labs & Shanghai Qi Zhi Institute, xiexiangiscas@gmail.com
[**]Wuhan University, hedebiao@163.com

Assuming the long-term dk has enough entropy and PRF generates pseudorandom output, $\boldsymbol{r}$ will be pseudorandomly and deterministically derived for each message msg. The EdDSA signature has been proven as secure as the original Schnorr scheme that consumes fresh random for each signature.

Multi-party signing. Threshold signature (primarily for ECDSA and Schnorr) was studied in the late 1980s and the 1990s, c.f. [BN06, Des88, DF90, GJKR96, SG98, Sho00, MR01, MOR01, NKDM03], but has recently gained a lot of interest due to its application to the key protection [Lin17, DKLs18, LN18, GG18, DKLs19, CGG+20, YCX21, ANO+22, KOR23, GKMN21]. Generally, a t-out-of-n threshold signature allows n parties to secretly share the private key with a property that any subset of t parties can jointly sign a message while preventing any subset of less than t parties from doing so. Multi-party signature protocol is a special case of n-out-of-n, also called full-threshold signature. It can protect the secret key for any $t < n$ corruptions, making sense in some core areas by keeping the key shares in independent devices or emulating multiple signers on a proposal, specifically covering the two-party case. Recently, NIST [BP23] published the first call for multi-party threshold cryptography and made a specific comment on EdDSA signing. Therefore, in this work, we aim to translate the benefit of the multi-party setting to EdDSA signature.

Multi-party EdDSA signing. The core challenge in multi-party EdDSA signing is the nonlinear PRF-based nonce derivation operation. The first attempt is to combine the secret r with all parties' "deterministic" nonce contributions. Unfortunately, it was broken soon by Maxwell et al. [MPSW19]. Consider two executions to sign the same message msg, malicious adversary $\mathcal{A}$ can launch the forking attack to extract the honest party's secret key: in the first session, $\mathcal{A}$ runs the honest signing program with correct r_i and obtains the correct transcripts from honest party P_j:

$$R = R_i + R_j, e = \mathsf{H}(\mathsf{pk}, R, \mathsf{msg}), \sigma_j = s_j \cdot e + r_j \tag{1}$$

Then it cheats the honest party to sign the same message msg with different nonce r_i^*. The honest party still uses the same r_j (as the long-time derived key and message are identical in both sessions). Therefore, $\mathcal{A}$ obtains another set of transcripts as:

$$R^* = R_i^* + R_j, e^* = \mathsf{H}(\mathsf{pk}, R^*, \mathsf{msg}), \sigma_j^* = s_j \cdot e^* + r_j \tag{2}$$

At this time, the adversary can easily extract the secret key share s_j via Eq.(1) and Eq.(2).

Why Stateless? Folklore would suggest that the problem is simple: invoking the PRF with a fresh counter to "simulate" a random tape. Cryptographically, this strategy is secure if the counter is well maintained. However, this approach inherently relies on a *state continuity* may not be cheaper than reliability storing the long-time secret key. Parno et al. [PLD+11] found that even secure hardened devices with strong isolation guarantees can not take this property for granted. Malicious attackers or circumstance mistakes (software errors, power interruptions, etc.) may lead to a high delay or bad randomization reuse. Therefore, Bonte et al. [BST21] made another attempt to *exactly* evaluate the nonce derivation function with a prepossessed secretly-shared seed by MPC protocol in the *honest majority* setting. Their work does not require a continuous state, yet is limited by non-deterministic processing and expensive costs.

Why Deterministic? The fact that uniformly random chosen during the signing process probably introduces an attack hock in practice, because a consistent source of entropy being available for use has repeatedly turned out to be ill-founded. For example, Kumari et al. [KASN15] given the detailed random number generator (RNG) experiments in different settings. Their results indicate that a guest running in a virtual machine tends to generate entropy at a lower rate and that it may be possible to predict the RNG output by increasing the disk activity rate on the guest. Consequently, even a slight noticeable deviation of the chosen value from uniform distribution can be used to break the scheme [HGS01, ANT+20, DH20]. Therefore,

deterministic nonce generation has gained traction [MNPV99, KW03]. Kondi et al. [KOR23] proposed a two-round stateless deterministic two-party Schnorr signature using the pseudorandom correlation function (PCF). Although their work achieves high performance, the randomness derived from the PCF part does not meet the PRF function in the EdDSA standard.

Following the type-II comment on multi-party threshold signing in NIST document [BP23], where pseudorandom per quorum via a zero-knowledge proof of pseudorandom contribution per party, a line of works have been promised to build the deterministic, stateless multi-party signing based on zero-knowledge proof [NRSW20, GKMN21]. Nick et al. [NRSW20] proposed a two-round multi-signature scheme (MuSign-DN). They opt for Bulletproofs [BBB+18] to prove the correctness of nonce derivation with roughly a kilobyte of proof but incurring high computation costs. Besides, MuSign-DN does not support EdDSA signature. The closest work to ours is that of Garillot et al. [GKMN21]. They partially addressed these issues by incorporating the zero-knowledge from the garbled circuit (ZKGC) [JKO13] paradigm. Garillot et al. consider the dishonest majority adversarial model and achieve provable security based on the DDH assumption over a standardized elliptic curve. Still, its applicability is somewhat limited by heavy communication overheads.

1.1 Our Contributions

Protocol	PRF in EdDSA	Asymptotic Comm.	Concrete Comm.	Rounds	Assumptions
[CGM16]	yes	$O(\|\mathcal{C}\| \cdot \kappa + s \cdot \|q\| \cdot \kappa)$	1.94 MB	1	PRG+Hash+DH
[NRSW20]	no	$O(\|q\|)$	1.1 KB	2	RO + DDH
[GKMN21]	yes	$O(\|\mathcal{C}\| \cdot \kappa + \|q\| \cdot \kappa)$	1.01 MB	3	RO+PRF
[KOR23]	no	$O(\ell' - \|q\| + 2\kappa)$	0.87 KB	1	RO+DCR+Strong RSA
This work	yes	$O(\|\mathcal{C}\| + (\log(\|\mathcal{C}\|) + \ell) \cdot \kappa)$	16.33 KB	4	LPN+RO

Table 1: Comparing two-party verifiable nonce derivation protocol with prior related works. Costs to prove $R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G$. The offline phase includes the input and preprocessing phases. Concrete costs are given for $\ell' = 3597$, $|q| = 256$, $\kappa = 128$, $\ell = 512$, $|\mathcal{C}| = 58k$ for SHA-512 as standard PRF for EdDSA [nis19].

Protocol	Stateless and deterministic	Asymptotic Comm.	Concrete Comm.	Rounds	Corruptions	Type
[BST21]	no	$O(n \cdot \|\mathcal{C}\| \cdot \kappa)$	2294.3 MB	9	$t < n/2$	IV
[GKMN21]	yes	$O(n \cdot (\|\mathcal{C}\| \cdot \kappa + \|q\| \cdot \kappa))$	5.37 MB	3	$t < n$	II
This work	yes	$O(n \cdot (\|\mathcal{C}\| + (\log(\|\mathcal{C}\|) + \ell) \cdot \kappa))$	49.19 KB	6	$t < n$	II

Table 2: **Comparison of multi-party EdDSA signature protocols.** We do not consider the key generation phase as it is executed only once. Costs are given for the Ed25519 curve among *three parties*. Type IV is functionality equivalent to HashEdDSA via MPC hashing and type II is the mode of pseudo-random per quorum via a ZKP commented by the NIST document

In this paper, we present a multi-party stateless and deterministic EdDSA that is secure in the presence of a malicious adversary corrupting any number of parties. Our circuit-based zero-knowledge proof (in the

multi-verifier setting) for standard nonce derivation only needs to send few bits per multiplication gate per party and take constant rounds of communication.

IT-MACs over Group in the Multi-Verifier Setting. In this paper, we consider the information-theoretic message authenticated codes (IT-MACs) over group for verifiable nonce derivation. According to the definition of Smart et al. [STA19], for a group element $X \in \mathbb{G}$, its IT-MAC correlation for two-party setting enforces that $[\![X]\!]_q := \{(X, M), K\}$ of the form $M = K + \Lambda \cdot X$. We extend it to a multi-verifier IT-MACs as $[\![X]\!]_q := \{(X, \{M_i\}_{i\in[N]}), \{K_i\}_{i\in[N]}\}$ of the form $M_i = K_i + \Lambda^i \cdot X$. Besides, Smart et al. [STA19] leaves an open problem to construct the IT-MACs from bit vectors of the form $\boldsymbol{m} = \boldsymbol{k} + \boldsymbol{x} \cdot \Delta \in \mathbb{F}_{2^\kappa}$. We fill this gap by extending the edaBits [EGK+20] in the multi-verifier setting. This paradigm could be of independent interest. Specifically, for input wires $\boldsymbol{r}[1], \ldots, \boldsymbol{r}[\ell]$, they will be masked with the boolean parts of edaBits using a ℓ-bit adder circuit, followed by revealing $c = r + \rho \mod q$. This enables each party to locally subtract and obtain $[\![r]\!]_q$, which can be used as the authenticated context over the group for element $[\![R]\!]_q := [\![r]\!]_q \cdot G$. The communication complexity is roughly $O(|\mathsf{Gen.edaBits}| + |\mathcal{C}_{\mathsf{add}}|)$ with constant rounds.

Stateless Deterministic Nonce Derivation. Table 1 summarizes a comparison of two-party deterministic, stateless verifiable nonce derivation. Both garbling protocols of Chase et al. [CGM16] and Garillot et al. [GKMN21] are achieved by a garbled circuit with the output of a bit string $\boldsymbol{r}$ (i.e., nonce $\boldsymbol{r} := \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$) and a transformation gadget with the output of an algebraic encoding of value $\sum_{i=1}^{\ell}(2^{i-1} \cdot \boldsymbol{r}[i]) \cdot G$. However, their constructions by garbling an explicit PRF circuit are costly. Kondi et al. [KOR23] takes another PCF-style approach of having much better performance. However, their nonce derivation method in the malicious setting is $r = \mathsf{PCF.Eval}(\mathsf{PCF.Key}, \mathsf{msg})$ for each party with a specific PRF function, leaving a gap to the standard definition of EdDSA, i.e., $\boldsymbol{r} = \mathsf{SHA512}(\mathsf{dk}, \mathsf{msg})$. Our verifiable nonce derivation protocol develops on top of LPN-based PCF [BCG+22] and edaBits [EGK+20], offering asymptotically $O(\kappa)\times$ improvement with comparable round complexity and PRF in the standard EdDSA definition.

Multi-Party EdDSA Signing. Table 2 summarizes a comparison of multi-party EdDSA signing. Consider the Ed25519 version of EdDSA, which aims at the security level of $\kappa = 128$ bits, and sets $|q| = 256$ bits and PRF as SHA512 (i.e., $\ell = 512$). Bonte et al. [BST21] securely evaluate the PRF function of EdDSA by MPC protocol in the *honest majority* setting, costing constant round complexity during the generation of daBits. Another zero-knowledge proof-based multi-party EdDSA signing protocol presented by Garillot et al. [GKMN21] also has constant rounds. However, Garillot et al. [GKMN21] relies on the garbled circuit between each pair of parties and takes higher communication overheads. This work follows Garillot et al. [GKMN21] to securely prove SHA512 in a zero-knowledge manner, we handle the gate-by-gate circuit by n bit per AND gate per party. For the correctness check of multiplication triples, we extend the Mac'n'Cheese [BMRS21] to the *multi-verifier* setting, costing $O(n \cdot \log(t) \cdot \kappa)$ bits for each party to send. An important note is that our approach can also be applied to deterministic ECDSA in the secret-shared-input model, where the hash e of the message is secret-shared by the client.

1.2 Road-map

The remainder of this paper is organized as follows. Section 2 establishes the technical overview. Section 3 presents the preliminaries, such as the security definition and cryptographic primitives. Section 4 shows how to construct multi-verifier verifiable nonce derivation and to build a communication-efficient multi-party EdDSA signing. Section 5 will present a performance evaluation. Finally, Section 6 concludes this paper.

2 Technical Overview

Notations 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. 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 $\mathbb{Z}_q$.

This section provides a technical overview of our work. The full descriptions and security proofs are left in the later section. To give an overview, we define the IT-MACs over group and adopt them to design a multi-party stateless deterministic EdDSA signing protocol. We provide a multi-verifier verifiable nonce derivation protocol with optimal communication performance.

2.1 Multi-Verifier IT-MACs over Group

We use multi-verifier *information-theoretic message authenticated codes (IT-MACs)*, originally proposed for two-party computation. We authenticate values in $\mathbb{F}_2$, and the authentication is done over the binary extension field $\mathbb{F}_{2^\kappa}$. Specifically, let $\mathcal{V}_1, \ldots, \mathcal{V}_N$ be N verifiers and $\Delta^i \in \mathbb{F}_{2^\kappa}$ be a uniform *global key* known only to $\mathcal{V}_i$. A value $x \in \mathbb{F}_2$ known by the prover $\mathcal{P}$ is authenticated by

$$[\![x]\!]_2 = \{(x, m_1, \ldots, m_N), k_1, \ldots, k_N\},$$

satisfying $m_i = k_i + x \cdot \Delta^i \in \mathbb{F}_{2^\kappa}$ with the same x. Each $\mathcal{V}_i$ holds a *local MAC key* $k_i, i \in [N]$, and $\mathcal{P}$ holds the secret value and corresponding *MAC tags* $(x, m_1, \ldots, m_N)$. Besides, we extend the above notation to vectors, arithmetic, or group of authenticated values as well. In this case:

- $[\![\boldsymbol{x}]\!]_2 = \{(\boldsymbol{x}, \boldsymbol{m}_1, \ldots, \boldsymbol{m}_N), \boldsymbol{k}_1, \ldots, \boldsymbol{k}_N\}$ means 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 local key $\Delta^i \in \mathbb{F}_{2^\kappa}$ and local MAC key $\boldsymbol{k}_i \in \mathbb{F}_{2^\kappa}^\ell$ with $\boldsymbol{m}_i = \boldsymbol{k}_i + \Delta^i \cdot \boldsymbol{x} \in \mathbb{F}_{2^\kappa}$.
- $[\![x]\!]_q = \{(x, m_1, \ldots, m_N), k_1, \ldots, k_N\}$ means that $\mathcal{P}$ holds $x, m_1, \ldots, m_N \in \mathbb{Z}_q$ while $\mathcal{V}_i$ holds *glocal key* $\Lambda^i \in \mathbb{Z}_q$ and *local MAC key* $k_i \in \mathbb{Z}_q$ with $m_i = k_i + \Lambda^i \cdot x \mod q$.
- $[\![X]\!]_q = \{(X, M_1, \ldots, M_N), K_1, \ldots, K_N\}$ means 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 $K_i \in \mathbb{G}$ with $M_i = K_i + \Lambda^i \cdot X$.

All authenticated values are additively homomorphic. For example, 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 calculate $[\![y]\!]_2 = \sum_{i=1}^{\ell} c_i \cdot [\![x_i]\!]_2 + c$ locally. Here, we give the definition of macros used in this paper as follows.

Shr. On input $\boldsymbol{x} \in \mathbb{F}_2^\ell$ from $\mathcal{P}$, it first encodes $\boldsymbol{x}$ into a public value v with secret code β_x. In $\mathsf{PCF.Gen}_{\mathsf{mv}}$, instead of using random β, we now input the β_x which will be used to define the secret value $\boldsymbol{x}$. Then, all the parties execute $\mathsf{PCF.Eval}_{\mathsf{mv}}$ and obtain $[\![\boldsymbol{x}]\!]_2$ as a linear combination of the output (c.f. Figure 12 of [BCG+22] for more details). We denote this sharing procedure using $\mathsf{Shr}(x)$. Remark that Shr macro is only used in the key generation with a reliable source of randomness.

Random. Generate an authenticated value $[\![r]\!]_2$, where $r \in \mathbb{F}_2$ is a uniformly random value. This can be achieved by $\mathcal{P}$ and $\mathcal{V}$s invoke $\mathsf{PCF.Eval}_{\mathsf{mv}}$ (c.f. Figure 8) with prepared k_0 and k_i for $i \in [N]$, respectively. We use Random to denote this subroutine.

Assign. On input $x \in \mathbb{F}_2$ from $\mathcal{P}$, $\mathcal{P}$ and $\mathcal{V}$s execute $[\![r]\!]_2 \leftarrow \mathsf{Random}$. Then $\mathcal{P}$ sends $y = r - x \in \mathbb{F}_2$ to $\mathcal{V}$s and all parties computes $[\![x]\!]_2 = [\![r]\!]_2 + y$. We denote this assigning procedure using $\mathsf{Assign}(x)$.

Checking Zero. An authenticated value $[\![x]\!]_2$ can be checked if $x = 0$ by having $\mathcal{P}$ send m_i to corresponding verifier $\mathcal{V}_i$, who verifies if $m_i = k_i$ holds. We use $\mathsf{CheckZero}([\![x]\!]_2)$ to denote this checking procedure.

Remark that we do not need broadcast here, which means malicious $\mathcal{P}$ might send correct (m_i) to $\mathcal{V}_i$ and send incorrect (m_j) to $\mathcal{V}_j$, that is $m_i = k_i$, but $m_j \neq k_j$, causing $\mathcal{V}_i$ continues yet $\mathcal{V}_j$ aborts. All verifiers could fix this by announcing their response and checking consistency. As long as one verifier is honest, this inconsistency will be found. Since the signature scheme is verifiable, we retrench the broadcast channels here. Therefore, the security is following the two-party IT-MACs and PCF.

2.2 Stateless Deterministic Nonce Derivation in the Multi-Verifier Setting

The previous section introduces why the stateless and deterministic manner is essential. Recent works [NRSW20, GKMN21, KOR23] have contributed to the zero-knowledge proof-based pattern. These works all follow the same paradigm below.

1. The prover $\mathcal{P}$ commits to the same nonce derivation key dk in the key generation phase, which is in the form of verifiable commitments. Specifically, $\mathcal{P}$ commits to each bit of dk, while verifier $\mathcal{V}$ (i.e., the other party who checks the correctness of nonce derivation) keeps authentication keys.

2. Subsequently, $\mathcal{P}$ proves an unbounded number of statements (i.e., correctly PRF and exponentiation evaluation circuit) in the signing phase. Specifically, $\mathcal{P}$ and $\mathcal{V}$ jointly evaluate the target circuit gate-by-gate while masking the input with committed derived keys. If $\mathcal{P}$ uses the correct dk, they must open to the same nonce $R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G$.

Unlike prior works, we want to prove the nonce derivation against multiple verifiers simultaneously.

In our multi-verifier verifiable nonce derivation protocol, each wire value is secretly shared in the multi-verifier IT-MACs. Therefore, ADD gates can be calculated locally by each party and MULT gates are jointly processed by invoking the $\mathsf{Assign}(\omega_\alpha \cdot \omega_\beta)$ procedure. Finally, all parties obtain t multiplication gates with wire values $(\omega_{\alpha,j}, \omega_{\beta,j}, \omega_{\gamma,j}), j \in [t]$. They can prove the correctness by randomized linear combination on $\omega_{\alpha,j} \cdot \omega_{\beta,j} - \omega_{\gamma,j} = 0$. Here, we follow the polynomial-based batch verification technique [BMRS21] and extend their work to the multi-verifier setting. Let's first arrange 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. In particular, if $\mathcal{P}$ is honest, it will define $\frac{t}{2}$ polynomials for all α-wires as $f_1, \dots, f_{\frac{t}{2}}$ and another $\frac{t}{2}$ polynomials for all β-wires as $g_1, \dots, g_{\frac{t}{2}}$. 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 a product polynomial as $h = \sum_{i\in[\frac{t}{2}]} f_i \cdot g_i \in \mathbb{F}_{2^\kappa}[X]$. If all multiplication triples are correct, the aggregation of outputs must be a point of h. $\mathcal{P}$ further commits to the coefficients of $h(\cdot)$ to all the verifiers. At this time, all parties could jointly check if $\sum_{j\in[2]} [\![h(j)]\!]_2 - [\![z]\!]_2$ is $[\![0]\!]_2$. This is achieved by invoking $\mathsf{CheckZero}$ subroutine.

To complete the proof, $\mathcal{P}$ also needs to demonstrate that the commitment on polynomial h is exactly the inter-product of $f_1, \dots, f_{\frac{t}{2}}$ and $g_1, \dots, g_{\frac{t}{2}}$ The key insight is that all parties can use the IT-MACs $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2\}_{i\in[t]}$ to homomorphically derive the authenticated sharing of those $\frac{t}{2}$ polynomials, i.e., $[\![f_1]\!]_2, \dots, [\![f_{\frac{t}{2}}]\!]_2$ and $[\![g_1]\!]_2, \dots, [\![g_{\frac{t}{2}}]\!]_2$, a further check on these commitments is performed as

$\sum_{i\in[\frac{t}{2}]}[\![f_i]\!]_2 \cdot [\![g_i]\!]_2 - [\![h]\!]_2 = [\![\tilde{0}]\!]_2$, where $\tilde{0}$ denotes a zero-polynomial that is *always* evaluated to 0. By Schwartz-Zippel, this can be done by checking that

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

with a random $\eta \in \mathbb{F}_{2^\kappa}$. To keep deterministic, we generate η using Fiet-Shamir heuristic. Observe that the equivalent verification of $[\![f_1(\eta)]\!]_2, \ldots, [\![f_{\frac{t}{2}}(\eta)]\!]_2$, $[\![g_1(\eta)]\!]_2, \ldots, [\![g_{\frac{t}{2}}(\eta)]\!]_2$ and $[\![h(\eta)]\!]_2$ evaluated on a public value η boils down to another $\frac{t}{2}$-batched verification on multiplication triples. Thus, we can recursively apply to rearrange the triples, multiplying the two polynomials, committing to the inter-product polynomial, and checking until one or two triples are left. The last multiplication triples could be efficiently verified using the sacrifice technique [KOS16]. If any check fails, the party's output is false. Otherwise, all parties get a correct vector $[\![\boldsymbol{r}]\!]_2$ that is the evaluation result of derivation function $\mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$.

Smart et al. [STA19] defined an authenticated secret sharing over group, i.e., for an elliptic curve point $R \in \mathbb{G}$ with $R = r \cdot G$, it will be secretly shared via a correlation $\{R_i, M_i\}$ for $i \in [n]$ such as $R = \sum_{i\in[n]} R_i$ and $\sum_{i\in[n]} M_i = \Lambda \cdot R$, where $\Lambda \in \mathbb{Z}_q$ is the same global key as used in $[\![r]\!]_q$. We extend their work by converting $[\![\boldsymbol{r}]\!]_2$ to $[\![R]\!]_q$. The core challenge is that $\boldsymbol{r} \in \mathbb{F}_2^\ell$ are secretly-shared over $\mathbb{F}_{2^\kappa}$, while $[\![R]\!]_q$ and $[\![r]\!]_q$ are secretly-shared over $\mathbb{Z}_q$. Inspired by prior work [BST21], we use the idea of extended doubly-authenticated bits (edaBits) [EGK+20]. We extend original edaBits to the multi-verifier friendly form, i.e., $\mathsf{mv\text{-}edaBits} := \{([\![\rho]\!]_q, [\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2)\}$ such that the random value $\rho \in \mathbb{Z}_q$ is secret-shared over the arithmetic field *in the multi-verifier setting* and its binary representations (i.e., $\rho = \sum_{j=1}^{\ell} 2^{j-1}\boldsymbol{\rho}[j] \mod q$) are secret-shared over the boolean field *also in the multi-verifier setting*. An important observation is that when $\mathcal{P}$ is corrupted, we allow the adversary to choose different ρ. That is, we do not require the consistency check of edaBits, which is heavily expensive and performed by cut-and-choose technique. These are sufficient to design multi-party signing protocols in the malicious setting as the signature is inherently verifiable. Therefore, we can efficiently generate the edaBits by calling two times $\mathsf{PCF.Eval}_{\mathsf{mv}}$, implemented with $\mathbb{F}_2$ and $\mathbb{Z}_q$ respectively. Therefore, incorporated with an addition circuit, $\mathcal{P}$ and $\mathcal{V}$s can evaluate the PRF circuit with inputs of $[\![\boldsymbol{r}[1]]\!]_2, \ldots, [\![\boldsymbol{r}[\ell]]\!]_2$ and $[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2$, opening to the result of $c = r + \rho \mod q$ in clear. We could naturally define the IT-MACs over group by $[\![R]\!]_q = (c - [\![\rho]\!]_q) \cdot G$.

Suppose a cheating $\mathcal{P}$ does not use a consistent r and ρ. This can be repaired by adding a batch verification. Specifically, when n parties get $\{[\![R_1]\!]_q, \ldots, [\![R_n]\!]_q\}$ with $\{[\![\rho_1]\!]_q, \ldots, [\![\rho_n]\!]_q\}$, they jointly open $\{R_1, \ldots, R_n\}$ and

$$\begin{aligned} S &= (\sum_{i\in[n]} \chi_i \cdot c_i) \cdot G = (\sum_{i\in[n]} \chi_i \cdot r_i + \chi_i \cdot \rho_i) \cdot G \\ &= \sum_{i=1}^{n} \chi_i \cdot R_i + (\sum_{i\in[n]} \chi_i \cdot \rho_i) \cdot G \end{aligned}$$

where $\chi_1, \ldots, \chi_n \in \mathbb{Z}_q$ are common random values generated using Fiet-Shamir heuristic and check if

$$\begin{aligned} \sum_{i\in[n]} (\Lambda^i \cdot S) = \Lambda \cdot \sum_{i=1}^{n} \chi_i \cdot R_i \\ + \sum_{i\in[n]} (\sum_{j\neq i} (\chi_j \cdot k_\rho^{j,i} + \chi_i \cdot m_\rho^{i,j}) + \chi_i \cdot \rho_i \cdot \Lambda^i) \cdot G \end{aligned}$$

holds, where for $[\![\rho_i]\!]_q$ of P_i, its corresponding MAC tags are $m_\rho^{i,j} \in \mathbb{Z}_q$ and local MAC keys are $k_\rho^{i,j} \in \mathbb{Z}_q$ for all $j \in [n], j \neq i$. This is workable because if the equation holds, we can obtain $(\Lambda \cdot \sum_{i=1}^{n} \chi_i \cdot c_i -$

$\sum_{i,j}(\chi_i \cdot \rho_i \cdot \Lambda^j)) \cdot G = \Lambda \cdot \sum_{i=1}^{n} \chi_i \cdot R_i$. That is $(\sum_{i=1}^{n} \chi_i \cdot (\rho_i + r_i) - \sum_{i=1}^{n} \chi_i \cdot \rho_i) \cdot G = \sum_{i=1}^{n} \chi_i \cdot R$ (the red ρ_i is aggregated by binary representations of mv-edaBits and the blue ρ_i is generated from integer part of mv-edaBits). If Λ is uniformly random and $\chi_1, \ldots, \chi_n \in \mathbb{Z}_q$ are computationally unpredictable, the advantage of $\mathcal{A}$ to forge an inconsistent mv-edaBits will be $\mathsf{negl}(\kappa)$. After all n signers prove their derived nonce acting as $\mathcal{P}$, they obtain the correct nonce from others if no failure happens.

2.3 Multi-Party EdDSA Signing

We concentrate on the Ed25519 version of the EdDSA signature, with the case of Ed448 being virtually identical (except using a different elliptic curve). Parameterized by the EdDSA standard parameters, the multi-party EdDSA signing could be achieved in two phases. In the key generation phase, each party generates all the keys, i.e., secret key sk_i, random seed k_i^*, derived key dk_i, global keys Δ^i and Λ^i. Furthermore, the signing public key is easy to construct based on the additive cycle group.

Given the message msg, a core step is to derive signing session id as $\mathsf{sid} = \mathsf{H}_0(\mathsf{msg}, R)$ and all the randomness needed by $\mathsf{H}_1(k_i^*, \mathsf{sid}, \mathsf{MVND})$ in a deterministic, stateless manner. Then the verifiable nonce derivation could be performed as follows:

1. Each P_i proves its R_i acting as the prover $\mathcal{P}$ while all other P_j, $j \in [n] \backslash \{i\}$, acting as the verifiers $\mathcal{V}$s.
2. P_i aborts if it finds any false in the multi-verifier nonce derivation process. Otherwise, P_i obtains R.

The remaining steps are easy. Each P_i computes $h = \mathsf{H}_{\mathsf{sig}}(R, \mathsf{pk}, \mathsf{msg})$ and $\sigma_i = r_i + h \cdot s_i \mod q$. All parties open the $\sigma = \sum_{i=1}^{n} \sigma_i \mod q$. Finally, each party checks if $\mathsf{verify}(\mathsf{pk}, \mathsf{msg}, (R, \sigma)) = \mathsf{false}$, then it aborts. Otherwise, the parties output (R, σ). Thus, the computation and communication costs heavily depend on the multi-verifier nonce derivation.

3 Preliminaries

This section presents the preliminaries used in the multi-party EdDSA signing protocol. We first review the universal definitions, such as UC security model, commitment functionality $\mathcal{F}_{\mathsf{Com}}$, and committed NIZK functionality for DLP $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$. Then we present the definitions of EdDSA schemes and PCF.

3.1 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.

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 $(\mathsf{deliver}, i)$ 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.

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

To realize threshold EdDSA signing, we use an ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$, formally defined in Figure 1. 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 simply defining $\mathsf{Com}(x) = \mathsf{H}(x)$ for high-entropy input x, where $\mathsf{H}(\cdot) : \{0,1\}^* \to \{0,1\}^\kappa$ is a cryptographic hash function with security parameter κ. This has no impact on security, as the simulation of commitments and the extraction of inputs still work in the random-oracle model.

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 $(\mathsf{decommit}, sid, i)$ 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.

Figure 1: The Commitment Functionality

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

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.

Figure 2: The Committed NIZK Functionality for DL Relation

Figure 2 overview a description of the committed non-interactive zero-knowledge proof functionality for discrete logarithm relation, denoted as $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$. The NIZK proof of knowledge in the random-oracle model can be achieved following many multi-party signing protocols such as [Lin17, LN18, DKLs18, DKLs19]. In this paper, we apply the standard Schnorr proof [Sch91] to prove knowledge of the discrete logarithm of an elliptic-curve point, which is shown as follows:

- **Prove:** Given a statement $Q = w \cdot G$ and a witness w, sample $r \leftarrow \mathbb{Z}_q$, compute $R = r \cdot G$, $c = \mathsf{H}(G, Q, R)$ and $s = r + c \cdot w \mod q$. Output a proof $\pi = (s, R)$.
- **Verify:** Given a statement $(\mathbb{G}, q, G, Q)$ and a proof $\pi = (s, R)$, compute $c' = \mathsf{H}(G, Q, R)$ and accept the proof if and only if $s \cdot G = R + c' \cdot Q$.

Here, $\mathsf{H}(\cdot) : \{0,1\}^* \to \mathbb{Z}_q$ denotes a cryptographic hash-to-integer function. This protocol can be transformed to the non-interactive version using Fiat-Shamir heuristic [FS87]. Combining above instantiation for $\mathcal{F}_{\mathsf{Com}}$ with Schnorr proof, we can obtain an efficient instantiation for functionality $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$, which will

be used in our *key generation* phase of the multi-party EdDSA signing protocol. Note that "high-entropy random source" is available in the key generation phase of EdDSA, therefore, the above instantiation does not impact our contribution.

3.4 EdDSA Signature Algorithm

This section introduces details of EdDSA. Following the standards of NIST and IRTF [JL17, nis19], there are two variants of EdDSA, depending on how the randomness is generated: the first case is $r = \mathsf{PRF}(\mathsf{dk}, \mathsf{msg})$ while the second case is $r = \mathsf{PRF}(\mathsf{dk}, \mathsf{H}(\mathsf{msg}))$. This paper focuses on the second case, as the input of PRF circuit is flexible in the first case. For the sake of readability, we write $\mathsf{H}(\mathsf{msg})$ as simple msg, which has few impact on a publicly known message. In addition, there are two versions of EdDSA, based on the Edwards curves Ed25519 and Ed448 respectively. We only consider the Ed25519 curve, as mostly implementations for EdDSA-based applications adopt this curve. In addition, we can easily modify our multi-party EdDSA signature protocol to the Ed448 curve. Figure 3 presents the detailed three algorithms of EdDSA scheme.

Algorithm EdDSA

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 private key $\mathsf{sk} \leftarrow \{0,1\}^{\ell_b}$, and compute a hash value $(\boldsymbol{h}[1], \boldsymbol{h}[2], \ldots, \boldsymbol{h}[2\ell_b]) := \mathsf{PRF}(\mathsf{sk})$.
2. Assign $\boldsymbol{h}[1] = \boldsymbol{h}[2] = \boldsymbol{h}[3] = \boldsymbol{h}[\ell_b] = 0$ and $\boldsymbol{h}[\ell_b - 1] = 1$. 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).

Figure 3: The EdDSA Scheme

3.5 Pseudorandom Correlation Function

Pseudorandom correlation function (PCF) was originally presented by Boyle [BCG+20] using LPN assumption and has been studied since than [BCG+22, CD23]. It incrementally allows the local generation of an arbitrary polynomial amount of pseudorandom correlations on demand from a pair of short correlated keys. Below, we provide the definitions for 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* 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 give the distribution for VOLE correlations if $\mathcal{Y}(1^\kappa, \Delta)$ samples $x \leftarrow \mathbb{F}, k \leftarrow \mathbb{F}$, computes $m = k + x \cdot \Delta \in \mathbb{F}$ 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)*;*
- $\mathsf{PCF.Eval}(b, k_b, v)$ *is a deterministic polynomial-time algorithm that on input* $b \in \{0,1\}$*, key* k_b *and input value* $v \in \{0,1\}^{n(\kappa)}$*, outputs a value* $y_b \in \{0,1\}^{\tau_b(\kappa)}$

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

- **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 Figure 4 and Figure 5 (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 Figure 6 and Figure 7 (again, with $Q(\kappa)$ samples).

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

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

$v^i \leftarrow \{0,1\}^{n(\kappa)}$

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

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

Figure 4: Correlated outputs of the $\mathcal{Y}$-function

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

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

$v^i \leftarrow \{0,1\}^{n(\kappa)}$

for $b \leftarrow \{0,1\}$: $y_b^{(i)} \leftarrow \mathsf{PCF.Eval}(b, k_b, v^{(i)})$

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

Figure 5: Pseudorandom $\mathcal{Y}$-correlated outputs of a PCF

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

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

$v^i \leftarrow \{0,1\}^{n(\kappa)}$

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

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

Figure 6: Output distributions of a PCF

3.6 Multi-Verifier Programmable PCF

The PCF as described above works for the two-party setting. In the multi-verifier setting, $\mathcal{P}$ would generate a VOLE correlation with every verifier $\mathcal{V}_i$, for $i \in [N]$ such that $\mathcal{P}$ obtains the same $\boldsymbol{x} \in \mathbb{F}^\ell$ and $\mathcal{V}_i$ obtains the same $\Delta^i \in \mathbb{F}$ among all VOLE correlations. Note that the existing PCF schemes satisfy *programmability* defined in [BCG+22], i.e., PCF.Gen takes additional inputs $(\boldsymbol{x}, \Delta^i)$ and outputs a pair of seeds that are expanded to a VOLE correlation with fixed $\boldsymbol{x}$, Δ^i. Based on the programmability, we are able to construct PCF for VOLE in the multi-verifier setting (see [BCG+22] for details). Building upon multi-verifier PCF for VOLE, we show the construction of a PCF scheme ($\mathsf{PCF.Gen}_{\mathsf{mv}}$, $\mathsf{PCF.Eval}_{\mathsf{mv}}$) of

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

Figure 7: Output distributions with RSample algorithm as in Definition 1

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, $\mathsf{PCF.Gen}_{\mathsf{mv}}\ (1^\kappa)$ executes as follows:

1. Sample $\boldsymbol{u} \leftarrow \mathbb{F}^\ell$ and $\boldsymbol{\beta} \leftarrow \mathbb{F}^\ell$. For each $i \in [N]$, sample $\Delta^i \leftarrow \mathbb{F}_p$.
2. For each $i \in [N]$, run $\mathsf{PCF.Gen}(1^\kappa, \boldsymbol{u}, \boldsymbol{\beta} \cdot \Delta^i)$ 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 $\mathcal{V}_i$. Then, output $k_0 := \{\boldsymbol{u}, \boldsymbol{\beta}, k_0^i\}_{i\in[N]}$ 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 $\boldsymbol{m}^{(i)} = \boldsymbol{k}^{(i)} + \boldsymbol{x} \cdot \Delta^i$.

Figure 8: Multi-verifier PCF scheme

multi-verifier authenticated sharing as defined in Section 2.1, while guaranteeing the security.

We can further optimize the PCF in a client-server model, where the key management servers escrow the client's key and the client executes $\mathsf{PCF.Gen}_{\mathsf{mv}}$ (even the multi-party key generation protocol) for the servers. It is a trending service in the lightweight internet (e.g., internet of things or mobile internet) with the features of availability and elasticity. Many internet companies have recently released key management services, such as Google, Microsoft, Apple and Alibaba, to help encrypt and protect sensitive data assets. Benefiting from the stateless and deterministic advantages, our protocol may be a promising proposal as the deployment costs, specifically state synchronization and high-entropy random number generator are unnecessary for the key management servers.

4 The Designed Protocols

Recall that we have defined multi-party IT-MACs over groups in Section 2.1. Thus, this section directly shows the detailed multi-verifier friendly extended doubly-authenticated bits (mv-edaBits) nonce derivation

protocol. In Section 4.3, we present the practical application of the multi-verifier nonce derivation protocol in multi-party EdDSA signatures.

4.1 Extended Doubly-Authenticated Bits for Multi-Verifier Zero-Knowledge Proof

The extended doubly-authenticated bits in the multi-verifier setting mv-edaBits is a key tool in this work to efficiently convert $[\![\boldsymbol{x}]\!]_2$ to an authenticated sharing $[\![X]\!]_q$ over group. A mv-edaBits is defined as a tuple $([\![\rho]\!]_q, \{[\![\boldsymbol{\rho}[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}[\ell]]\!]_2\})$ where the identical random value $\rho \in \mathbb{F}_{2^\kappa}$ 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 provide the macro definition for mv-edaBits in Figure 9.

Macro MV-edaBits

This macro 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.

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

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. Similarly, $\mathcal{P}$ 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 $\mathcal{V}_i$.

Create mv-edaBits runs $\mathsf{PCF.Eval}_{\mathsf{mv}}$ to derive authenticated mv-edaBits for each $\mathcal{V}_i, i \in [N]$ and $\mathcal{P}$. Specifically, Create executes with common strings v_1 and v_2 as follows:

1. $\mathcal{P}$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(0, k_0^{(q)}, v_1)$, while for each $i \in [N]$, $\mathcal{V}_i$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(i, k_i^{(q)}, v_1)$ to generate $[\![\rho]\!]_q$.
2. In parallel, $\mathcal{P}$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(0, k_0^{(2)}, v_2)$ and for each $i \in [N]$, $\mathcal{V}_i$ runs $\mathsf{PCF.Eval}_{\mathsf{mv}}(i, k_i^{(2)}, v_2)$ to generate $[\![\boldsymbol{\rho}]\!]_2$.

Figure 9: The Generation Macro of mv-edaBits

The prover $\mathcal{P}$ and N verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$ can generate faulty mv-edaBits by simply invoking PCF macro. Observed we need to check the consistency of bits $\{\boldsymbol{\rho}[1], \ldots, \boldsymbol{\rho}[\ell]\}$ and random value ρ shared in two fields. It can be further achieved by cut-and-choose verification phase of [EGK+20]. In this paper, we observe that these values can be succinctly checked with the aid of verifiability of the signature. See the proof given in Appendix. B), if red ρ_is (the Boolean parts) are not consistent with the blue ρ (the arithmetic part), the check subroutine will fail except with probability at most $\frac{1}{q}$. Thus, we omit the detailed protocol here.

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 Figure 8). $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ shown in Figure 10 defines our multi-verifier nonce derivation functionality.

Before going into the achievement of $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, we need to present a core check subroutine in used. It works in the multi-verifier setting and adopts polynomial-based batch verification technique. We show the

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

This functionality runs with $P_1, \ldots, P_n$. This functionality is parameterized by the nonce derivation circuit $\mathcal{C}^* := 1\{R = \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg}) \cdot G\}$ where G is a generator of the elliptic curve group $\mathbb{G}$ with order q.

1. For each $i \in [n]$:
   - Upon receiving $(\mathsf{mvzk\text{-}input}, i, \mathsf{dk}_i)$ from P_i (acting as the prover $\mathcal{P}$) and $(\mathsf{mvzk\text{-}input}, i)$ from all the other parties (acting as the verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_N$), with a fresh identifier sid, store (i, dk_i).
   - Upon receiving $(\mathsf{mvzk\text{-}prove}, i, (\mathsf{msg}, R_i))$ from P_i (acting as $\mathcal{P}$) and $(\mathsf{mvzk\text{-}verify}, i, (\mathsf{msg}, R_i))$ from all the other parties (acting as $\mathcal{V}_1, \ldots, \mathcal{V}_N$). If (i, dk_i) has been stored, set $res_i = \mathsf{true}$ if $\mathcal{C}^*(\mathsf{dk}, \mathsf{msg}, R) = 1$ and $res_i = \mathsf{false}$ otherwise.
2. If all the $res_1, \ldots, res_n$ is true, set $res = R_1 + \cdots + R_n$; $res = \perp$ otherwise.
3. Send res to the adversary. Wait for input from the adversary, and perform as follows. If it is $\mathsf{continue}$, send $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, res)$ to all the parties. If it is abort, send abort.

Figure 10: The Multi-Verifier Zero-Knowledge Proof Functionality for Nonce Derivation

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

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 Figure 8) and $\mathcal{F}_{\text{MV-ND}}$ (shown in Figure 10). Let the honest verifiers be $\mathbb{H} \subset [n]$ be the honest verifiers. Furthermore, this functionality is invoked by the following commands.

Check Multiplications: Upon receiving $(\mathsf{CheckMuls}, sid, \{[\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j}]\!]_2\}_{j\in[t]})$ 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 $\mathcal{V}_i$.

Figure 11: The Multi-Verifier Check Multiplications Functionality

ideal functionality $\mathcal{F}_{\text{MV-CheckMULs}}$ and its instantiation protocol CheckMuls in Figure 11 and Figure 12, respectively, which are generalizations in the multi-verifier setting from AssertMultVec protocol of [BMRS21]. Therefore, the security of $\mathcal{F}_{\text{MV-CheckMULs}}$ is similar to one-verifier protocol, except that we allow an adversary to control which honest verifier aborts while other verifiers output results. This could be fixed by all verifiers broadcasting their results and checking consistency. As long as one verifier is honest, this inconsistency will be found. Since the simulator can simulate the input round of any subprotocol following the description, it is sufficient even if the adversary is inconsistent. Additionally, consistency is guaranteed in the protocol $\prod_{\text{MV-ND}}$, where the parties broadcast c and check the aggregated values. Thus, the malicious behavior where corrupted prover sends different values to different honest verifiers would be detected, which may be of independent interest. Figure 13 gives an example of CheckMuls when $t = 2^4$ multiplication triples given.

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

Protocol CheckMuls

The CheckMuls is a subroutine for $\prod_{\text{MV-ND}}$. 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 $str := \mathsf{MVND}||\mathsf{sid}||k_i^*||\{d_j\}_{j\in[t]}$ [a], run $\mathsf{H}_1(str)$ 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 $\mathsf{H}_1(str)$ 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 $\mathsf{H}_1(str)$ 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 $\prod_{\text{MV-ND}}$.

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

Figure 12: Multi-Verifier Check Multiplication Subprotocol

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 $\frac{t+4\log t+1}{2^\kappa-2}$. □

We then have the following theorem. We provide a sketch here and the full security proof can be found in Appendix A

Theorem 1. CheckMuls *UC-realizes* $\mathcal{F}_{\text{MV-CheckMULs}}$ *in the presence of an adversary statically corrupting*

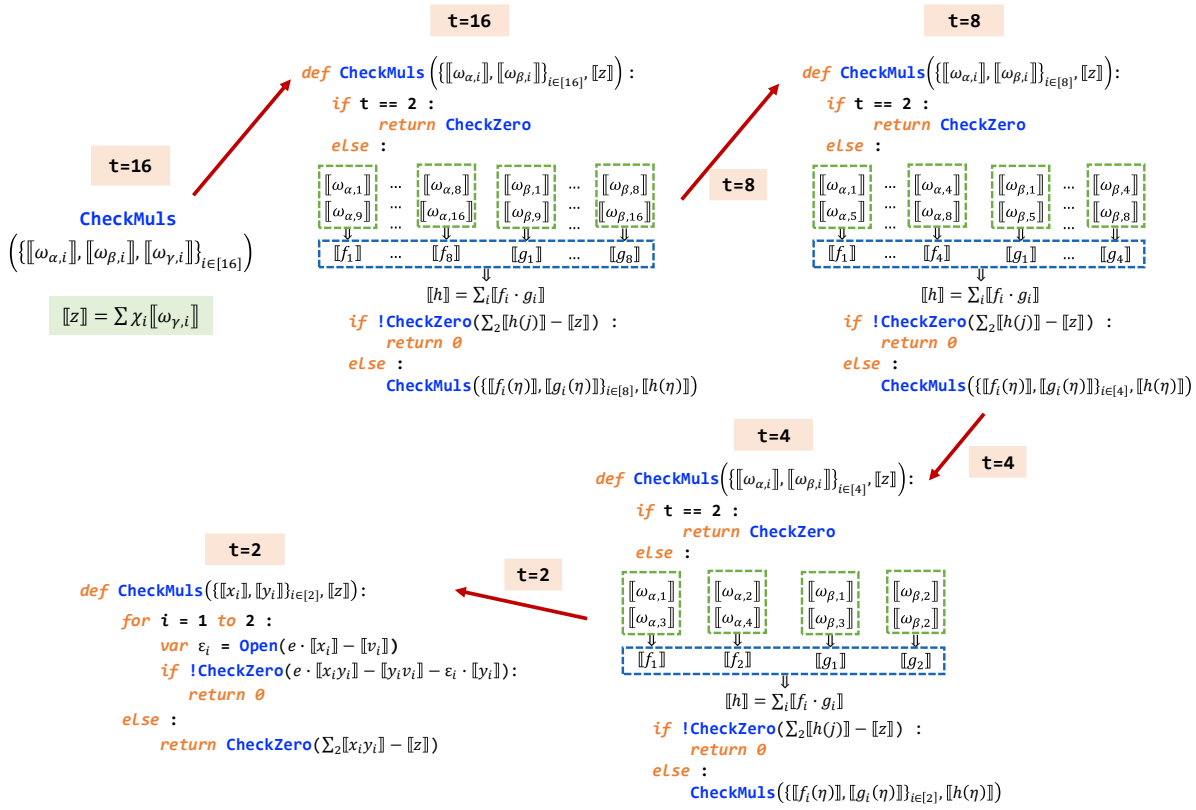

Figure 13: An Example of CheckMuls When $t = 2^4$ Multiplication Triples Given

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

Parameterized by the security parameter κ, an elliptic curve group $\mathbb{G}$ generated by G with order q. n parties hold a circuit for keyed PRF function $\mathcal{C} := \mathsf{PRF}_{\mathsf{dk}}(\mathsf{msg})$. $\mathcal{C}$ consists of t multiplication gates and inputs of derived key $\mathsf{dk} \in \mathbb{F}_{2^\kappa}$ and message $\mathsf{msg} \in \{0,1\}^\ell$. Furthermore, all parties consensus on an adder 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. Let $\mathsf{H}_0 : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathsf{H}_1 : \{0,1\}^* \to (\mathbb{F}_{2^\kappa})^*$

Setup: Run once, with each P_i (acting as $\mathcal{P}$) using private derived key $\mathsf{dk}_i \in \mathbb{F}_2^\ell$:

1. All parties run the setup of MV-edaBits, such that for each verifier $\mathcal{V}_j$ obtains the global keys $\Delta^j \in \mathbb{F}_{2^\kappa}$ and $\Lambda^j \in \mathbb{Z}_q$.
2. P_i invokes $\mathsf{Shr}(\mathsf{dk}_i)$ with constant input session id $\mathsf{sid}_{\mathsf{dk}}$ to generate the secretly shared $[\![\mathsf{dk}_i]\!]_2$
3. P_i samples $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^\kappa$ as a random PRF key, and then sends it to P_j over a private channel for each $j < i$.
4. P_i samples $k_i^* \leftarrow \mathbb{F}_{2^\kappa}$.

Figure 14: Multi-Verifier Stateless Deterministic Nonce Derivation based on MVZK

Protocol $\prod_{\mathsf{MV\text{-}ND}}$ (continued)

Proof Phase: Now the message msg and R is known by the parties:

1. Each P_i computes $\mathsf{sid} = \mathsf{H}_0(\mathsf{msg}, R)$.
2. Each P_i initializes a string for random generator as $str := \mathsf{H}_1(\mathsf{MVND}||\mathsf{sid}||k_i^*)$.
3. All parties invoke (Create mv-edaBits) of MV-edaBits, which returns secret tuples $\{[\![\rho_i]\!]_q, [\![\boldsymbol{\rho}_i[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}_i[\ell]]\!]_2\}$ for each $i \in [n]$.

   // Gate-by-Gate Evaluation

4. For each $i \in [n]$, P_i's derived key dk_i and random bits $\boldsymbol{\rho}_i[1], \ldots, \boldsymbol{\rho}_i[\ell]$ have been secretly shared between P_i (acting as $\mathcal{P}$) and all the other parties (acting as $\mathcal{V}_1, \ldots, \mathcal{V}_N$), they evaluate the circuit $\tilde{\mathcal{C}}$ 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 $[\![\omega_\gamma]\!]_2 \leftarrow \mathsf{Assign}(\omega_\alpha \cdot \omega_\beta)$. (Denote the transcript sent by $\mathcal{P}$ here as d_j)

   (b) $\mathcal{P}$ and $\mathcal{V}$s jointly check the correctness of multiplication triples by invoking
   $$\mathsf{CheckMuls}(\{[\![\omega_{\alpha,j}]\!]_2, [\![\omega_{\beta,j}]\!]_2, [\![\omega_{\gamma,j}]\!]_2\}_{j\in[\tilde{t}]}).$$
   If any failure happens, the parties output false.

   // Aggregated Check

5. For each $i \in [n]$, all parties now obtain the output wires $\{[\![\boldsymbol{c}_i[1]]\!]_2, \ldots, [\![\boldsymbol{c}_i[\ell]]\!]_2\}$ that satisfy $\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{c}_i[j] = (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}_i[j]) + (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}_i[j])$ and $\{\boldsymbol{r}_i[1], \ldots, \boldsymbol{r}_i[\ell]\} = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg})$. Each party P_i opens them to $\boldsymbol{c}_i \in \mathbb{F}_2^\ell$. If any of them is invalid, the parties output false and abort.
6. For each $i \in [n]$, all parties compute $c_i = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{c}_i[j] \mod q$, and compute $[\![R_i]\!]_q = (c_i - [\![\rho_i]\!]_q) \cdot G$. Note that $c_i - [\![\rho_i]\!]_q$ is actually $[\![r_i]\!]_q$ defined with global key $\Lambda^j, j \in [N]$ over $\mathbb{Z}_q$.
7. Each party P_i continuously append $str = str||\{c_1||\ldots||c_n\}$, generate $n+1$ common randoms by $h, \chi_1, \ldots, \chi_n := \mathsf{H}_1(str)$ and append $str := str||h||\chi_1||\ldots||\chi_n$. Then P_i sends $S_i = ((n-1)^{-1} \cdot \sum_{j\neq i} \chi_j \cdot c_j + \sum_{j<i}(\mathsf{PRF}_{\mathsf{seed}_{i,j}}(h)) - \sum_{j>i}(\mathsf{PRF}_{\mathsf{seed}_{j,i}}(h))) \cdot G$ to all the others.
8. P_i computes $S = \sum_{i\in[n]} S_i$, $\hat{R} = \sum_{i\in[n]} \chi_i \cdot R_i$ and $Z_i = \Lambda^i \cdot (S - \hat{R}) + (\sum_{j\neq i}(\chi_j \cdot k_\rho^{j,i} - \chi_i \cdot m_\rho^{i,j}) - \chi_i \cdot \rho_i \cdot \Lambda^i) \cdot G$, where $k_\rho^{j,i} \in \mathbb{Z}_q$ is P_i's local MAC key corresponding to ρ_j of P_j's edaBits and $m_\rho^{i,j} \in \mathbb{Z}_q$ is P_i's MAC tag of ρ_i established with P_j, $j \neq i$. Then P_i sends Z_i to all the others using $\mathcal{F}_{\mathsf{Com}}$.
9. P_i checks if $\sum_{i\in[n]} Z_i = \mathcal{O}$ holds. If this check fails, P_i aborts. Otherwise, each party outputs the correct R.

Figure 15: Multi-Verifier Stateless Deterministic Nonce Derivation based on MVZK (continued)

up to $N-1$ verifiers, in the PCF *assumption.*

Proof. (Sketch) Assume that if $\mathcal{P}$ is corrupted, $\mathcal{S}$ samples $\Delta^i \leftarrow \mathbb{F}_{2^\kappa}$ and receives $\{\omega_{\alpha,i}, \omega_{\beta,i}, \omega_{\gamma,i}\}_{i\in[\tau]}$, randoms and their MAC tags by simulating PCF. $\mathcal{S}$ defines $\boldsymbol{k}_\alpha, \boldsymbol{k}_\beta, \boldsymbol{k}_\gamma$ and local MAC keys of randoms using Δ^i. $\mathcal{S}$ executes the protocol honestly and checks the transcripts using its local MAC keys and global keys. Finally, if any check fails, $\mathcal{S}$ aborts and outputs whatever $\mathcal{A}$ outputs. The simulated view of $\mathcal{A}$ has

the identical distribution as its view in the real execution. Whenever honest verifier $\mathcal{V}_i, i \in \mathbb{H}$ in the real execution aborts, $\mathcal{S}$ acting as $\mathcal{V}_i$ in the ideal execution aborts. Thus, it remains to bound the probability that $\mathcal{V}_i$ in the real execution accepts but the transcripts received by $\mathcal{S}$ fail to pass the CheckZero subcheck. In this case, the malicious $\mathcal{P}$ will successfully trick honest $\mathcal{V}_i$ into accepting a forged MAC tag. According to Lemma 1, the probability that the honest $\mathcal{V}_i$ in the real execution does not abort is at most $\frac{t+4\log t+1}{2^\kappa-2}$.

If $\mathcal{P}$ is honest and $\mathbb{H}$ is the set of honest verifiers, $\mathcal{S}$ firstly receives res from $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, receives $\Delta^i \in \mathbb{F}_{2^\kappa}$ and all the MAC keys for secret values (i.e., $\boldsymbol{k}_\alpha, \boldsymbol{k}_\beta, \boldsymbol{k}_\gamma$) and randoms from the simulating of PCF. $\mathcal{S}$ executes the protocol as an honest prover, except that sending random values to $\mathcal{V}$s. $\mathcal{S}$ computes the check transcripts using its MAC keys and global keys. Finally, if $res = \mathsf{true}$ received from $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, $\mathcal{S}$ sends 0-sharing on final CheckZero to $\mathcal{A}$, otherwise, $\mathcal{S}$ sends random values and aborts. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. This simulation is identical to a real execution since the local MAC keys are uniformly random and perfectly hidden against the view of adversary A. Regarding honest $\mathcal{P}$, there is no output in the real protocol. Thus, the view of $\mathcal{A}$ simulated by $\mathcal{S}$ is distributed identically to its view in the real execution. This completes the proof sketch. □

Given the correctness check subroutine, we can design the instance protocol of multi-verifier zero-knowledge proof for nonce derivation in the ($\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$,$\mathcal{F}_{\mathsf{Com}}$)-hybrid model and PCF assumption, as shown in Figure 14. The $\prod_{\mathsf{MV\text{-}ND}}$ phase will be *deterministic, stateless* to serve the multi-party EdDSA signing setting, where the common random generators are realized by $\mathsf{H}_1(\mathsf{long\text{-}term\ key}||\mathsf{transcripts})$ with a cryptographic hash function $\mathsf{H}_1 : \{0,1\}^* \to (\mathbb{F}_{2^\kappa})^*$. In particular, (1) the gate-by-gate paradigm consumes $\tilde{t}$ bits, where $\tilde{t}$ is the number of combined circuit $\tilde{\mathcal{C}}$; (2) the polynomial-based batch verification technique consumes $\log(\tilde{t}) \cdot 4\kappa + 9\kappa$ bits and the communication complexity is roughly $O(\tilde{t} + \log(\tilde{t}) \cdot \kappa)$ in a *non-interactive* setting; (3) the final aggregated check phase consumes $\kappa \cdot \ell + 2\ell_{\mathbb{G}} + \ell_{\mathsf{com}}$ per party, with three rounds. Therefore, the communication complexity of a $\prod_{\mathsf{MV\text{-}ND}}$ protocol is $O(\tilde{t} + \log(\tilde{t}) \cdot \kappa + \kappa \cdot \ell)$ bits for each party and four rounds in total.

Note that $\mathcal{P}$ does *not* broadcast d_i for $i \in [\tilde{t}]$ here. The core challenge is that $\mathcal{A}$ may bias an *inconsistent* authenticated bits, i.e., the adversary could choose different $x_1, \ldots, x_N$ such that for each $i \in [N]$, $m_i = k_i + x_i \cdot \Delta^i$, therefore, we must prove that the consistency check techniques are UC-secure in the presence of an adversary statically corrupting up to $n-1$ parties. The security of $\prod_{\mathsf{MV\text{-}ND}}$ is proved in the following theorem. The details of security proof can be found in Appendix B.

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 ($\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, $\mathcal{F}_{\mathsf{Com}}$)-hybrid random oracle model and PCF assumption.*

4.3 Multi-Party EdDSA Signature Protocol

Based on prior definitions [LN18, BST21], we present an EdDSA ideal functionality as shown in Figure 16, where the (`KeyGen`) command is allowed to be called *only once* and the (`Sign`) command could be called multiple times.

In Figure 17, we describe the multi-party EdDSA signing protocol details, which enable us to obtain $O(n)$ communication bandwidth. This protocol works in ($\mathcal{F}_{\mathsf{MV\text{-}ND}}$, $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$)-hybrid model and is executed by n parties. In the key generation phase, each party $P_i, i \in [n]$ generates sk_i and jointly computes $\mathsf{pk} = \sum_{i\in[n]} s_i \cdot G$. P_i also invokes $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ to commit to the PRF key $\mathsf{dk}_i := \mathsf{PRF}(\mathsf{sk}_i)[\ell+1; 2\ell]$, acting as the prover and other $n-1$ parties act as all the verifiers. Remark that in this phase, each party P_i samples all the uniformly random long-term keys, such as $k_i^*, \Delta^i, \Lambda^i, \mathsf{seed}_{i,j}$ and PCF keys for $\prod_{\mathsf{MV\text{-}ND}}$.

In the signing phase, each party proves its derived nonce $R_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$ by calling $\mathcal{F}_{\mathsf{MV\text{-}ND}}$. If all the parties see $res \neq \perp$ from $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, they obtain R. Still, we do *not* require to broadcast transcripts in this protocol. The communication overheads is $|\mathcal{F}_{\mathsf{MV\text{-}ND}}| + \ell_{\mathbb{G}} + q$ for each party. As discussed in

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

This functionality is parameterized by EdDSA $\mathtt{params} = (\mathbb{E}_p, \mathbb{G}, q, G, \ell_b, \ell, \mathsf{H}_{\mathsf{sig}}, \mathsf{PRF})$ 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 store $(\mathsf{msg}, (\sigma, R))$. Then, wait for an input from the adversary, either abort or $\mathsf{continue}$. If $\mathsf{continue}$, send (σ, R) to all parties.

Figure 16: The EdDSA Functionality

Section 2.3, we instantiate $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ by the Fiat-Shamir heuristic; the communication rounds of the signing phase are six rounds.

The security of this protocol is proved in the following theorem, and the detailed proof is presented in Appendix C

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.*

5 Performance and Evaluation

The evaluation is configured in the Ed25519 curve. In particular, it provides $\kappa = 128$ security and $\mathsf{SHA512}$ as PRF nonce derivation function specified by EdDSA standard. According to the estimated by [AAL+24], SHA-512 has $58k$ AND gates. The multi-party signing protocol is essentially a thin wrapper on the top of $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, and consequently, the cost is dominated by running $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ among parties. Note that by the structure of our signing protocol $\prod_{\mathsf{MV\text{-}ND}}$, instantiating $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ in both directions induces little computational overhead on top of a single instantiation: while $\mathcal{P}$ evaluates the circuit $\mathcal{V}$s sit idle, and while $\mathcal{V}$s check the circuit $\mathcal{P}$ has nothing to do. This means that when each party P_i acting as $\mathcal{P}$ in its nonce verification session, it will be idling in its $\mathcal{V}_j$ for $j \in [N]$ role in the other parties' nonce verification sessions. Therefore, the workload for each party is roughly the same.

- Gate-by-Gate Evaluation. As discussed in Section 4.2, a single transfer and secret addition cost one VOLE instance for each AND gate. Therefore, each party requires roughly $58k\times$ VOLEs for the gate-by-gate evaluation phase.
- $\mathsf{CheckMuls}$. For EdDSA, plugging in the Fiet-Shamir heuristic, the parties require hashing 50.9KB messages, followed by $\log(58k)$ computation iterations within a single transfer. An iteration consists of hashing 112B messages, one polynomial inter-product over $\mathbb{F}_{2^\kappa}[X]$ with the degree of 2, 3κ VOLEs, and $2t$ polynomial evaluation with the degree of 1. The final iteration consumes four VOLE correlations and hashing 128B messages. Therefore, each party requires $3\kappa \times \log(58k) + 4 \approx 6.1k$ VOLEs and some little overhead.
- Aggregated Check. Each party additionally performs up to four elliptic curve multiplications, $2n$ elliptic curve additions and $(n-1)$ PRF, at little overheads.

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

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

Distributed Key Generation: Upon receiving $(\mathsf{KeyGen}, \texttt{params})$, 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 $\mathsf{dk}_i := \{\boldsymbol{h}_i[\ell_b + 1], \ldots, \boldsymbol{h}_i[2\ell_b])\}$ and sends $(\mathsf{mvzk\text{-}input}, i, \mathsf{dk}_i)$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$.
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 broadcast 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. P_i derives nonce vector by $\boldsymbol{r}_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg})$ and computes $r_i = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}_i[j] \mod q, R_i = r_i \cdot G$.
2. P_i sends R_i to all the others, receives R_j for $j \neq i$ and computes $R = \sum_{i \in [n]} R_i$.
3. P_i proves the R_i by sending $(\mathsf{mvzk\text{-}prove}, i, (\mathsf{msg}, R_i))$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ acting as prover $\mathcal{P}$ and all other P_j, $j \neq i$, send $(\mathsf{mvzk\text{-}verify}, i, (\mathsf{msg}, R_i))$ to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$ acting as verifiers $\mathcal{V}$s.
4. Wait to receive $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, res)$ from $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, P_i aborts if any $res = \perp$. Otherwise, each P_i obtains correct R.
5. 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 $\mathcal{F}_{\mathsf{Com}}$.
6. 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).

Figure 17: The Stateless Deterministic Multi-Party EdDSA Signing Protocol

In summary, the workload for each party is $\approx 64.1k$ VOLEs, additionally with little overhead. Boyle et al. [BCG+22] estimate PCF evaluation times. Specifically, the VOLE can be instantiated using fixed-key AES, measured by AES-NI instructions of modern CPUs and a 3GHz processor, one PCF evaluation consumes 3.57×10^{-3} milliseconds. Concretely, we estimate around 230 ms for one signature among two parties with one corruption. The number can be scaled up linearly with GPU as the AES calls in PCF are perfectly independent and, therefore, parallelizable. Unfortunately, PCF [BCG+22], no open source code released to date, puts a formidable barrier on our evaluation. Therefore, the timings here are estimated theoretically.

Compare with two existing works on the multi-party EdDSA signing protocol, i.e., Bonte et al. [BST21] and Garillot et al. [GKMN21]. Bonte et al. [BST21] implemented their experiments using SCALE-MAMBA and tested in a LAN setting, with each party running on an Intel i7-7700K CPU (4 cores at 4.2GHz with 2 threads per core) with 32GB of RAM over a 10Gb/s network switch. The running time is 1406 ms under

the Shamir (3,1) access structure, averaged over 100 experiments. The overall burden for each party of Garillot et al. [GKMN21] is roughly the same with 132k AES invocations of 128 bit-ciphertext, hashing a 245KB message, three curve multiplications, and 256 additions in $\mathbb{Z}_q$. Garillot et al. also does not empirically measure their $\pi_{n,\mathsf{Sign}}$ protocol. To give a P2P comparison, we estimate Garillot et al.' protocol by the measurement of Boyle et al. [BCG+22], where one byte of fixed-key AES can be computed in 1.3 CPU cycles. Thus, using a 3GHz processor, it consumes 102 ms for a signature. We notice that our computation time is not huge, but achieves one or two orders of magnitude of communication overheads over what is consumed in prior related works.

6 Conclusion

This paper presents a communication-friendly verifiable nonce derivation in the multi-verifier setting and builds multi-party EdDSA signing on top of it. Our protocol has made huge progress in proving the standard PRF execution per party. However, there are still limitations to our multi-party EdDSA signing protocol that deserve further exploration in future works. Firstly, although our MVZK protocol can prove to multiple verifiers at a time, the participants are chosen in advance, which has little impact on the signing scenario yet puts constraints on dynamic applications compared to non-interactive ZK proofs like zk-SNARKs. Secondly, we observed a high computation overhead for PCF-based VOLE, which may potentially be further optimized, for example using half-tree [GYW+23].

Acknowledgments

Qi Feng and Debiao He are 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 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 and 61932019). Xiao Wang is supported by NSF award #2236819. Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 61872236). Yu Yu's work has also been supported by the New Cornerstone 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.

[ANO+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

[ANT+20] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 225–242, Virtual Event, USA, November 9–13, 2020. ACM Press.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[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.

[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.

[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.

[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.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[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.

[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.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 499–530, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[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.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[DH20] Gabrielle De Micheli and Nadia Heninger. Recovering cryptographic keys from partial information, by example. Cryptology ePrint Archive, Report 2020/1506, 2020. https://eprint.iacr.org/2020/1506.

[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.

[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.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 330–362, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[HGS01] Nick A Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23:283–290, 2001.

[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.

[KASN15] Rashmi Kumari, Mohsen Alimomeni, and Reihaneh Safavi-Naini. Performance analysis of Linux RNG in virtualized environments. In *Proceedings of the 2015 ACM Workshop on Cloud Computing Security Workshop – CCSW '15*, pages 29–39, New York, NY, USA, October 2015. Association for Computing Machinery.

[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.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164, Washington, DC, USA, October 27–30, 2003. 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.

[MNPV99] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. Computational alternatives to random number generators. In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 72–80, Kingston, Ontario, Canada, August 17–18, 1999. Springer, Heidelberg, Germany.

[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. *Designs, Codes and Cryptography*, 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.

[nis19] FIPS PUB 186-5 (Draft): Digital Signature Standard (DSS). https://doi.org/10.6028/NIST.FIPS.186-5, 2019.

[NKDM03] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*, San Diego, CA, USA, February 5–7, 2003. The Internet Society.

[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.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[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.

[YCX21] Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 481–511, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany.

A Security Proof of Theorem 1

Proof. The security of CheckMuls shown in Figure A 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, and 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 $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, which runs an adversary $\mathcal{A}$ as a subroutine when emulating PCF and RO. In both case, we show that no environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

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 $\{\omega_{\alpha,i}, \omega_{\beta,i}, \omega_{\gamma,i}\}_{i\in[\tau]}$ 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 $\{[\![\omega_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2, [\![\omega_{\gamma,i}]\!]_2\}_{i\in[\tau]}$ to functionality $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$ on behalf of corrupted prover $\mathcal{P}$ and corrupted verifiers $\mathcal{V}_2, \ldots, \mathcal{V}_N$.

It is clear that 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 $\mathcal{A}$ is 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, $\mathcal{S}$ acting 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 $[\![z]\!]_2, \{[\![\hat{\omega}_{\alpha,i}]\!]_2, [\![\omega_{\beta,i}]\!]_2\}_{i\in[t]}$.

4. While $t > 2$:

   (a) $\mathcal{S}$ emulates PCF by recording the local MAC keys for the random value used in the Assign subroutine, which is 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 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, $\mathcal{S}$ 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 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.

- $\mathbf{Hybrid_0}$. This is the real world.

- $\mathbf{Hybrid_1}$. 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 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 $j \in \{0, 1, 2\}$, simulator $\mathcal{S}$ computes $k^i_{\mu,j} := \mathsf{PCF.Eval}(\sigma, k_\sigma, \mathsf{sid} + \mathsf{j})$. Then it queries 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 for the transcript of 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.

- $\mathbf{Hybrid_2}$. 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.

- $\mathbf{Hybrid_3}$. 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.

- $\mathbf{Hybrid_4}$. 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 $\mathsf{CheckMuls}$ to run $\mathsf{CheckZero}$ 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 2

Proof. We consider the case that $n-1$ parties are corrupted. First, we analyze its correctness as follows.

Correctness. The correctness of MV-ND protocol $\prod_{\mathsf{MV\text{-}ND}}$ as described in Figure 14 and Figure 15 relies on the gate-by-gate evaluation and aggregated check. In the honest case, for each $i \in [n]$, the parties obtain $\boldsymbol{c}_i := \{\boldsymbol{c}_i[1], \ldots, \boldsymbol{c}_i[\ell]\}$ such that $\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{c}_i[j] = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}_i[j] + \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}_i[j]$ and therefore

$$\begin{aligned} R_i &= (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{r}_i[j] \mod q) \cdot G \\ &= (\sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{c}_i[j] \mod q - \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}_i[j] \mod q) \cdot G \\ &= (c_i - \rho_i) \cdot G \end{aligned}$$

Furthermore, for any $h, \chi_1, \ldots, \chi_n$ which are generated by a hash chain as

$$\mathsf{H}_1(\mathsf{H}_1(\ldots \mathsf{H}_1(\mathsf{MVND}||\mathsf{sid}||k_i^*)||\mathsf{transcripts})||\{c_i\}_{i\in[n]}),$$

we have the following:

$$\sum_{i\in[n]}(\sum_{j<i}(\mathsf{PRF}_{\mathsf{seed}_{i,j}}(h)) - \sum_{j>i}(\mathsf{PRF}_{\mathsf{seed}_{j,i}}(h))) = 0.$$

Then, we observe that

$$\begin{aligned}
S = \sum_{i\in[n]} S_i &= \sum_{i\in[n]} (\frac{\sum_{j\neq i} \chi_j \cdot c_j}{n-1}) \cdot G \\
&\quad + \sum_{i\in[n]} (\sum_{j<i} \mathsf{PRF}_{\mathsf{seed}_{i,j}}(h) - \sum_{j>i} \mathsf{PRF}_{\mathsf{seed}_{j,i}}(h)) \cdot G \\
&= \sum_{i\in[n]} \chi_i \cdot c_i \cdot G \\
&= \sum_{i\in[n]} \chi_i \cdot (r_i + \rho_i) \cdot G \\
&= \sum_{i\in[n]} \chi_i \cdot R_i + \sum_{i\in[n]} \chi_i \cdot \rho_i \cdot G
\end{aligned}$$

According to the definition of mv-edaBits, each party P_i holds $\{[\![\rho_i]\!]_q, [\![\boldsymbol{\rho}_i[1]]\!]_2, \ldots, [\![\boldsymbol{\rho}_i[\ell]]\!]_2\}$ where the integer part $[\![\rho_i]\!]_q := \{\rho_i, \{k_\rho^{i,j}, m_\rho^{i,j}\}_{j\in[N]}\} \in \mathbb{Z}_q \times (\mathbb{Z}_q \times \mathbb{Z}_q)^N$ satisfying $\rho_i = \sum_{j=1}^{\ell} 2^{j-1} \cdot \boldsymbol{\rho}_i[j] \mod q$ and $m_\rho^{i,j} = k_\rho^{i,j} + \rho_i \cdot \Lambda^j \in \mathbb{Z}_q$. Thus, if all parties execute correctly, we have

$$\begin{aligned}
\sum_{i\in[n]} Z_i &= \sum_{i\in[n]} \Lambda^i \cdot (S - \sum_{i\in[n]} \chi_i \cdot R_i) \\
&\quad + \sum_{i\in[n]} (\sum_{j\neq i} (\chi_j \cdot k_\rho^{j,i} - \chi_i \cdot m_\rho^{i,j}) - \chi_i \cdot \rho_i \cdot \Lambda^i) \cdot G \\
&= \sum_{i\in[n]} \Lambda^i \cdot (S - \sum_{i\in[n]} \chi_i \cdot R_i) \\
&\quad + \sum_{i\in[n]} (\sum_{j\neq i} (\chi_i \cdot k_\rho^{i,j} - \chi_i \cdot m_\rho^{i,j}) - \chi_i \cdot \rho_i \cdot \Lambda^i) \cdot G \\
&= \sum_{i\in[n]} \Lambda^i \cdot \sum_{i\in[n]} \chi_i \cdot \rho_i \cdot G \\
&\quad + \sum_{i\in[n]} (\sum_{j\neq i} (-\chi_i \cdot \rho_i \cdot \Lambda^j) - \chi_i \cdot \rho_i \cdot \Lambda^i) \cdot G \\
&= \sum_{i,j\in[n]} \Lambda^j \cdot \chi_i \cdot \rho_i \cdot G + \sum_{i,j\in[n]} \chi_i \cdot (-\rho_i \cdot \Lambda^j) \cdot G \\
&= \mathcal{O}
\end{aligned}$$

In the following, we prove the security of our $\prod_{\mathsf{MV\text{-}ND}}$ protocol in the multi-party malicious setting. Without loss of generality, we let P_1 denote the honest party and $P_i, i \in \mathcal{I} = [2, n]$ denote the corrupted parties. We always implicitly assume that $\mathcal{S}$ passes all communication between adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Besides, $\mathcal{S}$ needs to simulate honest prover when P_1 acts as $\mathcal{P}$ and honest verifier when P_1 acts as $\mathcal{V}_1$. First, the simulator $\mathcal{S}$ 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 ($\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$, $\mathcal{F}_{\mathsf{Com}}$)-hybrid model and PCF assumption $\mathcal{S}$ receives the secret inputs from $\mathcal{A}$.

For the input phase:

1. If P_1 acts as $\mathcal{P}$ (the case of honest prover), $\mathcal{S}$ executes as follows:

   (a) $\mathcal{S}$ receives the global key $\Delta^1, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ and $\Lambda^1, \ldots, \Lambda^N \in \mathbb{Z}_q$ from MV-edaBits.

(b) $\mathcal{S}$ receives the MAC keys for all secret values (i.e., $\boldsymbol{k}_{\mathsf{dk}_i} \in \mathbb{F}_{2^\kappa}^\ell$) from $\mathcal{A}$.

(c) In the zero-sharing setup, on behalf of P_1, $\mathcal{S}$ receives $\mathsf{seed}_{i,1}$ from $\mathcal{A}$ for each $i \in [2,n]$ and sets $k_1^* \leftarrow \mathbb{F}_{2^\kappa}$.

2. If P_1 acts as $\mathcal{V}_1$ (the case of corrupt prover), $\mathcal{S}$ executes as follows:

(a) $\mathcal{S}$ samples uniformly random $\Delta^1 \leftarrow \mathbb{F}_{2^\kappa}, \Lambda^1 \leftarrow \mathbb{Z}_q$.

(b) $\mathcal{S}$ receives the global key $\Delta^2, \ldots, \Delta^N \in \mathbb{F}_{2^\kappa}$ and $\Lambda^2, \ldots, \Lambda^N \in \mathbb{Z}_q$ from MV-edaBits.

(c) $\mathcal{S}$ records all the values (i.e., $\mathsf{dk}_i[1], \ldots, \mathsf{dk}_i[\ell] \in \mathbb{F}_2$) and their corresponding MAC tags (i.e., $\boldsymbol{m}_{\mathsf{dk}_i} \in \mathbb{F}_{2^\kappa}^\ell$) by simulating PCF for $\mathcal{A}$. Here, $\mathcal{S}$ can define the corresponding MAC keys of these values (i.e., $\boldsymbol{k}_{\mathsf{dk}_i} \in \mathbb{F}_{2^\kappa}^\ell$).

(d) Similarly, on behalf of P_1, $\mathcal{S}$ receives $\mathsf{seed}_{i,1}$ from $\mathcal{A}$ for each $i \in [2,n]$ and sets $k_1^* \leftarrow \mathbb{F}_{2^\kappa}$.

For the proof phase: For given message msg,

1. $\mathcal{S}$ computes $\mathsf{sid} = \mathsf{RO}(\mathsf{msg}, R)$.

2. $\mathcal{S}$ records all the randomness used by $\mathcal{A}$ from the set of queries made by P_i to RO. $\mathcal{S}$ sets random tape as $\mathsf{RO}(k_1^*, \mathsf{sid}, \mathsf{MVND})$ acting as P_1

3. $\mathcal{S}$ extracts the secret edaBits values as follows:

- If P_1 acts as $\mathcal{P}$ (the case of honest prover), $\mathcal{S}$ receives the MAC keys for all edaBits values (i.e., $\boldsymbol{k}_{\rho_i} \in \mathbb{F}_{2^\kappa}^\ell, \hat{k}_{\rho_i} \in \mathbb{Z}_q$) by emulating MV-edaBits.
- If P_1 acts as $\mathcal{V}_1$ (the case of corrupt prover), $\mathcal{S}$ records all the edaBits values (i.e., $\rho_i[1], \ldots, \rho_i[\ell] \in \mathbb{F}_2, \rho_i \in \mathbb{Z}_q$) and corresponding MAC tags (i.e., $\boldsymbol{m}_{\rho_i} \in \mathbb{F}_{2^\kappa}^\ell, \hat{m}_{\rho_i} \in \mathbb{Z}_q$), which are extracted by emulating MV-edaBits for the adversary. Similarly, $\mathcal{S}$ define the corresponding MAC keys of these values (i.e., $\boldsymbol{k}_{\rho_i} \in \mathbb{F}_{2^\kappa}^\ell, \hat{k}_{\rho_i} \in \mathbb{Z}_q$).

4. $\mathcal{S}$ evaluate the circuit $\tilde{\mathcal{C}}$ cooperated with the other parties $P_i, i \in [2,n]$:

- If P_1 acts as $\mathcal{P}$ (the case of honest prover), $\mathcal{S}$ simulates as follows:
  (a) 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.
  (b) $\mathcal{S}$ 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.
  (c) $\mathcal{S}$ 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.
- If P_1 acts as $\mathcal{V}_1$ (the case of corrupt prover), $\mathcal{S}$ simulates as follows:
  (a) In the subroutine Assign, $\mathcal{S}$ records all the values (i.e., $\mu_i[1], \ldots, \mu_i[\tilde{t}] \in \mathbb{F}_2$) and their corresponding MAC tags (i.e., $\boldsymbol{m}_{\mu_i} \in \mathbb{F}_{2^\kappa}^{\tilde{t}}$) by emulating PCF for $\mathcal{A}$. Here, $\mathcal{S}$ can define the corresponding MAC keys of these values (i.e., $\boldsymbol{k}_{\mu_i} \in \mathbb{F}_{2^\kappa}^{\tilde{t}}$).
  (b) $\mathcal{S}$ executes Step 4.(a) as an honest verifier.
  (c) $\mathcal{S}$ 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 P_i and sends true to all the other $P_j, j \in [2,n]\backslash\{i\}$ on behalf of $\mathcal{F}_{\mathsf{MV\text{-}CheckMULs}}$.

5. $\mathcal{S}$ samples random $c_1 \leftarrow \mathbb{Z}_q$ and reveals its binary representation $[\![c_1]\!]_2$ to all the other parties.
6. Upon receiving $c_i, i \in [2,n]$ from the other parties, if any c_j is invalid, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues.
7. $\mathcal{S}$ sends $R_1 = R - \sum_{i\in[2,n]} R_i$ with $R_i = \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$ to all other parties on behalf of P_1.
8. On receiving R_i from each P_i, for each $i \in [2,n]$, if any of the $R_i \neq \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$ is incorrect, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues.
9. $\mathcal{S}$ sends $\chi_1, \ldots, \chi_t \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$ and computes S_1 using $\{c_i, \mathsf{seed}_{i,1}\}_{i\in[2,n]}$, sends it to other parties and receives $S_i, i \in [2,n]$ from $\mathcal{A}$ and $Z_i, i \in [2,n]$ on behalf of $\mathcal{F}_{\mathsf{Com}}$.
10. $\mathcal{S}$ checks whether $\sum_{i\in[2,n]} Z_i = (\sum_{i\in[2,n]} \Lambda^i)(S_1 - \chi_1 \cdot R_1) + (\sum_{i\in[2,n]} \chi_i \cdot k_\rho^{1,i}) - (\sum_{i\in[2,n]} \chi_1 \cdot m_\rho^{i,1})$ holds. If this check fails, $\mathcal{S}$ aborts.
11. Otherwise, it opens $Z_1 = \mathcal{O} - \sum_{i\in[2,n]} Z_i$ to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs.

Indistinguishability of the simulation is argued as follows: first, the distributions of the random d_j, c_1 value in the simulated protocol and $d_j = \omega_{\alpha,j} \cdot \omega_{\beta,j} + \mu_j, c_1 = r_1 + \rho_1$ in the real protocol are identical. Then, the only non-syntactic difference between the simulation and the real protocol is that when $\exists i \in [2,n]$ such that $R_i \neq \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$. As the CheckMuls subroutine guarantees that the $c_i = r_i + \rho_i$ is correctly computed from binary parts of edaBits except with the negligible probability $\mathsf{negl}(\kappa)$. Now we consider when red ρ_i (aggregated by binary representations of mv-edaBits) is not consistent with blue ρ_i (generated from integer part of mv-edaBits). Assume that $\rho_i = \rho_i + e_i$ and $R_i^* = R_i + e_{R,i} \cdot G$, if $\sum_{i\in[n]} Z_i = \mathcal{O}$ holds, we have $\hat{R}^* = \hat{R} + \sum_{i=2}^n \chi_i \cdot e_{R,i} \cdot G$ and

$$
\begin{aligned}
\mathcal{O} &= \sum_{i\in[n]} \left(\Lambda^i (S - \hat{R}^*) + \left(\sum_{j\neq i} (\chi_j \cdot k_\rho^{j,i} - \chi_i \cdot m_\rho^{i,j}) - \chi_i \rho_i \cdot \Lambda^i \right) \cdot G \right) \\
&= \sum_{i\in[n]} \Lambda^i \cdot (S - \hat{R} - \sum_{i=2}^{n} e_{R,i} \cdot G) \\
&\quad + \sum_{i\in[n]} \left(\sum_{j\neq i} \chi_i \cdot (k_\rho^{i,j} - m_\rho^{i,j}) - \rho_i \cdot \Lambda^i \right) \cdot G \\
&= \sum_{i\in[n]} \Lambda^i \cdot \Bigg((\sum_{i\in[n]} \chi_i \cdot r_i + \chi_1 \cdot \rho_1 + \sum_{i\in[2,n]} \chi_i \cdot \rho_i) \cdot G \\
&\quad - \sum_{i\in[n]} \chi_i \cdot R_i - \sum_{i\in[2,n]} \chi_i \cdot e_{R,i} \cdot G \Bigg) \\
&\quad - \left(\chi_1 \cdot \rho_1 \cdot \sum_{i\in[n]} \Lambda^i + \sum_{i\in[2,n]} \chi_i \cdot \rho_i \cdot \sum_{i\in[n]} \Lambda^i \right) \cdot G
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow q = \sum_{i\in[n]} \Lambda^i \cdot \Bigg(& \chi_1 \cdot \rho_1 + \sum_{i\in[2,n]} \chi_i \cdot \rho_i - \sum_{i\in[2,n]} \chi_i \cdot e_{R,i} \Bigg) \\
& - \chi_1 \cdot \rho_1 \cdot \sum_{i\in[n]} \Lambda^i - \sum_{i\in[2,n]} \chi_i \cdot \rho_i \cdot \sum_{i\in[n]} \Lambda^i \\
= & \left(- \sum_{i\in[2,n]} \chi_i \cdot e_i \cdot \sum_{i\in[n]} \Lambda^i \right) + \sum_{i\in[n]} \Lambda^i \cdot \left(- \sum_{i\in[2,n]} \chi_i \cdot e_{R,i} \right) \\
= & -\Lambda^1 \cdot \sum_{i\in[2,n]} \chi_i \cdot (e_{R,i} + e_i) \\
& - \sum_{i\in[2,n]} \Lambda^i \cdot \sum_{i\in[2,n]} \chi_i \cdot (e_{R,i} + e_i) \\
\Rightarrow \Lambda^1 = & \frac{q + (\sum_{i\in[2,n]} \Lambda^i) \cdot \sum_{i\in[2,n]} \chi_i \cdot (e_{R,i} + e_i)}{\sum_{i\in[2,n]} \chi_i \cdot (e_{R,i} + e_i)}
\end{aligned}
$$

Since that $\Lambda^1 \in \mathbb{Z}_q$ is uniformly random and kept secret from $\mathcal{A}$, and $\chi_1, \ldots, \chi_n$ are unpredictable random, 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 3

Proof. We begin by showing that $\prod_{\mathsf{MP,Sign}}$ 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:

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

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. The simulator $\mathcal{S}$ 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 the transcript $(\mathsf{proof\text{-}receipt}, sid_{pk,1})$ 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 $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$, 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. The simulator $\mathcal{S}$ receives $(\mathsf{nonce}, \mathsf{msg}, R)$ from $\mathcal{F}_{\mathsf{EdDSA}}$, and sends $R_1 = R - \sum_{i\in[2,n]} \mathsf{PRF}_{\mathsf{dk}_i}(\mathsf{msg}) \cdot G$ to $\mathcal{A}$. It is computable as $\mathcal{S}$ knows all the dk_i of the corrupted parties.

2. On receiving R_i from $\mathcal{A}$, if R_i has previously been seen, $\mathcal{S}$ reuses the value σ_1 stored in the memory from the last time. Otherwise, $\mathcal{S}$ proceeds to the next step.

3. Upon receiving $(\mathsf{mvzk\text{-}verify}, 1, (\mathsf{msg}, R_1^*))$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{MV\text{-}ND}}$, $\mathcal{S}$ sends $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, res)$ to $\mathcal{A}$, with $res = \perp$ if $R_1^* \neq R_1$.

4. Otherwise, $\mathcal{S}$ receives $(\mathsf{mvzk\text{-}prove}, i, (\mathsf{msg}, R_i))$ 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 $\beta = \perp$ happens, $\mathcal{S}$ sends $(\mathsf{mvzk\text{-}proof}, \mathsf{msg}, res)$ to $\mathcal{A}$ with $res = R$ on behalf of $\mathcal{F}_{\mathsf{MV\text{-}ND}}$.

6. $\mathcal{S}$ sends $(\mathsf{proceed}, \mathsf{msg})$ to $\mathcal{F}_{\mathsf{EdDSA}}$ and receives $(\mathsf{msg}, (\sigma, R))$ in response.

7. $\mathcal{S}$ simulates $(\mathsf{receipt}, sid, 1)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{Com}}$ and receives $(\mathsf{commit}, sid, i, \sigma_i^*)$ from $\mathcal{A}$.

8. $\mathcal{S}$ 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 $\mathcal{A}$.

9. 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 $(\mathcal{F}_{\mathsf{MV\text{-}ND}}, \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}, \mathcal{F}_{\mathsf{Com}})$-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 $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathcal{R}_{\mathsf{DL}}}$. 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. □