Efficient 1-Round MVZK for Dishonest Majority with Superior Online Overhead

Yuanyuan Duan[1 [0009−0009−8380−5122]], Hongxu Yi[1 [0009−0000−9323−4970]], and Yu Chen[1 [0000−0003−2553−1281]]

School of Cyber Science and Technology, Shandong University, Qingdao 266237, CN
{202417040,tcs.hongxu.yi}@mail.sdu.edu.cn, yuchen@sdu.edu.cn

Abstract. Zero-knowledge proofs allow to prove a statement while not revealing anything about the witness. In particular, non-interactive zero-knowledge (NIZK) has emerged as the preferred paradigm due to its public verifiability. However, existing NIZK protocols often suffer from high prover overhead for large statements. Conversely, designated-verifier zero-knowledge (DVZK) offers superior prover efficiency and scalability, but is restricted to a single, designated verifier. This leaves a critical gap in scenarios requiring a prover to efficiently convince a group of n verifiers without incurring massive prover overhead of non-interactive systems.
To reconcile this, we focus on multi-verifier zero-knowledge (MVZK) protocols, which efficiently prove the same statement to n verifiers simultaneously. MVZK protocols inherit the prominent prover efficiency of DVZK and extend its utility to the multi-party setting. In this work, we propose a highly efficient MVZK protocol in the preprocessing model, secure against a dishonest majority. It tolerates a corrupted prover and up to $n-1$ corrupted verifiers, where the prover sends only a single message to verifiers, while several prior works require at least two online rounds. Furthermore, compared to the state-of-the-art 1-round protocol by Zhou et al. (PKC 2025), our protocol eliminates the extension-field multiplications from $4n$ to 0 over any field, achieving superior online efficiency. Experimental results demonstrate that our protocol achieves $9\times$ faster proving and $47\times$ faster verification when evaluating Boolean circuits with 10^4 AND gates.

Keywords: Multi-Verfier Zero-Knowledge Protocol · VOLE-based ZKP · Online Efficiency.

1 Introduction

Zero-knowledge (ZK) proofs allow a prover to convince a verifier of the truth of a statement, without revealing any further information [14]. A prominent type of zero-knowledge proofs is non-interactive zero-knowledge (NIZK) proofs, where the prover sends only a single message to the verifier. NIZK proofs are crucial for asynchronous environments such as online transactions [16,8]. Despite significant recent improvements in NIZK efficiency, most optimizations focus heavily on succinctness and fast verification, inherently imposing substantial computational

overhead on the prover. Another important type of zero-knowledge proofs is designated-verifier zero-knowledge (DVZK) proofs, which have remarkable concrete efficiency, particularly in terms of prover-side costs. Compared to NIZK, DVZK proof systems [12,5,20,11,18] achieve higher efficiency and can scale to massive circuits with minimal memory footprints. However, DVZK is inherently restricted to a single designated verifier who holds a secret state, limiting its utility in collaborative or decentralized environments.

To bridge this gap, multi-verifier zero-knowledge (MVZK) proofs have been developed [21,4,2], which allow for multiple verifiers to verify a statement of a prover while maintaining its privacy. The soundness ensures that honest verifiers are not convinced even if a corrupted prover colludes with a subset of verifiers; zero-knowledge guarantees that the witness remains hidden from any corrupted coalition. Crucially, regarding performance, MVZK seamlessly extends the designated-verifier setting to a multi-verifier setting, directly inheriting the superior prover efficiency and scalability of DVZK. Therefore, MVZK enables large-scale zero-knowledge proofs for multiple verifiers.

In the MVZK setting, early protocols [1] or those derived from general techniques [6,7] either suffer from prohibitive overhead or are restricted to specific circuit structures. Although recent practical solutions [21,4,22] support generic circuits, they entail at least two communication rounds. More recently, Zhou et al. [23] achieve optimal round complexity (1-round); however, the online computational overhead remains a notable bottleneck, as the number of extension-field multiplications required for each gate scales linearly with the number of verifiers n. This linear dependence renders the protocol prohibitive for large-scale multi-party settings. Thus, it is imperative to develop a construction that maintains round-optimality while decoupling the computational complexity from the number of verifiers to achieve superior prover efficiency.

1.1 Our Contribution

Efficient 1-Round MVZK with Superior Online Efficiency. We propose a 1-round MVZK protocol that is UC-secure against dishonest majority. Our construction achieves optimality in two key dimensions:

- *Optimal Round Complexity and Corruption Threshold.* Our protocol meets the theoretical lower bound of a single online round under a dishonest majority, following the framework established in [23].
- *Superior Online Overhead.* Compared to the state-of-the-art protocol in [23], which requires $4n$ extension-field ($\mathbb{F}_{p^r}$) multiplications per gate for the prover, our protocol completely eliminates the extension-field multiplications as highlighted in Table 1. Thus our construction is exceptionally efficient for large-scale circuits over small fields. For example, the prover and each verifier in ours are $9\times$ and $47\times$ faster than [23] evaluated on Boolean circuits with 10^4 AND gates.

MV-qsVOLE: A New Form of VOLE Correlation. To achieve the aforementioned performance improvements, we introduce a new primitive called *multi-*

verifier quadratic subfield vector oblivious linear evaluation (MV-qsVOLE), extending the highly efficient two-party qsVOLE [18] into the multi-party setting. This primitive serves as our core building block, it allows for secure authentication of shared values across verifiers and minimizing online computation. We define its ideal functionality in Section 3.2 and provide an efficient UC-secure construction in Section 3.3.

Implementation and Benchmark. We implement our protocol in C++ using the EMP toolkit and benchmark it on both arithmetic and Boolean circuits. Experimental results show that our protocol significantly outperforms the state-of-the-art protocol (referred to as SIF) [23]. For AES-128 circuit proofs, we achieve a speedup of 5.2× to 17.4×. In small-field settings (e.g., Boolean circuits), our prover is on average 9× faster than SIF, while the verifier achieves a substantial 47× speedup.

1.2 Related Works

Early research on MVZK primarily focused on expanding functional guarantees. Burmester and Desmedt [9] first explored MVZK to reduce broadcast overhead. Subsequent works introduced specialized security properties, such as fair zero-knowledge proofs [17] to prevent information leakage during prover collusion, and crowd-verifiable zero-knowledge [3] for scenarios with low-entropy verifiers. These protocols enriched the MVZK landscape, while they are impractical for large-scale circuits due to high computational costs or excessive round complexity.

Abe et al. [1] and Groth et al. [15] proposed non-interactive MVZK protocols based on Pedersen commitments and verifiable secret sharing. However, these constructions require public-key operations for every non-linear gate, resulting in significant computational overhead that limited its practical scalability. To bypass the inefficiency of public-key operations, recent practical MVZK protocols have transitioned toward symmetric-key techniques. Yang and Wang [21] and Baum et al. [4] developed 2-round protocols that utilize symmetric-key primitives to achieve high performance. However, these solutions operate under the honest-majority assumption (e.g., $t < n/2$ or $n/4$) and necessitate at least two rounds of online communication, which can be a bottleneck in high-latency network environments. Most recently, Zhou et al. [23] recently proposed the first 1-round MVZK for dishonest majorities, yet their construction relies on the batch multiplication proofs from Quicksilver [20], which incur notable overhead for small-field particularly Boolean circuits.

1.3 Technical Overview

To clarify our key points, we first outline a generic framework for MVZK derived from existing VOLE-based ZKP constructions, and then detail how our protocol redefines this framework to achieve superior efficiency.

A Generic Framework for MVZK. A straightforward approach to constructing an MVZK protocol is to execute a standard VOLE-based ZKP (e.g., Mac'n'Cheese [5] or Quicksilver [20]) n times between a prover $\mathcal{P}$ and each verifier $\mathcal{V}_t$. This baseline framework generally follows the "Commit-and-Prove" paradigm: (1) Distributed Preprocessing: $\mathcal{P}$ invokes n independent VOLE functionalities to obtain commitments $[\boldsymbol{u}]^{(t)}$ for each $\mathcal{V}_t \in \{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$; (2) Uniform Commitment: $\mathcal{P}$ commits to the same witness $\boldsymbol{w}$ by sending $\boldsymbol{d}^{(t)} := \boldsymbol{w} - \boldsymbol{u}^{(t)}$ to all verifiers. Due to the additive homomorphism, all parties locally compute wire commitments of the addition gates; (3) Multiplication Check: $\mathcal{P}$ and each $\mathcal{V}_t$ cooperate to check all multiplication gates of the circuit $\mathcal{C}$ have been correctly computed. Compared to this, our scheme mainly offers two key construction:

- **Preprocessing with MV-qsVOLE Correlations.** Instead of instancing the VOLE functionality n times independently, we propose a new primitive called MV-qsVOLE (detailed in Section 3), MV-qsVOLE allows $\mathcal{P}$ to authenticate the same vector $\boldsymbol{u}$ across n verifiers simultaneously and supports authenticated quadratic relations. As depicted in the Quadratic phase of our functionality, once a set of values $\{u_i\}$ are authenticated, the prover computes $y_j := f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ and directly obtains the corresponding authenticated shares across all verifiers. Specifically, for each evaluation result y_j and each verifier $\mathcal{V}_t$, the functionality outputs a local key $k_{y_j}^{(t)}$ to $\mathcal{V}_t$ and a tag $m_{y_j}^{(t)}$ to $\mathcal{P}$ such that $k_{y_j}^{(t)} = m_{y_j}^{(t)} + y_j \cdot \Delta^{(t)}$.
- **Superior 1-Round Multiplication Check.** By leveraging the unique advantages of MV-qsVOLE, we extend the online proof introduced in [18] from the two-party to the multi-party setting. This transition eliminates the requirement for extension-field multiplications during verification, enabling our MVZK protocol to achieve superior online efficiency compared to the state-of-the-art work in [23]. First, $\mathcal{P}$ broadcasts $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{u}$ to the verifiers to generate the commitments of the wire values. Then $\mathcal{P}$ proves that all multiplication gates are processed properly. Consider a multiplication gate with input w_α, w_β. Given random preprocessed values $[u_\alpha], [u_\beta], [u_\gamma]$, alongside $[y = u_\alpha \cdot u_\beta]$ generated using MV-qsVOLE. After $\mathcal{P}$ broadcasts $d_i := w_i - u_i$ for $i \in \{\alpha, \beta, \gamma\}$, the verifiers only need to compute $[w_\gamma] = [u_\gamma] + d_\gamma$. In this way, for $t \in [n]$, $\mathcal{P}$ holds $u_i, m_{u_i}^{(t)}$ and $\mathcal{V}_t$ holds $\Delta^{(t)}, k_{u_i}^{(t)}$ for $i \in \{\alpha, \beta\}$, $\mathcal{P}$ holds $w_\gamma, m_{w_\gamma}^{(t)}$ and $\mathcal{V}_t$ holds corresponding $k_{w_\gamma}^{(t)}$. Given linear polynomials $p_i^{(t)}(X) = m_{u_i}^{(t)} + u_i \cdot X$ for $i \in \{\alpha, \beta\}, p_\gamma^{(t)}(X) = m_{w_\gamma}^{(t)} + w_\gamma \cdot X$, and $p_y^{(t)}(X) = m_y^{(t)} + y \cdot X$. We have the following $f^{(t)}(X)$:

$$\begin{aligned} f^{(t)}(X) &= d_\beta \cdot p_{u_\alpha}^{(t)}(X) + d_\alpha \cdot p_{u_\beta}^{(t)}(X) + p_y^{(t)}(X) + d_\beta \cdot d_\alpha \cdot X - p_{w_\gamma}^{(t)}(X) \cdot X \\ &= \underbrace{a_0^{(t)}}_{d_\beta \cdot m_{u_\alpha}^{(t)} + d_\alpha \cdot m_{u_\beta}^{(t)} + m_y^{(t)} - m_{w_\gamma}^{(t)}} + \underbrace{a_1^{(t)}}_{(u_\alpha + d_\alpha) + (u_\beta + d_\beta) - w_\gamma} \cdot X \end{aligned} \tag{1}$$

$f^{(t)}(X)$ has the following characteristics: its coefficients are held by $\mathcal{P}$, and evaluation at $\Delta^{(t)}$ (called $B^{(t)}$) is held by $\mathcal{V}_t$. If the multiplication gate is

evaluated correctly, we have $a_1^{(t)} = (u_\alpha + d_\alpha) \cdot (u_\beta + d_\beta) - w_\gamma = w_\alpha w_\beta - w_\gamma = 0$. $f^{(t)}(X)$ would degenerate from a linear polynomial to a constant polynomial. $\mathcal{P}$ sends $A^{(t)} := a_0^{(t)}$ to $\mathcal{V}_t$ privately to check if $A^{(t)} \stackrel{?}{=} B^{(t)}$. The constant polynomial introduced in [18] designs an efficient way to batch check ℓ multiplication gates using a random oracle H. Specifically, after obtain $A_i^{(t)}, B_i^{(t)}$ for $i = 1$ to l, $\mathcal{P}$ computes and sends $h^{(t)} = \mathsf{H}(A_1^{(t)} \| \cdots \| A_l^{(t)})$ to $\mathcal{V}_t$ for $t \in [n]$, who checks if $h^{(t)} \stackrel{?}{=} \mathsf{H}(B_1^{(t)} \| \cdots \| B_l^{(t)})$ to verify at once. We defer the details of our main protocol in Section 4.1.

2 Preliminaries

2.1 Notations

Let $\lambda \in \mathbb{N}$ be the security parameter, $\mathcal{V}_t$ denote the t-th verifier. For any integer $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. We write $x \leftarrow \mathcal{S}$ for sampling x uniformly from a finite set $\mathcal{S}$, and use bold lowercase letters (e.g., $\boldsymbol{x}$) for vectors. Let $\mathcal{H}$ denote the set of honest parties and S be the storage space for quadratic correlations. We consider a circuit $\mathcal{C}$ over a base field $\mathbb{F}_p$ and its extension $\mathbb{F}_{p^r}$, with input wires $\mathcal{I}_{in}$ of length m. The circuit is defined by a set of gates $\mathcal{G}$, where each gate $j \in \mathcal{G}$ is a tuple $(f_j, \mathcal{I}_j, \gamma_j)$ with input indices $\mathcal{I}_j$ and output index γ_j. Each f_j is a degree-2 polynomial that decomposes into $f_j = \sum_{k=0}^{2} g_{j,k}$, where $g_{j,k}$ is a homogeneous polynomial of degree k. Finally, $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ denotes a cryptographic hash function, modeled as a random oracle for online batch verification.

2.2 Security Model

The security analysis of our proposed MVZK scheme is conducted within the UC framework introduced by Canetti [10]. We demonstrate that for any PPT real-world adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ in the ideal world such that no PPT environment $\mathcal{Z}$ can distinguish between the real execution $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ and the ideal execution $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Communication Model. We assume secure point-to-point channels between the prover and each verifier, alongside authenticated channels among all verifiers. To instantiate a broadcast channel, we utilize the standard echo-broadcast protocol [13]. Consequently, our protocol concludes immediately after the prover broadcasts the proof and sends the auxiliary proofs to all verifiers privately in a single round. The verifiers then independently check the proof's validity.

2.3 VOLE and VOLE-based Commitments

Vector Oblivious Linear Evaluation. A VOLE correlation allows a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ to establish correlated randomness. $\mathcal{P}$ inputs $(\boldsymbol{u}, \boldsymbol{m}) \in \mathbb{F}_p^n \times \mathbb{F}_p^n$ and $\mathcal{V}$ inputs $\Delta \in \mathbb{F}_p$ and learns $\boldsymbol{k} \in \mathbb{F}_p^n$ such that $\boldsymbol{k} = \boldsymbol{u} \cdot \Delta + \boldsymbol{m}$.

Information-Theoretic Message Authentication Code (IT-MAC). The VOLE is typically utilized as an IT-MAC. For an element $\boldsymbol{u} \in \mathbb{F}_p^n$, an IT-MAC commitment $[\boldsymbol{u}]$ is represented by a pair $(\boldsymbol{m}, \boldsymbol{k})$, where the prover $\mathcal{P}$ holds the tag $\boldsymbol{m}$ and the verifier $\mathcal{V}$ holds the local key $\boldsymbol{k}$ satisfying $\boldsymbol{k} = \boldsymbol{m} + \boldsymbol{u} \cdot \Delta$, where $\Delta \in \mathbb{F}_p$ is a global key known only to $\mathcal{V}$. $\mathcal{P}$ opens $[\boldsymbol{u}]$ by sending $(\boldsymbol{u}, \boldsymbol{m})$ to $\mathcal{V}$, who verifies if $\boldsymbol{k} = \boldsymbol{u} \cdot \Delta + \boldsymbol{m}$ holds. This commitment provides perfect hiding (since $\boldsymbol{m}$ acts as a one-time pad) and statistical binding (since $\mathcal{P}$ cannot open $[\boldsymbol{u}]$ to a different value $\boldsymbol{u}' \neq \boldsymbol{u}$ without successfully guessing the secret Δ, except with negligible probability). Furthermore, this commitment is linearly homomorphic.

Zero-Knowledge Proof Functionality. We adopt the ideal functionality $\mathcal{F}_{\mathsf{MVZK}}$ from [21] (see Fig. 1). While [21] assumes an honest majority, we consider the more stringent dishonest-majority setting, where the adversary $\mathcal{A}$ can corrupt the prover and up to $n-1$ verifiers. Our functionality achieves *security with abort*, ensuring privacy and correctness conditional on protocol completion.

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

The functionality interacts with a prover $\mathcal{P}$, n verifiers $\mathcal{V}_1, \ldots \mathcal{V}_n$ and an adversary $\mathcal{A}$. Let $\mathcal{H}$ denote the set of honest parties. $\mathcal{F}_{\mathsf{MVZK}}$ operates as follows:

1. Upon receiving $(\textsc{Prove}, \mathrm{sid}, \mathcal{C}, \boldsymbol{w})$ from $\mathcal{P}$ and $(\textsc{Verify}, \mathrm{sid}, \mathcal{C})$ from $\mathcal{V}_t$ for all $t \in [n]$, set $b := \mathsf{true}$ if $\mathcal{C}(\boldsymbol{w}) = 1$ and $b := \mathsf{false}$ otherwise.
2. Send (b, sid) to $\mathcal{A}$. For each $\mathrm{H} \in \mathcal{H}$, wait for an input from $\mathcal{A}$, then do:
   - If it is $(\textsc{Continue}, \mathrm{sid}, \mathrm{H})$, send $(\textsc{Output}, \mathrm{sid}, b)$ to H.
   - If it is $(\textsc{Abort}, \mathrm{sid}, \mathrm{H})$, send $(\textsc{Abort}, \mathrm{sid})$ to H.

Fig. 1: The Functionality $\mathcal{F}_{\mathsf{MVZK}}$: Zero-Knowledge for Multiple Verifiers

2.4 Programmable VOLE

Generating a large volume of VOLE correlated randomness efficiently is crucial for practical applications. State-of-the-art approaches rely on pseudorandom correlation generators (PCGs). Informally, a PCG enables parties to locally expand short seeds into massive correlations. The VOLE correlations generated by PCGs are inherently random, meaning that $\mathcal{P}$ cannot freely choose the authenticated vector $\boldsymbol{u}$. This becomes problematic when the same $\boldsymbol{u}$ must be reused across multiple VOLE instances. To address this, Rachuri and Scholl proposed *programmable subfield VOLE (psVOLE)* in [19], In psVOLE, the prover $\mathcal{P}$ chooses a seed sd and expand it into a vector of ℓ field elements via a deterministic function Expand (instantiated with a secure PRG). This mechanism allows $\mathcal{P}$ to enforce consistency across multiple instances by reusing the same seed. Rachuri et al. [19] provided a PCG-style protocol to efficiently realize the $\mathcal{F}_{\mathrm{psVOLE}}^{p,r}$ functionality (see Fig. 2). Interested readers may refer to [19] for further details.

Functionality $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$

Define a deterministic expansion function $\mathsf{Expand} : \mathcal{S} \times \mathbb{Z} \to \mathbb{F}_p^\ell$.
Initialize: Upon receiving $(\textsc{Init}, \mathrm{sid})$ from $\mathcal{P}$ and $\mathcal{V}$, do:
- If $\mathcal{V}$ is honest, sample $\Delta \leftarrow \mathbb{F}_{p^r}$; otherwise, receive Δ from $\mathcal{A}$.
- Store Δ, send $(\textsc{Init}, \mathrm{sid}, \Delta)$ to $\mathcal{V}$, and ignore subsequent $(\textsc{Init})$ commands.

Authenticate over Subfield: Upon receiving $(\textsc{Authsub}, \mathrm{sid}, \ell, \mathrm{sd})$ from $\mathcal{P}$ and $(\textsc{Authsub}, \mathrm{sid}, \ell)$ from $\mathcal{V}$, do:
- Compute $\boldsymbol{u} := \mathsf{Expand}(\mathrm{sd}, \ell) \in \mathbb{F}_p^\ell$.
- If $\mathcal{V}$ is honest, sample $\boldsymbol{k} \leftarrow \mathbb{F}_{p^r}^\ell$; otherwise, receive $\boldsymbol{k}$ from $\mathcal{A}$.
- If $\mathcal{P}$ is honest, compute $\boldsymbol{m} := \boldsymbol{k} - \boldsymbol{u} \cdot \Delta \in \mathbb{F}_{p^r}^\ell$; otherwise, receive $\boldsymbol{m} \in \mathbb{F}_{p^r}^\ell$ from $\mathcal{A}$. Then recompute $\boldsymbol{k} := \boldsymbol{u} \cdot \Delta + \boldsymbol{m} \in \mathbb{F}_{p^r}^\ell$.
- Output $(\textsc{Authsub}, \mathrm{sid}, \boldsymbol{u}, \boldsymbol{m})$ to $\mathcal{P}$ and $(\textsc{Authsub}, \mathrm{sid}, \boldsymbol{k})$ to $\mathcal{V}$.

Fig. 2: The Functionality $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$: Programmable Subfield VOLE

3 Multi-Verifier Quadratic Subfield VOLE

3.1 Programmable Quadratic Subfield VOLE

We define *programmable quadratic subfield VOLE (p-qsVOLE)* functionality, denoted as $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ in Fig. 3. The functionality incorporates the features of $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$ and $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, enabling the authentication of specific values and generating additional correlations to minimize online complexity. We provide the protocol $\Pi^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ in Appendix A.

Functionality $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$

Initialize: Upon receiving $(\textsc{Init}, \mathrm{sid})$ from $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ works as follows:
- If $\mathcal{V}$ is honest, sample $\Delta \leftarrow \mathbb{F}_{p^r}$; otherwise, receive Δ from $\mathcal{A}$.
- Store Δ and send $(\textsc{Init}, \mathrm{sid}, \Delta)$ to $\mathcal{V}$, and ignore all subsequent $(\textsc{Init})$ commands.
- $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ initializes the set S as empty.

Authenticate over Subfield: Upon receiving $(\textsc{Authsub}, \mathrm{sid}, \ell, \mathrm{sd}, \{\tau_i\}_{i\in[\ell]})$ from $\mathcal{P}$ and $(\textsc{Authsub}, \mathrm{sid}, \ell, \{\tau_i\}_{i\in[\ell]})$ from $\mathcal{V}$, do:
- If $(\tau_i, \cdot)$ exists in S for any $i \in [\ell]$, it aborts.
- $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ follows the same procedure as $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$ to generate ℓ correlations.
- Store $(\tau_i, u_{\tau_i}, m_{u_{\tau_i}}, k_{u_{\tau_i}})$ in S for every $i \in [\ell]$.

Quadratic: On input $(\textsc{Quad}, \mathrm{sid}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ from $\mathcal{P}$ and $\mathcal{V}$, do:
- If $(i, \cdot)$ exists for every $i \in \mathcal{I}_j$, retrieve $\{(i, u_i, \cdot)\}_{i\in\mathcal{I}_j}$ from S. Otherwise, abort.
- Compute $y_j := f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ for every $j \in \mathcal{G}$.
- If $\mathcal{V}$ is honest, sample $\{k_{y_j}\}_{j\in\mathcal{G}} \leftarrow \mathbb{F}_{p^r}^{|\mathcal{G}|}$; otherwise, receive $\{k_{y_j}\}_{j\in\mathcal{G}}$ from $\mathcal{A}$.
- If $\mathcal{P}$ is honest, compute $\{m_{y_j} := k_{y_j} - y_j \cdot \Delta\}_{j\in\mathcal{G}}$; otherwise, receive $\{m_{y_j}\}_{j\in\mathcal{G}}$ from $\mathcal{A}$, and recompute $\{k_{y_j} := y_j \cdot \Delta + m_{y_j}\}_{j\in\mathcal{G}}$.
- Output $(\textsc{Quad}, \mathrm{sid}, \{[y_j]\}_{j\in\mathcal{G}})$ to $\mathcal{P}$ and $\mathcal{V}$.

Fig. 3: The Functionality $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$: Programmable Quadratic Subfield VOLE

3.2 Multi-Verifier Quadratic Subfield VOLE

To construct a multi-verifier proof, we generalize the two-party $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ to the multi-party setting, we call this new form of correlation *multi-verifier quadratic subfield VOLE (MV-qsVOLE)*. More precisely, in MV-qsVOLE, there is a prover $\mathcal{P}$ and n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$. Each verifier $\mathcal{V}_t$ has $\Delta^{(t)} \in \mathbb{F}_{p^r}$. Given a circuit description $\{(f_j, \mathcal{I}_j)_{j\in\mathcal{G}}\}$, the $\mathcal{P}$ holds $\boldsymbol{u} \in \mathbb{F}_p^\ell$. For every verifier, $\mathcal{P}$ holds $\boldsymbol{m}^{(t)} \in \mathbb{F}_{p^r}^\ell$ and $\mathcal{V}_t$ holds $\boldsymbol{k}^{(t)} \in \mathbb{F}_{p^r}^\ell$ such that $\boldsymbol{k}^{(t)} = \boldsymbol{u} \cdot \Delta^{(t)} + \boldsymbol{m}^{(t)}$. In addition, $\boldsymbol{u}$ satisfies $f_i(\boldsymbol{u}) = u_i$ for all $i \in \mathcal{I}$, and $u_i \leftarrow \mathbb{F}_p$ for $i \notin \mathcal{I}$. We define the concrete functionality of $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ in Fig. 4.

Functionality $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$

The functionality interacts with $\mathcal{P}$, $\mathcal{V}_1, \ldots, \mathcal{V}_n$ and an adversary $\mathcal{A}$.

Initialize: Upon receiving $(\textsc{Init}, \mathrm{sid})$ from $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$, do:

- For each $t \in [n]$, if $\mathcal{V}_t \in \mathcal{H}$, sample $\Delta^{(t)} \leftarrow \mathbb{F}_{p^r}$; otherwise, receive $\Delta^{(t)}$ from $\mathcal{A}$.
- Store $\{\Delta^{(t)}\}_{t\in[n]}$ and send $(\textsc{Init}, \mathrm{sid}, \Delta^{(t)})$ to $\mathcal{V}_t$ for every $t \in [n]$.

Authenticate over Subfield: On input $(\textsc{Authsub}, \mathrm{sid}, \ell, \{\tau_i\}_{i\in[\ell]})$ from $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$, do:

- If $(\tau_i, \cdot)$ exists in S for any $i \in [\ell]$, it aborts.
- If $\mathcal{V}_t^* \notin \mathcal{H}$, receive $\boldsymbol{k}^{(t)} \in \mathbb{F}_{p^r}^\ell$ from $\mathcal{A}$; otherwise, sample $\boldsymbol{k}^{(t)} \leftarrow \mathbb{F}_{p^r}^\ell$.
- If $\mathcal{P}$ is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}_p^\ell$ and compute $\boldsymbol{m}^{(t)} := \boldsymbol{k}^{(t)} - \boldsymbol{u} \cdot \Delta^{(t)} \in \mathbb{F}_{p^r}^\ell$ for each $t \in [n]$; otherwise, receive $\boldsymbol{u} \in \mathbb{F}_p^\ell$ and $\boldsymbol{m}^{(t)} \in \mathbb{F}_{p^r}^\ell$ from $\mathcal{A}$, then recompute $\boldsymbol{k}^{(t)} := \boldsymbol{u} \cdot \Delta^{(t)} + \boldsymbol{m}^{(t)} \in \mathbb{F}_{p^r}^\ell$ for each $t \in [n]$.
- Store $(\tau_i, u_{\tau_i}, m^{(t)}_{u_{\tau_i}}, k^{(t)}_{u_{\tau_i}})$ in S for every $i \in [\ell], t \in [n]$.
- Send $(\textsc{Continue}, \mathrm{sid})$ to $\mathcal{A}$. Upon receiving $(\textsc{Continue}, \mathrm{H})$ from $\mathcal{A}$, send respective outputs to $\mathrm{H} \in \mathcal{H}$; otherwise, send $(\textsc{Abort}, \mathrm{sid})$ to H.

Quadratic: On input $(\textsc{Quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ from $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$, do:

- If $(i, \cdot)$ exists for every $i \in \mathcal{I}_j$, retrieve $\{(i, u_i, \cdot)\}_{i\in\mathcal{I}_j}$ from S; otherwise, abort.
- Compute $y_j := f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ for every $j \in \mathcal{G}$.
- If $\mathcal{V}_t^* \notin \mathcal{H}$, receive $\{k^{(t)}_{y_j}\}_{j\in\mathcal{G}} \in \mathbb{F}_{p^r}^{|\mathcal{G}|}$ from $\mathcal{A}$; otherwise, sample $\{k^{(t)}_{y_j}\}_{j\in\mathcal{G}} \leftarrow \mathbb{F}_{p^r}^{|\mathcal{G}|}$.
- If $\mathcal{P}$ is honest, compute $\{m^{(t)}_{y_j} := k^{(t)}_{y_j} - y_j \cdot \Delta^{(t)}\}_{j\in\mathcal{G}}$ for $t \in [n]$; otherwise, receive $\{m^{(t)}_{y_j}\}_{j\in\mathcal{G}}$ from $\mathcal{A}$, and recompute $\{k^{(t)}_{y_j} := y_j \cdot \Delta^{(t)} + m^{(t)}_{y_j}\}_{j\in\mathcal{G}}$ for $t \in [n]$.
- Send $(\textsc{Continue}, \mathrm{sid})$ to $\mathcal{A}$. Upon receiving $(\textsc{Continue}, \mathrm{H})$ from $\mathcal{A}$, send respective outputs to $\mathrm{H} \in \mathcal{H}$; otherwise, send $(\textsc{Abort}, \mathrm{sid})$ to H.

Fig. 4: The Functionality $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$: Multi-Verifier Quadratic Subfield VOLE

3.3 Efficient Construction for $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$

The Protocol for $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$.

High-Level Description. Before detailing our concrete construction, we give a high-level overview. Concisely, the $\mathcal{P}$ samples a random seed sd from the seed space $\mathcal{S}$ and compute $\boldsymbol{u} := \mathsf{Expand}(\mathrm{sd}, \ell') \in \mathbb{F}_p^{\ell'}$. Then, for each $t \in [n]$, $\mathcal{P}$ and $\mathcal{V}_t$ invoke an instance of $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$. Specifically, $\mathcal{P}$ sends sd to $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$,

which returns $\boldsymbol{u}$ and $\boldsymbol{m}^{(t)}$ for all $t \in [n]$ to $\mathcal{P}$ and returns $\boldsymbol{k}^{(t)}$ to $\mathcal{V}_t$, such that $\boldsymbol{k}^{(t)} = \boldsymbol{u} \cdot \Delta^{(t)} + \boldsymbol{m}^{(t)}$. Subsequently, all parties execute a consistency check to ensure that $\mathcal{P}$ uses consistent seeds across all instances of $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$. Note that, as long as $\mathsf{Expand}(\mathrm{sd}_1, \ell') = \mathsf{Expand}(\mathrm{sd}_2, \ell')$, it is guaranteed that $\mathcal{P}$ uses consistent seeds. Once the check passes, $\mathcal{P}$ generates the additional quadratic subfield VOLE correlations with all verifiers.

Consistency Check. Our MV-qsVOLE ensures that the prover $\mathcal{P}$ authenticates consistent random values across all verifiers. By integrating the programmable nature of psVOLE and adapting the consistency check mechanism from Zhou et al. [23], we guarantee the consistency of authenticated values. Moreover, we introduce an improvement to get a more efficient consistency check. We extend the base-field operations to the extension field, bypassing $\mathcal{O}(\lambda)$ redundant executions, thereby achieving superior computational efficiency (discussed in detail later). The process proceeds as follows: Let $\ell' := \ell + r$ denote the payload length and let $\boldsymbol{e} = (e_1, \ldots, e_r)^\top \in \mathbb{F}_{p^r}^r$ be a fixed, publicly known basis of $\mathbb{F}_{p^r}$ over $\mathbb{F}_p$. The parties invoke the coin-tossing functionality $\mathcal{F}^{p,r}_{\mathsf{COIN}}$ [23] to obtain a common random challenge vector $\boldsymbol{s} \leftarrow \mathbb{F}_{p^r}^{\ell'}$, instead of sampling from the base field $\mathbb{F}_p^{\ell'}$ as in [23]. To mask the payload in the extension field, $\mathcal{P}$ parses its PRG-generated vector $\boldsymbol{u}$ into a payload part $\boldsymbol{u}_a \in \mathbb{F}_p^{\ell'}$ and a padding part $\boldsymbol{u}_b \in \mathbb{F}_p^r$. Then $\mathcal{P}$ broadcasts $x := \boldsymbol{s}^\top \cdot \boldsymbol{u}_a + \boldsymbol{e}^\top \cdot \boldsymbol{u}_b \in \mathbb{F}_{p^r}$. Similarly, for every $t \in [n]$, $\mathcal{P}$ parses the MAC vector $\boldsymbol{m}^{(t)}$ into $\boldsymbol{m}_a^{(t)}$ and $\boldsymbol{m}_b^{(t)}$, and privately sends the corresponding MAC tag $w^{(t)} := \boldsymbol{s}^\top \cdot \boldsymbol{m}_a^{(t)} + \boldsymbol{e}^\top \cdot \boldsymbol{m}_b^{(t)} \in \mathbb{F}_{p^r}$ to $\mathcal{V}_t$. Concurrently, $\mathcal{V}_t$ parses its local key vector $\boldsymbol{k}^{(t)}$ into $\boldsymbol{k}_a^{(t)}$ and $\boldsymbol{k}_b^{(t)}$, and computes $v^{(t)} := \boldsymbol{s}^\top \cdot \boldsymbol{k}_a^{(t)} + \boldsymbol{e}^\top \cdot \boldsymbol{k}_b^{(t)} \in \mathbb{F}_{p^r}$. Upon receiving x and $w^{(t)}$, $\mathcal{V}_t$ checks whether $v^{(t)} \stackrel{?}{=} x \cdot \Delta^{(t)} + w^{(t)}$. We demonstrate that if a malicious $\mathcal{P}^*$ uses inconsistent seeds, they will be caught with overwhelming probability. $\Pi^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ is presented in Fig. 5.

Protocol $\Pi^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$

Public Parameters: Let $\boldsymbol{e} = (e_1, \ldots, e_r)^\top \in \mathbb{F}_{p^r}^r$ be a fixed and publicly known basis of the extension field $\mathbb{F}_{p^r}$ over the base field $\mathbb{F}_p$.
Initialization: On input $(\textsc{Init}, \mathrm{sid})$, for each $t \in [n]$, $\mathcal{P}$ and $\mathcal{V}_t$ send $(\textsc{Init}, \mathrm{sid})$ to the t-th instance of $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$, which returns $\Delta^{(t)} \in \mathbb{F}_{p^r}$ to $\mathcal{V}_t$.
Authentications over Subfield: On input $(\textsc{Authsub}, \mathrm{sid}, \ell)$, $\mathcal{P}$ and verifiers do:
1. All parties set $\ell' := \ell + r$. Then $\mathcal{P}$ picks a random seed $\mathrm{sd} \leftarrow \mathcal{S}$.
2. For each $t \in [n]$, $\mathcal{P}$ sends $(\textsc{Authsub}, \mathrm{sid}, \ell', \mathrm{sd})$ to the t-th instance of $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ while $\mathcal{V}_t$ sends $(\textsc{Authsub}, \mathrm{sid}, \ell')$ to the same instance.
3. $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ returns $\boldsymbol{u} \in \mathbb{F}_p^{\ell'}$, $\boldsymbol{m}^{(t)} \in \mathbb{F}_{p^r}^{\ell'}$ to $\mathcal{P}$, where $\boldsymbol{u} := \mathsf{Expand}(\mathrm{sd}, \ell')$, and returns $\boldsymbol{k}^{(t)} \in \mathbb{F}_{p^r}^{\ell'}$ to $\mathcal{V}_t$ such that $\boldsymbol{k}^{(t)} = \boldsymbol{u} \cdot \Delta^{(t)} + \boldsymbol{m}^{(t)}$.
   Let $\boldsymbol{u}$ be parsed as $(\boldsymbol{u}_a \parallel \boldsymbol{u}_b)$, where $\boldsymbol{u}_a \in \mathbb{F}_p^{\ell}$ and $\boldsymbol{u}_b \in \mathbb{F}_p^r$. Similarly, parse $\boldsymbol{m}^{(t)} = (\boldsymbol{m}_a^{(t)} \parallel \boldsymbol{m}_b^{(t)})$ and $\boldsymbol{k}^{(t)} = (\boldsymbol{k}_a^{(t)} \parallel \boldsymbol{k}_b^{(t)})$.
4. All parties perform the following consistency check:
   (a) $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$ send $(\textsc{Toss}, \mathrm{sid}, \ell')$ to $\mathcal{F}^{p,r}_{\mathsf{COIN}}$, which returns $\boldsymbol{s} \leftarrow \mathbb{F}_{p^r}^{\ell'}$.

(b) $\mathcal{P}$ broadcasts $x := \boldsymbol{s}^\top \cdot \boldsymbol{u}_a + \boldsymbol{e}^\top \cdot \boldsymbol{u}_b \in \mathbb{F}_{p^r}$ to all verifiers. Then for each $t \in [n]$, $\mathcal{P}$ sends $w^{(t)} := \boldsymbol{s}^\top \cdot \boldsymbol{m}_a^{(t)} + \boldsymbol{e}^\top \cdot \boldsymbol{m}_b^{(t)} \in \mathbb{F}_{p^r}$ to $\mathcal{V}_t$ privately.
(c) For each $t \in [n] : \mathcal{V}_t$ computes $v^{(t)} := \boldsymbol{s}^\top \cdot \boldsymbol{k}_a^{(t)} + \boldsymbol{e}^\top \cdot \boldsymbol{k}_b^{(t)} \in \mathbb{F}_{p^r}$. Then $\mathcal{V}_t$ checks if $v^{(t)} \stackrel{?}{=} x \cdot \Delta^{(t)} + w^{(t)}$. If not, $\mathcal{V}_t$ aborts.

5. $\mathcal{P}$ outputs $\{u_j, m_{u_j}^{(t)}\}_{t\in[n], j\in[\ell]}$ and for each $t \in [n]$, $\mathcal{V}_t$ outputs $\{k_{u_j}^{(t)}\}_{j\in[\ell]}$.

Quadratic:

- $\mathcal{P}$ sends $(\text{QUAD}, \text{sid}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ to the t-th instance of $\mathcal{F}_{\mathsf{p\text{-}qsVOLE}}^{p,r}$, so $\mathcal{V}_t$ is.
- $\mathcal{F}_{\mathsf{p\text{-}qsVOLE}}^{p,r}$ returns $(\{y_j, m_{y_j}^{(t)}\}_{j\in\mathcal{G}}$ for $t \in [n])$ to $\mathcal{P}$ and $(\{k_{y_j}^{(t)}\}_{j\in\mathcal{G}})$ to $\mathcal{V}_t$.
- $\mathcal{P}$ and every $\mathcal{V}_t$ outputs $\{[y_j]\}_{j\in\mathcal{G}}$.

Fig. 5: The Multi-Verifier qsVOLE Protocol that UC-Realizes $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}$

Consistency Check Performance Comparison. Our consistency check outperforms that in [23] over small-field circuits. Taking Boolean circuits as example (the extension degree $r = \lambda$). Costs are normalized to $\mathcal{O}(1)$ per base-field $\mathbb{F}_p$ operation. Under this metric, the computational complexity of the protocol by [23] is $\mathcal{O}(n \cdot \ell' \cdot \lambda^2)$. In contrast, our construction significantly improves this computational overhead to $\mathcal{O}(n \cdot \ell' \cdot \lambda \log \lambda)$. Regarding communication complexity, their protocol requires $\mathcal{O}(n \cdot \ell' \cdot \lambda^2)$ bits of communication, whereas our work optimizes this strictly to $\mathcal{O}(n \cdot \ell' \cdot \lambda)$ bits.

Security Analysis. For any field $\mathbb{F}_p$, we select an extension degree r such that $p^r \geq 2^\lambda$. First, we describe as following two cases.
Corrupted Prover. Let $\boldsymbol{u_1} := \mathsf{Expand}(\text{sd}_1, \ell')$ and $\boldsymbol{u_2} := \mathsf{Expand}(\text{sd}_2, \ell')$. Suppose $\mathcal{P}^*$ generates the same correlations $[\boldsymbol{u_1}]$ with $n-1$ verifiers but authenticates $[\boldsymbol{u_2}]$ with $\mathcal{V}_t$. During the consistency check, all parties obtain $\boldsymbol{s} \leftarrow \mathbb{F}_{p^r}^{\ell'}$ from $\mathcal{F}_{\mathsf{COIN}}^{p,r}$. $\mathcal{P}^*$ broadcasts $x = \boldsymbol{s}^\top \cdot \boldsymbol{u}_{a_1} + \boldsymbol{e}^\top \cdot \boldsymbol{u}_{b_1}$ to all verifiers. For $\mathcal{V}_t$, after obtaining $w^{(t)}$ sent by $\mathcal{P}^*$ privately, it checks $v^{(t)} \stackrel{?}{=} x \cdot \Delta^{(t)} + w^{(t)}$. Due to the binding property of VOLE-based commitment, the probability of passing the check equals:

$$\Pr[\boldsymbol{s}^\top \cdot \boldsymbol{u}_{a_1} + \boldsymbol{e}^\top \cdot \boldsymbol{u}_{b_1} = \boldsymbol{s}^\top \cdot \boldsymbol{u}_{a_2} + \boldsymbol{e}^\top \cdot \boldsymbol{u}_{b_2}] \tag{2}$$

we denote by $\mathcal{I}$ the set of indices where $\boldsymbol{u}_a \neq \boldsymbol{u}_b$. Above probability equals

$$\Pr\left[\sum_{i\in\mathcal{I}} s_i^\top \cdot \delta_i = -\boldsymbol{e}^\top \cdot (\boldsymbol{u}_{b_1} + \boldsymbol{u}_{b_2})\right] \tag{3}$$

$\boldsymbol{\delta} = \boldsymbol{u}_{a_1} - \boldsymbol{u}_{a_2} \in \mathbb{F}_p^{\ell'}$. If $\text{sd}_1 \neq \text{sd}_2$, then $\boldsymbol{\delta} \neq \boldsymbol{0}$ with overwhelming probability. Since $\boldsymbol{s}$ is sampled uniformly from $\mathbb{F}_{p^r}^{\ell'}$, by Lemma 1, the probability that the inner product evaluates to any fixed value (including $-\boldsymbol{e}^\top(\boldsymbol{u}_{b_1} - \boldsymbol{u}_{b_2})$) is exactly $1/p^r$. Therefore, a malicious $\mathcal{P}^*$ is caught with probability $1 - 1/p^r \geq 1 - 2^{-\lambda}$.
Corrupted Verifiers. The value broadcast by $\mathcal{P}$ is $x = \boldsymbol{s}^\top \cdot \boldsymbol{u}_a + \boldsymbol{e}^\top \cdot \boldsymbol{u}_b$. Since $\boldsymbol{e} = (e_1, \ldots, e_r)^\top$ is a basis of $\mathbb{F}_{p^r}$ over $\mathbb{F}_p$, the linear combination $\boldsymbol{e}^\top \cdot \boldsymbol{u}_b =$

$\sum_{i=1}^{r} u_{b,i} e_i$ can represent any element in the extension field $\mathbb{F}_{p^r}$. Given that each $u_{b,i}$ is chosen uniformly and independently from $\mathbb{F}_p$, the term $\boldsymbol{e}^\top \cdot \boldsymbol{u}_b$ is uniformly distributed in $\mathbb{F}_{p^r}$ and acts as a one-time pad to $\boldsymbol{s}^\top \cdot \boldsymbol{u}_a$. Consequently, the broadcast x is computationally indistinguishable from a uniform element in $\mathbb{F}_{p^r}$ to $\mathcal{V}_t^*$ who does not know $\boldsymbol{u}_b$. The same logic applies to $w^{(t)}$. Thus, the malicious verifiers learn no additional information about $\mathcal{P}$'s input $\boldsymbol{u}$ and $\boldsymbol{m}$.

Theorem 1. *Let* Expand *be a secure* PRG*. Then the protocol* $\Pi^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ *depicted in Fig. 5 UC-realizes the functionality* $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ *depicted in Fig. 4 in the* $\{\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}, \mathcal{F}^{p,r}_{\mathsf{COIN}}\}$*-hybrid world, in the presence of a static malicious adversary corrupting up to the prover and* $n-1$ *verifiers.*

Proof. The proof is provided in Appendix B.

Lemma 1. *Let* $\mathbb{F}_p$ *be a field of prime order* p*. Let* $\boldsymbol{u}$ *be a column vector over field* $\mathbb{F}_p^\ell$ *whose elements are all non-zero. Let* $\boldsymbol{s}$ *be a column vector that is uniformly sampled from* $\mathbb{F}_{p^r}^\ell$*. Then we have* $\Pr[\boldsymbol{u}^\top \cdot \boldsymbol{s} = 0] = 1/p^r$.

4 1-Round MVZK with Superior Online Overhead

4.1 MVZK Protocol

Our multi-verifier zero-knowledge protocol Π_{MVZK} is shown in Fig. 6. We describe the details about our protocols applied to arbitrary degree-2 polynomial gates. As established earlier, the target circuit $\mathcal{C}$ is represented as $\{(f_j, \mathcal{I}_j, \gamma_j)\}_{j\in\mathcal{G}}$. During the preprocessing phase, $\mathcal{P}$ and $\mathcal{V}_1, \ldots \mathcal{V}_n$ run $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ to generate random authenticated values for the circuit inputs and the output wires of gates. Subsequently, for each multiplication gate in the circuit, $\mathcal{P}$ obtains the additional authenticated values $[y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]$ via $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$ with every $\mathcal{V}_t$. During the online phase, $\mathcal{P}$ broadcasts the masked input $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{u} \in \mathbb{F}_p^m$ to all verifiers. All parties locally reconstruct the authenticated input values via $[\boldsymbol{w}] := [\boldsymbol{u}] + \boldsymbol{d}$. Then, $\mathcal{P}$ broadcasts $\boldsymbol{d}_{\boldsymbol{\gamma}} \in \mathbb{F}_p^{|\mathcal{G}|}$ corresponding to all gate output wires to verifiers, enabling every party to derive $[w_{\gamma_j}]$ locally. To ensure that each gate is correctly computed, we extend the techniques from [18], originally designed for qsVOLE correlations to our MV-qsVOLE correlations, yielding more efficient zero-knowledge proofs for multiple verifiers. More precisely, given $\{d_i\}_{i\in\mathcal{I}_j}$ from $\mathcal{P}$, and f_j with k degree-2 terms, let the set $\{(c_i, \alpha_i, \beta_i)\}_{i\in[k]}$ consist of tuples where $c_i \in \mathbb{F}_p$ are coefficients and α_i, β_i are input indices for each degree-2 term of f_j. Besides, we define the polynomials $p^{(t)}_{u_i}(X) = m^{(t)}_{u_i} + u_i \cdot X$ for $i \in \{(\alpha_i, \beta_i)\}_{i\in[k]}$ and $t \in [n]$, $p^{(t)}_{\gamma_j}(X) = m^{(t)}_{\gamma_j} + w^{(t)}_{\gamma_j} \cdot X$, and $p^{(t)}_{y_j}(X) = m^{(t)}_{y_j} + y_j \cdot X$. Then $f^{(t)}(X)$

equals:

$$\begin{aligned}
&\sum_{i\in[k]} c_i \cdot (d_{u_{\alpha_i}} \cdot p^{(t)}_{u_{\beta_i}}(X) + d_{u_{\beta_i}} \cdot p^{(t)}_{u_{\alpha_i}}(X)) \\
&\qquad\qquad + p^{(t)}_{y_j}(X) + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot X - p^{(t)}_{\gamma_j}(X) \\
&= \sum_{i\in[k]} c_i \cdot (d_{u_{\alpha_i}} \cdot m^{(t)}_{u_{\beta_i}} + d_{u_{\beta_i}} \cdot m^{(t)}_{u_{\alpha_i}}) + m^{(t)}_{y_j} - m^{(t)}_{w_{\gamma_j}} \\
&\qquad\qquad + (f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) - w^{(t)}_{\gamma_j}) \cdot X
\end{aligned} \tag{4}$$

The coefficients of $f^{(t)}(X)$ are held by $\mathcal{P}$, and its evaluation at $\Delta^{(t)}$ is held by $\mathcal{V}_t$. $f^{(t)}(X)$ is a constant polynomial if and only if $f^{(t)}_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) = w_{\gamma_j}$, After this, $\mathcal{P}$ and $\mathcal{V}_t$ conduct the batch check and verify the correctness of the final output wire.

Protocol Π_{MVZK}

Inputs: $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$ hold a circuit $\mathcal{C}$ over any field $\mathbb{F}_p$. $\mathcal{P}$ also holds the witness $\boldsymbol{w}$ such that $\mathcal{C}(\boldsymbol{w}) = 1$ and $|\boldsymbol{w}| = m$ (i.e., $|\mathcal{I}_{\text{in}}| = m$). Let $\{\tau_i\}_{i\in[m]}$ represent the indices of the input wires of $\mathcal{C}$, and let $\{\tau_i\}_{i\in[m+1,m+|\mathcal{G}|]}$ represent the indices of the output wires of gates in $\mathcal{C}$.

Preprocessing Phase: The values of the circuit $\mathcal{C}$ are unknown at this phase.

1. $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$ send $(\textsc{Init}, \text{sid})$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$, which returns $\Delta^{(t)}$ to $\mathcal{V}_1, \ldots, \mathcal{V}_n$.
2. $\mathcal{P}$ and $\mathcal{V}_1, \ldots, \mathcal{V}_n$ send $(\textsc{Authsub}, \text{sid}, m+|\mathcal{G}|, \{\tau_i\}_{i\in[m+|\mathcal{G}|]})$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$, which returns $\{[\boldsymbol{u}_{\tau_i}]\}_{i\in[m+|\mathcal{G}|]}$ to the parties.
3. Precomputing for gates $\{(f_j, \mathcal{I}_j, \cdot) \in \mathcal{C}\}_{j\in\mathcal{G}}$:
   - $\mathcal{P}$ and every $\mathcal{V}_t$ send $(\textsc{Quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$, which returns $\{[y_j] = [f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]\}_{j\in\mathcal{G}}$ to the parties.

Online Phase: The input values are known by the parties respectively.

1. For each $\tau_i \in \mathcal{I}_{\text{in}}$: $\mathcal{P}$ broadcasts $d_{\tau_i} := w_{\tau_i} - u_{\tau_i} \in \mathbb{F}_p$. All parties locally compute $[w_{\tau_i}] := [u_{\tau_i}] + d_{\tau_i}$.
2. In a topological order, for each gate $(f_j, \mathcal{I}_j, \gamma_j) \in \mathcal{C}$, this is the j'-th gate in $\mathcal{G}$:
   - $\mathcal{P}$ broadcasts $d_{\gamma_j} := f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{m+j'}}$. All parties compute $[w_{\gamma_j}] := [u_{\tau_{m+j'}}] + d_{\gamma_j}$.
3. For each gate j, and for $t \in [n]$, $\mathcal{P}$ and $\mathcal{V}_t$ hold $(\{[w_i]\}_{i\in\mathcal{I}_j}, [w_{\gamma_j}])$ where $w_i = u_i + d_i$ for every $i \in \mathcal{I}_j \cup \gamma_j$ and $w_{\gamma_j} = f_j(\{w_i\}_{i\in\mathcal{I}_j})$. That is $\mathcal{P}$ holds $w_i, m^{(t)}_i$, $\mathcal{V}_t$ holds $k^{(t)}_i, \Delta^{(t)}$ such that $k^{(t)}_i = w_i \cdot \Delta^{(t)} + m^{(t)}_i$ as well as $\mathcal{P}$ holds $w_{\gamma_j}, m^{(t)}_{\gamma_j}$, $\mathcal{V}_t$ holds $k^{(t)}_{\gamma_j}, \Delta^{(t)}$ such that $k^{(t)}_{\gamma_j} = w_{\gamma_j} \cdot \Delta^{(t)} + m^{(t)}_{\gamma_j}$. Then for $t \in [n]$:
   - $\mathcal{P}$ computes $m^{(t)}_{j,\text{zero}} := g_{j,2}(\{m^{(t)}_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m^{(t)}_{u_i}\}_{i\in\mathcal{I}_j}) + m^{(t)}_{y_j} - m^{(t)}_{w_{\gamma_j}}$.
   - $\mathcal{V}_t$ computes $k^{(t)}_{j,\text{zero}} := g_{j,2}(\{k^{(t)}_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{k^{(t)}_{u_i}\}_{i\in\mathcal{I}_j}) + k^{(t)}_{y_j} - k^{(t)}_{w_{\gamma_j}} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot \Delta^{(t)}$.
4. For $t \in [n]$, $\mathcal{P}$ and $\mathcal{V}_t$ perform the following check to verify that $m^{(t)}_{j,\text{zero}} = k^{(t)}_{j,\text{zero}}$ for all $j \in \mathcal{G}$.
   - $\mathcal{P}$ computes $A^{(t)} := \mathsf{H}(m^{(t)}_{1,\text{zero}}, \ldots, m^{(t)}_{|\mathcal{G}|,\text{zero}})$, and sends $A^{(t)}$ to $\mathcal{V}_t$ privately.
   - $\mathcal{V}_t$ computes $B^{(t)} := \mathsf{H}(k^{(t)}_{1,\text{zero}}, \ldots, k^{(t)}_{|\mathcal{G}|,\text{zero}})$, and checks whether $A^{(t)} = B^{(t)}$. If the check fails, $\mathcal{V}_t$ outputs false and aborts.

5. For the output wire h, $\mathcal{P}$ and $\mathcal{V}_t$ hold $[w_h]$ with $k_h^{(t)} = w_h \cdot \Delta^{(t)} + m_h^{(t)}$. They verify that $w_h = 1$ as follows.
   - Concurrently with the previous step, $\mathcal{P}$ sends $m_h^{(t)}$ to $\mathcal{V}_t$ privately.
   - $\mathcal{V}_t$ checks whether $k_h^{(t)} = m_h^{(t)} + \Delta^{(t)}$. If the check fails, $\mathcal{V}_t$ outputs false and aborts. Otherwise, $\mathcal{V}_t$ outputs true.

Fig. 6: The Multi-Verifier Zero-Knowledge Protocol that UC-Realizes $\mathcal{F}_{\mathsf{MVZK}}$

Theorem 2. Π_{MVZK} *depicted in Fig. 6 UC-realizes* $\mathcal{F}_{\mathsf{MVZK}}$ *depicted in Fig. 1 with computational security in the* $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$*-hybrid model, in the presence of a static malicious adversary corrupting the prover and up to* $n-1$ *verifiers.*

Proof. We prove the UC-security of Π_{MVZK} by constructing a simulator $\mathcal{S}$ that emulates $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$ and the random oracle H while interacting with the adversary $\mathcal{A}$. We first describe that how $\mathcal{S}$ simulate H. $\mathcal{S}$ maintains a hash list L_h. For any query $y \in \{0,1\}^*$ to H, if $\exists(y, h_y) \in L_h$, return h_y, else samples $h_y \leftarrow \{0,1\}^\lambda$, return h_y to $\mathcal{A}$ and add (y, h_y) to L_h. We outline the workflow of $\mathcal{S}$ and prove that $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{MVZK}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}^{\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r},\mathcal{RO}}_{\Pi_{\mathsf{MVZK}},\mathcal{A},\mathcal{Z}}$.

Corrupted Verifiers. Up to $n-1$ verifiers are malicious. The security purpose is that the malicious verifiers cannot learn any information about the prover's $\boldsymbol{w}$. We prove this by constructing a simulator $\mathcal{S}$ that does not hold $\boldsymbol{w}$, but is able to generate the "fake proof" that any honest verifier would accept.

1. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$ for the adversary $\mathcal{A}$. Upon receiving $\textsc{Init}$, $\Delta^{(t)}$ and $\{k_{u_{\tau_i}}^{(t)}\}_{i\in[m+|\mathcal{G}|],t\in[n]}, \{k_{y_j}^{(t)}\}_{j\in\mathcal{G}}$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$, $\mathcal{S}$ stores these values.
2. $\mathcal{S}$ generates $\tilde{\boldsymbol{w}} \leftarrow \mathbb{F}_p^m$ and uses $\tilde{\boldsymbol{w}}$ to execute the step 1-3 in the online phase of Π_{MVZK}. $\mathcal{S}$ computes $\{k_{j,\mathrm{zero}}^{(t)}\}_{j\in\mathcal{G},t\in[n]}$ based on the stored values. If $((k_{1,\mathrm{zero}}^{(t)}, \ldots, k_{|\mathcal{G}|,\mathrm{zero}}^{(t)}), h_y^{(t)}) \notin L_h$, $\mathcal{S}$ sends $A^{(t)} \leftarrow \{0,1\}^\lambda$ to $\mathcal{V}_t^* \notin \mathcal{H}$, and add $((k_{1,\mathrm{zero}}^{(t)}, \ldots, k_{|\mathcal{G}|,\mathrm{zero}}^{(t)}), A^{(t)})$ to L_h; otherwise, sends $A^{(t)} = h_y^{(t)}$ to $\mathcal{V}_t^*$.
3. $\mathcal{S}$ computes $k_h^{(t)}$ and then sets $m_h^{(t)} = k_h^{(t)} - \Delta^{(t)}$, where h is the single output wire. Then $\mathcal{S}$ sends $m_h^{(t)}$ to $\mathcal{V}_t^* \notin \mathcal{H}$.
4. Whenever $\mathcal{A}$ attempts to make an honest party $\mathrm{H} \in \mathcal{H}$ abort, $\mathcal{S}$ returns $(\textsc{Abort}, \mathrm{sid}, \mathrm{H})$ to $\mathcal{F}_{\mathsf{MVZK}}$ when $\mathcal{F}_{\mathsf{MVZK}}$ sends $(\textsc{Continue}, \mathrm{sid})$.

Due to the security of $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$, $\mathcal{A}$ learns nothing about the random vectors $\boldsymbol{u}$ used to mask the wire values. Therefore, even if $\mathcal{S}$ uses a randomly selected $\tilde{\boldsymbol{w}}$ as the prover's secret input, $\mathcal{A}$ cannot detect this during steps 1-4 in the online phase of Π_{MVZK}. In the final step of the online phase, the MAC $m_h^{(t)}$ are computed via the keys from $\mathcal{V}_t^*$, $\mathcal{V}_t^*$ naturally verifies $m_h^{(t)} = k_h^{(t)} - \Delta^{(t)}$ and accepts. We easily obtain that $\mathrm{EXEC}^{\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r},\mathcal{RO}}_{\Pi_{\mathsf{MVZK}},\mathcal{A},\mathcal{Z}} \equiv \mathrm{EXEC}_{\mathcal{F}_{\mathsf{MVZK}},\mathcal{S},\mathcal{Z}}$.

Corrupted Prover. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}$ receives $\textsc{Init}$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$, samples $\Delta^{(t)} \leftarrow \mathbb{F}_{p^r}$, and stores $\{u_{\tau_i}, m_{u_{\tau_i}}^{(t)}\}_{i\in[m+|\mathcal{G}|],t\in[n]}$ that $\mathcal{P}$ sends to $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}^{p,r}$. Then $\mathcal{S}$ computes the

corresponding keys $\{k_{u_{\tau_i}}^{(t)}\}_{i\in[m+|\mathcal{G}|],t\in[n]}$. Upon receiving $\{m_{y_j}^{(t)}\}_{j\in\mathcal{G},t\in[n]}$ from $\mathcal{A}$. $\mathcal{S}$ computes and stores $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}, \{k_{y_j}^{(t)}\}_{j\in\mathcal{G}}$ for every $j \in \mathcal{G}$.
2. For $\tau_i \in \mathcal{I}_{\text{in}}$, upon receiving d_{τ_i} from $\mathcal{A}$, $\mathcal{S}$ computes and stores $w_{\tau_i} = u_{\tau_i} + d_{\tau_i}$.
3. $\mathcal{S}$ acts as the honest verifiers to execute the protocol. If the honest verifier outputs false, $\mathcal{S}$ sends $\boldsymbol{w} = \perp$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{MVZK}}$ and aborts. Otherwise, $\mathcal{S}$ sends the extracted ($\boldsymbol{w}$ and $\mathcal{C}$) to $\mathcal{F}_{\mathsf{MVZK}}$.

If the verifier in the real-world outputs true while the witness $\boldsymbol{w}$ sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{MVZK}}$ satisfies $\mathcal{C}(\boldsymbol{w}) = 0$, Then $\mathcal{A}$ can distinguish between the ideal and real-world executions. The values associated with the inputs and the output wires of ADD gates are computed correctly. Therefore, the required probability is bounded by the sum of the probabilities in the following two cases.

Case 1. $\mathcal{P}^*$ cannot forge the MAC tags for the output wires. Namely $\mathcal{C}(\boldsymbol{w}) = 0$. We assume that the output wire values of the former $j-1$ multiplication gates are correct. For the j-th multiplication gate, the parties hold $\{[w_i]\}_{i\in\mathcal{I}_j}, [w_{\gamma_j}]$. $\{w_i\}_{i\in\mathcal{I}_j}$ denotes the input wires MACs of j-th multiplication gate and w_{γ_j} denotes the output MACs of j-th multiplication gate. Let $[w_{\gamma_j}] = [f_j(\{w_i\}_{i\in\mathcal{I}_j})] + e_{\gamma_j}$ where $e_{\gamma_j} \in \mathbb{F}_p$ is the error introduced by $\mathcal{A}$ sending an incorrect value $d'_{\gamma_j} = d_{\gamma_j} + e_{\gamma_j}$, an honest $\mathcal{V}_t$ passes the check in step 4, meaning $A_j^{(t)} = B_j^{(t)}$. Concretely, the parties hold $\{[w_i]\}_{i\in\mathcal{I}_j}, [w'_{\gamma_j}] = [w_{\gamma_j}] + e_{\gamma_j}$. Given $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ and $f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) = w_{\gamma_j}$, for each honest $\mathcal{V}_t \in \mathcal{H}$, the value $k_{j,\text{zero}}^{(t)} = g_{j,2}(\{k_{u_i}^{(t)} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{k_{u_i}^{(t)}\}_{i\in\mathcal{I}_j}) + k_{y_j}^{(t)} - k_{w'_{\gamma_j}}^{(t)} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot \Delta^{(t)} = m_{j,\text{zero}}^{(t)} - e_{\gamma_j} \cdot \Delta^{(t)}$. $m_{j,\text{zero}}^{(t)}$ is computed following the protocol description. $\{m_{j,\text{zero}}^{(t)}\}_{j\in\mathcal{G}}$ and e_{γ_j} are known to $\mathcal{A}$. Since $\Delta^{(t)}$ has p^r possible values and $\mathcal{P}^*$ does not know $\Delta^{(t)}$, as does $\{k_{j,\text{ zero}}^{(t)}\}_{j\in\mathcal{G}}$. The probability $\mathsf{H}(x_1) = \mathsf{H}(x_2)$ where x_1 consists of $\{m_{i,\text{zero}}^{(t)}\}_{j\in\mathcal{G}}$ and x_2 consists of $\{k_{j,\text{zero}}^{(t)}\}_{j\in\mathcal{G}}$ is less than the sum of following two cases.

1. x_2 has been queried by $\mathcal{A}$ with probability q_{H}/p^r.
2. x_2 was not queried by $\mathcal{A}$ and $\mathsf{H}(x_1) = \mathsf{H}(x_2)$ with probability $1/2^\lambda$.

Case 2. $\mathcal{C}(\boldsymbol{w}) = 0$ and the MAC and keys of the output wire value are correctly computed from the MAC and keys of the input wires for every multiplication gate, meaning $[w_{\gamma_j}] = [f_j(\{w_i\}_{i\in\mathcal{I}_j})]$, $\mathcal{V}_t$ outputs true. Then $\mathcal{A}$ must send $m_h^{(t)} + \Delta^{(t)}$ to convince the honest verifier $\mathcal{C}(\boldsymbol{w}) = 1$, where $m_h^{(t)}$ is a MAC tag on the output wires known by $\mathcal{A}$. In other words, $\mathcal{A}$ learns $\Delta^{(t)}$, which occurs with probability at most $1/p^r$.

By the union bound, we conclude that $\text{EXEC}_{\mathcal{F}_{\mathsf{MVZK}},\mathcal{S},\mathcal{Z}}$ is statistically indistinguishable from $\text{EXEC}^{\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}},\mathcal{RO}}_{\Pi_{\mathsf{MVZK}},\mathcal{A},\mathcal{Z}}$ with adversarial advantage at most $n(q_{\mathsf{H}}+1)/p^r + n/2^\lambda$.

4.2 Cost Analysis

In this section, we quantify the cost analysis of computation for all parties in the online phase, and compare our MVZK protocol with the state-of-the-art

SIF [23]. In terms of communication, Both our protocol and SIF achieve optimal communication round complexity in the online phase, requiring only a single round of interaction from $\mathcal{P}$ to all verifiers.

Prover Cost. For j-th multiplication gate, $\mathcal{P}$ computes $d_{\gamma_j} := f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{m+j'}}$ and $\{m^{(t)}_{j,\text{zero}}\}_{t\in[n]}$, since $m^{(t)}_{y_j} - g_{j,2}(\{m^{(t)}_{u_i}\}_{i\in\mathcal{I}_j}) - m^{(t)}_{w_{\gamma_j}}$ and $m^{(t)}_{w_{\gamma_j}} = m^{(t)}_{u_{\tau_{m+j'}}}$ are independent of the witness $\boldsymbol{w}$, they can be precomputed during the preprocessing phase. Consequently, $\mathcal{P}$ only requires one evaluation of f_j and one addition over $\mathbb{F}_p$ to compute d_{γ_j}. Additionally, for each $t \in [n]$, $\mathcal{P}$ performs one evaluation of $g_{j,2}$ and one addition over $\mathbb{F}_{p^r}$ to compute $m^{(t)}_{j,\text{zero}}$.

Verifier Cost. For j-th multiplication gate, each $\mathcal{V}_t$ computes $k^{(t)}_{\gamma_j} := k^{(t)}_{u_{\tau_{m+j'}}} + d_{\gamma_j} \cdot \Delta^{(t)} \in \mathbb{F}_{p^r}$ and $k^{(t)}_{j,\text{zero}}$, where $k^{(t)}_{y_j} - g_{j,2}(\{k^{(t)}_{u_i}\}_{i\in\mathcal{I}_j})$ is independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{V}_t$ requires one scalar product of $d_{\gamma_j} \in \mathbb{F}_p$ with $\Delta^{(t)}$ and one addition over $\mathbb{F}_{p^r}$ to compute $k^{(t)}_{w_{\gamma_j}}$. And $\mathcal{V}_t$ requires one evaluation of $g_{j,2}$ over $\mathbb{F}_{p^r}$, one evaluation of f_j over $\mathbb{F}_p$, one scalar product on $\Delta^{(t)}$ over $\mathbb{F}_{p^r}$, and three additions over $\mathbb{F}_{p^r}$ to compute $k^{(t)}_{j,\text{zero}}$.

Comparison with SIF [23]. We present a detailed comparison between our protocol and the state-of-the-art SIF in Table 1. Assume two protocols all operate between $\mathcal{P}$ and n verifiers. The data in the table represents the amortized computation per multiplication gate. *Prover Comparison.* In SIF, for $t \in [n]$, $\mathcal{P}$ require four multiplication over $\mathbb{F}_{p^r}$ to compute $A^{(t)}_0$ and batch check, two scalar products over $\mathbb{F}_{p^r}$ to compute $A^{(t)}_1$. By contrast, our protocol completely eliminates expensive extension-field multiplications. for each $\mathcal{V}_t$. The only additional operation in ours is one hash operation for batch checking, which is very small. *Verifier Comparison.* $\mathcal{V}_t$ in SIF performs one scalar product over $\mathbb{F}_{p^r}$ to compute $k^{(t)}_{w_{\gamma_j}}$, four multiplication over $\mathbb{F}_{p^r}$ to compute $B^{(t)}_j$ and batch check. Though our work introduces a slight increase in addition and scalar operations, these computational overheads remain marginal. Our primary contribution lies in the elimination of expensive multiplications in the extension field, leading to significantly superior overall efficiency.

5 Implementation and Evaluation

We implement our MVZK protocol in C++ using the EMP toolkit. All experiments were conducted on Ubuntu 22.04 LTS with 16 vCPUs and 64 GB RAM. Each experiment was conducted ten times, Average results are reported. For arithmetic circuits, we use a 61-bit field ($p = 2^{61} - 1, r = 1$), for Boolean circuits, we use a binary field ($p = 2, r = 128$). For large-scale circuits, we instantiate undergoing VOLE with recent PCG-style protocols. For widely used benchmark circuits (e.g. the AES-128 circuit), which are typically small or median size Boolean

circuits, we instantiate the VOLE with the IKNP-style COT protocol. Code available at: https://github.com/yuanyuanduan/MVZKP-Implementation.

Comparison with Related Works. Our work mainly improve the online overhead based on the work by Zhou et al. [23], which we denote by SIF protocol. We denote our protocol by MVZK.

Performance on AES-128. We evaluated both protocols on an AES-128 circuit with nP $\in \{3, 8, 12, 16\}$ parties. As shown in Fig. 7b, our protocol significantly outperforms SIF in the online phase, achieving an impressive speedup of 5.2× to 17.4×. This efficiency gain is attained at the cost of a slightly higher preprocessing time, which is attributed to the necessary quadratic authentication. Nevertheless, the trade-off remains highly favorable given the critical importance of online latency.

Evaluation on Multiplication Gates. We benchmarked our multiplication gate optimizations using Arithmetic (*and_arith*) and Boolean (*and_bool*) circuits, each with 10,000 multiplication gates. Our construction shows its most significant advantage in the Boolean setting, where the Prover and Verifier are 9× and 47× faster than SIF, respectively. In arithmetic circuits, we achieve a consistent speedup of 1.2× for the Prover and 2.3× for the Verifier.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (Grant No. 62272269) and Scientific Research Innovation Capability Support Project for Young Faculty (Grant No. ZYGXQNJSKYCXNLZCXMI21).

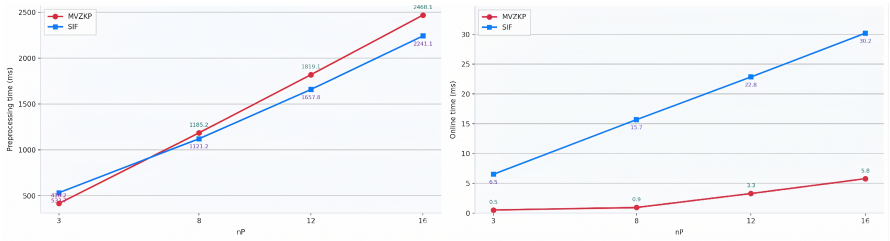
A Protocol that UC-Realizes $\Pi^{p,r}_{\mathsf{p\text{-}qsVOLE}}$

The protocol $\Pi^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$ is shown in Fig. 9.

Table 1: Online Cost per Multi-gate over Any Field.

	$\mathbb{F}_p$		$\mathbb{F}_{p^r}$			
$\mathcal{P}$	×	+	×	+	Scal†	H‡
Zhou et al. [23]	1	1	$4n$§	$4n$	$2n$	0
This work	1	1	**0**	$2n$	$2n$	< 1
$\mathcal{V}_t$	×	+	×	+	Scal	H
Zhou et al. [23]	0	0	4	3	1	0
This work	1	0	**0**	5	3	< 1

† × between an element in $\mathbb{F}_{p^r}$ and an element in $\mathbb{F}_p$.
‡ The hash operation.
§ The number of verifiers.



(a) Preprocess time **(b)** Online time

Fig. 7: Performance Comparison between Ours and SIF Evaluated on AES-128

Functionality $\Pi^{p,r}_{\mathsf{p\text{-}qsVOLE}}$

The Protocol interacts with $\mathcal{P}$ and $\mathcal{V}$. Let H be a random oracle $\{0,1\}^* \to \mathbb{F}_{p^r}$.

Initialize: Upon receiving $(\textsc{Init}, \mathsf{sid})$ from $\mathcal{P}$ and $\mathcal{V}$:

- $\mathcal{P}$ and $\mathcal{V}$ send $(\textsc{Init}, \mathsf{sid})$ to $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$, which returns $\Delta \in \mathbb{F}_{p^r}$ to $\mathcal{V}$.
- $\mathcal{P}$ and $\mathcal{V}$ locally initialize an empty set S.

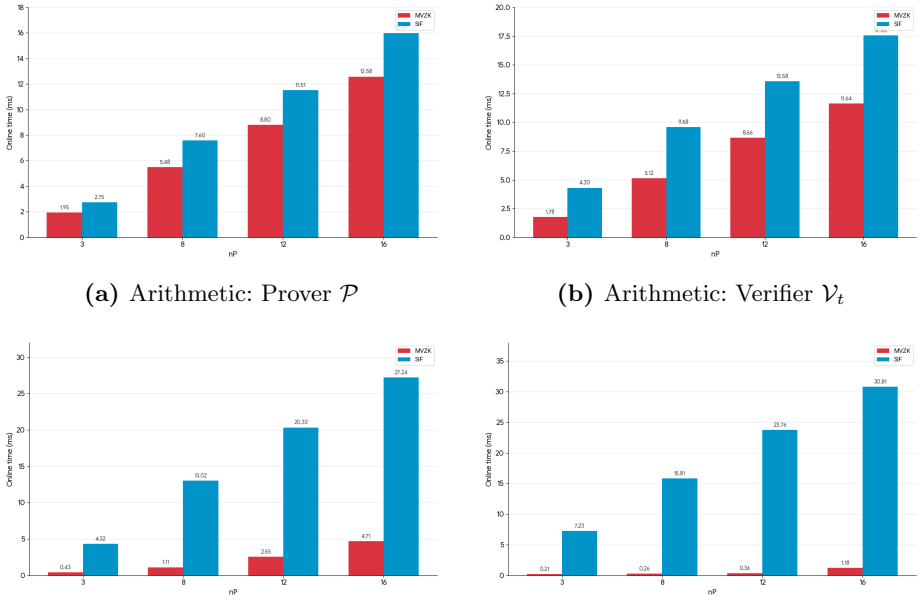
Authenticate over Subfield: Upon receiving $(\textsc{Authsub}, \mathsf{sid}, \ell, \mathsf{sd}, \{\tau_i\}_{i\in[\ell]})$ from $\mathcal{P}$ and $(\textsc{Authsub}, \mathsf{sid}, \ell, \{\tau_i\}_{i\in[\ell]})$ from $\mathcal{V}$, do:

- If $(\tau_i, \cdot)$ exists in S for any $i \in [\ell]$, the parties abort.
- $\mathcal{P}$ sends $(\textsc{Authsub}, \mathsf{sid}, \ell, \mathsf{sd})$ and $\mathcal{V}$ sends $(\textsc{Authsub}, \mathsf{sid}, \ell)$ to $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$, which returns $\{[u_{\tau_i}]\}_{i\in[\ell]}$ to the parties.
- $\mathcal{P}$ and $\mathcal{V}$ store $(\tau_i, [u_{\tau_i}])$ in S for every $i \in [\ell]$.
- $\mathcal{P}$ samples $s \leftarrow \Omega$, where Ω is the seed space of the expansion function Expand. Then $\mathcal{P}$ sends $(\textsc{Authsub}, \mathsf{sid}, r, s)$ to $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$ while $\mathcal{V}$ send $(\textsc{Authsub}, \mathsf{sid}, r)$ to the same instance. Finally, $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$ returns $\{u_i, m_i\}_{i\in[r]}$ to $\mathcal{P}$, and returns $\{k_i\}_{i\in[r]}$ to $\mathcal{V}$ such that $k_i = m_i + u_i \cdot \Delta$ for each $i \in [r]$.
- $\mathcal{P}$ computes $M_0 := \sum_{i=1}^{r} u_i \cdot X^{i-1} \in \mathbb{F}_{p^r}$, $M_1 := \sum_{i=1}^{r} m_i \cdot X^{i-1} \in \mathbb{F}_{p^r}$ and $\mathcal{V}$ computes $K := \sum_{i=1}^{r} k_i \cdot X^{i-1} \in \mathbb{F}_{p^r}$. Note that, $K = M_1 + M_0 \cdot \Delta$ holds.

Quadratic: On input $(\textsc{Quad}, \mathsf{sid}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ from $\mathcal{P}$ and $\mathcal{V}$:

1. If $(i, \cdot)$ exists for $i \in \mathcal{I}_j$, $\mathcal{P}$ and $\mathcal{V}$ retrieve $\{[u_i]\}_{i\in\mathcal{I}_j}$ from S; otherwise, abort.
2. $\mathcal{P}$ computes $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0} \in \mathbb{F}_p$ for every $j \in \mathcal{G}$.
3. $\mathcal{P}$ and $\mathcal{V}$ send $(\textsc{Authsub}, \mathsf{sid}, |\mathcal{G}|)$ to $\mathcal{F}^{p,r}_{\mathsf{psVOLE}}$, which returns $\{[v_j]\}_{j\in\mathcal{G}}$.
4. $\mathcal{P}$ sends $d_j = y_j - v_j$ to $\mathcal{V}$, then $\mathcal{P}$ and $\mathcal{V}$ compute $[y_j] = [v_j] + d_j$.
5. For each $j \in \mathcal{G}$, $\mathcal{P}$ and $\mathcal{V}$ execute the following:
   - $\mathcal{P}$ computes $a_{0,j} = g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) \in \mathbb{F}_{p^r}$, and $a_{1,j} = f_j(\{m_{u_i} + u_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - f_j(\{u_i\}_{i\in\mathcal{I}_j}) - m_{y_j} \in \mathbb{F}_{p^r}$.
   - $\mathcal{V}$ computes $b_j = (g_{j,1}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) - k_{y_j}) \cdot \Delta + g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) \in \mathbb{F}_{p^r}$.
6. $\mathcal{P}$ and $\mathcal{V}$ cooperate to ensure $b_{j'} = a_{0,j'} + a_{1,j'} \cdot \Delta$ for every j'-th gate in $\mathcal{G}$.
   - $\mathcal{V}$ samples $\chi \leftarrow \mathbb{F}_{p^r}$ and sends it to $\mathcal{P}$.
   - $\mathcal{P}$ computes $u_0 = \sum_{j'\in[|\mathcal{G}|]} a_{0,j'} \cdot \chi^{j'} + M_0$ and $u_1 = \sum_{j'\in[|\mathcal{G}|]} a_{1,j'} \cdot \chi^{j'} + M_1$, and sends (u_0, u_1) to $\mathcal{V}$.
   - $\mathcal{V}$ computes $w = \sum_{j'\in[|\mathcal{G}|]} b_{j'} \cdot \chi^{j'} + K$, If $w \neq u_0 + u_1 \cdot \Delta$, $\mathcal{V}$ aborts.

Fig. 9: The Programmable qsVOLE Protocol that UC-Realizes $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}$



(a) Arithmetic: Prover $\mathcal{P}$ **(b)** Arithmetic: Verifier $\mathcal{V}_t$

(c) Boolean: Prover $\mathcal{P}$ **(d)** Boolean: Verifier $\mathcal{V}_t$

Fig. 8: Online Time Comparison between MVZK (Ours) and SIF [23]

B Proof for Theorem 1

Proof. We prove that $\Pi_{\mathsf{MV\text{-}qsVOLE}}$ UC-realizes $\mathcal{F}_{\mathsf{MV\text{-}qsVOLE}}$ in the $(\mathcal{F}_{\mathsf{p\text{-}qsVOLE}}, \mathcal{F}_{\mathsf{COIN}})$-hybrid model. We construct a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$ and environment $\mathcal{Z}$, the ideal-world is computationally indistinguishable from the real-world execution.

Corrupted Verifiers. Up to $n-1$ verifiers are corrupted and the malicious verifiers attempt to learn the information about the prover's secret input. We describe the simulation strategy of $\mathcal{S}$ as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}, \mathcal{F}^{p,r}_{\mathsf{COIN}}$ for the adversary $\mathcal{A}$. Upon receiving $\Delta^{(t)}, \boldsymbol{k}^{(t)}$ sent by $\mathcal{V}^*_t \notin \mathcal{H}$. $\mathcal{S}$ sends $\Delta^{(t)}$ and $\boldsymbol{k}^{(t)}$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$.
2. $\mathcal{S}$ samples $\tilde{\boldsymbol{u}} \leftarrow \mathbb{F}_p^{\ell'}$. Then for $\mathcal{V}^*_t \notin \mathcal{H}$, $\mathcal{S}$ computes $\tilde{\boldsymbol{m}}^{(t)} := \boldsymbol{k}^{(t)} - \tilde{\boldsymbol{u}} \cdot \Delta^{(t)} \in \mathbb{F}_{p^r}^{\ell'}$. Note that, $\mathcal{S}$ can compute $\tilde{\boldsymbol{m}}^{(t)}$ since $\mathcal{S}$ knows $\Delta^{(t)}$ and $\boldsymbol{k}^{(t)}$.
3. $\mathcal{S}$ completes the protocol execution honestly in Fig. 5.
4. Whenever $\mathcal{A}$ attempts to make an honest party $\mathrm{H} \in \mathcal{H}$ abort, $\mathcal{S}$ simply halts simulating this party and returns (ABORT, sid, H) to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$.

Here we show that any adversary $\mathcal{A}$ cannot learn any information about $\boldsymbol{u}$ in the real-world execution. When $\mathcal{S}$ uses a randomly selected $\tilde{\boldsymbol{u}}$. We define x (resp.$\tilde{x}$) the value that broadcast by $\mathcal{P}$(resp.$\mathcal{S}$) using $\boldsymbol{u}$ (resp.$\tilde{\boldsymbol{u}}$). Take x as an example, $x = \boldsymbol{s}^\top \cdot \boldsymbol{u}_a + \boldsymbol{e}^\top \cdot \boldsymbol{u}_b$. Due to the security of consistency check in Section 3.3, x is computationally indistinguishable from a uniform element in $\mathbb{F}_{p^r}$ and so is $\tilde{x}$.

The real-world is computationally indistinguishable from ideal-world.

Corrupted Prover. In this case, the malicious prover $\mathcal{P}^*$ may use an inconsistent seed sd, which would result in an inconsistent $\boldsymbol{u}$. We prove that if $\mathcal{P}^*$ cheats, it will be caught with overwhelming probability. The below simulation strategy is straightforward: $\mathcal{S}$ acts as honest verifiers and follows the protocol faithfully.

1. $\mathcal{S}$ emulates $\mathcal{F}^{p,r}_{\mathsf{p\text{-}qsVOLE}}, \mathcal{F}^{p,r}_{\mathsf{COIN}}$ honestly for the adversary $\mathcal{A}$. In doing so, $\mathcal{S}$ receives sd from $\mathcal{P}^*$, and $\mathcal{S}$ knows whether $\mathcal{P}^*$ is using inconsistent seeds.
2. $\mathcal{S}$ completes the protocol execution honestly on behalf of the honest verifiers.
3. If $\mathcal{P}^*$ uses inconsistent seeds yet passes the consistency check, $\mathcal{S}$ aborts.
4. If $\mathcal{P}^*$ uses consistent sd and passes the consistency check (i.e., $\mathcal{P}^*$ sends the correct $\boldsymbol{m}^{(t)}$ to each honest verifier $\mathcal{V}_t$), $\mathcal{S}$ sends sd and $\{\boldsymbol{m}^{(t)}\}$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$.
5. Whenever $\mathcal{A}$ attempts to make an honest party $\mathrm{H} \in \mathcal{H}$ abort, $\mathcal{S}$ halts simulating this party and returns $(\textsc{Abort}, \mathrm{sid}, \mathrm{H})$ to $\mathcal{F}^{p,r}_{\mathsf{MV\text{-}qsVOLE}}$.

It is easy to see that $\mathcal{A}$ would distinguish real-world from ideal-world if $\mathcal{S}$ aborts. We show that the probability of a cheating $\mathcal{P}^*$ passing the consistency check is negligible. Suppose $\mathcal{P}^*$ uses inconsistent seeds, $\mathrm{sd}_1, \mathrm{sd}_2$ such that $\boldsymbol{u}_1 \neq \boldsymbol{u}_2$, where $\boldsymbol{u}_1 := \mathsf{Expand}(\mathrm{sd}_1, \ell')$ and $\boldsymbol{u}_2 := \mathsf{Expand}(\mathrm{sd}_2, \ell')$. By Theorem 1, we know that $\Pr[\boldsymbol{s}^\top \cdot \boldsymbol{u}_{a_1} + \boldsymbol{e}^\top \cdot \boldsymbol{u}_{b_1} = \boldsymbol{s}^\top \cdot \boldsymbol{u}_{a_2} + \boldsymbol{e}^\top \cdot \boldsymbol{u}_{b_2}] = p^{-r}$, which is negligible. Therefore, unless $\mathcal{P}^*$ is able to forge a MAC tag which happens with probability p^{-r}, the cheating $\mathcal{P}^*$ can pass the consistency check. In conclusion, the ideal-world is computationally indistinguishable from real-world execution.

References

1. Abe, M., Cramer, R., Fehr, S.: Non-interactive distributed-verifier proofs and proving relations among commitments. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 206–224. Springer (2002)
2. Applebaum, B., Kachlon, E., Patra, A.: Verifiable relation sharing and multi-verifier zero-knowledge in two rounds: trading nizks with honest majority. In: Annual International Cryptology Conference. pp. 33–56. Springer (2022)
3. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Crowd verifiable zero-knowledge and end-to-end verifiable multiparty computation. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 717–748. Springer (2020)
4. Baum, C., Jadoul, R., Orsini, E., Scholl, P., Smart, N.P.: Feta: Efficient threshold designated-verifier zero-knowledge proofs. In: Proceedings of the 2022 acm sigsac conference on computer and communications security. pp. 293–306 (2022)
5. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Annual International Cryptology Conference. pp. 92–122. Springer (2021)
6. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: Annual international cryptology conference. pp. 67–97. Springer (2019)
7. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 244–276. Springer (2020)

8. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
9. Burmester, M., Desmedt, Y.: Broadcast interactive proofs. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 81–95. Springer (1991)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
11. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving line-point zero knowledge: Two multiplications for the price of one. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 829–841 (2022)
12. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. Cryptology ePrint Archive (2020)
13. Goldwasser, S., Lindell, Y.: Secure multi-party computation without agreement. Journal of Cryptology **18**(3), 247–287 (2005)
14. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Symposium on the Theory of Computing (1985), https://api.semanticscholar.org/CorpusID:209402113
15. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. Journal of cryptology **27**(3), 506–543 (2014)
16. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 21–30 (2007)
17. Lepinski, M., Micali, S., Shelat, A.: Fair-zero knowledge. In: Theory of Cryptography Conference. pp. 245–263. Springer (2005)
18. Liu, M., Heng, Y., Lu, X., Au, M.H.: Jesseq: Efficient zero-knowledge proofs for circuits over any field. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 3402–3420. IEEE (2025)
19. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid mpc for dishonest majority. In: Annual International Cryptology Conference. pp. 719–749. Springer (2022)
20. Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2986–3001 (2021)
21. Yang, K., Wang, X.: Non-interactive zero-knowledge proofs to multiple verifiers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 517–546. Springer (2022)
22. Zhou, Z., Zhang, B., Zhou, H.S., Ren, K.: Practical constructions for single input functionality against a dishonest majority. In: 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P). pp. 398–414. IEEE (2024)
23. Zhou, Z., Zhang, B., Zhou, H.S., Ren, K.: Single-input functionality against a dishonest majority: Practical and round-optimal. In: IACR International Conference on Public-Key Cryptography. pp. 3–36. Springer (2025)