JesseQ: Efficient Zero-Knowledge Proofs for Circuits over Any Field

Mengling Liu, Yang Heng, Xingye Lu, Man Ho Au
The Hong Kong Polytechnic University, Hong Kong

***Abstract*—Recent advances in Vector Oblivious Linear Evaluation (VOLE) protocols have enabled constant-round, fast, and scalable (designated-verifier) zero-knowledge proofs, significantly reducing prover computational cost. Existing protocols, such as QuickSilver [CCS'21] and LPZKv2 [CCS'22], achieve efficiency with prover costs of 4 multiplications in the extension field per AND gate for Boolean circuits, with one multiplication requiring a $O(\kappa \log \kappa)$-bit operation where $\kappa = 128$ is the security parameter, and 3-4 field multiplications per multiplication gate for arithmetic circuits over a large field.**

We introduce JesseQ, a suite of two VOLE-based protocols: JQv1 and JQv2, which advance state of the art. JQv1 requires only 2 scalar multiplications in an extension field per AND gate for Boolean circuits, with one scalar needing a $O(\kappa)$-bit operation, and 2 field multiplications per multiplication gate for arithmetic circuits over a large field. In terms of communication costs, JQv1 needs just 1 field element per gate. JQv2 further reduces communication costs by half at the cost of doubling the prover's computation.

Experiments show that, compared to the current state of the art, both JQv1 and JQv2 achieve at least $3.9\times$ improvement in the online phase for Boolean circuits. For large field circuits, JQv1 has a similar performance, while JQv2 offers a $1.3\times$ improvement. Additionally, both JQv1 and JQv2 maintain the same communication cost as the current state of the art. Notably, on the cheapest AWS instances, JQv1 can prove 9.2 trillion AND gates (or 5.8 trillion multiplication gates over a 61-bit field) for just one US dollar. JesseQ excels in applications like inner products, matrix multiplication, and lattice problems, delivering 40%-200% performance improvements compared to QuickSilver. Additionally, JesseQ integrates seamlessly with the sublinear Batchman framework [CCS'23], enabling further efficiency gains for batched disjunctive statements.

1. Introduction

Constant-round zero-knowledge proofs (ZKPs) based on vector oblivious linear evaluation (VOLE) [1], [2], [3], [4], [5] have recently gained notable attention for their efficiency and scalability. State-of-the-art implementations [6], [7] can prove tens of millions of gates per second, and scale smoothly to prove trillions of gates. Additionally, they significantly reduce the prover's computational overhead compared to all existing succinct zero-knowledge proof systems. Specifically, [7] demonstrates that VOLE-based ZKPs are at least $3 \sim 10\times$ faster than other existing constant-round ZKPs (e.g., Groth16 [8], Virgo [9], and Cerberus [10]) when proving one million gates.

Constant-round VOLE-based ZKPs The performance of existing constant-round VOLE-based ZKPs is shown in Table 1. Among the ZKPs in this paradigm, Line Point Zero Knowledge (LPZK) [13] reduced communication and, for the first time, reached the milestone of approximately 1 element per gate for large field arithmetic circuits [1]. It was later improved by QuickSilver [6] to support any field. Meanwhile, in another direction of improvement, LPZKv2 [7] halves LPZK's communication and also reduces its computational cost. In these constructions, the communication cost is linear to the circuit size. AntMan [14] gave the first construction with sublinear communication at $\mathcal{O}(|C|^{3/4})$, where $|C|$ is the circuit size for an arithmetic circuit over a large field. While the computational cost remains fairly efficient in practice, it incurs an overhead of $\mathcal{O}(\log |C|)$ in computation, i.e., the overall cost is quasilinear in $|C|$.

In terms of computational cost, QuickSilver and LPZKv2 delivered the best performance for the Boolean circuit and arithmetic circuit over large fields, respectively. It is also worth noting that there is a large difference in performance between VOLE-based ZKPs for circuits over any field versus their counterparts that work over large fields only.

Along another direction, several works [12], [15] improves constant-round VOLE-based ZKPs for circuits with specific structure, i.e., circuit representing disjunctive statements of the form $\mathcal{C}_1(w) = 1 \vee \mathcal{C}_2(w) = 1 \cdots \vee \mathcal{C}_B(w) = 1$ for B different subcircuits (known as branches). The state-of-the-art Batchman [15] achieves sublinear communication and computation for batched disjunctions. It can be viewed as a framework that can be instantiated using constant-round VOLE-based ZKPs as a blackbox to handle multiplication.

Motivation We consider it is worthwhile to investigate the design of constant-round VOLE-based ZKPs for any field with better communication and computation performance. Therefore, this work aim to address the following question:

> *Can we develop a constant-round, practical and scalable proof system for circuits supporting any field that outperforms existing constant-round VOLE-based ZKPs? In addition, can it achieve better concrete performance when it is used together with the recent sublinear framework for batched disjunction statements?*

1. The random oracle model version. The information-theoretic construction requires approximately 2 elements per gate.

TABLE 1. COMPARISON TABLE FOR CONSTANT-ROUND (S)VOLE-BASED ZKPS IN THE ONLINE PHASE

	Boolean Circuit		Arithmetic Circuit	
	Size	Speed	Size	Speed
Wolverine [11]	7	1.25 M/sec	4	0.96 M/sec
Mac'n'Cheese [12]	–	–	3	3.6 M/sec
IT-LPZK [13]	–	–	$2+\frac{1}{t}$	19.6 M/sec
QuickSilver [6]	1	8.6 M/sec	1	7.8 M/sec
IT-LPZKv2 [7]	–	–	$1+\frac{1}{t}$	21.8 M/sec
ROM-LPZKv2 [7]	–	–	$\frac{1}{2}$	9.8 M/sec
AntMan [14]	–	–	sublinear	7.01 M/sec
JQv1	1	64.1 M/sec	1	23.3 M/sec
JQv2	$\frac{1}{2}$	34.2 M/sec	$\frac{1}{2}$	13.7 M/sec

This table compares the prover cost of our works (JQv1 and JQv2) with prior related work using data reported in their studies and running experiments on the same hardware, except for AntMan, which uses a larger instance.
Size represents the number of field elements to send for each multiplication gate, except for AntMan, which has sublinear communication. Speed represents the number of multiplication gates that can be executed per second with unlimited bandwidth and a single thread, except for AntMan, which uses a 1 Gbps network bandwidth and runs on four threads. Additionally, unlike other works that set the prime $p = 2^{61} - 1$ for the arithmetic circuit, AntMan requires p to be set as a smaller prime $2^{59} - 2^{28} + 1$ due to the properties of the homomorphic encryption it uses.

1.1. Contributions

We introduce JesseQ (JQv1 and JQv2), a new design of constant-round VOLE-based ZKPs for the circuit over any field in the preprocessing model, as in LPZKv2. They are applicable to arbitrary and layered circuits, respectively, with better online performance[2]. The comparisons are shown in Table 1. As in previous work, we focus on comparing prover speeds in our analysis. For any field circuit, both the prover and verifier in JQv1 and JQv2 require fewer operations than the state-of-the-art. The only exception is that the verifier in JQv1 requires one more multiplication per multiplication gate than LPZKv2 for large field circuits. Nonetheless, our protocols retain their advantage by supporting any field.

JQv1 For Circuit over Any Field. We introduce JQv1, an efficient constant-round VOLE-based ZKP for arbitrary arithmetic circuits over any field, with a communication cost of one field element per multiplication gate. For large field circuits, the prover in JQv1 requires only two field multiplications per multiplication gate, compared to three in the state-of-the-art IT-LPZKv2, a version of the LPZKv2 protocol in [7] designed for arbitrary large field circuits. For Boolean circuits, JQv1 requires two scalar multiplications (e.g., $a \in \mathbb{F}_2, b \in \mathbb{F}_{2^\kappa}, a \cdot b$) in a κ-bit extension field per AND gate, where $\kappa = 128$ is the security parameter, while QuickSilver requires four multiplications in the extension field (e.g., $a \in \mathbb{F}_{2^\kappa}, b \in \mathbb{F}_{2^\kappa}, a \cdot b$). The details of theoretical comparison are given in Section 3.3.

JQv2 for Layered Circuit over Any Field. We introduce JQv2, an efficient constant-round VOLE-based ZKP supporting layered circuits[3] over any field with a communication cost of $\frac{1}{2}$ field element per multiplication gate. Compared to ROM-LPZKv2, a version of the LPZKv2 protocol in [7] designed for layered circuits over large fields with the same communication cost, JQv2 reduces prover computation from an average of 8.5 to 4 field multiplications per multiplication gate. For Boolean circuits, even though the prover in JQv2 requires two additional multiplications in the extension field compared to JQv1, it still uses two fewer than QuickSilver per multiplication gate. Our approach can be applied to general circuits with varying communication savings. Notably, for random circuits discussed in Section 1.2, we achieve about a 38% communication reduction compared to JQv1. The details of theoretical comparison are given in Section 4.1.

Efficient Implementations for JQv1 and JQv2. We implement JQv1 and JQv2 based on the publicly available implementation of QuickSilver [16]. The experiment results show that when running on the cheapest AWS instances, our protocol requires only one US **cent** to verify **92 billion** AND gates (or **58 billion** multiplication gates over a 61-bit field). Furthermore, compared to QuickSilver, in the online phase, JQv1 and JQv2 yield approximately a $7\times$ and $3.9\times$ improvement in computation for Boolean circuits, respectively, and at least $2.9\times$ and $1.7\times$ for arithmetic circuits, respectively, even though the total running time (preprocessing + online) of JQv1 increases by 21% for Boolean circuits and 40% for arithmetic circuits. Although JQv1 has the similar performance as IT-LPZKv2, JQv2 achieves $1.3\times$ improvement compared to ROM-LPZKv2. The detailed performance of our protocols and the comparisons are provided in Section 6.1.

Practical Applications. Our protocols can be effectively applied to practical applications, including proving inner product, matrix multiplication, and proving knowledge of solutions to lattice problems (e.g., short integer solution (SIS) problems). We have also implemented those applications. Experimental results show that, compared to those applications in QuickSilver, our protocol has the same communication complexity and achieves around $2\times$ improvement in computation for proving inner products, a 40%

2. Following LPZKv2, we report the cost of the online phase when generating the proof after the witness is known, assuming all preprocessing is complete. Section 6.1 provides a breakdown of the preprocessing time.

3. Each gate is assigned to a layer k, with the inputs coming from the outputs of gates in layer $k-1$.

improvement for proving matrix multiplication, and at least a $2\times$ improvement for proving knowledge of solutions to SIS problems. We provide the details in Section 5 and 6.2.

Sublinear Framework Extensions. We apply the sublinear framework Batchman for batched disjunctive statements to JQv1 and JQv2. Experiments show that Batchman based on our designs achieves the improvement at least from $1.1\times$ to $2.2\times$, compared to Batchman based on QuickSilver. The comparisons are shown in Section 6.3.

1.2. Constant-round VOLE-Based ZKPs

We first review the relevant background of prior works before presenting a technical overview of our protocols.

1.2.1. Information-Theoretic Message Authentication Codes and VOLE. Most VOLE-based ZKPs utilize the VOLE correlation as an information-theoretic message authentication code (IT-MAC) [17], [18], [19], [20], enabling $\mathcal{P}$ to commit wire values to $\mathcal{V}$. Let $\mathbb{F}_p$ be a finite field, $x \in \mathbb{F}_p$ be a global key known to $\mathcal{V}$, and $u \in \mathbb{F}_p$ known to $\mathcal{P}$. An IT-MAC commitment to u consists of a pair of values, m and k, known by $\mathcal{P}$ and $\mathcal{V}$ respectively, such that $m = k - u \cdot x$. Additionally, to open the commitment, $\mathcal{P}$ sends (m, u) to $\mathcal{V}$, who checks if $m = k - u \cdot x$. We denote the commitment to u under the global key x as $[u]$. We also refer $[u]$ as an authenticated value, with m being the MAC tag of u. Note that this MAC is information-theoretically hiding and binding.

This MAC is also homomorphic, allowing $\mathcal{P}$ and $\mathcal{V}$ to compute $[a+b] = [a]+[b]$. Additionally, we define the MAC for a public constant c as $(m := 0, c)$, $(k := c \cdot x, x)$, which allows for any affine operation from the committed values.

The VOLE functionality allows $\mathcal{P}$ and $\mathcal{V}$ to construct a vector of authenticated random values. After invocation, $\mathcal{P}$ receives two vectors of random field elements $(\boldsymbol{m}, \boldsymbol{u})$, and $\mathcal{V}$ obtains a global key x and a vector $\boldsymbol{k}$ such that $\boldsymbol{k} = \boldsymbol{m} + \boldsymbol{u} \cdot x$. This process effectively authenticates the random vector $[\boldsymbol{u}]$.

A commitment to a random vector $[\boldsymbol{u}]$ can be easily transformed into a commitment to a vector $[\boldsymbol{w}]$ when $[\boldsymbol{w}]$ is known to $\mathcal{P}$. To achieve this, $\mathcal{P}$ sends $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{u}$ to $\mathcal{V}$. $\mathcal{V}$ then updates $\boldsymbol{k}$ by setting it to $\boldsymbol{k} + \boldsymbol{d} \cdot x$.

1.2.2. The Framework of VOLE-based ZKPs. Most constant-round VOLE-based ZKPs follow the "commit-and-prove" framework and include a preprocessing phase to enhance efficiency. Specifically, they consist of the following phases.

1) (Preprocessing) $\mathcal{P}$ and $\mathcal{V}$ invoke the VOLE functionality to obtain commitments to a vector of random values $[\boldsymbol{u}]$, which will be consumed during the online phase.
2) (Online Phase, Commit) $\mathcal{P}$ commits all wire values in the circuit to $\mathcal{V}$. Specifically, for the circuit's input wires and the output wires of multiplication gates, $\mathcal{P}$ sends $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{u}$ to $\mathcal{V}$ to transform the commitments of random values generated in step 1 to the commitments of wire values. Due to the additive homomorphism of the IT-MAC, parties can locally compute the commitment for the output wire of the addition gate when given the commitments of its input wires. The total number of field elements sent is equal to the sum of the number of multiplication gates and the number of input wires.
3) (Online Phase, Prove [Correct Multiplication]) $\mathcal{P}$ and $\mathcal{V}$ cooperate to check $[w]$ has been correctly computed for each output wire of multiplication gates.
4) (Online Phase,Prove [Correct Output]) $\mathcal{P}$ sends the MAC tag m of the circuit's output wire to $\mathcal{V}$. $\mathcal{V}$ then verifies that $k = m + x$ when the output wire w is expected to be 1.

The main difference among existing constant-round VOLE-based ZKPs lies in step 3.

1.2.3. Correct Multiplication: LPZK and QuickSilver. LPZK [13], and subsequently QuickSilver [6], use a similar idea to check correct multiplication. Specifically, we will review how $\mathcal{P}$ proves to $\mathcal{V}$ that a multiplication gate is evaluated correctly after its wire values have been committed. Consider a multiplication gate with input wires w_α, w_ρ, and an output wire w_υ. Given the committed wire values $[w_\alpha], [w_\rho], [w_\upsilon]$, $\mathcal{P}$ needs to convince $\mathcal{V}$ that $w_\alpha \cdot w_\rho = w_\upsilon$.

The idea is to interpret the IT-MACs as linear polynomials in X. Specifically, let $p_i(X) = m_{w_i} + w_i \cdot X$ for $i \in \{\alpha, \rho, \upsilon\}$. Then consider the degree-2 polynomial $f(X) = p_\alpha(X) \cdot p_\rho(X) - X \cdot p_\upsilon(X)$. $\mathcal{V}$ knows its evaluation at x, given by $f(x) = k_{w_\alpha} \cdot k_{w_\rho} - x \cdot k_{w_\upsilon}$. Moreover, $\mathcal{P}$ knows its coefficients. To understand this, note that $f(X) = a_0 + a_1 \cdot X + a_2 \cdot X^2$, where $a_0 = m_{w_\alpha} \cdot m_{w_\rho}$, $a_1 = w_\alpha \cdot m_{w_\rho} + w_\rho \cdot m_{w_\alpha} - m_{w_\upsilon}$, and $a_2 = w_\alpha \cdot w_\rho - w_\upsilon$ known to $\mathcal{P}$.

More importantly, $f(X)$ is a linear polynomial if and only if $w_\alpha \cdot w_\rho = w_\upsilon$, indicating that the multiplication is correctly evaluated. To prove this, $\mathcal{P}$ can send values a_0 and a_1 to $\mathcal{V}$, who then checks if $a_0 + a_1 \cdot x \stackrel{?}{=} k_{w_\alpha} \cdot k_{w_\rho} - x \cdot k_{w_\upsilon}$. However, this would require sending two field elements for each gate, so instead, a batch check is conducted.

Here, we review the batch check of QuickSilver. There are L linear polynomials $f_j(X)$, corresponding to L multiplication gates, for $j = 1$ to L. $\mathcal{V}$ knows the evaluations at x of these polynomials, while $\mathcal{P}$ knows their coefficients. To perform the batch check, $\mathcal{V}$ first sends a random challenge χ to $\mathcal{P}$, who computes $F(X) := \sum_{j=1}^{L} f_j(X) \cdot \chi^j = A + B \cdot X$. $\mathcal{P}$ then sends A and B to $\mathcal{V}$, who checks if $A + B \cdot x \stackrel{?}{=} \sum_{j=1}^{L} f_j(x) \cdot \chi^j$. To achieve zero-knowledge, additional VOLE correlations are required to mask A and B. We omit such detail here.

The above is adequate for arithmetic circuits over large fields. However, when the field size is small, such as $\mathbb{F}_p = \mathbb{F}_2$ (i.e., Boolean circuit), security does not hold. One vulnerability is that the global key x can be guessed with a non-negligible probability, allowing a cheating prover to forge the IT-MAC for any value. To address this, QuickSilver makes use of a variant of VOLE known as subfield VOLE (sVOLE). The sVOLE functionality generates correlated

randomness in the form $\boldsymbol{k} = \boldsymbol{m} + \boldsymbol{u} \cdot x$, where $\boldsymbol{u} \in \mathbb{F}_p^n$ and $\boldsymbol{k}, \boldsymbol{m} \in \mathbb{F}_{p^r}^n$, and $x \in \mathbb{F}_{p^r}$.

1.2.4. Correct Multiplication: LPZKv2. LPZKv2 improves LPZK through two technical ideas. First, the IT-MACs are redefined by placing the witness value in m instead of u. In this setup, when m is authenticated, $\mathcal{P}$ holds u, and $\mathcal{V}$ holds k and x, satisfying $k = m + u \cdot x$. This change allows $\mathcal{V}$ to save one multiplication per gate, as it can update k with $k + d$ after receiving $d := w - m$ from $\mathcal{P}$, rather than using $k + d \cdot x$.

The second one is using quadratically certified VOLE (qVOLE), which generates additional correlated randomness, to further enhance the efficiency of the online phase. For example, in addition to $k_i = m_i + u_i \cdot x$ for $i \in \{\alpha, \rho, \upsilon\}$, the qVOLE functionality also generates $k_y = m_y + u_y \cdot x$ where $u_y = u_\alpha \cdot u_\beta$. We will briefly review how qVOLE contributes to reducing complexity.

Assume $\mathcal{P}$ needs to prove that $w_\alpha \cdot w_\rho = w_\upsilon$. In LPZKv2, the committed value is placed in the constant term, meaning $k_i = w_i + u_i \cdot x$ for $i \in \{\alpha, \rho, \upsilon\}$. Additionally, during the preprocessing phase, $\mathcal{P}$ and $\mathcal{V}$ invoke qVOLE to obtain (m_y, u_y) and (k_y, x) respectively, such that $k_y = m_y + u_y \cdot x$ where $u_y = u_\alpha \cdot u_\rho$.

Consider the degree-2 polynomial $f(X) = p_\alpha(X) \cdot p_\rho(X) - p_\upsilon(X) - X \cdot p_y(X)$. $\mathcal{V}$ knows its evaluation at x, given by $f(x) = k_{w_\alpha} \cdot k_{w_\rho} - k_{w_\upsilon} - x \cdot k_y$. Moreover, $\mathcal{P}$ knows its coefficients. To understand this, note that $f(X) = a_0 + a_1 \cdot X + a_2 \cdot X^2$, where $a_0 = w_\alpha \cdot w_\rho - w_\upsilon$, $a_1 = w_\alpha \cdot u_{w_\rho} + w_\rho \cdot u_{w_\alpha} - u_\upsilon - m_y$, and $a_2 = u_\alpha \cdot u_\rho - u_y$ known to $\mathcal{P}$. Since $u_y = u_\alpha \cdot u_\rho$, $a_2 = 0$. Furthermore, $a_0 = 0$ if and only if $w_\alpha \cdot w_\rho = w_\upsilon$. Then, $\mathcal{P}$ can send a_1 to $\mathcal{V}$ who checks if $a_1 \cdot x = k_{w_\alpha} \cdot k_{w_\rho} - k_{w_\upsilon} - x \cdot k_y$.

The batch check of LPZKv2 operates as follows. Assume $\mathcal{P}$ needs to prove that $f_j(X)/X = a_{j,1}$ for $j = 1$ to L are all constant polynomials such that $\mathcal{V}$ knows their evaluations at x. $\mathcal{P}$ computes and sends $A = \prod_{j=1}^{L} a_{j,1}$ to $\mathcal{V}$ who checks if $A = \prod_{j=1}^{L} f_j(x)/x$. Roughly speaking, the batch check relies on the fact that a set of polynomials are all constant if and only if their product is a constant polynomial.

Additionally, LPZKv2 offers additional optimizations for a broad class of circuits described in Section 1.3.2. The key observation is that the expression $g(x) = p_\alpha(x) \cdot p_\rho(x) - x \cdot p_y(x) = a_0 + a_1 \cdot x$ where $a_0 = w_\alpha \cdot w_\rho$, $a_1 = w_\alpha \cdot u_\rho + w_\rho \cdot u_\alpha - m_y$, already represents an authentication of $w_\alpha \cdot w_\rho$ without requiring any communication at all.

1.3. Technical Overview of Our Construction

In this section, we provide the intuition behind JesseQ. We utilize quadratic subfield VOLE (qsVOLE) to enjoy the advantage of both QuickSilver and LPZKv2. Furthermore, we design a more efficient multiplication check that is suitable to be used with qsVOLE. Unlike existing methods that rely on degree-2 polynomials, our multiplication check uses degree-1 polynomials, making it more efficient. We have also designed a more efficient batch check.

1.3.1. JQv1 For Circuit over Any Field. Consider a multiplication gate with input wires (w_α, w_ρ) and output wire w_υ. During preprocessing, two parties first obtain random authenticated values $[u_\alpha], [u_\rho], [u_\upsilon]$, alongside $[y = u_\alpha \cdot u_\rho]$ generated via quadratic sVOLE (qsVOLE). After $\mathcal{P}$ sending $d_i := w_i - u_i$ for $i \in \{\alpha, \rho, \upsilon\}$, two parties only computes $[w_\upsilon] = [u_\upsilon] + d_\upsilon$.

Given linear polynomials $p_i(X) = m_{u_i} + u_i \cdot X$ for $i \in \{\alpha, \rho\}$, $p_\upsilon(X) = m_{w_\upsilon} + w_\upsilon \cdot X$, and $p_y(X) = m_y + y \cdot X$, we define the degree-1 polynomial $f(X)$ as

$$\begin{aligned} & d_\rho \cdot p_\alpha(X) + d_\alpha \cdot p_\rho(X) + p_y(X) + d_\rho \cdot d_\alpha \cdot X - p_\upsilon(X) \\ = & \underbrace{\qquad\qquad}_{d_\rho \cdot m_{u_\alpha} + d_\alpha \cdot m_{u_\rho} + m_y - m_{w_\upsilon}} + \underbrace{\qquad}_{(u_\alpha + d_\alpha) \cdot (u_\rho + d_\rho) - w_\upsilon} \cdot X \end{aligned}$$

whose coefficients are held by $\mathcal{P}$, and evaluation at x is held by $\mathcal{V}$. If and only if the multiplication gate is evaluated correctly, we have $a_1 = (u_\alpha + d_\alpha) \cdot (u_\rho + d_\rho) - w_\upsilon = w_\alpha w_\rho - w_\upsilon = 0$, meaning that $f(X)$ is a constant polynomial. While $\mathcal{P}$ can send a_0 to $\mathcal{V}$ to check if $a_0 \stackrel{?}{=} f(x)$, this would necessitate transmitting one element per multipication gate.

We also design a way to batch check that a set of polynomials are constant polynomials. Unlike previous work, our batch checking utilizes a hash function H (modeled as a random oracle). Specifically, to prove that $f_j(X)$ for $j = 1$ to L are all constant polynomials ($= a_{0,j}$), $\mathcal{P}$ computes and sends $h = H(a_{0,1} || a_{0,1} || \cdots || a_{0,L})$ to $\mathcal{V}$, who checks if $h \stackrel{?}{=} H(f_1(x) || f_2(x) || \cdots || f_L(x))$.

Like LPZKv2 [7], our result can extend from arithmetic circuits with fan-in 2 addition and multiplication gates to circuits with arbitrary degree-2 polynomial gates. For more details, please refer to Section 3.

Remarks: We would like to highlight that our definition of degree-1 $f(X)$ leads to significant savings for Boolean circuit. Notably, the computation of $f(X)$ avoids the multiplication of extension field elements. In Boolean circuit, the scalar multipication (e.g.,$d_\alpha \cdot m_{u_\rho}; d_\alpha \in \mathbb{F}_2, m_{u_\rho} \in \mathbb{F}_{2^\kappa}$) is more efficient than the multiplication (e.g.,$m_{w_\rho} \cdot m_{w_\alpha}; m_{w_\rho}, m_{w_\alpha} \in \mathbb{F}_{2^\kappa}^2$) in the extension fied.

Our batch checking is also more efficient than existing approaches. The batch checking in QuickSilver and LPZKv2 requires three multiplications over $\mathbb{F}_{p^r}$ and approximately one multiplication over $\mathbb{F}_p$ when p is large, respectively, per multiplication gate. Utilizing hash function for batch checking is desirable as it yields better concrete performance. For instance, on an Amazon EC2 *m5.2xlarge* instance, processing 10 million 61-bit or 128-bit field elements using BLAKE3 [21] is at least twice as fast as performing 10 million multiplications in the same 61-bit or 128-bit field. A comprehensive performance comparison between multiplication and hashing can be found in Section 6.4.

1.3.2. JQv2 for Layered Circuit over Any Field. JQv2 reduces the amortized number of field elements sent by $\mathcal{P}$ per multipication gate to $\frac{1}{2}$ in the layered circuit. This is achieved through a similar observation to LPZKv2:

$$\begin{aligned} g(X) &= d_\rho \cdot p_\alpha(X) + d_\alpha \cdot p_\rho(X) + p_y(X) + d_\rho \cdot d_\alpha \cdot X \\ &= \underbrace{\qquad\qquad}_{d_\rho \cdot m_{u_\alpha} + d_\alpha \cdot m_{u_\rho} + m_y} + \underbrace{\qquad}_{(u_\alpha + d_\alpha) \cdot (u_\rho + d_\rho)} \cdot X \end{aligned}$$

already represents an authentication of $w_\alpha \cdot w_\rho$ without requiring any communication at all. For simplicity, we assume the even layer has more wires than the odd layers here. Specifically, we allow $\mathcal{P}$ and $\mathcal{V}$ to directly compute $[w_v]$ for the gate's output in the even layers using values associated with the input wires, eliminating the need for $\mathcal{P}$ send d_v to $\mathcal{V}$. For gates in odd layers, however, $\mathcal{P}$ still sends d_v to $\mathcal{V}$ to compute $[w_v]$. Our checking method in JQv1 can not be applied here since values associated with the inputs of odd layers (outputs of even layers) are not the sum of random u values and d values known to both parties. Therefore, given authenticated values of input and output wires, we perform the similar check to QuickSilver, which is based on the evaluation of degree-2 polynomial at x. We defer the details to Section 4.

Applied to Arbitrary Circuits. For a broad class of non-layered circuits, substantial savings are also possible. For example, as described in [7], in a random circuit $\mathcal{C}$ composed entirely of multiplication gates with inputs chosen randomly from previous outputs, one can achieve an approximately 38% reduction in communication.

Compared with Polynomial-based ZK. QuickSilver also offers a ZK protocol for polynomial sets, replacing the entire circuit with a single polynomial of degree d, requiring $\mathcal{I}_{\mathsf{in}} + dr$ communication. For layered circuits, introducing gates representing polynomials of degree up to 2^c can remove gates to those at layers $i \mod c$. This reduces communication to $\leq |\mathcal{I}_{\mathsf{in}}| + |\mathcal{C}|/c + 2^c r$, but requires at least $2^c r$ computation per gate. Our approach, similar to LPZKv2, optimizes online phase and aligns with the informal definition of the gate-by-gate paradigm.

1.4. Discussion

The main advantage of constant-round VOLE-based protocols is their significant reduction in prover computation compared to protocols like Virgo and zk-SNARKs, though they incur linear communication costs relative to circuit size. Thus, VOLE-based protocols generally outperform other ZK protocols in fast network environments or where computational costs are critical.

Cloud platforms like AWS charge for computational resources but not for intra-platform communication, rendering communication costs irrelevant unless they drastically impact application feasibility. Hence, the cost is primarily determined by computational demands. Additionally, with dynamic pricing—higher rates during peak times and discounts for off-peak or flexible usage of spare computing power—these platforms incentivize the adoption of online-offline models, promoting a shift of computations to the offline phase to capitalize on cost efficiencies.

Limitation. Similar to LPZKv2 [7], our protocol requires a more expensive preprocessing phase that includes the invocation of qsVOLE. We can instantiate the protocol in [7] for the large field, which has sublinear communication costs and relies on the ring-LPN based pseudorandom correlation generators in [22]. However, for the binary or small fields, we adopt a QuickSilver-based instantiation, requiring a linear amount of communication (efficiently pushing at least one-third of the computation into the offline phase).

1.5. Related Works

The construction provided by AntMan [14] is the first to achieve sublinear communication, but it comes with a computational overhead of $\mathcal{O}(\log |C|)$. On the other hand, Interactive Line-Point Zero-Knowledge (ILPZK) [23] achieves sublinear communication with linear computation, but it has a round complexity of $O(d \log S)$, where d represents the depth of the circuit and S is the number of gates. Unlike QuickSilver and LPZKv2, which only introduce extra multiplications beyond circuit evaluation, ILPZK's computation is dominated by the sum-check protocol. Additionally, rather than adopting the streamlined approaches used in other works (e.g. [6], [7]), both AntMan and ILPZK process wires in batches. In this work, we focus on constant-round VOLE-based ZKPs that are streamable.

Recently, Baum et al. [24] introduces a method known as VOLE-in-the-head, which can upgrade the VOLE-based ZKPs to support public verifiability. They give the construction based on the checking method in QuickSilver to guarantee the multipication gate is correctly evaluated. Our current focus is on providing a more efficient checking method, while deferring how to apply the technique to make our protocols publicly verifiable in our further work.

2. Preliminaries

Let κ and ϵ denote the computational and statistical security parameters, respectively. Let $x \xleftarrow{R} F$ denote that sampling x uniformly at random from a finite set F, $[n]$ denote a set $\{1, \cdots, n\}$, and $[a, b]$ denote a set $\{a, \cdots, b\}$ where $n, a, b \in \mathbb{N}$. We also abuse the notation $[u]$ to denote the authenticated value where $u \in \mathbb{F}_p$. The specific meaning should be clear from the context. For the column vector with n entries, we represent it as a set $\{x_{\alpha_i}\}_{i\in[n]}$ or the bold letter $\boldsymbol{x}$ where α_i denotes the index of the element in $\boldsymbol{x}$.

Let $\mathbb{F}_p$ be a finite field and $\mathbb{F}_{p^r}$ its extension field, such that $\mathbb{F}_{p^r} \cong \mathbb{F}_p[X]/f(X)$, where $p \geq 2$ is a prime or a power of a prime, $r \geq 1$ is an integer, and $f(X)$ is a fixed monic irreducible polynomial of degree r. Every element $w \in \mathbb{F}_{p^r}$ can be uniquely represented as $w = \sum_{i\in[r]} a_i \cdot X^{i-1}$, where each $a_i \in \mathbb{F}_p$. When elements from both $\mathbb{F}_p$ and $\mathbb{F}_{p^r}$ appear in the same arithmetic expression, it is understood that elements of $\mathbb{F}_p$ are viewed as polynomials in $\mathbb{F}_{p^r}$ with only constant terms. A circuit $\mathcal{C}$ over a field $\mathbb{F}_p$ is defined by a set of input wires $\mathcal{I}_{\mathsf{in}}$ and a list of gates, each represented as $(f_j, \mathcal{I}_j, \upsilon_j)$. Here, j is the gate index in gate indices set $\mathcal{G}$, f_j is a degree-2 polynomial acting on the input wires of the gate, $\mathcal{I}_j$ contains the indices of the gate's input wires. In this case, $f_j = g_{j,2} + g_{j,1} + g_{j,0}$ where $g_{j,2}$ is homogeneous polynomial of degree 2, $g_{j,1}$ is homogeneous polynomial of degree 1, and $g_{j,0}$ denotes the constant. The index υ_j represents the gate's output wire. For example, given the input indices $\mathcal{I}_j = \{\alpha, \rho, \beta\}$, $f_j = w_{\upsilon_j} = w_\alpha \cdot w_\rho + w_\beta + c$ where $g_{j,2} = w_\alpha \cdot w_\rho$, $g_{j,1} = w_\beta$, and $g_{j,0} = c$.

Initialize Upon receiving (init) from $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ works as follows,

- If $\mathcal{V}$ is honest, sample $x \xleftarrow{R} \mathbb{F}_{p^r}$. Otherwise, waits for x from the adversary. $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ stores x and sends it to $\mathcal{V}$, and ignore all subsequent (init) commands.

Extension On input (extend, n) from $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ works as follows,

- If $\mathcal{V}$ is honest, sample $\boldsymbol{k} \xleftarrow{R} \mathbb{F}_{p^r}^n$. Otherwise, receive $\boldsymbol{k}$ from the adversary.
- If $\mathcal{P}$ is honest, sample $\boldsymbol{u} \xleftarrow{R} \mathbb{F}_p^n$ and compute $\boldsymbol{m} := \boldsymbol{k} - \boldsymbol{u} \cdot x \in \mathbb{F}_{p^r}^n$. Otherwise, waits for $\boldsymbol{u}, \boldsymbol{m}$ from the adversary, and then recompute $\boldsymbol{k} := \boldsymbol{m} + \boldsymbol{u} \cdot x \in \mathbb{F}_{p^r}^n$.
- Output $\boldsymbol{k}$ to $\mathcal{V}$ and output $\boldsymbol{u}, \boldsymbol{m}$ to $\mathcal{P}$.

Vector Oblivious Polynomial Evaluation On input (VOPE, d) from $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ works as follows,

- If $\mathcal{V}$ is honest, sample $K \xleftarrow{R} \mathbb{F}_{p^r}$. Otherwise, receive $K \in \mathbb{F}_{p^r}$ from the adversary.
- If $\mathcal{P}$ is honest, sample $M_i \xleftarrow{R} \mathbb{F}_{p^r}$ for $i \in [d]$ and compute $M_0 := K - \sum_{i\in[d]} M_i \cdot x^i \in \mathbb{F}_{p^r}$. Otherwise, waits for $\{M_i\}_{i\in[0,d]}$ with $M_i \in \mathbb{F}_{p^r}$ from the adversary, and then recompute $K := \sum_{i\in[0,d]} M_i \cdot x^i \in \mathbb{F}_{p^r}$.
- Output K to $\mathcal{V}$ and output $\{M_i\}_{i\in[0,d]}$ to $\mathcal{P}$.

Figure 1. Functionality for extended subfield VOLE.

Input Upon receiving (input, id, w) and (input, id) from $\mathcal{P}$ and $\mathcal{V}$, respectively, it stores (id, w) if id a fresh identifier.

Prove Upon receiving $(\mathsf{prove}, \mathcal{C}, id_1, ..., id_n)$ and $(\mathsf{verify}, \mathcal{C}, id_1, ..., id_n)$ from $\mathcal{P}$ and $\mathcal{V}$ respectively, it retrieves (id_i, w_i) for $i \in [n]$ where $id_1, ..., id_n$ are stored in memory. Send true to $\mathcal{V}$ if $\mathcal{C}(\boldsymbol{w}) = 1$ and false otherwise.

Figure 2. The zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathsf{cir}}$ for circuit satisfiability.

The layered circuit is defined in the same way, except for the requirement that $\mathcal{L}(\upsilon_j) = \mathcal{L}(i) + 1$ for every $i \in \mathcal{I}_j$ where $\mathcal{L} : \mathbb{N} \to \mathbb{N}$ is a grading that assigns the index of each wire to a layer. Let $\mathcal{L}(i) = 0$ for $i \in \mathcal{I}_{\mathsf{in}}$.

Extended (Subfield) VOLE Functionality. We recall the extended sVOLE functionality $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ used in [6] here, and $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ is shown in Fig. 1. The standard sVOLE functionality, which includes Initialize and Extension, sends a uniform global $x \in \mathbb{F}_{p^r}$ and keys $\boldsymbol{k} \in \mathbb{F}_{p^r}^n$ to the $\mathcal{V}$, while sends values $(\boldsymbol{m} \in \mathbb{F}_{p^r}^n, \boldsymbol{u} \in \mathbb{F}_p^n)$ to the $\mathcal{P}$. In order to support our protocols, we require extended sVOLE functionality. This extended functionality is similar to the standard sVOLE, but it also allows both parties to derive VOPE correlations over $\mathbb{F}_{p^r}$, ensuring that a unified global key is utilized for both sVOLE and VOPE. To be specific, when both parties input a degree-d polynomial, this functionality selects $d+1$ uniform coefficients over $\mathbb{F}_{p^r}$ to define a random polynomial, then provides the coefficients to the $\mathcal{P}$ and the evaluation at x to the $\mathcal{V}$. The extended functionality is realized in [6] based on the standard sVOLE functionality which can be efficiently realized as silent using recent LPN-based protocols [1], [3], [5], [11], [22], [25].

Zero-knowledge Proof Functionality. The zero-knowledge proof functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathsf{cir}}$ is shown in Fig. 2.

3. JQv1 for Circuit over Any Field

In this section, we first introduce the qsVOLE used in the preprocessing phase of JQv1 and JQv2. We then present our JQv1 protocol for circuit satisfiability over any field.

3.1. Quadratic Subfield VOLE

The $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ functionality defined in the Fig 3 is proposed to compute $[y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]$. This functionality can be realized from the extended sVOLE functionality by the method of QuickSilver that extends to support degree-2 polynomial gates. We provide the protocol $\Pi_{\mathsf{qsVOLE}}^{p,r}$ that UC-realizes $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ and its security proof in Appendix A.

Theorem 1. *The protocol $\Pi_{\mathsf{qsVOLE}}^{p,r}$ UC-realizes functionality $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ in the $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$-hybrid model.*

3.2. JQv1 Protocol

Our JQv1 $\Pi_{\mathsf{ZK}}^{\mathsf{cir}}$ protocol is shown in Fig. 4. We have discussed the intuition of our protocols in Section 1.3.1, and thus here describe details about our protocols applied to arbitrary degree-2 polynomial gates. Recall that the circuit $\mathcal{C}$ is defined as $\{(f_j, \mathcal{I}_j, \upsilon_j)\}_{j\in\mathcal{G}}$.

In the preprocessing phase, two parties first obtain random authenticated values for the circuit inputs and the output wires of gates. They then generate $[y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]$ for every gate via $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$.

During the online phase, after $\mathcal{P}$ sends d values corresponding to the circuit inputs and all gate output wires, two parties compute $[w_{\upsilon_j}]$ for each gate. To ensure each gate j is correctly computed, we define the degree-1 checking polynomial $f(X)$ as follows. Given $\{d_i\}_{i\in\mathcal{I}_j}$ sent by $\mathcal{P}$, and f_j with t degree-2 terms, let the set $\{(c_i, \alpha_i, \rho_i)\}_{i\in[t]}$ consist of tuples where $c_i \in \mathbb{F}_p$ are coefficients and α_i, ρ_i are input indices for each degree-2 term of f_j. Besides, we define polynomials $p_{u_i}(X) = m_{u_i} + u_i \cdot X$ for $i \in \{(\alpha_i, \rho_i)\}_{i\in[t]}$, $p_{\upsilon_j}(X) = m_{\upsilon_j} + w_{\upsilon_j} \cdot X$, and $p_{y_j}(X) = m_{y_j} + y_j \cdot X$, then we have $f(X)$ equals

$$\begin{aligned}
&\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot p_{u_{\rho_i}}(X) + d_{u_{\rho_i}} \cdot p_{u_{\alpha_i}}(X)) \\
&\qquad + p_{y_j}(X) + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot X - p_{\upsilon_j}(X) \\
&= \sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot m_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot m_{u_{\alpha_i}}) + m_{y_j} - m_{w_{\upsilon_j}} \\
&\qquad + (f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) - w_{\upsilon_j}) \cdot X
\end{aligned}$$

Initialize Upon receiving (init) from $\mathcal{P}$ and $\mathcal{V}$, and $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ works as follows,

- If $\mathcal{V}$ is honest, sample $x \xleftarrow{R} \mathbb{F}_{p^r}$. Otherwise, waits for x from the adversary. $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ stores x and sends it to $\mathcal{V}$, and ignore all subsequent (init) commands.
- $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ initializes the set S as empty.

Extension On input $(\mathsf{extend}, n, \{\tau_i\}_{i\in[n]})$ from $\mathcal{P}$ and $\mathcal{V}$, and $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ works as follows,

- If $(\tau_i, \cdot)$ exist in S for any $i \in [n]$, it aborts.
- $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ follows the same procedure as $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ to generate n relations, and outputs $\{[u_{\tau_i}]\}_{i\in[n]}$ to parties.
- Store $(\tau_i, u_{\tau_i}, m_{u_{\tau_i}}, k_{u_{\tau_i}})$ in S for every $i \in [n]$.

Vector Oblivious Polynomial Evaluation On input (VOPE, d) from $\mathcal{P}$ and $\mathcal{V}$,

- $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ follows the same procedure as $\mathcal{F}_{\mathsf{ext\text{-}sVOLE}}^{p,r}$ to generate a random polynomial with degree-d, then outputs coefficients to $\mathcal{P}$ and the evaluation at x to $\mathcal{V}$.

Quadratic On input $(\mathsf{quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ from $\mathcal{P}$ and $\mathcal{V}$, and $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ works as follows.

- If $(i, \cdot)$ exist for every $i \in \mathcal{I}_j$, retrieve $\{(i, u_i, \cdot)\}_{i\in\mathcal{I}_j}$ from S. Otherwise, aborts.
- 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 $\{k_{y_j}\}_{j\in\mathcal{G}} \xleftarrow{R} \mathbb{F}_{p^r}^{|\mathcal{G}|}$. Otherwise, wait for $\{k_{y_j}\}_{j\in\mathcal{G}}$ from the adversary.
- If $\mathcal{P}$ is honest, compute $m_{y_j} := k_{y_j} - y_j \cdot x \in \mathbb{F}_{p^r}$ for every $j \in \mathcal{G}$. Otherwise, wait for $\{m_{y_j}\}_{j\in\mathcal{G}}$ from the adversary, and then recompute $k_{y_j} := m_{y_j} + y_j \cdot x \in \mathbb{F}_{p^r}$ for every $j \in \mathcal{G}$.
- Output $\{[y_j]\}$ to $\mathcal{P}$ and $\mathcal{V}$ for every $j \in \mathcal{G}$.

Figure 3. Quadratic subfield VOLE functionality

whose coefficients are held by $\mathcal{P}$, and evaluation at x is held by $\mathcal{V}$. If and only if $f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) = w_{v_j}$, $f(X)$ is a constant polynomial. After that, two parties conduct the batch check via the hash function (modeled as the random oracle) and check the correctness of the circuit output wire.

Large Field Circuit Optimization. In the context of a large field where r can be set to 1, we can optimize the computation of degree-2 terms in f_j by reducing the number of multiplications. Instead of performing two multiplications for each term by multiplying d values with m values (resp. k values), we can reduce this to a single multiplication. Specifically, rather than calculating the term $\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot m_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot m_{u_{\alpha_i}})$, we compute $g_{j,2}(\{d_i + m_{u_i}\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j})$ where $g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j})$ can be precomputed. This substitution is valid because

$$\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot m_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot m_{u_{\alpha_i}}) =$$
$$g_{j,2}(\{d_i + m_{u_i}\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{d_i\}_{i\in\mathcal{I}_j})$$

This approach can also be applied to the term $\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot k_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot k_{u_{\alpha_i}})$. Additionally, the subtraction of $g_{j,2}(\{d_i\}_{i\in\mathcal{I}_j})$ can be omitted since $f(x) + g_{j,2}(\{d_i\}_{i\in\mathcal{I}_j}) = a_0 + g_{j,2}(\{d_i\}_{i\in\mathcal{I}_j})$ if $f(x) = a_0$.

Theorem 2. *Assuming the existence of random oracles, $\Pi_{\mathsf{ZK}}^{\mathsf{cir}}$ UC-realizes functionality $\mathcal{F}_{\mathsf{ZK}}$ that proves the circuit satisfiability in the $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$-hybrid model with soundness error $(q_{\mathsf{H}} + 1)/p^r + 1/2^\kappa$ and computational security.*

We present proof sketches here and defer the detailed proof to Appendix B.1. The simulator $\mathcal{S}$ runs the corrupted party as the subroutine and acts as $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ to interact with it. For a corrupted $\mathcal{P}$, simulating the view is straightforward since the honest $\mathcal{V}$ does not send any values to $\mathcal{P}$, and the witness can be reconstructed by obtaining u values from $\mathcal{P}$ while acting as $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ and receiving d values sent by $\mathcal{P}$. Additionally, $\mathcal{S}$ will abort if it receives an abort from $\mathcal{F}_{\mathsf{ZK}}$ or the honest $\mathcal{V}$ aborts. To analyze the soundness error, we analyze the probability that a corrupted $\mathcal{P}$ deviates from the protocol yet still passes the batch check or the final check of the circuit output on the verifier's side. In the first scenario, a cheating $\mathcal{P}$ could succeed by guessing either the input or output of H in the verifier's check. With the unknown global key $x \in \mathbb{F}_{p^r}$, the verifier's evaluations at x yield p^r possible hash inputs from the $\mathcal{P}$'s view. Consequently, a malicious $\mathcal{P}$ with q_H random oracle queries achieves $\leq q_{\mathsf{H}}/p^r$ success probability for input guessing. Simultaneously, the probability of output guessing is $\leq 1/2^\kappa$. In the second scenario, a cheating prover could succeed by guessing x with a probability of at most $1/p^r$.

For a corrupted $\mathcal{V}$, $\mathcal{S}$ acting as $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$ obtains all k values generated during the preprocessing phase. To simulate the view, $\mathcal{S}$ first samples and sends random d values to $\mathcal{V}$. During the batch checking, $\mathcal{S}$ computes $\{k_{j,\mathsf{zero}}\}_{j\in\mathcal{G}}$ from the previously stored k values, samples a random A, and programs the random oracle so that $\mathsf{H}(k_{1,\mathsf{zero}}, \ldots, k_{|\mathcal{G}|,\mathsf{zero}}) := A$, then sends A to $\mathcal{V}$. For the final check, $\mathcal{S}$ sends $m_h := k_h - x$ to $\mathcal{V}$.

3.3. Cost Analysis

In this section, we present the cost analysis of computation for the prover and verifier in the online phase, and com-

$\mathcal{P}$ and $\mathcal{V}$ hold a circuit $\mathcal{C}$ over any field $\mathbb{F}_p$. $\mathcal{P}$ also holds witnesses $\boldsymbol{w}$ such that $\mathcal{C}(\boldsymbol{w}) = 1$ and $|\boldsymbol{w}| = n$ (i.e., $|\mathcal{I}_{\mathsf{in}}| = n$). Let $\{\tau_i\}_{i\in[n]}$ represent the indices of the input wires of $\mathcal{C}$, and let $\{\tau_i\}_{i\in[n+1,n+|\mathcal{G}|]}$ represent the indices of the output wires of gates in $\mathcal{C}$, respectively. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle.

PREPROCESS

1) $\mathcal{P}$ and $\mathcal{V}$ sends init to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, which returns a uniform $x \in \mathbb{F}_p$ to $\mathcal{V}$.
2) $\mathcal{P}$ and $\mathcal{V}$ sends $(\mathsf{extend}, n+|\mathcal{G}|, \{\tau_i\}_{i\in[n+|\mathcal{G}|]})$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ which returns $\{[u_{\tau_i}]\}_{i\in[n+|\mathcal{G}|]}$ to parties.
3) Precomputing for gates $\{(f_j, \mathcal{I}_j, \cdot) \in \mathcal{C}\}_{j\in\mathcal{G}}$:
   - $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{poly}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ which returns $\{[y_j] = [f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]\}_{j\in\mathcal{G}}$ to parties.

ONLINE

1) For $\tau_i \in \mathcal{I}_{\mathsf{in}}$, $\mathcal{P}$ sends $d_{\tau_i} := w_{\tau_i} - u_{\tau_i} \in \mathbb{F}_p$ to $\mathcal{V}$, and two parties compute $[w_{\tau_i}] := [u_{\tau_i}] + d_{\tau_i}$.
2) In a topological order, for each gate $(f_j, \mathcal{I}_j, v_j) \in \mathcal{C}$ this is the j'-th gate in $\mathcal{G}$:
   - $\mathcal{P}$ sends $d_{v_j} := f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{n+j'}}$ to $\mathcal{V}$, and two parties compute $[w_{v_j}] := [u_{\tau_{n+j'}}] + d_{v_j}$.
3) For each gate j, $\mathcal{P}$ and $\mathcal{V}$ hold $(\{[w_i]\}_{i\in\mathcal{I}_j}, [w_{v_j}])$ where $w_i = u_i + d_i$ for every $i \in \mathcal{I}_j \cup v_j$, and $w_{v_j} = f_j(\{w_i\}_{i\in\mathcal{I}_j})$ (resp. $w_{v_j} = f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j})$):
   - $\mathcal{P}$ computes $m_{j,\mathsf{zero}} := g_{j,2}(\{m_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) + m_{y_j} - m_{w_{v_j}}$.
   - $\mathcal{V}$ computes $k_{j,\mathsf{zero}} := g_{j,2}(\{k_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) + k_{y_j} - k_{w_{v_j}} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot x$.
4) $\mathcal{P}$ and $\mathcal{V}$ perform the following check to verify that $m_{j,\mathsf{zero}} = k_{j,\mathsf{zero}}$ for all $j \in \mathcal{G}$.
   - $\mathcal{P}$ computes $A := \mathsf{H}(m_{1,\mathsf{zero}}, \ldots, m_{|\mathcal{G}|,\mathsf{zero}})$, and sends A to $\mathcal{V}$.
   - $\mathcal{V}$ computes $B := \mathsf{H}(k_{1,\mathsf{zero}}, \ldots, k_{|\mathcal{G}|,\mathsf{zero}})$, and checks that $A = B$. If the check fails, $\mathcal{V}$ outputs false and aborts.
5) For the output wire h, both parties hold $[w_h]$ with $k_h = m_h + w_h \cdot x$. They verify that $w_h = 1$ as follows.
   - Concurrently with the previous step, $\mathcal{P}$ sends m_h to $\mathcal{V}$.
   - $\mathcal{V}$ checks that $k_h = m_h + x$. If the check fails, $\mathcal{V}$ outputs false and aborts. Otherwise, $\mathcal{V}$ outputs true.

Figure 4. UC-secure zero-knowledge protocol for circuit satisfiability over any field.

pare our work with the state-of-the-arts. In terms of communication, our work JQv1, QuickSilver, and IT-LPZKv2 all require sending one field element in $\mathbb{F}_p$ per multiplication gate. Note that both $\mathcal{P}$ and $\mathcal{V}$ should each invoke H once to perform the check for all gates.

Prover. In the online phase, $\mathcal{P}$ computes $d_{v_j} := f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{n+j'}}$ and $m_{j,\mathsf{zero}}$ for each gate, where $m_{y_j} - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - m_{w_{v_j}}$ and $m_{w_{v_j}} = m_{u_{\tau_{n+j'}}}$ are independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{P}$ requires one evaluation of f_j and one addition over $\mathbb{F}_p$ to compute d_{v_j}, one evaluation of $g_{j,2}$ and one addition over $\mathbb{F}_{p^r}$ to compute $m_{j,\mathsf{zero}}$.

Verifier. In the online phase, $\mathcal{V}$ computes $k_{v_j} := k_{u_{\tau_{n+j'}}} + d_{v_j} \cdot x \in \mathbb{F}_{p^r}$ and $k_{j,\mathsf{zero}}$ for each gate, where $k_{y_j} - g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j})$ is independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{V}$ requires one scalar product of $d_{v_j} \in \mathbb{F}_p$ on x and one addition over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$. $\mathcal{V}$ 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 x over $\mathbb{F}_{p^r}$, and three additions over $\mathbb{F}_{p^r}$ to compute $k_{j,\mathsf{zero}}$.

Comparison with QuickSilver. We provide the comparison between our JQv1 and QuickSilver Table 2. Since

TABLE 2. ONLINE COST PER MULT-GATE OVER ANY FIELD.

Prover	$\mathbb{F}_p$		$\mathbb{F}_{p^r}$			
	$\times$	$+$	$\times$	$+$	Scal	H
QuickSilver [6]	1	1	4	4	2	0
JQv1	1	1	0	2	2	< 1
Verifier	$\times$	$+$	$\times$	$+$	Scal	H
QuickSilver [6]	0	0	4	3	1	0
JQv1	1	0	0	5	3	< 1

QuickSilver does not provide a detailed cost analysis, we analyze it here and compare their work with our JQv1. When compared with QuickSilver for the small field r can not be set to 1, the evaluation of $g_{j,2}$ includes costly multiplications over $\mathbb{F}_{p^r}$. Therefore, instead of computing $g_{j,2}(\{m_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j})$ to reduce the number of the multiplications between b values with m values, $\mathcal{P}$ performs the scalar products of $d_i \in \mathbb{F}_p$ on $m_{u_i} \in \mathbb{F}_{p^r}$ within the protocol. This is similar to $\mathcal{V}$. For the prover-side computation, compared to our work, $\mathcal{P}$ in QuickSilver requires four additional multiplications over $\mathbb{F}_{p^r}$ per multiplication gate. In our protocol, $\mathcal{P}$ needs

TABLE 3. ONLINE COST PER MULT-GATE OVER LARGE FIELDS AFTER APPLYING THE OPTIMIZATIONS DISCUSSED IN SECTION 3.2.

Prover	×	+	H
IT-LPZKv2 [7]	3	5	0
JQv1	2	2	< 1
Verifier	×	+	H
IT-LPZKv2 [7]	3	2	0
JQv1	4	4	< 1

one multiplication and one addition over $\mathbb{F}_p$ to compute d_{v_j}, and two scalar products and two additions over $\mathbb{F}_{p^r}$ to compute $m_{j,\mathsf{zero}}$, since $m_{y_j} - m_{w_{v_j}}$ can be precomputed where $m_{w_v} = m_{u_{\tau_{n+j'}}}$. In contrast, QuickSilver requires $\mathcal{P}$ to perform one multiplication and one addition over $\mathbb{F}_p$ to compute d_{v_j}, one multiplication over $\mathbb{F}_{p^r}$ to compute A_0, two scalar products and two addition over $\mathbb{F}_{p^r}$ to compute A_1, one multiplication over $\mathbb{F}_{p^r}$ to compute χ^j given χ^{j-1}, and two multiplications and two additions over $\mathbb{F}_{p^r}$ to batch associated values into one for the check.

For the verifier-side computation, compared to our work, $\mathcal{V}$ in QuickSilver requires four additional multiplication over $\mathbb{F}_{p^r}$. In our protocol, $\mathcal{V}$ needs one scalar product and one addition over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$, two scalar products and four additions over $\mathbb{F}_{p^r}$ to compute $k_{j,\mathsf{zero}}$. In contrast, QuickSilver requires $\mathcal{V}$ to perform one scalar product and one addition over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$, two multiplication and one addition over $\mathbb{F}_{p^r}$ to compute B_j, one multiplication over $\mathbb{F}_{p^r}$ to compute χ^j given χ^{j-1}, and one multiplication and one addition over $\mathbb{F}_{p^r}$ for each multiplication gate to batch into one value for the check.

Comparison with IT-LPZKv2 for Large Field. We present the comparison between our JQv1 and IT-LPZKv2 for the large field in Table 3 using the number of IT-LPZKv2 reported in their work. The measurement of both their work and our work is based on the large field $\mathbb{F}_{2^{61}-1}$.

4. JQv2 for Layered Circuit over Any Field

In this section, we introduce our $\Pi_{\mathsf{ZK}}^{\mathsf{laycir}}$ protocol for the layered circuit satisfiability over any field, achieving amortized communication of $\frac{1}{2}$ field element per gate via $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$. Let $\mathcal{G}_0$ and $\mathcal{G}_1$ represent the sets of gate indices for the even and odd layers, respectively. For simplicity, we assume the even layers have more gates than the odd layers here. The $\Pi_{\mathsf{ZK}}^{\mathsf{laycir}}$ protocol is shown in Fig. 5.

During the preprocessing phase, compared to JQv1, the difference is that, in addition to handling circuit inputs, two parties generate random authenticated values only for the output wires of gates in the odd layers and generate $[y_j]$ only for each gate in the even layers.

In the online phase, two parties compute $[w_{v_j}]$ directly without further communication for the even layers. Specifically, they compute $k_{v_j} = m_{v_j} + w_{v_j} \cdot x$ as follows:

$$\underbrace{\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot k_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot k_{u_{\alpha_i}}) + k_{y_j} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot x}_{\text{Known to } \mathcal{V}\text{, denoted as } k_{v_j}}$$

$$= \underbrace{\sum_{i\in[t]} c_i \cdot (d_{u_{\alpha_i}} \cdot m_{u_{\rho_i}} + d_{u_{\rho_i}} \cdot m_{u_{\alpha_i}}) + m_{y_j}}_{\text{Known to } \mathcal{P}\text{, denoted as } m_{v_j}} + w_{v_j} \cdot x$$

Additionally, the optimization for large field circuits discussed in Section 3.2, can be applied here to reduce the number of multiplications performed by the two parties. For the odd layers, they compute $[w_{v_j}]$ by having $\mathcal{P}$ send d_{v_j}, then two parties perform a batch check similar to QuickSilver, based on the evaluation of degree-2 polynomial at x.

We can apply the Fiat-Shamir heuristic to make the online phase non-interactive. Specifically, both parties can compute χ from the transcript up to that point using a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{F}_{p^r}$, modeled as a random oracle, where $p^r \geq 2^\kappa$.

Theorem 3. *The protocol $\Pi_{\mathsf{ZK}}^{\mathsf{laycir}}$ UC-realizes functionality $\mathcal{F}_{\mathsf{ZK}}$ that proves the circuit satisfiability in the $\mathcal{F}_{\mathsf{qsVOLE}}^{p,r}$-hybrid model with soundness error $(3 + |\mathcal{G}_1|)/p^r$ and information-theoretic security.*

We have deferred the formal proof to Appendix B.2.

4.1. Cost Analysis

In this section, we present the cost analysis of computation for the even and odd layers in the online phase from both the prover and verifier sides. We then compare our work with the state-of-the-art works QuickSilver and ROM-LPZKv2. In terms of communication, our work JQv1 and ROM-LPZKv2 require sending $\frac{1}{2}$ field element in $\mathbb{F}_p$ per multiplication gate, while QuickSilver requires sending one.

Prover. For $j \in \mathcal{G}_0$, $\mathcal{P}$ computes $m_{w_{v_j}}$ where the value $m_{y_j} - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j})$ and $m_{w_{v_j}} = m_{u_{\tau_{n+j'}}}$ are independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{P}$ requries one evaluation of f_j over $\mathbb{F}_p$ to compute w_{v_j}, one evaluation of $g_{j,2}$ and one addition over $\mathbb{F}_{p^r}$ to compute $m_{w_{v_j}}$. For $j \in \mathcal{G}_1$, $\mathcal{P}$ computes $d_{v_j} = f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{n+j'}}$, $a_{0,j}$, $a_{1,j}$, u_0 and u_1 where $m_{w_{v_j}} = m_{u_{\tau_{n+j'}}}$ is independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{P}$ requires one evaluation of f_j on $\{w_i\}_{i\in\mathcal{I}_j}$ and one addition over $\mathbb{F}_p$ to compute d_{v_j}, one evaluation of $g_{j,2}$ over $\mathbb{F}_{p^r}$ to compute $a_{0,j}$, one evaluation of f_j and three additions over $\mathbb{F}_{p^r}$ to compute $a_{1,j}$ where the evaluation f_j on $\{w_i\}_{i\in\mathcal{I}_j}$ and $g_{j,2}(\{m_{w_i}\}_{i\in\mathcal{I}_j})$ have been computed above, one multiplication over $\mathbb{F}_{p^r}$ to compute $\chi^{j'}$ given $\chi^{j'-1}$, and two multiplications and two additions over $\mathbb{F}_{p^r}$ to batch associated values into one for the check.

Verifier. For $j \in \mathcal{G}_0$, $\mathcal{V}$ computes $k_{w_{v_j}}$ where the value $k_{y_j} - g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j})$ is independent of the witnesses, and

$\mathcal{P}$ and $\mathcal{V}$ hold a layered circuit $\mathcal{C}$ over any field $\mathbb{F}_p$. $\mathcal{P}$ also holds witnesses $\boldsymbol{w}$ such that $\mathcal{C}(\boldsymbol{w}) = 1$ and $|\boldsymbol{w}| = n$ (i.e., $|\mathcal{I}_{\mathsf{in}}| = n$). Let $\mathcal{G}_0$ and $\mathcal{G}_1$ contains indices of gates in all even layers and odd layers, respectively. Let $\{\tau_i\}_{i\in[n]}$ represent the indices of the input wires of $\mathcal{C}$, and let $\{\tau_i\}_{i\in[n+1,n+|\mathcal{G}_1|]}$ represent the indices of the output wires of gates in odd layers, respectively.

PREPROCESS

1) $\mathcal{P}$ and $\mathcal{V}$ sends init to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, which returns a uniform $x \in \mathbb{F}_p$ to $\mathcal{V}$.
2) $\mathcal{P}$ and $\mathcal{V}$ sends $(\mathsf{extend}, n + |\mathcal{G}_1|, \{\tau_i\}_{i\in[n+|\mathcal{G}_1|]})$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ which returns $\{[u_{\tau_i}]\}_{i\in[n+|\mathcal{G}_1|]}$ to parties.
3) $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{VOPE}, 1)$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ which returns M_0, M_1 to $\mathcal{P}$ and K to $\mathcal{V}$.
4) Precomputing for gates in even layers $\{(f_j, \mathcal{I}_j, \cdot) \in \mathcal{C}\}_{j\in\mathcal{G}_0}$:
   - $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}_0})$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ which returns $\{[y_j] = [f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}]\}_{j\in\mathcal{G}_0}$ to parties.

ONLINE

1) For $\tau_i \in \mathcal{I}_{\mathsf{in}}$, $\mathcal{P}$ sends $d_{\tau_i} := w_{\tau_i} - u_{\tau_i} \in \mathbb{F}_p$ to $\mathcal{V}$, and two parties compute $[w_{\tau_i}] := [u_{\tau_i}] + d_{\tau_i}$.
2) In a topological order, for each gate $(f_j, \mathcal{I}_j, v_j) \in \mathcal{C}$:
   - If $j \in \mathcal{G}_0$, with $w_i = u_i + d_i$ for $i \in \mathcal{I}_j$, $\mathcal{P}$ computes $m_{w_{v_j}} := g_{j,2}(\{m_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) + m_{y_j}$, and $\mathcal{V}$ computes $k_{w_{v_j}} := g_{j,2}(\{k_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) + k_{y_j} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot x$.
   - If $j \in \mathcal{G}_1$ and this is the j'-th gate in $\mathcal{G}_1$, then $\mathcal{P}$ sends $d_{v_j} := f_j(\{w_i\}_{i\in\mathcal{I}_j}) - u_{\tau_{n+j'}}$ to $\mathcal{V}$, and two parties compute $[w_{v_j}] := [u_{\tau_{n+j'}}] + d_{v_j}$.
3) For each $j \in \mathcal{G}_1$ gate with $f_j = g_{j,2} + g_{j,1} + g_{j,0}$, $\mathcal{P}$ and $\mathcal{V}$ hold authenticated values $(\{[w_i]\}_{i\in[\mathcal{I}_j]}, [w_{v_j}])$ where $k_i = m_i + w_i \cdot x$ for $i \in \mathcal{I}_j \cup v_j$ from the previous step and execute the following:
   - $\mathcal{P}$ computes $a_{0,j} = g_{j,2}(\{m_{w_i}\}_{i\in\mathcal{I}_j}) \in \mathbb{F}_{p^r}$, and $a_{1,j} = f_j(\{m_{w_i} + w_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{w_i}\}_{i\in\mathcal{I}_j}) - f_j(\{w_i\}_{i\in\mathcal{I}_j}) - m_{w_{v_j}} \in \mathbb{F}_{p^r}$.
   - $\mathcal{V}$ computes $b_j = g_{j,2}(\{k_{w_i}\}_{i\in\mathcal{I}_j}) + (g_{j,1}(\{k_{w_i}\}_{i\in\mathcal{I}_j}) + g_{j,0} \cdot x - k_{w_{v_j}}) \cdot x \in \mathbb{F}_{p^r}$
4) $\mathcal{P}$ and $\mathcal{V}$ corperate to ensure $b_{j'} = a_{0,j'} + a_{1,j'} \cdot x$ for every j'-th gate in $\mathcal{G}_1$.
   - $\mathcal{V}$ samples $\chi \xleftarrow{R} \mathbb{F}_{p^r}$, and sends it to $\mathcal{P}$.
   - $\mathcal{P}$ computes $u_0 := \sum_{j'\in[|\mathcal{G}_1|]} a_{0,j'} \cdot \chi^{j'} + M_0$, $u_1 := \sum_{j'\in[|\mathcal{G}_1|]} a_{1,j'} \cdot \chi^{j'} + M_1$, and sends (u_0, u_1) to $\mathcal{V}$.
   - $\mathcal{V}$ computes $w := \sum_{j'\in[|\mathcal{G}_1|]} b_{j'} \cdot \chi^{j'} + K$ and if $w \neq u_0 + u_1 \cdot x$, $\mathcal{V}$ aborts.
5) For the output wire h, both parties hold $[w_h]$ with $k_h = m_h + w_h \cdot x$. They verify that $w_h = 1$ as follows.
   - Concurrently with the previous step, $\mathcal{P}$ sends m_h to $\mathcal{V}$.
   - $\mathcal{V}$ checks that $k_h = m_h + x$. If the check fails, $\mathcal{V}$ outputs false and aborts. Otherwise, $\mathcal{V}$ outputs true.

Figure 5. UC-secure zero-knowledge protocol for layered circuit satisfiability over any field.

can be precomputed in the preprocessing phase. Therefore, $\mathcal{V}$ requries one evaluation of f_j over $\mathbb{F}_p$, one evaluation of $g_{j,2}$ over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$. For $j \in \mathcal{G}_1$, $\mathcal{V}$ computes $k_{w_{v_j}}$, b_j and w where $g_{j,0} \cdot x$ is independent of the witnesses, and can be precomputed in the preprocessing phase. Therefore, $\mathcal{V}$ requires one scalar product of $d_{v_j} \in \mathbb{F}_p$ on x and one addition over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$. Then, $\mathcal{V}$ requires one evaluation of $g_{j,2}, g_{j,1}$, one multiplication and three additions over $\mathbb{F}_{p^r}$ to compute b_j. Finally, $\mathcal{V}$ performs one multiplication over $\mathbb{F}_{p^r}$ to compute χ^j given χ^{j-1}, one multiplication and one addition over $\mathbb{F}_{p^r}$ for each multiplication gate to batch into one value for the check.

Comparison with QuickSilver. We compare our JQv2 and QuickSilver in Table 4. Similar to the previous analysis for JQv1 that compared with QuickSilver, we let $\mathcal{P}$ perform the scalar products of $d_i \in \mathbb{F}_p$ on $m_{u_i} \in \mathbb{F}_{p^r}$ within the protocol, akin to $\mathcal{V}$. The cost analysis of QuickSilver is the same as in Section 3.3, so we omit it here. The data in the table represents the amortized computation per multiplication gate, which is the sum of the costs for $j \in \mathcal{G}_0$ and $j \in \mathcal{G}_1$, divided by 2. For $j \in \mathcal{G}_0$, $\mathcal{P}$ performs one multiplication over $\mathbb{F}_p$ to evaluate the multiplication gate and two additions and two scalar products to compute $m_{w_{v_j}}$, and

TABLE 4. ONLINE COST PER MULT-GATE OVER ANY FIELD.

Prover	$\mathbb{F}_p$		$\mathbb{F}_{p^r}$			
	$\times$	$+$	$\times$	$+$	Scal	H
QuickSilver [6]	1	1	4	4	2	0
JQv2	1	0.5	2	3	2	0
Verifier	$\times$	$+$	$\times$	$+$	Scal	H
QuickSilver [6]	0	0	4	3	1	0
JQv2	0.5	0	2	3	2	0

TABLE 5. ONLINE COST PER MULT-GATE OVER LARGE FIELDS AFTER APPLYING THE OPTIMIZATIONS DISCUSSED IN SECTION 3.2.

Prover	×	+	H
ROM-LPZKv2 [7]	8.5	8.5	< 1
JQv2	4	3.5	0
Verifier	**×**	**+**	**H**
ROM-LPZKv2 [7]	8.5	8.5	< 1
JQv2	4	3.5	0

$\mathcal{V}$ performs one multiplication over $\mathbb{F}_p$, three addtions and three scalar products over $\mathbb{F}_{p^r}$ to compute $k_{w_{v_j}}$. For $j \in \mathcal{G}_1$, the analysis is the same as in QuickSilver, as discussed in Section 3.3, for both $\mathcal{P}$ and $\mathcal{V}$, and is therefore omitted here.

Comparison with ROM-LPZKv2. We present the comparison between our JQv2 and ROM-LPZKv2 for the large field in Table 5 using the number of ROM-LPZKv2 reported in their work. In LPZKv2 [7], the measurement is based on the large field $\mathbb{F}_{2^{61}-1}$ with a computational security parameter $\kappa = 128$. In JQv2, setting $r = 1$ is sufficient for the information-theoretic security when $q \geq 2^{\epsilon}$.

5. Practical Applications

In this section, we demonstrate that our protocol can prove the inner product, matrix multiplication, and solutions to lattice problems. For JQv1, H outputs the element with κ bits. For JQv2, we always assume that $p^r \approx 2^{\kappa}$ as the Fiat-Shamir heuristic is assumed to be implicitly used.

Inner Product. $\mathcal{P}$ holds $\{x_i\}_{i\in[n]}$ and intends to prove $\sum_{i\in[n/2]} c_i \cdot x_i \cdot x_{n/2+i} = c$ with the public coefficients $\{c_i\}_{i\in[n]}$ and the public sum c. The inner product can be present as a degree-2 polynomial $f(x_1, \cdots, x_n) = \sum_{i\in[n/2]} c_i \cdot x_i \cdot x_{n/2+i} - c = 0$. $\mathcal{P}$ first commit to inputs by sending n fields elements. When applying the underlying technology of JQv1, two parties compute $[f(x_1, \cdots, x_n)]$ by having $\mathcal{P}$ send $d := f(x_1, \cdots, x_n) - u$ where $[u]$ is random authenticated value. However, using the technology of JQv2, two parties directly compute $[f(x_1, \cdots, x_n)] := \sum_{i\in[n/2]} c_i \cdot ([u_i] + d_i) \cdot ([u_{n/2+i}] + d_{n/2+i}) - c$. Since there is one-layer multiplication, the remaining task is to check $[f(x_1, \cdots, x_n)] = [0]$ in both JQv1 and JQv2, which can be done by having $\mathcal{P}$ send m of $[f(x_1, \cdots, x_n)]$ to $\mathcal{V}$, who performs the check. Therefore, it needs communication of $(n+1)\log p + \kappa$ bits in JQv1 and $n \log p + \kappa$ bits in JQv2 in the online phase.

Matrix Multiplication. $\mathcal{P}$ holds two secret matrices $\mathbf{A}, \mathbf{B} \in (\mathbb{F}_p^{n\times n})^2$, and intends to prove $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ with a public matrix $\mathbf{C} \in \mathbb{F}_p^{n\times n}$ known to the verifier. The matrix multiplication can be viewed as n^2 inner product. Besides, two parties can perform the batch check for n^2 inner products via the random oracle. Therefore, it needs communication of $3n^2 \log p + \kappa$ bits for JQv1 and $2n^2 \log p + \kappa$ bits for JQv2 in the online phase.

Proving Solutions to Lattice Problems. $\mathcal{P}$ holds $\mathbf{s} \in \{0,1\}^m$, and intends to prove $\mathbf{A} \cdot \mathbf{s} = \mathbf{t}$ with the public matrix $\mathbf{A} \in \mathbb{Z}_p^{n\times m}$ and vector $\mathbf{t} \in \mathbb{Z}_p^n$. In this case, $\mathcal{P}$ should prove the following: 1) the relation holds, 2) s_i is indeed a bit. Specifically, $\mathcal{P}$ should prove $\sum_{j\in[m]} a_{i,j} \cdot s_j - t_i = 0$ for $i \in [n]$ and $s_i^2 - s_i = 0$ for $i \in [m]$ where $a_{i,j}$ is the entry in the i-th row and j-th column of matrix $\mathbf{A}$. All of the above can be formalized as $n + m$ polynomials. Therefore, JQv1 offers a ZK protocol with communication $(2m+n)\log p + \kappa$ bits, while JQv2 provides a ZK protocol with communication $m \log p + \kappa$ bits. If the secret vector $\mathbf{s}$ lies within $[-B, B]$ (where B is a small integer) instead of being a binary vector, as addressed in prior works [6], [11], [26], [27], [28], we can prove it using the relation $\prod_{j\in[-B,B]}(s_i - j) = 0$ for $i \in [m]$, which requires $2mB$ multiplications for all s_i. Using our protocols, this requires a communication cost of $(m + n + 2mB)\log p + \kappa$ bits in JQv1 and $(m + mB)\log p + 3\kappa$ bits in JQv2.

6. Implementation and Benchmarking

We implemented JQv1 and JQv2 based on the publicly available implementation of QuickSilver [16] and subsequently evaluated their performance. We employ the same hardware configuration in previous works [6], [7], [15]. Specifically, experiments of JQv1 and JQv2, along with their applications, were conducted on two Amazon EC2 *m5.2xlarge*[4] using a single thread. Our implementations achieve a computational security of $\kappa = 128$. For arithmetic circuits over a 61-bit field, we achieve $\kappa = 128$ and $\epsilon = 40$ where Mersenne prime $p = 2^{61} - 1$, consistent with prior work. In our implementation, we instantiated the COT protocol (i.e., sVOLE with $p = 2$ and $r = \kappa$) and the VOLE protocol over a 61-bit field by using the recent protocols [5], [11]. In our testing, we found that when using an Intel-type CPU, the cryptographic hash function Blake3 outperformed SHA-256 by at least a factor of 10 when batching 30 millions 61-bit fied elements. Conversely, when using an AMD-type CPU, SHA-256 proved to be at least 2 times faster than Blake3. Therefore, we employ the faster version depending on the CPU type of the instance.

6.1. Benchmarking JQv1 and JQv2

We evaluate the performance of JQv1 and JQv2 by proving circuits with 3×10^8 AND/MULT gates and report the number of gates per second that can be proven using our protocols in Table 6. We evaluate JQv1 using the same circuit structure as implemented in QuickSilver, while JQv2 is evaluated on the random circuit described in Section 1.3.2.

Comparison with Prior Work. We have compared the performance of our JQv1 and JQv2 with prior work in Table 1 using the same configuration. We used the most recent numbers of QuickSilver [6] reported on their Github [16]. Although the number 7.01 M/sec of AntMan is obtained under a 1 Gbps network bandwidth, our experiments under different bandwidth settings shown in Table 6 present that

4. Intel Xeon Platinum 8259CL CPU@2.50GHz, 8 vCPUs, 32GiB RAM, throttled Network

TABLE 6. BENCHMARK THE ONLINE PERFORMANCE OF OUR PROTOCOLS WITHIN A SINGLE THREAD.

	Boolean Circuit				Arithmetic Circuit			
	20 Mbps	30 Mbps	50 Mbps	Local-host	500 Mbps	1 Gbps	2 Gbps	Local-host
JQv1	19.5 M/sec	40.6 M/sec	64.1 M/sec	64.1 M/sec	7.4 M/sec	14.2 M/sec	23.3 M/sec	23.3 M/sec
JQv2	34 M/sec	34 M/sec	34 M/sec	34 M/sec	12.5 M/sec	12.5 M/sec	12.5 M/sec	12.5 M/sec

This table shows the number of million 'M' gates per second that each protocol can prove under different network settings within a single thread.

TABLE 7. DETAILED BREAKDOWN OF TIME COSTS FOR PROVING 3×10^8 AND/MULT GATES WITHIN A SINGLE THREAD

	Boolean Circuit (Second)				Arithmetic Circuit (Second)			
	Bandwidth	Cir-Ind	Cir-Dep	Online	Bandwidth	Cir-Ind	Cir-Dep	Online
JQv1	30 Mbps	3 s	32 s	7.5 s	1 Gbps	3 s	38 s	21 s
	50 Mbps/Local-host	3 s	32 s	4.5 s	2 Gbps/Local-host	3 s	32 s	13 s
JQv2	30 Mbps	2 s	20 s	9 s	1 Gbps	2 s	24 s	24 s
	50 Mbps/Local-host	2 s	20 s	9 s	2 Gbps/Local-host	2 s	20 s	24 s
QuickSilver	30 Mbps	3 s	-	32 s	1 Gbps	3 s	-	42 s
	50 Mbps/Local-host	3 s	-	32 s	2 Gbps/Local-host	3 s	-	36 s

Cir-Ind generates random authenticated values when the upper bound of multiplication gates is known, while Cir-Dep executes quadratic subfield VOLE once the circuit is known. The online phase occurs when the witness is available. The results are obtained under different network settings.

TABLE 8. STRESS-TESTING THE ONLINE PHASE OF OUR PROTOCOLS WITHIN A SINGLE THREAD.

	Instance Information			Boolean Circuit		Arithmetic Circuit	
	Type	cents/hour	CPU	gates/sec	gates/cent	gates/sec	gates/cent
JQv1	t3a.small	1.8	AMD	46 M	92 B	29 M	58 B
JQv2				32 M	64 B	8.9 M	18 B
JQv1	t3.small	2	Intel	53.8 M	96 B	14.1 M	25 B
JQv2				30 M	54 B	7.9 M	14 B

This table presents the number of million 'M' gates each protocol can prove per second and the number of billion 'B' gates each protocol can prove per cent. All instances have 2 vCPUs, 2 GiB memory, and at most 5 Gbps network bandwidth.

JQv1 and JQv2 achieve roughly $2\times$ and $1.7\times$ improvements, respectively, under the same 1 Gbps bandwidth. Additionally, JQv2 requires a minimum network speed of around 10 Mbps for the boolean circuit and around 400 Mbps for the arithmetic circuit.

Detailed Breakdown of Time Costs. We divide the total running time into three phases: circuit-independent preprocessing (Cir-Ind), circuit-dependent preprocessing (Cir-Dep), and the online phase for JQv1, JQv2, and QuickSilver with the results detailed in Table 7. There are two preprocessing models for VOLE-based ZKPs: Model 1(circuit-independent only, e.g.,QuickSilver) and Model 2(combined circuit-independent/dependent phases, e.g.,LPZKv2). JesseQ achieves state-of-the-art performance in Model 2 concerning total running time but trails QuickSilver in Model 1 (except JQv2 in the layered Boolean circuit).

Stress-testing of JQv1 and JQv2. To show our JQv1 and JQv2 are *affordable*, we benchmarked them on the cheapest Amazon EC2 instances (*t3a.small*[5], *t3.small*[6]). These instances cost around 2 cents per hour, offering a cheaper choice compared to QuickSilver's stress test environment, which costs from 2 to 5 cents per hour. Our results are presented in Table 8.

5. AMD EPYC 7571@2.50GHz, 2 vCPUs, 2GiB RAM, $\leq$5 Gpbs

6. Intel Skylake 8175M@2.50GHz, 2 vCPUs, 2GiB RAM, $\leq$5 Gpbs

6.2. Benchmarking Practical Applications

For the applications discussed in Section 5, the primary difference between JQv1 and JQv2 is that JQv1 lets two parties interactively compute the values associated with the outputs for degree-2 polynomials. This requires $\mathcal{P}$ to send one additional element per gate and $\mathcal{V}$ to adjust the value associated with the output for each gate. Besides, QuickSilver only reports their performance for applications based on the degree-2 polynomial-based ZK protocol. Therefore, we also only benchmark JQv2 for applications involving one-depth multiplication. Our experiments use the same network configuration as QuickSilver for those applications. Specifically, we use a network bandwidth of 20 Mbps for binary fields and 500 Mbps for 61-bit fields, always utilizing a single thread. Below, we provide comparisons between our results and previous work, with the number reported in QuickSilver [6](Section 6.2).

Inner Product. We benchmark the inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i \in [n]} x_i \cdot y_i$ of two vectors $\mathbf{x}, \mathbf{y}$ where each vector has n entries, and report the cost of the online phase (when

TABLE 9. ONLINE PERFORMANCE COMPARISON BETWEEN QuickSilver AND OUR PROTOCOLS FOR INNER PRODUCT.

	$\mathbb{F}_2$ (ms)			$\mathbb{F}_{2^{61}-1}$ (ms)		
Length of vectors	10^6	10^7	10^8	10^6	10^7	10^8
QuickSilver [6]	36	69	423	42	100	703
JQv2	3.2	38	400	5.9	60.6	648

TABLE 10. ONLINE PERFORMANCE OF VARIOUS PROTOCOLS FOR PROVING MATRIX MULTIPLICATION.

	Execution Time	Communication
Spartan [29]	≥ 5000 s	≤ 100 KB
Virgo [9]	357 s	221 KB
Wolverine [11]	1627 s	34 GB
Mac'n'Cheese [12]	2684 s	25.8 GB
QuickSilver [6]	10 s	25.2 MB
JQv2	7 s	16.7 MB

Results are based on two 1024×1024 matrices over a 61-bit field, with their product being public.

the witness is known) for proving the inner product in Table 9. In QuickSilver, the costs of processing the witness and proving the inner product are reported separately. As the process for processing the witness between QuickSilver and our protocols is similar, we will only present the cost of proving the inner product.

Matrix Multiplication. We benchmark the process of proving knowledge of two 1024×1024 matrices over a 61-bit field. The comparison of the online phase with prior work, as reported in QuickSilver, is presented in Table 10. Wolverine and Mac'n'Cheese are executed on local-host, while our protocols, QuickSilver and Virgo were tested over a 500 Mbps network.

Proving Knowledge of Solutions to Lattice Problems. We perform a benchmark of the process for proving knowledge of a solution vector $\mathbf{s}$ to a Short Integer Solution (SIS) problem. In this scenario, the equation $\mathbf{A} \cdot \mathbf{s} = \mathbf{t}$ must be satisfied, where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{t} \in \mathbb{Z}_q^n$ are known to two parties. Furthermore, we enforce a constraint that the elements of $\mathbf{s}$ should fall within the range of $[-1, 1]$, which aligns with the parameters utilized in QuickSilver. In our experiments, we set $n = 2048$, $m = 1024$, and $\log q = 61$. We also assessed the cost of proving knowledge of an SIS solution for QuickSilver using their open-source implementation in this setting, as QuickSilver only provides performance metrics for a prime q of 31 bits.

6.3. Benchmarking Sublinear Framework

Our experiments for the sublinear framework based on JQv1 and JQv2 are conducted on two Amazon EC2 *m5.8xlarge*[7], also using a single thread. We use the number of AntMan and Batchman reported in their works and run the experiments on the same hardware. AntMan runs with 16 threads, while Batchman uses only 1 thread. The circuit in AntMan [14] is defined over $\mathbb{F}_{2^{59}-2^{28}+1}$, while the circuit in both QS-Batchman and JQ*-Batchman is defined over $\mathbb{F}_{2^{61}-1}$. JQv2-Batchman outperforms JQv1-Batchman, as Batchman tests on the circuit of the matrix multiplication, and the first only needs to send one element to perform the final batch check in the online phase.

7. Intel Xeon Platinum 8259 CPU@2.5GHz, 32 vCPUs, 128GiB RAM, 10Gbps

TABLE 11. ONLINE PERFORMANCE OF PROVING KNOWLEDGE OF AN SIS SOLUTION WITH PARAMETERS $n = 2048$, $m = 1024$, AND $\log q = 61$

	Execution Time	Communication
QuickSilver [6]	22 ms	8.2 KB
JQv2	7 ms	8.2 KB

TABLE 12. ONLINE COMPARISON OF SUBLINEAR WORKS.

	100 Mbps	500 Mbps	1 Gbps
AntMan [14]	15.86 M/sec	17.51 M/sec	17.74 M/sec
QS-Batchman [6]	104.91 M/sec	335.02 M/sec	461.82 M/sec
JQv1-Batchman	122.26 M/sec	569.44 M/sec	1051.01 M/sec
JQv2-Batchman	144.76 M/sec	666.47 M/sec	1190.22 M/sec

This table presents the number of million 'M' gates each protocol can prove per second. Protocols execute batches, with each repetition having 2^{21} multiplication gates under different bandwidth setting.

TABLE 13. PERFORMANCE OF MULTIPLICATIONS AND HASH.

	61-bit Field (ms)			128-bit Field (ms)		
Num. of fields	10^6	10^7	10^8	10^6	10^7	10^8
MUL	3.865	38.54	386.1	8.984	90.72	901.3
SHA256	20.33	203.4	2029	40.24	401.3	4007
BLAKE3	1.833	17.61	173.8	3.576	35.42	349.1

6.4. Benchmarking Hash and Multiplications

We benchmarked the execution time of $10^6 \sim 10^8$ multiplications as well as the hash function with $10^6 \sim 10^8$ field inputs over the 61-bit field (when $p = 2^{61} - 1$) and the 128-bit field (when $p = 2$). These benchmarks were conducted using the Amazon EC2 instance type *m5.2xlarge* with a single thread, and the performance is presented in Table 13. We benchmarked the multiplications using the Streaming SIMD Extensions (SSE) instruction set, as implemented in QuickSilver [16]. For hashing, we utilized the OpenSSL library [30] for SHA256 and the official library [21] for BLAKE3. The results show that the BLAKE3 hash function processes inputs $10^6 \sim 10^8$ fields at least twice as fast as performing the same number of multiplications, whether operating over a 61-bit or 128-bit field.

References

[1] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 896–912.

[2] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 489–518.

[3] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, "Distributed vector-ole: Improved constructions and implementation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1055–1072.

[4] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 291–308.

[5] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1607–1626.

[6] K. Yang, P. Sarkar, C. Weng, and X. Wang, "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*, 2021, pp. 2986–3001.

[7] S. Dittmer, Y. Ishai, S. Lu, and R. Ostrovsky, "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*, 2022, pp. 829–841.

[8] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 2016, pp. 305–326.

[9] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 859–876.

[10] J. Lee, S. Setty, J. Thaler, and R. Wahby, "Linear-time and post-quantum zero-knowledge snarks for r1cs," *Cryptology ePrint Archive*, 2021.

[11] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1074–1091.

[12] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl, "Mac'n'cheese mac' n' cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*. Springer, 2021, pp. 92–122.

[13] S. Dittmer, Y. Ishai, and R. Ostrovsky, "Line-point zero knowledge and its applications," *Cryptology ePrint Archive*, 2020.

[14] C. Weng, K. Yang, Z. Yang, X. Xie, and X. Wang, "Antman: Interactive zero-knowledge proofs with sublinear communication," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2901–2914.

[15] Y. Yang, D. Heath, C. Hazay, V. Kolesnikov, and M. Venkitasubramaniam, "Batchman and robin: Batched and non-batched branching for interactive zk," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1452–1466.

[16] X. Wang, A. J. Malozemoff, J. Katz *et al.*, "Emp-toolkit: efficient multiparty computation toolkit (2016)," 2021.

[17] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 169–188.

[18] D. Catalano and D. Fiore, "Practical homomorphic macs for arithmetic circuits," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 336–352.

[19] I. Damgård and S. Zakarias, "Constant-overhead secure computation of boolean circuits using preprocessing," in *Theory of Cryptography Conference*. Springer, 2013, pp. 621–641.

[20] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[21] "Blake3: https://github.com/blake3-team/blake3," 2021.

[22] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators from ring-lpn," in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40*. Springer, 2020, pp. 387–416.

[23] F. Lin, C. Xing, and Y. Yao, "Interactive line-point zero-knowledge with sublinear communication and linear computation," *Cryptology ePrint Archive*, 2024.

[24] C. Baum, L. Braun, C. D. de Saint Guilhem, M. Klooß, E. Orsini, L. Roy, and P. Scholl, "Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head," in *Annual International Cryptology Conference*. Springer, 2023, pp. 581–615.

[25] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: silent vole and oblivious transfer from hardness of decoding structured ldpc codes," in *Annual International Cryptology Conference*. Springer, 2021, pp. 502–534.

[26] J. Bootle, V. Lyubashevsky, and G. Seiler, "Algebraic techniques for short (er) exact lattice-based zero-knowledge proofs," in *Annual International Cryptology Conference*. Springer, 2019, pp. 176–202.

[27] C. Boschini, J. Camenisch, M. Ovsiankin, and N. Spooner, "Efficient post-quantum snarks for rsis and rlwe and their applications to privacy," in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 2020, pp. 247–267.

[28] M. F. Esgin, N. K. Nguyen, and G. Seiler, "Practical exact proofs from lattices: New techniques to exploit fully-splitting rings," in *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. Springer, 2020, pp. 259–288.

[29] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[30] "Openssl: https://github.com/openssl/openssl," 2021.

Appendix A.
Protocol that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$

The $\Pi^{p,r}_{\mathsf{qsVOLE}}$ that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ is shown in Fig 6. This protocol requires the correlation $K = M_0 + M_1 \cdot x$ where $K \in \mathbb{F}_{p^r}$ is held by $\mathcal{V}$, and $M_0, M_1 \in \mathbb{F}_{p^r}$ are held by $\mathcal{P}$ to mask the value sent by $\mathcal{P}$ in our protocols. This correlation can be generated by the extended subfield VOLE functionality presented in Section 2. We can apply the Fiat-Shamir heuristic to make the online phase non-interactive. , albeit at the expense of degrading information-theoretic security to computational security.

Theorem 4. *The protocol $\Pi^{p,r}_{\mathsf{qsVOLE}}$ UC-realizes functionality $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ in the $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$-hybrid model.*

Let H be the random oracle $\{0,1\}^* \to \mathbb{F}_{p^r}$.

Initialize Upon receiving (init) from $\mathcal{P}$ and $\mathcal{V}$,

- $\mathcal{P}$ and $\mathcal{V}$ send (init) $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ which returns $x \in \mathbb{F}_{p^r}$ to $\mathcal{V}$.
- $\mathcal{P}$ and $\mathcal{V}$ locally initialize the set S as empty.

Extension On input $(\mathsf{extend}, n, \{\tau_i\}_{i\in[n]})$ from $\mathcal{P}$ and $\mathcal{V}$,

- If $(\tau_i, \cdot)$ exist in S for any $i \in [n]$, parties aborts.
- $\mathcal{P}$ and $\mathcal{V}$ send (extend, n) to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ which returns $\{[u_{\tau_i}]\}_{i\in[n]}$ to parties.
- $\mathcal{P}$ and $\mathcal{V}$ store $(\tau_i, [u_{\tau_i}])$ in S for every $i \in [n]$.

Vector Oblivious Polynomial Evaluation On input (VOPE, d) from $\mathcal{P}$ and $\mathcal{V}$,

- $\mathcal{P}$ and $\mathcal{V}$ send (VOPE, d) to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ which returns M_0, M_1 to $\mathcal{P}$ and K to $\mathcal{V}$.

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

1) If $(i, \cdot)$ exist for every $i \in \mathcal{I}_j$, $\mathcal{P}$ and $\mathcal{V}$ retrieve $\{[u_i]\}_{i\in\mathcal{I}_j}$ from S. Otherwise, aborts.
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 $(\mathsf{extend}, |\mathcal{G}|)$ to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ which returns $\{[v_j]\}_{j\in\mathcal{G}}$ to parties.
4) $\mathcal{P}$ and $\mathcal{V}$ send $(\mathsf{VOPE}, 1)$ to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ which returns M_0, M_1 to $\mathcal{P}$ and K to $\mathcal{V}$.
5) $\mathcal{P}$ sends $d_j = y_j - v_i$ to $\mathcal{V}$, then $\mathcal{P}$ and $\mathcal{V}$ compute $[y_j] = [v_j] + d_j$.
6) 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 $b_j = (g_{j,1}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) - k_{y_j}) \cdot x + g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) \in \mathbb{F}_{p^r}$.
7) $\mathcal{P}$ and $\mathcal{V}$ corperate to ensure $b_{j'} = a_{0,j'} + a_{1,j'} \cdot x$ for every j'-th gate in $\mathcal{G}$.
   - $\mathcal{V}$ samples $\chi \xleftarrow{R} \mathbb{F}_{p^r}$, and sends to $\mathcal{P}$.
   - $\mathcal{P}$ computes $u_0 = \sum_{j'\in[|\mathcal{G}|]} a_{0,j'} \cdot \chi^{j'} + M_0$, $u_1 = \sum_{j'\in[|\mathcal{G}|]} a_{1,j'} \cdot \chi^{j'} + M_1$, and sends (u_0, u_1) to $\mathcal{V}$.
   - $\mathcal{V}$ computes $w = \sum_{j'\in[|\mathcal{G}|]} b_{j'} \cdot \chi^{j'} + K$ and if $w \neq u_0 + u_1 \cdot x$, $\mathcal{V}$ aborts.

Figure 6. The Quadratic Subfield VOLE protocol that UC-realizes $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.

Proof. We first consider the case of a malicious prover and then consider the case of a malicious verifier. In each case, we construct a simulator $\mathcal{S}$ could only access to an ideal functionality $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, and running the adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$ for $\mathcal{A}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

Corrupted prover. $\mathcal{S}$ interacts with $\mathcal{A}$ and $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ as follows,

1) Upon receiving init query, $\mathcal{S}$ sends init to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
2) Upon receiving $(\mathsf{extend}, n, \{\tau_i\}_{i\in[n]})$ query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $\{u_{\tau_i}, m_{u_{\tau_i}}\}_{i\in[n]}$ that $\mathcal{P}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores and sends those values to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
3) Upon receiving (VOPE, d) query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $\{M_i\}_{i\in[0,d]}$ that $\mathcal{P}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores and sends those values to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
4) Upon receiving $(\mathsf{quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $\{v_j, m_{v_j}\}_{j\in\mathcal{G}}$, M_0, M_1 that $\mathcal{P}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores those values. Then $\mathcal{S}$ computes $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}, d_j = y_j - v_j, m_{y_j} = m_{v_j}$ for every $j \in \mathcal{G}$. Upon receiving $\{d'_j\}_{j\in\mathcal{G}}$ from $\mathcal{A}$, if there exists any j such that $d'_j \neq d_j$, $\mathcal{S}$ sends $\perp$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ and aborts. Otherwise, $\mathcal{S}$ samples and sends $\chi \xleftarrow{R} \mathbb{F}_{p^r}$ to $\mathcal{A}$, and sends $\{m_{y_j}\}_{j\in\mathcal{G}}$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
5) Upon receiving (u_0, u_1) from $\mathcal{A}$, $\mathcal{S}$ computes u'_0, u'_1 as the protocol does based on the values stored. If $u_0 \neq u'_0$ or $u'_1 \neq u_1$, $\mathcal{S}$ sends $\perp$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ and aborts.

The adversary $\mathcal{A}$'s views have identical distributions in both the ideal and real-world executions. Whenever the verifier in the real-world execution aborts, the verifier in the ideal-world execution also aborts, since $\mathcal{S}$ sends $\perp$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ in this case. It is trivial that the computation in init, extend and VOPE process are correct. Therefore, it remains to bound the probability that the verifier in the real-world execution passes the final check in quadratic process while $\mathcal{A}$ sends incorrect values in step 5 or step 7 or both steps. The probability that the honest verifier in the real-world execution passes the final check in this case is at most $(2 + |\mathcal{G}|)/p^r$.

For the gate j, which is j'-th gate in $\mathcal{G}$, two parties hold $\{[u_i]\}_{i\in\mathcal{I}_j}, [y_j]$ where $[y_j] = [f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}] + e_{v_j}$. Therefore, we have $k_{y_j} = m_{y_j} + (f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}) \cdot x +$

$e_{v_j} \cdot x$. Given the equation is hold:

$$g_{j,1}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) + \sum_{i\in[|\mathcal{G}|]} c_i \cdot (u_{\alpha_i} \cdot m_{u_{\rho_i}} + u_{\rho_i} \cdot m_{u_{\alpha_i}})$$
$$= f_j(\{m_{u_i} + u_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - f_j(\{u_i\}_{i\in\mathcal{I}_j})$$

we have the following,

$$\begin{aligned} b_j =& (g_{j,1}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) - k_{y_j}) \cdot x + g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) \\ =& g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) + (g_{j,1}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) \\ &+ \sum_{i\in[|\mathcal{G}|]} c_i \cdot (u_{\alpha_i} \cdot m_{u_{\rho_i}} + u_{\rho_i} \cdot m_{u_{\alpha_i}}) - m_{y_j}) \cdot x \\ &+ (g_{j,2}(\{u_i\}_{i\in\mathcal{I}_j}) + g_{j,1}(\{u_i\}_{i\in\mathcal{I}_j}) - y_j) \cdot x^2 \\ =& g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - e_{v_j} \cdot x^2 + (f_j(\{m_{u_i} + u_i\}_{i\in\mathcal{I}_j}) \\ &- g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) - f_j(\{u_i\}_{i\in\mathcal{I}_j}) - m_{y_j}) \cdot x \\ =& a_{0,j} + a_{1,j} \cdot x - e_{v_j} \cdot x^2 \end{aligned}$$

In the step 7, $\mathcal{A}$ sends $u'_0 = u_0 + e_{u,0}$ and $u'_1 = u_1 + e_{u,1}$ to the honest $\mathcal{V}$ where u_0, u_1 are computed following the protocol description, $e_{u,0}, e_{u,1} \in \mathbb{F}_{p^r}$ are the adversarially chosen errors. Furthermore, we have the following:

$$\begin{aligned} w =& \sum_{j'\in[|\mathcal{G}|]} b_{j'} \cdot \chi^{j'} + K \\ =& \sum_{j'\in[|\mathcal{G}|]} (a_{0,j'} + a_{1,j'} \cdot x - e_{v_{j'}} \cdot x^2)\chi^{j'} + M_0 + M_1 \cdot x \\ =& (u'_0 - e_{u,0}) + (u'_1 - e_{u,1}) \cdot x - (\sum_{j'\in[|\mathcal{G}|]} e_{v_{j'}} \cdot \chi^{j'}) \cdot x^2 \end{aligned}$$

If the check passes in step 4, then we have $w = u'_0 + u'_1 \cdot x$. Therefore, we can obtain that

$$e_{u,0} + e_{u,1} \cdot x + (\sum_{j'\in[|\mathcal{G}|]} e_{v_{j'}} \cdot \chi^{j'}) \cdot x^2 = 0 \qquad (1)$$

Case 1: $\sum_{j'\in[|\mathcal{G}|]} e_{v_{j'}} \cdot \chi^{j'} \neq 0$, the equation 1 is held. Since $x \in \mathbb{F}_{p^r}$ is uniformly random and hidden from $\mathcal{A}$'s view, the equation 1 holds with probability at most $2/p^r$.

Case 2: $\sum_{j'\in[|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'} = 0$, the equation 1 is held. The probability of this case is less than the probability of $\sum_{j'\in[|\mathcal{G}|]} e_{v_{j'}} \cdot \chi^{j'} = 0$. If there exists some $j' \in [|\mathcal{G}|]$ such that $e_{v_{j'}} \neq 0$, the probability of $\sum_{j'\in[|\mathcal{G}|]} e_{v_{j'}} \cdot \chi^{j'} = 0$ is at most $|\mathcal{G}|/p^r$, as χ is sampled uniformly at random after all $e_{v_{j'}}$ have been determined.

In conclusion, any unbounded environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, except with probability $(2 + |\mathcal{G}|)/p^r$.

Corrupted verifier. $\mathcal{S}$ interacts with $\mathcal{A}$ and $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ as follows,

1) Upon receiving init query, $\mathcal{S}$ sends init to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $x \in \mathbb{F}_{p^r}$ that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores x and sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
2) Upon receiving $(\mathsf{extend}, n, \{\tau_i\}_{i\in[n]})$ query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $\{k_{u_{\tau_i}}\}_{i\in[n]}$ that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores and sends those values to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
3) Upon receiving (VOPE, d) query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives K that $\mathcal{P}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, it stores and sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
4) Upon receiving $(\mathsf{quad}, \{f_j, \mathcal{I}_j\}_{j\in\mathcal{G}})$ query, $\mathcal{S}$ sends it to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. When $\mathcal{S}$ receives $\{k_{v_j}\}_{j\in\mathcal{G}}$, K that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{ext\text{-}sVOLE}}$, $\mathcal{S}$ samples $d_j \xleftarrow{R} \mathbb{F}_p$ and computes $k_{y_j} = k_{v_j} + d_j \cdot x$ for every $j \in \mathcal{G}$. Then $\mathcal{S}$ sends $\{d_j\}_{j\in\mathcal{G}}$ to $\mathcal{A}$ and sends $\{k_{y_j}\}_{j\in\mathcal{G}}$ to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. Upon receiving χ from $\mathcal{A}$, $\mathcal{S}$ samples $u_1 \xleftarrow{R} \mathbb{F}_{p^r}$, and compute $u_0 = w - u_1 \cdot x$ where w is computed using x, K and the keys received from $\mathcal{A}$ following the protocol specification. Then $\mathcal{S}$ sends u_0, u_1 to $\mathcal{A}$.

Since v_j, d_j and M_1 are uniformly random and v_j and M_1 are hidden against the view of adversary $\mathcal{A}$, we easily obtain that the view of $\mathcal{A}$ simulated by $\mathcal{S}$ is distributed identically to its view in the real protocol execution. This completes the proof.

Appendix B. Deferred Proofs

B.1. Proof of Theorem 2

Proof. We consider the case of a malicious prover (i.e., soundness and knowledge extraction), and then consider the case of a malicious verifier (i.e., zero knowledge). In each case, we construct a simulator $\mathcal{S}$ that could only access an ideal functionality $\mathcal{F}_{\mathsf{ZK}}$ and run the adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ and random oracle for $\mathcal{A}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

Corrupted prover. $\mathcal{S}$ maintains a hash list L_h for the random oracle H. On any query $y \in \{0,1\}^*$ to H, if $\exists(y, h_y) \in L_h$, return h_y, else return a uniformly random $h_y \in \{0,1\}^\kappa$ and add (y, h_y) to L_h. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows,

1) $\mathcal{S}$ receives init that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, samples $x \xleftarrow{R} \mathbb{F}_{p^r}$, and stores $\{u_{\tau_i}, m_{u_{\tau_i}}\}_{i\in[n+|\mathcal{G}|]}$ that $\mathcal{P}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. Then $\mathcal{S}$ compute the corresponding keys $\{k_{u_{\tau_i}}\}_{i\in[n+|\mathcal{G}|]}$. Upon receiving $\{m_{y_j}\}_{j\in\mathcal{G}}$ that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, $\mathcal{S}$ computes and stores $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ for every $j \in \mathcal{G}$ where u_is are stored before, and the corresponding keys $\{k_{y_j}\}_{j\in\mathcal{G}}$.
2) For $\tau_i \in \mathcal{I}_{\mathsf{in}}$, upon receiving d_{τ_i} from $\mathcal{A}$, $\mathcal{S}$ computes and stores $w_{\tau_i} = u_{\tau_i} + d_{\tau_i}$.
3) $\mathcal{S}$ executes the rest of the protocol as an honest verifier, using x and the keys stored in the first step. If the honest verifier outputs false, then $\mathcal{S}$ sends $\boldsymbol{w} = \perp$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}$ and aborts. If the honest verifier outputs true, then $\mathcal{S}$ sends $\boldsymbol{w}$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}$ where $\boldsymbol{w} = (w_1, \cdots, w_n)$ stored in the second step.

The adversary $\mathcal{A}$'s views have identical distributions in both the ideal and real-world executions. Whenever the verifier

in the real-world execution outputs false, the verifier in the ideal-world execution also outputs false, since $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{\mathsf{ZK}}$ in this case. Therefore, it remains to bound the probability that the verifier in the real-world execution outputs true while the witness $\boldsymbol{w}$ sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{ZK}}$ satisfies $\mathcal{C}(\boldsymbol{w}) = 0$. Then the probability that the honest verifier in the real-world execution outputs true is at most $(q_{\mathsf{H}}+1)/p^r + 1/2^\kappa$.

It is trivial that the values associated with the input wires and the output wires of ADD gates are computed correctly. Therefore, the required probability is at most the sum of the probabilities of the following two cases.

Case 1: When $\mathcal{C}(\boldsymbol{w}) \neq 1$ and there exists a multiplication gate where the MAC and keys of the output wire value are not correctly computed from the MAC and keys of the input wires during step 2, which means that $[w_v] = [f_j(\{w_i\}_{i\in\mathcal{I}_j})] + e_v$ where $e_v \in \mathbb{F}_p$ is an error introduced by the adversary $\mathcal{A}$ by sending an incorrect value $d'_v = d_v + e_v$, $\mathcal{V}$ pass the check in step 4 which means that $A = B$.

Let the multiplication gate that is not correctly computed be the i-th multiplication gate. Two parties hold $\{[w_i] = [u_i] + d_i\}_{i\in\mathcal{I}_{\mathsf{in}}}, [w'_{v_j}] = [w_{v_j}] + e_{v_j} = [u_{\tau_{n+j'}}] + d_{v_j} + e_{v_j}$ where $[u_i], d_i, [u_{\tau_{n+j'}}], d_{v_j}$ are computed following the protocol description. Given that the correctly computed values $y_i = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ and $f_j(\{u_i + d_i\}_{i\in\mathcal{I}_j}) = w_{v_j}$, we have $k_{j,\mathsf{zero}}$ eqauls

$$
\begin{aligned}
& g_{j,2}(\{k_{u_i} + d_i\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{k_{u_i}\}_{i\in\mathcal{I}_j}) \\
&\qquad + k_{y_j} - k_{w'_{v_j}} + f_j(\{d_i\}_{i\in\mathcal{I}_j}) \cdot x \\
=& g_{j,2}(\{d_i + m_{u_i}\}_{i\in\mathcal{I}_j}) - g_{j,2}(\{m_{u_i}\}_{i\in\mathcal{I}_j}) \\
&\qquad + m_{y_j} - m_{w_{v_j}} - e_{v_j} \cdot x \\
=& m_{j,\mathsf{zero}} - e_{v_j} \cdot x
\end{aligned}
$$

where $m_{j,\mathsf{zero}}$ is computed following the protocol description. Since $\{m_{j,\mathsf{zero}}\}_{j\in\mathcal{G}}$ and e_{v_j} are known by $\mathcal{A}$, x have p^r possible values, as does $\{k_{j,\mathsf{zero}}\}_{j\in\mathcal{G}}$. Then the probability $\mathsf{H}(x_1) = \mathsf{H}(x_2)$ where x_1 consists of $\{m_{i,\mathsf{zero}}\}_{j\in\mathcal{G}}$ and x_2 consists of $\{k_{j,\mathsf{zero}}\}_{j\in\mathcal{G}}$ is less than the sum of following two probabilities,

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

Case 2: When $\mathcal{C}(\boldsymbol{w}) = 0$ 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 $[w_v] = [f_j(\{w_i\}_{i\in\mathcal{I}_j})]$, $\mathcal{V}$ outputs true. Then $\mathcal{A}$ must send $m_h + x$ to the honest verifier where m_h is an MAC tag on the output wire known by $\mathcal{A}$. In other words, $\mathcal{A}$ learns x, which occurs with probability at most $1/p^r$.

In conclusion, any unbounded environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, except with probability $(q_{\mathsf{H}}+1)/p^r + 1/2^\kappa$.

Corrupted verifier. $\mathcal{S}$ maintains a hash list L_h for the random oracle H. On any query $y \in \{0,1\}^*$ to H, if $\exists(y, h_y) \in L_h$, return h_y, else return a uniformly random $h_y \in \{0,1\}^\kappa$ and add (y, h_y) to L_h. If $\mathcal{S}$ receives false from $\mathcal{F}_{\mathsf{ZK}}$, then is simply aborts. Otherwise, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows,

1) Upon receiving $\mathsf{init}, x \in \mathbb{F}_{p^r}$ and $\{k_{u_{\tau_i}}\}_{i\in[n+|\mathcal{G}|]}$ that $\mathcal{V}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, $\mathcal{S}$ stores those values. $\mathcal{S}$ stores $\{k_{y_j}\}_{j\in\mathcal{G}}$ that $\mathcal{V}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
2) $\mathcal{S}$ sends $\{d_{\tau_i}\}_{\tau_i\in\mathcal{I}_{\mathsf{in}}} \xleftarrow{R} \mathbb{F}_p^{|\mathcal{I}_{\mathsf{in}}|}$, and sends $d_{v_j} \xleftarrow{R} \mathbb{F}_p$ for every $j \in \mathcal{G}$ to $\mathcal{V}$.
3) $\mathcal{S}$ computes $\{k_{j,\mathsf{zero}}\}_{j\in\mathcal{G}}$ (based on the keys stored before and sent to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ by $\mathcal{V}$). If there not exists $((k_{1,\mathsf{zero}}, \cdots, k_{|\mathcal{G}|,\mathsf{zero}}), h_y) \in L_h$, $\mathcal{S}$ sends a uniformly random $A \in \{0,1\}^\kappa$ to $\mathcal{V}$ and add $((k_{1,\mathsf{zero}}, \cdots, k_{|\mathcal{G}|,\mathsf{zero}}), A)$ to L_h. Otherwise, $\mathcal{S}$ sends $A = h_y$ to $\mathcal{V}$.
4) $\mathcal{S}$ computes k_h (based on the keys stored before and sent to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ by $\mathcal{V}$) and then sets $m_h = k_h - x$, where h is the single output wire. Then $\mathcal{S}$ sends m_h to $\mathcal{A}$.

Since $\{u_{\tau_i}\}_{i\in[n+|\mathcal{G}|]}$ and A are uniformly random and $\{u_{\tau_i}\}_{i\in[n+|\mathcal{G}|]}$ perfectly hidden against the view of adversary $\mathcal{A}$, we easily obtain that the view of $\mathcal{A}$ simulated by $\mathcal{S}$ is distributed identically to its view in the real protocol execution. This completes the proof.

B.2. Proof of Theorem 3

Proof. We first consider the case of a malicious prover (i.e., soundness and knowledge extraction) and then consider the case of a malicious verifier (i.e., zero knowledge). In each case, we construct a simulator $\mathcal{S}$ could only access to an ideal functionality $\mathcal{F}_{\mathsf{ZK}}$, and run the adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ for $\mathcal{A}$. We always implicitly assume that $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

Corrupted prover. $\mathcal{S}$ interacts with $\mathcal{A}$ as follows,

1) $\mathcal{S}$ receives init that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, samples $x \xleftarrow{R} \mathbb{F}_{p^r}$, and stores $\{u_{\tau_i}, m_{u_{\tau_i}}\}_{i\in[n+|\mathcal{G}_1|]}$ that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$. Then $\mathcal{S}$ compute the corresponding keys $\{k_{u_{\tau_i}}\}_{i\in[n+|\mathcal{G}_1|]}$. $\mathcal{S}$ stores M_0, M_1 that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, and computes the corresponding K. Upon receiving $\{m_{y_j}\}_{j\in\mathcal{G}_0}$ that $\mathcal{A}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$, $\mathcal{S}$ computes and stores $y_j = f_j(\{u_i\}_{i\in\mathcal{I}_j}) - g_{j,0}$ for every $j \in \mathcal{G}_0$, and corresponding keys $\{k_{y_j}\}_{j\in\mathcal{G}_0}$.
2) For $\tau_i \in \mathcal{I}_{\mathsf{in}}$, upon receiving d_{τ_i} from $\mathcal{A}$, $\mathcal{S}$ computes and stores $w_{\tau_i} = u_{\tau_i} + d_{\tau_i}$.
3) $\mathcal{S}$ executes the rest of the protocol as an honest verifier, using x and the keys stored in the first step. If the honest verifier outputs false, then $\mathcal{S}$ sends $\boldsymbol{w} = \perp$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}$ and aborts. If the honest verifier outputs true, then $\mathcal{S}$ sends $\boldsymbol{w}$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}$ where $\boldsymbol{w} = (w_1, \cdots, w_n)$ stored in the second step.

The adversary $\mathcal{A}$'s views have identical distributions in both the ideal and real-world executions. Whenever the verifier in the real-world execution outputs false, the verifier in

the ideal-world execution also outputs false, since $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{\mathsf{ZK}}$ in this case. Therefore, it remains to bound the probability that the verifier in the real-world execution outputs true while the witness $\boldsymbol{w}$ sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{ZK}}$ satisfies $\mathcal{C}(\boldsymbol{w}) = 0$. Then the probability that the honest verifier in the real-world execution outputs true is at most $(3 + |\mathcal{G}_1|)/p^r$.

It is trivial that values associated with the input wires and the output wires of the gate j are computed correctly where $j \in \mathcal{G}_0$. Therefore, the required probability is at most the sum of the probabilities of the following two cases.

Case 1: When $\mathcal{C}(\boldsymbol{w}) = 0$ and there exists some $j \in \mathcal{G}_1$ the the MAC and keys of the output wire value of gate j are not correctly computed from the MAC and keys of the input wires during step 2, which means that $[w_{v_j}] = [f_j(\{w_{i'}\}_{i' \in \mathcal{I}_j})] + e_{v_j}$ where $e_{v_j} \in \mathbb{F}_p$ is an error introduced by the adversary $\mathcal{A}$ by sending an incorrect value $d'_{v_j} = d_{v_j} + e_{v_j}$, $\mathcal{V}$ pass the check in step 4 which means that $w = u_0 + u_1 \cdot x$ where u_0, u_1 sent by $\mathcal{A}$.

For the gate j, which is j'-th gate in $\mathcal{G}_1$, two parties hold $\{[w_i]\}_{i \in \mathcal{I}_j}, [w'_{v_j}]$ where $[w'_{v_j}] = [w_{v_j}] + e_{v_j} = [f_j(\{w_i\}_{i \in \mathcal{I}_j})] + e_{v_j}$. Therefore, we have $k_{w'_{v_j}} = m_{w_{v_j}} + w_{v_j} \cdot x + e_{v_j} \cdot x$. Further we have b_j equals

$$
\begin{aligned}
&(g_{j,1}(\{k_{w_i}\}_{i \in \mathcal{I}_j}) + g_{j,0} \cdot x - k_{w'_{v_j}}) \cdot x + g_{j,2}(\{k_{w_i}\}_{i \in \mathcal{I}_j}) \\
=&g_{j,2}(\{m_{w_i}\}_{i \in \mathcal{I}_j}) - e_{v_j} \cdot x^2 + (f_j(\{m_{w_i} + w_i\}_{i \in \mathcal{I}_j}) \\
&- g_{j,2}(\{m_{w_i}\}_{i \in \mathcal{I}_j}) - f_j(\{w_i\}_{i \in \mathcal{I}_j}) - m_{w_{v_j}}) \cdot x \\
=&a_{0,j} + a_{1,j} \cdot x - e_{v_j} \cdot x^2
\end{aligned}
$$

In the step 4, $\mathcal{A}$ sends $u'_0 = u_0 + e_{u,0}$ and $u'_1 = u_1 + e_{u,1}$ to the honest $\mathcal{V}$ where u_0, u_1 are computed following the protocol description, $e_{u,0}, e_{u,1} \in \mathbb{F}_{p^r}$ are the adversarially chosen errors. Furthermore, we have the following:

$$
\begin{aligned}
w =& \sum_{j' \in [|\mathcal{G}_1|]} b_{j'} \cdot \chi^{j'} + K \\
=& \sum_{j' \in [|\mathcal{G}_1|]} (a_{0,j'} + a_{1,j'} \cdot x - e_{v_{j'}} \cdot x^2)\chi^{j'} + M_0 + M_1 \cdot x \\
=&(u'_0 - e_{u,0}) + (u'_1 - e_{u,1}) \cdot x - (\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'}) \cdot x^2
\end{aligned}
$$

If the check passes in step 4, then we have $w = u'_0 + u'_1 \cdot x$. Therefore, we can obtain that

$$
e_{u,0} + e_{u,1} \cdot x + (\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'}) \cdot x^2 = 0 \qquad (2)
$$

Then the probability of case 1 is less than the sum of following two probabilities,

1) $\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'} \neq 0$, the equation 2 is held. Since $x \in \mathbb{F}_{p^r}$ is uniformly random and hidden from $\mathcal{A}$'s view, the equation 2 holds with probability at most $2/p^r$.
2) $\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'} = 0$, the equation 2 is held. The probability of this case is less than the probability of $\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'} = 0$. If there exists some $j' \in [|\mathcal{G}_1|]$ such that $e_{v_{j'}} \neq 0$, the probability of $\sum_{j' \in [|\mathcal{G}_1|]} e_{v_{j'}} \cdot \chi^{j'} = 0$ is at most $|\mathcal{G}_1|/p^r$, as χ is sampled uniformly at random after all $e_{v_{j'}}$ have been determined.

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

In conclusion, any unbounded environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, except with probability $(3 + |\mathcal{G}_1|)/p^r$.

Corrupted verifier. If $\mathcal{S}$ receives false from $\mathcal{F}_{\mathsf{ZK}}$, then is simply aborts. Otherwise, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows,

1) $\mathcal{S}$ stores x, $\{[k_{u_{\tau_i}}]\}_{i \in [n+|\mathcal{G}_1|]}$, K, and $\{k_{y_j}\}_{j \in \mathcal{G}_0}$ that $\mathcal{V}$ sends to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$.
2) $\mathcal{S}$ sends $\{d_{\tau_i}\}_{\tau_i \in \mathcal{I}_{\mathsf{in}}} \xleftarrow{R} \mathbb{F}_p^n$, and sends $d_{v_j} \xleftarrow{R} \mathbb{F}_p$ for every $j \in \mathcal{G}_1$ to $\mathcal{V}$.
3) $\mathcal{S}$ samples $u_1 \xleftarrow{R} \mathbb{F}_{p^r}$ and computes $u_0 = w - u_1 \cdot x$ where w is computed using x, K and the keys received from $\mathcal{A}$ following the protocol specification. Then $\mathcal{S}$ sends u_0, u_1 to $\mathcal{A}$.
4) $\mathcal{S}$ computes k_h (based on the keys stored before and sent to $\mathcal{F}^{p,r}_{\mathsf{qsVOLE}}$ by $\mathcal{V}$) and then sets $m_h = k_h - x$, where h is the single output wire. Then $\mathcal{S}$ sends m_h to $\mathcal{A}$.

Since $\{u_{\tau_i}\}_{i \in [n+|\mathcal{G}_1|]}$ and M_1 are uniformly random and perfectly hidden against the view of adversary $\mathcal{A}$, we easily obtain that the view of $\mathcal{A}$ simulated by $\mathcal{S}$ is distributed identically to its view in the real protocol execution.