LogRobin++: Optimizing Proofs of Disjunctive Statements in VOLE-Based ZK

Carmit Hazay[1(✉)], David Heath[2], Vladimir Kolesnikov[3], Muthuramakrishnan Venkitasubramaniam[4], and Yibin Yang[3]

[1] Bar-Ilan University, Ramat Gan, Israel
Carmit.Hazay@biu.ac.il
[2] University of Illinois Urbana-Champaign, Urbana, USA
daheath@illinois.edu
[3] Georgia Institute of Technology, Atlanta, USA
{kolesnikov,yyang811}@gatech.edu
[4] Ligero Inc., Rochester, USA
muthu@ligero-inc.com

Abstract. In the Zero-Knowledge Proof (ZKP) of a *disjunctive statement*, $\mathcal{P}$ and $\mathcal{V}$ agree on B fan-in 2 circuits $\mathcal{C}_0, \ldots, \mathcal{C}_{B-1}$ over a field $\mathbb{F}$; each circuit has n_{in} inputs, $n_\times$ multiplications, and one output. $\mathcal{P}$'s goal is to demonstrate the knowledge of a witness ($id \in [B]$, $\boldsymbol{w} \in \mathbb{F}^{n_{in}}$), s.t. $\mathcal{C}_{id}(\boldsymbol{w}) = 0$ where neither $\boldsymbol{w}$ nor id is revealed. Disjunctive statements are effective, for example, in implementing ZKP based on sequential execution of CPU steps.

This paper studies ZKP (of knowledge) protocols over disjunctive statements based on Vector OLE. Denoting by λ the statistical security parameter and let $\rho \triangleq \max\{\log |\mathbb{F}|, \lambda\}$, the previous state-of-the-art protocol Robin (Yang et al. CCS'23) required $(n_{in} + 3n_\times) \log |\mathbb{F}| + \mathcal{O}(\rho B)$ bits of communication with $\mathcal{O}(1)$ rounds, and Mac′n′Cheese (Baum et al. CRYPTO'21) required $(n_{in} + n_\times) \log |\mathbb{F}| + 2n_\times \rho + \mathcal{O}(\rho \log B)$ bits of communication with $\mathcal{O}(\log B)$ rounds, both in the VOLE-hybrid model. Our novel protocol LogRobin++ achieves the same functionality at the cost of $(n_{in} + n_\times) \log |\mathbb{F}| + \mathcal{O}(\rho \log B)$ bits of communication with $\mathcal{O}(1)$ rounds in the VOLE-hybrid model. Crucially, LogRobin++ takes advantage of two new techniques – (1) an $\mathcal{O}(\log B)$-overhead approach to prove in ZK that an IT-MAC commitment vector contains a zero; and (2) the realization of VOLE-based ZK over a disjunctive statement, where $\mathcal{P}$ commits only to $\boldsymbol{w}$ and multiplication outputs of $\mathcal{C}_{id}(\boldsymbol{w})$ (as opposed to prior work where $\mathcal{P}$ commits to $\boldsymbol{w}$ and all three wires that are associated with each multiplication gate).

We implemented LogRobin++ over Boolean (i.e., $\mathbb{F}_2$) and arithmetic (i.e., $\mathbb{F}_{2^{61}-1}$) fields. In our experiments, including the cost of generating VOLE correlations, LogRobin++ achieved up to $170\times$ optimization over Robin in communication, resulting in up to $7\times$ (resp. $3\times$) wall-clock time improvements in a WAN-like (resp. LAN-like) setting.

Keywords: Zero-Knowledge · Disjunctions · VOLE-Based ZK

K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15488, pp. 367–401, 2025.
https://doi.org/10.1007/978-981-96-0935-2_12

1 Introduction

Zero-Knowledge (ZK) Proofs (ZKPs) [26] allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that some statement is true without disclosing further information. ZKPs are essential in applications such as private blockchain [8], private program analysis [22,38], private bug-bounty [31,55], privacy-preserving machine learning [35,49], and many more. In the past decade, ZKPs have received much attention, with schemes varying in performance, assumptions, and interactivity.

VOLE-Based ZK. One recent popular line of work builds ZKP protocols from *Vector Linear Oblivious Evaluation* (VOLE). This paradigm is known as *VOLE-based ZK*; see e.g. [3,4,6,19–21,34,48,51]. This thrust is facilitated by cheaply generated VOLE correlations (i.e., the random VOLE instances); see, e.g., [11–13,29,45,52]. In VOLE-based ZK, once the cryptographic task of generating VOLE correlations is complete, the remaining protocol can be (and typically is) simple, information-theoretic[1], and extremely efficient.

For a ZK statement expressed as a fan-in 2 circuit $\mathcal{C}$ over some field. Let $|\mathcal{C}|$ denote the number of gates in $\mathcal{C}$. VOLE-based ZK only requires cost (i.e., communication and computation of each party) of a small constant factor over $|\mathcal{C}|$ in terms of (extension) field elements and operations. Concretely, state-of-the-art VOLE-based ZK (e.g., QuickSilver [51]) can handle millions of (multiplication) gates per second on modest hardware and network. For this reason, VOLE-based ZK has proved useful in applications where the statement is large, e.g., privacy-preserving ML [36,49], privacy-preserving static analysis [37–39], privacy-preserving string matching [40], privacy-preserving databases [33], etc.

We focus on VOLE-based ZK because it offers by far the shortest end-to-end proof time among all ZKP approaches (e.g., zkSNARKs, MPC-in-the-Head, etc.), allowing for unprecedented scale and complexity of proven statements, such as applications mentioned above. See more discussion in Sect. 1.2.

ZK Disjunctions. Traditionally, ZKP schemes (including those based on VOLE) express statements as circuits (e.g., [1,21,51]) or constraint systems (e.g., [9,43]). In theory, these formats support arbitrary statements (including those written in a high-level language, e.g., C/C++) with polynomial overhead. On the other hand, these models discard useful program structures – particularly conditional control flow – which can be leveraged to improve efficiency. Namely, ZKP protocols that can non-trivially handle *disjunctive statements* – where one of B possible statements is proved – are highly desirable. For example, a real-world physical CPU performs a disjunction over the instruction set in each step.

In a disjunctive statement, $\mathcal{P}$ and $\mathcal{V}$ agree on B circuits $\mathcal{C}_0, \ldots, \mathcal{C}_{B-1}$. Each of these circuits is referred to as a *branch*. $\mathcal{P}$ wishes to prove her ability to evaluate one such branch to 0 without disclosing which branch is taken or *active*. The naïve strategy for handling such a disjunctive proof is to evaluate each branch separately, then use a subsequent *multiplexer* circuit to select the output of the active branch. This strategy results in a large circuit with more than $\sum_{i \in [B]} |\mathcal{C}_i|$

[1] Exceptions are the works [14,50], exploiting the additively homomorphic encryption.

Table 1. The performance of our protocol LogRobin++, compared with the prior work, in the VOLE-hybrid model (i.e., we do not account here for the cost of preprocessing random VOLEs, see Sect. 2.5 and 2.4). We consider the disjunctive statement as $\mathcal{C}_0 \vee \cdots \vee \mathcal{C}_{B-1}$ where each $\mathcal{C}_{i\in[B]}$ has n_{in} inputs, $n_\times$ multiplications and one output. We remark that *all* protocols (including prior work) support any field. For better comparison, we list the performance over two classical fields – the Boolean field and a sufficiently large arithmetic field $\mathbb{F}$ where $|\mathbb{F}| = \lambda^{\omega(1)}$. The computation is estimated by the number of (extension) field operations. $|\mathcal{C}|$ denotes the number of gates in each branch. The gray box indicates the term that only appears in $\mathcal{P}$'s computation, not $\mathcal{V}$'s.

Protocol	Field	Communication (Bits)	Rounds	Computation
Mac'n'Cheese [6]	Boolean	$n_{in} + n_\times + 2\lambda n_\times + \mathcal{O}(\lambda \log B)$	$\mathcal{O}(\log B)$	$\mathcal{O}(B\|\mathcal{C}\|)$
	Arithmetic	$(n_{in} + 3n_\times)\log\|\mathbb{F}\| + \mathcal{O}(\log B \log\|\mathbb{F}\|)$		
Robin [53]	Boolean	$n_{in} + 3n_\times + \mathcal{O}(\lambda B)$	$\mathcal{O}(1)$	$\mathcal{O}(B\|\mathcal{C}\|)$
	Arithmetic	$(n_{in} + 3n_\times)\log\|\mathbb{F}\| + \mathcal{O}(B \log\|\mathbb{F}\|)$		
LogRobin++	Boolean	$n_{in} + n_\times + \mathcal{O}(\lambda \log B)$	$\mathcal{O}(1)$	$\mathcal{O}(B\|\mathcal{C}\| + B\log B)$
	Arithmetic	$(n_{in} + n_\times)\log\|\mathbb{F}\| + \mathcal{O}(\log B \log\|\mathbb{F}\|)$		

gates. This is obviously wasteful, as only the gates in the single active branch affect the output of the overall instruction.

The study of ZKP over disjunctive statements can be traced back to the work of Cramer et al. [17]. This research problem has become very popular in recent years due to the development of the Stacked Garbling technique [30] and its natural application to efficient ZKP of statements expressed as high-level programs; see e.g. [6,23–25,30,53,54]. In this line of work, the researchers investigated custom protocols for handling general-purpose disjunctive statements, where the cost scales only with the size of a single branch. Recent work [6,53] has brought such techniques to the VOLE-based ZK setting. Combining VOLE-based ZK and disjunctive statements is natural, as disjunctions are common and useful in large and complex statements. This is the focus of our work.

1.1 Our Results

In this work, we improve the handling of disjunctive statements in the VOLE-based ZK paradigm. W.l.o.g., let the B branches (circuits over some field $\mathbb{F}$) be of equal size, with n_{in} input wires and $n_\times$ multiplication gates. Let λ be the statistical security parameter and $\rho \triangleq \max\{\log|\mathbb{F}|, \lambda\}$. Then, the state-of-the-art protocol Robin [53] requires $(n_{in} + 3n_\times)\log|\mathbb{F}| + \mathcal{O}(\rho B)$ bits of communication and $\mathcal{O}(1)$ rounds in the VOLE-hybrid model.

We propose a novel protocol LogRobin++[2] that requires only $(n_{in} + n_\times)\log|\mathbb{F}| + \mathcal{O}(\rho \log B)$ bits of communication and $\mathcal{O}(1)$ rounds in the VOLE-

[2] We note that our main protocol LogRobin++ does *not* follow the Robin's underlying paradigm or technique. We follow the Robin naming line as Robin stands for refined oblivious branching for interactive ZK [53].

hybrid model. See Table 1 for a detailed comparison with prior state-of-the-art protocols. LogRobin++ outperforms Robin in communication in two aspects: (1) its communication cost incurs an additive $\mathcal{O}(\rho \log B)$ term rather than $\mathcal{O}(\rho B)$; and (2) it saves transmission of $2n_\times$ field elements, resulting in $\approx 3\times$ improvement. To achieve these two improvements, we introduce two novel techniques:

– Inspired by [27], we propose a new technique for proving in ZK that a length-B committed vector (of IT-MAC commitments used by VOLE-based ZK) contains at least one zero element. Our technique requires transmission of only $\mathcal{O}(\log B)$ (extension) field elements. It can be directly plugged into the Robin protocol [53] to improve its communication to $(n_{in} + 3n_\times)\log|\mathbb{F}| + \mathcal{O}(\rho \log B)$ while keeping $\mathcal{O}(1)$ rounds, in the VOLE-hybrid model. We call this intermediate stepping-stone protocol LogRobin.
– We develop a new way of realizing VOLE-based ZKP of disjunctive statements. Namely, we show that with $\mathcal{P}$ committing to *only* the inputs and multiplication outputs on the active branch (using VOLE correlations), the problem of proving a disjunction reduces to the following problem of proving the existence of an *affine* correlation among a set of quadratic ones:
  $\mathcal{P}$ holds B quadratic polynomials $p_{i\in[B]}(X)$, (at least) one of which has leading coefficient 0 (i.e., it is an *affine* polynomial). $\mathcal{V}$ holds a private evaluation point Δ and obtains a commitment to each polynomial as $p_{i\in[B]}(\Delta)$.
  $\mathcal{P}$ must prove in ZK to $\mathcal{V}$ that one of p_i's is affine.
  The affine-polynomial-correlation problem can be solved using VOLE correlations.
  Put together, this reduction leads to our second stepping-stone protocol Robin++, which requires $(n_{in} + n_\times)\log|\mathbb{F}| + \mathcal{O}(\rho B)$ bits of communication and $\mathcal{O}(1)$ rounds in the VOLE-hybrid model.

Our final protocol LogRobin++, as indicated by its name, combines the underlying techniques of LogRobin and Robin++. At a high level, we show that the technical insight underlying LogRobin's optimized 0-membership proof can be adapted to solve the affine-polynomial-correlation problem exploited by Robin++. Combining our two technical ideas requires care; directly combining the two techniques would either require $\mathcal{O}(B)$ communication or break the ZK property. See Sect. 3 for a concise technical overview of our protocols.

We remark that our paradigm of constructing LogRobin can be trivially generalized beyond VOLE-based ZK. In particular, it can be instantiated based on a commit-and-prove ZK [16] where the commitment scheme is linear homomorphic (e.g., the Pedersen commitment [44]).

We implemented LogRobin++ over Boolean (i.e., $\mathbb{F}_2$) and arithmetic (i.e., $\mathbb{F}_{2^{61}-1}$) fields. The experimental results closely reflect the analytic costs in Table 1, as LogRobin++'s (and Robin's) costs contain small hidden constants in $\mathcal{O}$. Our costs include VOLE generation. Compared to prior state-of-the-art Robin [53], LogRobin++ improves communication by up to $170\times$ for disjunctions with many small branches. In terms of end-to-end execution time, LogRobin++ outperforms Robin by up to $7\times$ (resp. $3\times$) in a 10 Mbps WAN-like network (resp. 1 Gbps LAN-like network) for a wide range of parameters. See Sect. 5 for details.

We remark that LogRobin++ is secure against a static *unbounded* adversary (i.e., it is information-theoretically secure) in the VOLE-hybrid model. Somewhat surprisingly, when considering information-theoretic ZKP protocols in the VOLE-hybrid model, the price of evaluating one of many branches is now minimal in the following sense: LogRobin++ incurs only *additive* (poly)logarithmic communication as compared to the state-of-the-art (information-theoretically secure) VOLE-based ZK [21,51] over a single active branch. Thus, the additional cost of private branching is now similar to the $\log B$ bits that would be required for $\mathcal{P}$ to non-privately identify the active branch index to $\mathcal{V}$.

1.2 Related Work

VOLE-Based ZK. With the seminal work of [11] enabling cheap generation of VOLE correlations, a productive line of work on VOLE-based ZKP protocols soon emerged [3,4,6,14,19–21,29,34,48,51]. See also [5] for a survey. VOLE-based ZK is simple, information-theoretic in the VOLE-hybrid model, and efficient. Because of its efficient scaling, VOLE-based ZK is particularly useful for applications where the statement is large.

Consider a standard fan-in 2 circuit $\mathcal{C}$ defined over some field $\mathbb{F}$ with n_{in} inputs, $n_\times$ multiplications, and $|\mathcal{C}|$ gates in total. State-of-the-art (information-theoretically secure) VOLE-based ZK [21,51] incurs only linear costs with small constant factors – (1) $\mathcal{P}$ transmits $n_{in} + n_\times$ field elements and $\mathcal{O}(1)$ extension field elements, (2) $\mathcal{V}$ transmits $\mathcal{O}(1)$ extension field elements, and (3) $\mathcal{P}$ and $\mathcal{V}$ perform $\mathcal{O}(|\mathcal{C}|)$ extension field operations.

VOLE-based ZK communication cost can be further cut in half by leveraging a Random Oracle [20], or it can be reduced to sublinear by leveraging additively homomorphic encryption [50]. However, these optimizations do not substantially improve concrete performance as compared to [21,51].

VOLE-based ZK proofs are not succinct, with the exception of [50] and [14]; [50] achieves $\mathcal{O}(|\mathcal{C}|^{3/4})$ and [14] achieves $\mathcal{O}(|\mathcal{C}|^{1/2})$ communication. Constructing a VOLE-based ZK proof system incurring $o(|\mathcal{C}|^{1/2})$ communication remains an open problem.

ZK Disjunctions. The study of ZKP protocols for disjunctive statements can be traced back to 90s, starting with the work of Cramer et al. [17]. This problem was later revisited and refined by [30], which targeted improvements to ZKPs based on *Garbled Circuits* [32,56]. [30] described the possibility of reusing transmitted cryptographic material of the active branch to evaluate (to garbage and privately discard) inactive branches (they call this technique "stacking"). This limits communication cost to that of the single largest branch, but it still requires computation over all branches.

Following [30], a rich line of work [3,4,6,19–21,34,48,51] studies "stacking" ZKP protocols in the context of various ZK techniques. Among these, [6,53] are the most relevant here, as they similarly focus on VOLE-based ZK. Our protocol LogRobin++ outperforms these prior works theoretically (see Table 1) and concretely (see Sect. 5). Note, [53] also studied the *batched* disjunctions – a same disjunction is repeated. We only focus on the non-batched setting.

Proving a Committed Vector Contains 0. Our work is partially inspired by the elegant work of Groth and Kohlweiss [27]. [27] proposed a public coin special honest verifier zero-knowledge proof (i.e., a Σ-protocol) that can be used to show that a vector of cryptographic commitments (with special properties) contains a zero. [27] applies this type of proof to ring signatures and zerocoin [41]. The technique underlying our stepping-stone protocol LogRobin can be viewed as adapting their technique to the setting of IT-MAC commitments (see Sect. 2.4). We remark that we consider a malicious $\mathcal{V}$ and apply this 0-membership proof over disjunctive statements. Our final protocol LogRobin++ does *not* use a proof of 0 membership; instead, it leverages a sub-component of our LogRobin technique.

Other Related Work. ZKP is an *enormous* and fast-growing field of research. We make a few remarks about other works in the area.

Recent work [2] showed that by applying a so-called VOLE-in-the-Head cryptographic compiler, all ZK protocols relying *only* on VOLE – including ours – can be made non-interactive and publicly verifiable.

Outside VOLE-based ZK, succinct ZK proofs enjoy significant attention. Although this remarkable line of work enables incredibly small proofs and fast verification, it suffers from expensive computation on behalf of $\mathcal{P}$. This highlights a strength of VOLE-based ZK: in VOLE-based ZK, $\mathcal{P}$'s computation is lightweight and efficient.

2 Preliminaries

2.1 Notation

- λ is the statistical security parameter (e.g., 40 or 60).
- κ is the computation security parameter (e.g., 128 or 256).
- The prover is $\mathcal{P}$. We refer to $\mathcal{P}$ by she, her, hers...
- The verifier is $\mathcal{V}$. We refer to $\mathcal{V}$ by he, him, his...
- $x \triangleq y$ denotes that x is *defined* as y. $x := y$ denotes that y is *assigned* to x.
- We denote that x is uniformly drawn from a set S by $x \xleftarrow{\$} S$.
- We denote the set $\{0, \ldots, n-1\}$ by $[n]$.
- We denote a finite field of size p by $\mathbb{F}_p$ where $p \geq 2$ is a prime or a power of a prime. We use $\mathbb{F}$ to represent a sufficiently large field, i.e., $|\mathbb{F}| = \lambda^{\omega(1)}$.
- We denote row vectors by bold lower-case letters (e.g., $\boldsymbol{a}$), where a_i (or $a[i]$) denotes the i-th component of $\boldsymbol{a}$ (0-based).
- Let M be a matrix. $M_{i,j}$ is the element of i-th column and j-th row (0-based).
- We use i to index branches (e.g., $i \in [B]$), id to index the *active* branch. I.e., the id-th branch is the one that $\mathcal{P}$ holds a valid witness.

2.2 Schwartz-Zippel-DeMillo-Lipton Lemma

The soundness of our protocols heavily relies on the *Schwartz-Zippel-DeMillo-Lipton* (SZDL) lemma [18,46,57], stated in Lemma 1.

Lemma 1 (Schwartz-Zippel-DeMillo-Lipton). *Let $\mathbb{F}$ be a field and $p \in \mathbb{F}[x_1, \dots, x_n]$ be a (multivariate) polynomial of degree d. Suppose $|\mathbb{F}| > d$, then*

$$\Pr\left[p(\boldsymbol{v}) = 0 \mid \boldsymbol{v} \stackrel{\$}{\leftarrow} \mathbb{F}^n\right] \leq \frac{d}{|\mathbb{F}|}$$

2.3 Security Model

We formalize our protocol using the universally composable (UC) framework [15]. We use UC to prove security in the presence of a *malicious, static* adversary. For simplicity, we omit standard UC session (and sub-session) IDs.

2.4 IT-MACs

Information Theoretic Message Authentication Codes (IT-MACs) [10,42] are two-party (here, between $\mathcal{P}$ and $\mathcal{V}$) distributed correlated randomness that can be used as commitments. In IT-MACs over $\mathbb{F}$, $\mathcal{V}$ holds a uniformly sampled *global* key $\varDelta \stackrel{\$}{\leftarrow} \mathbb{F}$. For $\mathcal{P}$ to commit a value $x \in \mathbb{F}$, $\mathcal{V}$ samples a uniform *local* key $k_x \stackrel{\$}{\leftarrow} \mathbb{F}$ and $\mathcal{P}$ will learn a MAC for x as $m_x \triangleq k_x - x\varDelta$. We use $[x]_\varDelta \triangleq \langle (x, m_x), k_x \rangle$ to denote the IT-MAC correlation of x. $\varDelta$ will be eliminated when it is clear from the context. We recall the following useful properties of IT-MACs:

1. **Hiding:** k_x and $\varDelta$, held by $\mathcal{V}$, are independent of the committed value x.
2. **Binding:** $\mathcal{P}$ can open $[x]$ by sending x and m_x, where $\mathcal{V}$ would check if $k_x \stackrel{?}{=} x\varDelta + m_x$. To *maliciously* open $[x]$ to $x' \neq x$ (i.e., to forge x), $\mathcal{P}$ must guess $\varDelta$ – an attack would succeed with only $\frac{1}{|\mathbb{F}|}$ probability.
3. **Linear Homomorphism:** IT-MACs support linear operations – addition/scalar multiplication/constant addition – without communication. That is, for any *public* constants $c_0, c_1, \dots, c_n$ each in $\mathbb{F}$, $\mathcal{P}$ and $\mathcal{V}$ can *locally* generate $[c_0 + c_1x_1 + \cdots + c_nx_n]$ from $[x_1], \dots, [x_n]$.[3] In particular, we denote $[c_0 + c_1x_1 + \cdots + c_nx_n] = c_0 + c_1 \cdot [x_1] + \cdots + c_n \cdot [x_n]$. Note, this implies that an IT-MAC of a public constant can be generated *for free*.

2.5 VOLE Correlation

Random IT-MAC instances (over $\mathbb{F}_p$) can be generated by *Vector Oblivious Linear Evaluation* (VOLE) correlation functionality, formalized as $\mathcal{F}^{p,1}_{\mathsf{VOLE}}$ in Fig. 1. This functionality has been widely studied, e.g., in [12,13,45,48,52]. In the

[3] I.e., if $k_x = x\varDelta + m_x$ and $k_y = y\varDelta + m_y$, we have $(k_x + k_y) = (x + y)\varDelta + (m_x + m_y)$. Moreover, for any constant $c \in \mathbb{F}$, $\mathcal{P}$ can set $m_c = 0$ and $\mathcal{V}$ can set $k_c = c\varDelta$.

Functionality $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$

$\mathcal{F}_{\mathsf{VOLE}}$, parameterized by a base field $\mathbb{F}_p$ and an extension field $\mathbb{F}_{p^q}$, proceeds as follows, running with a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and an adversary $\mathcal{S}$:

Initialize. Upon receiving (`init`) from $\mathcal{P}$ and $\mathcal{V}$, if $\mathcal{V}$ is honest, sample $\Delta \xleftarrow{\$} \mathbb{F}_{p^q}$, else receive Δ from $\mathcal{S}$. Store Δ and send it to $\mathcal{V}$. Ignore subsequent (`init`).

Extend. Upon receiving ($\mathtt{extend}, n$) from $\mathcal{P}$ and $\mathcal{V}$:

- If $\mathcal{V}$ is honest, sample $\boldsymbol{k_u} \xleftarrow{\$} \mathbb{F}_{p^q}^n$, else receive $\boldsymbol{k_u} \in \mathbb{F}_{p^q}^n$ from $\mathcal{S}$.
- If $\mathcal{P}$ is honest, sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{F}_p^n$ and compute $\boldsymbol{m_u} := \boldsymbol{k_u} - \boldsymbol{u} \cdot \Delta \in \mathbb{F}_{p^q}^n$, else receive $\boldsymbol{u} \in \mathbb{F}_p^n$ and $\boldsymbol{m_u} \in \mathbb{F}_{p^q}^n$ from $\mathcal{S}$ and compute $\boldsymbol{k_u} := \boldsymbol{m_u} + \boldsymbol{u} \cdot \Delta \in \mathbb{F}_{p^q}^n$.
- Send $(\boldsymbol{u}, \boldsymbol{m_u})$ to $\mathcal{P}$ and $\boldsymbol{k_u}$ to $\mathcal{V}$.

Fig. 1. The (subfield) VOLE correlation functionality.

VOLE-based ZK, $\mathcal{P}$ and $\mathcal{V}$ generate n instances of IT-MACs, where each IT-MAC commits an independent (pseudo-)random element $u_{j\in[n]}$. Later, it is standard [7] to *consume* one random instance $[u_j]$ to generate $[x]$ where x is chosen by $\mathcal{P}$. I.e., $\mathcal{P}$ can send $x - u_j$ to allow parties to *locally* compute $[u_j] + (x - u_j) = [x]$. Note, each u_j can only be used once.

Subfield VOLE. Figure 1 also defines *subfield* VOLE correlations. This is useful when working over a small field $\mathbb{F}_p$. In particular, consider the Boolean field $\mathbb{F}_2$. Obviously, IT-MACs over $\mathbb{F}_2$ do *not* provide a strong enough binding property since $\mathcal{P}$ can successfully guess Δ with probability $\frac{1}{2}$. Naturally, we can embed values in $\mathbb{F}_2$ into a large enough extension field (i.e., $\mathbb{F}_{2^\lambda}$) to overcome this. However, since committed values are restricted to $\mathbb{F}_2$, it is an overkill to use VOLE correlations over $\mathbb{F}_{2^\lambda}$ (i.e., $\mathcal{F}_{\mathsf{VOLE}}^{2^\lambda,1}$) to generate IT-MACs. Instead, we can exploit the subfield VOLE correlation $\mathcal{F}_{\mathsf{VOLE}}^{2,\lambda}$ (also known as the *random correlated OT*) where each $u_{j\in[n]} \in \mathbb{F}_2$ – $\mathcal{P}$ sends a single bit $u_j \oplus x$ to get $[x]$.

$\mathcal{F}_{\mathsf{VOLE}}^{p,q}$ *from LPN.* Recent works (e.g., [11–13,45,52]) show that $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$ can be instantiated efficiently via the *Learning Parity with Noise* (LPN) assumption to achieve *sublinear* costs – the `extend` instruction to generate (subfield) VOLE correlations of length n requires only $o(n)$ communications.

2.6 VOLE-Based ZK for a Single Circuit and LPZK Technique [21]

Prior work [3,4,6,20,21,48–51] has shown that (subfield) VOLE correlations can be used as a hybrid functionality (see Fig. 1) to enable efficient ZK proofs.

Consider a circuit $\mathcal{C}$ defined over some field $\mathbb{F}_p$. $\mathcal{P}$ wishes to prove in ZK that she knows the inputs that evaluate $\mathcal{C}$ to zero. Let q be a large enough positive integer such that $p^q = \lambda^{\omega(1)}$. VOLE-based ZK works in the commit-and-prove paradigm [16]. In particular, by exploiting functionality $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$, $\mathcal{P}$ can commit to its inputs (i.e., the witness) and each multiplication output (i.e., the extended witness) using IT-MACs over $\mathbb{F}_{p^q}$. Recall that IT-MACs are linear homomorphic. Therefore, $\mathcal{P}$ and $\mathcal{V}$ can *locally* evaluate $\mathcal{C}$ over these IT-MACs. That is, the parties can put these IT-MAC commitments on $\mathcal{C}$'s input and each multiplication output, then evaluates $\mathcal{C}$ gate by gate over IT-MACs. After the local evaluation, $\mathcal{P}$ and $\mathcal{V}$ would obtain an IT-MAC on each wire of $\mathcal{C}$, including the output of $\mathcal{C}$ as $[res]$. Now, it suffices to show that each multiplication gate is formed correctly. That is, each multiplication gate connects to three wires (left input, right input and output) where each holds an IT-MAC; and $\mathcal{P}$ needs to show that they form a multiplication triple (inside the commitments). Note, an extra multiplication needed to be added to capture the proof to show that the output of $\mathcal{C}$ is 0, i.e., $res \cdot res = 0$ (where $[0]$ can be generated locally).

LPZK Technique. The advanced approach to proving that the multiplication relationship holds inside one IT-MAC triple is the *Line-Point Zero-Knowledge* (LPZK) technique [21,51]. Consider $[x], [y], [z]$ where $\mathcal{P}$ wants to prove in ZK that $z = xy$. The crucial observation is:

$$\overbrace{k_x k_y - k_z \Delta}^{\text{known by } \mathcal{V}} = (x\Delta + m_x)(y\Delta + m_y) - (z\Delta + m_z)\Delta \tag{1}$$

$$= \underbrace{(xy - z)}_{\text{known by } \mathcal{P}} \Delta^2 + \underbrace{(xm_y + ym_x - m_z)}_{\text{known by } \mathcal{P}} \Delta + \underbrace{m_x m_y}_{\text{known by } \mathcal{P}} \tag{2}$$

Hence, if $xy - z = 0$, $\mathcal{P}$ can send two coefficients M_1 and M_0 and $\mathcal{V}$ can check if $M_1 \Delta + M_0 \stackrel{?}{=} k_x k_y - k_z \Delta$. If $xy - z \neq 0$, the equality would only hold with $\frac{2}{p}$ probability since $\mathcal{P}$ does not know Δ. Indeed, sending $xm_y + ym_x - m_z$ breaks ZK. To recover ZK, it suffices to consume another random IT-MAC $[r]$. I.e., $\mathcal{V}$ can compute $k_x k_y - k_z \Delta + k_r$ and $\mathcal{P}$ can send $xm_y + ym_x - m_z + r$ and $m_x m_y + m_r$. The ZK holds since the coefficient is (uniformly) one-time padded.

Batched LPZK. Note that to prove a batch of multiplication IT-MAC triples, $\mathcal{V}$ can issue challenges to random linearly combine coefficients induced by each triple as Equation (1). Namely, $\mathcal{V}$ can linearly aggregate over the values known by him induced by each multiplication triple, with a $\mathcal{V}$-sampled public weight vector. Crucially, if each multiplication is formed correctly, $\mathcal{V}$ should obtain a value (after the aggregation) that can be interpreted as a $\mathcal{P}$-known affine polynomial evaluated at Δ. On the other hand, if some multiplication does not hold, $\mathcal{V}$ should w.h.p. obtain a value that can only be interpreted as a $\mathcal{P}$-known quadratic polynomial evaluated at Δ. Starting from here, the proof can be completed as the non-batched setting. We denote this procedure as the batched LPZK (check).

To further save communication, it is standard to generate the challenges (operating as the weight vector) by expanding a PRG over a κ-bit seed assuming the Random Oracle (RO) or powering an uniform field element.

By deploying the batched LPZK, the ZKP of $\mathcal{C}$ is achieved. To summarize[4]:

[4] We note that VOLE-based ZK works over any field.

Functionality $\mathcal{F}_{\mathsf{ZK}}^{p,B}$

$\mathcal{F}_{\mathsf{ZK}}^{p,B}$ is parameterized by positive integers p and B, where $\mathbb{F}_p$ exists. Upon receiving $(\texttt{prove}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1}, \boldsymbol{w}, id)$ from prover $\mathcal{P}$, where each $\mathcal{C}_{i\in[B]}$ is defined over $\mathbb{F}_p$:

- If $\mathcal{C}_{id}(\boldsymbol{w}) = 0$, then output $(\texttt{true}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1})$ to $\mathcal{V}$ and $\mathcal{S}$;
- otherwise, output $(\texttt{false}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1})$ to $\mathcal{V}$ and $\mathcal{S}$.

Fig. 2. The disjunctive ZK functionality.

Lemma 2 (Single-Circuit VOLE-based ZK, Informal). *For a circuit $\mathcal{C}$ defined over $\mathbb{F}_p$ with n_{in} inputs, $n_\times$ multiplications and one output. Let $q \in \mathbb{N}$ such that $p^q = \lambda^{\omega(1)}$. There exists a constant-round ZKP protocol over $\mathcal{C}$ with $(n_{in} + n_\times)\log p + 3q \log p + \mathcal{O}(1)$ bits of communication in $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$-hybrid model.*

Remark 1. The computation complexity of VOLE-based ZK protocol of the circuit $\mathcal{C}$ for both parties is $\mathcal{O}(|\mathcal{C}|)$ where $|\mathcal{C}|$ denotes the number of gates, in terms of field operations over $\mathbb{F}_{p^q}$ and in the VOLE-hybrid model.

2.7 Disjunctive Statements in VOLE-Based ZK: Robin [53]

Our work focuses on studying VOLE-based ZK over *disjunctive* statements. Formally, consider B circuits $\mathcal{C}_0, \ldots, \mathcal{C}_{B-1}$ defined over some field $\mathbb{F}_p$. $\mathcal{P}$'s objective is to prove to $\mathcal{V}$ that she knows an input that evaluates (at least) 1 out of these B circuits to zero, *without revealing the identity of that branch*. We use "active branch" to denote the branch for which the prover knows a witness and let it be the id-th one. Figure 2 formalizes the disjunctive ZK functionality.

A straightforward approach to handle a disjunctive statement is to combine B circuits into one large circuit, where each circuit is included, evaluated, and finally multiplexed to determine the output. This naïve approach is undesirable as the cost would be proportional to $\mathcal{O}(B|\mathcal{C}|)$, where $|\mathcal{C}|$ denotes the maximum circuit size among *all* branches. Robin [53] shows that the communication can be optimized to be proportional to $\mathcal{O}(B+|\mathcal{C}|)$. Roughly speaking, this is achieved by reusing the "multiplication triples" of the active branch on the inactive branches.

We review Robin in slightly more detail. W.l.o.g., assume B circuits are of the same size – each has the same numbers of inputs (denoted as n_{in}) and multiplications (denoted as $n_\times$). In Robin, $\mathcal{P}$ uses IT-MACs to commit to the n_{in} inputs (denoted as $[\boldsymbol{w}]$) and $3n_\times$ wires (denoted as $[\boldsymbol{\ell}], [\boldsymbol{r}], [\boldsymbol{o}]$) associated with multiplications (left/right/output) *on the active branch*. To ensure that each multiplication is formed correctly, $\mathcal{P}$ and $\mathcal{V}$ perform the batched LPZK check (see Sect. 2.6). I.e., the check ensures that $\boldsymbol{\ell}$ element-wise times $\boldsymbol{r}$ is $\boldsymbol{o}$.

Then, for each branch $\mathcal{C}_{i\in[B]}$, $\mathcal{P}$ and $\mathcal{V}$ can evaluate $\mathcal{C}_i$ over the committed inputs $[\boldsymbol{w}]$ and multiplication outputs $[\boldsymbol{o}]$, just like the regular VOLE-based ZK over $\mathcal{C}_i$ (see Sect. 2.6). Note that here $\mathcal{P}$ and $\mathcal{V}$ reuse $[\boldsymbol{w}]$ and $[\boldsymbol{o}]$ on each branch. After evaluation, each wire on $\mathcal{C}_i$ has an IT-MAC.

For each such branch $\mathcal{C}_{i\in[B]}$, denote (1) the IT-MAC vector consisting of the left wires on each multiplication as $[\boldsymbol{\ell}^{(i)}]$; (2) the IT-MAC vector consisting of the right wires on each multiplication as $[\boldsymbol{r}^{(i)}]$; and (3) the IT-MAC on the output of $\mathcal{C}_i$ as $[res^{(i)}]$. The crucial observation exploited by Robin is as follows: the committed $\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}$ are the correct extended witness for $\mathcal{C}_i$ *if and only if* the IT-MAC vector $[\boldsymbol{\ell} - \boldsymbol{\ell}^{(i)}]\|[\boldsymbol{r} - \boldsymbol{r}^{(i)}]\|[res^{(i)}]$ commits $0^{2n_\times+1}$.

Therefore, to prove that $\mathcal{P}$ indeed commits to an extended witness that satisfies one branch (conditioned on correct multiplications), it suffices to show that $0^{2n_\times+1}$ is committed by 1-out-of-B induced IT-MAC vectors. This can be proved efficiently: by $\mathcal{V}$ issuing a length-$(2n_\times + 1)$ random challenge vector[5], parties can *locally* generate B IT-MACs by computing the inner product between the random challenge and each vector. Finally, it suffices to show that one of B inner products is 0 – Robin achieves this by showing that the product of these B IT-MACs is 0, which requires transmission of $\mathcal{O}(B)$ elements in $\mathbb{F}_{p^q}$.

Note that Robin uses the LPZK technique to prove the multiplication triples of IT-MACs in a *black-box* manner. Also note that when the circuits are defined over a small field (e.g., the Boolean field $\mathbb{F}_2$), the random challenge vector issued by $\mathcal{V}$ must be defined over an extension field (e.g. $\mathbb{F}_{2^\lambda}$) to ensure soundness. We conclude this section with the following lemma and remark:

Lemma 3 (Robin, **Informal).** *Let $\mathcal{C}_{i\in[B]}$ denote B circuits (defined over $\mathbb{F}_p$) of the same size, where each has n_{in} inputs, $n_\times$ multiplications and one output. Let $q \in \mathbb{N}$ such that $p^q = \lambda^{\omega(1)}$. Then, there exists a constant-round ZKP protocol for the disjunctive statement $\mathcal{C}_0 \vee \cdots \vee \mathcal{C}_{B-1}$ using $(n_{in} + 3n_\times)\log p + \mathcal{O}(Bq \log p)$ bits of communication in $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$-hybrid model.*

Remark 2. Compared to the naïve approach, the computation complexity for Robin is still $\mathcal{O}(B|\mathcal{C}|)$ in terms of number of field operations over $\mathbb{F}_{p^q}$.

3 Technical Overview

In this section, we provide a technical overview of our constructions. We note that understanding how Robin [53] works (see Sect. 2.7 for a concise review) would be very helpful to contextualize the components in this section.

While our protocols work over any field, for simplicity, throughout this section, consider a sufficiently large field $\mathbb{F}$ (i.e., $|\mathbb{F}| = \lambda^{\omega(1)}$). In particular, $\mathcal{P}$ and $\mathcal{V}$ agree on B circuits $\mathcal{C}_{i\in[B]}$ defined over $\mathbb{F}$, each with n_{in} inputs and $n_\times$ multiplications. Suppose $\mathcal{P}$ wishes to prove to $\mathcal{V}$ in ZK that she knows $\boldsymbol{w} \in \mathbb{F}^{n_{in}}$ that can evaluate the id-th circuit to zero. Note that id, unknown to $\mathcal{V}$, must be kept *private*. Moreover, let $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}$ ($|\boldsymbol{\ell}| = |\boldsymbol{r}| = |\boldsymbol{o}| = n_\times$) denote $\mathcal{P}$'s extended witness – $\mathcal{P}$ evaluates $\mathcal{C}_{id}(\boldsymbol{w})$ to obtain $\boldsymbol{\ell}$ (resp. $\boldsymbol{r}$, $\boldsymbol{o}$), which are the values on the left (resp. right, output) wire of each multiplication, in the topology order.

Roadmap. Recall that the state-of-the-art protocol Robin requires $\mathcal{P}$ to commit to $\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}$ with additive $\mathcal{O}(B)$ communication of field elements. Our final protocol LogRobin++ achieves communication costs where $\mathcal{P}$ only needs to commit to

[5] Again, this can be generated from a PRG or an uniform element to its powers.

$\boldsymbol{w}$ and $\boldsymbol{o}$ with additive $\mathcal{O}(\log B)$ communication of field elements. Our overview is presented with stepping stones and structured as follows:

1. In Sect. 3.1, we overview our first stepping stone – a technique to allow $\mathcal{P}$ to prove to $\mathcal{V}$ in ZK that 1-out-of-B IT-MAC commitments is 0 with $\mathcal{O}(\log B)$ communication costs. Directly plugging in this technique into Robin results in a protocol – LogRobin – that requires $\mathcal{P}$ to commit to $\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}$ with additive $\mathcal{O}(\log B)$ communication of field elements.
2. In Sect. 3.2, we overview our second stepping stone – a different way to construct VOLE-based ZK for a disjunctive statement. Essentially, we show that, by $\mathcal{P}$ committing to only $\boldsymbol{w}$ and $\boldsymbol{o}$, the proof can be reduced to show the existence of an affine correlation, where $\mathcal{P}$ holds B *all-but-one-affine* quadratic polynomials and $\mathcal{V}$ holds B values that are generated by evaluating these B polynomials at $\varDelta$. We construct a sub-optimal (i.e., with $\mathcal{O}(B)$ communication costs) ZK protocol to prove the existence of such an affine correlation, ultimately resulting in a protocol – Robin++ – that requires $\mathcal{P}$ to commit to $\boldsymbol{w}, \boldsymbol{o}$ with additive $\mathcal{O}(B)$ communication of field elements.
3. In Sect. 3.3, we overview our final protocol LogRobin++, non-trivially combining techniques underlying LogRobin and Robin++. At a very high level, we show that the technique behind proving 0 among 1-out-of-B IT-MACs (used in LogRobin) can be adapted to solve the affine-polynomial-correlation problem inside Robin++ with $\mathcal{O}(\log B)$ communication costs.

3.1 LogRobin: Optimizing the Proof of IT-MACs Containing 0

In this section, we overview the first stepping-stone protocol LogRobin. Recall that the $\mathcal{O}(B)$ communication overhead in Robin comes from $\mathcal{P}$ proving $\mathcal{V}$ that there is a 0 among B IT-MACs $[t_0], \ldots, [t_{B-1}]$ (see Sect. 2.7). In Robin, this is done by simply multiplying the B values and opening the result to $\mathcal{V}$, which costs $\mathcal{O}(B)$. (If at least one multiplicand is 0, the product is 0.) The crucial technique behind LogRobin is to improve the cost of this sub-procedure to $\mathcal{O}(\log B)$.

Intuitively, this is possible as $\mathcal{P}$ knows *where the* 0 *is*, while Robin only exploits the fact that the 0 exists. Informally, $\mathcal{O}(\log B)$ can be interpreted as the *minimal* amount of information required for $\mathcal{P}$ to "point out" which element is 0 (i.e., which branch is active) *correctly and obliviously.*

A straightforward way to allow $\mathcal{P}$ to obliviously encode which branch is active (i.e., the id-th) with $\mathcal{O}(\log B)$ overhead is to require $\mathcal{P}$ to commit to id bit by bit (via IT-MACs). That is, w.l.o.g., let $B = 2^b$ for some $b \in \mathbb{N}$. Then, $\mathcal{P}$ can decompose $id \in [B]$ into b bits $id_0, \ldots, id_{b-1}$ such that $id = \sum_{i=0}^{b-1} 2^i \cdot id_i$. Next, $\mathcal{P}$ commits to each id_i as $[id_i]$ and proves in ZK that each $[id_{i \in [b]}]$ commits a bit (namely, $\mathcal{P}$ proves that $\forall i \in [B], id_i \cdot (id_i - 1) = 0$ via the batched LPZK).

Path Matrix. Committing these bits alone is insufficient. However, it turns out that they can be exploited to further generate a powerful so-called *path matrix*, inspired by [27] (a useful technique that allows $\mathcal{P}$ to obliviously point the active branch). To construct the path matrix, besides $[\boldsymbol{id}]$, $\mathcal{P}$ prepares b random IT-MACs $[\delta_0], \ldots, [\delta_{b-1}]$ where each $\delta_{i\in[b]} \xleftarrow{\$} \mathbb{F}$. Next, $\mathcal{V}$ issues a uniform challenge $\Lambda \xleftarrow{\$} \mathbb{F}$. Consider the following $2 \times b$ matrix $[\mathcal{M}]$ of IT-MACs:

$$[\mathcal{M}] = \begin{pmatrix} [\Lambda \cdot (1 - id_0) + \delta_0] & [\Lambda \cdot (1 - id_1) + \delta_1] & \cdots & [\Lambda \cdot (1 - id_{b-1}) + \delta_{b-1}] \\ [\Lambda \cdot id_0 - \delta_0] & [\Lambda \cdot id_1 - \delta_1] & \cdots & [\Lambda \cdot id_{b-1} - \delta_{b-1}] \end{pmatrix}$$

The committed matrix $\mathcal{M}$ is called the path matrix with the following properties:

- The two elements in each column differ by Λ. E.g., the two elements in the first column (within the IT-MACs) sum to $\Lambda \cdot (1 - id_0) + \delta_0 + \Lambda \cdot id_0 - \delta_0 = \Lambda$.
- Each element inside $\mathcal{M}$ can be revealed to $\mathcal{V}$ as $\delta_{i\in[b]}$ is uniform.
- For each column $i \in [b]$, if $id_i = 0$, the column vector $\mathcal{M}_i$ would be $(\Lambda + \delta_i, -\delta_i)$; if $id_i = 1$, the column vector $\mathcal{M}_i$ would be $(\delta_i, \Lambda - \delta_i)$. Essentially, Λ term *must* exist and *only* exists on the id_i-th row.

Thus, $\mathcal{P}$ can open $\mathcal{M}$ to $\mathcal{V}$ without disclosing id. Note that since Λ is public, each element of $[\mathcal{M}]$ can be *locally* generated from $[\boldsymbol{id}]$ and $[\boldsymbol{\delta}]$. With the path matrix $\mathcal{M}$, the parties can bit decompose each $a \in [B]$ into $a_0, \ldots, a_{b-1}$, then compute $\mathscr{C}_a \triangleq \prod_{i=0}^{b-1} \mathcal{M}_{i,a_i}$.

A crucial observation about each $\mathscr{C}_{a\in[B]}$ is that $\mathscr{C}_a$ is a product of b elements involving Λ only when $a = id$. I.e., $\mathscr{C}_{id}$ can be interpreted as a degree-b polynomial evaluated at point Λ. On the other hand, for each $a \neq id$, $\mathscr{C}_a$ is a polynomial of degree at most $b-1$ evaluating at Λ. The procedure to generate $\mathscr{C}$ can be viewed as $\mathcal{P}$'s ability to obliviously put the degree-b polynomial at $\mathscr{C}_{id}$.

Proving 0 *exists among IT-MACs* $[t_0], \ldots, [t_{B-1}]$. We now present how the path matrix $\mathcal{M}$ (in particular, the associated $\mathscr{C}_{a\in[B]}$) can be used to design a ZKP showing that $t_{id} = 0$ among $\left[t_{i\in[B]}\right]$ without disclosing id. Note that $\mathcal{P}$ and $\mathcal{V}$ can *locally* compute the following IT-MAC:

$$[S] \triangleq \mathscr{C}_0 \cdot [t_0] + \mathscr{C}_1 \cdot [t_1] + \cdots + \mathscr{C}_{B-1} \cdot [t_{B-1}]$$

Crucially, $\mathscr{C}_{id} \cdot [t_{id}] = [0]$ since $t_{id} = 0$. Thus, S can be interpreted as a polynomial $s(X)$ *of degree at most* $b-1$, evaluated at Λ. I.e., $s(X) \triangleq \sum_{i=0}^{b-1} s_i \cdot X^i$ such that $S = s(\Lambda)$. More importantly, the coefficients $s_0, \ldots, s_{b-1}$ of $s(X)$ are known to $\mathcal{P}$ and independent of Λ. Thus, $\mathcal{P}$ can commit to $s_0, \ldots, s_{b-1}$ as $[s_0], \ldots, [s_{b-1}]$ *before* Λ is sampled. Once Λ is public, $\mathcal{P}$ proves that

$$[S] - [s_0] - \Lambda \cdot [s_1] - \cdots - \Lambda^{b-1} \cdot [s_{b-1}] = [S - s(\Lambda)]$$

commits a 0 to finish the proof. Note that the entire procedure is taken within the IT-MACs, so the ZK holds. Moreover, it only requires $\mathcal{O}(b = \log B)$ commitments, meeting our communication budget.

We briefly argue why the soundness holds. Indeed, generating the path matrix $\mathcal{M}$ *forces* $\mathcal{P}$ to select an id to claim $t_{id} = 0$. If $t_0, \ldots, t_{B-1}$ are all non-zero, $\mathscr{C}_{id} \cdot [t_{id}]$ *must* commit a degree-b polynomial evaluated at Λ. This infers that $[S]$ commits a degree-b polynomial evaluating at Λ as well. Note that Λ is uniformly chosen by $\mathcal{V}$ and $s(X)$ is a degree-($< b$) polynomial chosen by $\mathcal{P}$ before knowing Λ. Therefore, $s(\Lambda) \neq S$ w.h.p. by the SZDL lemma (see Lemma 1).

Remark 3. To prepare $s_{i\in[b]}$, $\mathcal{P}$ needs to perform $\mathcal{O}(B\log B)$ field operations. To prepare $\mathscr{C}_{i\in[B]}$, $\mathcal{P}$ and $\mathcal{V}$ each only needs to perform $\mathcal{O}(B)$ field operations.

Remark 4. LogRobin is *constant-round* in the VOLE-hybrid model. While this is not our focus, this asymptotically improves over Mac′n′Cheese protocol [6].

3.2 Robin++: Committing to Lesser Values Within the Active Branch

In this section, we overview the second stepping-stone protocol Robin++. Robin++ improves over Robin by roughly 3× where $\mathcal{P}$ only needs to commit to $\boldsymbol{w}$ and $\boldsymbol{o}$, whereas in Robin, $\mathcal{P}$ commits to $\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}$.

It may seem that committing to $\boldsymbol{\ell}$ and $\boldsymbol{r}$ in the disjunctive setting is inherent since it allows multiplication triples on the active branch to be reused on the inactive branch (which is the secret sauce of Robin). However, this is not the case since Robin++ only allows $\mathcal{P}$ to commit to $\boldsymbol{w}$ and $\boldsymbol{o}$. To see how Robin++ works, it is instructive to see what happens if $\mathcal{P}$ commits to $\boldsymbol{w}$ and $\boldsymbol{o}$, then $\mathcal{P}$ and $\mathcal{V}$ try to execute the *single-circuit* VOLE-based ZK [21,51] (see Sect. 2.6) on each branch *reusing the committed extended witness and* $\mathcal{V}$ *'s challenges*. Ensured by the soundness of the single-circuit VOLE-based ZK, the proof on the inactive branch would fail. In particular, the proofs introduce two cases for each $\mathcal{C}_{i\in[B]}$:

- **Valid (Affine):** If $\boldsymbol{w}$ and $\boldsymbol{o}$ are the valid extended witness of $\mathcal{C}_i$, based on the correctness of the single-circuit VOLE-based ZK for $\mathcal{C}_i$, $\mathcal{P}$ will learn $M_1^{(i)}, M_0^{(i)} \in \mathbb{F}$ and $\mathcal{V}$ will learn $K^{(i)} \in \mathbb{F}$ where

$$K^{(i)} = M_1^{(i)}\Delta + M_0^{(i)}$$

  Recall that to finish the proof, $\mathcal{P}$ sends two (randomized) coefficients.
- **Invalid (Quadratic):** If $\boldsymbol{w}$ and $\boldsymbol{o}$ are *not* the valid extended witness of $\mathcal{C}_i$, based on the soundness of the single-circuit VOLE-based ZK for $\mathcal{C}_i$, $\mathcal{P}$ will learn $M_2^{(i)}, M_1^{(i)}, M_0^{(i)} \in \mathbb{F}$ and $\mathcal{V}$ will learn $K^{(i)} \in \mathbb{F}$ where

$$K^{(i)} = M_2^{(i)}\Delta^2 + M_1^{(i)}\Delta + M_0^{(i)}$$

  and crucially, $M_2^{(i)} \neq 0$ w.h.p. The proof fails by sending two coefficients.

Now, consider the disjunctive statement. Clearly, to show that there is an active branch, it is sufficient for $\mathcal{P}$ to show that there is an "affine equality/correlation". That is, instead of finishing all B proofs, $\mathcal{P}$ and $\mathcal{V}$ stop at the point where $\mathcal{V}$ holds B values $K^{(i\in[B])} \in \mathbb{F}$ where each value can be interpreted as a $\mathcal{P}$-known quadratic polynomial evaluating at Δ (i.e., $\mathcal{P}$ holds $p^{(i\in[B])}(X) \triangleq M_2^{(i)}X^2 + M_1^{(i)}X + M_0^{(i)}$ whereas $\mathcal{V}$ holds $K^{(i\in[B])} \triangleq p^{(i)}(\Delta)$ and a private Δ). Starting from here, it suffices for $\mathcal{P}$ to show in ZK that one of B evaluation points learned by $\mathcal{V}$ is introduced by an affine polynomial. I.e., the disjunctive VOLE-based ZK proof is reduced to the above affine-polynomial-correlation problem.

A Sub-optimal Approach to Solve the Affine-Polynomial-Correlation Problem. We show a sub-optimal way to solve this problem with $\mathcal{O}(B)$ costs, resulting in Robin++. In Robin++, $\mathcal{P}$ commits to all $M_2^{(i\in[B])}$ via IT-MACs as $\left[M_2^{(i\in[B])}\right]$ and proves in ZK to $\mathcal{V}$ that there is a 0 among them. This step can be done using the technique used by LogRobin or just simply showing that their product is 0 as Robin. We remark that the technique used by LogRobin will *not* improve overall communication costs here since the step to commit to all $M_2^{(i\in[B])}$ costs $\mathcal{O}(B)$.

Clearly, this is insufficient – we need to further ensure that $\mathcal{P}$ indeed commits to the correct $M_2^{(i\in[B])}$ w.r.t. each $K^{(i)}$ held by $\mathcal{V}$. In the non-private case without ZK, this can be done trivially by $\mathcal{P}$ opening $M_2^{(i)}$ for each $i \in [B]$ and sending $M_1^{(i)}$ and $M_0^{(i)}$ where $\mathcal{V}$ checks that $K^{(i)} \stackrel{?}{=} M_2^{(i)}\Delta^2 + M_1^{(i)}\Delta + M_0^{(i)}$. (Recall that Δ, sampled by $\mathcal{V}$, is private.) The ZK does *not* hold because (1) if $M_2^{(i)} = 0$, $\mathcal{V}$ would know this is the active branch, and more importantly (2) $M_{1/2}^{(i)}$ are correlated with $\mathcal{P}$'s witness. It is classic to use fresh random IT-MACs to achieve privacy by deploying them as masks. In detail, consider two random IT-MAC instances $\left[r_1^{(i)}\right], \left[r_2^{(i)}\right]$ and the following equality:

$$\begin{aligned}\overbrace{k_{r_2^{(i)}}\Delta + k_{r_1^{(i)}}}^{\text{known by } \mathcal{V}} &= \left(r_2^{(i)}\Delta + m_{r_2^{(i)}}\right)\Delta + r_1^{(i)}\Delta + m_{r_1^{(i)}} \\ &= \underbrace{r_2^{(i)}}_{\text{known by } \mathcal{P}}\Delta^2 + \underbrace{\left(r_1^{(i)} + m_{r_2^{(i)}}\right)}_{\text{known by } \mathcal{P}}\Delta + \underbrace{m_{r_1^{(i)}}}_{\text{known by } \mathcal{P}}\end{aligned}$$

Hence, $\mathcal{V}$ can compute $K^{(i)} + k_{r_2^{(i)}}\Delta + k_{r_1^{(i)}}$ where $\mathcal{P}$ would open $\left[M_2^{(i)} + r_2^{(i)}\right]$ and sends $M_1^{(i)} + r_1^{(i)} + m_{r_2^{(i)}}$ and $M_0^{(i)} + m_{r_1^{(i)}}$. ZK holds now as coefficients $M_2^{(i)}$ and $M_1^{(i)}$ each is one-time-pad encrypted. In particular, $\mathcal{V}$ would not know which branch is active since all correlations look quadratic from $\mathcal{V}$'s perspective. Note, QuickSilver [51] also showed a similar approach to generate and exploit this "padding" correlation, but they consume 3 random IT-MACs instead of 2.

Finally, note that the above check for each $i \in [B]$ is identical. Hence, all B checks can be performed in a batched manner. That is, $\mathcal{V}$ issues random challenges $\chi_0, \ldots, \chi_{B-1}$ and computes $\sum_{i=0}^{B-1}\chi_i K^{(i)}$ whereas $\mathcal{P}$ computes $\sum_{i=0}^{B-1}\chi_i M_0^{(i)}$ and $\sum_{i=0}^{B-1}\chi_i M_1^{(i)}$. Furthermore, $\mathcal{P}$ and $\mathcal{V}$ can *locally* compute $\left[\sum_{i=0}^{B-1}\chi_i M_2^{(i)}\right]$. Random masks over the coefficients are still required to ensure the ZK property, but now only two random IT-MACs are needed in total.

To conclude, our stepping-stone protocol Robin++ exploits the reduction and the sub-optimal protocol to solve the affine-polynomial-correlation problem.

Remark 5. It is worth noting that when $B = 1$, Robin++ is (almost) identical to QuickSilver [51] – the state-of-the-art VOLE-based ZK for a single circuit. In particular, the asymptotic and concrete costs are identical.

3.3 LogRobin++: Non-trivially Combining LogRobin and Robin++

In this section, we overview our final protocol LogRobin++. As its name indicates, LogRobin++ combines the techniques exploited by Robin++ and LogRobin. With both techniques, (1) $\mathcal{P}$ only needs to commit to $\boldsymbol{w}$ and $\boldsymbol{o}$ as Robin++; and (2) LogRobin++ incurs additive $\mathcal{O}(\log B)$ communication overhead as LogRobin. We remark that the combination is non-trivial as, looking ahead, a naïve attempt would either require $\mathcal{O}(B)$ costs or break the ZK property.

Recall that, by $\mathcal{P}$ committing to only $\boldsymbol{w}$ and $\boldsymbol{o}$ (cf. Robin++), the disjunctive proof can be reduced to the affine-polynomial-correlation problem. I.e., $\mathcal{P}$ and $\mathcal{V}$ jointly hold the following correlated values:

$$\overbrace{K^{(i)}}^{\text{known by } \mathcal{V}} = \overbrace{M_2^{(i)}}^{\text{known by } \mathcal{P}} \Delta^2 + \overbrace{M_1^{(i)}}^{\text{known by } \mathcal{P}} \Delta + \overbrace{M_0^{(i)}}^{\text{known by } \mathcal{P}} \tag{3}$$

for each $i \in [B]$ (where Δ is privately sampled by $\mathcal{V}$), such that $\mathcal{P}$ wishes to prove to $\mathcal{V}$ in ZK that $\exists id \in [B], M_2^{(id)} = 0$. Robin++ achieves this by requiring $\mathcal{P}$ to commit $M_2^{(i\in[B])}$ as $\left[M_2^{(0)}\right], \ldots, \left[M_2^{(B-1)}\right]$, prove the committed B containing 0, and open a random linear combination of them (with extra uniform pads to ensure ZK). Note that committing $M_2^{(i\in[B])}$ requires $\mathcal{O}(B)$ costs!

In LogRobin++, we propose a $\mathcal{O}(\log B)$-communication protocol to solve the affine-polynomial-correlation problem, ultimately achieving our objective.

Intuition. To get a sense of why this is possible, note that the correlation in Eq. (3) can be viewed as a "conceptual commitment" over $M_2^{(i)}$ (from $\mathcal{P}$ to $\mathcal{V}$). In particular, $\mathcal{P}$ can open the commitment via sending $M_0^{(i)}, M_1^{(i)}$ and $M_2^{(i)}$ whereas $\mathcal{V}$ can check if Eq. (3) holds. Indeed, as the IT-MAC, if $\mathcal{P}$ wants to forge $M_2^{(i)}$ to a different value $\widetilde{M_2^{(i)}}$, she needs to guess Δ. Viewed this way, the affine-polynomial-correlation problem can be interpreted as $\mathcal{P}$ proving to $\mathcal{V}$ in ZK that one of these B "conceptual commitments" is 0. Our technical insight behind LogRobin++ is to adapt our technique in LogRobin, which is used to prove 1 out of B IT-MAC commitments is 0, to these "conceptual commitments". However, we remark that it is not ZK to open each "conceptual commitment" – the main challenge. This is because, as discussed in Sect. 3.2, $M_0^{(i)}, M_1^{(i)}$ and $M_2^{(i)}$ correlate with $\mathcal{P}$'s extended witness.

Adapting LogRobin*'s technique over "conceptual commitments".* Recall that $\mathcal{P}$ in LogRobin would commit to *id* bit by bit, and then the parties generate a so-called path matrix $\mathcal{M}$. This path matrix $\mathcal{M}$ induces B field elements $\mathscr{C}_{i\in[B]}$. By viewing each $K^{(i\in[B])}$ *conceptually* as a commitment, $\mathcal{V}$ can compute

$$S \triangleq \mathscr{C}_0 K^{(0)} + \mathscr{C}_1 K^{(1)} + \ldots + \mathscr{C}_{B-1} K^{(B-1)} \tag{4}$$

which can be viewed as a multivariate polynomial $s(\cdot,\cdot)$ evaluated at (Λ, Δ) as

$$S = s(\Lambda, \Delta) = \sum_{j=0}^{b} \sum_{k=0}^{2} s_{j,k} \Lambda^j \Delta^k \tag{5}$$

where w.l.o.g., let $B = 2^b$ for some $b \in \mathbb{N}$. Note that the $3(b+1)$ coefficients $\{s_{j,k}\}_{j\in[b+1],k\in[3]}$ are known to $\mathcal{P}$ as they are determined by $\left\{M_2^{(i)}, M_1^{(i)}, M_0^{(i)}\right\}_{i\in[B]}$ and the $\mathcal{P}$-chosen $\boldsymbol{id}, \boldsymbol{\delta}$ (see Sect. 3.1). Recall that there is only one value within $\mathscr{C}$ – the $\mathscr{C}_{id}$ where id is the index of the active branch – that can be interpreted as a degree-b polynomial evaluated at Λ. Therefore, the coefficient $s_{b,2}$ of $\Lambda^b\Delta^2$ can only be induced by $\mathscr{C}_{id}K^{(id)}$ and, if $\mathcal{P}$ is honest, *must* be 0 as $M_2^{(id)} = 0$. In other words, for $i \neq id$, since $\mathscr{C}_i$ can only be interpreted as a degree-$< b$ polynomial evaluated at Λ, it is impossible to induce the term $\Lambda^b\Delta^2$.

Just as LogRobin, based on the SZDL lemma, it suffices for $\mathcal{P}$ to show her ability to compute S from a degree-$(b+1)$ multivariate polynomial evaluated at (Λ, Δ) by specifying $3b+2$ coefficients – all $s_{j\in[b+1],k\in[3]}$ except $s_{b,2}$, *before* Λ *is issued*. I.e., $\mathcal{P}$ provides an oracle to $\mathcal{V}$ to compute a degree-$(b+1)$ multivariate polynomial $s(X,Y)$ at (Λ, Δ) whereas $\mathcal{V}$ needs to ensure that S (computed by Eq. (4)) is equal to $s(\Lambda, \Delta)$. Note that revealing these coefficients to $\mathcal{V}$ directly would break privacy since they are correlated with the $\mathcal{P}$'s witness.

As a failed attempt, we can try to mimic LogRobin to ask $\mathcal{P}$ to commit to all coefficients as IT-MACs and later linearly evaluate the polynomial within the IT-MACs. This fails because Δ must be kept private to $\mathcal{P}$ to preserve the binding property of the IT-MAC. That is, even after Λ is chosen, $\mathcal{P}$ is still not able to operate on these committed coefficients to obtain $[s(\Lambda, \Delta)]$ without knowing Δ. In fact, S itself should not be learned by $\mathcal{P}$, since it is correlated with Δ.

Randomization over S. Instead, similar to Robin++, LogRobin++ exploits random IT-MACs correlations (generated from VOLE) to mask the coefficients. I.e., with masking, most of them can be directly revealed.

To see how it works, first consider the coefficients of $j = b$. I.e., the coefficients $s_{b,0}$ and $s_{b,1}$ (where $s_{b,2} = 0$ if $\mathcal{P}$ is honest). These two coefficients are related to the following additive term in Equation (5):

$$s_{b,1}\Lambda^b\Delta + s_{b,0}\Lambda^b$$

Consider one *fresh* VOLE correlation $[r_b]$, where $\mathcal{V}$ holds k_{r_b} and $\mathcal{P}$ holds r_b, m_{r_b} such that $k_{r_b} = r_b\Delta + m_{r_b}$. If $\mathcal{V}$ adds $k_{r_b}\Lambda^b = r_b\Lambda^b\Delta + m_{r_b}\Lambda^b$ into S (i.e., Eq. (4)), the (above) additive term induced by $\Lambda^b\Delta$ and Λ^b would become:

$$(s_{b,1} + r_b)\Lambda^b\Delta + (s_{b,0} + m_{r_b})\Lambda^b \tag{6}$$

Crucially, r_b looks (pseudo-)random to $\mathcal{V}$. Thus, $\mathcal{P}$ can directly send $s_{b,1} + r_b$ to $\mathcal{V}$. However, as we will discuss at the end of this section, $s_{b,0} + m_{r_b}$ cannot be disclosed to $\mathcal{V}$ since this would break privacy – a malicious $\mathcal{V}^*$ can learn the active index id by manipulating it. Instead, $\mathcal{P}$ will commit to $s_{b,0} + m_{r_b}$ as $[s_{b,0} + m_{r_b}]$. It will become clear soon how this IT-MAC is used.

Let us proceed to consider coefficients of $j = 0, \ldots, b-1$. I.e., the coefficients $s_{j,0}, s_{j,1}, s_{j,2}$ for each $j \in [b]$. These three coefficients are related to the following additive term in Eq. (5):

$$s_{j,2}\Lambda^j\Delta^2 + s_{j,1}\Lambda^j\Delta + s_{j,0}\Lambda^j$$

Consider two *fresh* VOLE correlations $[r_{j,2}]$ and $[r_{j,1}]$ for each $j \in [b]$, where $\mathcal{V}$ holds $k_{r_{j,2}}, k_{r_{j,1}}$ and $\mathcal{P}$ holds $r_{j,2}, r_{j,1}, m_{r_{j,2}}, m_{r_{j,1}}$ such that $k_{r_{j,2}} = r_{j,2}\Delta + m_{r_{j,2}}$ and $k_{r_{j,1}} = r_{j,1}\Delta + m_{r_{j,1}}$. Similarly, $\mathcal{V}$ can add the term $k_{r_{j,2}}\Lambda^j\Delta + k_{r_{j,1}}\Lambda^j = r_{j,2}\Lambda^j\Delta^2 + (m_{r_{j,2}} + r_{j,1})\Lambda^j\Delta + m_{r_{j,1}}\Lambda^j$ into S (i.e., Eq. (4)), the (above) additive term induced by $\Lambda^j\Delta^2$, $\Lambda^j\Delta$ and Λ^j would become:

$$(s_{j,2} + r_{j,2})\Lambda^j\Delta^2 + (s_{j,1} + m_{r_{j,2}} + r_{j,1})\Lambda^j\Delta + (s_{j,0} + m_{r_{j,1}})\Lambda^j$$

Again, $\mathcal{P}$ can directly send $s_{j,2} + r_{j,2}$ and $s_{j,1} + m_{r_{j,2}} + r_{j,1}$ as they are one-time padded by uniform $r_{j,2}$ and $r_{j,1}$. Similarly, $\mathcal{V}$ should not learn $s_{j,0} + m_{r_{j,1}}$ so $\mathcal{P}$ commits to $s_{j,0} + m_{r_{j,1}}$ as $\left[s_{j,0} + m_{r_{j,1}}\right]$. (We will explain at the end of this section why this cannot be directly disclosed to $\mathcal{V}$.)

Informally, via sending these values (i.e., $2b+1$ randomized coefficients and $b+1$ IT-MACs), $\mathcal{P}$ commits to a multivariate polynomial of degree less than $b+2$, *before knowing* Λ. In particular, they will be used as the polynomial oracle.

We are now ready to show how these coefficients inside IT-MACs are used. Naturally, they are used to let $\mathcal{V}$ evaluate the committed polynomial at (Λ, Δ). Note that $\mathcal{V}$ is missing $b+1$ coefficients $s_{0,0} + m_{r_{0,1}}, \ldots, s_{b-1,0} + m_{r_{b-1,1}}, s_{b,0} + m_{r_b}$ to evaluate the committed polynomial. However, the additive term in the committed polynomial related to these coefficients is independent of Δ (i.e., $\Delta^0 = 1$). Therefore, once Λ is public, parties can *locally* compute then open:

$$\Lambda^0 \cdot \left[s_{0,0} + m_{r_{0,1}}\right] + \cdots + \Lambda^{b-1} \cdot \left[s_{b-1,0} + m_{r_{b-1,1}}\right] + \Lambda^b \cdot \left[s_{b,0} + m_{r_b}\right] \tag{7}$$

which, together with $2b+1$ randomized coefficients, helps $\mathcal{V}$ evaluate the committed polynomial at (Λ, Δ). Finally, if the evaluation output equals the randomized S (cf. Eq. (4)), $\mathcal{V}$ accepts the proof. Indeed, the above protocol overcomes the difficulty in the failed attempt as it does not require $\mathcal{P}$ to know Δ.

We remark that the polynomial must be committed before $\mathcal{P}$ knowing Λ, which is crucial for the soundness analysis. In particular, if $\mathcal{P}$ is cheating with all $M^{(2)}_{i\in[B]}$ being non-zeros, S should be interpreted as a degree-$(b+2)$ polynomial evaluated at point (Λ, Δ), even after the randomization. Hence, it is with a negligible probability that the committed polynomial (with a degree less than $b+2$) can evaluate to the same value at (Λ, Δ), based on the SZDL lemma.

To conclude, the above technique solves the polynomial-affine-correlation problem with $\mathcal{O}(\log B)$ communication, ultimately resulting in LogRobin++.

Why Can't the Coefficients Inside the IT-MACs be Disclosed? Perhaps surprisingly, unlike other $2b+1$ (randomized) coefficients, the $b+1$ coefficients inside IT-MACs should *not* be directly disclosed to $\mathcal{V}$. Here, we justify this design choice by showing how a malicious $\mathcal{V}$ (corrupted by $\mathcal{A}$) could learn the active branch

Sub-procedure Eval-IT-MAC($\mathcal{C}, [\boldsymbol{in}], [\boldsymbol{o}]$)

Eval-IT-MAC is a *local* sub-procedure executed by $\mathcal{P}$ and $\mathcal{V}$. It takes (1) a circuit $\mathcal{C}$ with n_{in} inputs, $n_\times$ multiplications and 1 output; (2) an IT-MAC vector $[\boldsymbol{in}]$ such that $|\boldsymbol{in}| = n_{in}$; and (3) an IT-MAC vector $[\boldsymbol{o}]$ such that $|\boldsymbol{o}| = n_\times$;, then produces a vector of IT-MAC triples $\boldsymbol{t}$ where $|\boldsymbol{t}| = n_\times + 1$. Eval-IT-MAC proceeds as follows:

$\mathcal{P}$ (or $\mathcal{V}$) sets $\boldsymbol{t} = \perp$, then evaluates $\mathcal{C}$ gate-by-gate in the topology order:

1. If it is an input gate, for the j-th input, put $[in_j]$ on the wire.
2. If it is an addition gate, for the j-th addition, take the IT-MAC $[x_j]$ on the left wire and the IT-MAC $[y_j]$ on the right wire, put $[x_j] + [y_j]$ on the output wire.
3. If it is a multiplication gate, for the j-th multiplication, put $[o_j]$ on the output wire. Take the IT-MAC $[x_j]$ on the left wire and the IT-MAC $[y_j]$ on the right wire, append the IT-MAC triple $([x_j], [y_j], [o_j])$ to $\boldsymbol{t}$.

After the evaluation, take the IT-MAC $[res]$ on the output of $\mathcal{C}$, append the IT-MAC triple $([res], [res], [0])$ to $\boldsymbol{t}$. $\mathcal{P}$ (or $\mathcal{V}$) returns $\boldsymbol{t}$.

Fig. 3. Eval-IT-MAC: The sub-procedure for parties to evaluate $\mathcal{C}$ over IT-MACs. This sub-procedure is local since parties only perform additions over IT-MACs.

index if they were disclosed. Note that $\mathcal{A}$ is allowed to choose global key Δ and local keys $\boldsymbol{k}$ in the VOLE correlation functionality (see Fig. 1). Therefore, by $\mathcal{A}$ setting $\Delta = 0$, each local key k equals the corresponding MAC m held by $\mathcal{P}$. This implies that $\mathcal{A}$ knows each $M_0^{(i\in[B])}$ in Eq. (3). Similarly, $\mathcal{A}$ knows m_{r_b} where $[r_b]$ is used to randomize S (see Eq. (6)). Furthermore, according to Eq. (4), $s_{b,0}$ (i.e., the coefficient of Λ^b) is equal to $M_0^{(id)}$. Thus, if the coefficient $s_{b,0} + m_{r_b}$ is disclosed (see Eq. (6)), $\mathcal{A}$ learns $s_{b,0}$. By comparing $s_{b,0}$ with each $M_0^{i\in[B]}$, $\mathcal{A}$ can infer which $id \in [B]$ gives $M_0^{(id)} = s_{b,0}$.

4 Formalization

We UC formalize our final protocol LogRobin++. For completeness, we also formalize our stepping-stone protocols LogRobin/Robin++ in our full version [28].

4.1 Sub-procedures

In this section, we define two sub-procedures that will be used by LogRobin++ (also used by LogRobin/Robin++) as subroutines. These sub-procedures are *local*.

Eval-IT-MAC: *Evaluating IT-MACs over a Circuit* $\mathcal{C}$. The first sub-procedure allows $\mathcal{P}$ and $\mathcal{V}$ to evaluate a circuit $\mathcal{C}$ on IT-MAC commitments. The sub-procedure (called Eval-IT-MAC) is formalized in Fig. 3. Clearly, the computation complexity of this sub-procedure is $\mathcal{O}(|\mathcal{C}|)$.

Sub-procedure $\mathsf{Acc}^{\mathcal{P}}(\boldsymbol{t}, \gamma)$

$\mathsf{Acc}^{\mathcal{P}}$ is a *local* sub-procedure executed by $\mathcal{P}$. It takes (1) a vector of IT-MAC triples $\boldsymbol{t}$ where $|\boldsymbol{t}| = n$ and each triple is of form $([\cdot], [\cdot], [\cdot])$; and (2) a vector of field elements $\boldsymbol{\gamma}$ where $|\boldsymbol{\gamma}| = n$, then produces three field elements $M^{(2)}, M^{(1)}, M^{(0)}$. Here, the field is the one associated with the IT-MACs. $\mathsf{Acc}^{\mathcal{P}}$ proceeds as follows:

$\mathcal{P}$ sets $M^{(2)}, M^{(1)}, M^{(0)}$ be 0s. Then, for each $j \in [n]$, let $t_j = ([x_j], [y_j], [z_j])$, $\mathcal{P}$ updates the $M^{(2)}, M^{(1)}, M^{(0)}$ as follows:

$$M^{(2)} := M^{(2)} + \gamma_j(x_j y_j - z_j)$$
$$M^{(1)} := M^{(1)} + \gamma_j(x_j m_{y_j} + y_j m_{x_j} - m_{z_j})$$
$$M^{(0)} := M^{(0)} + \gamma_j m_{x_j} m_{y_j}$$

After the iteration, $\mathcal{P}$ returns $M^{(2)}, M^{(1)}, M^{(0)}$.

Sub-procedure $\mathsf{Acc}^{\mathcal{V}}(\boldsymbol{t}, \gamma)$

$\mathsf{Acc}^{\mathcal{V}}$ is a *local* sub-procedure executed by $\mathcal{V}$. It takes the same input format as $\mathsf{Acc}^{\mathcal{P}}$, but produces a single field element K. $\mathsf{Acc}^{\mathcal{V}}$ proceeds as follows: $\mathcal{V}$ sets K be 0s. Then, for each $j \in [n]$, let $t_j = ([x_j], [y_j], [z_j])$, $\mathcal{V}$ updates the K as follows:

$$K := K + \gamma_j(k_{x_j} k_{y_j} - k_{z_j} \Delta)$$

After the iteration, $\mathcal{V}$ returns K.

Fig. 4. $\mathsf{Acc}^{\mathcal{P}}/\mathsf{Acc}^{\mathcal{V}}$: The sub-procedures for $\mathcal{P}$ and $\mathcal{V}$ to accumulate correlations generated by IT-MAC triples. Note, if each triple in $\boldsymbol{t}$ forms a multiplication, $M^{(2)}$ is always equal to 0 *regardless* of $\boldsymbol{\gamma}$.

$\mathsf{Acc}^{\mathcal{P}}/\mathsf{Acc}^{\mathcal{V}}$: *Linearly Accumulating IT-MAC Triples.* The second sub-procedure allows $\mathcal{P}$ and $\mathcal{V}$ to accumulate linearly a sequence of IT-MAC triples into a single affine or quadratic distributed correlation in Δ. This (asymmetric) sub-procedure (called $\mathsf{Acc}^{\mathcal{P}}/\mathsf{Acc}^{\mathcal{V}}$) is formalized in Fig. 4. This sub-procedure takes a vector of IT-MAC triples $\boldsymbol{t} = (([x_j], [y_j], [z_j]))_{j \in [n]}$ where $n = |\boldsymbol{t}|$ and n coefficients $\gamma_0, \ldots, \gamma_{n-1}$ as inputs. Then, $\mathcal{P}$ accumulates $M^{(2)} := \sum_{j=0}^{n-1} \gamma_j(x_j y_j - z_j)$, $M^{(1)} := \sum_{j=0}^{n-1} \gamma_j(x_j m_{y_j} + y_j m_{x_j} - m_{z_j})$, $M^{(0)} := \sum_{j=0}^{n-1} \gamma_j m_{x_j} m_{y_j}$ and $\mathcal{V}$ accumulates $K := \sum_{j=0}^{n-1} \gamma_j(k_{x_j} k_{y_j} - k_{z_j} \Delta)$. Recall that the IT-MAC correlations ensure that $M^{(2)}\Delta^2 + M^{(1)}\Delta + M^{(0)} = K$ and, in particular, if all triples are multiplications, $M^{(2)}$ must be 0 regardless of $\boldsymbol{\gamma}$. Since $\mathcal{P}$ and $\mathcal{V}$ perform different algorithms, we split Acc into $\mathsf{Acc}^{\mathcal{P}}$ and $\mathsf{Acc}^{\mathcal{V}}$, but either $\mathsf{Acc}^{\mathcal{P}}$ or $\mathsf{Acc}^{\mathcal{V}}$ is *local* with $\mathcal{O}(n)$ computation complexity. Our protocols will only set $\boldsymbol{\gamma}$ as *public coins.*

Protocol $\Pi^{p,q}_{\mathsf{LogRobin++}}$

Inputs. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ hold B circuits $\mathcal{C}_0, \ldots, \mathcal{C}_{B-1}$ over field $\mathbb{F}_p$, where each circuit has n_{in} inputs, $n_\times$ multiplications and 1 output. $\mathcal{P}$ also holds a witness $\boldsymbol{w} \in \mathbb{F}_p^{n_{in}}$ and an integer $id \in [B]$ such that $\mathcal{C}_{id}(\boldsymbol{w}) = 0$. **Generate extended witness on $\mathcal{C}_{id}$.**

0. $\mathcal{P}$ evaluates $\mathcal{C}_{id}(\boldsymbol{w})$ and generates $\boldsymbol{o} \in \mathbb{F}_p^{n_\times}$ where $\boldsymbol{o}$ denotes the values on the output wires of each multiplication gate, in topological order.

Initialize/Preprocess.

1. $\mathcal{P}$ and $\mathcal{V}$ send (`init`) to $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$, which returns a uniform $\Delta \xleftarrow{\$} \mathbb{F}_{p^q}$ to $\mathcal{V}$.
2. $\mathcal{P}$ and $\mathcal{V}$ generate IT-MACs (over $\mathbb{F}_{p^q}$) of random values over $\mathbb{F}_p$ as $\{[\mu_j]\}_{j\in[n_{in}]}$, $\{[\rho_j]\}_{j\in[n_\times]}$ and $\{[\zeta_i]\}_{i\in[b]}$ by sending $(\mathtt{extend}, n_{in} + n_\times + b)$ to $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$.
3. $\mathcal{P}$ and $\mathcal{V}$ generate IT-MACs (over $\mathbb{F}_{p^q}$) of random values over $\mathbb{F}_{p^q}$ as $\{[\delta_i]\}_{i\in[b]}$, $[r_b]$, $\{[r_{j,2}], [r_{j,1}]\}_{j\in[b]}$ and $\{[\tau_j]\}_{j\in[b+1]}$ by sending $(\mathtt{extend}, (2+4b)q)$ to $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$ then locally combining (see [51]) them.

Commit to extended witness on $\mathcal{C}_{id}$.

4. For $j \in [n_{in}]$, $\mathcal{P}$ sends $d_j := w_j - \mu_j \in \mathbb{F}_p$, then both compute $[w_j] := [\mu_j] + d_j$.
5. For $j \in [n_\times]$, $\mathcal{P}$ sends $d_j := o_j - \rho_j \in \mathbb{F}_p$, then both compute $[o_j] := [\rho_j] + d_j$.

Evaluate committed IT-MACs on each branch and accumulate the correlations generated by each induced IT-MAC triples for this branch.

6. $\mathcal{V}$ samples a random vector $\boldsymbol{\gamma} \xleftarrow{\$} \mathbb{F}_{p^q}^{n_\times+1}$ and sends it to $\mathcal{P}$.
7. For each branch $i \in [B]$, $\mathcal{P}$ and $\mathcal{V}$ call sub-procedure $\mathsf{Eval\text{-}IT\text{-}MAC}(\mathcal{C}_i, [\boldsymbol{w}], [\boldsymbol{o}])$ (see Fig. 3), which returns a vector of IT-MAC triples $\boldsymbol{t}^{(i)}$ such that $|\boldsymbol{t}^{(i)}| = n_\times + 1$; then, $\mathcal{P}$ calls sub-procedure $\mathsf{Acc}^{\mathcal{P}}(\boldsymbol{t}^{(i)}, \boldsymbol{\gamma})$ (see Fig. 4), which returns $M_2^{(i)}, M_1^{(i)}, M_0^{(i)} \in \mathbb{F}_{p^q}$, and $\mathcal{V}$ calls sub-procedure $\mathsf{Acc}^{\mathcal{V}}(\boldsymbol{t}^{(i)}, \boldsymbol{\gamma})$(see Fig. 4), which returns $K^{(i)} \in \mathbb{F}_{p^q}$. Recall that the following equality holds:

$$\forall i \in [B], M_2^{(i)}\Delta^2 + M_1^{(i)}\Delta + M_0^{(i)} = K^{(i)}; \quad M_2^{(id)} = 0$$

Fig. 5. LogRobin++: ZKP protocol for disjunctive circuits over any field $\mathbb{F}_p$ in the $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$-hybrid (see Fig. 1) model. Proceed with Fig. 6.

4.2 LogRobin++

We formalize our protocol LogRobin++ as $\Pi^{p,q}_{\mathsf{LogRobin++}}$ in Figs. 5 and 6. We defer the reader to Sect. 3.3 for a concise technical overview of this protocol. The main security theorem associated with $\Pi^{p,q}_{\mathsf{LogRobin++}}$ is as follows:

Theorem 1 (LogRobin++). *$\Pi^{p,q}_{\mathsf{LogRobin++}}$ (Figs. 5 and 6) UC-realizes $\mathcal{F}^{p,B}_{\mathsf{ZK}}$ (Fig. 2) in the $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$-hybrid model (Fig. 1) with soundness error $\frac{B+b+7}{p^q}$ (where, w.l.o.g., let $B = 2^b$ for some $b \in \mathbb{N}$) and perfect zero-knowledge, in the presence of a static unbounded adversary.*

Protocol $\Pi^{p,q}_{\mathsf{LogRobin++}}$ (Cont.)

Commit to id bit-by-bit, $\mathcal{P}$ constructs the randomized final multivariate polynomial and declares (or commits to) its $3b+2$ coefficients.

8. $\mathcal{P}$ bit decomposes id as $\sum_{i=0}^{b-1} id_i \cdot 2^i$. $\mathcal{P}$ sends $\boldsymbol{id} - \boldsymbol{\zeta}$ to construct $[\boldsymbol{id}]$ from $[\boldsymbol{\zeta}]$.
9. $\mathcal{P}$ and $\mathcal{V}$ execute (batched) LPZK to prove $id_i \cdot (id_i - 1) = 0$ for each $i \in [b]$.
10. $\mathcal{P}$ constructs the following $2 \times b$ matrix, consisting of affine polynomials in X

$$\mathcal{M}(X) = \begin{pmatrix} X \cdot (1 - id_0) + \delta_0 & \cdots & X \cdot (1 - id_{b-1}) + \delta_{b-1} \\ X \cdot id_0 - \delta_0 & \cdots & X \cdot id_{b-1} - \delta_{b-1} \end{pmatrix}$$

11. $\mathcal{P}$ constructs the multivariate polynomial in X, Y

$$s(X,Y) = \sum_{a=0}^{B-1} \left(\left(M_2^{(a)} Y^2 + M_1^{(a)} Y + M_0^{(a)} \right) \cdot \prod_{i=0}^{b-1} \mathcal{M}_{i,a_i}(X) \right)$$

where $a_{i \in [b]}$ is the bit-decomposed a, i.e., $a = \sum_{i=0}^{b-1} a_i \cdot 2^i$. Since $M_2^{(id)} = 0$, $s(X)$ is a degree-($< b+2$) multivariate polynomial.

12. $\mathcal{P}$ randomizes $s(X,Y)$: for $[r_b]$, $\mathcal{P}$ holds r_b, m_{r_b} and computes $s(X,Y) := s(X,Y) + (r_b Y + m_{r_b}) X^b$. Then, for each $j \in [b]$, for $[r_{j,2}]$ and $[r_{j,1}]$, $\mathcal{P}$ holds $r_{j,2}, r_{j,1}, m_{r_{j,2}}, m_{r_{j,1}}$ and computes

$$s(X,Y) := s(X,Y) + \left(r_{j,2} Y^2 + (r_{j,1} + m_{r_{j,2}}) Y + m_{r_{j,1}} \right) X^j$$

After the randomization, let $s(X,Y) = \sum_{j=0}^{b} \sum_{k=0}^{2} s_{j,k} X^j Y^k$ where each $s_{j,k} \in \mathbb{F}_{p^q}$. In particular, if $\mathcal{P}$ is honset, $s_{b,2} = 0$.

13. $\mathcal{P}$ sends $s_{b,1}$ and for each $j \in [b]$, $\mathcal{P}$ sends $s_{j,2}$ and $s_{j,1}$.
14. For each $j \in [b+1]$, $\mathcal{P}$ sends $d_j := s_{j,0} - \tau_j \in \mathbb{F}_{p^q}$ then parties construct $[s_{j,0}]$.

Evaluate the randomized multivariate polynomial at random point (Λ, Δ).

15. $\mathcal{V}$ samples a random element $\Lambda \xleftarrow{\$} \mathbb{F}_{p^q}$ and sends it to $\mathcal{P}$.
16. $\mathcal{P}$ and $\mathcal{V}$ can locally generate IT-MAC matrix $[\mathcal{M}(\Lambda)]$ from $[\boldsymbol{id}]$ and $[\boldsymbol{\delta}]$. Then, $\mathcal{P}$ opens each IT-MAC in the second row of $[\mathcal{M}(\Lambda)]$, resulting $\mathcal{P}$ and $\mathcal{V}$ hold

$$\mathcal{M}(\Lambda) = \begin{pmatrix} \Lambda \cdot (1 - id_0) + \delta_0 & \cdots & \Lambda \cdot (1 - id_{b-1}) + \delta_{b-1} \\ \Lambda \cdot id_0 - \delta_0 & \cdots & \Lambda \cdot id_{b-1} - \delta_{b-1} \end{pmatrix} \in \mathbb{F}_{p^q}^{2 \times b}$$

17. $\mathcal{V}$ computes

$$S := \sum_{a=0}^{B-1} \left(K^{(a)} \cdot \prod_{i=0}^{b-1} \mathcal{M}_{i,a_i}(\Lambda) \right)$$

where $a_{i \in [b]}$ is the bit-decomposed a, i.e., $a = \sum_{i=0}^{b} a_i \cdot 2^i$.

18. $\mathcal{V}$ adds the randomization to S: for $[r_b]$, $\mathcal{V}$ holds k_{r_b} and computes $S := S + k_{r_b} \Lambda^b$. Then, for each $j \in [b]$, for $[r_{j,2}]$ and $[r_{j,1}]$, $\mathcal{V}$ holds $k_{r_{j,2}}, k_{r_{j,1}}$ and computes $S := S + (r_{j,2} \Delta + r_{j,1}) \Lambda^j$.
19. $\mathcal{P}$ and $\mathcal{V}$ locally construct then open the IT-MAC $[S'] = \sum_{j=0}^{b} \Lambda^j \cdot [s_{j,0}]$.
20. $\mathcal{V}$ computes $S' := S' + s_{b,1} \Lambda^b \Delta$. Then, for each $j \in [b]$, $\mathcal{V}$ computes $S' := S' + s_{j,2} \Lambda^j \Delta^2 + s_{j,1} \Lambda^j \Delta$.
21. If $S = S'$, $\mathcal{V}$ outputs $(\texttt{true}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1})$. If not (or some prior proof/open fails), $\mathcal{V}$ outputs $(\texttt{false}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1})$.

Fig. 6. LogRobin++ (Continued): ZKP protocol for disjunctive circuits over any field $\mathbb{F}_p$ in the $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$-hybrid (see Fig. 1) model.

Proof. The proof is performed by constructing the simulator $\mathcal{S}$. We need to show **completeness** (trivial, omitted); **soundness** (constructing $\mathcal{S}$ for $\mathcal{P}^*$); and **Zero-Knowledge** (constructing $\mathcal{S}$ for $\mathcal{V}^*$).

Zero-Knowledge, $\mathcal{S}$ **for** $\mathcal{V}^*$: The $\mathcal{S}$ for $\mathcal{V}^*$ is straightforward. This is because $\mathcal{V}^*$ receives either some elements that each is one-time padded by a uniform element (i.e., the VOLE correlation) or some elements that are determined by his transcripts (including his shares of IT-MACs and the global key Δ). That is, $\mathcal{S}$ will interact with $\mathcal{V}^*$ and emulate the hybrid VOLE functionality $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$ for him. Essentially, $\mathcal{S}$ proceeds as follows:

1. For Step 1, $\mathcal{S}$ samples the Δ for $\mathcal{V}^*$. Note that $\mathcal{V}^*$ can specify his own Δ *by revealing its* Δ *to* $\mathcal{S}$ (i.e., to the hybrid functionality $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$).
2. For Step 2 and 3, $\mathcal{S}$ samples the local keys (i.e., the $\mathcal{V}^*$'s IT-MAC shares of VOLE correlations) for him. Note that $\mathcal{V}^*$ can specify his own local keys *by revealing its local keys to* $\mathcal{S}$ (i.e., to the hybrid functionality $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$).
3. For Step 4 and 5, $\mathcal{S}$ samples and sends uniform elements in $\mathbb{F}_p$.
4. For Step 6, $\mathcal{S}$ receives the challenges γ from $\mathcal{V}^*$.
5. For Step 7, $\mathcal{S}$ can also execute sub-procedures $\mathsf{Eval\text{-}IT\text{-}MAC}$ and $\mathsf{Acc}^{\mathcal{V}}$ (as $\mathcal{V}$) since it has all associated values held by $\mathcal{V}^*$ – $\mathcal{S}$ has $K^{(i)}$ for each $i \in [B]$.
6. For Step 8, $\mathcal{S}$ samples and sends uniform elements in $\mathbb{F}_p$.
7. For Step 9, $\mathcal{S}$ can trivially forge the ZKP by knowing Δ and all local keys. I.e., since $\mathcal{S}$ knows all local keys and Δ, it knows what $\mathcal{V}^*$ expects as a valid proof. Suppose this value is $\Pi \in \mathbb{F}_{p^q}$. To forge the proof, $\mathcal{S}$ sends $C_1 \xleftarrow{\$} \mathbb{F}_{p^q}$ and $C_0 := \Pi - C_1\Delta$. (See also ZK $\mathcal{S}$ in LPZK [21,51].)
8. For Step 13, $\mathcal{S}$ samples and sends uniform elements in $\mathbb{F}_{p^q}$. Note that, in the real-world execution, each element sent by $\mathcal{P}$ in this step is *still* one-time padded by a uniform element in the corresponding VOLE correlation.
9. For Step 14, $\mathcal{S}$ samples and sends uniform elements in $\mathbb{F}_{p^q}$.
10. For Step 15, $\mathcal{S}$ receives the challenges Λ from $\mathcal{V}^*$.
11. For Step 16, $\mathcal{S}$ opens each IT-MAC (in the second row of $[\mathcal{M}(\Lambda)]$) to a uniform sample in $\mathbb{F}_{p^q}$. This is possible since $\mathcal{S}$ knows Δ and can open an IT-MAC to *any* value successfully. Now, $\mathcal{S}$ obtains a "path matrix" $\widetilde{\mathcal{M}}$.

12 For Step 16 and 17, $\mathcal{S}$ performs the identical computation taken by $\mathcal{V}$. This is possible since it has all associated values held by $\mathcal{V}^*$. Then, $\mathcal{S}$ obtains S.

13 For Step 18, $\mathcal{S}$ computes $\widetilde{S'} := S - s_{b,1}\Lambda^b\Delta - \sum_{j=0}^{b-1}\left(s_{j,2}\Lambda^j\Delta^2 + s_{j,1}\Lambda^j\Delta\right)$. Here, all s values are those sampled and sent by $\mathcal{S}$ for Step 13. Now, $\mathcal{S}$ opens $[S']$ to $\widetilde{S'}$. This possible because $\mathcal{S}$ knows Δ. Note that what computed by $\mathcal{S}$ is essentially the correct proof that $\mathcal{V}$ needs to see in this step. I.e., $\mathcal{V}^*$ would accept the proof since the equality in Step 21 *must* hold.

Indeed, the distributions seen by $\mathcal{V}^*$ in the ideal world and the real world are *identical*. This is because $\mathcal{S}$ replaces all one-time padded values with uniform samples (including each element in the second row of the path matrix and those coefficients sent by $\mathcal{P}$ in Step 13) and simply determines other correlated values. The simulation is *perfect*.

Soundness, $\mathcal{S}$ for $\mathcal{P}^*$: Note that $\mathcal{V}$ in $\Pi^{p,q}_{\mathsf{LogRobin++}}$ only sends uniform elements. Thus, $\mathcal{S}$, emulating $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$ for $\mathcal{P}^*$, can interact with $\mathcal{P}^*$ as an honest $\mathcal{V}$. Since $\mathcal{S}$ emulates $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$, it can trivially extract the (extended) witness $\boldsymbol{w}, \boldsymbol{o}$ used by $\mathcal{P}^*$ in Step 2 and 3. In particular, this can be done by removing the one-time pads, which are generated by $\mathcal{F}^{p,q}_{\mathsf{VOLE}}$ and known by $\mathcal{S}$. Now, if the emulated honest $\mathcal{V}$ (inside $\mathcal{S}$) outputs `false`, $\mathcal{S}$ simply sends `abort` to $\mathcal{F}^{p,B}_{\mathsf{ZK}}$, so the ideal $\mathcal{V}$ would also output `false`. Instead, if the emulated honest $\mathcal{V}$ (inside $\mathcal{S}$) outputs `true`, $\mathcal{S}$ tries and finds $id \in [B]$ such that $\mathcal{C}_{id}(\boldsymbol{w}) = 0$ (if there is no such id, just set id as 0); then, $\mathcal{S}$ sends $(\texttt{prove}, \mathcal{C}_0, \ldots, \mathcal{C}_{B-1}, \boldsymbol{w}, id)$ to $\mathcal{F}^{p,B}_{\mathsf{ZK}}$. Finally, $\mathcal{S}$ sends the UC environment $\mathcal{E}$ whatever outputted by $\mathcal{P}^*$.

We now argue why this is a valid simulator. Note that the distributions seen by $\mathcal{P}^*$ in the ideal world and the real world are *identical* (i.e., just some uniform challenges), so the distribution outputted by $\mathcal{P}^*$ in the real-world execution is the same as the distribution outputted by $\mathcal{S}$ in the ideal world. As a result, we only need to quantify the probability of the event where the ideal $\mathcal{V}$'s output is different from the real-world $\mathcal{V}$'s output. Furthermore, when the emulated honest $\mathcal{V}$ (inside $\mathcal{S}$) outputs `false`, the ideal world $\mathcal{V}$ must output `false`. Thus, we only need to quantify the probability of the event where the emulated honest $\mathcal{V}$ outputs `true` but the ideal-world $\mathcal{V}$ outputs `false`. Note that this happens when $\mathcal{P}$ uses a wrong (extended) witness (in the sense that $\boldsymbol{w}$ does not make any $\mathcal{C}_{i\in[B]}$ output 0) but still passes all checks. I.e., this is the soundness error.

This bad event would (only) happen in the following (chained) events:

- In Step 7, even though there exists (at least) one non-multiplication triple in each $\boldsymbol{t}^{(i)}$, some accumulated $M_2^{(i\in[B])}$ becomes 0. Namely, among B length-$(n_\times+1)$ vectors where none of them is all 0's, there exists (at least) 1 of them, after inner producting with the (uniformly sampled) $\boldsymbol{\gamma}$ in Step 6, results in 0. This would only happen with up to $\frac{B}{p^q}$ probability [53, Lemma 5.1].
- In Step 9, even though $\mathcal{P}^*$ commits to some id_i that is not a bit, the batched LPZK does not catch it. This would only happen with up to $\frac{3}{p^q}$ probability, i.e., the soundness error of the batched LPZK technique (where the batched check is achieved via a fresh random linear combination, cf. [51]).
- In Step 16, $\mathcal{P}^*$ forges the opening of some element(s) in $\mathcal{M}(\Lambda)$. This would only happen with up to $\frac{1}{p^q}$ based on the binding property of the IT-MAC.
- In Step 19, $\mathcal{P}^*$ forges the opening of the IT-MAC $[S']$. This would only happen with up to $\frac{1}{p^q}$ based on the binding property of the IT-MAC.
- In Step 21, $S = S'$ (accidentally) for some sampled Λ and Δ, conditioned over all previous bad events not happening. Note that if so, (Λ, Δ) must be the root of a $\mathcal{P}^*$-specified (multivariate) degree-$(b+2)$ polynomial. This is because the coefficient before $\Lambda^b\Delta^2$ must be non-zero. Thus, this would only happen with up to $\frac{b+2}{p^q}$ based on the SZDL lemma (see Lemma 1).

Hence, the union soundness error bound (i.e., the summed errors) is $\frac{B+b+7}{p^q}$.

Remark 6. Step 9 is not needed if $p = 2$ (i.e., consider Boolean circuits). This is because $\mathcal{P}$ can only commit to bits in Step 8.

Cost Analysis. We tally the computation and communication cost of LogRobin++, in the $\mathcal{F}_{\mathsf{VOLE}}^{p,q}$-hybrid model (Fig. 1). The (unidirectional) communication from $\mathcal{P}$ to $\mathcal{V}$ consists of the following components:

1. In Step 3 and 4, $\mathcal{P}$ sends $n_{in} + n_\times$ elements in $\mathbb{F}_p$ to commit to her extended witness.
2. In Step 8, $\mathcal{P}$ sends b elements in $\mathbb{F}_p$ to commit to bit-decomposed id.
3. In Step 9, $\mathcal{P}$ sends 2 elements in $\mathbb{F}_{p^q}$ for the batched LPZK check.
4. In Step 13, $\mathcal{P}$ sends $2b + 1$ elements in $\mathbb{F}_{p^q}$ as coefficients.
5. In Step 14, $\mathcal{P}$ sends $b + 1$ elements in $\mathbb{F}_{p^q}$ to commit to coefficients.
6. In Step 16, $\mathcal{P}$ sends $2b$ elements in $\mathbb{F}_{p^q}$ to open the IT-MAC commitments in the second row of the path matrix.
7. In Step 19, $\mathcal{P}$ sends 2 elements in $\mathbb{F}_{p^q}$ to open the IT-MAC commitment.

To conclude, the overall communication from $\mathcal{P}$ to $\mathcal{V}$ consists of $n_{in} + n_\times + b$ elements in $\mathbb{F}_p$ and $5b+6$ elements in $\mathbb{F}_{p^q}$. In the other direction, the communication from $\mathcal{V}$ to $\mathcal{P}$ only consists of random challenges in $\mathbb{F}_{p^q}$. Indeed, if $\mathcal{V}$ samples each challenge independently, this will result in sending $\Omega(n_\times + b)$ elements in $\mathbb{F}_{p^q}$. To further save the communication from $\mathcal{V}$ to $\mathcal{P}$, there are the following alternative approaches to generate these challenges:

- **RO variant:** It is standard to generate each sequence of uniform challenges via expanding the PRG from a uniformly chosen κ-bit seed. This optimizes the communication from $\mathcal{V}$ to $\mathcal{P}$ down to $\mathcal{O}(\kappa)$. However, this variant of Robin++ requires the Random Oracle assumption. Furthermore, the soundness error would now be bounded by $\frac{B+b+7}{p^q} + \frac{Q}{2^\kappa}$, where Q denotes the number of random oracle queries made by the adversary.
- **IT variant:** We can also generate each sequence of uniform challenges via powering a single uniform element. I.e., $\mathcal{V}$ can sample and send $\alpha \xleftarrow{\$} \mathbb{F}_{p^q}$, then parties set the challenge vector as $(1, \alpha, \alpha^2, \ldots)$. Clearly, This optimizes the communication from $\mathcal{V}$ to $\mathcal{P}$ down to $\mathcal{O}(q \log p)$, which can be set as $\mathcal{O}(\lambda)$. While this modification preserves the information-theoretic security, the soundness error would increase because of a larger probability of creating undesirable "zeros". E.g., in Step 7, even though a malicious $\mathcal{P}^*$ uses an invalid extended witness that does not evaluate any branch circuit to 0, the probability that one $M_2^{(i\in[B])}$ becomes 0 would now be $\frac{Bn_\times}{p^q}$. (This is because a malicious $\mathcal{P}^*$ wins the game if γ happens to be a root of one out of B degree-$n_\times$ polynomials.) After adjusting these bounds, the overall soundness error would now be bounded by $\frac{Bn_\times+2b+4}{p^q}$.

For computation, clearly, $\mathcal{P}$'s cost is dominated by $\mathcal{O}(B|\mathcal{C}|)$ field operations over $\mathbb{F}_{p^q}$ in Step 7 and $\mathcal{O}(B \log B)$ field operations over $\mathbb{F}_{p^q}$ in Step 11 to compute the coefficients of $s(X, Y)$; and $\mathcal{V}$'s cost is dominated by $\mathcal{O}(B|\mathcal{C}|)$ field operations over $\mathbb{F}_{p^q}$ in Step 7 only. Note that Step 17 only requires $\mathcal{O}(B)$ field operations.

Remark 7. The cost listed in Table 1 is based on the IT variant of LogRobin++.

5 Implementation and Benchmark

5.1 Setup

Implementation. We implemented LogRobin++ based on the open-source Robin repository [53], whose VOLE correlation functionality is implemented via the EMP Toolkit [47]. We instantiated our protocols over (1) the Boolean field $\mathbb{F}_2$ with $\lambda \geq 100$ and (2) the arithmetic field $\mathbb{F}_{2^{61}-1}$ with $\lambda \geq 40$, using the corresponding (subfield) VOLE functionality. For completeness, we also implemented our stepping-stone protocols LogRobin/Robin++. These simpler protocols can be useful for certain parameters.

Baseline. We use Robin [53] as our baseline. We did not compare our implementations with Mac′n′Cheese [6], as their implementation is not publicly available. However, Robin concretely outperforms Mac′n′Cheese [6]; see [53, Figure 7].

Code availability. Our implementation is publicly available at https://github.com/gconeice/logrobinplus.

Hardware and Network Settings. Our experiments were executed over two AWS EC2 `m5.xlarge` machines[6] that respectively instantiated $\mathcal{P}$ and $\mathcal{V}$. Each party ran single-threaded. (Our protocols can support multi-threading naturally by handling each branch in parallel; we leave research and implementation of parallelism as valuable future work.) We configured different network bandwidth settings, varying from a WAN-like 10 Mbps connection to a LAN-like 1 Gbps connection, via the Linux `tc` command.

Benchmark. We tested our implementations on statements where each branch (represented as a circuit) is chosen randomly. To reduce the physical memory needed to load all branches when B is large, we consider B *identical* randomly generated circuits. We performed experiments to show that the performance difference between executing B different circuits and B identical circuits is negligible; see Sect. 5.4. This choice of benchmark is just a proof of concept. One can always save different circuits in files and load them as needed, or programmatically generate large circuits from constant-sized descriptions as e.g. EMP Toolkit. All considered protocols only need to process each circuit once, so there is no need to load each circuit into main memory twice.

RO v.s. IT. Recall that our $\mathcal{V}$ must flip public coins. We implemented two variants of each protocol, depending on how coin flips are handled (see discussion in Sect. 4). Coins are flipped either by (1) expanding PRGs over several κ-bit seeds chosen by $\mathcal{V}$, requiring a Random Oracle (RO), or (2) having $\mathcal{V}$ uniformly sample $\mathcal{O}(1)$ elements, which is information-theoretic (IT). Our results show that the performance difference between these two variants is negligible; see Sect. 5.5. In the remainder of this section, we flip coins via RO.

[6] Intel Xeon Platinum 8175 CPU @ 3.10 GHz, 4 vCPUs, 16 GiB Memory.

Table 2. Experiment Results with $B = 2^{22}, n_{in} = 10, n_\times = 100$. The time reflects the wall-clock (or end-to-end) execution time from $\mathcal{P}$ starting the proof until $\mathcal{V}$ accepting it. The improvements are computed as the ratio of the corresponding data between our protocols and the baseline Robin – the larger, the better.

Field	Protocol	Comm.				LAN (1 Gbps)		WAN (10 Mbps)	
		$\mathcal{P} \to \mathcal{V}$	$\mathcal{V} \to \mathcal{P}$	Total	Impr.	Time(s)	Impr.	Time(s)	Impr.
$\mathbb{F}_2$	Robin	64 MB	28 MB	92 MB		51.2		114.1	
	LogRobin	9 KB	540 KB	549 KB	172×	15.1	3.4×	14.8	7.7×
	Robin++	128 MB	56 MB	184 MB	0.5×	94.6	0.5×	212.2	0.5×
	LogRobin++	10 KB	540 KB	550 KB	172×	16.4	3.1×	16.1	7.1×
$\mathbb{F}_{2^{61}-1}$	Robin	32 MB	2 MB	34 MB		25.8		54.3	
	LogRobin	0.8 MB	1.7 MB	2.5 MB	13.6×	27.0	1.0×	28.6	1.9×
	Robin++	64 MB	2 MB	66 MB	0.5×	13.8	1.9×	68.7	0.8×
	LogRobin++	0.8 MB	1.7MB	2.5 MB	13.6×	15.3	1.7×	17.3	3.1×

5.2 Overall Performance

We evaluated our approach with respect to the following parameters:

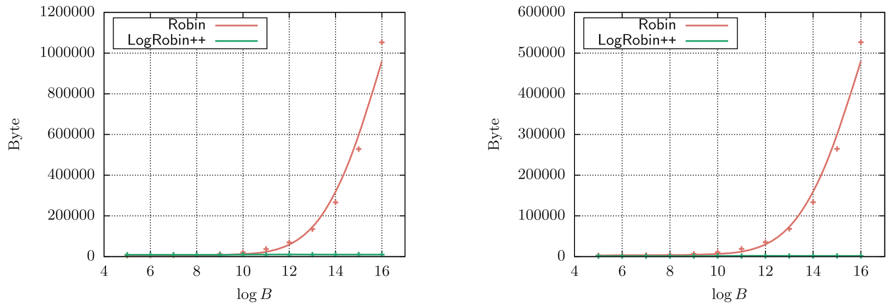
- **Benchmark "Many":** $B = 2^{22}, n_{in} = 10, n_\times = 100$: Namely, there are a large number of branches, and each branch is relatively small. In this case, LogRobin++ and LogRobin should outperform Robin++ and Robin.
- **Benchmark "Large":** $B = 2, n_{in} = 10, n_\times = 10^7$: Namely, there are a small number of branches, and each branch is large. In this case, LogRobin++ and Robin++ should outperform LogRobin and Robin.

Experimental Results with Many Branches. Table 2 tabulates experimental results for Benchmark "Many". We note the following:

1. LogRobin++ (and LogRobin) achieves a significant improvement in communication cost. This improvement leads to reduced wall-clock execution time.
2. Almost all communication from $\mathcal{V}$ to $\mathcal{P}$ is used to generate VOLE correlations. Recall, we use the VOLE implementation from the EMP-Toolkit [47]. In their implementation, each extension generates a fixed-size ($\approx 10^7$ instances) pool of VOLE correlations [52], and in some cases, we did not exhaust the entire pool (e.g., LogRobin++ and LogRobin++ in $\mathbb{F}_2$ test cases). Communication from $\mathcal{V}$ to $\mathcal{P}$ could be fine-tuned by configuring parameters in the VOLE implementation to generate a precise number of correlations.
3. Robin++ incurs 2× overhead as compared to Robin, when operating over both $\mathbb{F}_2$ and $\mathbb{F}_{2^{61}-1}$. This is because $n_\times$ is small. In Robin++, $\mathcal{P}$ must commit to an additional $\approx B$ elements, and, in this benchmark, this cost supercedes Robin++'s multiplication gate improvement.
4. In our LAN setting and when considering circuits over $\mathbb{F}_{2^{61}-1}$, LogRobin did not outperform Robin in end-to-end execution time. This LAN network is fast, so communication is not the bottleneck.

Table 3. Experimental results with $B = 2, n_{in} = 10, n_{\times} = 10^7$. The time reflects the wall-clock execution time from the moment $\mathcal{P}$ starts the proof until the moment $\mathcal{V}$ accepts it. Improvements are computed as the ratio of the corresponding data between our protocols and the baseline Robin – larger is better.

Field	Protocol	Comm.				LAN (1 Gbps)		WAN (10 Mbps)	
		$\mathcal{P} \to \mathcal{V}$	$\mathcal{V} \to \mathcal{P}$	Total	Impr.	Time(s)	Impr.	Time(s)	Impr.
$\mathbb{F}_2$	Robin	3.6 MB	1.0 MB	4.6 MB		8.1		10.1	
	LogRobin	3.6 MB	1.0 MB	4.6 MB	1.0×	8.1	1.0×	10.0	1.0×
	Robin++	1.2 MB	0.5 MB	1.7 MB	2.7×	5.3	1.5×	5.9	1.7×
	LogRobin++	1.2 MB	0.5 MB	1.7 MB	2.7×	5.4	1.5×	6.1	1.7×
$\mathbb{F}_{2^{61}-1}$	Robin	230 MB	3 MB	233 MB		11.7		205.8	
	LogRobin	230 MB	3 MB	233 MB	1.0×	11.7	1.0×	206.1	1.0×
	Robin++	77 MB	1 MB	78 MB	3.0×	6.5	1.8×	71.7	2.9×
	LogRobin++	77 MB	1 MB	78 MB	3.0×	6.4	1.8×	71.7	2.9×



(a) Boolean Field $\mathbb{F}_2$ (b) Arithmetic Field $\mathbb{F}_{2^{61}-1}$

Fig. 7. Communication of LogRobin++ vs. Robin in the VOLE-hybrid Model. We fixed $n_{in} = 10, n_{\times} = 100$ and increased $b = \log B$ from 4 to 16.

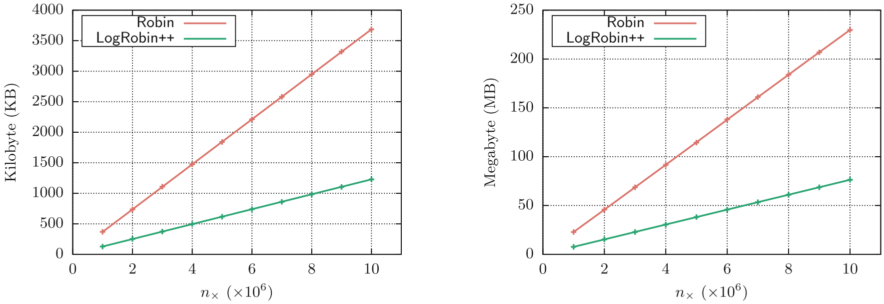
Experimental Results with Large Branches. Table 3 tabulates the experimental results for Benchmark "Large". We note the following:

1. LogRobin++ (resp. Robin++) improved communication by 3×, reflecting our analysis.
2. In our $\mathbb{F}_2$ test cases, communication was relatively small. Hence, the WAN setting was not significantly slower than the LAN setting.

Conclusion. LogRobin++ indeed combines the improvements made by LogRobin and Robin++. Clearly, it outperforms the baseline Robin and is the best choice.

5.3 Growth Trend of Communication in the VOLE-Hybrid Model

We performed experiments to show how communication grows w.r.t. (1) increasing B, and (2) increasing $|\mathcal{C}|$. To better reflect our analysis in Sect. 4, we tested



(a) Boolean Field $\mathbb{F}_2$ (b) Arithmetic Field $\mathbb{F}_{2^{61}-1}$

Fig. 8. Communication of LogRobin++ vs. Robin in the VOLE-hybrid Model. We fixed $B = 2, n_{in} = 10$ and increased $n_\times$ from 1×10^6 to 10×10^6.

Table 4. *B* Different Circuits v.s. *B* Identical Circuits in Wall-Clock Time. We set $B = 2^{10}, n_{in} = 10, n_\times = 10^5$ and considered both LAN and WAN settings.

Protocol	Time (s)			
	LAN (1 Gbps)		WAN (10 Mbps)	
	Different	Identical	Different	Identical
Robin	14.1	14.6	17.0	18.3
LogRobin	13.8	13.1	17.6	17.5
Robin++	6.7	6.6	8.8	8.3
LogRobin++	6.7	7.0	8.7	8.7

and reported the communication of LogRobin++ and Robin *without* counting communication used to generate random VOLE correlations.

Communication as a Function of B. We fixed $n_{in} = 10$ and $n_\times = 100$ and then tested LogRobin++ and Robin with $b = \log B$ ranging 5-16, in both the Boolean and arithmetic settings. Figure 7 plots the results. Our plots confirm that Robin's communication grows exponentially in b while LogRobin++'s grows linearly in b.

Communication as a Function of $|\mathcal{C}|$. We fixed $B = 2$ and $n_{in} = 10$ and then tested LogRobin++ and Robin with $n_\times$ ranging 1-10 $\times 10^6$, in both the Boolean and arithmetic settings. Figure 8 plots the results. Our plots confirm that (1) both Robin's and LogRobin++'s communication grows linearly in $|\mathcal{C}|$ and (2) Robin's communication is $\approx 3\times$ that of LogRobin++'s.

5.4 *B* Identical Branches v.s. *B* Different Branches

We tested Robin/LogRobin/Robin++/LogRobin++ where B (randomly generated) circuits are identical or different on the arithmetic setting. The results are tabulated in Table 4. Obviously, the difference is negligible. Note that it is trivially true that the communication of these two branch configurations is the same.

Table 5. RO Variant v.s. IT Variant in Wall-Clock Time.

Parameters	Field	Protocol	Time (s) LAN (1 Gbps) RO	Time (s) LAN (1 Gbps) IT	Time (s) WAN (10 Mbps) RO	Time (s) WAN (10 Mbps) IT
$B = 2^{22}, n_{in} = 10, n_\times = 100$	$\mathbb{F}_2$	Robin	51.2	49.5	114.1	115.6
		LogRobin	15.1	15.8	14.8	14.7
		Robin++	94.6	93.7	212.2	211.4
		LogRobin++	16.4	15.7	16.1	15.5
	$\mathbb{F}_{2^{61}-1}$	Robin	25.8	25.2	54.3	53.3
		LogRobin	27.0	26.9	28.6	29.1
		Robin++	13.8	13.1	68.7	69.5
		LogRobin++	15.3	15.4	17.3	17.6
$B = 2, n_{in} = 10, n_\times = 10^7$	$\mathbb{F}_2$	Robin	8.1	8.2	10.1	10.2
		LogRobin	8.1	8.2	10.0	10.1
		Robin++	5.3	5.3	5.9	6.0
		LogRobin++	5.4	5.4	6.1	6.2
	$\mathbb{F}_{2^{61}-1}$	Robin	11.7	11.7	205.8	205.7
		LogRobin	11.7	11.6	206.1	205.8
		Robin++	6.5	6.4	71.7	71.7
		LogRobin++	6.4	6.4	71.7	71.8

5.5 RO Variant v.s. IT Variant

We tested Robin/LogRobin/Robin++/LogRobin++ each on both the RO and the IT variants. The results are tabulated in Table 5. Obviously, the difference between these two variants on each protocol is negligible. Note that it is trivially true that the communication of these two variants is the same.

Acknowledgments. This work is supported in part by Visa research award, Cisco research award, and NSF awards CNS-2246353, CNS-2246354, and CCF-2217070.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3134104
2. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38554-4_19

3. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z2k. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 192–211. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021).https://doi.org/10.1145/3460120.3484812
4. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 329–358. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15985-5_12
5. Baum, C., Dittmer, S., Scholl, P., Wang, X.: Sok: vector ole-based zero-knowledge protocols. Des. Codes Cryptogr. **91**(11), 3527–3561 (2023).https://doi.org/10.1007/S10623-023-01292-8
6. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84259-8_4
7. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 97–109. Springer, Berlin, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 27–31, 1995).https://doi.org/10.1007/3-540-44750-4_8
8. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press, Berkeley, CA, USA (May 18–21, 2014).https://doi.org/10.1109/SP.2014.36
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Berlin, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1_6
10. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Berlin, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011).https://doi.org/10.1007/978-3-642-20465-4_11
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018).https://doi.org/10.1145/3243734.3243868
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 291–308. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3354255
13. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-8_16
14. Bui, D., Chu, H., Couteau, G., Wang, X., Weng, C., Yang, K., Yu, Y.: An efficient ZK compiler from SIMD circuits to general circuits. Cryptology ePrint Archive, Report 2023/1610 (2023), https://eprint.iacr.org/2023/1610

15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001).https://doi.org/10.1109/SFCS.2001.959888
16. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC. pp. 494–503. ACM Press, Montréal, Québec, Canada (May 19–21, 2002). https://doi.org/10.1145/509907.509980
17. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 174–187. Springer, Berlin, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1994). https://doi.org/10.1007/3-540-48658-5_19
18. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Inf. Process. Lett. **7**(4), 193–195 (1978)
19. Dittmer, S., Eldefrawy, K., Graham-Lengrand, S., Lu, S., Ostrovsky, R., Pereira, V.: Boosting the performance of high-assurance cryptography: Parallel execution and optimizing memory access in formally-verified line-point zero-knowledge. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 2098–2112. ACM Press, Copenhagen, Denmark (Nov 26–30, 2023).https://doi.org/10.1145/3576915.3616583
20. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving line-point zero knowledge: Two multiplications for the price of one. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 829–841. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). https://doi.org/10.1145/3548606.3559385
21. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-Point Zero Knowledge and Its Applications. In: Tessaro, S. (ed.) 2nd Conference on Information-Theoretic Cryptography (ITC 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 199, pp. 5:1–5:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ITC.2021.5
22. Fang, Z., Darais, D., Near, J.P., Zhang, Y.: Zero knowledge static program analysis. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2951–2967. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). https://doi.org/10.1145/3460120.3484795
23. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 458–487. Springer, Cham, Switzerland, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-07085-3_16
24. Goel, A., Hall-Andersen, M., Kaptchuk, G.: Dora: Processor expressiveness is (nearly) free in zero-knowledge for ram programs. Cryptology ePrint Archive, Paper 2023/1749 (2023), https://eprint.iacr.org/2023/1749
25. Goel, A., Hall-Andersen, M., Kaptchuk, G., Spooner, N.: Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 347–378. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023).https://doi.org/10.1007/978-3-031-30617-4_12
26. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC. pp. 291–304. ACM Press, Providence, RI, USA (May 6–8, 1985).https://doi.org/10.1145/22145.22178
27. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II.

LNCS, vol. 9057, pp. 253–280. Springer, Berlin, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46803-6_9
28. Hazay, C., Heath, D., Kolesnikov, V., Venkitasubramaniam, M., Yang, Y.: LogRobin++: Optimizing proofs of disjunctive statements in VOLE-based ZK. Cryptology ePrint Archive, Paper 2024/1427 (2024), https://eprint.iacr.org/2024/1427
29. Hazay, C., Yang, Y.: Toward malicious constant-rate 2PC via arithmetic garbling. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 401–431. Springer, Cham, Switzerland, Zurich, Switzerland (May 26–30, 2024).https://doi.org/10.1007/978-3-031-58740-5_14
30. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 569–598. Springer, Cham, Switzerland, Zagreb, Croatia (May 10–14, 2020).https://doi.org/10.1007/978-3-030-45727-3_19
31. Heath, D., Yang, Y., Devecsery, D., Kolesnikov, V.: Zero knowledge for everything and everyone: Fast ZK processor with cached ORAM for ANSI C programs. In: 2021 IEEE Symposium on Security and Privacy. pp. 1538–1556. IEEE Computer Society Press, San Francisco, CA, USA (May 24–27, 2021).https://doi.org/10.1109/SP40001.2021.00089
32. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press, Berlin, Germany (Nov 4–8, 2013). https://doi.org/10.1145/2508859.2516662
33. Li, X., Weng, C., Xu, Y., Wang, X., Rogers, J.: Zksql: Verifiable and efficient query evaluation with zero-knowledge proofs. Proceedings of the VLDB Endowment **16**(8), 1804–1816 (2023)
34. Lin, F., Xing, C., Yao, Y.: More efficient zero-knowledge protocols over $\mathbb{Z}_{2^k}$ via galois rings. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IX. LNCS, vol. 14928, pp. 424–457. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68400-5_13
35. Liu, T., Xie, X., Zhang, Y.: zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2968–2985. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021).https://doi.org/10.1145/3460120.3485379
36. Lu, T., Wang, H., Qu, W., Wang, Z., He, J., Tao, T., Chen, W., Zhang, J.: An efficient and extensible zero-knowledge proof framework for neural networks. Cryptology ePrint Archive, Paper 2024/703 (2024), https://eprint.iacr.org/2024/703
37. Luick, D., Kolesar, J.C., Antonopoulos, T., Harris, W.R., Parker, J., Piskac, R., Tromer, E., Wang, X., Luo, N.: ZKSMT: A VM for proving SMT theorems in zero knowledge. In: Balzarotti, D., Xu, W. (eds.) 33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024. USENIX Association (2024), https://www.usenix.org/conference/usenixsecurity24/presentation/luick
38. Luo, N., Antonopoulos, T., Harris, W.R., Piskac, R., Tromer, E., Wang, X.: Proving UNSAT in zero knowledge. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2203–2217. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). https://doi.org/10.1145/3548606.3559373
39. Luo, N., Judson, S., Antonopoulos, T., Piskac, R., Wang, X.: ppSAT: Towards two-party private SAT solving. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022. pp. 2983–3000. USENIX Association, Boston, MA, USA (Aug 10–12, 2022)

40. Luo, N., Weng, C., Singh, J., Tan, G., Piskac, R., Raykova, M.: Privacy-preserving regular expression matching using nondeterministic finite automata. Cryptology ePrint Archive, Paper 2023/643 (2023), https://eprint.iacr.org/2023/643
41. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed E-cash from Bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013). https://doi.org/10.1109/SP.2013.34
42. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Berlin, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_40
43. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013).https://doi.org/10.1109/SP.2013.47
44. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Berlin, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1992). https://doi.org/10.1007/3-540-46766-1_9
45. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1055–1072. ACM Press, London, UK (Nov 11–15, 2019).https://doi.org/10.1145/3319535.3363228
46. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM (JACM) **27**(4), 701–717 (1980)
47. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit (2016)
48. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy. pp. 1074–1091. IEEE Computer Society Press, San Francisco, CA, USA (May 24–27, 2021). https://doi.org/10.1109/SP40001.2021.00056
49. Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 501–518. USENIX Association (Aug 11–13, 2021)
50. Weng, C., Yang, K., Yang, Z., Xie, X., Wang, X.: AntMan: Interactive zero-knowledge proofs with sublinear communication. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2901–2914. ACM Press, Los Angeles, CA, USA (Nov 7–11, 2022). https://doi.org/10.1145/3548606.3560667
51. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press, Virtual Event, Republic of Korea (Nov 15–19, 2021). https://doi.org/10.1145/3460120.3484556
52. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1607–1626. ACM Press, Virtual Event, USA (Nov 9–13, 2020). https://doi.org/10.1145/3372297.3417276

53. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Batchman and robin: Batched and non-batched branching for interactive ZK. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 1452–1466. ACM Press, Copenhagen, Denmark (Nov 26–30, 2023). https://doi.org/10.1145/3576915.3623169
54. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Tight zk cpu: Batched zk branching with cost proportional to evaluated instruction. Cryptology ePrint Archive, Paper 2024/456 (2024), https://eprint.iacr.org/2024/456
55. Yang, Y., Heath, D., Kolesnikov, V., Devecsery, D.: EZEE: epoch parallel zero knowledge for ANSI C. In: 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022. pp. 109–123. IEEE, Genoa, Italy (2022). https://doi.org/10.1109/EuroSP53844.2022.00015
56. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986). https://doi.org/10.1109/SFCS.1986.25
57. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: International symposium on symbolic and algebraic manipulation. pp. 216–226. Springer (1979)