Batchman and Robin: Batched and Non-batched Branching for Interactive ZK

Yibin Yang* David Heath† Carmit Hazay‡ Vladimir Kolesnikov§
Muthuramakrishnan Venkitasubramaniam¶

September 26, 2024

Abstract

Vector Oblivious Linear Evaluation (VOLE) supports fast and scalable interactive Zero-Knowledge (ZK) proofs. Despite recent improvements to VOLE-based ZK, compiling proof statements to a control-flow oblivious form (e.g., a circuit) continues to lead to expensive proofs. One useful setting where this inefficiency stands out is when the statement is a disjunction of clauses $\mathcal{L}_1 \vee \cdots \vee \mathcal{L}_B$. Typically, ZK requires paying the price to handle all B branches. Prior works have shown how to avoid this price in communication, but not in computation.

Our main result, Batchman, is asymptotically and concretely efficient VOLE-based ZK for batched disjunctions, i.e. statements containing R repetitions of the same disjunction. This is crucial for, e.g., emulating CPU steps in ZK. Our prover and verifier complexity is only $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$, where $|\mathcal{C}|$ is the maximum circuit size of the B branches. Prior works' computation scales in $RB|\mathcal{C}|$.

For non-batched disjunctions, we also construct a VOLE-based ZK protocol, Robin, which is (only) communication efficient. For small fields and for statistical security parameter λ, this protocol's communication improves over the previous state of the art (Mac'n'Cheese, Baum et al., CRYPTO'21) by up to factor λ.

Our implementation outperforms prior state of the art. E.g., we achieve up to 6× improvement over Mac'n'Cheese (Boolean, single disjunction), and for arithmetic batched disjunctions our experiments show we improve over QuickSilver (Yang et al., CCS'21) by up to 70× and over AntMan (Weng et al., CCS'22) by up to 36×.

*Georgia Institute of Technology, yyang811@gatech.edu.
†University of Illinois Urbana-Champaign, daheath@illinois.edu.
‡Bar-Ilan University and Ligero Inc., Carmit.Hazay@biu.ac.il.
§Georgia Institute of Technology, kolesnikov@gatech.edu.
¶Ligero Inc., muthu@ligero-inc.com.

Contents

1 Introduction

Zero Knowledge (ZK) proofs [GMR85] allow a prover $\mathcal{P}$ to demonstrate to a verifier $\mathcal{V}$ the validity of a given statement while revealing nothing beyond the validity of the statement. The past decade has seen an explosion in the design, implementation and deployment of concretely efficient zero-knowledge proofs systems.

Large overheads of $\mathcal{P}$ and $\mathcal{V}$ remain a bottleneck in scaling zero-knowledge to very large statements. One major overhead comes from complex control flow, which, explicitly or implicitly, includes repeated evaluation of disjunctions. Examples include complex statements like 'this program has a bug' [HK20b] or even (the more complex) 'this program *does not* have a bug' [LAH$^+$22].

We focus on minimizing *total end-to-end proof time*, which includes communication and total computation by both $\mathcal{P}$ and $\mathcal{V}$.

VOLE-based ZK. Under this total end-to-end proof time metric, designated-verifier interactive ZKP is particularly appealing for its concrete efficiency. A recent line of work constructs such ZKP systems from a cryptographic primitive called *vector oblivious linear evaluation* (VOLE) [WYKW21, DIO21, BMRS21, YSWW21, WYX$^+$21, BBMH$^+$21, BBMHS22, DILO22, WYY$^+$22]. State-of-the-art VOLE-base ZK is efficient. For instance, [YSWW21] handles >7 million arithmetic gates per second.

While VOLE-based ZK is fast, its costs (communication, $\mathcal{P}$ and $\mathcal{V}$ computation) still scale linearly in the size of the proof statement.[1] It is interesting to exploit statement structure (e.g., disjunctions) to achieve further improvement with the ultimate goal of costs that grow sublinearly (with small constants) in the proof statement.

ZK proofs of disjunctions. Seeking improved performance and motivated by the structure of real-world programs, prior works [Kol18, HK20b, BMRS21, GGHAK22, GHAKS23] specifically optimized for proofs of *disjunctive* statements of the form $\mathcal{C}_1(\boldsymbol{w}) = 0 \vee \cdots \vee \mathcal{C}_B(\boldsymbol{w}) = 0$ for B different subcircuits referred to as *branches*. Their underlying techniques are often referred as *stacking*, following the notation introduced by [HK20b]. For disjunctions, because $\mathcal{P}$ only needs to demonstrate the truth of *one* branch, it is possible to design custom systems that achieve up to factor B improvement.

Our first contribution, Robin, (see Section 1.1) is a protocol that improves cost of disjunctive statements in VOLE-based ZK.

ZK proofs of batched disjunctions. We also consider proof statements that consist of a *batch* of the same disjunctive statement.[2] I.e., suppose $\mathcal{P}$ holds R distinct witnesses $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(R)}$, and she wishes to prove $\mathcal{C}_1(\boldsymbol{w}^{(j)}) = 0 \vee \cdots \vee \mathcal{C}_B(\boldsymbol{w}^{(j)}) = 0$ for *each* $\boldsymbol{w}^{(j)}$.

The crucial application of such statements is the emulation of a CPU inside ZK. Indeed, each step of a basic CPU executes one out of a possibly large set of instruction types, and this is repeated many times until the program halts. ZK systems that emulate a CPU are interesting, because they enable end users to express complex proof statements as programs written in commodity

[1]The recent proof system of [WYY$^+$22] achieved sublinear communication cost, but at the cost of asymptotically *increased* computation; see Section 1.2.

[2]Previous constructions, including our first contribution, have linear in B computation, and cannot simply be batched to achieve sublinear computation in $BR|\mathcal{C}|$.

programming languages, see e.g. [HYDK21]. More generally, a program can be compiled into a single large `forloop` over `switch` statement executing one of many (hundreds or thousands) of straight-line program segments. This is called *loop coalescing* [GF95]; loop coalescing is known to be useful for fast RAM-based ZKP [WSR+15].

Concretely efficient ZK systems (sublinear in computation and communication) for batched disjunctions have not been considered.

Our second – and most exciting – contribution, Batchman, is an interactive VOLE-based ZKP system that efficiently handles batched disjunctions. The surprising property of this proof system is that, as compared to naïve handling, it achieves not only factor B communication improvement, but also *up to factor* B *computation improvement for both* $\mathcal{P}$ *and* $\mathcal{V}$. Thus, our protocol is sublinear in the proof statement. While our total end-to-end runtime scales with the number of branches B and repetitions R, it crucially *does not* scale in the product RB. This enables CPU-emulation-based ZK operating over large and expressive instruction sets.

Batching zero-knowledge proofs has proven an important step towards in determining the feasibility of full-fledged zero-knowledge. Understanding the complexity of disjunctive statements has also been of theoretic interest and traces back to the work of Feige and Shamir [FS90] and Cramer, Damgård and Schoenmakers [CDS94] for the design of witness indistinguishable proofs and efficient Σ-protocols respectively.

Open-sourced implementation. Our code is available at https://github.com/gconeice/stacking-vole-zk.

An optimization for Batchman via recent ZK ROM [YH24]. We note that a straightforward optimization for our Batchman protocol can be achieved by directly plugging in the advanced ZK Read-Only Memory (ROM) built in [YH24]. This can improve the communication complexity of Batchman to $\mathcal{O}(B + R|\mathcal{C}|)$. In fact, we implicitly build a (worse) ZK ROM in this work. Any future optimizations to ZK ROM can easily be plugged into Batchman to achieve further improvement.

1.1 Our Contribution

As mentioned above, we construct two VOLE-based ZK protocols:

- **Single disjunctions.** Robin (Refined Oblivious Branching for INteractive zk) is a VOLE-based ZK for disjunctive statements expressed as an arithmetic circuit over an arbitrary field $\mathbb{F}$. For a disjunction with B branches, each consisting of a maximum of $|\mathcal{C}|$ (multiplication) gates, $\mathcal{P}$ and $\mathcal{V}$ each compute $\mathcal{O}(B|\mathcal{C}|)$ field operations and communicate $\mathcal{O}(B + |\mathcal{C}|)$ field elements. More precisely, $\mathcal{P}$ and $\mathcal{V}$ communicate only $\mathcal{O}(B\lceil \frac{\lambda}{\log|\mathbb{F}|}\rceil + |\mathcal{C}|)$ field elements.
- **Batched disjunctions.** Batchman extends Robin to *batches* of R proofs of the same disjunction. Here, $\mathcal{P}$ and $\mathcal{V}$ each compute $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$ field operations and communicate $\mathcal{O}(RB + R|\mathcal{C}|)$ field elements (assuming $\log|\mathbb{F}| = \Omega(\lambda)$ where λ is the statistical security parameter).

In our batched protocol, except for a one-time additive $B|\mathcal{C}|$ cost, $\mathcal{P}$'s and $\mathcal{V}$'s computation costs are *independent* of the number of disjunctions. In comparison, "flattening" out the circuit would result in computational complexity proportional to $RB|\mathcal{C}|$.

Protocol	**Prover Comp.**	**Comm.**	**Verifier Comp.**
[DIO21, YSWW21]	$\mathcal{O}(RB\|\mathcal{C}\|)$	$\mathcal{O}(RB\|\mathcal{C}\|)$	$\mathcal{O}(RB\|\mathcal{C}\|)$
[WYY$^+$22]	$\mathcal{O}(RB\|\mathcal{C}\| \log R)$	$\mathcal{O}(B\|\mathcal{C}\| + R)$	$\mathcal{O}(RB\|\mathcal{C}\| \log R)$
[BMRS21]	$\mathcal{O}(RB\|\mathcal{C}\|)$	$\mathcal{O}(R \log B + R\|\mathcal{C}\|)$	$\mathcal{O}(RB\|\mathcal{C}\|)$
our, Robin	$\mathcal{O}(RB\|\mathcal{C}\|)$	$\mathcal{O}(RB + R\|\mathcal{C}\|)$	$\mathcal{O}(RB\|\mathcal{C}\|)$
our, Batchman	$\mathcal{O}(RB + R\|\mathcal{C}\| + B\|\mathcal{C}\|)$	$\mathcal{O}(RB + R\|\mathcal{C}\|)$	$\mathcal{O}(RB + R\|\mathcal{C}\| + B\|\mathcal{C}\|)$

Table 1: Cost of recent VOLE-based ZK systems for batched disjunctions of arithmetic circuits. B denotes number of branches, $|\mathcal{C}|$ denotes branch size, and R denotes batch size.

Our protocols are concretely performant. For instance, Robin scales in branches up to $6\times$ better than Mac'n'Cheese [BMRS21] when $|\mathcal{C}| = 10^9$, and demonstrates up to $16\times$ improvement over QuickSilver [YSWW21] when $B = 100$. Batchman demonstrates up to $36\times$ improvement over AntMan [WYY$^+$22] when $B = 64$ and $R = 1024$, and up to $70\times$ improvement over QuickSilver [YSWW21] when $B = R = 400$. We provide a summary of our comparison to prior work in Table 1; see detailed comparison to prior work in Sections 1.2 and 7.

A bird's-eye view of our protocols. We remark that achieving computational cost sublinear in $RB|\mathcal{C}|$ is possible when we wish to evaluate the same disjunctive statement $R > 1$ times, if we are allowed non-black-box access to some underlying cryptographic primitive. Suppose $\mathcal{P}$ and $\mathcal{V}$ in a pre-processing step compute the hash of the description of each of the B branches under a collision-resistant hash function. Then, for each instance of the disjunction $\mathcal{P}$ includes in her witness the circuit description of the active branch and proves via a universal circuit that the circuit on some input witness returns 0 and that the hash of the circuit description belongs to the set of precomputed hashes. The complexity is $\tilde{\mathcal{O}}(B|\mathcal{C}|)$ for the first instance (to compute B hashes) and $\tilde{\mathcal{O}}(B + |C|)$ thereafter.

Such an approach is impractical due to its use of universal circuits and its non-black-box use of a hash function.

Seeking concretely efficient constructions, we restrict ourselves to black-box use of underlying primitives only. Surprisingly, in the same batched setting, we design an efficient construction that builds on the high level concept of "checking circuit hashes", but our construction achieves asymptotic complexity $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$ while making only black-box use of VOLE (and while using no other cryptographic primitives). In short, our approach shows that the "hash" of each branch can be determined by a random challenge that is chosen by $\mathcal{V}$ *after* $\mathcal{P}$ has committed to her witness. To compute and check these "hashes", each party computes simple linear combinations of field elements. See Section 3 for details.

1.2 Related Work

VOLE-based interactive zero-knowledge protocols. Consider a fan-in-2 circuit $\mathcal{C}$ with $|\mathcal{C}|$ multiplication/addition gates over some field. [DIO21] achieved 1 field element communication per multiplication gate based on a technique called Line-Point Zero Knowledge (LPZK). [YSWW21] further improved LPZK and implemented the technique. Our work handles multiplication gates by directly applying the LPZK technique, as well as [YSWW21]'s improvement for proving inner products. Our implementation (see Section 7) builds on [YSWW21]'s open source repo [WMK16].

[DILO22] improved LPZK communication to 0.5 field element per multiplication gate at the cost of increased computation and requiring random oracle. Concrete performance of [DILO22] is similar to [YSWW21]; we build on [YSWW21] for simplicity.

[WYY+22] for the first time achieved a VOLE-based ZK system with sublinear communication and achieved communication cost $\mathcal{O}(|\mathcal{C}|^{3/4})$. While the approach remains quite efficient, its performance is not strictly better than prior work, because it achieves its improved communication at the cost of computation. The technique performs polynomial interpolation, incurring factor $\mathcal{O}(\log |\mathcal{C}|)$ overhead, and it also employs relatively expensive additively homomorphic encryption. [WYY+22] consider batching, but not batched disjunctions; we compare with them in Section 7.

ZK disjunctions. A line of works [HK20b, BMRS21, GGHAK22, GHAKS23] augments ZK proofs with efficient handling for disjunctions. Consider B circuits $\mathcal{C}_{i\in[B]}$ each with the same number of inputs/multiplications, and suppose $\mathcal{P}$ holds a single witness for $\mathcal{C}_{a\in[B]}$ (the *active branch*). These works achieve communication that scales with $|\mathcal{C}|$ rather than $B|\mathcal{C}|$. Such works say that they "stack" the branches, following terminology introduced by [HK20b].

Most related to our work, [BMRS21] was the first (and the only) work to stack in the VOLE-based ZK setting. [BMRS21] implements multiplication gates with a custom protocol, and it is incompatible with the LPZK technique. Their multiplication procedure communicates 2 extra *extension* field elements per multiplication. Our protocols are compatible with LPZK. Our protocols communicate extra 2 field elements (not extension field elements) per multiplication, and our work is the first to consider batched disjunctions.

Even at a high level, our approach seems quite different from these prior approaches. In prior works, $\mathcal{P}$ proves satisfiability of each branch (thus paying computation scaling with B), but even honest $\mathcal{P}$ can "cheat" on each inactive branch. For example, [HK20b] allows $\mathcal{P}$ to learn cryptographic seeds used to garble the inactive branch circuits. Our approach instead achieves branching by leveraging (1) simple properties of VOLE correlations and (2) a random challenge from $\mathcal{V}$ to "compress" the description of each branch.

RAM-based ZK (RAM-ZK). Prior works have considered ZK statements expressed as RAM programs, e.g. [BCG+13, BFR+13, BCTV14, WSR+15, HMR15, MRS17, BCG+18, BHR+20, HK20a, HK21, BHR+21, FKL+21, DdSGOTV22, YHKD22]. These works present the exciting possibility of structuring ZK proof statements as programs written in commodity languages.

The RAM model of computation is relevant to the batched disjunctions setting. Indeed, because of constant RAM access cost in ZK [FKL+21, DdSGOTV22], for batch size $R \geq B$, RAM-ZK can be used to achieve batched disjunctions. By simply structuring the proof statement as a RAM program, loading the program (of size $B|C|$) into the RAM memory, and running the RAM-ZK, the proof will naturally terminate in time $\mathcal{O}(B|\mathcal{C}| + R|\mathcal{C}|)$.

RAM-based ZK is not competitive with our batched protocol for two reasons. First, our approach demonstrates a theoretical advantage. Suppose the batch is relatively small, i.e. $R = o(B)$. In this case, the RAM approach is less appealing, since it is necessary to load the program into memory, immediately incurring $\mathcal{O}(B|\mathcal{C}|)$ cost. At the same time, our communication cost is *independent* of the quantity $\mathcal{O}(B|\mathcal{C}|)$, and so it works well in this setting. More importantly, our approach elides the expensive machinery required for RAM emulation and is concretely performant. Indeed, our motivating application is the acceleration *of* such RAM-based proofs, so our low constant costs are crucial.

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

Upon receiving $(\texttt{prove}, \mathcal{C}_1, \ldots, \mathcal{C}_B, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_R, a_1, \ldots, a_R)$ from prover $\mathcal{P}$ and $(\texttt{verify}, \mathcal{C}_1, \ldots, \mathcal{C}_B)$ from verifier $\mathcal{V}$:

- If for all $i \in [R]$ it holds that $\mathcal{C}_{a_i}(\boldsymbol{w}_i) = 0$, then output $(\texttt{true})$ to $\mathcal{V}$; else output $(\texttt{false})$ to $\mathcal{V}$.

Figure 1: The batched disjunctive ZK functionality.

2 Preliminaries

2.1 Notation

- λ is the statistical security parameter (e.g., 40).
- κ is the computational security parameter (e.g., 128).
- 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.
- We denote that x is uniformly drawn from a set S by $x \in_{\$} S$.
- We denote $\{1, \ldots, n\}$ by $[n]$.
- We denote column vectors by bold lower-case letters (e.g., $\boldsymbol{a}$), where a_i (or $a[i]$) denotes the ith component of $\boldsymbol{a}$ (starting from 1) and $\boldsymbol{a}[i:j]$ the subvector $[a_i, \ldots, a_j]^T$. We use $\mathsf{glue}(\cdot)$ to stitch column vectors (e.g., $\mathsf{glue}(\boldsymbol{a}, \boldsymbol{b}) \triangleq [\boldsymbol{a}^T | \boldsymbol{b}^T]^T$).
- We denote matrices by bold upper-case letters (e.g., $\boldsymbol{A}$), where $\boldsymbol{A}(i)$ denotes the ith row vector of $\boldsymbol{A}$ (starting from 1) and $\boldsymbol{A}[i]$ denotes the ith column vector of $\boldsymbol{A}$ (starting from 1). $\boldsymbol{A}(i)[j]$ denotes jth value in ith row.
- We prove batches of disjunctions. We call each member of the batch a *repetition*. B denotes the number of branches and R denotes the number of repetitions.
- We use i to index branches (e.g., $i \in [B]$), j to index repetitions (e.g., $j \in [R]$), and k to index gates (e.g., $k \in [|\mathcal{C}|]$).
- 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. Extension fields are defined and denoted in the standard way.

2.2 Security Model

We formalize our protocols under the universally composable (UC) framework [Can01]. We use UC to formalize our protocols and to prove security in the presence of a *malicious*, *static* adversary. The functionality $\mathcal{F}_{\mathsf{ZK}}^{R,B}$ (C.f., Figure 1) is used to realize a zero-knowledge proof (of knowledge) for R-repetitions disjunction of B circuits. When $R = B = 1$, $\mathcal{F}_{\mathsf{ZK}}^{1,1}$ is the standard ZK functionality. When $R = 1$, $\mathcal{F}_{\mathsf{ZK}}^{1,B}$ is the ZK functionality for a single disjunction. Looking ahead, our protocol

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

Consider a base field $\mathbb{F}_p$ and an extension field $\mathbb{F}_{p^q}$. Functionality interacts with $\mathcal{P}$, $\mathcal{V}$ and the adversary $\mathcal{A}$ as follows:

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

Extend. Upon receiving (`extend`, n) from $\mathcal{P}$ and $\mathcal{V}$, do the following:

- If $\mathcal{V}$ is honest, sample $\boldsymbol{v} \in_\$ \mathbb{F}_{p^q}^n$, else receive $\boldsymbol{v} \in \mathbb{F}_{p^q}^n$ from $\mathcal{A}$.
- If $\mathcal{P}$ is honest, sample $\boldsymbol{u} \in_\$ \mathbb{F}_p^n$ and compute $\boldsymbol{w} \triangleq \boldsymbol{v} - \boldsymbol{u} \cdot \Delta \in \mathbb{F}_{p^q}^n$, else receive $\boldsymbol{u} \in \mathbb{F}_p^n$ and $\boldsymbol{w} \in \mathbb{F}_{p^q}^n$ from $\mathcal{A}$ and compute $\boldsymbol{v} \triangleq \boldsymbol{w} + \boldsymbol{u} \cdot \Delta \in \mathbb{F}_{p^q}^n$.
- Send $(\boldsymbol{u}, \boldsymbol{w})$ to $\mathcal{P}$ and $\boldsymbol{v}$ to $\mathcal{V}$.

Figure 2: The subfield VOLE correlation functionality

for single disjunction implements $\mathcal{F}_{\mathsf{ZK}}^{1,B}$ (for $B \in \mathbb{Z}^+$) and our protocol for batched disjunctions implements $\mathcal{F}_{\mathsf{ZK}}^{R,B}$ (for $R, B \in \mathbb{Z}^+$). See Appendix A.1 for UC framework background.

2.3 VOLE and IT-MACs

Recent works [BCGI18, BCG+19a, SGRR19, BCG+19b, YWL+20, CRR21] have improved the efficiency of subfield VOLE (i.e., Figure 2). The state-of-the-art VOLE implementation requires only linear computation and *sublinear* communication in the number of generated VOLE correlations.

In VOLE-based ZK, VOLE correlations allow $\mathcal{P}$ to commit to wire values using information-theoretic MACs (IT-MACs). Let $x \in \mathbb{F}$ be a field element known to $\mathcal{P}$ (e.g., part of her witness). An IT-MAC commitment to x is a pair of values held respectively by $\mathcal{P}$ and $\mathcal{V}$. Specifically, $\mathcal{V}$ holds a key $k \in_\$ \mathbb{F}$ and $\mathcal{P}$ holds $m = k - x \cdot \Delta$, where $\Delta \in_\$ \mathbb{F}$ is a key which is global to the entire proof, known to $\mathcal{V}$, and hidden from $\mathcal{P}$. We denote a commitment to x under global key Δ by writing $[x]_\Delta$, where Δ will be omitted if it is clear from the context. I.e., $[x]_\Delta$ is a pair of tuples (m_x, x), held by $\mathcal{P}$, and (k_x, Δ), held by $\mathcal{V}$. We use $[\boldsymbol{x}]_\Delta$ to denote IT-MACs of vector $\boldsymbol{x}$. Note that $\mathcal{P}$ can efficiently open a commitment $[x]_\Delta$ by sending (m_x, x).

An IT-MAC $[x]_\Delta$ has the following notable features:

- **Hiding:** $\mathcal{V}$' s share the (k_x, Δ) is independent of the secret x, so the share trivially hides x.
- **Binding:** Malicious $\mathcal{P}$ cannot cheat and open a commitment $[x]_\Delta$ to some $x' \neq x$. Indeed, forging a suitable opening is as hard as guessing Δ. Note that (m_x, x) conveys no information about Δ.
- **Linear homomorphism:** $[x+y]_\Delta = [x]_\Delta + [y]_\Delta$. $[cx]_\Delta = c[x]_\Delta$ and $[x+c]_\Delta = [x]_\Delta + c$, for some public c.

The VOLE functionality allows $\mathcal{P}$ and $\mathcal{V}$ to construct n IT-MAC commitments, each to a uniformly random value $[r]_\Delta$ where $r \in_\$ \mathbb{F}$. A random commitment $[r]_\Delta$ can be easily translated into a commitment $[x]_\Delta$ where x is a value chosen by $\mathcal{P}$: $\mathcal{P}$ simply sends to $\mathcal{V}$ the single field element $(x - r)$, and $\mathcal{V}$ correspondingly locally shifts his key by $(x - r) \cdot \Delta$. Thus, to commit to n field elements, $\mathcal{P}$ and $\mathcal{V}$ first execute VOLE, and then $\mathcal{P}$ transmits $n \cdot \lceil \log |\mathbb{F}| \rceil$ bits.

Field extension. When the ZK statement is defined over a small field $\mathbb{F}_p$ (e.g., Boolean), we need to use IT-MACs defined over an extension field $\mathbb{F}_{p^q}$ to ensure that Δ cannot be easily guessed. In this case, it suffices to consider random IT-MACs $[r]_\Delta$ where r is drawn from the *base field* $\mathbb{F}_p$ because r is only used to mask $x \in \mathbb{F}_p$. There exists a VOLE variant that works over $\mathbb{F}_{p^q}$, but generates IT-MACs of such r values from the *subfield* $\mathbb{F}_p$. This variant is called subfield VOLE. I.e., an IT-MAC $[r]_\Delta$ generated by subfield VOLE will have $m_r, k_r, \Delta \in \mathbb{F}_{p^q}$ *but* $r \in \mathbb{F}_p$.

It is sometimes necessary to mix VOLE and subfield VOLE correlations in a single proof. This is easy: we can linearly combine q subfield VOLE correlations into 1 VOLE correlation over the extension field $\mathbb{F}_{p^q}$. This incurs factor q blowup.

2.4 LPZK [DIO21] and QuickSilver [YSWW21]

VOLE-based ZK works in the "commit-and-prove" paradigm: $\mathcal{P}$ commits to her extended witness with IT-MACs, and later proves to $\mathcal{V}$ that committed values are consistent with the proof statement.

Consider a proof statement encoded as an arithmetic circuit $\mathcal{C}$, and let $\mathcal{P}$ hold a witness $\boldsymbol{w}$. To prove $\mathcal{C}(\boldsymbol{w}) = 0$, $\mathcal{P}$ first computes her *extended witness* $\boldsymbol{w}_{ext}$, which consists of $\boldsymbol{w}$ along with each multiplication gate's output value upon evaluating $\mathcal{C}(\boldsymbol{w})$. The parties then construct commitments to $\boldsymbol{w}_{ext}$, as described above. From here, $\mathcal{P}$ and $\mathcal{V}$ can use the linear homomorphism property of IT-MACs to construct commitments to the output value of each addition gate.

Note that at this point, $\mathcal{P}$ is now committed to a particular value for every wire in the circuit. Two tasks remain:

- $\mathcal{P}$ must prove to $\mathcal{V}$ that the committed value of the output wire is 0. This is achieved simply by opening.
- For each multiplication gate, $\mathcal{P}$ must prove that the gate's committed input values indeed multiply to the gate's committed output value.

Previous VOLE-based ZKs mainly differ in the way they handle multiplication gates. State-of-the-art VOLE-based ZK [DIO21, YSWW21] handles multiplication gates via a technique called *line-point zero-knowledge* (LPZK). At a high level, LPZK [DIO21] proves that n IT-MAC tuples $\{[l_i]_\Delta, [r_i]_\Delta, [o_i]_\Delta\}_{i\in[n]}$ satisfy the multiplication relation $o_i = l_i \cdot r_i$ by utilizing (1) another random IT-MAC and (2) algebra over IT-MAC shares. The technique can be achieved at the cost of (1) $\mathcal{V}$ sending a random challenge and (2) $\mathcal{P}$ sending 2 field elements. Each party computes $\mathcal{O}(n)$ field operations. We denote the procedure to prove multiplications for IT-MACs as $\mathsf{LPZK}(\{[l_i]_\Delta, [r_i]_\Delta, [o_i]_\Delta\}_{i\in[n]})$, which we use as a black-box. LPZK has $(n+2)/p^q$ soundness error and information-theoretic security in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model [YSWW21][3].

QuickSilver [YSWW21] subsequently generalized LPZK to efficiently handle arbitrary polynomials over committed values. Our protocols use this trick for proving the inner-product of IT-MACs. Namely, given $2m$ IT-MACs $([x_1]_\Delta, \ldots, [x_m]_\Delta)$ and $([y_1]_\Delta, \ldots, [y_m]_\Delta)$, QuickSilver shows how to efficiently prove that $x_1y_1 + \ldots + x_my_m = 0$. The proof requires $\mathcal{O}(1)$ communication and $\mathcal{O}(m)$ computation. Further, incorporating a random challenge from $\mathcal{V}$, k inner-product proofs can be batched, preserving $\mathcal{O}(1)$ communication. We denote the procedure to prove k-batched inner-products for IT-MACs as $\mathsf{QS}(\{[x_i^{(j)}]_\Delta, [y_i^{(j)}]_\Delta\}_{i\in[m],j\in[k]})$. We will use it as a black-box subprotocol.

[3][YSWW21] uses "extended subfield VOLE", which handles higher degree polynomials. We only use degree-2 polynomials, so $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$ is sufficient.

This sub-proof, as shown by [YSWW21], is zero-knowledge with $(k+2)/p^q$ soundness error and information-theoretic security in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model[3].

We defer additional details on LPZK and QS to Appendix A.2.

3 Technical Overview

In this section, we present our techniques with sufficient detail to understand our contribution. Our ZK protocol considers standard arithmetic circuits with fan-in-2 gates. For ease of presentation, throughout this section, we consider circuits defined over a prime field $\mathbb{F}_p$ where p is large enough to achieve the desired soundness (e.g., $p = 2^{61} - 1$) without using VOLE with an extension field.

Recall that our goal is to extend VOLE-based ZK such that it can efficiently handle proofs of (batched) disjunctive statements.

Consider B circuits $\mathcal{C}_1, \ldots, \mathcal{C}_B$ (each called a branch) with the same number of input wires and multiplication gates, which is padded if needed. To begin, suppose $\mathcal{P}$ wishes to prove a single disjunction (we will discuss batching shortly). I.e., $\mathcal{P}$ wishes to prove $\mathcal{C}_1(\boldsymbol{w}) = 0 \vee \cdots \vee \mathcal{C}_B(\boldsymbol{w}) = 0$.

Note that basic VOLE-based ZK (e.g., [YSWW21]) scales with the number of branches B, both in communication and computation. The primary source of this cost is simply the commitment of $\mathcal{P}$'s extended witness, which is linear in the total number of multiplication gates.

In our approach, $\mathcal{P}$ commits to a much shorter string whose length scales only with the number of multiplications (and inputs) in a *single* branch. This reduction in the size of the committed string is the source of our improvement.

In slightly more detail, $\mathcal{P}$ commits to a *modified* version of the extended witness $\boldsymbol{w}_{ext}$. In addition to the inputs of the *active* circuit, $\boldsymbol{w}_{ext}$ includes input/output wire values of each multiplication gate of the active branch[4]. We use $\mathsf{out}(\boldsymbol{w}_{ext})$ (resp. $\mathsf{left}(\boldsymbol{w}_{ext}), \mathsf{right}(\boldsymbol{w}_{ext})$) to denote the vector of multiplication-gate outputs (resp. mult-gate left inputs, mult-gate right inputs) taken from $\boldsymbol{w}_{ext}$. From here, $\mathcal{P}$ proves that the committed multiplication inputs/outputs indeed respect multiplication. Namely, $\mathcal{P}$ proves $\mathsf{out}(\boldsymbol{w}_{ext}) = \mathsf{left}(\boldsymbol{w}_{ext}) \circ \mathsf{right}(\boldsymbol{w}_{ext})$ where $\circ$ denotes the element-wise product. This check is performed using the techniques of prior work (LPZK). Note that the number of checks does not scale with the number of branches.

So far, $\mathcal{P}$ has simply introduced and committed to a length-n_{in} vector of inputs and a length-$n_\times$ vector of tuples, each of which respects multiplication. The remaining task is to force $\mathcal{P}$ to choose this vector such that it satisfies the structure of *some* active branch $\mathcal{C}_a$. That is, $\mathcal{P}$ must prove that $\boldsymbol{w}_{ext}$ respects the *topology* of branch $\mathcal{C}_a$, as well as the linear constraints implied by $\mathcal{C}_a$'s addition gates. As we will describe in detail later, we can enforce such constraints by introducing *public* matrices $\boldsymbol{M}_i$ of size $O(|\mathcal{C}|) \times O(|\mathcal{C}|)$, encoding the topology of $\mathcal{C}_i$ *a-la* adjacency matrix. For each branch $\mathcal{C}_i$ with matrix $\boldsymbol{M}_i$, consider the following crucial equation:

$$\boldsymbol{M}_i \times \boldsymbol{w}_{ext} = \mathbf{0} \tag{1}$$

Equation (1) has two notable properties:

- If $\mathcal{P}$ is honest and holds a witness that satisfies active branch a, Equation (1) will hold for branch a.

[4]This means that our extended witness is up to $3\times$ longer than the one considered by prior work if $B = 1$. While we pay a small constant factor overhead *on the active branch*, we asymptotically decrease the size of $\mathcal{P}$'s commitment by up to factor B.

- If $\mathcal{P}$ attempts to cheat and does not have a valid witness, w.h.p. Equation (1) will not hold for any i.

We defer further details on the structure of these matrices until Section 3.3. It is worth noting that although the size of these matrices is $\mathcal{O}(|\mathcal{C}|^2)$, we will demonstrate that all relevant operations we used on these matrices can be computed in time $\mathcal{O}(|\mathcal{C}|)$.

Terminology. Our approach centers on the manipulation of matrices $\boldsymbol{M}_i$ which encode the topology of circuits $\mathcal{C}_i$. We find it helpful to introduce terminology for these matrices.

- We refer to each matrix $\boldsymbol{M}$ as a *topology matrix*. $\boldsymbol{M}$ is a matrix of dimension $O(|\mathcal{C}|) \times O(|\mathcal{C}|)$.
- We will allow $\mathcal{V}$ to select random challenge vectors $\boldsymbol{s}$, and we will consider products $\boldsymbol{s}^T \times \boldsymbol{M}$. We refer to the resulting length $O(|\mathcal{C}|)$ vector as a *compressed topology vector*.
- Additionally, we will right multiply compressed topologies by vectors. The result of such a multiplication is a scalar that we refer to as a *compressed topology token*.

3.1 Robin: Single Disjunction Protocol

In the single instance setting, we wish to improve communication. Recall that we consider statements of the form $(\mathcal{C}_1(\boldsymbol{w}) = 0 \vee \cdots \vee \mathcal{C}_B(\boldsymbol{w}) = 0)$. Our goal is to achieve communication that scales with $B + |\mathcal{C}|$, not $B|\mathcal{C}|$, while preserving the low computation cost of the basic VOLE-based ZK.

Our key insight is that $\mathcal{V}$ can challenge $\mathcal{P}$ with a random vector $\boldsymbol{s}$ *after* $\mathcal{P}$ commits to $\boldsymbol{w}_{ext}$. Both parties can then use the IT-MAC commitment $[\boldsymbol{w}_{ext}]_\Delta$ to homomorphically (i.e., without further communication) derive B IT-MAC commitments of the following compressed topology tokens:

$$\left[\boldsymbol{s}^T \times \boldsymbol{M}_1 \times \boldsymbol{w}_{ext}\right]_\Delta, \ldots, \left[\boldsymbol{s}^T \times \boldsymbol{M}_B \times \boldsymbol{w}_{ext}\right]_\Delta$$

Crucially, we prove that from the properties of Equation (1), these B IT-MAC commitments have the following induced properties:

- If $\mathcal{P}$ is honest and commits a witness that satisfies active branch a, $[\boldsymbol{s}^T \times \boldsymbol{M}_a \times \boldsymbol{w}_{ext}]_\Delta$ will *always* be $[0]_\Delta$.
- If cheating $\mathcal{P}$ does not have a valid witness, then *with overwhelming probability*, none of these values will be $[0]_\Delta$.

To complete the proof, $\mathcal{P}$ simply needs to demonstrate that one of these committed tokens is a 0. We achieve this in a direct way: we run a (much smaller) VOLE-based ZK proof demonstrating that the product of the B elements is 0.

All in all, $\mathcal{P}$ demonstrates: There exists an extended witnesses $\boldsymbol{w}_{ext}$ s.t. for a random challenge $\boldsymbol{s}$, the following holds:

$$\begin{aligned} &(\mathsf{left}(\boldsymbol{w}_{ext}) \circ \mathsf{right}(\boldsymbol{w}_{ext}) = \mathsf{out}(\boldsymbol{w}_{ext})) && \text{multiplication check} \\ \wedge &\left(\left(\prod_i^B \boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{w}_{ext}\right) = 0\right) && \text{topology check} \end{aligned}$$

Note that the order of quantifiers in the above statement is crucial, implying the order in which the proof proceeds. In short, the full proof proceeds as follows:

1. $\mathcal{P}$ commits to the extended witness $\boldsymbol{w}_{ext}$.
2. $\mathcal{P}$ and $\mathcal{V}$ check that multiplication wires are properly committed by using the existing LPZK technique.
3. $\mathcal{V}$ sends to $\mathcal{P}$ the random challenge vector $\boldsymbol{s}$.
4. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[\boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{w}_{ext}]_\Delta$ for each $i \in [B]$.
5. $\mathcal{P}$ and $\mathcal{V}$ use VOLE-based ZK to prove that the product of these B commitments is 0.

3.2 Batchman: Batched Disjunctions Protocol

In the batched setting, we wish to improve not only communication, but also computation. Recall, we consider the statement $(\mathcal{C}_1(\boldsymbol{w}_j) = 0 \vee \cdots \vee \mathcal{C}_B(\boldsymbol{w}_j) = 0)$ on R different witnesses.[5] Our goal is to achieve computation that scales with $B + R$, not BR.

As a first attempt, one could try simply applying our single instance approach R times; this fails, because computing each commitment $[s^T \times \boldsymbol{M}_i \times \boldsymbol{w}_{ext}]_\Delta$ requires $O(|\mathcal{C}|)$ field operations, and so ultimately this attempt uses $O(RB|\mathcal{C}|)$ field operations.

As a second attempt, one could use RAM-based ZKs. While this works for large R, it does not match our asymptotics for small R and is concretely expensive; see Section 1.2.

Our batched approach relies on three key insights:

1. $\mathcal{P}$ knows the active branch $\mathcal{C}_a$/matrix $\boldsymbol{M}_a$ for each repetition.
2. It is safe to re-use the challenge vector $\boldsymbol{s}$ across all R instances.
3. The compressed topology vector $\boldsymbol{s}^T \times \boldsymbol{M}_a$ is *small*, having length only $\mathcal{O}(|\mathcal{C}|)$ field elements.

Thus, for each repetition $j \in [R]$, we can require that $\mathcal{P}$ commits to her extended witness $\boldsymbol{w}_{ext}^{(j)}$ *and* to her desired branch $a^{(j)}$. In particular, if $\mathcal{P}$ is honest, she will commit to the compressed topology vector of the active branch as $[\boldsymbol{cv}^{(j)} \triangleq \boldsymbol{s}^T \times \boldsymbol{M}_{a^{(j)}}]_\Delta$. From here, the parties use a regular VOLE-based ZK proof (QS) to show that $\mathcal{P}$'s committed witness respects the committed compressed topology vector. Namely, they check:

$$\left(\boldsymbol{cv}^{(j)}\right)^T \times \boldsymbol{w}_{ext}^{(j)} = 0, \text{ for all } j \in [R]$$

Crucially, the computation cost of this inner product check *does not* scale with the number of branches B, because $\mathcal{P}$ directly chooses and commits to *only* the active branch.

Suppose that a cheating $\mathcal{P}$ does not have a witness for some repetition j. Based on our reasoning in Section 3.1, passing the above check is negligibly likely, if $\boldsymbol{cv}^{(j)}$ is equal to the compressed topology vector of some branch. Of course, it might be the case that cheating $\mathcal{P}$ committed to some vector $\boldsymbol{cv}^{(j)}$ which is *not* equal to any branch's compressed topology $\boldsymbol{s}^T \times \boldsymbol{M}_{i \in [B]}$.

[5]Of course, it is not useful to prove the same statement more than once without imposing additional constraints; it is easy to incorporate extra mechanisms that force $\mathcal{P}$ to R times prove the statement wrt *related* witnesses. See discussion in Section 6.

To repair this, we add a step to validate that $\mathcal{P}$ indeed committed to the compressed topology of some branch. In particular, we allow $\mathcal{V}$ to issue a second challenge vector $\boldsymbol{t}$, and then the parties once and for all precompute the following compressed topology tokens:

$$ct_i \triangleq \boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{t}, \text{ for each } i \in [B]$$

Computing these values takes time proportional to B, but *not proportional to* R as each value is computed exactly *once*; once computed, $\mathcal{P}$ and $\mathcal{V}$ re-use these values in each of the R batched proof instances.

The above validation will catch a cheating $\mathcal{P}$ with overwhelming probability (in the size of the field $\mathbb{F}$). More precisely, we observe (and prove) that if $\boldsymbol{cv} \notin \{\boldsymbol{s}^T \times \boldsymbol{M}_i\}_{i\in[B]}$, *with overwhelming probability,* $(\boldsymbol{cv}^T \times \boldsymbol{t}) \notin \{ct_i\}_{i\in[B]}$. Furthermore, for each $j \in [R]$, parties already hold $[\boldsymbol{cv}^{(j)}]_\Delta$, so they can locally compute $[(\boldsymbol{cv}^{(j)})^T \times \boldsymbol{t}]_\Delta$ and perform a regular VOLE-based ZK proof to show:

$$\left((\boldsymbol{cv}^{(j)})^T \times \boldsymbol{t}\right) \in \{ct_i\}_{i\in[B]}$$

Each token ct_i is a single field element, so this check is efficient.

All in all, $\mathcal{P}$ demonstrates: There exist R extended witnesses $\boldsymbol{w}_{ext}^{(1)}, \ldots, \boldsymbol{w}_{ext}^{(R)}$ s.t. for a random challenge $\boldsymbol{s}$ there exist R vectors $\boldsymbol{cv}^{(1)}, \ldots, \boldsymbol{cv}^{(R)}$ s.t. for a random challenge $\boldsymbol{t}$, the following holds: for each $j \in [R]$,

$$\begin{aligned}
&\left(\mathsf{left}(\boldsymbol{w}_{ext}^{(j)}) \circ \mathsf{right}(\boldsymbol{w}_{ext}^{(j)}) = \mathsf{out}(\boldsymbol{w}_{ext}^{(j)})\right) && \text{mult. check}\\
\wedge &\left((\boldsymbol{cv}^{(j)})^T \times \boldsymbol{w}_{ext}^{(j)} = 0\right) && \text{topo. check}\\
\wedge &\left(\left((\boldsymbol{cv}^{(j)})^T \times \boldsymbol{t}\right) \in \{\boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{t}\}_{i\in[B]}\right) && \text{topo. valid}
\end{aligned}$$

The order of quantifiers in the above statement is crucial, implying the order in which the proof proceeds. In short, the full batched proof proceeds as follows:

- $\mathcal{P}$ commits to each extended witness $\boldsymbol{w}_{ext}^{(j)}$.
- $\mathcal{P}$ and $\mathcal{V}$ check multiplication wires are properly committed by using the existing LPZK technique.
- $\mathcal{V}$ sends to $\mathcal{P}$ the random challenge vector $\boldsymbol{s}$.
- $\mathcal{P}$ commits to each compressed topology $\boldsymbol{cv}^{(j)}$.
- $\mathcal{P}$ and $\mathcal{V}$ check each inner-product $(\boldsymbol{cv}^{(j)})^T \times \boldsymbol{w}_{ext}^{(j)}$ is equal to zero by using the existing QS technique.
- $\mathcal{V}$ sends to $\mathcal{P}$ the random challenge vector $\boldsymbol{t}$.
- $\mathcal{P}$ and $\mathcal{V}$ locally compute $\boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{t}$ for each $i \in [B]$.
- $\mathcal{P}$ and $\mathcal{V}$ use VOLE-based ZK to prove that each committed vector $(\boldsymbol{cv}^{(j)})^T \times \boldsymbol{t}$ is a valid compressed topology token.

3.3 Topology Matrices

We now discuss how we construct and use branch-specific public topology matrices $\boldsymbol{M}$. Recall, these matrices allow $\mathcal{V}$ to verify that $\mathcal{P}$'s extended witness indeed satisfies the structure of *some* branch of $\mathcal{C}_1, \ldots, \mathcal{C}_B$. This verification is achieved by Equation (1).

Recall, our extended witness $\boldsymbol{w}_{ext}$ includes (1) $\mathcal{P}$'s witness $\boldsymbol{w}$ and, for each multiplication gate in the active branch, (2) its output wire and (3) its two input wires.

Consider a branch $\mathcal{C}$, and suppose that we remove each multiplication gate from $\mathcal{C}$. Whenever we remove a multiplication gate, we replace its input and output wires with inputs to $\mathcal{C}$. What remains is a skeleton of the circuit containing only addition gates that expresses a linear relationship on the extended witness (and $\mathcal{C}$'s output). It is convenient to encode this linear relationship as a matrix $\boldsymbol{M} \in \mathbb{F}^{(2\mathcal{C}_\times+1)\times(\mathcal{C}_{in}+3\mathcal{C}_\times+1)}$, and we refer to this matrix as a *topology matrix*. Figure 3 shows an example.

Note that $\boldsymbol{w}_{ext}$ is a *valid* extended witness for $\mathcal{C}$ if and only if:

1. Multiplication gates in the $\boldsymbol{w}_{ext}$ are formed correctly.
2. $\boldsymbol{M} \times \mathsf{glue}(\boldsymbol{w}_{ext}, 1) = \boldsymbol{0}$, where glue appends 1 to vector $\boldsymbol{w}_{ext}$[6].

The above requirements imply that, for an *invalid* extended witness $\boldsymbol{w}_{ext}$, if Item 1 is satisfied, Item 2 will not be satisfied. This is precisely our Equation (1) and associated properties with one caveat: we did not append 1 to $\boldsymbol{w}_{ext}$. This can be trivially fixed, because $\mathcal{P}$ and $\mathcal{V}$ can *locally* generate shares of $[1]_\Delta$.

Efficient operations on topology matrices. Recall that we left multiply topology matrices $\boldsymbol{M}$ by vectors $\boldsymbol{s}^T$: we compute $\boldsymbol{s}^T \times \boldsymbol{M}$.

Computed naïvely, the above multiplication is expensive. Indeed, $\boldsymbol{M}$ can be *dense*, due to *unlimited* fan-out from addition gates. Therefore, storing $\boldsymbol{M}$ and naïvely computing the product will incur $O(|\mathcal{C}|^2)$ overhead, far exceeding our asymptotic budget.

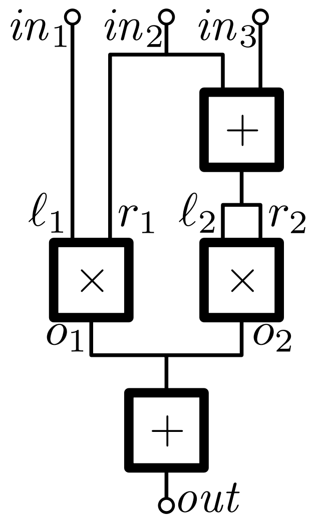
Perhaps surprisingly, given the gate-by-gate representation of $\mathcal{C}$, this multiplication can be computed in time $O(|\mathcal{C}|)$ with our technique "evaluating $\mathcal{C}$ backwards" – see Section 4. We name the corresponding algorithm $\mathsf{MUL}_{\mathsf{LEFT}}$. $\mathsf{MUL}_{\mathsf{LEFT}}$ never explicitly computes $\boldsymbol{M}$. Thus, the topology matrix $\boldsymbol{M}$ is merely an analysis tool, and our protocols work entirely with efficient gate-by-gate circuit representations. In other words, it suffices to *think* of circuits as topology matrices, while in reality all algorithms *operate* on compact gate-by-gate representations.

4 Formalizing Topology Matrices

In this section, we formalize *topology matrices*, a tool used to prove the correctness and security of our approach; see Section 3. We also give an algorithm that allows efficient vector-matrix multiplication on topology matrices.

Linear constraint on a wire. Consider a wire w_k in a circuit $\mathcal{C}$. The wire w_k can be defined as a linear combination of input wires of $\mathcal{C}$ and output wires of all multiplication gates. We call this linear combination *the linear constraint* on w_k.

[6]This 1 enables *offset gates*, which take a public constant as input.



$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} in_1 \\ in_2 \\ in_3 \\ \ell_1 \\ \ell_2 \\ r_1 \\ r_2 \\ o_1 \\ o_2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} in_1 - \ell_1 & in_2 + in_3 - \ell_2 & in_2 - r_1 & in_2 + in_3 - r_2 & o_1 + o_2 \end{bmatrix}^T$$

Figure 3: A simple arithmetic circuit computing $(in_1 \cdot in_2) + (in_2 + in_3)^2$ (left) and its corresponding topology matrix (right). Note, the shaded portion of the matrix is *dense.*

Following this, a circuit's linear constraints can be captured by its associated *topology matrix* (see Figure 3 for an example):

Definition 1 (Topology Matrix)**.** *Let $\mathcal{C}$ denote a circuit over some field $\mathbb{F}$ such that $\mathcal{C}$ has n_{in} input wires and $n_\times$ multiplication gates. The* ***topology matrix*** *associated with $\mathcal{C}$ is a $(2n_\times + 1) \times (n_{in} + 3n_\times + 1)$ matrix over $\mathbb{F}$ defined as follows.*

Let $\boldsymbol{aux} \triangleq (in_1, \ldots, in_{n_{in}}, \ell_1, \ldots, \ell_{n_\times}, r_1, \ldots, r_{n_\times}, o_1, \ldots, o_{n_\times}, 1)^T$ denote a vector of circuit metadata. Here, in_k represents the kth input, ℓ_k (resp. r_k, o_k) represents the left (resp. right, output) wire of the kth multiplication gate, and 1 is the multiplicative identity of $\mathbb{F}$. The rows of the topology matrix M are:

1. ***Left wires:*** *For the first $n_\times$ rows, for each $k \in [n_\times]$, $\boldsymbol{M}(k) \times \boldsymbol{aux}$ is the linear constraint on wire ℓ_k. E.g., $in_1 - \ell_1 = 0$. We require $\boldsymbol{M}(k)[n_{in} + k] = -1$.*

2. ***Right wires:*** *For the second $n_\times$ rows, for each $k \in [n_\times]$, $\boldsymbol{M}(n_\times + k) \times \boldsymbol{aux}$ is the linear constraint on wire r_k. E.g., $in_2 - r_1 = 0$. We require $\boldsymbol{M}(n_\times + k)[n_{in} + n_\times + k] = -1$.*

3. ***Circuit output:*** *For the last row $\boldsymbol{M}(2n_\times + 1)$, $\boldsymbol{M}(2n_\times + 1) \times \boldsymbol{aux}$ is the linear constraint on the output of the circuit. E.g., $o_1 + o_2$ is the circuit output.*

Item 3 can be naturally extended to capture circuits with multiple outputs. Note that for $\boldsymbol{aux}$ to be a valid extended witness, $\boldsymbol{M} \times \boldsymbol{aux}$ must be the all zeros vector.

Left multiplication for topology matrices. Recall, our $\mathcal{P}$ and $\mathcal{V}$ left multiply topology matrices $\boldsymbol{M}_i$ by random vectors $\boldsymbol{s}^T$. Using naïve vector-matrix multiplication, computing $\boldsymbol{s}^T \times \boldsymbol{M}$ requires $\mathcal{O}(|\mathcal{C}|^2)$ field operations, exceeding our asymptotic budget. Instead, we propose an efficient

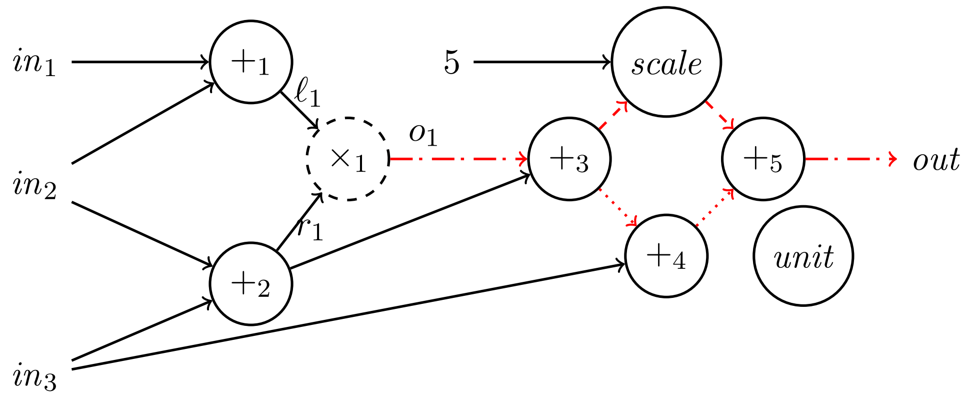


Figure 4: A circuit's induced DAG. In the topology matrix of this circuit, the last row defines $out \triangleq 6in_2 + 7in_3 + 6o_1$. In this DAG, there are 6 paths from in_2 to out, 7 paths from in_3 to out, and 6 paths from o_1 to out. E.g., from o_1, there are 5 paths (dashed) passing through the scale gate and 1 path (dotted) passing through the addition gate $+_4$, so there are 6 paths in total. The topology matrix reflects these numbers of paths. Since there is no offset gate, the unit vertex is isolated.

Algorithm 1: $\mathsf{MUL}_{\mathsf{LEFT}}$ takes as input (1) an arithmetic circuit $\mathcal{C}$ over a field $\mathbb{F}$ written in gate-by-gate representation (see Definition 3 in Appendix B) and (2) a vector $\boldsymbol{s}$ over some extension field of $\mathbb{F}$ with length $2n_\times + 1$. It outputs $s^T \times \boldsymbol{M}$ where $\boldsymbol{M}$ is the topology matrix associated with $\mathcal{C}$. Array indices start at 1.

```
    Data: circuit $\mathcal{C}$, vector $\boldsymbol{s}$
    Result: $\boldsymbol{s}^T \times \boldsymbol{M}$
1  $\boldsymbol{w} = \mathbf{0}^{(|\mathcal{C}.\mathsf{wid}|)}$ defined over the extension field;
2  $acc = 0$ defined over the extension field;
3  $w[|\mathcal{C}.\mathsf{wid}|] = s[2n_\times + 1]$;
4  for each multiplication gate $(\ell_k, r_k, o_k)$ do
5      $w[\ell_k] = w[\ell_k] + s[k]$;  $w[r_k] = w[r_k] + s[k + n_\times]$;
6  for each linear gate $G$ in reverse topological order do
7      if $G$ is an addition gate $(\ell'_k, r'_k, o'_k)$ then
8          $w[\ell'_k] = w[\ell'_k] + w[o'_k]$;  $w[r'_k] = w[r'_k] + w[o'_k]$;
9      if $G$ is a scale gate $(c_k, x_k, y_k)$ then
10         $w[x_k] = w[x_k] + c_k \cdot w[y_k]$;
11     if $G$ is an offset gate $(c'_k, x'_k, y'_k)$ then
12         $w[x'_k] = w[x'_k] + w[y'_k]$;  $acc = acc + c'_k \cdot w[y'_k]$;
13 $\boldsymbol{res} = \{\}$;
14 for each input wire $in_k$ in order do  $\boldsymbol{res}.\mathsf{append}(w[in_k])$ ;
15 for each $k \in [2n_\times]$ do  $\boldsymbol{res}.\mathsf{append}(-s[k])$ ;
16 for each multiplication gate $(\ell_k, r_k, o_k)$ in order do
17     $\boldsymbol{res}.\mathsf{append}(w[o_k])$;
18 $\boldsymbol{res}.\mathsf{append}(acc)$;
19 return $\boldsymbol{res}$
```

algorithm called $\mathsf{MUL_{LEFT}}$ to support the above operation. Given the gate-by-gate circuit representation[7] of $\mathcal{C}$, our algorithm essentially gate-by-gate evaluates $\mathcal{C}$ "backwards" in $\mathcal{O}(|\mathcal{C}|)$ operations *without ever writing down the matrix* $\boldsymbol{M}$.

It is not obvious that this multiplication can be achieved in $O(|\mathcal{C}|)$ operations, as the matrix $\boldsymbol{M}$ can be *dense* due to high fan-out addition gates (see Figure 3 as an example). While the matrix $\boldsymbol{M}$ is not sparse, it *is* highly structured: indeed, the circuit $\mathcal{C}$ is itself a succinct representation of $\boldsymbol{M}$, and our algorithm exploits this.

Our $\mathcal{O}(|\mathcal{C}|)$ solution. Algorithm 1 presents $\mathsf{MUL_{LEFT}}$. To understand our algorithm, we analyze the semantics of topology matrices. Let $\mathcal{C}$ denote a circuit, and consider $\mathcal{C}$'s underlying directed acyclic graph G; i.e, the vertices in G represent gates and edges in G represent wires (see Figure 4 as an example). Now, remove each vertex corresponding to a multiplication gate in $\mathcal{C}$. Additionally, add one special vertex to G called the *unit* vertex, which will denote a wire holding value 1 to capture *offset gates*.

Let $\boldsymbol{M}$ denote the topology matrix associated with $\mathcal{C}$. Now, consider the first row of $\boldsymbol{M}$ (denoted as $\boldsymbol{M}(1)$). As specified by Definition 1, this row defines the linear constraint on ℓ_1, the left input wire of $\mathcal{C}$'s first multiplication gate. The first element of $\boldsymbol{M}(1)$ can be understood as the number of paths in G that start at vertex in_1 and terminate at vertex ℓ_1. (Arithmetic circuits admit *scalar* gates which scale the input by a public constant; for a gate with scalar s, we say that there are s paths from that gate's input to its output. We also consider *offset* gates which add a public constant to a wire; for a gate with offset s, we say that there are s paths from the unit vertex to the gate output.) See Figure 4 for an example.

More generally, $\boldsymbol{M}(i)[j]$ can be understood as the number of paths from auxiliary wire (see Definition 1) aux_j to multiplication gate input i. There are two special cases: (1) we define the number of paths from a wire to itself to be -1; (2) the last row determines the number of paths to the circuit output wire (not multiplication).

Now, consider the first column of $\boldsymbol{M}$ (denoted as $\boldsymbol{M}[1]$). Based on the above analysis, this column can be understood as the number of paths from in_1 to $\ell_1, \ldots, \ell_{n_\times}, r_1, \ldots, r_{n_\times}, out$. The crucial point is this: in the graph G, the number of paths from wire a to wire b is trivially equal to the number of *backwards* paths (i.e., paths through the graph with all edges reversed) from b to a. Thus, if we wish to compute the inner product of some vector $\boldsymbol{s}$ with $\boldsymbol{M}[1]$, we can (1) put those values of $\boldsymbol{s}$ onto the wires $\ell_1, \ldots, \ell_{n_\times}, r_1, \ldots, r_{n_\times}, out$, (2) evaluate the circuit (with multiplication gates removed) *backwards* and (3) output the value on wire in_1.

Note that backwards evaluation of linear gates has a clear interpretation. In particular, for an addition gate we add together its output wire values, then place the sum onto the two input wires.

Therefore, to compute the full vector-matrix product $\boldsymbol{s}^T \times \boldsymbol{M}$, we simply evaluate the arithmetic gates backwards, and then output wire values in the order prescribed by $\boldsymbol{aux}$. This is precisely the approach of Algorithm 1. Because we evaluate each linear gate exactly once, the complexity of Algorithm 1 is trivially $\mathcal{O}(|\mathcal{C}|)$.

[7]Appendix B defines a standard arithmetic circuit gate-by-gate representation.

5 Robin: Single Disjunction Protocol

5.1 Soundness Lemmas

As discussed in Section 3, our protocols heavily rely on the fact that $\mathcal{V}$ can issue random vectors to compress commitments, leading to small proofs. Formally, these random challenges preserve soundness based on the following lemmas and associated corollaries, which are the kernel of our protocols and proofs.

Lemma 1. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$. The following holds:*

$$\Pr[\exists i \in [k], (\boldsymbol{x}^{(i)})^T \times \boldsymbol{s} = 0 \mid \boldsymbol{s} \in_\$ \mathbb{F}^m] \leq k/|\mathbb{F}|$$

See Appendix C.1 for a proof.

Corollary 1. *If $\boldsymbol{s}$ is drawn from the extension field $\mathbb{F}^q$ where $q \in \mathbb{Z}^+$, the upper bound of Lemma 1 is $k/|\mathbb{F}|^q$.*

Corollary 2. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$ and a vector $\boldsymbol{y} \in \mathbb{F}^m$ such that $\boldsymbol{y} \notin \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(k)}\}$. The following holds:*

$$\Pr[(\boldsymbol{y}^T \times \boldsymbol{s}) \in \{(\boldsymbol{x}^{(1)})^T \times \boldsymbol{s}, \dots, (\boldsymbol{x}^{(k)})^T \times \boldsymbol{s}\} \mid \boldsymbol{s} \in_\$ \mathbb{F}^m] \leq k/|\mathbb{F}|$$

Lemma 2. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$. The following holds:*

$$\Pr[\exists i \in [k], (\boldsymbol{x}^{(i)})^T \times \boldsymbol{s} = 0 \mid \chi \in_\$ \mathbb{F}] \leq k(m-1)/|\mathbb{F}|$$

where $\boldsymbol{s} \triangleq (1, \chi, \dots, \chi^{m-1})$.

See Appendix C.2 for proof.

5.2 Formal Protocol and Analysis

We refer the reader to Section 3.1 for the intuition behind our ZK protocol for disjunctive circuit satisfiability in the single instance setting. Figure 5 formalizes our protocol; its main security property is as follows:

Theorem 1 (Single Disjunction Security). $\Pi^{p,q}_{\mathsf{Single}}$ *(Figure 5) UC-realizes $\mathcal{F}^{1,B}_{\mathsf{ZK}}$ (Figure 1) in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model with soundness error $\frac{n_\times + 2B + 4}{p^q}$ and information-theoretic security.*

We provide a detailed proof of Theorem 1 in Appendix C.3; for now, we sketch the main argument.

Proof Sketch. By constructing a simulator $\mathcal{S}$, and by extracting the witness from malicious $\mathcal{P}$.

For malicious verifier $\mathcal{A}$, $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}^{1,B}_{\mathsf{ZK}}$ by running $\mathcal{A}$ as a subroutine. $\mathcal{S}$ implements the ideal functionality $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$ on behalf of $\mathcal{A}$. Therefore, $\mathcal{S}$ knows Δ, and it can use Δ to prove *any* statement to $\mathcal{A}$ by opening commitments to whatever value it likes. $\mathcal{S}$ uses this capability to send to $\mathcal{A}$ messages identically distributed to honest $\mathcal{P}$'s real-world messages, which allows it to complete the ideal world execution.

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

Inputs. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ hold B circuits $\mathcal{C}_1, \ldots, \mathcal{C}_B$ over any field $\mathbb{F}_p$, where each circuit has n_{in} inputs and $n_\times$ multiplication gates. Prover $\mathcal{P}$ also holds a witness $\boldsymbol{w}$ and an integer $a \in [B]$ such that $\mathcal{C}_a(\boldsymbol{w}) = 0$ and $|\boldsymbol{w}| = n_{in}$.

Generate extended witness on $\mathcal{C}_a$.

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

Initialize/Preprocessing.

1. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{init})$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns a uniform $\Delta \in_\$ \mathbb{F}_{p^q}$ to $\mathcal{V}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{extend}, n_{in} + 3n_\times)$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns IT-MACs $\{[\mu_k]\}_{k\in[n_{in}]}$, $\{[\omega_k]\}_{k\in[n_\times]}$, $\{[\xi_k]\}_{k\in[n_\times]}$ and $\{[\rho_k]\}_{k\in[n_\times]}$ to the parties.
3. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{extend}, q(B-1))$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns $q(B-1)$ IT-MACs of random values over $\mathbb{F}_p$. $\mathcal{P}$ and $\mathcal{V}$ then combine these IT-MACs into $(B-1)$ IT-MACs of random values over $\mathbb{F}_{p^q}$ denoted as $\{[\tau_i]\}_{i\in[B-1]}$.

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

4. For $k \in [n_{in}]$, $\mathcal{P}$ sends $\delta_k := w_k - \mu_k \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[w_k] := [\mu_k] + \delta_k$.
5. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := \ell_k - \omega_k \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[\ell_k] := [\omega_k] + \delta_k$.
6. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := r_k - \xi_k \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[r_k] := [\xi_k] + \delta_k$.
7. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := o_k - \rho_k \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[o_k] := [\rho_k] + \delta_k$.

Check multiplication gates. $\mathcal{P}$ convinces $\mathcal{V}$ that the $n_\times$ committed multiplication gates are well-formed.

8. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based ZKP for multiplications as $\mathsf{LPZK}(\{[\ell_k], [r_k], [o_k]\}_{k\in[n_\times]})$; if ZKP fails, $\mathcal{V}$ outputs $(\texttt{false})$ and halts.

Check witness satisfies some topology. Denote $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_B \in \mathbb{F}_p^{(2n_\times+1)\times(n_{in}+3n_\times+1)}$ the topology matrices of $\mathcal{C}_1, \ldots, \mathcal{C}_B$. Let $\boldsymbol{w}_{ext} \triangleq \mathsf{glue}(\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}, 1) \in \mathbb{F}_p^{n_{in}+3n_\times+1}$ and associated IT-MAC $[\boldsymbol{w}_{ext}] \triangleq \mathsf{glue}([\boldsymbol{w}], [\boldsymbol{\ell}], [\boldsymbol{r}], [\boldsymbol{o}], [1])$. $\mathcal{P}$ convinces $\mathcal{V}$ that $\boldsymbol{M}_a \times \boldsymbol{w}_{ext} = \boldsymbol{0}$ without leaking a. I.e., there exists a satisfied circuit.

9. $\mathcal{V}$ samples a random vector $\boldsymbol{s} \in_\$ \mathbb{F}_{p^q}^{2n_\times+1}$ and sends it to $\mathcal{P}$.
10. For each $i \in [B]$, $\mathcal{P}$ and $\mathcal{V}$ compute $\boldsymbol{cv}_i := \boldsymbol{s}^T \times \boldsymbol{M}_i \in (\mathbb{F}_{p^q}^{n_{in}+3n_\times+1})^T$, then compute $[v_i] = \boldsymbol{cv}_i^T \times [\boldsymbol{w}_{ext}]$.
11. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof to show $\Pi_{i\in[B]} v_i = 0$ by using IT-MAC $[\boldsymbol{v}]$. Note that this is a B-product circuit defined over $\mathbb{F}_{p^q}$, so it can be performed with $\{[\tau_i]\}_{i\in[B-1]}$ and LPZK. If ZKP succeeds, $\mathcal{V}$ outputs $(\texttt{true})$; otherwise, $\mathcal{V}$ outputs $(\texttt{false})$.

Figure 5: Robin: ZKP protocol for disjunctive circuit satisfiability over any field $\mathbb{F}_p$ in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model.

For malicious prover $\mathcal{A}$, the witness can be trivially extracted from messages sent to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$. $\mathcal{S}$ runs a proof interaction with $\mathcal{A}$ by acting as honest $\mathcal{V}$, and it sends the extracted witness to $\mathcal{F}^{1,B}_{\mathsf{ZK}}$ if the interaction leads to a successful proof. The only difference between the two worlds occurs when $\mathcal{A}$ successfully proves a false statement; this can occur when $\mathcal{A}$ manages to pass checks built into the protocol. In such cases, real-world $\mathcal{V}$ will accept the proof, whereas ideal-world $\mathcal{V}$ will reject, because $\mathcal{S}$ does not hold a valid witness. This discrepancy occurs with low probability because the protocol is sound.

Indeed, $\mathcal{A}$ must pass all checks, and the probability that checks erroneously pass is bounded by the (statistical) soundness of LPZKs in Steps $8, 11$ ($\frac{n_\times + B + 4}{p^q}$ in total) and by the probability of the following bad event: Let $\boldsymbol{w}_{bad}$ denote a vector that is not an extended witness for any branch. Honest $\mathcal{V}$ samples a vector $\boldsymbol{s}$ such that $\boldsymbol{s} \times \boldsymbol{M}_i \times \boldsymbol{w}_{bad} = 0$ for some $i \in [B]$ in Step 10. Each $\boldsymbol{M}_i \times \boldsymbol{w}_{bad}$ is a non-zero vector, so this only happens with (statistical) probability at most $\frac{B}{p^q}$ (see Corollary 1). $\square$

Protocol cost. In total, $\Pi^{p,q}_{\mathsf{Single}}$ consumes the following resources:

- **Communication.** The parties transmit $n_{in} + (2q+3)n_\times + q(B+7) = \mathcal{O}(q|\mathcal{C}| + qB)$ field elements. We next explain how to adjust $\Pi^{p,q}_{\mathsf{Single}}$ such that the number of transmitted field elements is only $\mathcal{O}(|\mathcal{C}| + qB)$, suitable for small fields.
- **VOLE correlations.** The parties use $n_{in} + 3n_\times + q(B+1) = \mathcal{O}(|\mathcal{C}| + qB)$ subfield VOLE correlations.
- **Rounds.** The protocol runs in 5 rounds.
- **Computation.** Each party uses $\mathcal{O}(|\mathcal{C}|)$ field operations.

Appendix D provides detailed explanation of this cost accounting.

Generating random challenges. $\Pi^{p,q}_{\mathsf{Single}}$ Step 9 requires $\mathcal{V}$ send a random challenge $\boldsymbol{s}$ of size $\mathcal{O}(q|\mathcal{C}|)$ field elements. There are several methods to compress $\boldsymbol{s}$ such that it does not asymptotically dominate; these are standard, see e.g. discussion in [YSWW21]. These methods trade off in soundness, communication, and computation:

Powers of χ. $\mathcal{V}$ can send 1 random field element $\chi \in \mathbb{F}_{p^q}$ and define $\boldsymbol{s}$ as $(1, \chi, \chi^2, \ldots, \chi^{2n_\times})$. This variant uses $\mathcal{O}(|\mathcal{C}| + qB)$ communication and $\mathcal{O}(|\mathcal{C}|)$ computation. While this saves communication, it increases soundness error to $\frac{2Bn_\times + B + n_\times + 4}{p^q}$, because it increases the chance (see Lemma 2) that cheating $\mathcal{P}$ can randomly achieve an IT-MAC encoding of 0 on some branch.

Random Oracle. $\mathcal{V}$ can send a λ-bit seed, and the parties can use a random oracle (RO) to generate $\boldsymbol{s}$. This variant has $\mathcal{O}(|\mathcal{C}| + qB)$ communication, but the parties use computation to expand the RO. The soundness error (with extra random oracle assumption) is now $\frac{t}{2^\lambda} + \frac{n_\times + 2B + 4}{p^q}$, where t denotes an upper bound of the number of RO queries made by the adversary. **We implement this variant.**

6 Batchman: Batched Disjunctions

We refer the reader to Section 3.2 for the intuition of our ZK protocol for batched disjunctive circuit satisfiability. Figure 6 formalizes our protocol; its main security property is as follows:

Theorem 2 (Batched Disjunctions Security). *$\Pi^{p,q}_{\mathsf{Batch}}$ (Figure 6) UC-realizes $\mathcal{F}^{R,B}_{\mathsf{ZK}}$ (Figure 1) in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model with soundness error $\frac{Rn_\times+R+3B+6}{p^q}$ and information-theoretic security.*

We provide a proof of Theorem 2 in Appendix C.4. In short, the proof is very similar to that of Theorem 1, except that we must additionally account for (1) the soundness of QS in Step 13 and (2) an additional bad event made possible by the check on $\mathcal{P}$'s committed topology vectors in Step 15.

Protocol Cost. In total, $\Pi^{p,q}_{\mathsf{Batch}}$consumes the following resources:

- **Communication.** The parties transmit $(Rq + R)n_{in} + (3Rq + 3R)n_\times + qR(B + 1) + 2q = \mathcal{O}(qRB + qR|\mathcal{C}|)$ field elements.
- **VOLE correlations.** The parties use $(Rq + R)n_{in} + (3Rq + 3R)n_\times + qR(B + 1) + 2q = \mathcal{O}(qRB + qR|\mathcal{C}|)$ subfield VOLE correlations.
- **Rounds.** The protocol runs in 7 rounds.
- **Computation.** Each party computes $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$. field operations.

Appendix D provides detailed explanation of this cost accounting.

Field size. Unlike our single disjunction protocol, our batched protocol improves on prior work w.r.t. communication only for large fields. This is because in Step 12, $\mathcal{P}$ commits to R compressed topology vectors, and these are defined over the *extension field*. If one wishes to work with a small field (e.g., Boolean), repeating our single disjunction protocol is more effective w.r.t. communication.

Generating random challenges. As in our single disjunction protocol, we can reduce communication needed for $\mathcal{V}$'s random challenge vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ by applying standard methods. In particular, these challenges can be generated using a two-row Vandermonde matrix of two random field elements, or using a random oracle. **We implement the RO variant.**

Constraining Batch Witnesses. Batched disjunctions allow $\mathcal{P}$ to prove the same disjunction with respect to R witnesses. This is only interesting if we impose additional constraints on $\mathcal{P}$'s witnesses; otherwise, $\mathcal{P}$ with only one witness can trivially re-use her witness R times to satisfy the full statement. In Appendix E we discuss simple methods for extending our protocol with additional constraints that force $\mathcal{P}$ to prove her R witnesses are related.

7 Implementation and Benchmarking

We implemented our ZK protocols for both Boolean circuits (field $\mathbb{F}_2$) and for circuits of the Mersenne prime field $\mathbb{F}_{2^{61}-1}$.

Our implementation extends the publicly available implementation of QuickSilver [YSWW21] (their code is part of the EMP Toolkit [WMK16]). We use their VOLE and LPZK implementations.

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

Inputs. $\mathcal{P}$ and $\mathcal{V}$ agree on B circuits $\mathcal{C}_1, \ldots, \mathcal{C}_B$ over any field $\mathbb{F}_p$, where each circuit has n_{in} inputs and $n_\times$ multiplication gates. $\mathcal{P}$ holds R witnesses $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(R)}$ and R integers $a^{(1)}, \ldots, a^{(R)} \in [B]$ such that for all $j \in [R]$, $\mathcal{C}_{a^{(j)}}(\boldsymbol{w}^{(j)}) = 0$ and $|\boldsymbol{w}^{(j)}| = n_{in}$.

Generate extended witnesses for $\mathcal{C}_{a^{(1)}}, \ldots, \mathcal{C}_{a^{(R)}}$.
0. For each $j \in [R]$, $\mathcal{P}$ evaluates $\mathcal{C}_{a^{(j)}}(\boldsymbol{w}^{(j)})$ in cleartext to generate vectors $\boldsymbol{\ell}^{(j)}, \boldsymbol{r}^{(j)}, \boldsymbol{o}^{(j)} \in \mathbb{F}_p^{n_\times}$, where $\boldsymbol{\ell}^{(j)}$ (resp. $\boldsymbol{r}^{(j)}, \boldsymbol{o}^{(j)}$) denotes the values on the left (resp. right, output) wires of multiplication gates (listed in topological order).

Initialize/Preprocessing.
1. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{init})$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns a uniform $\Delta \in_\$ \mathbb{F}_{p^q}$ to $\mathcal{V}$.
2. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{extend}, R(n_{in} + 3n_\times))$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns IT-MACs $\{[\mu_k^{(j)}]\}_{k\in[n_{in}]}$, $\{[\omega_k^{(j)}]\}_{k\in[n_\times]}$, $\{[\xi_k^{(j)}]\}_{k\in[n_\times]}$ and $\{[\rho_k^{(j)}]\}_{k\in[n_\times]}$ for each $j \in [R]$.
3. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{extend}, qR(n_{in} + 3n_\times + 1))$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns $qR(n_{in} + 3n_\times + 1)$ IT-MACs of random $\mathbb{F}_p$ values. $\mathcal{P}$ and $\mathcal{V}$ combine these IT-MACs into $R(n_{in} + 3n_\times + 1)$ IT-MACs of random $\mathbb{F}_{p^q}$ values, denoted $\{[\eta_k^{(j)}]\}_{k\in[n_{in}+3n_\times+1]}$ for each $j \in [R]$.
4. $\mathcal{P}$ and $\mathcal{V}$ send $(\texttt{extend}, qR(B-1))$ to $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$, which returns $qR(B-1)$ IT-MACs of random $\mathbb{F}_p$ values. $\mathcal{P}$ and $\mathcal{V}$ combine these IT-MACs into $R(B-1)$ IT-MACs of random $\mathbb{F}_{p^q}$ values, denoted $\{[\tau_i^{(j)}]\}_{i\in[B-1]}$ for each $j \in [R]$.

Commit to extended witnesses on $\mathcal{C}_{a^{(1)}}, \ldots, \mathcal{C}_{a^{(R)}}$. For each $j \in [R]$, proceed as follows:

5. For $k \in [n_{in}]$, $\mathcal{P}$ sends $\delta_k := w_k^{(j)} - \mu_k^{(j)} \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[w_k^{(j)}] := [\mu_k^{(j)}] + \delta_k$.
6. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := \ell_k^{(j)} - \omega_k^{(j)} \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[\ell_k^{(j)}] := [\omega_k^{(j)}] + \delta_k$.
7. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := r_k^{(j)} - \xi_k^{(j)} \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[r_k^{(j)}] := [\xi_k^{(j)}] + \delta_k$.
8. For $k \in [n_\times]$, $\mathcal{P}$ sends $\delta_k := o_k^{(j)} - \rho_k^{(j)} \in \mathbb{F}_p$ to $\mathcal{V}$, and then both compute $[o_k^{(j)}] := [\rho_k^{(j)}] + \delta_k$.

Check multiplication gates. $\mathcal{P}$ convinces $\mathcal{V}$ that the $Rn_\times$ committed multiplication gates are well-formed.
9. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof for (batched) multiplications $\mathsf{LPZK}(\{[\ell_k^{(j)}], [r_k^{(j)}], [o_k^{(j)}]\}_{k\in[n_\times], j\in[R]})$. If ZKP fails, $\mathcal{V}$ outputs $(\texttt{false})$ and halts.

Generate compressed topology vectors. Let $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_B \in \mathbb{F}_p^{(2n_\times+1)\times(n_{in}+3n_\times+1)}$ denote the topology matrices of $\mathcal{C}_1, \ldots, \mathcal{C}_B$.
10. $\mathcal{V}$ samples a random vector $\boldsymbol{s} \in_\$ \mathbb{F}_{p^q}^{2n_\times+1}$ and sends it to $\mathcal{P}$.
11. For each $i \in [B]$, $\mathcal{P}$ and $\mathcal{V}$ compute $\boldsymbol{cv}_i := (\boldsymbol{s}^T \times \boldsymbol{M}_i)^T \in \mathbb{F}_{p^q}^{n_{in}+3n_\times+1}$.

Commit compressed topology vectors. For each $j \in [R]$:

12. For each $k \in [n_{in}+3n_\times+1]$, $\mathcal{P}$ sends $\delta_k := (cv_{a^{(j)}})_k - \eta_k^{(j)} \in \mathbb{F}_{p^q}$ to $\mathcal{V}$, and then both compute $[\widetilde{cv_k^{(j)}}] := [\eta_k^{(j)}] + \delta_k$.

Check satisfiability of committed compressed topology vectors. For each $j \in [R]$, Let $\boldsymbol{w}_{ext}^{(j)} \triangleq \mathsf{glue}(\boldsymbol{w}^{(j)}, \boldsymbol{\ell}^{(j)}, \boldsymbol{r}^{(j)}, \boldsymbol{o}^{(j)}, 1) \in \mathbb{F}_p^{n_{in}+3n_\times+1}$ and associated IT-MAC $[\boldsymbol{w}_{ext}^{(j)}] \triangleq \mathsf{glue}([\boldsymbol{w}^{(j)}], [\boldsymbol{\ell}^{(j)}], [\boldsymbol{r}^{(j)}], [\boldsymbol{o}^{(j)}], [1])$. $\mathcal{P}$ convinces $\mathcal{V}$ that $(\widetilde{\boldsymbol{cv}^{(j)}})^T \times \boldsymbol{w}_{ext}^{(j)} = 0$ for each $j \in [R]$. I.e., the committed circuits are satisfied.
13. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof for (batched) inner-products $\mathsf{QS}(\{[\widetilde{\boldsymbol{cv}^{(j)}}], [\boldsymbol{w}_{ext}^{(j)}]\})_{j\in[R]}$. If ZKP fails, $\mathcal{V}$ outputs $(\texttt{false})$ and halts.

Validate committed compressed topology vectors. $\mathcal{P}$ convinces $\mathcal{V}$ that $\widetilde{\boldsymbol{cv}^{(j)}} \in \{\boldsymbol{cv}_1, \ldots, \boldsymbol{cv}_B\}$ for each $j \in [R]$. I.e., the committed circuits are well-formed.
14. $\mathcal{V}$ samples a random vector $\boldsymbol{t} \in_\$ \mathbb{F}_{p^q}^{n_{in}+3n_\times+1}$ and sends it to $\mathcal{P}$. For each $i \in [B]$, $\mathcal{P}$ and $\mathcal{V}$ compute $ct_i := \boldsymbol{cv}_i^T \times \boldsymbol{t} \in \mathbb{F}_{p^q}$. For each $j \in [R]$, $\mathcal{P}$ and $\mathcal{V}$ compute IT-MAC $[\widetilde{ct^{(j)}}] := [\widetilde{\boldsymbol{cv}^{(j)}}]^T \times \boldsymbol{t}$.
15. For each $j \in [R]$, $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof to show $\Pi_{i\in[B]}\{\widetilde{ct^{(j)}} - ct_i\} = 0$ by using IT-MAC $[\widetilde{ct^{(j)}}]$ and public $\{ct_i\}_{i\in[B]}$. Note that this is a B-product circuit defined over $\mathbb{F}_{p^q}$ so can be performed with $\{[\tau_i^{(j)}]\}_{i\in[B-1]}$ and LPZK. If all R ZKPs succeed, $\mathcal{V}$ outputs $(\texttt{true})$; otherwise, $\mathcal{V}$ outputs $(\texttt{false})$.

Figure 6: Batchman: ZKP protocol for batched disjunctive circuit satisfiability over any field $\mathbb{F}_p$ in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model.

Our implementations achieve computational security parameter $\kappa = 128$ (for VOLE) and statistical security parameter $\lambda \geq 100$ for Boolean and $\lambda \geq 40$ for arithmetic circuits, matching QuickSilver.

Unless otherwise specified, our experiments were run on two Amazon EC2 m5.2xlarge machines[8] (respectively implementing $\mathcal{P}$ and $\mathcal{V}$). Our implementations run single threaded. Our implementation is publicly available at `https://github.com/gconeice/stacking-vole-zk`.

Benchmark. Unless otherwise specified, our experiments use a benchmark where each of the B branches features a matrix multiplication (implementing the naïve algorithm) where $\mathcal{P}$ wishes to prove that she knows two square $\ell \times \ell$ matrices whose product is equal to a public $\ell \times \ell$ matrix. Each such circuit has $\mathcal{O}(\ell^3)$ gates. We acknowledge that this benchmark is contrived; its purpose is to evaluate performance only.

7.1 Robin: Single Disjunction Protocol

7.1.1 Comparison with Mac′n′Cheese [BMRS21]

We compare Robin with the prior state-of-the-art VOLE-based ZK protocol supporting disjunctions: Mac′n′Cheese [BMRS21]. The Mac′n′Cheese implementation is not publicly available, so we use the numbers available in their paper.

[BMRS21] reported execution time when handling B branches, each consisting of ≈ 1 billion AND gates. Each branch computes 45000 iterations of the SHA-2 circuit.[9] For these large Boolean branches, [BMRS21] uses an elegant trick based on [BBC+19] to reduce the communication cost of each AND gate to only 1 bit (rather than paying two extension fields communication per AND gate); this trick increases round complexity by factor $\mathcal{O}(\log |\mathcal{C}|)$.

We ran Robin on the same branches and the same network configuration. Due to size, we ran our experiment on two Amazon EC2 m5.8xlarge machines[10]. Figure 7 tabulates the results.

[BMRS21]'s implementation does not include a VOLE backend, and it only achieves 40 bit statistical security. Our implementation includes a real VOLE backend with 100 bit statistical security. Because of these differences, it is difficult to present a completely fair comparison. Despite generating real VOLE correlations, Robin still improves performance. Figure 7 shows that we pay ≈ 25 seconds per *extra* branch, whereas [BMRS21] uses ≈ 150 seconds.

Figure 7 also tabulates the results for branches with matrix multiplication. This additional column demonstrates that our performance does not depend on branch structure.

7.1.2 Comparison with QuickSilver [YSWW21]

[YSWW21] is a state-of-the-art VOLE-based ZK protocol for Boolean/arithmetic circuits. It uses $\mathcal{O}(B|\mathcal{C}|)$ computation and communication with low constants, and it serves as the baseline for our approach. We compare our single disjunction protocol Robin with [YSWW21] on circuits defined over $\mathbb{F}_{2^{61}-1}$. Asymptotically, Robin improves communication from $\mathcal{O}(B|\mathcal{C}|)$ to $\mathcal{O}(B + |\mathcal{C}|)$.

[8] Intel Xeon Platinum 8175 CPU @ 3.10GHz, 8 vCPUs, 32GiB Memory, 10Gbps Network

[9] More precisely, they also consider a branch that computes 150000 iterations of AES, but this branch is smaller than the SHA-2 circuit.

[10] Intel Xeon Platinum 8175 CPU @ 3.10GHz, 32 vCPUs, 128GiB Memory, 10Gbps Network

# Branch	Mac'n'Cheese [BMRS21] 50 threads, $\lambda \geq 40$, *without* VOLE	Robin 1 thread, $\lambda \geq 100$, *with* VOLE	
	Rep. SHA2	Rep. SHA2	Mat. Mul.
2	307	468	466
4	568	520	517
8	1254	615	617
16	-	812	816
32	-	1209	1213
64	-	2004	2005

Figure 7: Comparison with Mac'n'Cheese [BMRS21]. We tabulate end-to-end runtime in seconds. Our reported numbers for [BMRS21] are directly from their paper. Rep. SHA2 denotes a circuit computing 45000 iterations of SHA-2. Mat. Mul denotes a circuit that multiplies two 1000×1000 Boolean matrices. Both circuits have $\approx$ 1 billion AND gates. [BMRS21] uses 124 MB of communication while ours uses 628 MB. As B increases, communication remains almost constant. The network has 30 Mbps bandwidth and 100 ms latency.

We compare using branches that each have 8 million multiplication gates, and we vary B between 5 and 100. Figure 8 plots our speedup. [YSWW21] requires 73.7 MB communication per branch; Robin requires $\approx$ 200 MB communication for *all* branches.

When network bandwidth is low (e.g., 100 Mbps), communication remains the bottleneck, and for $B > 40$ Robin achieves over $10\times$ improvement. Even when network bandwidth is high (e.g., 500 Mbps), Robin improves performance by $\approx 4\times$, because Robin computes fewer VOLE correlations.

7.1.3 More evaluation

Appendix F includes further evaluation.

7.2 Batchman: Batched Disjunctions Protocol

Our batched protocol Batchman is best for circuits over large fields. Therefore, our evaluation considers circuits over $\mathbb{F}_{2^{61}-1}$.

7.2.1 Comparison with AntMan [WYY+22]

AntMan [WYY+22] presents a protocol optimized for circuits with batched SIMD circuits, but AntMan does not consider batched disjunctions. To implement batched disjunctions in AntMan, one can consider a size $B|\mathcal{C}|$ instruction which is executed R times. Recall that AntMan incurs $\mathcal{O}(RB|\mathcal{C}|\log R)$ computation and $\mathcal{O}(B|\mathcal{C}| + R)$ communication. Our batched protocol improves in terms of computation, incurring $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$ computation and $\mathcal{O}(RB + R|\mathcal{C}|)$ communication.

The AntMan implementation is not publicly available, so we use numbers from the paper. To compare, we ran experiments on the same setup: two Amazon EC2 m5.8xlarge[10] machines. [WYY+22] reported the execution of a batch of 1024 circuits where each circuit has 2^{21} multiplication gates. Accordingly, we tested Batchman to ensure all branches in each repetition have 2^{21}

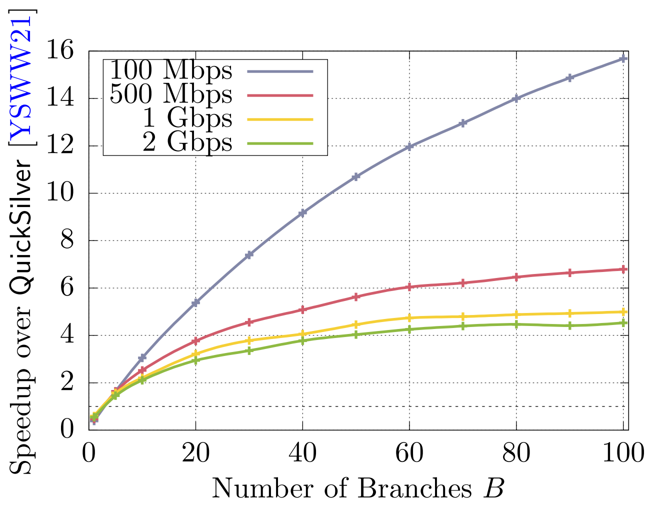


Figure 8: The speedup of our single disjunction protocol Robin over QuickSilver [YSWW21]. We report end-to-end proof runtime. Circuits are over $\mathbb{F}_{2^{61}-1}$; each branch has 8M mult. gates.

Protocol	Network Bandwidth			
	50 Mbps	100 Mbps	500 Mbps	1 Gbps
AntMan-1	2.00	2.05	2.08	2.09
AntMan-2	3.78	3.91	3.99	4.26
AntMan-4	6.88	6.69	6.99	7.01
Batchman-$(2^3, 2^{18})$	0.96	1.83	6.60	9.60
Batchman-$(2^6, 2^{15})$	7.55	14.43	51.28	75.40
Batchman-$(2^9, 2^{12})$	56.20	104.91	335.02	461.82

Figure 9: Comparison with AntMan [WYY+22]. We tabulate millions of multiplication gates executed per second (mgps). AntMan-t refers to AntMan with t threads (numbers from [WYY+22]). Batchman uses only 1 thread. Batchman-(B, C) refers to our batched protocol Batchman with B branches where each branch has C multiplication gates. Both protocols execute batches where each repetition has 2^{21} multiplication gates.

total multiplication gates. ([WYY+22] circuits are defined over $\mathbb{F}_{2^{59}-2^{28}+1}$.) Figure 9 tabulates the results; higher numbers are better.

Batchman is sensitive to network bandwidth due to its $\mathcal{O}(R|\mathcal{C}|)$ asymptotic scaling, but it is computation efficient. As B increases, our improvement also increases. In the extreme case where there are 512 branches and with 1 Gbps bandwidth, Batchman is 221× faster than (single thread) AntMan [WYY+22].

Of course, AntMan solves a more general problem than Batchman. However, for our special-case problem of batched disjunctions, we demonstrate significant improvement.

7.2.2 Comparison with QuickSilver [YSWW21] and Robin

We compare Batchman to the baseline QuickSilver [YSWW21] protocol and to repeated runs of Robin. We experiment with benchmarks satisfying $R = B$, and we consider branches with 1.25×10^5

multiplication gates. Figure 10 plots speedup as compared to QuickSilver.

Compared to QuickSilver, Robin only improves communication, limiting its speedup. On the other hand, Batchman improves both communication and computation, and our speedup is almost *independent* of network bandwidth. Our experiment shows that Batchman enjoys an *extra* $2 - 9\times$ improvement as compared to Robin.

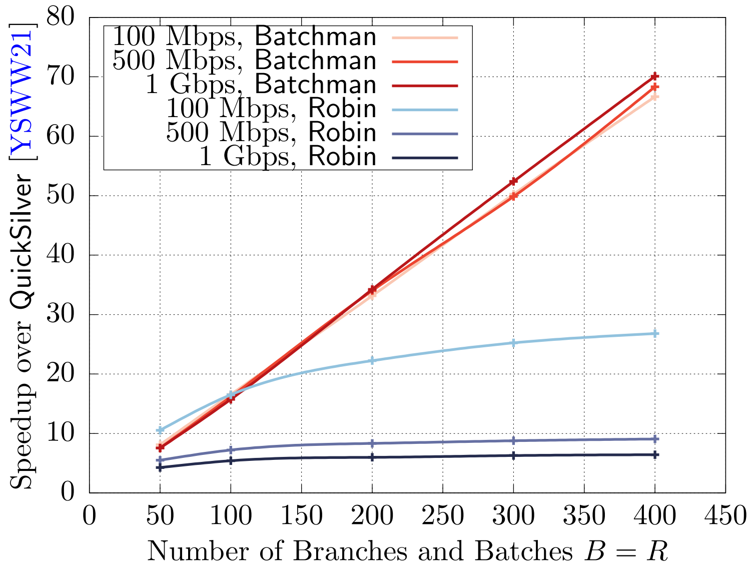


Figure 10: The speedup of Batchman and Robin over QuickSilver [YSWW21]. We plot factor improvement in terms of end-to-end runtime. Circuits are defined over $\mathbb{F}_{2^{61}-1}$ and each branch has 1.25×10^5 multiplication gates.

7.2.3 Fine-grained analysis

Figure 11 breaks down the runtime cost of Batchman. Most of the execution time is spent committing to the witness and to the compressed topology vectors. Figure 11 confirms $\mathsf{MUL}_{\mathsf{LEFT}}$'s high concrete efficiency.

7.2.4 CPU emulation benchmark

Our final benchmark shows that Batchman is suitable to the use-case of CPU-emulation-based ZK. We consider a proof-of-concept CPU (without RAM) with $B = 50$ instructions where each instruction has 125 multiplication gates. We vary R between 50K and 500K (guided by ZEE [HYDK21]) and calculate average CPU speed. While ZEE achieves a comparable Hz rate, it has a smaller branching factor $B = 20$, and, crucially, our CPU step is vastly more powerful in that it executes 125 multiplications per instruction, vs a single one in ZEE.

As shown in Figure 12, Batchman achieves $9\times$ improvement as compared to QuickSilver [YSWW21]. We note that this is purely a proof of concept. To implement true CPU emulation based on Batchman, one needs to carefully design the instruction set, and ZK RAM (e.g, [FKL+21, DdSGOTV22]) needs to be incorporated.

Bandwidth	B	R	multi. check		topo. check		topo. valid
				$\mathsf{MUL}_{\mathsf{LEFT}}$	commit topo.	inner-prod.	
100 Mbps	50	50	14.6	0.1	13.5	0.2	0.2
	100	100	28.1	0.2	26.9	0.3	0.4
	400	400	109.7	0.8	107.6	0.9	2.3
500 Mbps	50	50	4.8	0.1	3.6	0.1	0.2
	100	100	8.4	0.2	7.3	0.2	0.4
	400	400	30.4	0.8	28.7	0.7	2.2
1 Gbps	50	50	3.4	0.1	2.4	0.1	0.2
	100	100	6.2	0.2	4.9	0.2	0.4
	400	400	20.3	0.8	18.6	0.7	2.2

Figure 11: Fine-grained analysis of our batched disjunctions protocol Batchman. Measurements are in seconds.

Protocol	Network Bandwidth		
	100 Mbps	500 Mbps	1 Gbps
QuickSilver [YSWW21]	181 Hz	625 Hz	902 Hz
Batchman	1525 Hz	5375 Hz	7891 Hz

Figure 12: CPU speed in a proof-of-concept setting. We consider a CPU with $B = 50$ instructions; each instruction is an arithmetic circuit with 125 multiplication gates.

Acknowledgments

This work is supported in part by Cisco research award and NSF awards CNS-2246353, CNS-2246354, and CCF-2217070. This material is also based upon work supported in part by DARPA under Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

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

[BBMH+21] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, Benoît Razet, and Peter Scholl. Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z2k. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 192–211. ACM Press, November 2021.

[BBMHS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2022.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG$^+$18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Heidelberg, December 2018.

[BCG$^+$19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

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

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

[BFR$^+$13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 341–357, New York, NY, USA, 2013. Association for Computing Machinery.

[BHR$^+$20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 168–197. Springer, Heidelberg, November 2020.

[BHR$^+$21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 123–152, Virtual Event, August 2021. Springer, Heidelberg.

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Heidelberg.

[DdSGOTV22] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Titouan Tanguy, and Michiel Verbauwhede. Efficient proof of ram programs from any public-coin zero-knowledge system. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks*, pages 615–638, Cham, 2022. Springer International Publishing.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 829–841. ACM Press, November 2022.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-Point Zero Knowledge and Its Applications. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*, volume 199 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:24, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FKL$^+$21] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-overhead zero-knowledge for RAM programs. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 178–191. ACM Press, November 2021.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.

[GF95] Anwar M. Ghuloum and Allan L. Fisher. Flattening and parallelizing irregular, recurrent loop nests. *SIGPLAN Not.*, 30(8):58–67, aug 1995.

[GGHAK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In Orr

Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 458–487. Springer, Heidelberg, May / June 2022.

[GHAKS23] Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner. Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 347–378. Springer, Heidelberg, April 2023.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[HK20a] David Heath and Vladimir Kolesnikov. A 2.1 KHz zero-knowledge processor with BubbleRAM. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2055–2074. ACM Press, November 2020.

[HK20b] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, Heidelberg, May 2020.

[HK21] David Heath and Vladimir Kolesnikov. PrORAM - fast $P(\log n)$ authenticated shares ZK ORAM. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 495–525. Springer, Heidelberg, December 2021.

[HMR15] Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. Efficient zero-knowledge proofs of non-algebraic statements with sublinear amortized cost. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 150–169. Springer, Heidelberg, August 2015.

[HYDK21] David Heath, Yibin Yang, David Devecsery, and Vladimir Kolesnikov. Zero knowledge for everything and everyone: Fast ZK processor with cached ORAM for ANSI C programs. In *2021 IEEE Symposium on Security and Privacy*, pages 1538–1556. IEEE Computer Society Press, May 2021.

[Kol18] Vladimir Kolesnikov. Free IF: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 34–58. Springer, Heidelberg, December 2018.

[LAH$^+$22] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. Proving UNSAT in zero knowledge. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2203–2217. ACM Press, November 2022.

[MRS17] Payman Mohassel, Mike Rosulek, and Alessandra Scafuro. Sublinear zero-knowledge arguments for RAM programs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 501–531. Springer, Heidelberg, April / May 2017.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[WSR+15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS 2015*. The Internet Society, February 2015.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 501–518. USENIX Association, August 2021.

[WYY+22] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2901–2914. ACM Press, November 2022.

[YH24] Yibin Yang and David Heath. Two shuffles make a RAM: improved constant overhead zero knowledge RAM. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[YHKD22] Yibin Yang, David Heath, Vladimir Kolesnikov, and David Devecsery. 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*, pages 109–123, Genoa, Italy, 2022. IEEE.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

SUPPLEMENTARY MATERIAL

A Additional Preliminaries

A.1 Universal Composability

The Universal Composability (UC) framework [Can01] models the execution of protocols as interactions of probabilistic polynomial-time (PPT) Interactive Turing Machines (ITMs). We use UC to formalize our protocols and to prove security in the presence of a malicious, static adversary.

In UC, the execution of a protocol Π with hybrid functionality $\mathcal{G}$ can be viewed as the interaction between ITMs (as parties), where each ITM can interact with each other and the hybrid functionality $\mathcal{G}$. An ideal execution in the UC framework can be viewed as the interaction between dummy ITMs (as parties) that simply forward messages between the ideal functionality and an ITM referred to as the environment $\mathcal{E}$. The environment $\mathcal{E}$ provides inputs to the parties and can corrupt a subset of the parties by replacing their ITMs with arbitrary ITMs $\mathcal{A}$. The correctness and security of the protocol Π are ensured by the standard ideal-real paradigm where the crucial argument is that $\mathcal{E}$ cannot distinguish between the two worlds. Namely, the protocol Π is "as secure as" the ideal.

Formally, let Π be a protocol in the $\mathcal{G}$-hybrid model. The output of an environment $\mathcal{E}$ interacting with Π in the presence of an adversary $\mathcal{A}$ on input 1^λ and auxiliary input z is denoted as $\mathrm{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}}(1^\lambda, z)$. Consider another (trivial) protocol with ideal functionality $\mathcal{F}$, dummy parties, and a simulator $\mathcal{S}$. The output of an environment $\mathcal{E}$ interacting with this trivial protocol in the presence of a simulator $\mathcal{S}$ on input 1^λ and auxiliary input z is denoted as $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}(1^\lambda, z)$.

Definition 2. *We say that a protocol Π in the $\mathcal{G}$-hybrid model UC-emulates an ideal functionality $\mathcal{F}$ when, for any PPT adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$:*

$$\{\mathrm{EXEC}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}}(1^\lambda, z)\}_{\substack{\lambda\in\mathbb{N},\\ z\in\{0,1\}^*}} \overset{c}{\approx} \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}(1^\lambda, z)\}_{\substack{\lambda\in\mathbb{N},\\ z\in\{0,1\}^*}}$$

where "$\overset{c}{\approx}$" denotes computational indistinguishability of distribution ensembles, see [Gol09].

The definition requires computational indistinguishability; our protocols achieve stronger statistical indistinguishability.

A.2 LPZK [DIO21] and its Inner-product Generalization [YSWW21]

We provide more details of the sub-protocols LPZK and QS used in our approach.

Line-point zero-knowledge. The goal of LPZK is to allow $\mathcal{P}$ to convince $\mathcal{V}$ (in ZK) that a collection of IT-MAC tuples each satisfy the multiplication relation. Consider three IT-MACs $[x]_\Delta$, $[y]_\Delta$, $[z]_\Delta$ over a field $\mathbb{F}$ where $\mathcal{P}$ holds x, y, z, m_x, m_y, m_z and $\mathcal{V}$ holds m_x, m_y, m_z, Δ. By definition, $m_x = k_x - x\Delta$, $m_y = k_y - y\Delta$ and $m_z = k_z - z\Delta$. The key insight unlerlying LPZK is the following equation:

$$\begin{aligned} &k_x k_y - k_z\Delta \\ =&(m_x + x\Delta)(m_y + y\Delta) - (m_z + z\Delta)\Delta \\ =&(xy - z)\Delta^2 + (m_x y + m_y x - m_z)\Delta + m_x m_y \end{aligned}$$

Note, $\mathcal{P}$ knows $(m_x y + m_y x - m_z)$ and $m_x m_y$, and $\mathcal{V}$ knows $k_x k_y - k_z \Delta$. If we for now ignore Zero Knowledge, we can observe that if $\mathcal{P}$ sends these two coefficients to $\mathcal{V}$, this is convincing evidence that the coefficient of Δ^2 (i.e., $xy - z$) is 0. This proves that the committed values indeed satisfy $z = xy$, because if cheating $\mathcal{P}$ commits to $z \neq xy$, since $\mathcal{P}$ does not know Δ, she will succeed with probability at most $2/|\mathbb{F}|$.

The two transmitted coefficients are over exstension fields, not subfields. To achieve amortized 1 transmitted subfield element per multiplication, we must compress the coefficients of T multiplication tuples into 1. Specifically, suppose $\mathcal{P}$ holds T coefficient pairs $(A_{1,0}, A_{1,1}), \ldots (A_{T,0}, A_{T,1})$ and $\mathcal{V}$ holds T field elements $C_1, \ldots, C_T$ which are computed locally from T IT-MAC tuples which supposedly satisfy the multiplication relation. For each $i \in [T]$, $A_{i,1}\Delta + A_{i,0} = C_i$. To check the batch of multiplications are well formed, $\mathcal{V}$ sends a random challenge χ, and $\mathcal{P}$ sends two coefficients $A_0 \triangleq \sum_{i=1}^{T} A_{i,0}\chi^{i-1}$ and $A_1 \triangleq \sum_{i=1}^{T} A_{i,1}\chi^{i-1}$ to $\mathcal{V}$. $\mathcal{V}$ checks $A_1\Delta + A_0 = \sum_{i=1}^{T} C_i \chi^{i-1}$, which is convincing evidence that all T multiplications are well formed.

It is easy to reintroduce Zero Knowledge: $\mathcal{P}$ simply masks her coefficients by sending $A_1 + R_1$ and $A_0 + R_0$ where $\mathcal{V}$ knows $R_1\Delta + R_0$ and where R_1 and R_0 are selected randomly, which is just a single additional VOLE correlation. This protocol achieves soundness $(T+2)/|\mathbb{F}|$ and information-theoretic security. See [YSWW21] for details.

Line-point zero-knowledge for inner-products. Recall that the goal of QS is to allow $\mathcal{P}$ to convince $\mathcal{V}$ (in ZK) that two vectors of IT-MACs are formed such that the inner-product of the committed vectors is 0. Consider two length-n IT-MAC vectors $[\boldsymbol{x}]_\Delta$ and $[\boldsymbol{y}]_\Delta$ over field $\mathbb{F}$. For each $[x_{i\in[n]}]_\Delta$ (resp. $[y_{i\in[n]}]_\Delta$), $\mathcal{P}$ holds x_i, m_{x_i} (resp. y_i, m_{y_i}), and $\mathcal{V}$ holds k_{x_i} (resp. k_{y_i}) and Δ where $m_{x_i} = k_{x_i} - x_i\Delta$ (resp. $m_{y_i} = k_{y_i} - y_i\Delta$). The key insight of QS is the following equation:

$$\begin{aligned}\sum_{i=1}^{n} k_{x_i}k_{y_i} &= \sum_{i=1}^{n} (m_{x_i} + x_i\Delta)(m_{y_i} + y_i\Delta) \\ &= \sum_{i=1}^{n} m_{x_i}m_{y_i} + \sum_{i=1}^{n} (m_{x_i}y_i + m_{y_i}x_i)\Delta + \sum_{i=1}^{n} x_i y_i \Delta^2\end{aligned}$$

Similar to LPZK, $\mathcal{P}$ knows $\sum_{i=1}^{n}(m_{x_i}y_i + m_{y_i}x_i)$ and $\sum_{i=1}^{n} m_{x_i}m_{y_i}$ while $\mathcal{V}$ knows $\sum_{i=1}^{n} k_{x_i}k_{y_i}$. Thus, if we for now ignore Zero Knowledge, $\mathcal{P}$ can send these two coefficients to convince $\mathcal{V}$ that the coefficient of Δ^2 (i.e., $\sum_{i=1}^{n} x_i y_i$) is 0, which is exactly the inner-product of these two vectors. Cheating $\mathcal{P}$ holding $\sum_{i=1}^{n} x_i y_i \neq 0$ will pass this check with probability at most $2/|\mathbb{F}|$.

Just as in LPZK, we can ccompress T inner-product proofs into only two coefficients, improving communication. Furthermore, Zero Knowledge can be added by using one extra random VOLE correlation. This protocol achieves soundness $(T+2)/|\mathbb{F}|$ with information-theoretic security. This technique can be further generalized to allow proofs with respect to arbitrary polynomials. See [YSWW21] for details.

B Circuit Gate-by-gate Representation

We include a standard definition of the gate-by-gate representation of arithmetic circuits. Our definition includes offset gates, each of which adds a public constant to its input.

Definition 3 (Circuit Gate-by-Gate Representation). *A circuit $\mathcal{C}$ over some field $\mathbb{F} = (+, \cdot)$ is a tuple consisting of:*

- $n_{in}, n_\times, n_+, n_{scale}, n_{offset}$, *which respectively denote the number of input gates, multiplication gates, addition gates, scale gates, and offset gates.*
- $\mathsf{wid} \triangleq [m]$ *denotes the collection of wire identifiers; the last identifier m is identifies the output wire.*
- $\mathsf{IN} \triangleq \{in_k\}_{k\in[n_{in}]}$ *denotes the input wires.*
- $\mathsf{MUL} \triangleq \{(\ell_k, r_k, o_k)\}_{k\in[n_\times]}$ *denotes the multiplication gates where $\ell_k, r_k, o_k \in [m]$.*
- $\mathsf{ADD} \triangleq \{(\ell'_k, r'_k, o'_k)\}_{k\in[n_+]}$ *denotes the addition gates where $\ell'_k, r'_k, o'_k \in [m]$.*
- $\mathsf{SCALE} \triangleq \{(c_k, x_k, y_k)\}_{k\in[n_{scale}]}$ *denotes the scale gates where $c_k \in \mathbb{F}$ and $x_k, y_k \in [m]$.*
- $\mathsf{OFFSET} \triangleq \{(c'_k, x'_k, y'_k)\}_{k\in[n_{offset}]}$ *denotes the offset gates where $c'_k \in \mathbb{F}$ and $x'_k, y'_k \in [m]$.*

Theorem 3 (Circuit Satisfiability)**.** *A circuit $\mathcal{C}$ over some field $\mathbb{F} = (+, \cdot)$ represented gate-by-gate (i.e., Definition 3) is* satisfiable *if and only if there exists a vector $\boldsymbol{w} \in \mathbb{F}^m$ such that (1) $\boldsymbol{w}_m = 0$; (2) $\forall k \in [n_\times], \boldsymbol{w}_{l_k} \cdot \boldsymbol{w}_{r_k} = \boldsymbol{w}_{o_k}$; (3) $\forall k \in [n_+], \boldsymbol{w}_{l'_k} + \boldsymbol{w}_{r'_k} = \boldsymbol{w}_{o'_k}$; (4) $\forall k \in [n_{scale}], \boldsymbol{w}_{y_k} = c_k \cdot \boldsymbol{w}_{x_k}$ and (5) $\forall k \in [n_{offset}], \boldsymbol{w}_{y'_k} = \boldsymbol{w}_{x'_k} + c'_k$.*

Recall the definition of topology matrices in Section 4. The satisfiability of a circuit can be alternatively stated using its topology matrix:

Theorem 4 (Circuit Satisfiability from Topology Matrix Multiplication)**.** *Let $\mathcal{C}$ denote a circuit over some field $\mathbb{F}$, and let $\boldsymbol{M}$ denote $\mathcal{C}$'s associated topology matrix. $\mathcal{C}$ is* satisfiable *if and only if there exists a vector $\boldsymbol{w} \in \mathbb{F}^{n_{in}+3n_\times+1}$ such that:*

1. $\boldsymbol{w}_{n_{in}+3n_\times+1} = 1$,
2. $\boldsymbol{w}[n_{in}+1 : n_{in}+n_\times] \circ \boldsymbol{w}[n_{in}+n_\times+1 : n_{in}+2n_\times] = \boldsymbol{w}[n_{in}+2n_\times+1 : n_{in}+3n_\times]$ *where $\circ$ denotes the element-wise product,*
3. $\boldsymbol{M} \times \boldsymbol{w} = \boldsymbol{0}$.

Proof. Immediate from Definition 1. □

C Deferred Proofs

C.1 Proof of Lemma 1

Lemma 1. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$. The following holds:*

$$\Pr[\exists i \in [k], (\boldsymbol{x}^{(i)})^T \times \boldsymbol{s} = 0 \mid \boldsymbol{s} \in_\$ \mathbb{F}^m] \leq k/|\mathbb{F}|$$

Proof. For each non-zero vector $\boldsymbol{x}^{(i\in[k])}$, there is $|\mathbb{F}|^{m-1}$ different $\boldsymbol{s}$ such that $(\boldsymbol{x}^{(i)})^T \times \boldsymbol{s} = 0$. Thus, there will be at most $k|\mathbb{F}|^{m-1}$ choices of $\boldsymbol{s}$ to make the above event happen, which implies the upper bound as $k|\mathbb{F}|^{m-1}|/|\mathbb{F}|^m = k/|\mathbb{F}|$. □

C.2 Proof of Lemma 2

Lemma 2. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$. The following holds:*

$$\Pr[\exists i \in [k], (\boldsymbol{x}^{(i)})^T \times \boldsymbol{s} = 0 \mid \chi \in_{\$} \mathbb{F}] \leq k(m-1)/|\mathbb{F}|$$

where $\boldsymbol{s} \triangleq (1, \chi, \ldots, \chi^{m-1})$.

Proof. $(\boldsymbol{x}^{(i)})^T \times \boldsymbol{s}$ can be viewed as evaluating *non-zero* polynomial $f^{(i)}(X) \triangleq \sum_{j=0}^{m-1} x_{j+1}^{(i)} X^j$ at $X = \chi$. Thus, the above event happens if and only if χ is a root of some polynomial. Since each polynomial will have at most $m-1$ roots, k polynomials in total will have at most $k(m-1)$ roots, resulting in the above bound. $\square$

C.3 Proof of Theorem 1

Theorem 1. *$\Pi^{p,q}_{\mathsf{Single}}$ (Figure 5) UC-realizes $\mathcal{F}^{1,B}_{\mathsf{ZK}}$ (Figure 1) in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model with soundness error $\frac{n_\times+2B+4}{p^q}$ and information-theoretic security.*

Proof. By constructing a simulator $\mathcal{S}$. $\mathcal{S}$ interacts with $\mathcal{F}^{1,B}_{\mathsf{ZK}}$ and runs the adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$ for $\mathcal{A}$.

Malicious $\mathcal{P}$. Since $\mathcal{S}$ emulates $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$ for $\mathcal{A}$, the values committed by IT-MAC in Steps $4, 5, 6, 7$ can be trivially extracted by $\mathcal{S}$, and these committed values include $\mathcal{A}$'s witness $\tilde{\boldsymbol{w}}$ for the active circuit. Note, $\mathcal{S}$ still must extract $\tilde{a}$, which denotes the active circuit identifier, but this can be trivially extracted by trying computing $\mathcal{C}_i(\tilde{\boldsymbol{w}})$ for each i, and finding a value i where the result is 0 (recall, a zero output indicates a successful proof).

$\mathcal{S}$ simply acts as honest $\mathcal{V}$ and emulates $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$. It uses the $\mathcal{A}$'s VOLE inputs to extract $\tilde{\boldsymbol{w}}$ and $\tilde{a}$ ($\tilde{a}$ can be $\perp$). If honest $\mathcal{V}$ would output (`false`), $\mathcal{S}$ sends $\perp$ to $\mathcal{F}^{1,B}_{\mathsf{ZK}}$; otherwise, $\mathcal{S}$ sends $(\mathcal{C}_1, \ldots, \mathcal{C}_B, \tilde{\boldsymbol{w}}, \tilde{a})$ to $\mathcal{F}^{1,B}_{\mathsf{ZK}}$.

We argue $\mathcal{S}$ is a valid simulator. Note that $\mathcal{V}$ in $\Pi^{p,q}_{\mathsf{Single}}$ only sends uniformly random values to $\mathcal{P}$, so $\mathcal{A}$'s real-world view is *identical* to its ideal-world view. Whenever real-world honest $\mathcal{V}$ would output (`false`), ideal-world $\mathcal{V}$ also outputs (`false`) since $\mathcal{S}$ will send $\perp$ to $\mathcal{F}^{1,B}_{\mathsf{ZK}}$.

There is one event we must specially account for. Suppose that $\mathcal{A}$ that does not hold valid $\tilde{\boldsymbol{w}}$, but it nevertheless manages to make real world $\mathcal{V}$ output (`true`). In this case and in the ideal world, $\mathcal{S}$ emulates accepting honest $\mathcal{V}$, but it does not hold a valid witness, since $\mathcal{A}$ does not hold a valid witness. Thus, $\mathcal{S}$ will send an invalid input to $\mathcal{F}^{1,B}_{\mathsf{ZK}}$, which will make ideal-world $\mathcal{V}$ output (`false`). While this event is possible, it is unlikely. Indeed, the probability of this event is precisely the protocol soundness error.

We calculate a bound on soundness error. To make real-world $\mathcal{V}$ output (`true`), a malicious $\mathcal{P}$ must pass two checks:

1. The multiplication check performed by LPZK in Step 8.
2. The circuit check performed by issuing a random challenge and regular VOLE-based ZK in Steps $9, 10, 11$.

If $\mathcal{A}$'s witness does not satisfy the multiplication relation, $\mathcal{A}$ can pass Item 1 with probability at most $\frac{n_\times+2}{p^q}$ (i.e., the information-theoretic soundness of LPZK, see [YSWW21]). Now, consider

$\mathcal{A}$ whose invalid extended witness $\widetilde{\boldsymbol{w}_{ext}}$ satisfies the multiplication check, but does not satisfy any branch. Recall, $\boldsymbol{M}_i$ denotes the topology matrix of $\mathcal{C}_i$. We have $\forall i \in [B], \boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}, 1) \neq \mathbf{0}$. There are the following two possible events that will lead to $\mathcal{A}$ passing Item 2:

- The random challenge $\boldsymbol{s}$ from $\mathcal{V}$ in Step 9 creates a faulty 0. Namely, $\exists i \in [B], \boldsymbol{s}^T \times \boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}, 1) = 0$. Note that for each $i \in [B]$, $\boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}, 1) \neq \mathbf{0}$. By Corollary 1, this event only occurs with probability at most $\frac{B}{p^q}$.

- The random challenge $\boldsymbol{s}$ from $\mathcal{V}$ in Step 9 does *not* create a 0. Namely, $\forall i \in [B], \boldsymbol{s}^T \times \boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}, 1) \neq 0$. In this situation, $\mathcal{A}$ must forge the final product circuit evaluation on $B-1$ multiplication gates, which is another LPZK (and an extra opening of 0). The probability is at most $\frac{B+2}{p^q}$ (information-theoretic).

Thus, the two distributions seen by the environment $\mathcal{E}$ differ with probability at most $\frac{C_\times+2B+4}{p^q}$, and any *unbounded* environment $\mathcal{E}$ cannot distinguish the real-world execution and ideal-world execution, except with probability at most $\frac{C_\times+2B+4}{p^q}$.

Malicious $\mathcal{V}$. If $\mathcal{S}$ receives (`false`) from $\mathcal{F}_{\mathsf{ZK}}^{1,B}$, it simply aborts. If $\mathcal{S}$ receives (`true`) from $\mathcal{F}_{\mathsf{ZK}}^{1,B}$, it emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$, selects a uniformly random extended witness $\tilde{\boldsymbol{w}}_{ext}$, and acts as honest $\mathcal{P}$, except that it maliciously passes the check in Step 8, 11. $\mathcal{S}$ is able to pass these checks because it emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$, and thus it knows Δ as well as shares of $\mathcal{A}$'s IT-MACs (i.e., $\mathcal{S}$ knows what values $\mathcal{A}$ expects). The messages received by $\mathcal{A}$ are all uniformly distributed, and hence the distributions seen by the environment $\mathcal{E}$ in the two worlds are identical. □

C.4 Proof of Theorem 2

Theorem 2. *$\Pi_{\mathsf{Batch}}^{p,q}$ (Figure 6) UC-realizes $\mathcal{F}_{\mathsf{ZK}}^{R,B}$ (Figure 1) in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$-hybrid model with soundness error $\frac{Rn_\times+R+3B+6}{p^q}$ and information-theoretic security.*

Proof. By constructing a simulator $\mathcal{S}$. $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{ZK}}^{R,B}$ and run the adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$ for $\mathcal{A}$.

Malicious $\mathcal{P}$. Since $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$ for $\mathcal{A}$, $\mathcal{S}$ can trivially extract $\mathcal{A}$'s witness $\tilde{\boldsymbol{w}}^{(1)}, \ldots, \tilde{\boldsymbol{w}}^{(R)}$ from $\mathcal{A}$'s IT-MAC commitments. Note that $\mathcal{S}$ still must extract $\tilde{a}^{(1)}, \ldots, \tilde{a}^{(R)}$, which denote the active circuit identifiers, and these can be trivially extracted by trying each witness against each branch $\mathcal{C}_i$, and seeing which branch outputs zero. If no branch outputs zero, then $\tilde{\boldsymbol{w}}^{(j)}$ is an invalid witness.

$\mathcal{S}$ acts as honest $\mathcal{V}$ and emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$ to extract $\tilde{\boldsymbol{w}}^{(1)}, \ldots, \tilde{\boldsymbol{w}}^{(R)}$ and $\tilde{a}^{(1)}, \ldots, \tilde{a}^{(R)}$ ($\tilde{a}^{(j \in [R])}$ can be $\perp$). If honest woulld $\mathcal{V}$ output (`false`), $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{\mathsf{ZK}}^{R,B}$; otherwise, $\mathcal{S}$ sends $(\mathcal{C}_1, \ldots, \mathcal{C}_B, \tilde{\boldsymbol{w}}^{(1)}, \ldots, \tilde{\boldsymbol{w}}^{(R)}, \tilde{a}^{(1)}, \ldots, \tilde{a}^{(R)})$ to $\mathcal{F}_{\mathsf{ZK}}^{R,B}$.

We argue $\mathcal{S}$ is a valid simulator. Note that $\mathcal{V}$ in $\Pi_{\mathsf{BatchedStack}}^{p,q}$ only sends uniformly random values to $\mathcal{P}$, so $\mathcal{A}$'s real-world view is *identical* to its ideal-world view. Whenever real-world honest $\mathcal{V}$ would output (`false`), ideal-world $\mathcal{V}$ also outputs (`false`) since $\mathcal{S}$ will send $\perp$ to $\mathcal{F}_{\mathsf{ZK}}^{R,B}$.

There is one event we must specially account for. Suppose that $\mathcal{A}$ does not hold a valid witness, but it nevertheless manages to make real-world $\mathcal{V}$ output (`true`). In this case and in the ideal world, $\mathcal{S}$ emulates accepting $\mathcal{V}$ but it does not hold a valid witness, since $\mathcal{A}$ does not hold a valid witness. Thus, $\mathcal{S}$ will send an invalid input to $\mathcal{F}_{\mathsf{ZK}}^{R,B}$, which will make ideal-world $\mathcal{V}$ output (`false`). While this event is possible, it is unlikely. Indeed, the probability of this event is precisely the protocol soundness error.

We calculate a bound on soundness error. To make real-world $\mathcal{V}$ output (`true`), a malicious $\mathcal{P}$ must pass three checks in sequence:

1. The multiplication check performed by LPZK in Step 8.
2. The inner-product check performed by QS in Step 13.
3. The compressed topology token membership check performed by a random challenge and regular VOLE-based ZK in Steps 14, 15, 16.

If $\mathcal{A}$'s witness does not satisfy the multiplication relation, $\mathcal{A}$ can pass Item 1 with probability at most $\frac{Rn_\times+2}{p^q}$ (i.e., the information-theoretic soundness of LPZK, see [YSWW21]).

Now, suppose $\mathcal{A}$ uses R invalid extended witnesses $\widetilde{\boldsymbol{w}_{ext}}^{(1)}, \ldots, \widetilde{\boldsymbol{w}_{ext}}^{(R)}$ such that multiplications are well-formed (i.e., Item 1 passes).

If $\mathcal{A}$ commits to at least one invalid compressed topology vector, then $\mathcal{A}$ can pass Item 2 with probability at most $\frac{R+2}{p^q}$ (i.e., the information-theoretic soundness of QS, see [YSWW21]). This occurs when $\mathcal{A}$ uses R compressed topology vectors $\widetilde{\boldsymbol{cv}^{(1)}}, \ldots, \widetilde{\boldsymbol{cv}^{(R)}}$ such that there exists $j \in [R]$ where $(\widetilde{\boldsymbol{cv}^{(j)}})^T \times \widetilde{\boldsymbol{w}_{ext}}^{(j)} \neq 0$.

Now, suppose $\mathcal{A}$ commits to R valid compressed topology vectors that pass Item 2. We are left with Item 3, which checks that each of $\mathcal{A}$'s commited topogy vectors indeed corresponds to the topology of some branch, i.e. $\widetilde{\boldsymbol{cv}^{(j\in[R])}} \in \{\boldsymbol{cv}_1, \ldots, \boldsymbol{cv}_B\}$. For each i, $\boldsymbol{cv}_i \triangleq \boldsymbol{s}^T \times \boldsymbol{M}_i$ where $\boldsymbol{s}$ is uniformly sample by $\mathcal{V}$ and $\boldsymbol{M}_i$ is the topology matrix of $\mathcal{C}_i$. Note that each $\widetilde{\boldsymbol{cv}^{(j\in[R])}}$ will be checked individually, so we can focus on the case where $\mathcal{A}$ only has one faulty $\widetilde{\boldsymbol{w}_{ext}}^{(j\in[R])}$.

W.l.o.g., we assume the faulty one is $\widetilde{\boldsymbol{w}_{ext}}^{(1)}$ and associated compressed topology vector $\mathcal{A}$ committed is $\widetilde{\boldsymbol{cv}^{(1)}}$. Recall that $\widetilde{\boldsymbol{w}_{ext}}^{(1)}$ has already passed Item 1 check. This implies that $\forall i \in [B], \boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}^{(1)}, 1) \neq \boldsymbol{0}$. Recall $\mathcal{V}$ samples a uniformly random vector $\boldsymbol{s}$ to generate the compressed topology vectors $\boldsymbol{cv}_{i\in[B]}$ (Step 11). If there exists some $i \in [B]$ such that $\boldsymbol{s}^T \times \boldsymbol{M}_i \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}^{(1)}, 1) = 0$, then $\mathcal{A}$ can trivially pass Item 3 by setting $\widetilde{\boldsymbol{cv}^{(1)}}$ as $\boldsymbol{cv}_i$. However, by Corollary 1, this only happens with probability at most $\frac{B}{p^q}$.

Now consider that $\forall i \in [B], (\boldsymbol{cv}_i)^T \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}^{(1)}, 1) \neq 0$. Since $(\widetilde{\boldsymbol{cv}^{(1)}})^T \times \mathsf{glue}(\widetilde{\boldsymbol{w}_{ext}}^{(1)}, 1) = 0$ (Item 2 passed), this implies $\widetilde{\boldsymbol{cv}^{(1)}} \notin \{\boldsymbol{cv}_1, \ldots, \boldsymbol{cv}_B\}$. We bound the probability that Item 3 does not catch this. Recall that Item 3 will first further compress $\boldsymbol{cv}_i$ to some single element $ct_i \triangleq \boldsymbol{cv}_i^T \times \boldsymbol{t}$ for each $i \in [B]$, where t is uniformly sampled by $\mathcal{V}$ (Step 14). Then, $\mathcal{P}$ and $\mathcal{V}$ execute VOLE based-ZK on the IT-MAC $[(\widetilde{\boldsymbol{cv}^{(1)}})^T \times \boldsymbol{t}]$ to ensure $(\widetilde{\boldsymbol{cv}^{(1)}})^T \times \boldsymbol{t} \in \{ct_1, \ldots, ct_B\}$. $\mathcal{A}$ can trivially pass this check if $(\widetilde{\boldsymbol{cv}^{(1)}} \times \boldsymbol{t}) \in \{ct_i\}_{i\in[B]}$. However, by Corollary 2, this only happens with probability at most $\frac{B}{p^q}$. In the case that this does not happen, $\mathcal{A}$ must break soundness of the last VOLE-based ZK with $B-1$ multiplication gates, which is achieved by LPZK (and an extra opening on 0). The probability of this event is at most $\frac{B+2}{p^q}$.

Thus, the two distributions seen by the environment $\mathcal{E}$ differ with probability at most $\frac{RC_\times+R+3B+6}{p^q}$, and any *unbounded* environment $\mathcal{E}$ cannot distinguish the real-world execution and ideal-world execution, except with probability at most $\frac{RC_\times+R+3B+6}{p^q}$.

Malicious $\mathcal{V}$. If $\mathcal{S}$ receives (`false`) from $\mathcal{F}_{\mathsf{ZK}}^{R,B}$, it simply aborts. If $\mathcal{S}$ receives (`true`) from $\mathcal{F}_{\mathsf{ZK}}^{R,B}$, it emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$, selects R uniformly random extended witnesses $\widetilde{\boldsymbol{w}_{ext}}^{(1)}, \ldots, \widetilde{\boldsymbol{w}_{ext}}^{(R)}$, and acts as an honest $\mathcal{P}$, except that it maliciously passes the checks in Step $9, 13, 15$. $\mathcal{S}$ is able to pass these because it emulates $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$, and thus it knows Δ as well as shares of $\mathcal{A}$'s IT-MACs (i.e., $\mathcal{S}$ knows what values $\mathcal{A}$ expects). The messages received by $\mathcal{A}$ are all uniformly distributed, and hence the distributions seen by the environment $\mathcal{E}$ in the two worlds are identical. □

D Detailed Cost Accounting

D.1 Single Disjunction Protocol Costs

Communication. We analyze the communication complexity of $\Pi_{\mathsf{Single}}^{p,q}$ (Figure 5) in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$-hybrid model. In our analysis, we count the number of transmitted $\mathbb{F}_p$ elements:

- In Steps $4, 5, 6, 7$, $\mathcal{P}$ commits to her extended witness by transmitting $(n_{in} + 3n_\times)$ elements.
- In Step 8, the call to LPZK requires $\mathcal{V}$ to transmit a random challenge. This challenge contains q elements, and $\mathcal{P}$ replies to by transmitting $2q$ elements.
- In Step 9, $\mathcal{V}$ transmits the compressing vector $\boldsymbol{s}$, which consists of $q(2n_\times + 1)$ elements.
- In Step 11, $\mathcal{P}$ and $\mathcal{V}$ run a small VOLE-based proof to handle the small product circuit. This requires the following communication: $q(B-1)$ elements from $\mathcal{P}$ to commit to intermediate wire values, q elements from $\mathcal{P}$ to open the circuit output, q elements from $\mathcal{V}$ for a random LPZK challenge, and $2q$ elements from $\mathcal{P}$ for the LPZK response.

Tallying these costs, the total communication is $n_{in}+(2q+3)n_\times+q(B+7) = \mathcal{O}(q|\mathcal{C}|+qB)$ elements. We will soon show a simple variant that achieves $\mathcal{O}(|\mathcal{C}| + qB)$ communication by sacrificing some soundness. This variant is far more friendly to circuits over small fields (e.g., Boolean).
Number of required subfield VOLE correlations. $\Pi_{\mathsf{Single}}^{p,q}$ requires a total of $n_{in}+3n_\times+q(B+1) = \mathcal{O}(|\mathcal{C}|+qB)$ subfield VOLE correlations, almost all of which are used in the initialization phase; $2q$ subfield VOLEs are required for the two LPZK instances.
Computation. The computation for each party is dominated by Step 10, where they each compute $\boldsymbol{s}^T \times \boldsymbol{M}_i$ and corresponding IT-MACs for each $i \in [B]$. By leveraging $\mathsf{MUL}_{\mathsf{LEFT}}$ (Cf. Section 4), the computation cost is $\mathcal{O}(B|\mathcal{C}|)$ field operations. Other Steps require either $\mathcal{O}(B)$ or $\mathcal{O}(|\mathcal{C}|)$ operations.
5-round online phase. The VOLE correlations needed for LPZKs at Step $8, 11$ can be parallelized with initialization. Viewing initialization as preprocessing, our protocol can be viewed as a 5-round online phase:

1. $\mathcal{P}$ commits to her extended witness.
2. $\mathcal{V}$ sends the random challenge for the first LPZK and $\boldsymbol{s}$.
3. $\mathcal{P}$ sends the proof of the first LPZK and commits the intermediate values of the final product circuit.
4. $\mathcal{V}$ sends the random challenge for the second LPZK.
5. $\mathcal{P}$ sends the proof of the second LPZK and opens the final output (to prove it is 0).

D.2 Batched Disjunction Protocol Costs

We tally the costs of $\Pi^{p,q}_{\mathsf{Batch}}$:

Communication. We analyze the communication complexity of $\Pi^{p,q}_{\mathsf{Batch}}$ in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model. Our analysis counts the number of transmitted $\mathbb{F}_p$ elements:

- In Steps $5, 6, 7, 8$, $\mathcal{P}$ commits to her extended witness by transmitting $R(n_{in} + 3n_\times)$ elements.
- In Step 9, the call to LPZK requires $\mathcal{V}$ to transmit a random challenge. This challenge contains q elements, and $\mathcal{P}$ replies by transmitting $2q$ elements.
- In Step 10, $\mathcal{V}$ transmits the compressing vector $\boldsymbol{s}$, which consists of $q(2n_\times + 1)$ elements.
- In Step 12, $\mathcal{P}$ commits to her compressed topology vectors by transmitting $Rq(n_{in} + 3n_\times + 1)$ elements.
- In Step 13, the call to QS requires $\mathcal{V}$ to transmit a random challenge. This challenge contains q elements, and $\mathcal{P}$ replies by transmitting $2q$ elements.
- In Step 14, $\mathcal{V}$ transmits the second compressing vector $\boldsymbol{t}$, which consists of $q(n_{in} + 3n_\times + 1)$ elements.
- In Step 15, for each of the R repetitions, $\mathcal{P}$ and $\mathcal{V}$ run a small VOLE-based proof to handle each small product circuit. This requires the following communication: $Rq(B - 1)$ elements from $\mathcal{P}$ to commit to intermediate wire values, Rq elements to open each circuit output, Rq elements from $\mathcal{V}$ for random LPZK challenges, and $2Rq$ elements from $\mathcal{P}$ for LPZK replies.

Tallying these costs, the total communication is $(Rq+R+q)n_{in}+(3Rq+3R+5q)n_\times+Rq(B+4)+8q = \mathcal{O}(qRB + qR|\mathcal{C}|)$ elements.

Number of subfield VOLE correlations. $\Pi^{p,q}_{\mathsf{Batch}}$ requires a total of $(Rq + R)n_{in} + (3Rq + 3R)n_\times + qR(B + 1) + 2q = \mathcal{O}(qRB + qR|\mathcal{C}|)$ subfield VOLE correlations. Almost all of these are use to initialize; $(R + 1)q$ correlations are needed for the $(R + 1)$ calls to LPZK instances, and q correlations are used for the call to QS.

7-round online phase. Generating VOLE correlations needed for LPZKs and QS at Step $9, 13, 15$ can be parallelized with initialization. When initialization is viewed as preprocessing, our protocol has a 7 round online phase:

1. $\mathcal{P}$ commits to her R extended witnesses.
2. $\mathcal{V}$ sends $\boldsymbol{s}$ and the challenge for the first call to LPZK.
3. $\mathcal{P}$ replies to the first LPZK challenge and commits to her R compressed topology vectors.
4. $\mathcal{V}$ sends $\boldsymbol{r}$ and the challenge for the call to QS.
5. $\mathcal{P}$ replies to the QS challenge and commits to intermediate values of the R product circuits.
6. $\mathcal{V}$ sends a random challenge for each of the R calls to LPZK.
7. $\mathcal{P}$ replies to each of the R LPZK challenges and opens the R final outputs (to prove each product circuit outputs 0).

Computation. We analyze the computation cost of $\Pi^{p,q}_{\mathsf{Batch}}$ in the $\mathcal{F}^{p,q}_{\mathsf{sVOLE}}$-hybrid model. Our analysis counts field operations:

- In Step 0, $\mathcal{P}$ uses $\mathcal{O}(R|\mathcal{C}|)$ $\mathbb{F}_p$ operations to generate her R extended witnesses.
- In Steps $3, 4$, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(R|\mathcal{C}| + RB)$ operations to combine subfield VOLE correlations into VOLE correlations.
- In Steps $5, 6, 7, 8$, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(R|\mathcal{C}|)$ operations to generate commitments to the R extended witnesses.
- In Step 9, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(R|\mathcal{C}|)$ operations to execute LPZK.
- In Step 11, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(B|\mathcal{C}|)$ operations to compute B compressed topology vectors by using $\mathsf{MUL}_{\mathsf{LEFT}}$.
- In Step 12, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(R|\mathcal{C}|)$ operations to generate commitments to R compressed topology vectors.
- In Step 13, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(R|\mathcal{C}|)$ operations to execute QS.
- In Step 14, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(B|\mathcal{C}|)$ operations to compute B compressed topology tokens.
- In Step 15, $\mathcal{P}$ and $\mathcal{V}$ use $\mathcal{O}(RB)$ to execute R calls to LPZK, each on a circuit of size B.

The overall computation for each party is $\mathcal{O}(RB + R|\mathcal{C}| + B|\mathcal{C}|)$.

E Constraining Batch Witnesses

The batched disjunction setting allows $\mathcal{P}$ to prove the same disjunctive circuit with respect to R witnesses. This setting is only interesting if we impose additional constraints on $\mathcal{P}$'s witnesses; otherwise, $\mathcal{P}$ with only witness can trivially re-use her witness R times to satisfy the full statement. We present how to incorporate two typical types of additional constraints.

Per-repetition public parameters. One potential constraint is to associate with each repetition some public parameters. These public parameters are partial inputs to the branch circuit known to both $\mathcal{P}$ and $\mathcal{V}$, and $\mathcal{P}$'s witness must satisfy the circuit, even in the context of these extra inputs. Incorporating public parameters in our protocol is straightforward: $\mathcal{P}$ can simply open portions of the committed extended witness to prove to $\mathcal{V}$ that she indeed used the correct parameters. An even simpler (and less expensive) method would require $\mathcal{P}$ and $\mathcal{V}$ to generate IT-MACs of these public inputs directly. Note, because we must hide which branch is taken in each repetition, the parameters must be shared across branches.

Connecting repetitions. A more powerful constraint requires $\mathcal{P}$ to prove some consistency between the repetition witnesses. For instance, some wires of the first repetition should be used as particular input wires to the second repetition. We cannot ask $\mathcal{P}$ to open two different commitments to demonstrate equality, because these values are private. However, we can require $\mathcal{P}$ to provide extra proof demonstrating that the committed values are indeed the same. Such proves

# Branch	QuickSilver [YSWW21]					Robin				
	100 Mbps	500 Mbps	1 Gbps	2 Gbps	Comm.	100 Mbps	500 Mbps	1 Gbps	2 Gbps	Comm.
1	7.0	2.7	2.3	2.0	73.7MB	18.1	5.6	3.9	3.6	197MB
5	30.8	10.2	7.3	6.4	336MB	18.8	6.2	4.7	4.4	199MB
10	60.3	18.1	13.0	11.7	659MB	19.8	7.2	5.9	5.6	199MB
20	117.5	34.7	24.2	21.3	1.27GB	21.9	9.2	7.5	7.2	197MB
30	175.2	51.0	35.9	30.8	1.91GB	23.7	11.2	9.5	9.2	198MB
40	234.2	67.7	46.3	42.0	2.56GB	25.6	13.3	11.4	11.1	199MB
50	294.1	85.4	59.3	53.0	3.17GB	27.5	15.2	13.3	13.1	199MB
60	353.0	101.6	72.1	63.7	3.82GB	29.5	16.8	15.2	15.0	200MB
70	408.4	116.9	82.3	74.9	4.43GB	31.5	18.8	17.2	17.0	199MB
80	469.4	133.5	93.6	84.6	5.07GB	33.5	20.7	19.2	19.0	199MB
90	525.0	150.1	103.5	91.4	5.72GB	35.3	22.6	21.0	20.7	198MB
100	582.0	166.7	114.5	103.4	6.33GB	37.1	24.5	23.0	22.8	199MB

Figure 13: Robin vs. QuickSilver [YSWW21]. We tabulate end-to-end runtime in seconds. Figure 8 plots these results.

# Branch, Batch	QuickSilver [YSWW21]				Repeating Robin				Batchman			
	100 Mbps	500 Mbps	1 Gbps	Comm.	100 Mbps	500 Mbps	1 Gbps	Comm.	100 Mbps	500 Mbps	1 Gbps	Comm.
50, 50	231.6	67.4	46.5	2.53GB	22.0	12.2	10.9	160MB	28.6	8.9	6.2	311MB
100, 100	926.1	267.2	186.8	10.1GB	56.1	37.04	34.52	312MB	55.9	16.5	11.9	621MB
200, 200	3694.3	1065.8	735.9	40.1GB	166.1	127.9	123.0	620MB	111.4	31.3	21.5	1.20GB
300, 300	4296.2	2396.9	1670.2	90.1GB	330.3	273.1	265.8	922MB	165.9	48.1	31.9	1.81GB
400, 400	14747.3	4296.2	2983.6	160GB	550.3	474.0	464.6	1.21GB	221.2	62.9	42.6	2.41GB

Figure 14: Batchman vs. repeating Robin vs. QuickSilver [YSWW21]. We tabulate end-to-end runtime in seconds. Figure 10 plots results.

can be efficiently achieved by the IT-MAC linear homomorphism: the parties simply subtract two supposedly-equal values, and then $\mathcal{P}$ proves that the result is a IT-MAC of zero. Thus, $\mathcal{P}$ can finish the extra proof by sending one field element per constraint. By leveraging random oracle in a standard way, many such zero checks can be compressed into one element, yielding overall $\mathcal{O}(1)$ overhead. See [BMRS21] for details of this RO trick.

F Additional Evaluation

Robin vs. QuickSilver [YSWW21]. Recall that we test our single disjunction protocol Robin and QuickSilver on branches that each have 8 million multiplication gates. Figure 13 tabulates the results of these experiments, which were used to generate Figure 8.

Robin vs. QuickSilver [YSWW21] in the online phase. Many previous VOLE-based ZK protocols only consider the online phase. Namely, they assume that VOLE correlations are "free", or can be viewed as preprocessing. So far, our experiments consider the full end-to-end runtime, including generating VOLE correlations. We also tested Robin's online phase and compared it with QuickSilver's; Figure 15 shows the results.

Batchman vs. repeating Robin vs. QuickSilver [YSWW21]. Recall that we compared Batchman, (repeated uses of) Robin, and QuickSilver on branches that each have 1.25×10^5 multiplication gates. Figure 14 tabulates the results of these experiments, which were used to generate Figure 10.

Fine-grained analysis of Robin. Figure 16 breaks down the runtime cost of our single disjunction protocol Robin.

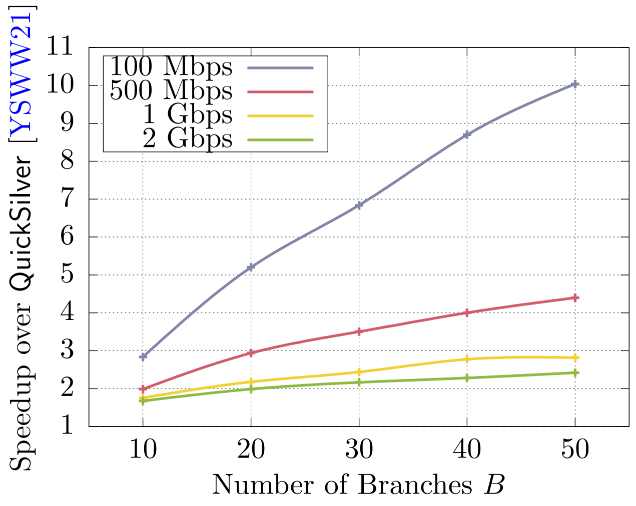


Figure 15: The speedup of our single disjunction protocol Robin over QuickSilver [YSWW21] in the online phase. We measure end-to-end runtime. The circuits are defined over $\mathbb{F}_{2^{61}-1}$, and each branch has 1.25×10^5 multiplication gates.

# Branch	Check	100 Mbps	500 Mbps	1 Gbps	2 Gbps
20	multi. check	18.0	5.4	3.7	3.7
	topo. check	3.9	3.8	3.8	3.5
40	multi. check	18.0	5.5	4.1	3.4
	topo. check	7.6	7.9	7.3	7.7
80	multi. check	17.9	5.3	3.7	3.6
	topo. check	15.6	15.4	15.5	15.3

Figure 16: Fine-grained analysis of our single disjunction protocol Robin. Measurements are in seconds.