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

Yibin Yang
Georgia Institute of Technology, USA
yyang811@gatech.edu

David Heath
University of Illinois
Urbana-Champaign, USA
daheath@illinois.edu

Carmit Hazay
Bar-Ilan University, Ligero Inc., Israel
Carmit.Hazay@biu.ac.il

Vladimir Kolesnikov
Georgia Institute of Technology, USA
kolesnikov@gatech.edu

Muthuramakrishnan Venkitasubramaniam
Ligero Inc., USA
muthu@ligero-inc.com

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 $O(RB + R|C| + B|C|)$, where $|C|$ is the maximum circuit size of the B branches. Prior works' computation scales in $RB|C|$.

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×.

CCS CONCEPTS

• **Security and privacy → Cryptography**; • **Theory of computation → Cryptographic protocols**.

KEYWORDS

Zero Knowledge; Disjunctions; Batched Disjunctions

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623169

ACM Reference Format:
Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. 2023. Batchman and Robin: Batched and Non-batched Branching for Interactive ZK. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623169

1 INTRODUCTION

Zero Knowledge (ZK) proofs [25] 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' [27] or even (the more complex) 'this program *does not* have a bug' [32].

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) [1–3, 18, 19, 37–40]. State-of-the-art VOLE-base ZK is efficient. For instance, [40] 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 [3, 23, 24, 27, 31] specifically optimized for proofs of *disjunctive* statements of the form $C_1(\boldsymbol{w}) = 0 \vee \cdots \vee C_B(\boldsymbol{w}) = 0$ for B different subcircuits referred to as *branches*. Their underlying techniques

[1] The recent proof system of [39] achieved sublinear communication cost, but at the cost of asymptotically *increased* computation; see Section 1.2.

are often referred as *stacking*, following the notation introduced by [27]. 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 $C_1(\boldsymbol{w}^{(j)}) = 0 \vee \cdots \vee C_B(\boldsymbol{w}^{(j)}) = 0$ 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 programming languages, see e.g. [29]. 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* [22]; loop coalescing is known to be useful for fast RAM-based ZKP [35].

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 [20] and Cramer, Damgård and Schoenmakers [16] for the design of witness indistinguishable proofs and efficient Σ-protocols respectively.

Full version. Full version of this paper is available at [42].

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 $|C|$ (multiplication) gates, $\mathcal{P}$ and $\mathcal{V}$ each compute $O(B|C|)$ field operations and communicate $O(B + |C|)$ field elements. More precisely, $\mathcal{P}$ and $\mathcal{V}$ communicate only $O(B\lceil\frac{\lambda}{\log|\mathbb{F}|}\rceil + |C|)$ field elements.

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

Protocol	Prover Comp.	Comm.	Verifier Comp.
[19, 40]	$O(RB\|C\|)$	$O(RB\|C\|)$	$O(RB\|C\|)$
[39]	$O(RB\|C\|\log R)$	$O(B\|C\| + R)$	$O(RB\|C\|\log R)$
[3]	$O(RB\|C\|)$	$O(R\log B + R\|C\|)$	$O(RB\|C\|)$
our, Robin	$O(RB\|C\|)$	$O(RB + R\|C\|)$	$O(RB\|C\|)$
our, Batchman	$O(RB + R\|C\| + B\|C\|)$	$O(RB + R\|C\|)$	$O(RB + R\|C\| + B\|C\|)$

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

- **Batched disjunctions.** Batchman extends Robin to *batches* of R proofs of the same disjunction. Here, $\mathcal{P}$ and $\mathcal{V}$ each compute $O(RB + R|C| + B|C|)$ field operations and communicate $O(RB + R|C|)$ 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 $B|C|$ 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 $RB|C|$.

Our protocols are concretely performant. E.g., Robin scales in branches up to 6× better than Mac'n'Cheese [3] when $|C| = 10^9$, and demonstrates up to 16× improvement over QuickSilver [40] when $B = 100$. Batchman demonstrates up to 36× improvement over AntMan [39] when $B = 64$ and $R = 1024$, and up to 70× improvement over QuickSilver [40] 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 $RB|C|$ 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 $\tilde{O}(B|C|)$ for the first instance (to compute B hashes) and $\tilde{O}(B + |C|)$ 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 $O(RB + R|C| + B|C|)$ 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 C with $|C|$ multiplication/addition gates over some field. [19] achieved 1 field element communication per multiplication gate based on a technique called Line-Point Zero Knowledge (LPZK). [40] further improved LPZK and implemented the technique. Our work handles multiplication gates by directly applying the LPZK technique, as well as [40]'s improvement for proving inner products. Our implementation (see Section 7) builds on [40]'s open source repo [36].

[18] improved LPZK communication to 0.5 field element per multiplication gate at the cost of increased computation and requiring random oracle. Concrete performance of [18] is similar to [40]; we build on [40] for simplicity.

[39] for the first time achieved a VOLE-based ZK system with sublinear communication and achieved communication cost $O(|C|^{3/4})$. 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 $O(\log |C|)$ overhead, and it also employs relatively expensive additively homomorphic encryption. [39] consider batching, but not batched disjunctions; we compare with them in Section 7.

ZK disjunctions. A line of works [3, 23, 24, 27] augments ZK proofs with efficient handling for disjunctions. Consider B circuits $C_{i\in[B]}$ each with the same number of inputs/multiplications, and suppose $\mathcal{P}$ holds a single witness for $C_{a\in[B]}$ (the *active branch*). These works achieve communication that scales with $|C|$ rather than $B|C|$. Such works say that they "stack" the branches, following terminology introduced by [27].

Most related to our work, [3] was the first (and the only) work to stack in the VOLE-based ZK setting. [3] 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, [27] 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. [4–7, 9, 13, 17, 21, 26, 28, 30, 33, 35, 43]. 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 [17, 21], 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 $O(B|C| + R|C|)$.

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

Upon receiving $(\texttt{prove}, C_1, \ldots, C_B, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_R, a_1, \ldots, a_R)$ from prover $\mathcal{P}$ and $(\texttt{verify}, C_1, \ldots, C_B)$ from verifier $\mathcal{V}$:

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

Figure 1: The batched disjunctive ZK functionality.

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 $O(B|C|)$ cost. At the same time, our communication cost is *independent* of the quantity $O(B|C|)$, 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.

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., $k \in [|C|]$).
- 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 [14]. We use UC to formalize our protocols and to prove

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

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 $(\texttt{extend}, n)$ 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

security in the presence of a *malicious*, *static* adversary. The functionality $\mathcal{F}_{\text{ZK}}^{R,B}$ (C.f., Figure 1) is used to realize a zero-knowledge proof (of knowledge) for R-repetitions disjunction of B circuits. When $R = B = 1$, $\mathcal{F}_{\text{ZK}}^{1,1}$ is the standard ZK functionality. When $R = 1$, $\mathcal{F}_{\text{ZK}}^{1,B}$ is the ZK functionality for a single disjunction. Looking ahead, our protocol for single disjunction implements $\mathcal{F}_{\text{ZK}}^{1,B}$ (for $B \in \mathbb{Z}^+$) and our protocol for batched disjunctions implements $\mathcal{F}_{\text{ZK}}^{R,B}$ (for $R, B \in \mathbb{Z}^+$).

2.3 VOLE and IT-MACs

Recent works [10–12, 15, 34, 41] 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 [19] and QuickSilver [40]

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 C, and let $\mathcal{P}$ hold a witness $\boldsymbol{w}$. To prove $C(\boldsymbol{w}) = 0$, $\mathcal{P}$ first computes her *extended witness* $\boldsymbol{w}_{ext}$, which consists of $\boldsymbol{w}$ along with each multiplication gate's output value upon evaluating $C(\boldsymbol{w})$. 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 [19, 40] handles multiplication gates via a technique called *line-point zero-knowledge* (LPZK). At a high level, LPZK [19] 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 $O(n)$ field operations. We denote the procedure to prove multiplications for IT-MACs as $\text{LPZK}(\{[l_i]_\Delta, [r_i]_\Delta, [o_i]_\Delta\}_{i \in [n]})$, which we use as a black-box. LPZK has $(n + 2)/p^q$ soundness error and information-theoretic security in the $\mathcal{F}_{\text{sVOLE}}^{p,q}$-hybrid model [40][3].

[3] [40] uses "extended subfield VOLE", which handles higher degree polynomials. We only use degree-2 polynomials, so $\mathcal{F}_{\text{sVOLE}}^{p,q}$ is sufficient.

QuickSilver [40] 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 $O(1)$ communication and $O(m)$ computation. Further, incorporating a random challenge from $\mathcal{V}$, k inner-product proofs can be batched, preserving $O(1)$ 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. This sub-proof, as shown by [40], is zero-knowledge with $(k+2)/p^q$ soundness error and information-theoretic security in the $\mathcal{F}_{\mathsf{sVOLE}}^{p,q}$-hybrid model[3].

We defer additional details on LPZK and QS to the full version.

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 $C_1, \ldots, C_B$ (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 $C_1(\boldsymbol{w}) = 0 \vee \cdots \vee C_B(\boldsymbol{w}) = 0$.

Note that basic VOLE-based ZK (e.g., [40]) 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 C_a. That is, $\mathcal{P}$ must prove that $\boldsymbol{w}_{ext}$ respects the *topology* of branch C_a, as well as the linear constraints implied by 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 $O(|C|)\times O(|C|)$, encoding the topology of C_i *a-la* adjacency matrix. For each branch C_i with matrix $\boldsymbol{M}_i$, consider the following crucial equation:

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

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.
- 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 $O(|C|^2)$, we will demonstrate that all relevant operations we used on these matrices can be computed in time $O(|C|)$.

Terminology. Our approach centers on the manipulation of matrices $\boldsymbol{M}_i$ which encode the topology of circuits 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 $O(|C|) \times O(|C|)$.
- 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 $O(|C|)$ 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 $(C_1(\boldsymbol{w}) = 0 \vee \cdots \vee C_B(\boldsymbol{w}) = 0)$. Our goal is to achieve communication that scales with $B + |C|$, not $B|C|$, 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:

[4]This means that our extended witness is up to 3× 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.

$$(\mathsf{left}(\boldsymbol{w}_{ext}) \circ \mathsf{right}(\boldsymbol{w}_{ext}) = \mathsf{out}(\boldsymbol{w}_{ext})) \quad \text{multiplication check}$$
$$\wedge \left(\left(\prod_{i}^{B} \boldsymbol{s}^T \times \boldsymbol{M}_i \times \boldsymbol{w}_{ext}\right) = 0\right) \quad \text{topology check}$$

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 $(C_1(\boldsymbol{w}_j) = 0 \vee \cdots \vee C_B(\boldsymbol{w}_j) = 0)$ 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 $O(|C|)$ field operations, and so ultimately this attempt uses $O(RB|C|)$ 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 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 $O(|C|)$ 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 $j \in [R]$,

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

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 $C_1, \ldots, C_B$. 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 C, and suppose that we remove each multiplication gate from C. Whenever we remove a multiplication gate, we replace its input and output wires with inputs to C. What remains is a skeleton of the circuit containing only addition gates that expresses a linear relationship on the extended witness (and

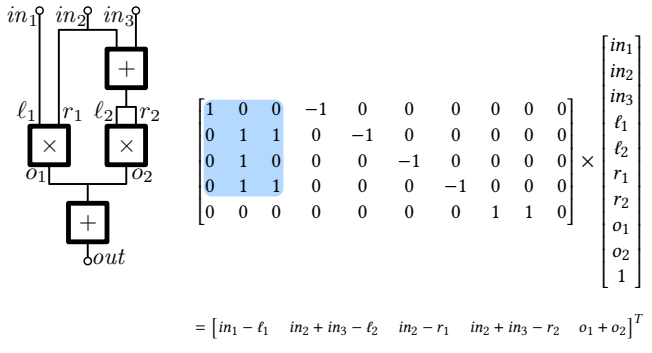

Figure 3: A simple arithmetic circuit computing $(in_1 \cdot in_2) + (in_2 + in_3)^2$ (left) and its corresponding topology matrix (right). Note, the shaded portion of the matrix is *dense*.

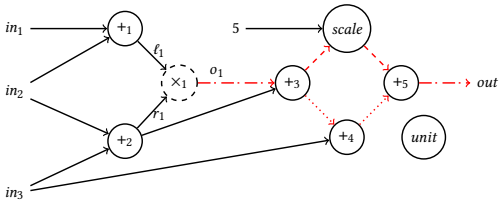

Figure 4: A circuit's induced DAG. In the topology matrix of this circuit, the last row defines $out \triangleq 6in_2 + 7in_3 + 6o_1$. In this DAG, there are 6 paths from in_2 to out, 7 paths from in_3 to out, and 6 paths from o_1 to out. E.g., from o_1, there are 5 paths (dashed) passing through the 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.

C's output). It is convenient to encode this linear relationship as a matrix $\boldsymbol{M} \in \mathbb{F}^{(2C_\times+1)\times(C_{in}+3C_\times+1)}$, 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 C if and only if:

(1) Multiplication gates in the $\boldsymbol{w}_{ext}$ are formed correctly.
(2) $\boldsymbol{M} \times \text{glue}(\boldsymbol{w}_{ext}, 1) = \boldsymbol{0}$, 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 $O(|C|^2)$ overhead, far exceeding our asymptotic budget.

Perhaps surprisingly, given the gate-by-gate representation of C, this multiplication can be computed in time $O(|C|)$ with our technique "evaluating 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 C. The wire w_k can be defined as a linear combination of input wires of C and output wires of all multiplication gates. We call this linear combination *the linear constraint* on w_k.

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

Definition 4.1 (Topology Matrix). Let C denote a circuit over some field $\mathbb{F}$ such that C has n_{in} input wires and $n_\times$ multiplication gates. The **topology matrix** associated with 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 $O(|C|^2)$ field operations, exceeding our asymptotic budget. Instead, we propose an efficient algorithm called $\mathsf{MUL}_{\mathsf{LEFT}}$ to support the above operation. Given the gate-by-gate circuit representation[7] of C, our algorithm essentially gate-by-gate evaluates C "backwards" in $O(|C|)$ operations *without ever writing down the matrix* $\boldsymbol{M}$.

It is not obvious that this multiplication can be achieved in $O(|C|)$ 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 C is itself a succinct representation of $\boldsymbol{M}$, and our algorithm exploits this.

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

[7] The full version defines a standard arithmetic circuit gate-by-gate representation.

Algorithm 1: $\mathsf{MUL}_{\mathsf{LEFT}}$ takes as input (1) an arithmetic circuit C over a field $\mathbb{F}$ written in gate-by-gate representation (see the full version for the formal definitions) 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 C. Array indices start at 1.

Input: circuit C, vector $\boldsymbol{s}$
Output: $\boldsymbol{s}^T \times \boldsymbol{M}$

$\boldsymbol{w} = \boldsymbol{0}^{(|C.\mathsf{wid}|)}$ defined over the extension field;
$acc = 0$ defined over the extension field;
$w[|C.\mathsf{wid}|] = s[2n_\times + 1]$;
for *each multiplication gate* (ℓ_k, r_k, o_k) **do**
  $w[\ell_k] = w[\ell_k] + s[k]$; $w[r_k] = w[r_k] + s[k + n_\times]$;
for *each linear gate* G *in reverse topological order* **do**
  if G *is an addition gate* (ℓ'_k, r'_k, o'_k) **then**
    $w[\ell'_k] = w[\ell'_k] + w[o'_k]$; $w[r'_k] = w[r'_k] + w[o'_k]$;
  if G *is a scale gate* (c_k, x_k, y_k) **then**
    $w[x_k] = w[x_k] + c_k \cdot w[y_k]$;
  if G *is an offset gate* (c'_k, x'_k, y'_k) **then**
    $w[x'_k] = w[x'_k] + w[y'_k]$; $acc = acc + c'_k \cdot w[y'_k]$;
$\boldsymbol{res} = \{\}$;
for *each input wire* in_k *in order* **do** $\boldsymbol{res}$.append$(w[in_k])$;
for *each* $k \in [2n_\times]$ **do** $\boldsymbol{res}$.append$(-s[k])$;
for *each multiplication gate* (ℓ_k, r_k, o_k) *in order* **do**
  $\boldsymbol{res}$.append$(w[o_k])$;
$\boldsymbol{res}$.append(acc);
return $\boldsymbol{res}$

Our $O(|C|)$ solution. Algorithm 1 presents $\mathsf{MUL}_{\mathsf{LEFT}}$. To understand our algorithm, we analyze the semantics of topology matrices. Let C denote a circuit, and consider 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 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 C. Now, consider the first row of $\boldsymbol{M}$ (denoted as $\boldsymbol{M}(1)$). As specified by Definition 4.1, this row defines the linear constraint on ℓ_1, the left input wire of 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 4.1) 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 $O(|C|)$.

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[8] and associated corollaries, which are the kernel of our protocols and proofs.

Lemma 5.1. *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:*

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

Corollary 5.2. *If $\boldsymbol{s}$ is drawn from the extension field $\mathbb{F}^q$ where $q \in \mathbb{Z}^+$, the upper bound of Lemma 5.1 is $k/|\mathbb{F}|^q$.*

Corollary 5.3. *Consider a field $\mathbb{F}$ and let $k, m \in \mathbb{Z}^+$. Consider k arbitrary non-zero vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)} \in \mathbb{F}^m$ and a vector $\boldsymbol{y} \in \mathbb{F}^m$ such that $\boldsymbol{y} \notin \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}\}$. The following holds:*

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

Lemma 5.4. *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 $\boldsymbol{s} \triangleq (1, \chi, \ldots, \chi^{m-1})$.

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 5.5 (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 5.5 in the full version; for now, we sketch the main argument.

[8] Check the full version for the proofs.

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

Inputs. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ hold B circuits $C_1, \ldots, C_B$ 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 $C_a(\boldsymbol{w}) = 0$ and $|\boldsymbol{w}| = n_{in}$.

Generate extended witness on C_a.
0. $\mathcal{P}$ evaluates $C_a(\boldsymbol{w})$ 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 (`init`) to $\mathcal{F}^{p,q}_{\text{sVOLE}}$, 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 $\mathcal{F}^{p,q}_{\text{sVOLE}}$, 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 $\mathcal{F}^{p,q}_{\text{sVOLE}}$, 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 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 zero-knowledge proof for multiplications as $\mathsf{LPZK}(\{[\ell_k], [r_k], [o_k]\}_{k\in[n_\times]})$; if ZKP fails, $\mathcal{V}$ outputs (`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 $C_1, \ldots, C_B$. Let $\boldsymbol{w}_{ext} \triangleq \text{glue}(\boldsymbol{w}, \boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{o}, 1) \in \mathbb{F}_p^{n_{in}+3n_\times+1}$ and associated IT-MAC $[\boldsymbol{w}_{ext}] \triangleq \text{glue}([\boldsymbol{w}], [\boldsymbol{\ell}], [\boldsymbol{r}], [\boldsymbol{o}], [1])$. $\mathcal{P}$ 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 (`true`); otherwise, $\mathcal{V}$ outputs (`false`).

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

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 $\mathcal{F}^{1,B}_{\text{ZK}}$ by running $\mathcal{A}$ as a subroutine. $\mathcal{S}$ implements the ideal functionality $\mathcal{F}^{p,q}_{\text{sVOLE}}$ 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.

For malicious prover $\mathcal{A}$, the witness can be trivially extracted from messages sent to $\mathcal{F}^{p,q}_{\text{sVOLE}}$. $\mathcal{S}$ runs a proof interaction with $\mathcal{A}$ by acting as honest $\mathcal{V}$, and it sends the extracted witness to $\mathcal{F}^{1,B}_{\text{ZK}}$ 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 5.2). □

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

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

Appendix A provides detailed explanation of this cost accounting.

Generating random challenges. $\Pi^{p,q}_{\text{Single}}$ Step 9 requires $\mathcal{V}$ send a random challenge $\boldsymbol{s}$ of size $O(q|C|)$ 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 [40]. 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 $O(|C| + qB)$ communication and $O(|C|)$ 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 5.4) 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 $O(|C|+qB)$ 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 6.1 (Batched Disjunctions Security). *$\Pi^{p,q}_{\text{Batch}}$ (Figure 6) UC-realizes $\mathcal{F}^{R,B}_{\text{ZK}}$ (Figure 1) in the $\mathcal{F}^{p,q}_{\text{sVOLE}}$-hybrid model with soundness error $\frac{Rn_\times+R+3B+6}{p^q}$ and information-theoretic security.*

We provide a proof of Theorem 6.1 in the full version. In short, the proof is very similar to that of Theorem 5.5, 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, $\Pi^{p,q}_{\text{Batch}}$ consumes the following resources:

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

Appendix A 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. 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 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 $O(1)$ overhead. See [3] for details of this RO trick.

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 [40] (their code is part of the EMP Toolkit [36]). We use their VOLE and LPZK implementations.

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[9] (respectively implementing $\mathcal{P}$ and $\mathcal{V}$). Our implementations run single threaded.

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 $O(\ell^3)$ 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 *[3].* We compare Robin with the prior state-of-the-art VOLE-based ZK protocol supporting disjunctions: Mac′n′Cheese [3]. The Mac′n′Cheese implementation is not publicly available, so we use the numbers available in their paper.

[3] reported execution time when handling B branches, each consisting of ≈ 1 billion AND gates. Each branch computes 45000 iterations of the SHA-2 circuit.[10] For these large Boolean branches, [3] uses an elegant trick based on [8] to reduce the communication cost of each AND gate to only 1 bit (rather than paying two extension

[9] Intel Xeon Platinum 8175 CPU @ 3.10GHz, 8 vCPUs, 32GiB Memory, 10Gbps Network
[10] More precisely, they also consider a branch that computes 150000 iterations of AES, but this branch is smaller than the SHA-2 circuit.

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

Inputs. $\mathcal{P}$ and $\mathcal{V}$ agree on B circuits $C_1, \ldots, C_B$ 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 $j \in [R]$, $C_{a^{(j)}}(\boldsymbol{w}^{(j)}) = 0$ and $|\boldsymbol{w}^{(j)}| = n_{in}$.

Generate extended witnesses for $C_{a^{(1)}}, \ldots, C_{a^{(R)}}$.
0. For each $j \in [R]$, $\mathcal{P}$ evaluates $C_{a^{(j)}}(\boldsymbol{w}^{(j)})$ 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 (`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 $C_{a^{(1)}}, \ldots, C_{a^{(R)}}$. 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 (`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 $C_1, \ldots, C_B$.
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 (`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 (`true`); otherwise, $\mathcal{V}$ outputs (`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.**

# Branch	Mac'n'Cheese [3] 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 **[3].** We tabulate end-to-end runtime in seconds. Our reported numbers for [3] 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 ≈ 1 billion AND gates. [3] 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.

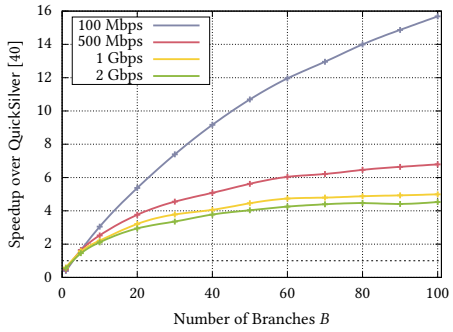

Figure 8: The speedup of our single disjunction protocol Robin **over** QuickSilver **[40].** We report end-to-end proof runtime. Circuits are over $\mathbb{F}_{2^{61}-1}$; each branch has 8M mult. gates.

fields communication per AND gate); this trick increases round complexity by factor $O(\log |C|)$.

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[11]. Figure 7 tabulates the results.

[3]'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 [3] 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 *[40].* [40] is a state-of-the-art VOLE-based ZK protocol for Boolean/arithmetic circuits. It uses $O(B|C|)$ computation and communication with low constants, and it serves as the baseline for our approach. We compare our single disjunction protocol Robin with [40] on circuits defined over $\mathbb{F}_{2^{61}-1}$. Asymptotically, Robin improves communication from $O(B|C|)$ to $O(B + |C|)$.

We compare using branches that each have 8 million multiplication gates, and we vary B between 5 and 100. Figure 8 plots our speedup. [40] requires 73.7 MB communication per branch; Robin requires ≈ 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× 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. The full version includes further evalution.

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 *[39].* AntMan [39] 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 $B|C|$ instruction which is executed R times. Recall that AntMan incurs $O(RB|C| \log R)$ computation and $O(B|C| + R)$ communication. Our batched protocol improves in terms of computation, incurring $O(RB + R|C| + B|C|)$ computation and $O(RB + R|C|)$ 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[11] machines. [39] 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} total multiplication gates. ([39] 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 $O(R|C|)$ 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 [39].

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 *[40] and* Robin. We compare Batchman to the baseline QuickSilver [40] 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.

7.2.3 Fine-grained Analysis. Figure 11 breaks down the runtime cost of Batchman. Most of the execution time is spent committing

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

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 **[39].** We tabulate millions of multiplication gates executed per second (mgps). AntMan-t refers to AntMan with t threads (numbers from [39]). 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.

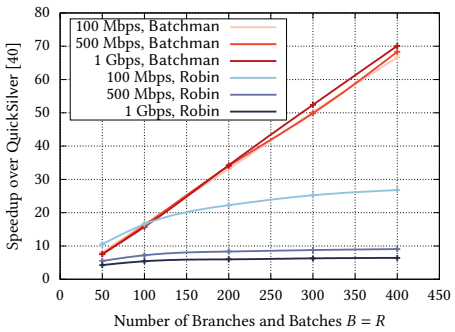

Figure 10: The speedup of Batchman **and** Robin **over** QuickSilver **[40].** We plot factor improvement in terms of end-to-end runtime. Circuits are defined over $\mathbb{F}_{2^{61}-1}$ and each branch has 1.25×10^5 multiplication gates.

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 [40]	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.

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 [29]) 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× improvement as compared to QuickSilver [40]. 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, [17, 21]) needs to be incorporated.

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

[1] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, Benoît Razet, and Peter Scholl. 2021. Appenzeller to Brie: Efficient Zero-Knowledge Proofs for Mixed-Mode Arithmetic and Z2k. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 192–211. https://doi.org/10.1145/3460120.3484812

[2] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. 2022. Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs over $\mathbb{Z}_{2^k}$. In *CRYPTO 2022, Part IV (LNCS, Vol. 13510)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 329–358. https://doi.org/10.1007/978-3-031-15985-5_12

[3] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. 2021. Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In *CRYPTO 2021, Part IV (LNCS, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Germany, Virtual Event, 92–122. https://doi.org/10.1007/978-3-030-84259-8_4

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *CRYPTO 2013, Part II (LNCS, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 90–108. https://doi.org/10.1007/978-3-642-40084-1_6

[5] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. In *CRYPTO 2014, Part II (LNCS, Vol. 8617)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 276–294. https://doi.org/10.1007/978-3-662-44381-1_16

[6] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. 2020. Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads. In *TCC 2020, Part II (LNCS, Vol. 12551)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, Heidelberg, Germany, Durham, NC, USA, 168–197. https://doi.org/10.1007/978-3-030-64378-2_7

[7] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. 2021. Time- and Space-Efficient Arguments from Groups of Unknown Order. In *CRYPTO 2021, Part IV (LNCS, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Germany, Virtual Event, 123–152. https://doi.org/10.1007/978-3-030-84259-8_5

[8] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 67–97. https://doi.org/10.1007/978-3-030-26954-8_3

[9] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. 2018. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program

Execution. In *ASIACRYPT 2018, Part I (LNCS, Vol. 11272)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 595–626. https://doi.org/10.1007/978-3-030-03326-2_20

[10] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing Vector OLE. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 896–912. https://doi.org/10.1145/3243734.3243868

[11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, London, UK, 291–308. https://doi.org/10.1145/3319535.3354255

[12] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 489–518. https://doi.org/10.1007/978-3-030-26954-8_16

[13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. 2013. Verifying Computations with State. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 341–357. https://doi.org/10.1145/2517349.2522733

[14] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, Las Vegas, NV, USA, 136–145. https://doi.org/10.1109/SFCS.2001.959888

[15] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. 2021. Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In *CRYPTO 2021, Part III (LNCS, Vol. 12827)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Germany, Virtual Event, 502–534. https://doi.org/10.1007/978-3-030-84252-9_17

[16] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94 (LNCS, Vol. 839)*, Yvo Desmedt (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 174–187. https://doi.org/10.1007/3-540-48658-5_19

[17] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Titouan Tanguy, and Michiel Verbauwhede. 2022. Efficient Proof of RAM Programs from Any Public-Coin Zero-Knowledge System. In *Security and Cryptography for Networks*, Clemente Galdi and Stanislaw Jarecki (Eds.). Springer International Publishing, Cham, 615–638.

[18] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. 2022. Improving Line-Point Zero Knowledge: Two Multiplications for the Price of One. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 829–841. https://doi.org/10.1145/3548606.3559385

[19] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. 2021. Line-Point Zero Knowledge and Its Applications. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 199)*, Stefano Tessaro (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 5:1–5:24. https://doi.org/10.4230/LIPIcs.ITC.2021.5

[20] Uriel Feige and Adi Shamir. 1990. Witness Indistinguishable and Witness Hiding Protocols. In *22nd ACM STOC*. ACM Press, Baltimore, MD, USA, 416–426. https://doi.org/10.1145/100216.100272

[21] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. 2021. Constant-Overhead Zero-Knowledge for RAM Programs. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 178–191. https://doi.org/10.1145/3460120.3484800

[22] Anwar M. Ghuloum and Allan L. Fisher. 1995. Flattening and Parallelizing Irregular, Recurrent Loop Nests. *SIGPLAN Not.* 30, 8 (aug 1995), 58–67. https://doi.org/10.1145/209937.209944

[23] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. 2022. Stacking Sigmas: A Framework to Compose Σ-Protocols for Disjunctions. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Heidelberg, Germany, Trondheim, Norway, 458–487. https://doi.org/10.1007/978-3-031-07085-3_16

[24] Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner. 2023. Speed-Stacking: Fast Sublinear Zero-Knowledge Proofs for Disjunctions. In *EUROCRYPT 2023, Part II (LNCS, Vol. 14005)*, Carmit Hazay and Martijn Stam (Eds.). Springer, Heidelberg, Germany, Lyon, France, 347–378. https://doi.org/10.1007/978-3-031-30617-4_12

[25] S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA) *(STOC '85)*. Association for Computing Machinery, New York, NY, USA, 291–304. https://doi.org/10.1145/22145.22178

[26] David Heath and Vladimir Kolesnikov. 2020. A 2.1 KHz Zero-Knowledge Processor with BubbleRAM. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 2055–2074. https://doi.org/10.1145/3372297.3417283

[27] David Heath and Vladimir Kolesnikov. 2020. Stacked Garbling for Disjunctive Zero-Knowledge Proofs. In *EUROCRYPT 2020, Part III (LNCS, Vol. 12107)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, Germany, Zagreb, Croatia, 569–598. https://doi.org/10.1007/978-3-030-45727-3_19

[28] David Heath and Vladimir Kolesnikov. 2021. PrORAM - Fast $P(\log n)$ Authenticated Shares ZK ORAM. In *ASIACRYPT 2021, Part IV (LNCS, Vol. 13093)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Heidelberg, Germany, Singapore, 495–525. https://doi.org/10.1007/978-3-030-92068-5_17

[29] David Heath, Yibin Yang, David Devecsery, and Vladimir Kolesnikov. 2021. Zero Knowledge for Everything and Everyone: Fast ZK Processor with Cached ORAM for ANSI C Programs. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 1538–1556. https://doi.org/10.1109/SP40001.2021.00089

[30] Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. 2015. Efficient Zero-Knowledge Proofs of Non-algebraic Statements with Sublinear Amortized Cost. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 150–169. https://doi.org/10.1007/978-3-662-48000-7_8

[31] Vladimir Kolesnikov. 2018. Free IF: How to Omit Inactive Branches and Implement S-Universal Garbled Circuit (Almost) for Free. In *ASIACRYPT 2018, Part III (LNCS, Vol. 11274)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 34–58. https://doi.org/10.1007/978-3-030-03332-3_2

[32] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. 2022. Proving UNSAT in Zero Knowledge. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 2203–2217. https://doi.org/10.1145/3548606.3559373

[33] Payman Mohassel, Mike Rosulek, and Alessandra Scafuro. 2017. Sublinear Zero-Knowledge Arguments for RAM Programs. In *EUROCRYPT 2017, Part I (LNCS, Vol. 10210)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, Heidelberg, Germany, Paris, France, 501–531. https://doi.org/10.1007/978-3-319-56620-7_18

[34] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. 2019. Distributed Vector-OLE: Improved Constructions and Implementation. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, London, UK, 1055–1072. https://doi.org/10.1145/3319535.3363228

[35] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS 2015*. The Internet Society, San Diego, CA, USA.

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

[37] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 1074–1091. https://doi.org/10.1109/SP40001.2021.00056

[38] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 501–518.

[39] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. 2022. AntMan: Interactive Zero-Knowledge Proofs with Sublinear Communication. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 2901–2914. https://doi.org/10.1145/3548606.3560667

[40] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 2986–3001. https://doi.org/10.1145/3460120.3484556

[41] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast Extension for Correlated OT with Small Communication. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 1607–1626. https://doi.org/10.1145/3372297.3417276

[42] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. 2023. Batchman and Robin: Batched and Non-batched Branching for Interactive ZK. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1257 https://eprint.iacr.org/2023/1257.

[43] Yibin Yang, David Heath, Vladimir Kolesnikov, and David Devecsery. 2022. 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*. IEEE, Genoa, Italy, 109–123. https://doi.org/10.1109/EuroSP53844.2022.00015

A DETAILED COST ACCOUNTING

A.1 Single Disjunction Protocol Costs

Communication. We analyze the communication complexity of $\Pi^{p,q}_{\text{Single}}$ (Figure 5) in the $\mathcal{F}^{p,q}_{\text{sVOLE}}$-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 $n_{in} + (2q+3)n_\times + q(B+7) = O(q|C| + qB)$ elements. We will soon show a simple variant that achieves $O(|C| + qB)$ 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. $\Pi^{p,q}_{\text{Single}}$ requires a total of $n_{in} + 3n_\times + q(B+1) = O(|C| + qB)$ 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 $\mathsf{MUL}_{\mathsf{LEFT}}$ (see Section 4), the computation cost is $O(B|C|)$ field operations. Other Steps require either $O(B)$ or $O(|C|)$ 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).

A.2 Batched Disjunction Protocol Costs

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

Communication. We analyze the communication complexity of $\Pi^{p,q}_{\text{Batch}}$ in the $\mathcal{F}^{p,q}_{\text{sVOLE}}$-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 $(Rq + R + q)n_{in} + (3Rq + 3R + 5q)n_\times + Rq(B+4) + 8q = O(qRB + qR|C|)$ elements.

Number of subfield VOLE correlations. $\Pi^{p,q}_{\text{Batch}}$ requires a total of $(Rq+R)n_{in} + (3Rq+3R)n_\times + qR(B+1) + 2q = O(qRB + qR|C|)$ 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 $\Pi^{p,q}_{\text{Batch}}$ in the $\mathcal{F}^{p,q}_{\text{sVOLE}}$-hybrid model. Our analysis counts field operations:

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

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