SoK: Vector OLE-Based Zero-Knowledge Protocols

Carsten Baum [*1], Samuel Dittmer [†2], Peter Scholl[3], and Xiao Wang[4]

[1] *DTU Compute, Denmark* cabau@dtu.dk
[2] *Stealth Software Technologies Inc., USA* samdittmer@stealthsoftwareinc.com
[3] *Aarhus University, Denmark* peter.scholl@cs.au.dk
[4] *Northwestern University, USA* wangxiao@northwestern.edu

June 7, 2023

1 Introduction

A zero-knowledge proof is a cryptographic protocol where a prover can convince a verifier that a statement is true, without revealing any further information except for the truth of the statement. More precisely, if x is a statement from an NP language verified by an efficient machine M, then a zero-knowledge proof aims to prove to the verifier that there exists a witness w such that $M(x, w) = 1$, without revealing any further information about w. We say that the proof is a proof of knowledge, if the prover additionally convinces the verifier that it knows the witness w, rather than just of its existence. For example, a prover can use a ZK protocol to convince others that it knows an input that can cause stack overflow for some public program (e.g., when submitting a bug report to the Common Vulnerabilities and Exposures system) without revealing the input.

This article is a survey of recent developments in building practical zero-knowledge proof systems using vector oblivious linear evaluation (VOLE), a tool from secure two-party computation. This approach offers several advantages:

Fast prover. VOLE-based proof systems are scalable, meaning that the computational resources required by the prover and verifier are not much larger than what's needed to verify the statement when given the witness in the clear[1]. Concretely, for statements given in a boolean circuit C, a multi-threaded prover incurs essentially no overhead in wall-clock evaluation time over a single-threaded evaluation of C in the clear. As a more concrete example, the authors of [BMRS21] recently demonstrated that evaluating AES as an optimized C-program (without using AES-specific CPU instructions) is $50x$ faster than evaluating AES in the ZK-proof system of [BMRS21].

[*] This work was done while visiting the Basic Algorithms Research Center Copenhagen at Copenhagen University.
[†] Corresponding author.
[1] We suggest to classify proof systems with an explicit focus on fast prover runtime as FLARKs: Fast Linear Arguments of Knowledge.

Small memory. An attractive feature of many VOLE-based protocols is their low memory overhead: just as with plain computation, memory requirements are often only proportional to the cost of verifying the statement (given the witness). This is especially useful for complex statements, where for instance, the witness may be so large that it does not fit into memory, even though the proof can be verified efficiently in a streaming manner.

Post-quantum. With the possibility of large-scale quantum computing on the horizon, protocols based on traditional factoring or discrete log assumptions could become insecure to a quantum attacker. VOLE-based protocols, however, are instead based on variants of the learning parity with noise (LPN) assumption, which is related to the hardness of decoding random linear codes and currently believed to be resistant to quantum attacks.

Conceptual simplicity. VOLE-based protocols can be divided into two phases: a preprocessing phase, which usually consists of running the VOLE protocol on random inputs and which is essentially independent of the statement, and an online phase, where the proof takes place. By abstracting away the properties of the preprocessing, the online phase is very simple to describe, and can even be information-theoretically secure. As one of the simplest, practical ways of constructing zero-knowledge proofs for general statements, it may also be a valuable pedagogical resource.

These benefits also come with a few drawbacks, that can be seen as tradeoffs. Firstly, most VOLE-based ZK proofs have a large communication cost, that is, the amount of data sent between the prover and verifier often scales linearly with the size of the circuit that verifies the statement being proven. This inherently means that the runtime of the verifier must also scale linearly. Another possible drawback is that current constructions of VOLE-based ZK require a designated verifier. That is, the verifier must store a private state needed to verify a proof, which cannot be made public. It's therefore more difficult to prove the same statement to many different verifiers, or in public, with these techniques.

1.1 Overview of this survey

In this work, we attempt to systematize the recent works on VOLE-based Zero-Knowledge proofs and make the state of the art accessible in one document.

In Section 2 we will outline the notation and tools that are the foundation for all VOLE-based ZK. This includes how proven statements can be formalized, the definition of Zero-Knowledge that we achieve as well as an overview of VOLE.

We will then, in Section 3, introduce a general abstraction that unifies the ideas behind the main VOLE-based ZK proofs into an Arithmetic Black Box, and how most of the ZK constructions follow given access to VOLE.

Section 4 is devoted to the main differences in which existing works implement the Multiplication in the Arithmetic Black Box, which is one of the main sources of interaction in the proof. Here, we give an introduction into how the Wolverine, Mac'n'Cheese, Line-Point ZK and QuickSilver protocols work.

Section 5 will discuss how follow-up works have added different operations to the Arithmetic Black Box, thus allowing the prove certain statements more efficiently. This includes ideas such as more efficient proofs of polynomial evaluation, SIMD circuits, disjunctions, conversions or proofs over rings instead of fields. We will moreover consider proofs of RAM programs based on VOLE protocols.

Finally, we will mention some interesting open questions in Section 6.

1.2 Related Techniques in ZK

Zero-knowledge proofs were first introduced by Goldwasser, Micali and Rackoff in 1985 [GMR85]. Since then, there has been a vast body of research in both theoretical and applied settings. Below, we mention a few of the techniques that are most relevant for those in this survey. For a more in-depth coverage, the ZKProof Community Reference[2] aims to give a comprehensive overview of the state-of-the-art.

Garbled Circuits. Zero-knowledge proofs based on garbled circuits were first proposed by Jawurek et al. [JKO13], with the key insight that garbled circuits already provide one-sided malicious security and ZK only needs one-sided privacy. It has many advantages of VOLE-based ZK but requires κ bits of communication per AND gate even after optimizations [FNO15, ZRE15]. More recent works have also expanded this approach to support more efficient disjunctive proofs [HK20].

MPC-in-the-Head. MPC-in-the-head by Ishai et al. [IKOS07] is an elegant way of constructing ZK proofs based on secure multi-party computation. Its concrete efficiency was first studied by Giacomelli et al. [GMO16], which has led to a long line of practical ZK proofs in recent years, particularly for designing digital signatures. It could be made non-interactive but often needs communication linear in the circuit size.

SNARKs. In recent years, there has been a large focus on Succint non-interactive arguments of knowledge (SNARKS) [GGPR13], which are protocols where the communication complexity, that is, the size of the proof, is very small, potentially even constant size or logarithmic in the witness length. A drawback of most SNARKs is that succinctness comes at the cost of a more expensive prover, which often has super-linear computational complexity (with a few exceptions [ZLW+21, GLS+21]) and large memory requirements (linear to the statement size).

1.3 Applications

VOLE-based ZK proofs enjoy high efficiency and scalability: they could prove tens of millions of gates even under a small bandwidth connection and low-configured hardware. As a result, it has the potential to enable many exciting applications.

1. **Proofs of program properties.** When expressing properties about a program Π formally, automated theorem provers can allow formally attesting the presence of certain properties using a proof π. By encoding the verification of a given proof for a public Π as the input to a ZK proof system, a prover can e.g. show that a program shows certain information leakage, without revealing the proof π that explains this behavior [PHP+].
2. **Proofs of machine-learning tasks.** Machine-learning tasks often involve sensitive data (e.g. biometric information) or valuable data (e.g., large models) where ZK proofs could help to enhance privacy. Until recent, ZK for ML is limited because statements on ML are usually

[2] `https://docs.zkproof.org/reference`

large. Recent works have shown the feasibility of proving inference of deep neural networks in ZK [WYX+21, LXZ21].

3. **Proofs of signature validity on private messages.** [PRO] considers a case where applicants would like to prove their medical qualification without revealing how they qualify and which healthcare provider signed the evidence. This requires proving that a private digital document is signed by someone from an public authorized list and that the same document implies medical validity.

4. **Proofs of unsatisfiability.** The correctness verification of computer programs is commonly done by showing the unsatisfiability of a certain SAT formula, which is decided by the program and the property to be proven. Proving formula unsatisfiability in ZK [LAH+22] could enable applications like 1) showing the correctness of a public program without revealing why; and 2) showing the correctness of a private program.

2 Preliminaries

We use lower case, bold symbols for vectors $\mathbf{x}$ and upper case, bold symbols for matrices $\mathbf{A}$. We use κ as the computational and σ as the statistical security parameter. Generally, the prover is denoted as $\mathcal{P}$ while the verifier is $\mathcal{V}$. In our UC functionalities and proofs, $\mathcal{Z}$ denotes the environment, and $\mathcal{S}$ is the simulator, while $\mathcal{A}$ will refer to the adversary. When we say that an algorithm is Probabilistic Polynomial Time (PPT), then we mean that it can be expressed as a probabilistic interactive turing machine whose worst-case runtime can be expressed as a polynomial in κ. For any finite set S, we denote by $|S|$ the cardinality of S. If instead s is a string, then $|s|$ denotes its length. If s_1, s_2 are strings then $s_1||s_2$ denotes the concatenation of strings.

2.1 The Computational Model: Arithmetic Circuits

The zero-knowledge proofs in this survey are used by a PPT prover $\mathcal{P}$ to convince a PPT verifier $\mathcal{V}$ that a certain statement is true, and furthermore, that $\mathcal{P}$ knows a witness for the statement. This means they are proofs of knowledge. Abstractly, both $\mathcal{P}$ and $\mathcal{V}$ consider a language $\mathcal{L}$ together with a relation $\mathcal{R}_{\mathcal{L}}$. For a string x input to both $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{P}$ will convince $\mathcal{V}$ that it knows a w such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$, i.e. $x \in \mathcal{L}$. Here, $\mathcal{R}_{\mathcal{L}}$ is an NP relation, which means that there exists a Turing Machine (TM) M which, on input x, w, accepts in time $poly(|x|)$ iff $x \in \mathcal{L}$. Instead of expressing computation as happening on a Turing machine M, we will require that each x can, in time $poly(|x|)$, be converted into a circuit C over a ring $(R, +, \times)$ whose gates correspond to efficiently computable[3] functions defined over R. We require that $C(w) = 0$ iff $(x, w) \in \mathcal{R}_{\mathcal{L}}$, except with negligible probability in κ. This requirement is without loss of generality, as such a circuit can always be constructed using the Cook-Levin Theorem.

More concretely, our statements are circuits C over a ring R. We define their semantics as follows: Consider the tuple $C = (n_{in}, n_{out}, n_g, \mathtt{I}, \mathtt{G})$ where

- $n_{in} \geq 2$ is the number of input wires, $n_{out} \geq 1$ the number of output wires and $n_g \geq 1$ the number of gates in the circuit. We let $n_w = n_{in} + n_g$ be the number of wires.

[3] Here, efficiently computable means that evaluating the gate should take time polynomial in the input- and output length of the gate as well as $\log(|R|)$.

- We define the sets $Inputs \leftarrow \{1, \ldots, n_{in}\}$, $Wires \leftarrow \{1, \ldots, n_w\}$ as well as $Outputs \leftarrow \{n_w - n_{out} + 1, \ldots, n_w\}$ and $Gates \leftarrow \{n_{in} + 1, ..., n_w\}$ to identify the respective elements in the circuit.
- The poly-time computable function $\mathtt{I} : Gates \mapsto 2^{Wires \setminus Outputs}$ identifies the incoming wires for each gate, with the restrictions that:
  - $\forall g \in Gates : \mathtt{I}(g) \neq \emptyset$.
  - $\forall g \in Gates : \max_{s \in \mathtt{I}(g)}\{s\} < g$.
- The poly-time computable mapping $\mathtt{G} : Gates \mapsto (R^+ \mapsto R)$ determines the function that is computed by a gate. We require that
  1. $\forall g \in Gates$: The input length of $\mathtt{G}(g)$ is identical to $|\mathtt{I}(g)|$.
  2. $\forall g \in Gates$: The function $\mathtt{G}(g)$ can be computed in time $poly(|\mathtt{I}(g)|, \log(|R|))$.

To obtain the outputs of the above circuit when evaluating it on an input $w = w_1 || \cdots || w_{n_{in}} \in R^{n_{in}}$ one evaluates C as follows:

$eval(C, w)$**:**

1. For $i \in \{1, \ldots, n_{in}\}$ set $x_i = w_i$.
2. For $g \in \{n_{in} + 1, \ldots, n_w\}$:
   (a) $(s_1, \ldots, s_{|\mathtt{I}(g)|}) \leftarrow \mathtt{I}(g)$ where $s_i < s_{i+1}$
   (b) $f \leftarrow \mathtt{G}(g)$
   (c) $x_g \leftarrow f(x_{s_1}, \ldots, x_{s_{|\mathtt{I}(g)|}})$
3. Output $x_{n_w - n_{out}+1} || \cdots || x_{n_w}$

We denote by $C(w)$ the aforementioned evaluation of C on input w.

2.2 Zero Knowledge Proofs for Circuits

We define Zero-Knowledge Proofs (of Knowledge) in the Ideal-Real paradigm. Let Π be an interactive protocol between two PPT interactive Turing Machines (iTMs) $\mathcal{P}, \mathcal{V}$. This means that parties might send messages to each other, as well as to idealized functionalities. Define the functionality $\mathcal{F}_{\mathsf{ZK}}$ as in Figure 1.

Let $\mathcal{A}$ be a PPT iTM algorithm, called the adversary. $\mathcal{A}$ is allowed to corrupt either of the parties, or none at all. If a party is corrupted, then $\mathcal{A}$ will have full control over that party and be allowed to read all its secrets and send any messages on its behalf. It has to specify in the beginning which party, if at all, $\mathcal{A}$ will corrupt.

We define security with respect to a PPT iTM $\mathcal{Z}$ called environment. The environment provides inputs to and receives outputs from the parties. Furthermore, the adversary $\mathcal{A}$ will corrupt a party in the name of $\mathcal{Z}$. To define security, let $\Pi \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance Π. Furthermore, let $\mathcal{S}$ denote an ideal world adversary and $\mathcal{F}_{\mathsf{ZK}} \circ \mathcal{S}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}_{\mathsf{ZK}}$ instead of Π and where $\mathcal{S}$ takes care of adversarial behavior.

Zero-Knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$

The functionality interacts with two parties $\mathcal{P}, \mathcal{V}$ as well as an adversary $\mathcal{S}$ who may corrupt either of these.

Prove: Upon input (Prove, w, C) by $\mathcal{P}$ where C is a circuit with gates in R, input length ℓ and output length 1, and $w \in R^\ell$ as well as (Prove, C) by $\mathcal{V}$

1. Compute $y = C(w)$ and send (Prove, C, y) to $\mathcal{S}$. If $\mathcal{S}$ sends (Abort) then send (Abort) to both parties and terminate, otherwise continue.
2. If $y = 0$ then send $(\mathsf{Correct}, C)$ to $\mathcal{V}$, otherwise send $(\mathsf{Incorrect}, C)$.

Figure 1: Functionality for ZK proofs over the ring R.

Definition 2.1 (Zero-Knowledge Proof of Knowledge)**.** We say that Π is a Zero-Knowledge Proof of Knowledge if for every PPT iTM $\mathcal{A}$ that maliciously corrupts at most 1 party there exists a PPT iTM $\mathcal{S}$ (with black-box access to $\mathcal{A}$) such that no PPT environment $\mathcal{Z}$ can distinguish $\Pi \circ \mathcal{A}$ from $\mathcal{F}_{\mathsf{ZK}} \circ \mathcal{S}$ with non-negligible probability in κ.

2.3 Vector Oblivious Linear Evaluation (VOLE)

A VOLE correlation is a pair of random variables $(\mathbf{u}, \mathbf{x})$ and $(\mathbf{v}, \Delta)$, where $\mathbf{x}, \mathbf{u}, \mathbf{v}$ are vectors and Δ is a scalar, which are all random subject to the constraint that

$$u_i = v_i + x_i \cdot \Delta$$

One party, in our case the prover $\mathcal{P}$, is given $\mathbf{u}, \mathbf{x}$, while the verifier $\mathcal{V}$ learns $(\mathbf{v}, \Delta)$.

We model the generation of VOLE correlations as an ideal functionality $\mathcal{F}_{\mathsf{VOLE}}$, given in Figure 2. The functionality works over a ring R; in most cases, we require that R is a finite field, but in Section 5.5 we also discuss how to support non-field rings such as $R = \mathbb{Z}_{2^k}$, the integers modulo 2^k. On initialization, the functionality samples a MAC key $\Delta \in R_{\mathsf{key}}^t$, where $R_{\mathsf{key}} \subset R$ for some parameter $t \geq 1$ such that $|R_{\mathsf{key}}^t|$ is exponentially large in the security parameter. When R is a field, we typically choose $R_{\mathsf{key}} = R$. After initialization, the Extend command may be called repeatedly. On each call, it samples one more "element" of the VOLE correlation, which we view as a MAC $M[x]$ on a random element x given to the prover, where the verifier learns only the corresponding key $K[x]$ (as well as the global key Δ).

In the rest of this paper, we assume that such a functionality can be efficiently realized using a secure VOLE protocol. To justify this, we outline which approaches currently exist to implement $\mathcal{F}_{\mathsf{VOLE}}$ efficiently with active security.

Instantiate $\mathcal{F}_{\mathsf{VOLE}}$ directly. The two most popular approaches use linearly homomorphic encryption or OT extension protocols. For homomorphic encryption, the approach is usually that $\mathcal{V}$ samples a public key/private key pair, then sends an encryption of Δ to $\mathcal{P}$. $\mathcal{P}$ picks $x, M[x]$ and, using the homomorphism, computes an encryption of $K[x]$ that it sends to $\mathcal{V}$. Finally, $\mathcal{V}$ can decrypt this result. While it is easy to achieve passive security by rerandomizing the ciphertext containing $K[x]$ appropriately, achieving active security usually requires additional consistency checks

Vector Oblivious Linear Evaluation functionality $\mathcal{F}_{\mathsf{VOLE}}$

Init: This method needs to be the first one called by the parties. On input (Init) from both parties:

1. If $\mathcal{V}$ is honest, the functionality samples $\Delta \in_R R_{\mathsf{key}}^t$ and sends Δ to $\mathcal{V}$.
2. If $\mathcal{V}$ is corrupt, the functionality receives $\Delta \in R_{\mathsf{key}}^t$ from $\mathcal{S}$.
3. Δ is then stored by the functionality.

All further Init queries are ignored.

Extend On input (Extend) from both parties the functionality proceeds as follows:

1. If both parties are honest, sample $x \in_R R, K[x] \in_R R^t$ and compute $M[x] \leftarrow \Delta \cdot x + K[x]$.
2. If $\mathcal{V}$ is corrupted, receive $K[x] \in R^t$ from $\mathcal{S}$ instead.
3. If $\mathcal{P}$ is corrupted, receive $x \in R, M[x] \in R^t$ from $\mathcal{S}$ instead, and compute $K[x] \leftarrow M[x] - \Delta \cdot x$.
4. $(x, M[x])$ is sent to $\mathcal{P}$ and $K[x]$ is sent to $\mathcal{V}$.

Figure 2: Functionality for VOLE over R^t with a message from the ring R, and scalar Δ from R_{key}^t, where $R_{\mathsf{key}} \subset R$.

such as specialized zero-knowledge proofs. See e.g. [BEPU+20, dCJV21, dCHI+22] for variants on this approach. An alternative solution is to use Oblivious Transfers to perform the multiplications, leading to highly efficient protocols such as [KOS16, Roy22], or [Sch18] when R is a ring such as $\mathbb{Z}_{2^k}$. The disadvantage of all these protocols is that the communication between $\mathcal{P}$ and $\mathcal{V}$ scales at least linearly in the number of VOLE correlations n, which can easily become a bottleneck when a large number of correlations are needed.

Extend VOLEs efficiently. Current state-of-the-art VOLE extension protocols all stem from the approach of Boyle et al. [BCGI18], which builds a pseudorandom correlation generator based on (variants of) the learning parity with noise (LPN) assumption. This approach exploits the fact that sparse LPN errors can be used to compress secret-sharings of pseudorandom vectors, allowing the two parties to generate a long, pseudorandom instance of a VOLE correlation in a succinct manner from a short vector of VOLE correlations.

These protocols usually proceed along the following lines:

1. Construct a protocol for single-point VOLE, where the sender's input vector has only a single non-zero entry.
2. The single-point VOLE protocol is repeated t times, to obtain a t-point VOLE where the sender's input is viewed as a long, sparse, LPN error vector.

3. Combine t-point VOLE and the LPN assumption, allowing the parties to locally obtain pseudorandom VOLE by applying a linear mapping.

Using this blueprint leads to (random) VOLE protocols with communication much smaller than the output length, which is sufficient to build Zero-Knowledge protocols as we shall see. It can be seen as a form of VOLE extension, where in the first step, a small "seed" VOLE of length $m \ll n$ is used to create the single-point VOLEs, and then extended into a longer VOLE of length n. In the Ferret protocol [YWL+20], it was additionally observed that when repeating this process, it can greatly help communication if m of the n extended outputs are reserved and used to bootstrap the next iteration of the protocol, saving generation of fresh seed VOLEs whose computation is usually more involved as outlined above.

To use VOLE as part of a ZK protocol, we will need that it is actively secure. If R is a field, then VOLE extension can efficiently be done by picking a protocol such as [BCG+19a, WYKW21]. These also allow the secret x to be from a subfield of R, which yields more efficient constructions when the proof circuit is defined over a small field such as $\mathbb{F}_2$. For $R = \mathbb{Z}_{2^k}$, the recent work of [BBMHS22] described how to adapt [BCGI18, WYKW21] with a consistency check that is secure if the underlying LPN instance tolerates a small amount of leakage on the noise vector.

2.4 Schwartz-Zippel Lemma

A crucial building block in all presented protocols is the Schwartz-Zippel Lemma over finite fields, which allows for efficient polynomial identity tests. The version which we use, proven by Ore [Ore22], works as follows:

Lemma 2.1 (Schwartz-Zippel Lemma)**.** *Let $\mathbb{F}$ be a finite field, $S \subseteq \mathbb{F}$ and $P \in \mathbb{F}[X]$ be a non-zero polynomial of degree $d \geq 0$. Then*

$$\Pr_{s \xleftarrow{\$} S}[P(s) = 0] \leq \frac{d}{|S|}.$$

3 A General Framework for VOLE-based ZK

3.1 Homomorphic MACs from VOLE

VOLE can be used to build a simple, information-theoretic MAC scheme with useful homomorphic properties. Prior works have shown numerous MAC schemes with different properties that follow a similar paradigm [BDOZ11, DPSZ12, DZ13, CF13]. The MAC scheme is oblivious, in the sense that the prover will hold MACs on certain values, while only the verifier knows the corresponding MAC key. For example, consider running VOLE over a finite field $\mathbb{F}$, i.e. $\mathcal{F}_{\mathsf{VOLE}}$ with $R = \mathbb{F}$, $R_{\mathsf{key}} = R$ and $t = 1$. A single output from a random VOLE can be seen as a MAC on the value $x \in \mathbb{F}$ obtained by the prover. The prover also learns the MAC $M[x]$, while the verifier holds the MAC key, which consists of a random $K[x] \in \mathbb{F}$ and the fixed key $\Delta \in \mathbb{F}$, satisfying

$$K[x] = M[x] - \Delta \cdot x$$

If the prover wants to send x to the verifier, this can be authenticated by additionally sending $M[x]$: the verifier simply checks the above equation holds.

The MAC cannot be forged with probability larger than $1/|\mathbb{F}|$. To see this, consider a cheating prover who sends $x' \neq x$ together with a MAC $M[x']$. If verification succeeds, we have $M[x] - \Delta \cdot x =$

$M[x'] - \Delta \cdot x'$, and so $(M[x] - M[x']) \cdot (x - x')^{-1} = \Delta$. This implies that the prover must have guessed Δ, by coming up with $x', M[x']$ that pass the check. Crucially, this check relies on $x - x'$ being invertible, which in the given case[4] is of course always true.

Linear Homomorphism. Since the MAC equation is linear, and Δ is fixed for every VOLE output, it's easy to see that any public, linear function can be applied to MACs. The parties can also create a MAC on a public constant $c \in \mathbb{F}$, by defining $M[c] = 0$ and $K[c] = -c \cdot \Delta$; this allows homomorphically computing affine functions.

Multiplicative Homomorphism. The MACs are also multiplicatively homomorphic, with the caveat that the storage complexity increases. To see this, let $(x, M[x])$ held by the prover define a linear polynomial $p(s) = M[x] + x \cdot s$ in s. The verifier then holds the random key Δ, and the evaluation $p(\Delta)$. Now consider a second such MAC on y, and polynomial $q(s) = M[y] + y \cdot s$. The product $p(s)q(s)$ is now a degree-two polynomial, whose coefficients are held by the prover.

The drawback of homomorphically multiplying MACs is that the size of the resulting MAC scales with the number of multiplications (i.e. the degree of the function). However, this can still be exploited, as we see in Sections 4.3 and 4.3.3.

MACs Over Small Fields. The approach outlined above does not achieve sufficient security if $|\mathbb{F}|$ is small. For example, when $\mathbb{F} = \mathbb{F}_2$, the MAC only delivers 1 bit of security! Luckily, the approach generalizes to arbitrary $t > 1$. Namely, let $R_{\mathsf{key}} = R^t$ where $R = \mathbb{F}$ for an arbitrary finite field. The same security argument as before does apply: if a cheating prover who sends $x' \neq x$ together with a MAC $M[x']$ succeeds in verification, then we have $M[x] - \Delta \cdot x = M[x'] - \Delta \cdot x'$, and therefore $(M[x] - M[x']) \cdot (x - x')^{-1} = \Delta$ which can be computed over R^t by coordinate-wise division by $x - x'$. Therefore, a forgery can now only happen with probability $1/|\mathbb{F}|^t$.

$\mathcal{F}_{\mathsf{VOLE}}$ for this setting can efficiently be instantiated by considering R as a subfield of R^t using the machinery from [BCG+19b]. It also lends itself to updates on the MACed value x with low communication, since only a value over R but not R^t must be communicated to do so. The linear homomorphism of the MAC scheme again follows directly from its setup. By considering R^t as the degree-t extension field of R instead of just a vector space, the multiplicative homomorphism (and how it is exploited in this work) can also be recovered.

In Section 5.5, we show how this type of homomorphic MAC can also be made to work over rings, with some differences to the soundness guarantees and repercussions on ring size.

3.2 Arithmetic Black Box for ZK

The functionality $\mathcal{F}_{\mathsf{ZK}}$ introduced in Figure 1 is only able to process a circuit C over a ring in a block. However, VOLE-based ZK can often provide a more flexible functionality where parties can prove the circuit progressively in a gate-by-gate manner. This is crucial e.g. for memory-friendliness, since wires which are no longer needed can be dropped from memory. To abstract this capability in VOLE-based ZK, we now refine $\mathcal{F}_{\mathsf{ZK}}$ into $\mathcal{F}_{\mathsf{ABB}}$ in Figure 3 which performs exactly this job. To use $\mathcal{F}_{\mathsf{ABB}}$ to realize $\mathcal{F}_{\mathsf{ZK}}$, two parties can use the Input to obtain handles to the committed witness; then they can traverse the circuit following topological order: each linear gate can be computed

[4] We will later in Section 5.5 see an example where this is not the case and why this leads to problems.

using Affine combination and each non-linear gate can be computed using multiplication check. In the end, two parties hold a handle for the output wire which can be asserted using $\mathsf{CheckZero}$.

Arithmetic black box $\mathcal{F}_{\mathsf{ABB}}$

The functionality communicates with two parties $\mathcal{P}, \mathcal{V}$ as well as an adversary $\mathcal{S}$ that may corrupt either party. $\mathcal{S}$ may at any point send a message (abort), upon which $\mathcal{F}_{\mathsf{ABB}}$ sends (abort) to all parties and terminates. $\mathcal{F}_{\mathsf{ABB}}$ contains a state st that is initially $\emptyset$.

Random On input $(\mathsf{Random}, \mathsf{id})$ from $\mathcal{P}, \mathcal{V}$ and where $(\mathsf{id}, \cdot) \notin \mathsf{st}$:

1. If $\mathcal{P}$ is corrupted, obtain $x_{\mathsf{id}} \in R$ from $\mathcal{S}$. Otherwise sample $x_{\mathsf{id}} \in_R R$ uniformly at random.
2. Set $\mathsf{st} \leftarrow \mathsf{st} \cup \{(\mathsf{id}, x_{\mathsf{id}})\}$ and send x_{id} to $\mathcal{P}$.

Input On inputs $(\mathsf{Input}, \mathsf{id}, x)$ from $\mathcal{P}$ and $(\mathsf{Input}, \mathsf{id})$ from $\mathcal{V}$ and where $(\mathsf{id}, \cdot) \notin \mathsf{st}$:

1. Set $\mathsf{st} \leftarrow \mathsf{st} \cup \{(\mathsf{id}, x)\}$.

Affine Combination On input $(\mathsf{Affine}, \mathsf{id}_o, \mathsf{id}_1, \ldots, \mathsf{id}_n, \alpha_0, \ldots, \alpha_n)$ from $\mathcal{P}, \mathcal{V}$ where $(\mathsf{id}_i, x_{\mathsf{id}_i}) \in \mathsf{st}$ for $i \in \{1, \ldots, n\}$ and $(\mathsf{id}_o, \cdot) \notin \mathsf{st}$:

1. Set $x_{\mathsf{id}_o} \leftarrow \alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot x_{\mathsf{id}_i}$ and $\mathsf{st} \leftarrow \mathsf{st} \cup \{(\mathsf{id}_o, x_{\mathsf{id}_o})\}$.

Check Zero On input $(\mathsf{CheckZero}, \mathsf{id}_1, \ldots, \mathsf{id}_n)$ from $\mathcal{P}, \mathcal{V}$ and where $(\mathsf{id}_i, x_{\mathsf{id}_i}) \in \mathsf{st}$ for $i \in \{1, \ldots, n\}$:

1. If $x_{\mathsf{id}_1} = \cdots = x_{\mathsf{id}_n} = 0$, then send $(\mathsf{success})$ to $\mathcal{V}$, otherwise send (abort) to all parties and terminate.

Multiplication Check Upon $\mathcal{P}$ & $\mathcal{V}$ inputting $(\mathsf{CheckMult}, (\mathsf{id}_{x,i}, \mathsf{id}_{y,i}, \mathsf{id}_{z,i})_{i=1}^{n})$ where $(\mathsf{id}_{x,i}, x_i), (\mathsf{id}_{y,i}, y_i), (\mathsf{id}_{z,i}, z_i) \in \mathsf{st}$ for $i \in \{1, \ldots, n\}$:

1. Send $(\mathsf{success})$ to $\mathcal{V}$ if $x_i \cdot y_i = z_i$ holds for all $i \in \{1, \ldots, n\}$, otherwise send (abort) to all parties and terminate.

Figure 3: Functionality modeling an arithmetic black box over the ring R.

This functionality $\mathcal{F}_{\mathsf{ABB}}$ is what protocols such as [WYKW21], [BMRS21] and follow-ups actually implement. Their observation is that Vector-OLEs output by $\mathcal{F}_{\mathsf{VOLE}}$ can be used to securely store inputs by $\mathcal{P}$ such that linear functions of secrets can be computed without interaction. In the following, whenever a value x is stored inside $\mathcal{F}_{\mathsf{ABB}}$, then we denote it as $[x]$. This is equivalent to the value x being MACed (as outlined in Section 3.1), which is why the same notation is used. If $R = \mathbb{Z}_s$ for some $s \in \mathbb{N}$, then we write $[x]_s$ to clarify the modulus used in $\mathcal{F}_{\mathsf{ABB}}$. We leave out the subscript if the ring is clear from the context.

To realize $\mathcal{F}_{\mathsf{ABB}}$ for the case where $R = \mathbb{F}$, both parties initially call **Init** of $\mathcal{F}_{\mathsf{VOLE}}$ to make commitments available. Then they proceed as follows.

Random

1. The parties call **Extend** on $\mathcal{F}_{\mathsf{VOLE}}$ and assign the returned $(r, M[r], K[r])$ the id id.

Input

1. The parties call **Extend** on $\mathcal{F}_{\mathsf{VOLE}}$, which returns $(r, M[r])$ to $\mathcal{P}$ and $K[r]$ to $\mathcal{V}$.
2. $\mathcal{P}$ computes $\delta = x - r$ over R and sends δ to $\mathcal{V}$.
3. $\mathcal{P}$ sets $M[x] \leftarrow M[r]$ while $\mathcal{V}$ sets $K[x] \leftarrow K[r] - \Delta\delta$ and both parties assign $(x, M[x], K[x])$ the id id.

Check Zero If only one id needs to be checked, then this can be done as follows:

1. $\mathcal{P}$ looks up $M[0]$ for the id id, while $\mathcal{V}$ looks up $K[0]$. If it is undefined, then abort.
2. $\mathcal{P}$ sends $M[0]$ to $\mathcal{V}$, who checks that $K[0] = M[0]$.

If more than one id needs to be checked, then $\mathcal{P}, \mathcal{V}$ can also apply a Collision-Resistant Hash Function to compress their values and save communication bandwidth.

Affine Combination

1. $\mathcal{P}$ looks up $x_1, \ldots, x_n, M[x_1], \ldots, M[x_n]$ for the ids $\mathsf{id}_1, \ldots, \mathsf{id}_n$, while $\mathcal{V}$ looks up $K[x_1], \ldots, K[x_n]$. If either of these is undefined, or id_0 is already defined, then abort.
2. $\mathcal{P}$ locally sets $x_0 \leftarrow \alpha_0 + \sum_i \alpha_i x_i$ and $M[x_0] \leftarrow \sum_i \alpha_i M[x_i]$, while $\mathcal{V}$ locally sets $K[x_0] = \sum_i \alpha_i K[x_i] - \alpha_0 \Delta$.

In the case where $R = \mathbb{F}$ then one can easily write a simulator, following Definition 2.1, to show that the aforementioned subprotocols implement the desired parts of the functionality $\mathcal{F}_{\mathsf{ABB}}$ securely. The main idea is that $\mathcal{V}$ does not learn any information, since the outputs of $\mathcal{F}_{\mathsf{VOLE}}$ leak no information about the outputs given to $\mathcal{P}$, while every input during **Input** is blinded using a uniformly-random value from $\mathcal{F}_{\mathsf{VOLE}}$. **Affine Combination** is entirely non-interactive, while the value that $\mathcal{V}$ obtains during **Check Zero** is predetermined. A cheating $\mathcal{P}$ can only change the outputs during **Check Zero** such that $\mathcal{V}$ accepts a non-zero value, but as shown in Section 3.1, this reduces to $\mathcal{P}$ being able to guess Δ. This value is never revealed to $\mathcal{P}$ by $\mathcal{F}_{\mathsf{VOLE}}$, which finishes the claim. The exact details are shown in the referenced works[5].

What is left to implement is the **Multiplication Check** of $\mathcal{F}_{\mathsf{ABB}}$. This is actually the core of much of the early work on VOLE-based Zero Knowledge. We will summarize the state-of-the-art in the following section.

4 Multiplication checks

4.1 Wolverine Multiplication Check

The Wolverine multiplication check protocol [WYKW21] can be viewed as a direct application of the bucketing technique introduced in the context of malicious secure computation [DPSZ12, NNOB12, NO09]. The only difference is that only the prover has a privacy requirement (i.e. zero-knowledge) and thus the bucketing only needs to be done for one layer. In more detail, the protocol proceeds in the following steps:

[5]The case where R is of different form will be treated in Section 5.5.

Wolverine Multiplication Check

Inputs and parameters: Two parties hold authenticated values $\{([x_i], [y_i], [z_i])\}_{i\in[n]}$. Fix parameters B and let $\ell = n \cdot B + c$.

Protocol:

1. Two parties use $\mathcal{F}_{\mathsf{VOLE}}$ to obtain authenticated values $\{([a_i], [b_i], [r_i])\}_{i\in[\ell]}$.
2. For $i \in [\ell]$, $\mathcal{P}$ sends $d_i := a_i \cdot b_i - r_i$ to $\mathcal{V}$, and then both parties compute $[c_i] := [r_i] + d_i$.
3. $\mathcal{V}$ samples a random permutation π on $\{1, \ldots, \ell\}$ and sends it to $\mathcal{P}$. The two parties use π to permute $\{([a_i], [b_i], [c_i])\}_{i\in[\ell]}$.
4. For each $i \in [n]$, two parties perform the following for $j = 1, \ldots, B$:
   (a) Let $([a], [b], [c])$ be the $\big((i-1)B+j\big)$th authenticated triple (after applying π above).
   (b) The prover sends $\delta_a = x_i - a$ and $\delta_b = y_i - b$ to the verifier. The two parties then compute $[\mu] := [c] - [z_i] + \delta_b \cdot [a] + \delta_a \cdot [b] + \delta_a \cdot \delta_b$, and finally run $\mathsf{CheckZero}([x_i] - [a] - [\delta_a], [y_i] - [b] - [\delta_b], [\mu])$.
5. For each of the remaining B authenticated triples, say $([a], [b], [c])$, the prover sends $a^* = a, b^* = b$ to the verifier. Two parties compute $[d] := [c] - a^* \cdot b^*$ and then run $\mathsf{CheckZero}([a] - [a^*], [b] - [b^*], [d])$.

Figure 4: The Wolverine multiplication check protocol.

1. Given a list of authenticated tuples to be checked, two parties generate $nB + c$ number of extra random authenticated multiplication tuples that are correct if the prover is honest.
2. The verifier randomly picks c tuples out of the $nB+c$ newly generated ones and checks if they have the correct relationship. If so, the remaining nB random authenticated multiplication triples must have a high proportion of good triples. Since all tuples are committed, the check can be done at the end of the protocol as well.
3. For the remaining nB triples, the verifier specifies a random permutation to group them randomly into n buckets each with B triples per bucket.
4. For the i-th input triple, all triples in the i-th bucket are "sacrificed" to check the correctness one at a time. The sacrifice procedure does detect cheating unless both the "input" triple and the "sacrificed" triple are incorrect simultaneously.

Details of the protocol can be found in Figure 4. A careful analysis shows that by setting c, B appropriately, we can ensure that the proportion of incorrect triples that survive after step 2 of the outlined protocol is low. Then, with overwhelming probability during Step 4 not all B "sacrificing" tuples in a bucket can be faulty, as the permutation is chosen at random. Therefore, if any "input" multiplication tuple was faulty, it would be detected during Step 4 and the check would fail. In summary:

Theorem 4.1. *For any field $R = \mathbb{F}$ and integer t and if $c \geq B$, the protocol in Figure 4 securely instantiates* CheckMult *in Figure 3 with statistical error of* $1/\binom{\ell}{B} + O(\ell/|\mathbb{F}|^t)$.

4.2 Mac'n'Cheese Multiplication Check

The protocol from Section 4.1 requires $3B$ sent ring elements per verified multiplication. We now discuss two different approaches. The first only sends 2 ring elements per multiplication check for large fields and builds on Beaver's circuit randomization technique[Bea92]. The second builds on a protocol from Boneh et al. [BBC+19], where the idea is to reduce proving n multiplicative relations to checking a dot product of length n. This comes at the cost of communicating $n + O(\log(n))$ R-elements. In particular, for $R = \mathbb{F}_p$ for $p = 2^{61} - 1$ their protocol requires around 64.3 bits of communication per multiplication.

4.2.1 Warm-up: Multiplication Checks using Circuit Randomization

Consider $\mathcal{P}$ has created $[x], [y], [z]$ and wants to show that $z = x \cdot y$. To do so, first both $\mathcal{P}, \mathcal{V}$ use $\mathcal{F}_{\mathsf{VOLE}}$ to create a random $[a]$. Additionally, $\mathcal{P}$ creates $[c]$ where $c = a \cdot y$.

Upon obtaining a challenge e from $\mathcal{V}$, both parties now compute $[\varepsilon] = e \cdot [x] - [a]$ and $\mathcal{P}$ sends ε to $\mathcal{V}$. Then, $\mathcal{P}$ shows that both $[\varepsilon] - \varepsilon$ and $e \cdot [z] - [c] - \varepsilon[y]$ are commitments to 0.

Assuming that $[\varepsilon]$ was indeed opened correctly, consider the case where $[z] = [x \cdot y + \delta]$ for a non-zero δ, while $[c] = [ay + \gamma]$ for a possibly non-zero γ. Assume that $e \cdot [z] - [c] - \varepsilon[y]$ is indeed a commitment to 0. Then once can easily show that this implies that $e\delta = \gamma$, implying that $e = \gamma/\delta$ for γ, δ that $\mathcal{P}$ has to choose before it knows e. This in turn only succeeds with probability $1/p^k$, as e is chosen uniformly at random. Therefore, for small fields this test has to be repeated multiple times to achieve low enough soundness error.

4.2.2 Multiplication Checks via Inner Product Checks

Boneh et al. [BBC+19] introduce a logarithmic-sized proof for "parallel-sum" circuits. In a "parallel-sum" circuit, identical subcircuits C' are evaluated in parallel on possibly different inputs, with the sum of the outputs of each C' being the output of the overall circuit. The high-level idea of the proof protocol is to embed checks for different instances of C' within a single polynomial, allowing $\mathcal{V}$ to verify n instances of C' in parallel. When letting C' be a single multiplication of its two inputs, can then be used to simultaneously verify the sum of n multiplications, which is equivalent to a dot product. Denote the protocol that checks the dot product AssertDotProduct.

The AssertDotProduct protocol works as follows. Suppose $\mathcal{P}$ wants to prove that $[z] = \sum_{i \in [n]} [x_i][y_i]$. $\mathcal{P}$ begins by defining n polynomials $f_1, \ldots, f_{n/2}, g_1, \ldots, g_{n/2}$ such that $f_i(j) = x_{(j-1)n/2+i}$ and $g_i(j) = y_{(j-1)n/2+i}$ for $j \in \{1, 2\}$, and then computing $h = \sum_{i \in [n/2]} f_i g_i$. $\mathcal{P}$ then commits to h by committing to its coefficients (denoted as $[h]$). $\mathcal{V}$ defines its own polynomials f'_i, g'_i over the values $[x_{(j-1)n/2+i}]$ and $[y_{(j-1)n/2+i}]$ that are stored in $\mathcal{F}_{\mathsf{ABB}}$ to check that $\sum_{i \in [n/2]} f'_i g'_i = h$. By Schwartz-Zippel, this can be done by checking that

$$\sum_{i \in [n/2]} f'_i(r) g'_i(r) = h(r)$$

for a random r chosen by $\mathcal{V}$. Here, observe that the evaluation of f'_i, g'_i, h in a public constant r boils down to multiplying the committed coefficients of each polynomial with appropriate powers of r and summing up the result, both of which are linear operations in $\mathcal{F}_{\mathsf{ABB}}$ that do not require any

interaction. Then, verifying the above equation after fixing r is again a dot product check, although over vectors of length $n/2$, and we can recursively apply AssertDotProduct until n is of constant size. Note that only 4 R-elements are communicated during one iteration of AssertDotProduct: 3 when committing to h and one when sending r. See Figure 5 for a formal presentation of the protocol. There, for the base-case of AssertDotProduct, one can e.g. use the multiplication checking procedure from Section 4.2.1.

Given AssertDotProduct, we can batch-verify n multiplications as follows:

1. Assume that n tuples $[x_i], [y_i], [z_i]$ have been committed by $\mathcal{P}$.
2. $\mathcal{V}$ chooses a randomization factor r that it sends to $\mathcal{P}$.
3. $\mathcal{P}$ shows that $\langle r^i[x_i], [y_i]\rangle = \sum_{i\in[n]} r^i[z_i]$. Since r is public, computing $r^i[x_i]$ and $\sum_{i\in[n]} r^i[z_i]$ is local.

This protocol, called AssertMultVec, is presented in Figure 5.

It is clear that both AssertDotProduct and AssertMultVec are complete and zero-knowledge. The following theorem, proven in [BMRS21], shows they are also sound.

Theorem 4.2. *If $R = \mathbb{F}_{p^k}$ and the protocol* AssertMultVec *passes, then the input commitments have the required relation except with probability* $\frac{n+4\log n+1}{p^k-2}$

The number of rounds of interaction in AssertDotProduct is logarithmic in the number of multiplications n. [BMRS21] also show that, using the Fiat-Shamir transform [FS87], the number of rounds can be made constant by assuming a random oracle.

An alternative version of AssertMultVec with a soundness error that is only logarithmic in n can be achieved as follows:

AssertMultVec$'(\{([x_i], [y_i], [z_i])\}_{i\in[n]})$:

1. $\mathcal{V}$ samples $r_1, \ldots, r_n \in_R R$ and sends them to $\mathcal{P}$.
2. AssertDotProduct$(r_1[x_1], \ldots, r_n[x_n], [y_1], \ldots, [y_n], \sum_{i\in[n]} r_i[z_i])$.

One can easily show that AssertMultVec$'$ has the desired soundness, although at the expense of communicating more random elements from $\mathcal{V}$ to $\mathcal{P}$. In practice, one can optimize this by having $\mathcal{V}$ choose a random PRG seed that it sends to $\mathcal{P}$, with $r_1, \ldots, r_n$ derived deterministically from the seed.

4.3 LPZK Multiplication Check

The Line Point Zero Knowledge (LPZK) work of [DIO21] introduces the concept of an LPZK proof system, where the prover constructs a line and the verifier queries a single point on that line, and determines from this point whether to accept or reject the proof (see Figure 6). This geometric presentation emphasizes the simplicity and algebraic character of the VOLE commitment scheme, which will be used in a non-black box construction.

In the LPZK multiplication check, the witness and all intermediate wire values are stored in the vector $\mathbf{a}$ (see Figure 6). The underlying intuition is that the verifier will perform a series of calculations on the vector $\mathbf{v} := \mathbf{a}\alpha + \mathbf{b}$ and the prover will mirror the verifier by performing the

Mac'n'Cheese Multiplication Check

The Mac'n'Cheese multiplication check AssertMultVec, for n a power of two, reduces to the check of a randomized dot product via AssertDotProduct as follows:

AssertMultVec($\{([x_i], [y_i], [z_i])\}_{i\in[n]}$):

1. $\mathcal{V}$ samples $r \in_R R\backslash\{0\}$ and sends it to $\mathcal{P}$.
2. Run AssertDotProduct$(r^1[x_1], \ldots, r^n[x_n], [y_1], \ldots, [y_n], \sum_{i\in[n]} r^i[z_i])$.

AssertDotProduct($\{([x_i], [y_i])\}_{i\in[n]}, [z]$):

If $n \leq 2$:

1. Check $\{([x_i], [y_i], [z_i])\}_{i\in[n]}$ using the protocol from Figure 4.
2. Run CheckZero$(\sum_{i\in[n]}[z_i] - [z])$.

Otherwise:

1. $\mathcal{P}$ defines polynomials of least degree $f_1, \ldots, f_{n/2}, g_1, \ldots, g_{n/2} \in R[X]$ such that for $j \in \{1, 2\}$: $f_i(j) = x_{(j-1)n/2+i}$, $g_i(j) = y_{(j-1)n/2+i}$.
2. $\mathcal{P}$ defines the polynomial $h = \sum_{i\in[n/2]} f_i g_i \in R[X]$. Note that h has degree ≤ 2. Let c_0, c_1, c_2 denote the coefficients of h.
3. For $i \in \{0, 1, 2\}$: $\mathcal{P}$ runs Input(c_i) of $\mathcal{F}_{\mathsf{ABB}}$ and both parties obtain $[c_i]$.
4. For $i \in [n/2]$: $\mathcal{P}$ and $\mathcal{V}$ define committed polynomials of least degree $[f'_i]$ and $[g'_i]$ satisfying for $j \in [2]$: $f'_i(j) = [x_{(j-1)n/2+i}]$, $g'_i(j) = [y_{(j-1)n/2+i}]$. The committed polynomial can be evaluated in points using Lagrange interpolation.
5. Let $[h']$ be the (committed) polynomial defined by the $[c_i]$ values.
6. $\mathcal{V}$ samples $r \in_R R\backslash\{0, 1\}$ and sends it to $\mathcal{P}$.
7. CheckZero$(\sum_{i\in[2]}[h'](i) - [z])$.
8. AssertDotProduct$([f'_1](r), \ldots, [f'_{n/2}](r), [g'_1](r), \ldots, [g'_{n/2}](r), [h'](r))$.

Figure 5: Protocols for efficient multiplications. See text for necessary notation.

same calculations on the vector of formal polynomial expressions given by $\mathbf{a}t + \mathbf{b}$, treating t as an indeterminate.

The results of these calculations are a collection of values held by the verifier that are the evaluations at α of corresponding polynomials held by the prover. Conditions on the coefficients of these polynomials correspond to conditions on the vectors $\mathbf{a}, \mathbf{b}$, and so can be used to prove that the extended witness satisfies the desired relation. Concretely, the LPZK multiplication check builds a series of quadratic polynomials (one per multiplication gate) whose leading coefficients are zero if and only if the corresponding gates are evaluated correctly. The resulting "polynomial checks" we need to verify these coefficients are zero can be efficiently batched together, saving on communication.

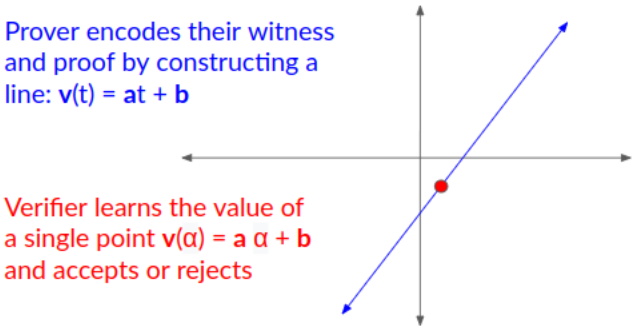

Figure 6: Geometry of Line Point Zero Knowledge

4.3.1 Single Gate Example

To demonstrate how the above language of LPZK translates into proving multiplication relations, we give a commit-and-prove protocol for the relation $R(x, y, z) := xy - z$ as an LPZK over $\mathbb{F}$ with binding and soundness error $\leq 2/|\mathbb{F}|$.

The (honest) prover chooses some triple (x, y, z) and constructs a line $\mathbf{a}t + \mathbf{b}$ by setting

$$\mathbf{a} = (a_1, a_2, a_3, a_4) := (x, y, z, xb_2 + yb_1 - b_3)$$

with b_1, b_2, b_3 chosen uniformly at random and $b_4 := b_1 b_2$. We write

$$\mathbf{v}(t) := \mathbf{a}t + \mathbf{b},$$

for the line held by the prover, and $\mathbf{v} = \mathbf{a}\alpha + \mathbf{b}$ for the point received by the verifier, for a random $\alpha \in \mathbb{F}$. We likewise write the prover's view of the entries as

$$\mathbf{v}(t) = (v_1(t), v_2(t), v_3(t), v_4(t)),$$

and write v_i for $v_i(\alpha)$. The verifier now checks whether

$$v_1 v_2 - \alpha v_3 - v_4 = 0.$$

If the prover is honest, we have

$$\begin{aligned} v_1 v_2 - \alpha v_3 - v_4 = \; & (xy - z)\alpha^2 + (xb_2 + yb_1 - b_3 - (xb_2 + yb_1 - b_3))\alpha \\ & + b_1 b_2 - b_4 \\ = \; & 0 \end{aligned}$$

identically, as long as $xy - z = 0$. In other words, $v_1 v_2 - \alpha v_3 - v_4$ is a quadratic in α that is identically zero if and only if the prover is honest. For a cheating prover, $v_1 v_2 - \alpha v_3 - v_4$ will be equal to some nonzero polynomial in α, and so breaking the binding property would be equivalent to $\mathcal{P}$ guessing α, while breaking soundness would be equivalent to $\mathcal{P}$ constructing a polynomial of degree 2 which has α as a root, which gives binding and soundness error $\leq 2/|\mathbb{F}|$, by the Schwartz-Zippel Lemma,

as desired.Note that this is a special case of the LPZK construction sketched above, since being identically zero is a stronger condition than having a zero leading coefficient.

When constructing LPZK from a random VOLE, this protocol requires communication for each entry of $\mathbf{a}$ and $\mathbf{b}$ which cannot be set randomly by the prover. Here, we require communication of five field elements: four elements for the values $a_1, \ldots, a_4$ and an additional element of communication for the value b_4.

4.3.2 Polynomial Checks

To emphasize the similarity of the one gate example to the IT-MAC that we defined in Section 3.1, we can instead write the triple $(\mathbf{a}, \mathbf{b}, \mathbf{v})$ as $(\mathbf{x}, M[\mathbf{x}], K[\mathbf{x}])$. Then setting values of $\mathbf{a}$ or $\mathbf{x}$ is accomplished by the **Input** step of $\mathcal{F}_{\mathsf{ABB}}$, and setting values of $\mathbf{b}$ or $M[\mathbf{x}]$ is accomplished similarly by sending the difference between a random value $M[r]$ and the desired value $M[x_i]$.

When extending this construction to a larger circuit, we generate an authenticated wire $[w]$ for each input wire and each output wire of a multiplication gate, and get authentications of the remaining wires from affine transformations. There are then two variant LPZK protocols, one with information theoretic security without a random oracle, and one in the random oracle model, which we call IT-LPZK and ROM-LPZK.

In both protocols, the prover constructs some quadratic polynomial in Δ for each multiplication gate, and the verifier learns the evaluation of those polynomials. The Δ^2 coefficient of the polynomial is the value $xy - z$, so if the prover is honest, the polynomial will be degenerate. For IT-LPZK, the polynomial will also have zero Δ coefficient, that is, the polynomial constructed by an honest prover is equal to a constant. For ROM-LPZK, the polynomial is linear.

For the ROM-protocol, the degenerate polynomial held by the prover is

$$K[x]K[y] - K[z]\Delta = (xy - z)\Delta^2 + (y \cdot M[x] + x \cdot M[y] - M[z])\Delta + M[x] \cdot M[y],$$

which will be linear if $xy = z$, with the prover holding the coefficients of the linear polynomial and the verifier holding the evaluation at Δ. For the IT-LPZK protocol, we set $u := xM[y] + yM[x] - M[z]$ and subtract $M[u] := u\Delta + K[u]$ from this polynomial, so that the prover and verifier hold the putative constant

$$K[x]K[y] - K[z]\Delta - K[u] = M[x]M[y] - M[u].$$

We therefore need two gadgets for ROM-LPZK and IT-LPZK that certify that a batch of quadratic polynomials are degenerate and actually of degree 1 or degree 0, respectively.

In the IT-LPZK protocol, we treat multiplication gates in batches of size n, (not necessarily equal to the total number of multiplication gates in the circuit) resulting in a soundness error of $2n/|\mathbb{F}|$. For each batch of n gates $x_i y_i = z_i$, the prover authenticates an additional $u_i := x_i M[y_i] + y_i M[x_i] - M[z_i]$, so that the prover holds $(u_i, M[u_i])$ and the verifier holds $K[u_i] = u_i\Delta + M[u_i]$. The verifier then computes the product of n successive instances of the polynomial above

$$m := \prod_{i=1}^{n} \iota(K[x_i]K[y_i] - K[z_i]\Delta - K[u_i]),$$

where ι is the identity function on nonzero values, with $\iota(0) = 1$ to ensure the product is nonzero. The prover sends the term $\hat{m} := \prod_{i=1}^{n} \iota(M[x_i]M[y_i] - M[u_i])$, and the verifier aborts if $m \neq \hat{m}$.

The value the verifier has computed is now the evaluation of a polynomial of degree $2n$, which is a constant polynomial if and only if the prover acted honestly on each of the n gates in the batch.

Otherwise, a cheating prover must construct a non-constant polynomial that has Δ as a root, and we can apply the Schwartz-Zippel lemma as above to bound the binding and soundness error. The correctness and security are proven in [DIO21].

Theorem 4.3. *The protocol in Figure 7, using Step 3 securely instantiates* CheckMult *in Figure 3 with soundness error* $2n/|\mathbb{F}|$.

LPZK and QuickSilver Multiplication Check

Inputs and parameters: Two parties holds authenticated values $\{([x_i], [y_i], [z_i])\}_{i\in[n]}$.

Protocol:

1. For $i \in [n]$, two parties execute the following:
   (a) Two parties parse the i-th authenticated multiplication tuple: $\mathcal{P}$ has $(M[x], x)$, $(M[y], y)$, $(M[z], z)$ and $\mathcal{V}$ has $K[x], K[y], K[z]$, such that $K[j] = M[j] + j \cdot \Delta$ for $j \in \{x, y, z\}$ and that $z = x \cdot y$.
   (b) $\mathcal{P}$ computes $A_{0,i} := M[x] \cdot M[y] \in \mathbb{F}^t$ and $A_{1,i} := x \cdot M[y] + y \cdot M[x] - M[z] \in \mathbb{F}^t$. $\mathcal{V}$ computes $B_i := K[x] \cdot K[y] - K[z] \cdot \Delta \in \mathbb{F}^t$.
2. **Quicksilver.** $\mathcal{P}$ and $\mathcal{V}$ perform the following check to verify that $B_i = A_{0,i} + A_{1,i} \cdot \Delta$ for all $i \in [n]$ using a fresh VOLE relation $B^* = A_0^* + A_1^*$.
   (a) $\mathcal{V}$ samples $\chi \leftarrow \mathbb{F}^t$ and sends it to $\mathcal{P}$.
   (b) $\mathcal{P}$ computes $U := \sum_{i\in[n]} A_{0,i} \cdot \chi^i + A_0^*$ and $V := \sum_{i\in[n]} A_{1,i} \cdot \chi^i + A_1^*$, and sends (U, V) to $\mathcal{V}$.
   (c) Then $\mathcal{V}$ computes $W := \sum_{i\in[n]} B_i \cdot \chi^i + B^*$ and checks that $W = U + V \cdot \Delta$. If the check fails, $\mathcal{V}$ outputs false and aborts.
3. **IT-LPZK, requires** $t = 1$**.** $\mathcal{P}$ and $\mathcal{V}$ perform the following check to verify that $B_i = A_{0,i} + A_{1,i} \cdot \Delta$ for all $i \in [n]$.
   (a) $\mathcal{P}$ authenticates the value $[u] = [A_{1,i}]$.
   (b) Let $\iota(x) = x$ if $x \neq 0$ and $\iota(0) = 1$ otherwise. Then $\mathcal{P}$ computes $m := \prod_{i=1}^n \iota(M[x]M[y] - M[u])$ and sends m to $\mathcal{V}$.
   (c) $\mathcal{V}$ checks that $m = \prod_{i=1}^n \iota(K[x]K[y] - K[z]\Delta - K[u])$. If the check fails, $\mathcal{V}$ outputs false and aborts.

Figure 7: The LPZK and QuickSilver multiplication check protocols.

4.3.3 QuickSilver Extension

The circuit-based QuickSilver [YSWW21] multiplication check can be viewed as an extension of the ROM version of the Line-Point ZK protocol [DIO21] to support any field. We provide an overview of the protocol. The key idea to support any field size is by extending the checking on

an extension field of the original field. We will abuse the notation and use $\mathbb{F}^t$ to also refer to the extension field; this means that multiplications between two $\mathbb{F}^t$ elements are performed according to field-extension multiplication. For each multiplication gate, the prover $\mathcal{P}$ has $(x, M[x])$, $(y, M[y])$, $(z, M[z]) \in \mathbb{F} \times \mathbb{F}^t$; the verifier $\mathcal{V}$ holds $K[x], K[y], K[z], \Delta \in \mathbb{F}^t$ such that the following four equations hold:

$$z = x \cdot y \quad \text{and} \quad M[i] = K[i] - i \cdot \Delta \ \text{ for } i \in \{x, y, z\}.$$

If $\mathcal{P}$ is malicious, the first equation could potentially be incorrect and the main task is to check that this relationship holds for all multiplication gates. Although the last three equations are linear equations from the perspective of the verifier, the first equation is not linear. The crucial observation from line-point ZK is that it is possible to convert the non-linear checking to a linear checking. Specifically, for the i-th multiplication gate with wire values (x, y, z), if it is computed correctly (i.e., $z = x \cdot y$), then we have:

$$\begin{aligned}
&\overbrace{B_i = K[x] \cdot K[y] - K[z] \cdot \Delta}^{\text{known to } \mathcal{V}} \\
&= (M[x] + x \cdot \Delta) \cdot (M[y] + y \cdot \Delta) - (M[z] + z \cdot \Delta) \cdot \Delta \\
&= M[x] \cdot M[y] + (y \cdot M[x] + x \cdot M[y] - M[z]) \cdot \Delta + (x \cdot y - z) \cdot \Delta^2 \\
&= \underbrace{M[x] \cdot M[y]}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_{0,i}}} + \underbrace{(y \cdot M[x] + x \cdot M[y] - M[z])}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_{1,i}}} \cdot \underbrace{\Delta}_{\substack{\text{known to } \mathcal{V} \\ \text{global key}}}
\end{aligned}$$

We can see that the above relationship is now linear and very similar to the IT-MAC relationship. What's more, if the underlying wire values (i.e., x, y, z) are not computed correctly, then the above relationship can hold only with probability $2/|\mathbb{F}|^t$ due to Schwartz-Zippel lemma: now it becomes a quadratic equation of Δ, where there are at most two values of Δ that satisfy the equation.

Now when we look at a circuit with t multiplication gates, we can obtain one such relationship for each multiplication gate. Namely, for each $i \in [n]$, $\mathcal{P}$ has $A_{0,i}, A_{1,i} \in \mathbb{F}^t$ and $\mathcal{V}$ has $B_i \in \mathbb{F}^t$ such that $B_i = A_{0,i} + A_{1,i} \cdot \Delta$. We can check all t linear relations in a batch using a random linear combination. In particular, the verifier samples a uniform element $\chi \in \mathbb{F}^t$ *after* the above values have been defined, and then checks that the following relationship holds:

$$\underbrace{\sum_{i \in [n]} B_i \cdot \chi^i}_{\substack{\text{known to } \mathcal{V} \\ \text{denoted as } B}} = \underbrace{\sum_{i \in [n]} A_{0,i} \cdot \chi^i}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_0}} + \underbrace{\left(\sum_{i \in [n]} A_{1,i} \cdot \chi^i \right)}_{\substack{\text{known to } \mathcal{P} \\ \text{denoted as } A_1}} \cdot \underbrace{\Delta}_{\substack{\text{known to } \mathcal{V} \\ \text{global key}}}$$

By the verifier sending just one field element (i.e., χ), we are able to reduce checking t equations in the circuit to checking the above single equation, that is $B = A_0 + A_1 \cdot \Delta$, where $\mathcal{V}$ has B and Δ, while $\mathcal{P}$ has A_0 and A_1. This could be easily checked by using a random linear relationship $B^* = A_0^* + A_1^* \cdot \Delta$ with $B^*, A_0^*, A_1^* \in \mathbb{F}^t$ to mask field elements A_0 and A_1, and then opening the masked elements. In particular, $\mathcal{P}$ sends $U = A_0 + A_0^*$ and $V = A_1 + A_1^*$ to $\mathcal{V}$, who checks that $B + B^* = U + V \cdot \Delta$. Finally, this random linear relationship over $\mathbb{F}^t$ can be easily obtained by generating subfield VOLE correlations on $\mathbb{F}_p$ and packing them to $\mathbb{F}^t$.

The details of the protocol can be found in Figure 7 and we have the following theorem.

Theorem 4.4. *For any field $\mathbb{F}$ and integer t, the protocol in Figure 7 using Step 2 securely instantiates* CheckMult *in Figure 3 with statistical error of $(n+3)/|\mathbb{F}|^t$.*

Note that the online phase of the ZK protocol where the circuit and witness are known, can be made non-interactive by computing χ using a random oracle to hash the transcript up to that point when $|\mathbb{F}|^t \geq 2^\kappa$.

4.4 Comparing the Multiplication check protocols

To compare the different Multiplication check protocols presented in this section, we focus on the number of communication rounds as well as elements in R that have to be sent per verified multiplication. Moreover, since Wolverine and Mac'n'Cheese become more efficient as n increases, we assume $n = 1,000,000$.

In the arithmetic case, i.e. when $R = \mathbb{Z}_p$ for a large p, Wolverine has to send 4 R-elements per multiplication, to achieve statistical security 2^{-40} with $B = 3$. The warm-up version of Mac'n'Cheese (Section 4.2) reduces this to 3 elements. The interactive versions of LPZK & QuickSilver only need to communicate 1 R-element per multiplication in 3 rounds of interaction, while the non-interactive and information-theoretic version of LPZK communicates 2 R-elements. In comparison, using the batch multiplication check in Mac'n'Cheese gives an amortized communication cost of 1 R-element per multiplication and requires 17 rounds with $n = 1\,000\,000$. Note that all of these protocols can be collapsed to be non-interactive (excluding the VOLE preprocessing) in the random oracle model using the Fiat-Shamir transform. The main advantage of Mac'n'Cheese over QuickSilver (which has better concrete and asymptotic performance) is that it supports so-called stacking proofs (see Section 5.3) which are not known to carry over to QuickSilver as easily. If $R = \mathbb{Z}_2$, i.e. for binary circuits, then Wolverine for $n = 1,000,000$ has to communicate 7 bits per proven multiplicative relation. In comparison, QuickSilver and Mac'n'Cheese both take approximately 1 bit. LPZK, on the other hand, only supports computations over large fields.

In terms of practical performance, [BMRS21] argue that QuickSilver shows approximately twice the throughput in proven multiplicative relations per time unit in comparison to Mac'n'Cheese. They caution, though, that the systems have not been compared on identical hardware. The benchmarking of [DILO22] shows that the information-theoretic version of LPZK is twice as fast as the interactive version of LPZK in terms of online computational costs, perhaps due to eliminating the cost of invoking a hash function, and only around 2.5× slower than evaluating the circuit in the clear.

5 Extensions

In Section 3 and Section 4, we provided an overview how to efficiently prove any computation provided that it can be written as a circuit over a field with linear and degree-2 multiplication gates only. However, in many settings, representing the statement as a such a circuit may not be ideal: 1) this specific circuit representation may be huge and thus lead to high overhead in the proof as it has to fit into this specific representation; 2) it prevents us from designing customized gadgets and gates that exploit the semantics of the problem and could be potentially more efficient than degree-2 circuit-based protocols.

In this section, we discuss efficient gadgets that are out of the regular degree-2 circuit-modal computation. The modular design of our approach means that these gadgets can be integrated with

the main protocol easily, and can e.g. be expressed as higher-degree gates which the model from Section 2.1 permits.

5.1 Low-Degree Polynomials Proofs

We introduce proofs for low-degree polynomials from [YSWW21]. As a starter, let us first generalize the multiplication check in Section 4.3.3 to prove an inner product between two vectors with communication of 1 field element.

5.1.1 Proving degree-2 polynomials

Let f be a degree-2 polynomial such that $f(x_1, \ldots, x_n) = c_0 + \sum_{i\in[n/2]} c_i \cdot x_i \cdot x_{n/2+i}$. Both parties hold authenticated values $[w_1], \ldots, [w_n]$, and the prover wants to prove $f(w_1, \ldots, w_n) = 0$. Using a circuit-based approach, this would need $n/2$ multiplication gates and thus at least $n/2$ communication. Here, we show a protocol that can use less communication: Observe that

$$
\begin{aligned}
&f(K[w_1], \ldots, K[w_n]) = c_0 + \sum_{i\in[n/2]} c_i \cdot K[w_i] \cdot K[w_{n/2+i}] \\
&= c_0 + \sum_{i\in[n/2]} c_i \cdot (M[w_i] + w_i \cdot \Delta) \cdot (M[w_{n/2+i}] + w_{n/2+i} \cdot \Delta) \\
&= c_0 + \sum_{i\in[n/2]} \big(c_i \cdot M[w_i] \cdot M[w_{n/2+i}] + c_i \cdot (M[w_i] \cdot w_{n/2+i} + M[w_{n/2+i}] \cdot w_i) \cdot \Delta + c_i \cdot w_i \cdot w_{n/2+i} \cdot \Delta^2\big) \\
&= \Big(c_0 + \sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot M[w_{n/2+i}]\Big) + \Big(\sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot w_{n/2+i} + c_i \cdot M[w_{n/2+i}] \cdot w_i\Big) \cdot \Delta + \\
&\quad \Big(\sum_{i\in[n/2]} c_i \cdot w_i \cdot w_{n/2+i}\Big) \cdot \Delta^2 \\
&= \Big(c_0 + \sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot M[w_{n/2+i}]\Big) + \Big(\sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot w_{n/2+i} + c_i \cdot M[w_{n/2+i}] \cdot w_i\Big) \cdot \Delta - c_0 \cdot \Delta^2.
\end{aligned}
$$

The last equation is due to the fact that $f(w_1, \ldots, w_n) = c_0 + \sum_{i\in[n/2]} c_i \cdot w_i \cdot w_{n/2+i} = 0$. Reorganizing the above equation a bit, we can obtain the following:

$$
\begin{aligned}
&\underbrace{f(K[w_1], \ldots, K[w_n]) + c_0 \cdot \Delta^2}_{\text{known to } \mathcal{V}\text{, denoted as } B} \\
&= \underbrace{\Big(c_0 + \sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot M[w_{n/2+i}]\Big)}_{\text{known to } \mathcal{P}\text{, denoted as } A_0} + \underbrace{\Big(\sum_{i\in[n/2]} c_i \cdot M[w_i] \cdot w_{n/2+i} + c_i \cdot M[w_{n/2+i}] \cdot w_i\Big) \cdot \Delta}_{\text{known to } \mathcal{P}\text{, denoted as } A_1}.
\end{aligned}
$$

This is still a linear relationship $B = A_0 + A_1 \cdot \Delta$, which could be proven just as in the Quicksilver protocol. Essentially, we can prove a degree-2 polynomial with $n/2$ multiplications with a communication cost of just $O(1)$, in addition to the cost of committing the witness. This is independent of the number of multiplications in the polynomial, which could be as many as $n/2 = O(n)$. One immediate observation is that if we have t such polynomials to be proven, the total communication cost is still $O(1)$ rather than $O(t)$, by using the same random-linear-combination idea to reduce all linear checks to a single check.

5.1.2 Generalizing to any low-degree polynomial

Now we generalize the above to support checking of low-degree polynomials. We assume that the witness is $(w_1, \ldots, w_n) \in \mathbb{F}^n$; there are t polynomials to be proven and each multivariate polynomial $f_i(X_1, \ldots, X_n)$ over $\mathbb{F}$ has a degree at most d. The prover wants to prove that $f_i(w_1, \ldots, w_n) = 0$ for all $i \in [t]$. Below, we show how to prove such polynomial set in communication of d field elements over $\mathbb{F}^t$,in addition to the n field elements over $\mathbb{F}$ to commit the witness. For every n-variable d-degree polynomial $f \in \{f_1, \ldots, f_t\}$, we will represent it as $f(X_1, \ldots, X_n) = \sum_{h\in[0,d]} g_h(X_1, \ldots, X_n)$, where g_h is a degree-h polynomial such that all terms in g_h have exactly degree h. Here we assume that each polynomial f has been written in a "degree-separated" format, and thus do consider the computation of this decomposition to be beyond scope.

We write each polynomial in a "degree-separated" format and shift each sub-polynomial. The verifier now computes

$$\begin{aligned}
\sum_{h\in[0,d]} g_h(K[w_1], \ldots, K[w_n]) \cdot \Delta^{d-h} &= \sum_{h\in[0,d]} g_h(M[w_1] + w_1 \cdot \Delta, \ldots, M[w_n] + w_n \cdot \Delta) \cdot \Delta^{d-h} \\
&= \sum_{h\in[0,d]} \left(g_h(w_1, \ldots, w_n) \cdot \Delta^d + \sum_{j\in[0,h-1]} A_h^j \cdot \Delta^{j+d-h} \right) \\
&= \sum_{h\in[0,d]} g_h(w_1, \ldots, w_n) \cdot \Delta^d + \sum_{h\in[0,d-1]} A_h \cdot \Delta^h \\
&= f(w_1, \ldots, w_n) \cdot \Delta^d + \sum_{h\in[0,d-1]} A_h \cdot \Delta^h \\
&= \sum_{h\in[0,d-1]} A_h \cdot \Delta^h.
\end{aligned}$$

Here A_h^j is defined as above, and A_h is the aggregated coefficient for all terms with Δ^h. Note that the prover with witnesses w_i and MACs $M[w_i]$ can compute all the coefficients locally. The coefficients A_h are polynomial coefficients when we treat it as a single-variable polynomial on Δ. Therefore, the prover can compute all A_h efficiently by evaluating the polynomial on $d+1$ points and then computing the polynomial coefficients using Lagrange interpolation. In many practical applications, the polynomial is usually simple and thus the coefficients can be derived without using the above generic approach. This relationship can be viewed as an oblivious polynomial evaluation (OPE), where the verifier has Δ and the prover has a polynomial $P(x) = \sum_{h\in[0,d-1]} A_h \cdot x^h$ over $\mathbb{F}^t$. The verifier wants to check that the resulting evaluation in the above equation is the same as $P(\Delta)$. It is not hard to check the above polynomial relation, as sVOLE can be used to generate (V)OPE in an efficient way. Similarly, we can perform the checks for all t polynomials in a batch using a random linear combination. This results in a total communication of $(n + dr) \log |\mathbb{F}|$ bits in the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model. When using the interpolation approach to compute the coefficients A_h, we have that the computational cost of the prover and verifier is $O(td^2z + dn)$ and $O(tdz)$ respectively, where z is the maximum number of terms in all t polynomials.

5.2 LPZKv2

The follow-up work to LPZK, [DILO22], presents another extension of of VOLE-based ZK that uses an extension of the VOLE correlation and is specialized for particular circuit formats. LPZKv2 im-

proves the online communication cost of LPZKv1 by a factor of roughly two for both the information-theoretic and random oracle variants (IT-LPZKv2 and ROM-LPZKv2, respectively).

There are two technical ideas that enable the improvements of LPZKv2 over LPZKv1. The first technical idea is to store the message in the constant term of the VOLE, instead of the linear term, i.e. to write $K^*[x] := \Delta M^*[x] + x$ instead of $K[x] := \Delta x + M[x]$. Storing the value in the constant term instead of the linear term reduces the verifier's computation, since on the step **Input** of $\mathcal{F}_{\mathsf{ABB}}$ described in 3.2, when the $\mathcal{P}$ sends $\delta = x - r$ to $\mathcal{V}$, $\mathcal{V}$ now computes $K^*[x] \leftarrow K^*[r] - \delta$ instead of $K[x] \leftarrow K[r] - \Delta\delta$. Because each multiplication gate requires an additional call to **Input**, this change reduces the verifier's computation by one multiplication per gate in both the information theoretic and random oracle variants.

The second technical idea is the use of an extension of VOLE, quadratically certified VOLE, or qVOLE, which allows for the imposition of additional quadratic relations on the entries of an instance of random VOLE. These quadratic relations essentially allow certain terms needed in LPZK to be precomputed, reducing the communication and computation required in the online step. For example, using qVOLE, we could generate three authenticated random values $([p], [q], [r])$ with the guarantee that the quadratic relation $M^*[p] \cdot M^*[q] - M^*[r] = 0$ is satisfied, which in turn would imply that $K^*[p] \cdot K^*[q] - K^*[r]$ is a polynomial with zero Δ^2 coefficient. This property still holds after calling the modified **Input** protocol described above to shift $(p, q, r) \to (x, y, r)$, a fact which we use extensively in the construction below.

The qVOLE functionality can be realized either by bootstrapping off of an existing instance of VOLE, which requires a linear amount of communication in the preprocessing phase (effectively pushing 50% of the communication of LPZKv1 to an offline phase). Alternatively, we can use ring-LPN to give a sublinear-communication qVOLE generation protocol that is concretely efficient in the SIMD setting or for circuits with repeated subcircuits (such as hash trees).

5.2.1 General circuits

The information-theoretic protocol IT-LPZKv2 can be used efficiently with general circuits by realizing the qVOLE functionality by bootstrapping off of an existing instance of VOLE. Then IT-LPZKv2 requires $1 + \frac{1}{n}$ elements of communication in the online phase, nearly matching the communication cost of ROM-LPZKv1, Quicksilver or Mac'n'Cheese over large fields. Here n is a constant representing batch size, as in IT-LPZKv1. For a multiplication gate $xy = z$, we have

$$K^*[x] \cdot K^*[y] = M^*[x]M^*[y]\Delta^2 + (xM^*[y] + yM^*[x])\Delta + xy,$$

where the leading coefficient can be pre-computed from the qVOLE functionality, and the constant coefficient $xy = z$ is the constant coefficient of the authenticated output value $K^*[z]$. If $K^*[r] = M^*[x]M^*[y]\Delta + M^*[r]$ is the precomputed qVOLE entry, then $K^*[x]K^*[y] - K^*[r]\Delta - K^*[z]$ will be a linear polynomial in Δ with zero constant term, which can be checked using the same batched proof given in Step 3 of Figure 7. Therefore the total online communication cost is 1 element for $K^*[z]$ and an amortized cost of $1/n$ per gate for the batched proofs. Additionally, 1 element of communication per gate is required in a preprocessing step for the generation of the values $K^*[r]$.

5.2.2 Layered circuits and other specialized circuits

As mentioned above, both IT-LPZKv2 and ROM-LPZKv2 give efficiency gains in the SIMD setting or for circuits with repeated subcircuits, since then qVOLE functionality can be efficiently realized

using Ring-LPN. Additionally, the polynomial techniques described above allow us to extend the LPZKv2 constructions from arithmetic circuits containing only fan-in 2 addition and multiplication gates to circuits with arbitrary degree 2 polynomial gates.

The ROM-LPZKv2 protocol offers some additional speed-up in online communication time for a broad class of circuits with a certain colorability property described below. The key observation behind the ROM-LPZKv2 protocol change is that the expression $K^*[x]K^*[y] - K^*[p]\Delta$ is a linear polynomial with constant term xy, and so already represents an authentication of xy without any communication required at all. However, for the authenticated value $K^*[z] := K^*[x]K^*[y] - K^*[p]\Delta$, the linear term is equal to $M^*[z] := xM^*[y] + yM^*[x] - M^*[p]$, which depends on the prover's input. Therefore if we wish to use z as the input wire to another multiplication gate with inputs z, t the prover can no longer compute the quadratic coefficient $M^*[z]M^*[t]$ using only precomputed randomness. This is the motivation behind the colorability property.

For the coloring, we use the color red to denote wires for which the value $M^*[t]$ is determined purely by the correlated randomness, and use blue to denote wires for which the value $M^*[t]$ depends on the prover's input. Color the input wires red, then color the remaining wires of the circuit such that, for any degree 2 polynomial gate with all blue inputs, or a mix of blue and red inputs, the output wire is red, while for a gate with all red inputs, the output wire may be red or blue. Then the communication cost of ROM-LPZKv2 under this coloring is equal to the number of red wires.

For *layered circuits*, where each gate is assigned to some layer k, and all the inputs to gates at layer k are outputs to gates at layer $k-1$, either all odd layers or all even layers can be colored red, so the amortized communication per degree 2 polynomial gate is at most $\frac{1}{2}$ elements of communication per gate in a layered circuit. For a broad class of non-layered circuits, substantial savings are also possible. For example, as described in [DILO22], for a random circuit made up entirely of multiplication gates and colored with a greedy algorithm, approximately 38% of the wires will be blue, so that one can achieve an approximately 38% reduction in communication.

5.3 Disjunctions and r-out-of-n proofs

[BMRS21] consider the setting where both $\mathcal{P}$ and $\mathcal{V}$ agree on m circuits $C_1, \ldots, C_m$ that define protocols based on a committed vector $[\boldsymbol{w}]$. Let these protocols each be public coin HVZK proofs over the same field $\mathbb{F}_{p^k}$. For this, they construct a communication-efficient protocol showing that from $[\boldsymbol{w}]$ one can extract a satisfying input $\boldsymbol{w}_{i*}$ to at least one of the circuits.

The classic OR-proof technique by Cramer et al. [CDS94] can be used to construct such a proof with message complexity $\approx \sum_{i\in[m]} \alpha_i$ where α_i is the communication necessary for the proof Π_i of C_i. This would be done by running all m proofs for all circuits in parallel (which means sending messages for evaluating all of them) and having their outputs being committed as $[y_1], \ldots, [y_m]$. The prover would then show that at least one finished successfully with the expected output using the OR-proof of [CDS94] on $[y_1], \ldots, [y_m]$. [BMRS21] show how to reduce the message complexity of such a proof to $2mk + \max\{\alpha_i\}$, where the soundness error grows by an additional additive $\approx p^{-k}$. What is required for the technique of [BMRS21] to work is that all messages from the prover in each protocol Π_i appear uniformly random. Moreover, they require that protocol messages in each round are of identical length, for any Π_i, Π_j. Towards this, observe that it is always possible to defer zero-tests in a protocol Π_i that relies on $\mathcal{F}_{\mathsf{ABB}}$ to the end, as a verifier doesn't have a secret that could be leaked through late application of the zero check. Achieving messages of identical length (and same number of rounds in each Π_i) can be achieved using padding.

At the same time, not all implementations of $\mathcal{F}_{\mathsf{ABB}}$ are compatible with the requirement that

messages from the prover are uniform, even with padding: The multiplication checks from Section 4.1 and 4.2 reduce to the prover making auxiliary commitments, the verifier sending random challenges and the prover then doing a zero-test. This can be made compatible with the desired protocol structure. Unfortunately, this is not true for the approach from Section 4.3.3 as it works directly on a MAC level.

Assume that we start m proofs Π_i, over $\mathbb{F}_{p^k}$, proving the individual circuits C_i. Note that p can be any prime power, with no restrictions on size. We make the simplifying assumption that each C_i has the same number of linear gates and that each Π_i has the same number of rounds of interaction and that prover messages in each round are of the same length. [BMRS21] show the more general case where these restrictions are not necessary.

Constructing the protocol We construct a protocol $\Pi_{\mathtt{OR}}$, defined over $\mathbb{F}_{p^k}$, for the aforementioned task as follows:

1. $\mathcal{P}$, having only w_{i^*} for one of the circuits C_{i^*}, will commit to $[\boldsymbol{w}]$ such that Π_{i^*} can access w_{i^*}. It then in its head runs each of the Π_i on inputs derived from $[\boldsymbol{w}]$. For this, it extends w_{i^*} with a random padding if necessary.

2. $\mathcal{P}$ and $\mathcal{V}$ will simultaneously run all $\Pi_1, \ldots, \Pi_m$, with the following modification: $\mathcal{P}$'s message c_h to $\mathcal{V}$ in round h will be chosen as the message created from running Π_{i^*}, while $\mathcal{V}$ uses the same message from the prover in all instances. Since the messages of all protocols by assumption appear uniformly random and are therefore indistinguishable, $\mathcal{V}$ can now execute all instances in parallel but cannot tell which of these is the true one.

3. Conversely, since all Π_i are public coin, $\mathcal{V}$ sends a randomness string that is long enough for any of the m instances in round h. $\mathcal{P}$ uses this identical randomness string in all simulated proofs $\Pi_1, \ldots, \Pi_m$.

The challenge now, is that $\mathcal{V}$ cannot simply perform the verification for all Π_i using the $\mathsf{CheckZero}$ queries for each instance, since this would reveal the index i^* of the true statement. Towards resolving this, we first observe that any "pure" zero-test can be turned into a Σ-protocol-like argument as follows:

1. For each $i \in \{1, \ldots, m\}$, let $[\mu_i]$ be the output of the circuit C_i run on $[\boldsymbol{w}]$. Assume that there exists a uniformly random commitment $[r_i]$, which can be generated without additional interaction from $\mathcal{F}_{\mathsf{VOLE}}$.

2. $\mathcal{P}$ sends r_i to $\mathcal{V}$, but crucially does not open the commitment $[r_i]$ yet.

3. $\mathcal{V}$ sends a challenge f_i to $\mathcal{P}$.

4. $\mathcal{P}$ uses $\mathsf{CheckZero}$ to show that $[r_i] + f_i[\mu_i] - r_i$ opens to 0.

This check, crucially, has a soundness error of $1/p^k$, as any prover knowing f_i in advance can generate r_i appropriately in order to cheat during the sigma-protocol.

[BMRS21] now perform a [CDS94]-style OR-proof to show that at least one of the outputs $[\mu_i]$ is zero, using the Σ-protocol version of $\mathsf{CheckZero}$. The basic idea behind [CDS94] is that given m Σ-protocols for proving relations, an OR proof can be done by having the prover choose the random challenge f_i for $m-1$ of the instances, so it can simulate the correct messages r_i to be sent in

every false instance using the simulator for the sigma protocol. This makes the verifier accept for the "false" instances automatically. For the correct instance i^*, $\mathcal{P}$ will choose r_i as in the correct Σ-protocol, which it can complete because the statement is actually true for one instance. Hence, after receiving the m initial messages of each Σ-protocol (honest in one case, simulated in all others), the verifier picks a challenge f, which defines the challenge $f_{i^*} = f - \sum_{i \neq i^*} f_i$ corresponding to the true instance i^*. The prover sends all these f_i to the verifier.

Finally, $\mathcal{V}$ checks that the f_i add up to f and that each $\mathsf{CheckZero}$ test for each C_i is indeed valid.

Threshold proofs In [CDS94] the authors describe how to additionally construct proofs of partial knowledge for any threshold, i.e., how to show that r out of the m statements are true. Their technique, together with a modification of Π_{OR}, can be used to construct a proof in the VOLE setting where we implicitly only communicate the transcript of r statements, and not all m of them.

Towards this, Π_{OR} can then be seen as a special case where $r = 1$. To generalize to arbitrary r one now simulates the $m - r$ possibly false proofs using false challenges. The prover then, based on the challenge f, computes the unique degree-$m - t$ polynomial s that evaluates to f at point 0 and to the simulated challenge f_i for each i where the sigma-protocol was simulated. It then derives the honest challenges by evaluating this polynomial at their indices, and sends s to the verifier. Towards compressing the messages, [BMRS21] then consider the r messages for the true branches as evaluations of a polynomial t of degree $r - 1$. Namely, for each true evaluated branch i, they let $t(i)$ be the message sent by Π_i. The prover then computes this unique polynomial in canonical coefficient form and sends it to $\mathcal{V}$, who derives the inputs to each simulated Π_i from t. Since both s, t are of canonical form, they do not leak which of the branches are actually true.

log**-overhead disjunctions** The drafted protocol Π_{OR} has the drawback that to verify one out of m statements, we still need $O(m)$ communication complexity in the OR-proof. One can construct an alternative protocol that obtains an overhead only logarithmic in m, as follows:

1. Any Π_i accepts iff $\mathsf{CheckZero}$ is true, i.e. the output commitment $[\mu_i]$ is 0.
2. If the prover can then compute the product $\mu_1 \cdots \mu_m$, and prove that this is 0, then at least one μ_j was 0 to begin with, i.e. one output was true.

A naive instantiation of the above approach is to perform $m - 1$ multiplications between the m implicit variables μ_i, and open the result. However, this would still give $O(m)$ overhead. Instead, one can carefully apply recursion to make this overhead logarithmic, using the fact that after augmenting the parallel evaluation of two protocols Π_1, Π_2 with a multiplication, we obtain a protocol which can recursively be fed into the same process.

5.4 Conversions between $\mathbb{F}_2$ and $\mathbb{F}_p$

In [BBMH+21] the authors show how, given two simultaneously running instances of $\mathcal{F}_{\mathsf{ABB}}$ for moduli $2, p$, one can efficiently prove that $[c_0]_2, \ldots, [c_m]_2$ is the correct binary decomposition of $[c]_p$. These correct decompositions, called Edabits, were introduced to secure computation protocols in [EGK+20] and are useful for computing/proving truncations and comparisons.

On a high level, in [EGK+20] first a set of random Edabits is created during a preprocessing phase. Later, one of these Edabits is used to perform the actual conversion in the online phase.

[BBMH+21] adapt and optimize the approach of [EGK+20] in multiple ways, mainly by observing that, since the prover already knows the conversions ahead of time, these can directly be checked using the preprocessing protocol and there is no need for the intermediary random Edabits. We want to stress that, concurrently to [BBMH+21], Weng et al. [WYX+21] also introduced an adaptation of Edabits to the ZK setting with a similar construction. In the following, we use the notation of [BBMH+21].

The protocol To define the check let us first, in addition to Edabits, define Dabits as pairs of commitments $[x]_2, [x]_p$ that are consistent. On a high level, the Edabits checking protocol of [BBMH+21] consists of four phases, using a bucketing approach similar to the one already introduced in Section 4.1:

1. Initially, $\mathcal{P}$ commits to auxiliary random Edabits and Dabits necessary for the check. The Dabits are verified separately for consistency, and then $\mathcal{V}$ chooses a random permutation.

2. After permuting the auxiliary Edabits, both parties run an implicit cut-and-choose phase. Here, $\mathcal{P}$ opens C of the auxiliary Edabits, which are checked by $\mathcal{V}$ for consistency.

3. Place each input Edabit (that we want to test for correctness) into one of N buckets, each of which also contains a set of B auxiliary Edabits $\{([r_0]_2, \ldots, [r_{m-1}]_2, [r]_p)\}_{i=0}^{B-1}$ that we use to perform verification. None of these auxiliary Edabits have been proven consistent, but C Edabits coming from the same pool have been opened in the previous step.

4. Now, over B iterations the prover and verifier for each $j \in [B]$ compute $[c + r_j]_p = [c]_p + [r_j]_p$ and use an addition circuit to check that $([e_0]_2, \ldots, [e_m]_2) = ([c_0]_2, \ldots, [c_{m-1}]_2) + ([r_0]_2, \ldots, [r_{m-1}]_2)$. The addition circuit is evaluated using the $\mathcal{F}_{\mathsf{ABB}}$ operations.

For the checks within each bucket, [BBMH+21] use an additional protocol which converts an authentication of a bit $[b]_2$ into an arithmetic authentication $[b]_p$ which is necessary to do so $[c+r_j]_p$ does not reveal any information. Additionally, the authors also observe that their protocol is still secure if the Dabits are "approximately correct", meaning that each pair $[x]_2, [x]_p$ has the same parity and x is bounded but $[x]_p$ necessarily commits to a bit. For this, they present a cheaper protocol to check this property.

5.4.1 Truncation and Comparison

[BBMH+21] provide protocols for verifying integer truncation and comparison. Here, truncation means that given integers l, m and two authenticated values x, x' of l and $l - m$ bits, verify that x' corresponds to the upper $l-m$ bits of x, i.e. $x' = \lfloor \frac{x}{2^m} \rfloor$ over the integers. Integer comparison is then the problem of taking two authenticated integers and outputting 0 or 1 (authenticated) depending on which input is the largest. Integers are considered as signed in the interval $[-2^{l-1}, 2^{l-1})$.

Truncation To perform a truncation check with one call to the conversion check introduced above, the prover in addition to each input $[a]_p$ also provides:

- the truncated value $[a_{tr}]_p$ of $[a]_p$ and its bit decomposition $([a_{tr}^0]_2, \ldots, [a_{tr}^{l-m-1}]_2)$
- the lower m bits of $[a]_p$; $[a']_p = [a \mod 2^m]_p$ as well as its bit decomposition $([a'_0]_2, \ldots, [a'_{m-1}]_2)$.

Having access to $[a_{tr}]_p$ and $[a']_p$ which are shown to have consistend decompositions then allows the verifier then to check that $a = 2^m \cdot a_{tr} + a'$, which proves the claim.

Comparison The authors also present a protocol to compare two signed, l-bit integers α and β. The way their protocol works is by having the prover (and verifier) compute $[\alpha]_p - [\beta]_p$ and have the prover compute the truncation of this which is only the most significant bit. Now one can run the aforementioned truncation verification protocol and use the truncation as the output of the comparison. Similarly to previous works in the MPC setting [Cd10, EGK+20], this gives the correct result as long as $\alpha, \beta \in [-2^{l-2}, 2^{l-2})$, so that $\alpha - \beta \in [-2^{l-1}, 2^{l-1})$, so this introduces a mild restriction on the range of values that can be supported.

5.5 Zero-Knowledge modulo 2^k

The ABB implementation outlined in Section 3 and its multiplication protocols in Section 4 are all tailored to fields. This requires that the computational problem that is encoded in the circuit C can be efficiently expressed over a field, which might not always be the case. For example, computer algorithms are generally expressed as operating on data items from $\mathbb{Z}_{2^{32}}$ or $\mathbb{Z}_{2^{64}}$. It is therefore interesting to develop an implementation of $\mathcal{F}_{\mathsf{ABB}}$ for this case. Here, we summarize how this was approached in [BBMH+21, BBMHS22].

Implementing Zero Checks modulo 2^k Assume that $\mathcal{P}$ wants to commit to secrets $x \in \mathbb{Z}_{2^k}$ using $\mathcal{F}_{\mathsf{VOLE}}$, such that also $\Delta, K[x], M[x]$ come from $\mathbb{Z}_{2^k}$. Assuming one would follow the approach from Section 3, then $\mathcal{P}$ sends $x, M[x]$ to open $[x]$, such that $\mathcal{V}$ checks that $K[x] + \Delta x = M[x] \bmod 2^k$.

Unfortunately, since $\mathbb{Z}_{2^k}$ is not a ring, such a check is not constraining $\mathcal{P}$ to only one valid opening $x, M[x]$. For example, if $\Delta = 2\Delta'$ which happens with probability $1/2$, then by setting $x' = x + 2^{k-1}$ we have that

$$x' \cdot \Delta = (x + 2^{k-1})(2\Delta') = x\Delta + 2^k = x\Delta \bmod 2^k,$$

meaning that $x', M[x]$ is also a valid opening.

Instead, [BBMH+21, BBMHS22] choose the information-theoretic MAC scheme from [CDE+18] as their starting point. For this, let s be an additional parameter, and $\ell = k + s$. To authenticate a value $x \in \mathbb{Z}_{2^k}$ known to $\mathcal{P}$ towards $\mathcal{V}$ (denoted as $[x]$), we choose the MAC keys $\Delta \in \mathbb{Z}_{2^s}$ and $K[x] \in \mathbb{Z}_{2^\ell}$, and compute the MAC tag as

$$M[x] := \Delta \cdot \tilde{x} + K[x] \in \mathbb{Z}_{2^\ell} \tag{1}$$

where $x = \tilde{x} \bmod 2^k$, i.e. $\tilde{x}$ is a representative of the corresponding congruence class of integers modulo 2^k. Then $\mathcal{P}$ gets $\tilde{x}$ and $M[x]$, whereas $\mathcal{V}$ receives Δ and $K[x]$. As before, linear operations can be applied in the same way. One can show that this again leads to $\mathcal{P}$ not being able to cheat, unless it can guess Δ which only succeeds with probability 2^{-s}. Setting $s = \sigma$ then achieves the required statistical security.

Observe that initially $\tilde{x}$ may be chosen as $\tilde{x} = x \in \{0, \ldots, 2^k - 1\}$. Applying arithmetic operations can result in larger values though, which do not get reduced modulo 2^k because all computation happens modulo 2^ℓ. Therefore, to safely open $[x]$ or show that $x = 0$, first the upper s bits of $\tilde{x}$ need to be randomized, by computing $[z] \leftarrow [x] + 2^k \cdot [r]$ with random $\tilde{r} \in_R \mathbb{Z}_{2^\ell}$ and then opening $[z]$.

Extending Line-Point Multiplication checks to $\mathbb{Z}_{2^k}$ It is clear that Equation 4.3.3 from the efficient multiplication check from Section 4.3.3 still holds. Therefore, one might hope that the same security argument still applies. Unfortunately, when just considering general quadratic equations

$$f(x) = ax^2 + bx + c$$

modulo 2^ℓ, then these may have many more than just 2 solutions.

In [BBMHS22] the authors observe that $f(x)$ actually is constrained in the given setting, since a cannot be chosen from the entire ring by $\mathcal{P}$, as $a \neq 0 \bmod 2^k$ for $\mathcal{P}$ to successfully cheat. Moreover, $\mathcal{P}$ may have to guess a root Δ of $f(x)$ that comes from $\{0, \ldots, 2^s - 1\}$. They therefore show the following:

Lemma 5.1. *Let $f(x) \in \mathbb{Z}[x]$ be a quadratic equation such that 2^r is the largest power of 2 dividing all coefficients. Then for any $\ell, s, s' \in \mathbb{N}$ such that $\ell - r > s'$ there are at most $2^{\max\{(2s-s')/2,1\}}$ solutions to $f(x) = 0 \mod 2^\ell$ in $\{0, \ldots, 2^s - 1\}$.*

This implies that by choosing $\ell = k + 2s$, one can achieve the required bound on the number of solutions.

Amortizing Multiplication checks in $\mathbb{Z}_{2^k}$ Also with respect to checking many multiplications simultaneously, the solution of QuickSilver from Equation 4.3.3 does not directly carry over. [BBMHS22] solve this by generalizing the techniques from [CDE+18] to their MAC scheme.

Here, [BBMHS22] show that by letting $\mathcal{V}$ choose the χ_i independently from the set $\{0, \ldots, 2^s - 1\}$ and requiring $s = \sigma + \log(\sigma) + 3$ and $\sigma \geq 7$, then this modified check from Equation 4.3.3 when done modulo 2^ℓ is sound except with probability $2^{-\sigma}$.

5.6 RAM-based Zero-knowledge Proofs

RAM-based zero-knowledge proofs have been studied in the context of zkSNARKs (starting from [BCG+13]). As we show below, the main components needed are zero-knowledge proofs of permutation, packing, and integer comparison. When representing the integers as a list of authenticated values in $\mathbb{F}_2$, proving integer comparison is a fair easy task since comparison becomes a straightforward circuit. For efficient permutation and packing, the problems are somewhat related. The folklore method to prove a permutation is by letting the prover commit to the permuted values and prove equality of the sets, which in turn can be converted to a polynomial identity check [Nef01]. When the field is large, it can be checked in linear time by checking the evaluation on a random point, where the soundness if ensured by the Schwartz-Zippel Lemma. However, converting between $\mathbb{F}_2$ and a large field can be costly even after optimizations such as in Section 5.4.

The key observation in [FKL+21] is that the Schwartz-Zippel Lemma works for any field as long as it is sufficiently large and thus one can choose a field that is friendly to conversions. In particular, we can embed a bit string either as $\mathbb{F}_2^\kappa$ or as $\mathbb{F}_{2^\kappa}$. Their authentications are different: authenticating elements in $\mathbb{F}_2^\kappa$ requires IT-MACs on each $\mathbb{F}_2$ value; authenticating $\mathbb{F}_{2^\kappa}$ only requires one IT-MAC for the whole element. Just like cleartext operation, conversions between *authenticated* elements in $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ can also be done efficiently with no communication: if we fix a degree-κ irreducible polynomial $f(X)$ and identify $\mathbb{F}_{2^\kappa}$ with $\mathbb{F}_2[X]/(f(X))$, then it can be derived that $x = \sum_{i \in [\kappa]} x_i \cdot \mathsf{X}^i$, where $\mathsf{X} \in \mathbb{F}_{2^\kappa}$ denotes the element corresponding to $X \in \mathbb{F}_2[X]/(f(X))$. The

parties can compute $[x]$ by having the prover compute $M[x] = \sum_{i\in[\kappa]} M[x_i]\cdot \mathsf{X}^i$ and the verifier compute $K[x] = \sum_{i\in[\kappa]} K[x_i]\cdot \mathsf{X}^i$; we then have

$$\begin{aligned} M[x] &= \sum_{i\in[\kappa]} M[x_i]\cdot \mathsf{X}^i \\ &= \sum_{i\in[\kappa]} (K[x_i] \oplus x_i\Delta)\cdot \mathsf{X}^i \\ &= \sum_{i\in[\kappa]} K[x_i]\cdot \mathsf{X}^i \;\oplus\; \Big(\sum_{i\in[\kappa]} x_i\cdot \mathsf{X}^i\Big)\cdot\Delta \\ &= K[x] \oplus x\cdot\Delta. \end{aligned}$$

This means that given a vector of bits, we can prove a Boolean predicate by treating them as authenticated bits and, without communication, we can prove arithmetic predicates on vectos of bits by treating vectors as extension field elements. Below we use ZK proof of read-only RAM as an example.

Read-only ZKRAM. Consider the case where the prover wants to prove in zero knowledge to the verifier that there exists an index i for which $\mathsf{Mem}_i = t$ (where t is a public value). The protocol in this case roughly proceeds as follows:

1. The prover commits to the list of values $\mathcal{L} = \big((0,\mathsf{Mem}_0),(1,\mathsf{Mem}_1),\ldots,(N-1,\mathsf{Mem}_{N-1})\big)$. (This can be done once-and-for-all, and before t is known.)

2. The prover commits to (i,t), where t is known to the verifier but i is not, and appends (i,t) to $\mathcal{L}$.

3. The prover then sorts the tuples in $\mathcal{L}$ by their first entry, giving an updated list $\mathcal{L}'$, and commits to the tuples in that list.

4. The prover proves that $\mathcal{L}'$ is consistent, namely, that if two tuples in $\mathcal{L}'$ agree in their first entry, then they also agree in their second entry. This can be done in a natural way by comparing all adjacent entries in $\mathcal{L}'$. All operations should be proven as a Boolean circuit over $\mathbb{F}_2$.

5. The prover proves that $\mathcal{L}'$ is a permuted version of $\mathcal{L}$ using the polynomial equality check ensured by Schwartz-Zippel. Specifically, let $\mathcal{L} = \big(x_0,\ldots\big)$ denote the tuples in $\mathcal{L}$ (where now we represent each tuple as elements in $\mathbb{F}_{2^\kappa}$), and let $\mathcal{L}' = \big(x'_0,\ldots\big)$ denote the tuples in $\mathcal{L}'$. If one defines the polynomials $L(R) = \prod_i (x_i - R)$ and $L'(R) = \prod_i (x'_i - R)$, then note that $\mathcal{L}$ and $\mathcal{L}'$ are permutations of each other iff $L(R) = L'(R)$. The verifier can efficiently test the equality of these polynomials by choosing a uniform field element r and verifying that $L(r) = L(r')$.

To extend the above idea to support write, one just needs to take the timestamp into consideration since a value can be updated in the middle of the execution. The sorting needs to respect both index and time stamp, and the consistency checks need to incorporate the update after the initialization.

6 Open Questions

We will now mention some research directions which we believe are interesting to develop VOLE-based ZK protocols further.

6.1 Theoretical questions

There are many intriguing theoretical aspects of VOLE-based Zero Knowledge protocols that are yet unexplored. In the following, we provide a list of interesting theoretical research directions.

1. **Using other correlations.** The efficiency of VOLE-based Zero Knowledge protocols relies on the recent breakthroughs in efficient VOLE extension. But also other pseudorandom correlations (see e.g. [BCG+20]) can be efficiently generated. The question is if these, or other, correlations can benefit ZK proofs in a similar way. One step into this direction was already done in [DILO22] (see Section 5.2) by using qVOLE.

2. **Sublinear communication and verification.** The idea of ZK proofs with sublinear communication and verification has flourished in the past 10 years, starting from [BCI+13, GGPR13, PHGR13] and leading to recent Blockchain-optimized proof systems. In VOLE-based ZK, similar properties may be interesting although non-interactivity is less important. Having a verifier with sublinear (in $|C|$) computation while keeping a concretely efficient, linear prover has to the best of our knowledge not yet been achieved. For the special case of disjunctions one could achieve an asymptotic solution by extending the techniques of [HHK+22], but concrete efficiency is unclear.

3. **Sublinear communication, also in the input length.**A recent work [WYY+22], which extends VOLE-based ZK, enables communication sublinear to $|C|$. Their protocol relies on vector oblivious polynomial evaluation (VOPE), an extension of VOLE. At the same time, it requires the proof to be of size at least $|w|$ (i.e. the circuit input) for knowledge extraction. It would be interesting to explore if VOLE-based protocols can be both sublinear in $|C|$ and $|w|$ by introducing a knowledge assumption or using a random oracle, while keeping the concrete efficiency of [WYY+22].

6.2 Practical questions

VOLE-based ZK enjoys many advantages that are intriguing to practical deployment. Here we summarize a list of interesting practical future directions.

1. **More efficient VOLE.** The concrete efficiency of the whole proof system relies on efficient VOLE protocols. Thus it is crucial to design more efficient VOLE protocols with improved computation and communication overhead.

2. **More applications.** State-of-the-art VOLE-based ZK can prove tens of millions of gates with ease. This opens the possibility of proving very large statements, something not feasible before this line of work. It would be interesting to explore the space of applications that need ZK but were limited so far by the scalability problems of existing schemes. The fact that this type of proof is designated-verifier could affect the design and thus poses new questions in designing ZK-based applications.

3. **Lowering the gap between $\mathbb{Z}_{2^k}$ and $\mathbb{F}_p$.** In their work, [BBMHS22] provide a software implementation of their instantiation of $\mathcal{F}_{\mathsf{ABB}}$ that operates over $\mathbb{Z}_{2^k}$. Their implementation achieves $\gtrapprox$ 1mio multiplications over $\mathbb{Z}_{2^{64}}$, which is up to an order of magnitude slower than what the implementations of the protocols from Section 4 can perform in similar settings over fields of similar size. Lowering this gap may be helpful if $\mathbb{Z}_{2^k}$ proof systems should be used for proofs of programs on current hardware architectures.

Acknowledgement

This work was funded by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085 and HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA). Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Competing Interests

The authors declare no competing financial interests.

References

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, CRYPTO 2019, Part III, volume 11694 of LNCS, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[BBMH+21] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, Benoît Razet, and Peter Scholl. Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z2k. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 192–211, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[BBMHS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In Yevgeniy Dodis and Thomas Shrimpton, editors, CRYPTO 2022, Part IV, volume 13510 of LNCS, pages 329–358, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran

Canetti and Juan A. Garay, editors, CRYPTO 2013, Part II, volume 8043 of LNCS, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, ACM CCS 2019, pages 291–308, London, UK, November 11–15, 2019. ACM Press.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, CRYPTO 2019, Part III, volume 11694 of LNCS, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, CRYPTO 2020, Part II, volume 12171 of LNCS, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, ACM CCS 2018, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, TCC 2013, volume 7785 of LNCS, pages 315–333, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, EUROCRYPT 2011, volume 6632 of LNCS, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[Bea92] Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, CRYPTO'91, volume 576 of LNCS, pages 377–391, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[BEPU+20] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-LWE. In Clemente Galdi and Vladimir Kolesnikov, editors, SCN 20, volume 12238 of LNCS, pages 130–149, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In

Tal Malkin and Chris Peikert, editors, CRYPTO 2021, Part IV, volume 12828 of LNCS, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[Cd10] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, SCN 10, volume 6280 of LNCS, pages 182–199, Amalfi, Italy, September 13–15, 2010. Springer, Heidelberg, Germany.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, CRYPTO 2018, Part II, volume 10992 of LNCS, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, CRYPTO'94, volume 839 of LNCS, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 336–352, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[dCHI+22] Leo de Castro, Carmit Hazay, Yuval Ishai, Vinod Vaikuntanathan, and Muthu Venkitasubramaniam. Asymptotically quasi-optimal cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, EUROCRYPT 2022, Part I, volume 13275 of LNCS, pages 303–334, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[dCJV21] Leo de Castro, Chiraag Juvekar, and Vinod Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. In WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021, pages 29–41. WAHC@ACM, 2021.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, ACM CCS 2022, pages 829–841, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-Point Zero Knowledge and Its Applications. In 2nd Conference on Information-Theoretic Cryptography (ITC 2021), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, CRYPTO 2012, volume 7417 of LNCS, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, TCC 2013, volume 7785 of LNCS, pages 621–641, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[EGK+20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, CRYPTO 2020, Part II, volume 12171 of LNCS, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[FKL+21] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-overhead zero-knowledge for RAM programs. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 178–191, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, EUROCRYPT 2015, Part II, volume 9057 of LNCS, pages 191–219, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, CRYPTO'86, volume 263 of LNCS, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021. `https://eprint.iacr.org/2021/1043`.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, USENIX Security 2016, pages 1069–1083, Austin, TX, USA, August 10–12, 2016. USENIX Association.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In 17th ACM STOC, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press.

[HHK+22] Abida Haque, David Heath, Vladimir Kolesnikov, Steve Lu, Rafail Ostrovsky, and Akash Shah. Garbled circuits with sublinear evaluator. Cryptology ePrint Archive, Report 2022/797, 2022. `https://eprint.iacr.org/2022/797`.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, EUROCRYPT 2020, Part III, volume 12107 of LNCS, pages 569–598, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, 39th ACM STOC, pages 21–30, San Diego, CA, USA, June 11–13, 2007. ACM Press.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, ACM CCS 2013, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, ACM CCS 2016, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

[LAH+22] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. Proving UNSAT in zero knowledge. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, ACM CCS 2022, pages 2203–2217, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[LXZ21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 2968–2985, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, ACM CCS 2001, pages 116–125, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, CRYPTO 2012, volume 7417 of LNCS, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, TCC 2009, volume 5444 of LNCS, pages 368–386. Springer, Heidelberg, Germany, March 15–17, 2009.

[Ore22] Øystein Ore. Über höhere kongruenzen. Norsk Mat. Forenings Skrifter, 1(7):15, 1922.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy, pages 238–252, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[PHP+] James Parker, William Harris, Stuart Pernsteiner, Santiago Cuellar, and Eran Tromer. Proving information leaks in zero knowledge. private communication, to appear soon.

[PRO] PROVENANCE. Making complex zero-knowledge proofs more practical. accessed on Jun 30th 2022.

[Roy22] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, CRYPTO 2022, Part I, volume 13507 of LNCS, pages 657–687, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[Sch18] Peter Scholl. Extending oblivious transfer with low communication via key-homomorphic PRFs. In Michel Abdalla and Ricardo Dahab, editors, PKC 2018, Part I, volume 10769 of LNCS, pages 554–583, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In 2021 IEEE Symposium on Security and Privacy, pages 1074–1091, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, USENIX Security 2021, pages 501–518. USENIX Association, August 11–13, 2021.

[WYY+22] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, ACM CCS 2022, pages 2901–2914, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 2986–3001, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, ACM CCS 2020, pages 1607–1626, Virtual Event, USA, November 9–13, 2020. ACM Press.

[ZLW+21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 159–177, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, EUROCRYPT 2015, Part II, volume 9057 of LNCS, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.