A Simple Technical Introduction to Zero-Knowledge Proofs

Ellen Kolesnikova

December 2024

1 Introduction

This writeup attempts to present the full technical details of recent Zero-Knowledge proof systems in a more beginner-friendly manner.

This article started as a set of notes that I took while reading an excellent technical Systematization of Knowledge paper for VOLE-based interactive ZK proofs [BDSW23]. I then thought that it may be useful to write it up as a hopefully somewhat beginner-friendly technical introduction to ZKP. The reader is expected to be comfortable with math and have a general sense of crypto (encryption, authentication), formal security properties, etc.

2 Definitions

- **Prover** - Party (or algorithm) that proves the truth of a statement - referred to as P.
- **Verifier** - party (or algorithm) that the prover convinces of the truth of a statement - referred to as V.
- **Zero-knowledge proof** - "a cryptographic protocol where a prover can convince a verier that a statement is true, without revealing any further information except for the truth of the statement".
- **Proof of knowledge** - the prover convinces the verifier that it knows a specific desired value, rather than simply that the value exists - the proofs discussed in this paper are proofs of knowledge.
- **Ring** - a set of elements with addition and multiplication defined over them. These operations have certain constraints similar to the constraints of operations within a group. A typical ring is a set of integers modulo 2^n. Rings are interesting because computers work with rings (e.g. `int` is a set of integers mod 2^{32}).

- **Homomorphic encryption** - an type of encryption scheme where encrypted values can be operated upon without the knowledge of the private key, and the values they represent will be appropriately changed. E.g. with additively-homomorphic encryption you can add two encrypted numbers.
- **Schwartz-Zippel Lemma** - Let F be a finite field, S be a subset of F, and P be a non-zero polynomial over F of degree ≥ 0. Then, given a random $s \in S$, the probability of $\text{Prob}[P(s) = 0] \geq \frac{\text{degree}(P)}{|S|}$. Essentially, it is unlikely for a random sampling to "hit" the root of the polynomial P.

3 Notation

- P denotes the prover
- V denotes the verifier
- $[x]$ denotes the VOLE-authenticated value of x, as discussed in Section 5. P possesses x and M, while V possesses k and Δ.

4 Overall goal of the paper

The paper 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".

5 Vector Oblivious Linear Evaluation (VOLE)

"A VOLE correlation is a pair of random variables $(\vec{M}, \vec{x})$ and $(\vec{k}, \Delta)$, where $\vec{x}, \vec{M}, \vec{k}$ are vectors and Δ is a scalar, which are all random subject to the constraint that $\vec{M} = \vec{k} + \vec{x} \cdot \Delta$. One party, in our case the prover P, is given M, x, while the verifier V learns $(\vec{k}, \Delta)$". $\vec{x}$ can be thought of as the vector of secrets, $\vec{M}$ can be thought of as their authenticators (MACs), and $\vec{k}$ can be thought of as the key to the MAC. Prover does not know Δ or $\vec{k}$, and thus cannot forge MACs. The random variables $\vec{x}, \vec{M}, \vec{k}, \Delta$ are chosen from a ring R.

6 How VOLEs are instantiated (through homomorphic encryption)

V chooses Δ first from a sufficiently large subset of R (could be R itself). V then "samples a public key/private key pair, [and] sends an encryption of Δ to P".

P chooses an x and an M and computes an encryption of k using homomorphism:

- P calculates the encryption $enc(x \cdot \Delta)$. This is done by adding $enc(\Delta)$ to itself x times (possible under additive homomorphism; note, use repeated doubling and addition for efficiency for large x).
- P calculates $enc(M)$ using V's public key.
- Finally P evaluates $enc(k) = enc(M) - enc(x \cdot \Delta)$ using homomorphism.
- P sends $enc(k)$ to V, who decrypts it to obtain k.

In this way, P is left with x and M, and V is left with Δ and k, such that $M = k + x \cdot \Delta$ - a VOLE correlation.

There are efficient ways to extend VOLEs (create new ones without having to instantiate them directly, which takes communication).

7 VOLE-based ZK

As we will see in Section 8 and **??**, VOLEs have convenient homomorphic properties: given $[x]$ and $[y]$, it is easy to obtain $[x+y]$ and $[x \cdot y]$. This homomorphism is what makes them so useful in ZK proofs of knowledge.

Indeed, suppose P wants to prove that he knows an input x, such that for some predicate represented as an arithmetic circuit C, it holds $C(x) = 0$. The players, P and V, start with generating these VOLE correlations on C's inputs, corresponding to P's witness x. This is directly achieved by the VOLE functionality (see Section 6 for an example instantiation). Then comes the crucial part: using VOLE homomorphism, P and V jointly evaluate C, gate by gate, starting with the input VOLEs (P computes on x_i and M_i; V computes on k_i and Δ). This results in authentication correlations assigned to each wire of the circuit. Then, if $C(x) = 0$, on C's output wire it should hold that $M = k + 0 \cdot \Delta = k$. If $M \neq k$ at the end, the proof fails. Otherwise, if $M = k$, V will be convinced that $C(x) = 0$ and hence that P knows the secret. This conviction comes from the correctness of the homomorphic operations and from the fact that P is unlikely to guess the "correct" M. Notice that V himself will learn no additional information, because he is only checking that $M = k$, something that is already known to him to be true.

In the next sections, we will see how P and V can efficiently homomorphically evaluate C. And this is actually all there is to it (at the 2^{15}-feet level).

8 Homomorphisms of VOLEs

Additive homomorphism. The problem: players hold authentication correlations for x_0 and x_1, that is P holds x_0, M_0, x_1 and M_1, and V holds k_0, Δ, and k_1, such that $M_i = k_i + x_i \cdot \Delta$. Note that Δ is the same for all VOLEs in the proof - otherwise, the homomorphisms wouldn't work. How can P and V arrive at an authentication correlation for the sum $x_2 = x_0 + x_1$? That is, we want P to obtain M_2 and V to obtain a random k_2, such that $M_2 = k_2 + x_2 \cdot \Delta$.

P sets $M_2 = M_0 + M_1$. Writing it out, $M_2 = M_0 + M_1 = k_0 + x_0 \cdot \Delta + k_1 + x_1 \cdot \Delta = (k_0 + k_1) + (x_0 + x_1) \cdot \Delta$. Evidently, k_2 is simply $k_0 + k_1$, which V can compute.

Since the secret x_2 is the sum of the secrets x_0 and x_1, VOLEs are additively homomorphic. Importantly, this homomorphism is cheap to compute and does not require any communication.

Adding scalar. Now let's consider adding a publicly known scalar c to an authenticated x_0, rather than adding two VOLEs. In this case, P holds x_0, M_0, and the constant c. V holds k_0, Δ, and the constant c. The goal is for P and V to arrive at the authentication correlation of $x_0 + c$, that is to obtain M_2 and k_2, such that $M_2 = k_0 + (x_0 + c) \cdot \Delta$.

This is achieved by P setting $M_2 = M_0$ and V setting $k_2 = k_0 - c \cdot \Delta$. It is easy to see that this results in authenticated $x_2 = x_0 + c$. Indeed: $k_2 + x_2 \cdot \Delta = k_2 + (x_0 + c) \cdot \Delta = k_0 - c \cdot \Delta + (x_0 + c) \cdot \Delta = k_0 - c \cdot \Delta + x_0 \cdot \Delta + c \cdot \Delta = k_0 + x_0 \cdot \Delta = M_0 = M_2$.

Multiplication by scalar. The problem: players hold an authentication correlation for x_0. P holds x_0 and M_0 and V holds k_0 and Δ, such that $M_0 = k_0 + x_0 \cdot \Delta$. Both P and V know the constant c. How can P and V arrive at the authenticated correlation of $c \cdot x_0$, that is, obtain M_2 and k_2 such that $M_2 = k_2 + x_2 \cdot \Delta = k_2 + c \cdot x_0 \cdot \Delta$?

This can be achieved by P setting $M_2 = c \cdot M_0$. Then, $M_2 = c \cdot (k_0 + x_0 \cdot \Delta) = ck_0 + cx_0 \cdot \Delta$. Clearly, k_2 is just $c \cdot k_0$ and x_2 is the desired cx_0.

Since the secret x_2 is the product of the secret x_0 and the constant c, VOLEs are homomorphic with respect to multiplication by constants.

9 VOLE-based ZK - example

Let's look at a simple example of a ZK proof that could be easily accomplished with VOLEs:

- Imagine P is trying to prove that he knows values a, b that satisfy this equation: $a + 2b = 0$. He wants to do this without explicitly revealing his values for a and b.
- P and V could evaluate arithmetic circuit implementing a function $C(a, b) = a + 2b$ using VOLEs to easily accomplish this.
- First, they initialize two VOLEs which encode the secrets a and b (as described in the earlier protocol). P gets M_a, a, M_b, and b, while V gets k_a, k_b, and Δ.
- P and V then multiply the VOLE corresponding to b by 2. P gets a new secret $2b$ and its corresponding MAC $2M_b$, while V gets the key $2k_b$. Δ remains unchanged.
- Finally P and V add the two VOLEs together. P obtains the secret $a + 2b$ with a MAC of $M_a + 2M_b$. V obtains the key $k_a + 2k_b$ and still has Δ.

Going back to the properties of VOLEs, $M_a + 2M_b$ must equal $(k_a + 2k_b) + (a + 2_b) \cdot \Delta$.

- Recall that, if P has correct values for a and b, $a + 2b = 0$. Thus, $M_a + 2M_b = (k_a + 2k_b) + (0) \cdot \Delta = k_a + 2k_b$ with an honest prover.
- P can send V $M_a + 2M_b$. If this is equal to V's value $k_a + 2k_b$, V will be convinced that P indeed knows values of a andb such that $a + 2b = 0$.
- The only way P could have come up with the correct value for $M_a + 2M_b$ without knowing the correct secrets a, b would be if he could guess Δ, which would give him access to all the k-values. However, the probability of P guessing Δ is extremely low, especially as the size of the ring R gets larger.

10 Multiplication of VOLEs

Unfortunately, multiplication does not fit in with VOLEs as neatly as linear transformations (addition and scalar multiplication). Given two VOLE encryptions of secrets, there is no way to easily (i.e. without communication) generate a VOLE encryption of the product of those secrets. Therefore, for every multiplication gate in the ZK circuit, P and V need to exchange messages to generate a new VOLE for the output. (This is, of course, possible since P knows both of the secret inputs and can compute the output secret). P then needs to prove to V that this VOLE was generated correctly; i.e. the secret it authenticates is actually the product of the gate's input secrets. This proof is called a "multiplication check". There are several state-of-the art ways to do this multiplication check, each of them quite different. We will discuss a few of them in the following sections.

Crucially, for efficiency, multiplication checks are usually processed in a batch (vectorized).

11 Wolverine multiplication check [WYKW21]

Brief overview: Let $[x_i]$ and $[y_i]$ be the (authenticated) inputs to a multiplication gate i. Let the gate's output be $z_i = x_i \cdot y_i$, and $[z_i]$ is the corresponding authentication correlation. P proves, with a high probability, that he calculated z_i correctly, resulting in the correct $[z_i]$.

With the Wolverine protocol, P can prove each multiplication gate simultaneously, either during or after the entire ZK circuit evaluation. At a high level, P and V start by generating many *random* VOLE-authenticated triples $([a], [b], [c])$ with the constraint that $a \cdot b = c$. V then asks P to reveal some of these triples. If a significant number of the triples are incorrectly constructed, V has a high chance of opening an incorrect one and thus catching P cheating. If all triples V checks are constructed correctly, he then goes on to match every unopened triple with a multiplication gate i (several triples are mapped to each

gate). In a procedure described later, he can detect cheating if either the triple or the actual gate is incorrectly constructed. If both are incorrect, V might not be able to detect cheating; however it is very unlikely that P will be lucky enough to not get caught in other stages of the proof with so many incorrect triples.

Wolverine multiplication check in more detail. Let the verification circuit C have n multiplication gates. P and V hold their parts of VOLE-authenticated triples $([x_i], [y_i], [z_i])$ for $i \in n$. P wishes to prove that for every $i, x_i \cdot y_i = z_i$. Players set parameters B and f and let $\ell = n \cdot B + f$. Here B is the "bucket size", i.e. the number of random triples used to verify each gate. Think of $B = 3$ or so. f is the number of opened random triples, and is proportional to the security parameter. B and f are set to achieve desired proof soundness, i.e. the probability of catching a cheating P.

For $i \in \ell$, P and V generate VOLE correlations for random values a_i, b_i, r_i for $i \in \{1, ..., \ell\}$ using a VOLE extension protocol. Of course, these are not multiplication triples, i.e. $a_i \cdot b_i = r_i$ is unlikely to hold.

Thus, P converts them to triples: For each a_i, b_i, r_i generated in the previous step, P sends V $d_i := a_i \cdot b_i - r_i$. Both parties compute $[c_i] := [r_i] + d_i$. Note that this authenticated addition is possible for P and V to do without communication, as discussed in Section 8. Additionally, note that c_i is the product $a_i \cdot b_i$ if the prover operates honestly. This is because $c_i = r_i + d_i = r_i + a_i \cdot b_i - r_i = a_i \cdot b_i$.

V then chooses a random permutation of the list of triples $([a_i], [b_i], [c_i])$, and sends the permutation to P. The two parties use this permutation to permute $\{([a_i], [b_i], [c_i])\}_{i \in \ell}$. We will see that later this permutation is necessary to prevent P from cheating.

For each $i \in n$, P and V iterate through $j = 1, ..., B$ and perform the following:

1. Select the $(i-1)B + j$th permuted triple. (Recall, triples were randomly permuted by V, ensuring that P cannot select which triples are selected together.) This selection goes sequentially through the list of triples, picking a new one for every j-value for each i-value. When i is 1 and j is 1, this expression selects the first triple; when i is n and j is B, this expression selects the last triple. Let's call the selected authenticated triple $([a], [b], [c])$.

2. P sends $\delta_a := x_i - a$ and $\delta_b := y_i - b$ to V. P and V then compute $[\mu] := [c] - [z_i] + \delta_b \cdot [a] + \delta_a \cdot [b] + \delta_a \cdot \delta_b$. It is easy to verify that if $x_i \cdot y_i = z_i$ and $a \cdot b = c$, then $\mu = 0$ (and conversely, if $x_i \cdot y_i \neq z_i$, then μ is unlikely to be 0, over the random choices of a, b and if $a \cdot b = c$).

   Thus, P and V check that $[\mu] = [0]$. Note that μ, c, z_i, a, and b must be authenticated to allow for this check. Further, all (a, b, c) triples must be hidden from V to hide the values x, y, z. As discussed in Section 8, once δ_a, δ_b are sent, P and V can compute $[\mu]$ using the homomorphism of the authenticated correlation without communication.

3. Then, P and V check that the values $[x_i]-[a]-[\delta_a]$ and $[y_i]-[b]-[\delta_b]$ both encode 0's. It is easy to see that, if δ_a and δ_b are constructed correctly, these expressions evaluate to [0]. The above check is needed to prevent cheating P from choosing incorrect δ_a, δ_b that would result in $\mu = 0$ while $x_i \cdot y_i \neq z_i$.

Crucially, the checks of Items 2 and 3 can be performed *simultaneously*! This reduces the number of rounds for the proof from $O(n)$ to constant. This has huge practical importance: e.g. for proving a 1M multiplication gate circuit on a fast network with $5ms$ latency, the 1M rounds alone will take $10^6 \cdot 5$ms, which is 5000s or 1hr23min!

Another important observation here is that all the [0] checks can be done simultaneously by P sending a *single* hash of all the MACs, which V then compares to the hash of all his keys.

Recall that, in the beginning of this protocol, P and V generated ℓ authenticated triples $([a], [b], [c])$, where $\ell = n \cdot B + f$. However, while iterating i from 1 to n and iterating j from 1 to B, we only covered $n \cdot B$ triples. P opens the remaining f triples (selected by V), providing V another way he can catch P cheating. The two parties compute $[d] := [c] - a \cdot b$ (recall that $c = a \cdot b$ and is authenticated). P and V then check to ensure that $[d]$ encodes a 0. If it does not, $c \neq a \cdot b$, and P cheated somewhere. V can then abort.

12 Multiplication check using circuit randomization

The goal of this check is the same as the goal of the Wolverine check: P has created $[x], [y], [z]$ and needs to prove that $z = x \cdot y$. To do so, P and V follow these steps:

- P and V create a random $[a]$. P creates $[c]$ where $c = a \cdot y$.
- V supplies P a random challenge e. P and V both compute $[\epsilon] = e \cdot [x] - [a]$.
- P opens $[\epsilon]$ by sending ϵ to V. Together, they check that $[\epsilon] - \epsilon = [0]$ - in other words, that P sent the correct ϵ to V. Remember that V already has his share of $[\epsilon]$, which he computed individually before.
- P and V check that $e \cdot [z] - [c] - \epsilon[y] = [0]$. Ignoring the authentications for the sake of checking whether this expression should evaluate to 0, we get $e \cdot z - c - \epsilon y$. We substitute ϵ for $e \cdot x - a$, and we obtain $e \cdot z - c - e \cdot x \cdot y - ay$. Finally, substituting z for $x \cdot y$ and c for $a \cdot y$, we get $e \cdot x \cdot y - a \cdot y - e \cdot x \cdot y + a \cdot y = 0$. Note that the original expression $e \cdot [z] - [c] - \epsilon[y]$ only includes scalar multiplication and addition, so it is simple to calculate without communication based on the homomorphism (see Section 8). If P was honest, P and V should obtain [0].

- If either $[\epsilon] - \epsilon \neq [0]$ or $e \cdot [z] - [c] - \epsilon[y] \neq [0]$, V knows that P has cheated and can abort.
- Now let's prove that if P did not choose z correctly, V would catch him with high probability, even if P opened ϵ correctly. Imagine P set $z = x \cdot y + \delta$ for $\delta \neq 0$. P can also choose $c = a \cdot y + \gamma$ for γ possibly not equal to 0. Given this, let's prove by contradiction that V will be able to catch P cheating by checking that $e \cdot [z] - [c] - \epsilon[y] = [0]$. Let's imagine that P somehow does manage to make $e \cdot [z] - [c] - \epsilon[y]$ a commitment to 0. That means that $e(x \cdot y + \delta) - a \cdot y - \gamma - e \cdot xy + ay = exy + e\delta - ay - \gamma - exy + ay = e\delta - \gamma = 0$. Thus, $e\delta = \gamma$, and $e = \frac{\gamma}{\delta}$. However, P has to choose γ, δ before he knows e. This means that he would have to guess e correctly before it is given to him if he wants to to obtain values for γ, δ that correspond to a commitment of 0. If e is chosen at random, this is very unlikely. Therefore, P will not be able to create a commitment of 0, leading V to catch him cheating.

Similarly to the Wolverine multiplication checks, these checks can be done simultaneously for all gates.

References

[BDSW23] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. SoK: vector ole-based zero-knowledge protocols. *Des. Codes Cryptogr.*, 91(11):3527–3561, November 2023.

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