AntMan: Interactive Zero-Knowledge Proofs with Sublinear Communication*

Chenkai Weng† Kang Yang‡ Zhaomin Yang§ Xiang Xie§ Xiao Wang†

Abstract

Recent works on interactive zero-knowledge (ZK) protocols provide a new paradigm with high efficiency and scalability. However, these protocols suffer from high communication overhead, often linear to the circuit size. In this paper, we proposed two new ZK protocols with communication sublinear to the circuit size, while maintaining a similar level of computational efficiency.

1. We designed a ZK protocol that can prove B executions of any circuit $\mathcal{C}$ in communication $O(B + |\mathcal{C}|)$ field elements (with free addition gates), while the best prior work requires a communication of $O(B|\mathcal{C}|)$ field elements. Our protocol is enabled by a new tool called as information-theoretic polynomial authentication code, which may be of independent interest.
2. We developed an optimized implementation of this protocol which shows high practicality. For example, with $B = 2048$, $|\mathcal{C}| = 2^{20}$, and under 50 Mbps bandwidth and 16 threads, QuickSilver, a state-of-the-art ZK protocol based on vector oblivious linear evaluation (VOLE), can only prove 0.78 million MULT gates per second (mgps) and send one field element per gate; our protocol can prove 14 mgps ($18\times$ improvement) and send 0.0064 field elements per gate ($156\times$ improvement) under the same hardware configuration.
3. Extending the above idea, we constructed a ZK protocol that can prove a single execution of any circuit $\mathcal{C}$ in communication $O(|\mathcal{C}|^{3/4})$. This is the first ZK protocol with sublinear communication for an arbitrary circuit in the VOLE-based ZK family.

1 Introduction

Zero-knowledge (ZK) proofs allow a prover to convince a verifier that a circuit $\mathcal{C}$ is satisfiable in such a way that the verifier learns nothing beyond the satisfiability of circuit $\mathcal{C}$. This work focuses on interactive (and thus designated-verifier) zero-knowledge proofs of knowledge that often feature high computational efficiency and scalability. ZKGC [JKO13] is an early result of such-type ZK proofs, and requires λ bits of communication per AND gate for a computational security parameter λ. More recently, ZK protocols based on vector oblivious linear evaluation (VOLE) [WYKW21, DIO21, BMRS21, YSWW21, WYX+21, BBMH+21, DILO22] have emerged with comparable or even better computational efficiency while reducing the communication to one field element per multiplication gate (thus 1 bit per AND gate for Boolean circuits). State-of-the-art implementations [YSWW21, BBMH+21, DILO22] can prove tens of millions of gates per second and easily scale to prove trillions of gates. Although huge concrete improvement in

*"Heroes don't get any bigger" (in communication).

†Northwestern University, {ckweng@u,wangxiao@cs}.northwestern.edu

‡State Key Laboratory of Cryptology, yangk@sklc.org

§Shanghai Key Laboratory of Privacy-Preserving Computation and MatrixElements Technologies, {yangzhaomin,xiexiang}@matrixelements.com

practicality, VOLE-based ZK protocols often require communication linear to the circuit size, something undesirable when the circuit is large.

There have been some attempts to address the communication complexity in VOLE-based ZK protocols, and prior works can reduce the communication for selected types of circuits. For example, protocol Mac′N′Cheese [BMRS21] incorporates the "stacking" [HK20] approach so that when proving a branch statement, the communication is only linear to the largest branch. QuickSilver [YSWW21] proposes a ZK protocol to prove the computation of multiple polynomials with communication only linear to the highest degree of all polynomials. We are also aware of a very recent work LPZKv2 [DILO22] that has up to $2\times$ improvement when the circuit structure is arbitrary, and the improvement can be higher when the circuit is more structured. Despite a fair amount of progress in reducing the communication, there are two central open questions:

1. **Sublinear communication for loops/SIMD circuits.** Loops appear in almost all common statements. In the context of ZK, they are closely related to another concept called as single instruction multiple data (SIMD) circuits, because the prover can provide an extended witness for the inputs and outputs of all iterations of a loop (and thus all iterations can be verified in parallel rather than sequential). A protocol that can take advantage of this structure would be very useful. For interactive ZK protocols, the closest is QuickSilver [YSWW21], which can only handle polylogarithmic-depth SIMD circuits: their protocol models circuits as polynomials, so a depth-d circuit could lead to a degree-2^d polynomial.

2. **Sublinear communication for arbitrary circuits.** When it comes to a single execution of an arbitrary circuit, no prior work can even break the communication barrier of one field element per gate, let alone sublinear communication. The most related result is LPZKv2 [DILO22], which needs 0.5 to 1 field elements per MULT gate and thus can achieve $2\times$ improvement at best.

Note that both of them are achievable in NIZKs, e.g., Ligero [AHIV17] and GKR-style proofs [GKR08, ZLW+21], but their performance and scalability are far less than interactive ZK proofs.

1.1 Our Contributions

In this paper, we make huge progress in designing scalable and computationally efficient interactive zero-knowledge protocols. Our new ZK protocols achieve sublinear communication complexity and maintain a similar level of computational efficiency compared to prior VOLE-based ZK proofs.

Sublinear ZK proof for SIMD circuits. We designed an interactive ZK protocol with sublinear communication for proving SIMD circuits. Our protocol only needs to communicate $O(B + |\mathcal{C}|)$ field elements to prove B executions of any circuit $\mathcal{C}$ (which we will refer to as a $(B, |\mathcal{C}|)$-SIMD circuit). The computational complexity is $O(B|\mathcal{C}| \log B)$. This protocol is free for addition gates, and can be seamlessly integrated with existing VOLE-based ZK protocols so that one can use our new protocol to prove SIMD sub-circuits and use other protocols to prove non-SIMD parts. This new protocol crucially relies on a new commitment scheme called as information-theoretic polynomial authentication code (IT-PAC)[1] that can commit to a high-degree polynomial with constant communication cost. The main ZK protocol uses IT-PACs to commit to the wire values across all executions of the circuit and prove the correctness of the committed values gate-by-gate.

We implemented this protocol, and it shows great concrete performance. For example, under a 50 Mbps bandwidth and 16 threads, our implementation on $(2^{11}, 2^{20})$-SIMD circuits (2^{31} multiplication gates) can prove 14 million multiplication gates per second, and communicates 0.0064 field elements per gate. Compared to a state-of-the-art VOLE-based ZK protocol QuickSilver, we achieve $18\times$ improvement in throughput and $156\times$ improvement in communication under the same network bandwidth and configuration.

[1] It is a generalization of the closely related information-theoretic MAC.

SIMD circuits in the context of ZK appear widely in many statements and thus our protocol is widely applicable.

1. **RAM-based computation.** In the state-of-the-art RAM-based computation [FKL+21], each RAM access requires proving a small statement about the data consistency. In this case, B is linear to the number of RAM operations, and $|\mathcal{C}|$ is linear to the bit-length of the index and the payload.

2. **Deep neural network inference.** Many neural networks require repetitive non-linear operations, e.g., activation functions like ReLU and Sigmoid, at many layers on each model parameter. In this case, B is linear to the neural-network model size, and $|\mathcal{C}|$ is linear to the size of each non-linear operation.

3. **Hash/encrypt long messages.** Almost all cryptographic hash functions and encryption schemes involve repetitive structures of a compression function or a block cipher. This is true for ZK-friendly schemes as well. In this case, B is linear to the message size; $\mathcal{C}$ is the circuit of a compression function or a block cipher.

What's more interesting, many applications above require a large $B \geq |\mathcal{C}|$, in which case the communication complexity can be further reduced to $O(\sqrt{B|\mathcal{C}|})$ by tuning the parameters.

Sublinear ZK proof for generic circuits. Further developing our ideas, we designed an interactive ZK protocol that achieves sublinear communication for proving a single execution of an arbitrary circuit. For any circuit $\mathcal{C}$, this protocol only needs to send $O(|\mathcal{C}|^{3/4})$ field elements with computational complexity of $O(|\mathcal{C}| \log |\mathcal{C}|)$ This is the first ZK protocol that achieves sublinear communication for proving an arbitrary circuit in the VOLE-based ZK family. The main idea is to break down the circuits into individual gates and prove them as SIMD circuits. Then we design a mechanism to check the consistency of wire values between different gates. Although IT-PACs can be used to check element-wise equality almost for free, the wiring of an arbitrary circuit can be highly complicated. This is addressed by a new cross-polynomial consistency check. We provide an in-depth technical overview of this protocol in Section 2.4.

Our improvements in communication come with heavier computation, i.e., the computational complexity is no longer linear to the circuit size compared to prior works [YSWW21, BBMH+21, DILO22]. However, we show that for SIMD circuits, our protocol achieves a very similar level of concrete computational efficiency, and we believe further improvement to our generic-circuit protocol can make it practical as well.

2 Technical Overview

2.1 VOLE-based ZK Proofs

We provide the relevant background of prior work on interactive zero-knowledge proofs [BMRS21, YSWW21, DIO21], which serves as a building block and also a starting point of our protocol. These ZK protocols work by using VOLE as an interactive commitment scheme and constructing a "commit-and-prove" protocol. Such a commitment is also referred to as information-theoretic message authentication code (IT-MAC) and we will use both names interchangeably. Briefly, to commit a value $x \in \mathbb{F}$ held by a prover $\mathcal{P}$, we let $\mathcal{P}$ obtain an MAC $\mathsf{M} \in \mathbb{F}$, and let a verifier $\mathcal{V}$ obtain a local key $\mathsf{K} \in \mathbb{F}$ and a uniform global key $\Delta \in \mathbb{F}$, such that $\mathsf{M} = \mathsf{K} + x \cdot \Delta$. Prover $\mathcal{P}$ can later open the value by sending (x, M) to $\mathcal{V}$, who can check that the above equation holds; a cheating prover trying to open to a different value (e.g., x') would need to find a value M' such that $\mathsf{M}' = \mathsf{K} + x' \cdot \Delta$, which is as hard as guessing Δ. A detailed description of IT-MACs can be found in Section 3.1.

A VOLE-based ZK protocol works in two phases. First, two parties obtain IT-MACs of all wire values in the circuit; since IT-MACs are additively homomorphic, $\mathcal{P}$ just needs to commit to all input wire values and the output wire values of all multiplication gates (where the commitments to the output wire values of

addition gates can be computed locally). Second, two parties run a sub-protocol to check that all multiplication gates are correctly computed, i.e., the committed output wire values are equal to the multiplication of the committed input wire values. The design of such checking is the core of prior works and state-of-the-art protocols only require communication sublinear to the circuit size. Overall, the communication mainly comes from the first step where committing one wire value requires $\mathcal{P}$ to send one field element. When proving a circuit with $|\mathcal{C}|$ multiplication gates, this step already requires communication of $|\mathcal{C}|$ field elements. As mentioned in the introduction, for circuits with certain structures (e.g., low-degree polynomials and branches), it is possible to obtain asymptotically better concrete communication, but no prior work can be applied to prove a single or multiple executions of an arbitrary circuit, for which the best-known result is still linear to the circuit size.

2.2 A New Commitment for Sublinear ZK

The main goal of this paper is to significantly reduce the communication of VOLE-based ZK protocols without giving up its merits: high concrete computational efficiency and memory scalability. To this end, our first step is to design a polynomial commitment scheme where the communication is sublinear to the polynomial degree. With computational tools, this is often easy: one could simply use a random oracle or, if additive homomorphism is needed, apply Reed-Solomon encoding first as in Ligero [AHIV17]. Here, the main challenge is to design a commitment scheme that is additively homomorphic, information-theoretic and has the extra properties that enable our sublinear ZK protocol.

Our idea is to extend VOLE-based commitments beyond linear relationships. However, the exact way of combining polynomials and IT-MACs is crucial. Suppose $\mathcal{P}$ wants to commit to a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most k. We let $\mathcal{V}$ sample a uniform key $\Lambda \in \mathbb{F}$ and make $\mathcal{V}$ obtain $f(\Lambda)$ in some way to be detailed later. By having $\mathcal{V}$ hold Λ and $f(\Lambda)$, the polynomial $f(\cdot)$ is statistically binding as long as $\mathcal{P}$ does not know Λ. This is similar to how VOLE-based commitment works. Furthermore, it is additively homomorphic: if $\mathcal{V}$ uses the same key Λ across different polynomials $f(\cdot)$ and $g(\cdot)$, then $\mathcal{V}$ can compute the binding of polynomial $h(\cdot) = f(\cdot) + g(\cdot)$ via $h(\Lambda) = f(\Lambda) + g(\Lambda)$. However, it is not hiding as $\mathcal{V}$ knows the polynomial value evaluated at the point Λ.

Adding the hiding property. Before we get into how we enable hiding, one crucial observation is that $f(\Lambda)$ should be hidden from the prover $\mathcal{P}$ as well, in addition to the verifier $\mathcal{V}$: if $\mathcal{P}$ sees the evaluation on many $\mathcal{P}$-known polynomials, it can solve Λ locally. One attempt is to mask the value $f(\Lambda)$ and then raise the degree. For example, we can let $\mathcal{P}$ and $\mathcal{V}$ obtain $\mathsf{M} \in \mathbb{F}$ and $\mathsf{K} \in \mathbb{F}$, respectively, such that $\mathsf{M} = \mathsf{K} + f(\Lambda) \cdot \Lambda$. This is a strict generalization of IT-MACs (when $f(\cdot)$ is a constant) and would work as a polynomial commitment. However, as it will be clear soon, our protocol requires $\mathcal{V}$ to open Λ to $\mathcal{P}$ at some point, and then $\mathcal{P}$ proves statements about $f(\Lambda)$ in zero-knowledge. This is challenging given this design: once Λ is revealed to $\mathcal{P}$, binding is not guaranteed.

Our idea is to compose an IT-MAC on top of a polynomial: we use VOLE (which has an independent global key Δ) to authenticate the value $f(\Lambda)$. For $\mathcal{P}$ to commit to a polynomial $f(\cdot)$, we let $\mathcal{P}$ hold $\mathsf{M} \in \mathbb{F}$ and $\mathcal{V}$ hold $\mathsf{K} \in \mathbb{F}$ such that

$$\mathsf{M} = \mathsf{K} + f(\Lambda) \cdot \Delta,$$

where Δ is a uniform global key of IT-MACs held by $\mathcal{V}$ and no party knows $f(\Lambda)$. The commitment is now hiding and is still binding: a prover opening to $(\mathsf{M}, f(\cdot)) \neq (\mathsf{M}', f'(\cdot))$ means that $\mathsf{M} - \mathsf{M}' = (f(\Lambda) - f'(\Lambda)) \cdot \Delta$. Except for probability $O(1/|\mathbb{F}|)$, a cheating prover would need to guess at least Λ or Δ to find such two tuples. Because this commitment scheme shares a lot of similarities with IT-MACs, we refer to this commitment scheme as information-theoretic polynomial authentication code (IT-PAC). It is information-theoretic because, after distributing the values (M and K) to the two parties, IT-PAC is both perfectly hiding and statistically binding. However, one can easily see that the distribution of IT-PACs

requires a computational assumption; this is similar to IT-MACs: it requires a computational assumption to generate IT-MACs but the format itself is information-theoretic. In the following, we will describe how to use IT-PACs to design sublinear ZK proofs, and then discuss how to distribute an IT-PAC with $O(1)$ amortized communication for each polynomial.

2.3 Sublinear ZK Proof for SIMD Circuits

We briefly talk about our ZK protocol that can prove B evaluations of any circuit $\mathcal{C}$ in communication $O(B + |\mathcal{C}|)$ rather than $O(B|\mathcal{C}|)$, required by state-of-the-art VOLE-based protocols. Our main idea is to encode the values over all B evaluations on a wire in circuit $\mathcal{C}$ into one polynomial and commit to it using an IT-PAC. Fixing any set of B coordinates, we can always encode a vector $\boldsymbol{x} \in \mathbb{F}^B$ to a degree-$(B-1)$ polynomial $f(\cdot) \in \mathbb{F}[X]$ using polynomial interpolation. Then prover $\mathcal{P}$ just needs to prove that all gates are computed correctly. Due to the additively homomorphic property of IT-PACs, the computation of addition gates is trivial to be correct. The biggest challenge is to prove the correct computation of multiplication gates in sublinear communication cost, which are all committed compactly under IT-PACs; we detail our approach below.

For B evaluations of a multiplication gate in circuit $\mathcal{C}$, the prover $\mathcal{P}$ holds the vectors of wire values $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}^B$ such that $x_i \cdot y_i = z_i$ for all $i \in [1, B]$, and encodes $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ into three degree-$(B-1)$ polynomials $f(\cdot), g(\cdot), h(\cdot) \in \mathbb{F}[X]$. Two parties $\mathcal{P}$ and $\mathcal{V}$ hold the IT-PACs of polynomials $f(\cdot), g(\cdot), h(\cdot)$, i.e., $\mathcal{P}$ has $\mathsf{M}_f, \mathsf{M}_g, \mathsf{M}_h$ and $\mathcal{V}$ has $\mathsf{K}_f, \mathsf{K}_g, \mathsf{K}_h$ such that

$$\mathsf{M}_u = \mathsf{K}_u + u(\Lambda) \cdot \Delta, \quad \text{for all } u \in \{f, g, h\}.$$

Our observation is that $\mathcal{P}$ could prove this relationship in two steps without knowing the polynomial values. The prover $\mathcal{P}$ computes the fourth degree-$(2B-2)$ polynomial $\widetilde{h}(\cdot) = f(\cdot) \cdot g(\cdot) \in \mathbb{F}[X]$, and then commits to it using an IT-PAC. Now $\mathcal{P}$ can prove that among all four IT-PACed polynomials: 1) the newly committed polynomial $\widetilde{h}(\cdot)$ is indeed the correct multiplication of input polynomials $f(\cdot), g(\cdot)$; and 2) the degree-$(B-1)$ polynomial $h(\cdot)$ and the degree-$(2B-2)$ polynomial $\widetilde{h}(\cdot)$ share the same values in the first B evaluations, i.e., $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [1, B]$ given the fixed points $\alpha_1, \ldots, \alpha_B \in \mathbb{F}$. Each of two properties is efficiently provable with little to no communication cost and we will discuss each step below.

Proving polynomial multiplication. To prove the correctness of polynomial multiplication, we simply need to prove that $\widetilde{h}(\Lambda) = f(\Lambda) \cdot g(\Lambda)$ because of the Schwartz–Zippel lemma and that Λ is uniform and not known to the prover. One would naturally want to extend the idea of QuickSilver [YSWW21] (which in turn is inspired by Line-Point ZK [DIO21]), but the setup is not exactly the same. Prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold the IT-MACs of the above three values $f(\Lambda), g(\Lambda), \widetilde{h}(\Lambda)$, but $\mathcal{P}$ does not know the values being MACed; what's more, $\mathcal{P}$ knowing multiple such polynomial values can break binding of the commitment scheme by solving the key Λ.

This prompts us to rethink the role of Λ in our protocol. As mentioned above, the randomness in Λ provides binding to IT-PACs and a prover knowing Λ is able to open a commitment to other values. However, there is no harm in revealing Λ to $\mathcal{P}$ if there is no more opening (or equivalently, if all opening messages from $\mathcal{P}$ are committed before seeing Λ). Therefore, we can check the correctness of the polynomial multiplication by: 1) letting $\mathcal{P}$ commit to all opening messages using IT-PACs; 2) $\mathcal{V}$ reveals Λ to $\mathcal{P}$ (if $\mathcal{V}$ sends an incorrect Λ, it will be caught easily when this check is incorporated with other parts of the protocol); 3) two parties check the aforementioned relationship using QuickSilver for all polynomial multiplications with $O(1)$ communication overhead.

Proving degree reduction. In this check, the prover $\mathcal{P}$ has polynomials $h(\cdot)$ and $\widetilde{h}(\cdot)$; two parties $\mathcal{P}$ and $\mathcal{V}$ hold the IT-PACs on these two polynomials, and want to prove that $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [1, B]$. We observe that this property is "sacrificable" [DPSZ12]: any two pairs of polynomials $(h_1(\cdot), \widetilde{h}_1(\cdot))$ and

$(h_2(\cdot), \widetilde{h}_2(\cdot))$ satisfy the property, if and only if their random linear combination also satisfies the same property except for probability $O(1/|\mathbb{F}|)$. Therefore, this property can be checked in a batch: two parties first generate two IT-PACed uniform polynomials $(r(\cdot), s(\cdot))$ that are supposed to satisfy the desired property when the prover is honest. Then two parties compute a random linear combination of IT-PACs between input polynomials to be checked and the newly generated random polynomials; random linear combination can be computed locally since the coefficients are public and IT-PACs are additively homomorphic. Now the prover $\mathcal{P}$ can open the resulting random-looking polynomials and the verifier can check the property locally. The protocol can check any number of pairs of polynomials with the same communication cost, so the overall communication cost to check any polynomial number of such relationships on degree-$(B-1, 2B-2)$ polynomials is $O(B)$.

Analysis of communication cost. Both checks can be done in a communication cost at most $O(B)$ assuming a random oracle; before the check we need to commit to $2|\mathcal{C}|$ polynomials each of degree either $(B-1)$ or $(2B-2)$. We will show later how to construct such IT-PACs that can distribute all IT-PACs in communication cost $O(B+|\mathcal{C}|)$. Therefore the total communication complexity is $O(B+|\mathcal{C}|)$. Note that one can further tune the parameters of the protocol: we can group k circuits into one bigger circuit and view the SIMD circuit as $B' = B/k$ subcircuits each of size $C' = k|\mathcal{C}|$, which needs $O(B' + C') = O(B/k + k|\mathcal{C}|)$ communication. When $B \geq |\mathcal{C}|$, we can let $k = \sqrt{B/|\mathcal{C}|}$, which leads to a total communication of $O(\sqrt{B|\mathcal{C}|})$ field elements.

2.4 Sublinear ZK Proof for Generic Circuits

Now we generalize the above idea from SIMD circuits to prove a single evaluation of an arbitrary circuit. We break down the circuit into gates and thus we can perform the SIMD idea on all gates. However, we still need to ensure the consistency between gates: the output wire of one gate may be the input wire of another gate. This is a more challenging task because the wiring of the circuit can be arbitrary.

Assuming a circuit with C_A addition gates and C_M multiplication gates, we can represent the circuit as $\{(a_i, b_i, c_i)\}_{i\in[1,C_A]}$, representing all addition gates, and $\{(a'_i, b'_i, c'_i)\}_{i\in[1,C_M]}$, representing all multiplication gates. We first commit all wire values in all gates: $\{w_{a_i}\}_{i\in[1,C_A]}$, $\{w_{b_i}\}_{i\in[1,C_A]}$, $\{w_{c_i}\}_{i\in[1,C_A]}$, $\{w_{a'_i}\}_{i\in[1,C_M]}$, $\{w_{b'_i}\}_{i\in[1,C_M]}$, and $\{w_{c'_i}\}_{i\in[1,C_M]}$. Since IT-PACs commit to values in a batch, we commit to wire values in each set in batches of size B. Now, two parties $\mathcal{P}$ and $\mathcal{V}$ can easily prove that $\{w_{a_i} + w_{b_i} = w_{c_i}\}_{i\in[1,C_A]}$ and $\{w_{a'_i} \cdot w_{b'_i} = w_{c'_i}\}_{i\in[1,C_M]}$ by using the SIMD technique above. Note that given a circuit, we commit to $3(C_A + C_M)$ wires values but there are only $C_A + C_M + I$ different wires so many wire values appear more than once in different IT-PACs at different slots. Now we just need to prove the consistency between different IT-PAC committed values.

To be more specific, prover $\mathcal{P}$ has $(\mathsf{M}_f, f(\cdot)), (\mathsf{M}_g, g(\cdot))$; verifier $\mathcal{V}$ has $\mathsf{K}_f, \mathsf{K}_g$ such that for $\mathsf{M}_u = \mathsf{K}_u + u(\Lambda)\cdot\Delta$ for $u \in \{f, g\}$. $\mathcal{P}$ wants to prove that $f(\alpha_i) = g(\alpha_j)$ for some $i, j \in [1, B]$. One observation is that such a check is also "sacrificable" just as discussed before. Thus, we can have $\mathcal{P}$ first commit to two random polynomials $r(\cdot)$ and $s(\cdot)$ using IT-PACs under the constraint that $r(\alpha_i) = s(\alpha_j)$. Then, two parties can check the consistency of the committed polynomials $f(\cdot), g(\cdot)$ by opening the IT-PAC commitments of $\chi \cdot f(\cdot) + r(\cdot)$ and $\chi \cdot g(\cdot) + s(\cdot)$, where $\chi \in \mathbb{F}$ is a public coefficient. Now $\mathcal{V}$ can locally check if the i-th and the j-th locations have the same value. This check has a communication of $O(B)$ mostly to open polynomials of degree B.

For a fixed pair of indices (i, j), the above check can be easily extended to check for a list of pairs of polynomials without increasing the communication. Although there are at most $2(|\mathcal{C}|/B)$ consistency checks in the circuit, each of them belongs to one of the B^2 consistency checks each with different (i, j). Therefore the total communication of the wire-value consistency check is $O(B^3)$. In summary, we need $O(|\mathcal{C}|/B + B)$ communication to commit and prove all gates and then $O(B^3)$ to check the consistency

between output wire values and input wire values. Here, we have the freedom to pick the parameter B, and can set $B = O(|\mathcal{C}|^{1/4})$ obtaining a total communication of $O(|\mathcal{C}|^{3/4})$.

2.5 Efficiently Generating IT-PACs

Finally, we introduce an efficient protocol for distributing IT-PACs with the amortized communication cost independent of the polynomial degree. We note that with this constraint, preprocessing IT-PACs does not appear as useful: even given an IT-PAC on a random polynomial, we still need communication linear to the polynomial degree to obtain an IT-PAC on a chosen polynomial. We found that the greatest challenge is to make the commitment extractable, something extremely difficult given that IT-PAC is information-theoretic. Our key observation is that, we do not need the commitment to be extractable except for circuit-input wires; for wires in the circuits, a stand-alone definition is already sufficient to achieve proof of knowledge in the underlying ZK protocol. This enables us to design a more efficient IT-PAC generation protocol, and below we sketch out our idea.

Suppose that $\mathcal{P}$ wants to commit to a polynomial f of degree at most k. First, $\mathcal{V}$ picks a uniform $\Lambda \in \mathbb{F}$, and computes encryption of Λ^i denoted by $\langle \Lambda^i \rangle$ for each $i \in [1, k]$ using an additively homomorphic encryption (AHE) scheme. $\mathcal{V}$ sends all the ciphertexts $\langle \Lambda \rangle, \ldots, \langle \Lambda^k \rangle$ to $\mathcal{P}$. Two parties also get a random IT-MAC where $\mathcal{P}$ holds $r, \mathsf{M}^* \in \mathbb{F}$, and $\mathcal{V}$ gets $\mathsf{K}^* \in \mathbb{F}$ and a uniform global key $\Delta \in \mathbb{F}$ such that $\mathsf{M}^* = \mathsf{K}^* + r \cdot \Delta$. Now the $\mathcal{P}$ can locally compute the encryption of $f(\Lambda) - r$ (with encryption of all powers of Λ, computing the encryption of $f(\Lambda)$ only involves linear operations) and send it to $\mathcal{V}$, who can decrypt to get this value $s = f(\Lambda) - r$. Finally, $\mathcal{P}$ outputs $\mathsf{M} = \mathsf{M}^*$ and $\mathcal{V}$ outputs $\mathsf{K} = \mathsf{K}^* - s \cdot \Delta$. The correctness holds because $\mathsf{M} - \mathsf{K} = \mathsf{M}^* - \mathsf{K}^* + (f(\Lambda) - r) \cdot \Delta = f(\Lambda) \cdot \Delta$.

The cost of generating random IT-MACs is negligible small in both communication and computation costs using the recent LPN-based VOLE protocols. After $\mathcal{V}$ sends the encryption of all powers of Λ (reusable across many commitments), $\mathcal{P}$ only needs to send the encryption of one field element per polynomial. Using a modern AHE scheme based on learning with errors (LWE), e.g., BGV [BGV12], the ciphertext overhead of encryption can be easily reduced to a small constant. Therefore, the communication complexity to commit to ℓ polynomials with each of degree k is $O(\ell + k)$ field elements. The use of an LWE-based AHE also ensures that the computation is light as we do not need any heavy operations like homomorphic multiplication. Note that this protocol is only secure against semi-honest adversaries but we will show how this protocol could be sufficient for our ZK protocol.

Using a weaker IT-PAC generation protocol. Recall that in our main ZK protocol, $\mathcal{V}$ needs to reveal Λ to $\mathcal{P}$, at which point the verifier only has Δ as the secret value. The protocol flow of establishing some secret value for soundness and revealing it later on has appeared in prior work: in the ZK proof from garbled circuits (GCs) [JKO13] the verifier sends a garbled circuit to the prover; after the prover evaluates it and commits to the output wire key, the verifier opens all secrets in the garbled circuit so that the prover can check the correctness before opening the output wire key. In our case, the encryption of all powers of Λ plays the role of GC and thus we can use the same "delayed open and check" technique. We let the verifier commit to a seed at first, which will be used to derive Λ and the randomness to encrypt its powers. All provers' messages are committed and are opened only after the $\mathcal{V}$ opens the seed, from which the prover gets Λ as well. If the ciphertexts that $\mathcal{V}$ sent are not computed correctly, $\mathcal{P}$ will abort and will not open the protocol messages. This means that we get security against malicious verifier $\mathcal{V}$ essentially for free. The above idea can essentially elevate any IT-PAC generation protocol that is secure against a semi-honest verifier to the malicious security. Regarding to the malicious prover $\mathcal{P}$, we note that the protocol is 2-round when ignoring the opening of commitments and thus the only thing that malicious $\mathcal{P}$ can do is to send a wrong ciphertext to the verifier who would decrypt to a different value. However, when this happens, it is equivalent to that the prover uses a different polynomial, and thus does not affect the soundness.

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

Upon receiving $(\mathsf{prove}, \mathcal{C}, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$ from a prover $\mathcal{P}$ and $(\mathsf{verify}, \mathcal{C})$ from a verifier $\mathcal{V}$ where the same circuit $\mathcal{C}$ is input, if $\mathcal{C}(\boldsymbol{w}_i) = 0$ for all $i \in [1, B]$, then output (true) to $\mathcal{V}$, else output (false) to $\mathcal{V}$.

Figure 1: The zero-knowledge functionality.

Adding extractability. As mentioned above, an IT-PAC on a polynomial $f(\cdot)$ is very close to an IT-MAC on $f(\Lambda)$, where Λ is known to the verifier. Given the unique structure of IT-PAC, it can be made extractable easily but with additional cost. In particular, we can commit to all coefficients of the polynomial via IT-MACs (that is extractable), and then just need to prove consistency between the values committed in these two representations. Suppose two parties have an IT-PAC on polynomial $f(\cdot) = \sum_{i\in[0,k]} f_i \cdot X^i \in \mathbb{F}[X]$ (i.e., M and K) as well as IT-MACs of all its coefficients (i.e., M_i and K_i for $i \in [0, k]$), such that

$$\mathsf{M} = \mathsf{K} + f(\Lambda) \cdot \Delta \quad \text{and} \quad \mathsf{M}_i = \mathsf{K}_i + f_i \cdot \Delta \text{ for all } i \in [0, k].$$

Observe that $f(\Lambda)$ is a linear combination of f_i where the coefficients are just powers of Λ. Therefore, if $\mathcal{P}$ knew Λ, then two parties could compute the IT-MAC of value $f(\Lambda) - \sum_{i\in[0,k]} f_i \cdot \Lambda^i$, which is supposed to be zero and can be checked essentially for free. However, $\mathcal{P}$ who knows Λ can easily break the binding property of IT-PACs. To address this issue, we require $\mathcal{P}$ to commit to all messages related to committing and opening of IT-PACs before $\mathcal{V}$ sends Λ to $\mathcal{P}$. Essentially, by using distinct keys Λ (for polynomials) and Δ (for IT-MACs), we provide the prover some partial advantage that does not damage the overall security of the protocol. Although enabling extractability requires communication cost linear to the degree of the polynomial, in our ZK protocol, we only need it at the input layer of a circuit, and still use non-extractable IT-PACs for other wires.

3 Preliminaries

Notation. We use λ and ρ to denote the computational and statistical security parameters, respectively. We use $a \leftarrow S$ to denote that sampling a uniformly at random from a finite set S. We will use bold lower-case letters like $\boldsymbol{x}$ for column vectors, and denote by x_i the i-th component of $\boldsymbol{x}$ with x_1 the first entry. For $a, b \in \mathbb{N}$, we write $[a, b] = \{a, \ldots, b\}$. For an algorithm A, we use $y \leftarrow \mathsf{A}(x)$ to denote the operation of running A on input x and setting y as the output. If A is probabilistic and adopts a randomness r, we explicitly write $y \leftarrow \mathsf{A}(x; r)$. We denote by $f(\cdot)$ a degree-k polynomial over a field $\mathbb{F}$. For simplifying the description, we say that $f(\cdot)$ is a degree-k polynomial, meaning that $f(\cdot)$ has a degree at most k unless otherwise specified. We will use $\mathsf{negl}(\cdot)$ to denote a negligible function such that $\mathsf{negl}(\lambda) = o(\lambda^{-c})$ for every positive constant c. A circuit $\mathcal{C}$ over a field $\mathbb{F}$ consists of input, output, addition and multiplication gates, where input gates use circuit-input wires as their output wires and output gates use circuit-output wires as their input wires. We use $|\mathcal{C}|$ to denote the number of all gates in the circuit $\mathcal{C}$, and always assume that $\mathcal{C}$ is evaluated B times with different inputs. If $B > 1$, then B copies of $\mathcal{C}$ constitute a single-instruction-multiple-data (SIMD) circuit, which is also referred to as a $(B, |\mathcal{C}|)$-SIMD circuit. Otherwise, $\mathcal{C}$ is a single generic circuit.

Security model and functionalities. Our protocol UC-realizes the zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{B}$ in the universal composability (UC) framework [Can01], where $\mathcal{F}_{\mathsf{ZK}}^{B}$ is described in Figure 1. If $B = 1$, $\mathcal{F}_{\mathsf{ZK}}^{B}$ is the standard ZK functionality. If $B > 1$, $\mathcal{F}_{\mathsf{ZK}}^{B}$ captures the case of proving satisfiability of $(B, |\mathcal{C}|)$-SIMD circuits. We refer to Appendix A for a brief description of the UC model as well as a review of the commitment functionality.

3.1 Information-Theoretic MAC

Information-theoretic message authentication code (IT-MAC) was originally proposed in the context of secure multi-party computation [BDOZ11, NNOB12]. Following prior work (e.g., [WYKW21, BMRS21, DIO21]), our ZK protocols will view IT-MACs as commitments.

- **Commitments with IT-MACs.** A commitment on a message x is denoted by $[x]$, meaning that a party $\mathcal{P}$ holds $x \in \mathbb{F}$ and an MAC $\mathsf{M} \in \mathbb{F}$, and a verifier $\mathcal{V}$ holds a *local* key $\mathsf{K} \in \mathbb{F}$ and the fixed *global* key $\Delta \in \mathbb{F}$, where K and Δ are uniformly random.
- **Opening.** To open a commitment $[x]$, $\mathcal{P}$ sends (x, M) to $\mathcal{V}$, who checks that $\mathsf{M} = \mathsf{K} + x \cdot \Delta$ holds. If the malicious $\mathcal{P}$ cheats to open a commitment $[x]$ to a message $x' \neq x$, then it has to forge an MAC $\mathsf{M}' = \mathsf{K} + x' \cdot \Delta$, and thus learns the global key $\Delta = (\mathsf{M} - \mathsf{M}')/(x - x')$, which occurs with probability $1/|\mathbb{F}|$. Here we require that $\mathbb{F}$ is a large field, meaning that $|\mathbb{F}| \geq 2^{\rho}$.
- **Linear combination.** The IT-MACs are *additively homomorphic*, meaning that given public coefficients $c_0, c_1, \ldots, c_\ell \in \mathbb{F}$ and commitments $[x_1], \ldots, [x_\ell]$, two parties $\mathcal{P}$ and $\mathcal{V}$ can *locally* compute $[y] = \sum_{i \in [1,\ell]} c_i \cdot [x_i] + c_0$.

For a public value $c \in \mathbb{F}$, $\mathcal{P}$ and $\mathcal{V}$ can directly define the IT-MAC $[c]$ without any communication by setting $\mathsf{M} = 0$ and $\mathsf{K} = -c \cdot \Delta$ respectively. We refer the reader to Appendix A.2 for the generation of IT-MACs from VOLE and the standard VOLE functionality.

The opening of commitments includes two procedures: one is the procedure that makes $\mathcal{P}$ send $x_1, \ldots, x_n$ to $\mathcal{V}$; the other is the $\mathsf{CheckZero}$ procedure that allows $\mathcal{V}$ to check that $[y_i] = [x_i] - x_i$ for all $i \in [1, n]$ are commitments of 0 using the keys of $\mathcal{V}$. Specifically, $\mathsf{CheckZero}$ has the total communication of λ bits and is non-interactive. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. The $\mathsf{CheckZero}$ procedure works by making $\mathcal{P}$ send $\mathsf{M} = \mathsf{H}(\mathsf{M}_1, \ldots, \mathsf{M}_n)$ it to $\mathcal{V}$ who checks whether $\mathsf{M} = \mathsf{H}(\mathsf{K}_1, \ldots, \mathsf{K}_n)$, where $\mathcal{P}$ holds M_i and $\mathcal{V}$ holds K_i such that $\mathsf{M}_i = \mathsf{K}_i$ for each commitment $[y_i]$. Following the security analysis by Damgård et al. [DNNR17], the probability that there exists some $i \in [1, n]$ such that $y_i \neq 0$ but the check passes is at most $1/|\mathbb{F}| + 1/2^\lambda$.

3.2 VOLE-based Zero-Knowledge Proofs

Vector oblivious linear evaluation (VOLE) is a simple yet useful tool in the design of cryptographic protocols. We provide a full summarization of its functionality and construction in Section 3.1. Recent advances of VOLE protocols with sublinear communication have inspired a family of streamable designated-verifier zero-knowledge (DVZK) proofs with fast prover time and a small memory footprint [WYKW21, DIO21, BMRS21, YSWW21, WYX+21, BBMH+21, DILO22]. In particular, given a set of VOLE-based commitments $\{([x_i], [y_i], [z_i])\}_{i \in [1,\ell]}$ over $\mathbb{F}$, these DVZK proofs can enable the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that $z_i = x_i \cdot y_i \in \mathbb{F}$ for all $i \in [1, \ell]$ hold, and only need a small communication (e.g., $\lambda + 2|\mathbb{F}|$ bits for QuickSilver [YSWW21]). We use $\mathsf{DVZK}\left\{([x_i], [y_i], [z_i])_{i \in [1,\ell]} \mid \forall i \in [1, \ell],\ z_i = x_i \cdot y_i\right\}$ to denote such a designated-verifier ZK proof. The proof on the correctness of a set of VOLE-based commitments needs two rounds of communication. The round complexity can be reduced to only one round using the Fiat-Shamir heuristic in the random-oracle model, where the size of $\mathbb{F}$ is at least 2^λ for this case. The DVZK proof securely realizes $\mathcal{F}_{\mathsf{ZK}}^1$ in the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model. We use ϵ_{dvzk} to denote the soundness error of the DVZK proof, where $\epsilon_{\mathsf{dvzk}} = 3/|\mathbb{F}| + q/2^\lambda = 3/|\mathbb{F}| + \mathsf{negl}(\lambda)$ for the case of two-round communication, and q is the number of queries to the random oracle that is used to generate the challenges from a random seed.

3.3 Additively Homomorphic Encryption

We describe the definition of additively homomorphic encryption (AHE) schemes in the private-key setting, which specifies the abstract properties that we need for our IT-PAC generation protocol. To simplify the description of our protocol, we assume that the plaintexts lie in a field $\mathbb{F}$. An AHE scheme consists of the Setup algorithm that generates the set of public parameters par, a key generation algorithm KeyGen, an encryption algorithm Enc and a decryption algorithm Dec. In our IT-PAC generation protocol, we will use $\langle m \rangle = \mathsf{Enc}(\mathsf{sk}, m; r)$ denotes the ciphertext on a message m encrypted with a secret key sk and a randomness r. We require that the AHE scheme satisfies the standard chosen plaintext attack (CPA) security, and achieves the *circuit privacy* [IP07]. Furthermore, we need that the AHE scheme provides the *degree-restriction* property: for some integer $k \geq 1$, given the set of public parameters par and the ciphertexts $\langle \Lambda \rangle, \ldots, \langle \Lambda^k \rangle$ for a uniform $\Lambda \in \mathbb{F}$ as input, it is hard to compute a ciphertext $\langle f(\Lambda) \rangle$ such that $f(\cdot)$ is a polynomial of degree at least $k+1$. The notion of *linear targeted malleability* defined by Bitansky et al. [BCI+13], which eliminates the computation on ciphertexts other than affine linear maps, implies that the above degree-restriction property holds. For the implementation, we instantiate the AHE scheme using the BGV homomorphic encryption with a single level [BGV12]. Following the analysis [KPR18], the BGV-AHE scheme is valid candidate for linear targeted malleability, and thus satisfies the degree-restriction property. We provide the definition of AHE and a brief introduction of BGV-AHE in Appendix A.3.

4 Information-Theoretic Polynomial Authentication Codes

In this section, we present the notion of information-theoretic polynomial authentication codes (IT-PACs). In our ZK protocol, IT-PACs are used as commitments on polynomials. We also describe a useful procedure to check consistency of the evaluation of two sets of polynomials at multiple points. Then, we present a concretely efficient protocol that generates a batch of IT-PACs.

4.1 Definition of IT-PACs

For the sake of simplicity, we define IT-PACs over a large field $\mathbb{F}$. Nevertheless, one can extend the definition of IT-PACs to a more general case in a straightforward manner, where the values are defined over a small field $\mathbb{F}$ (e.g., $\mathbb{F} = \mathbb{F}_2$) and authenticated over a large extension field $\mathbb{K}$. Specifically, the definition of IT-PACs over a large field $\mathbb{F}$ is described as follows:

- **Commitments with IT-PACs.** A verifier $\mathcal{V}$ holds a uniform *global key* $\Delta \in \mathbb{F}$ and another uniform key $\Lambda \in \mathbb{F}$ referred to as a *polynomial key*. Both keys are reused across different IT-PACs. An IT-PAC commitment on a degree-k polynomial $f(\cdot) = \sum_{i=0}^{k} c_i \cdot X^i \in \mathbb{F}[X]$ is denoted by $[f(\cdot)]$, where $\mathcal{P}$ holds a polynomial $f(\cdot) \in \mathbb{F}[X]$ and an MAC $\mathsf{M} \in \mathbb{F}$, while $\mathcal{V}$ holds keys $\mathsf{K}, \Delta, \Lambda \in \mathbb{F}$, such that $\mathsf{M} = \mathsf{K} + f(\Lambda) \cdot \Delta$. As in IT-MACs, the key K is also called as a *local key*. We could also consider an IT-PAC on polynomial $f(\cdot)$ as an IT-MAC on value $f(\Lambda)$.

- **Opening.** To open a commitment $[f(\cdot)]$, $\mathcal{P}$ sends $(f(\cdot), \mathsf{M})$ to $\mathcal{V}$, who checks whether $\mathsf{M} = \mathsf{K} + f(\Lambda) \cdot \Delta$.

  If the malicious $\mathcal{P}$ cheats, it must either forge an MAC $\mathsf{M}' = \mathsf{K} + f'(\Lambda) \cdot \Delta$ on message $f'(\Lambda) \neq f(\Lambda)$ with probability at most $1/|\mathbb{F}|$, or find a different polynomial $f'(\cdot) \neq f(\cdot)$ such that $f'(\Lambda) = f(\Lambda)$. If the second case occurs, then $f'(\cdot) - f(\cdot)$ is a non-zero polynomial of degree at most k, and thus the probability that it is equal to 0 at a random point Λ is at most $k/|\mathbb{F}|$, according to the Schwartz–Zippel lemma. Therefore, the probability that $\mathcal{P}$ succeeds to cheat is at most $(k+1)/|\mathbb{F}|$.

- **Linear combination.** Similar to IT-MACs, IT-PACs are also additively homomorphic. Specifically, given the public coefficients $c_0, c_1, \ldots, c_\ell \in \mathbb{F}$ and IT-PACs $[f_1(\cdot)], \ldots, [f_\ell(\cdot)]$, $\mathcal{P}$ and $\mathcal{V}$ can *locally* compute

Procedure $\mathsf{BatchCheck}_{k,m,t}$

Inputs. Two parties $\mathcal{P}$ and $\mathcal{V}$ hold the following inputs:

- Two sets of IT-PACs $\{[f_1(\cdot)], \ldots, [f_\ell(\cdot)]\}$ and $\{[g_1(\cdot)], \ldots, [g_\ell(\cdot)]\}$ where $f_i()$ is a degree-k polynomial and $g_i()$ is a degree-m polynomial for $i \in [1, \ell]$.
- Let $\{\alpha_1, \ldots, \alpha_t\}$ and $\{\beta_1, \ldots, \beta_t\}$ be two sets of public elements over $\mathbb{F}$. Let $\mathsf{H} : \{0,1\}^\lambda \to \mathbb{F}^\ell$ be a random oracle.

Consistency check. $\mathcal{P}$ and $\mathcal{V}$ check $f_j(\alpha_i) = g_j(\beta_i)$ for all $i \in [1, t], j \in [1, \ell]$ as follows.

LINEAR COMBINATION PHASE: Before the polynomial key Λ is opened, $\mathcal{P}$ and $\mathcal{V}$ execute as follows.

1. $\mathcal{P}$ samples two random polynomials $r(\cdot)$ and $s(\cdot)$ of respective degrees k and m in $\mathbb{F}[X]$ such that $r(\alpha_i) = s(\beta_i)$ for $i \in [1, t]$. Then, $\mathcal{P}$ and $\mathcal{V}$ generate the corresponding IT-PACs $[r(\cdot)]$ and $[s(\cdot)]$.
2. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$ and sends it to $\mathcal{P}$. Then, two parties compute $(\chi_1, \ldots, \chi_\ell) := \mathsf{H}(\mathsf{seed}) \in \mathbb{F}^\ell$.
3. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[f(\cdot)] := \sum_{j=1}^{\ell} \chi_j \cdot [f_j(\cdot)] + [r(\cdot)]$ and $[g(\cdot)] := \sum_{j=1}^{\ell} \chi_j \cdot [g_j(\cdot)] + [s(\cdot)]$. Then, $\mathcal{P}$ sends the polynomial pair $(f(\cdot), g(\cdot))$ to $\mathcal{V}$, who checks that $f(\cdot), g(\cdot)$ have the degrees k and m respectively and $f(\alpha_i) = g(\beta_i)$ for all $i \in [1, t]$. If the check fails, $\mathcal{V}$ aborts.

CHECK PHASE:

4. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[\mu] := [f(\Lambda)] - f(\Lambda)$ and $[\nu] := [g(\Lambda)] - g(\Lambda)$. Then, two parties run $\mathsf{CheckZero}([\mu], [\nu])$ to check that $\mu = 0$ and $\nu = 0$. If the check fails, then $\mathcal{V}$ aborts, else $\mathcal{V}$ accepts.

Figure 2: Procedure for checking the consistency of polynomial evaluation for two sets of IT-PACs.

$[g(\cdot)] = \sum_{i=1}^{\ell} c_i \cdot [f_i(\cdot)] + c_0$, where $g(\cdot) = \sum_{i=1}^{\ell} c_i \cdot f_i(\cdot) + c_0$, $\mathsf{M}_g = \sum_{i=1}^{\ell} c_i \cdot \mathsf{M}_{f_i}$ and $\mathsf{K}_g = \sum_{i=1}^{\ell} c_i \cdot \mathsf{K}_{f_i} - c_0 \cdot \Delta$.

For a public polynomial $f(\cdot) \in \mathbb{F}[X]$, $\mathcal{P}$ and $\mathcal{V}$ can directly define the MAC and key in the IT-PAC $[f(\cdot)]$ as 0 and $-f(\Lambda) \cdot \Delta$ respectively without any interaction. Given $k+1$ distinct elements $\alpha_1, \ldots, \alpha_{k+1} \in \mathbb{F}$, an IT-PAC $[f(\cdot)]$ also commits to the values $f(\alpha_1), \ldots, f(\alpha_{k+1})$, since $k+1$ values uniquely determine a degree-k polynomial and vice versa.

4.2 Batch Check of Polynomial Evaluation

We present an interactive procedure to check the consistency of polynomial evaluation of two sets of IT-PACs at a single point or multiple points. The consistency check is done in a batch with communication independent of the number of IT-PACs. Specifically, given two sets of public field elements $\{\alpha_i\}_{i\in[1,t]}$ and $\{\beta_i\}_{i\in[1,t]}$ and two sets of IT-PACs $\{[f_j(\cdot)]\}_{j\in[1,\ell]}$ and $\{[g_j(\cdot)]\}_{j\in[1,\ell]}$ as input, this procedure allows to check that $f_j(\alpha_i) = g_j(\beta_i)$ for each $i \in [1, t], j \in [1, \ell]$, where $f_j(\cdot)$ is a degree-k polynomial and $g_j(\cdot)$ is a degree-m polynomial for $j \in [1, \ell]$. We require that the procedure does *not* reveal any secret information on these polynomials except that the equalities hold. This is realized by first generating two random IT-PAC commitments $[r(\cdot)]$ and $[s(\cdot)]$ such that $r(\alpha_i) = s(\beta_i)$ for $i \in [1, t]$ if $\mathcal{P}$ is honest, and then opening $[f(\cdot)] = \sum_{i=1}^{\ell} \chi_i \cdot [f_i(\cdot)] + [r(\cdot)]$ and $[g(\cdot)] = \sum_{i=1}^{\ell} \chi_i \cdot [g_i(\cdot)] + [s(\cdot)]$ where $\chi_1, \ldots, \chi_\ell$ are public coefficients sampled at random by $\mathcal{V}$. To achieve better communication efficiency, we can use a random oracle to compress these public coefficients into a random seed. We denote the consistency-check procedure by $\mathsf{BatchCheck}$, which is described in Figure 2. We let $\mathcal{P}$ send two polynomials $f(\cdot)$ and $g(\cdot)$ to $\mathcal{V}$, meaning that $\mathcal{P}$ sends the coefficients of the two polynomials to $\mathcal{V}$. The communication complexity of this procedure is $O(k + m)$.

In the following, we give an important lemma, which will be used in the security proof of our ZK protocol. The proof of this lemma can be found in Appendix B.

Lemma 1. *Let $\mathcal{P}$ be a malicious party who interacts with an honest verifier $\mathcal{V}$ during the execution of* BatchCheck. *Let* H *be a random oracle. If there exists some $i \in [1,t], j \in [1,\ell]$ such that $f_j(\alpha_i) \neq g_j(\beta_i)$, then the probability that $\mathcal{V}$ accepts at the end of* BatchCheck *is at most* $\frac{\max\{k,m\}+2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

We can easily extend the BatchCheck procedure shown in Figure 2 to check the correctness of opening ℓ degree-k polynomials to the values of these polynomials at t different points. Specifically, $\mathcal{P}$ can send $(f_j(\alpha_1), \ldots, f_j(\alpha_t))$ for each $j \in [1,\ell]$ to $\mathcal{V}$. Both parties locally compute an IT-PAC $[g_j(\cdot)]$ for each $j \in [1,\ell]$, where $g_j(\cdot)$ is a public degree-$(t-1)$ polynomial reconstructed from the values $f_j(\alpha_1), \ldots, f_j(\alpha_t)$ using Lagrange interpolation. Then, $\mathcal{P}$ and $\mathcal{V}$ run the $\mathsf{BatchCheck}_{k,(t-1),t}$ procedure to check $f_j(\alpha_i) = g_j(\alpha_i)$ for all $i \in [1,t], j \in [1,\ell]$. In the special case of $t = k+1$, the above procedure is to check the correctness of opening the whole polynomials. In this case, it is unnecessary to mask the linear combination of input IT-PACs $[f_1(\cdot)], \ldots, [f_\ell(\cdot)]$ with a random IT-PAC. The extended procedure as described above may be useful in other ZK protocols and applications.

4.3 Efficient Protocol to Generate IT-PACs

Below, we present a concretely efficient protocol of generating IT-PACs. This protocol works in the ($\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{Com}}$)-hybrid model, and adopts AHE to generate the additive shares of the polynomial-evaluation values at the secret point Λ. Using the additive shares, a VOLE correlation can be locally transformed to a batch of IT-PACs.

For the AHE ciphertexts sent by a verifier $\mathcal{V}$, one can adopt a ZK proof to prove the validity of these ciphertexts. However, the usage of ZK proofs introduces significantly communication overhead (particularly, the size of ciphertexts need to be significantly larger to cover the so-called slack brought about by the ZK proofs). To reduce the communication overhead, we replace the ZK proof with the "commit-then-open" approach. In particular, the correctness of the ciphertexts produced by a verifier $\mathcal{V}$ is guaranteed by committing the randomness to generate the ciphertexts and then opening the randomness at some later point. The randomness can be generated with a random seed and a pseudorandom generator (PRG) to reduce the communication cost. When the ciphertexts sent by $\mathcal{V}$ may be *incorrect* before the randomness is opened, we let the party $\mathcal{P}$ first commit to its homomorphically computed ciphertexts and then open these ciphertexts after checking the correctness of the ciphertexts received from $\mathcal{V}$. This allows to remove the possible leakage of secret polynomials, which is incurred by homomorphically performing polynomial evaluation upon incorrect ciphertexts. The "commit-then-open" approach is sufficient, as the polynomial key Λ will be always opened.

Based on the "commit-then-open" approach, the concretely efficient protocol for generating IT-PACs is described in Figure 3. While the initialization phase to generate a global key Δ needs to be run only once, other phases can be executed multiple times where every execution creates a fresh polynomial key Λ and a batch of IT-PACs under the key Λ. For generating ℓ IT-PACs on degree-k polynomials, the total communication complexity is $O(k+\ell)$, where the communication for generating VOLE correlations is sublinear.

5 Zero-Knowledge Proofs with Sublinear Communication

5.1 Sublinear ZK Proof for SIMD Circuits

In Figure 4, we describe the details of our ZK protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ on proving satisfiability of SIMD circuits in the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model. This protocol also invokes $\Pi_{\mathsf{PAC}}^{(2B-2)}$ as a sub-protocol, and thus needs to call a

Protocol $\Pi^k_{\mathsf{IT\text{-}PAC}}$

Let $\mathsf{AHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. Suppose that two parties $\mathcal{P}$ and $\mathcal{V}$ have already agreed a set of public parameters $\mathsf{par} = \mathsf{Setup}(1^\lambda)$. Let G be a PRG. Let ℓ be the number of IT-PACs to be generated in one execution and k be the maximum degree of the polynomials committed in each IT-PAC.

Initialize. Two parties $\mathcal{P}$ and $\mathcal{V}$ send (init) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns a uniform $\Delta \in \mathbb{F}$ to $\mathcal{V}$.

Create and encrypt polynomial keys.

1. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$, and then $\mathcal{V}$ and $\mathcal{P}$ call the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ with input seed, which returns a handle τ_1 to $\mathcal{P}$.
2. $\mathcal{V}$ samples $\Lambda \leftarrow \mathbb{F}$ and runs $\langle \Lambda^i \rangle \leftarrow \mathsf{Enc}(\mathsf{sk}, \Lambda^i; r_i)$ for all $i \in [1,k]$ where $(r_0, r_1, \ldots, r_k) = \mathsf{G}(\mathsf{seed})$ and $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{par}; r_0)$. Then, $\mathcal{V}$ sends the AHE ciphertexts $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$ to $\mathcal{P}$.

Pre-generation of IT-PACs.

3. $\mathcal{P}$ and $\mathcal{V}$ sends (extend, ℓ) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{F}^\ell$ to $\mathcal{P}$ and $\boldsymbol{v} \in \mathbb{F}^\ell$ to $\mathcal{V}$, such that $\boldsymbol{w} = \boldsymbol{v} + \boldsymbol{u} \cdot \Delta$.
4. For each $j \in [1,\ell]$, on input the j-th polynomial $f_j(\cdot) = \sum_{i=0}^k f_{j,i} \cdot X^i \in \mathbb{F}[X]$, $\mathcal{P}$ computes a ciphertext $\langle b_j \rangle$ with $u_j + b_j = f_j(\Lambda)$ via $\langle b_j \rangle = \sum_{i=1}^k f_{j,i} \cdot \langle \Lambda^i \rangle + f_{j,0} - u_j$.
5. $\mathcal{P}$ and $\mathcal{V}$ call the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ with inputs $\langle b_1 \rangle, \ldots, \langle b_\ell \rangle$, which returns a handle τ_2 to $\mathcal{V}$.

Generation of IT-PACs and opening polynomial keys.

6. $\mathcal{V}$ and $\mathcal{P}$ call the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ on input τ_1, which returns (seed, τ_1) to $\mathcal{P}$. In parallel, $\mathcal{V}$ sends Λ to $\mathcal{P}$. Then, $\mathcal{P}$ computes $(r_0, r_1, \ldots, r_k) := \mathsf{G}(\mathsf{seed})$ and runs $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{par}; r_0)$. $\mathcal{P}$ checks that $\langle \Lambda^i \rangle = \mathsf{Enc}(\mathsf{sk}, \Lambda^i; r_i)$ for all $i \in [1,k]$, and aborts if the check fails. For each $j \in [1,\ell]$, $\mathcal{P}$ sets $\mathsf{M}_j := w_j$.
7. $\mathcal{P}$ and $\mathcal{V}$ call the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ on input τ_2, which returns $(\langle b_1 \rangle, \ldots, \langle b_\ell \rangle, \tau_2)$ to $\mathcal{V}$. Then, for each $j \in [1,\ell]$, $\mathcal{V}$ runs $b_j \leftarrow \mathsf{Dec}(\mathsf{sk}, \langle b_j \rangle)$, and then computes $\mathsf{K}_j := v_j - b_j \cdot \Delta \in \mathbb{F}$.
8. For each $j \in [1,\ell]$, two parties obtain an IT-PAC $[f_j(\cdot)]$, where $\mathcal{P}$ holds $(f_j(\cdot), \mathsf{M}_j)$ and $\mathcal{V}$ holds K_j.

Figure 3: Protocol for generating IT-PACs without ZK proofs in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.

commitment functionality $\mathcal{F}_{\mathsf{Com}}$ as well. When executing sub-protocol $\Pi^{(2B-2)}_{\mathsf{PAC}}$ to generate IT-PACs, the generation of the local keys held by the verifier $\mathcal{V}$ are delayed to the time point after the polynomial key Λ is opened. Thus, the computation of the local keys on the output IT-PACs of addition gates also has to be postponed. While prover $\mathcal{P}$ can execute the check phase of the underlying $\mathsf{BatchCheck}$ procedure *before* Λ is opened, the values from $\mathcal{P}$ could be checked by verifier $\mathcal{V}$*after* Λ is opened and the local keys on the IT-PACs are computed.

When $\mathcal{F}_{\mathsf{VOLE}}$ is instantiated by the recent LPN-based VOLE protocol with sublinear communication, protocol $\Pi^{\mathsf{SIMD}}_{\mathsf{ZK}}$ has the communication complexity of $O(B + |\mathcal{C}|)$ for proving $(B, |\mathcal{C}|)$-SIMD circuits, where note that addition gates are *free* for our protocol. If the underlying DVZK proof has at most two rounds (e.g., DVZK is instantiated by [DIO21, YSWW21, DILO22]), the protocol $\Pi^{\mathsf{SIMD}}_{\mathsf{ZK}}$ has 6 rounds in the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model. Note that all invocations of sub-protocol $\Pi^{(2B-2)}_{\mathsf{PAC}}$ can be made in parallel and all $\mathsf{CheckZero}$ executions can be combined into one execution. Since the LPN-based VOLE protocol realizing $\mathcal{F}_{\mathsf{VOLE}}$ has constant rounds, our ZK protocol has also constant rounds.

For the security of protocol $\Pi^{\mathsf{SIMD}}_{\mathsf{ZK}}$, we prove the following theorem. In this theorem, we assume that the soundness error ϵ_{dvzk} of the underlying ZK proof DVZK is at most $3/|\mathbb{F}| + \mathsf{negl}(\lambda)$, e.g., it is instantiated by QuickSilver [YSWW21]. The formal proof of the theorem is postponed to Appendix C.

Protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$

Inputs. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold a generic circuit $\mathcal{C}$ over a large field $\mathbb{F}$, where $\mathcal{C}$ contains $n = |\mathcal{C}|$ multiplication gates and m input gates. $\mathcal{P}$ holds B witnesses $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B \in \mathbb{F}^m$ such that $\mathcal{C}(\boldsymbol{w}_i) = 0$ for all $i \in [1, B]$. Let $\alpha_1, \ldots, \alpha_B \in \mathbb{F}$ be B distinct elements that are fixed for the whole protocol execution. Let $\delta_i(X) = \prod_{j \in [1,B], j \neq i} (X - \alpha_j)/(\alpha_i - \alpha_j) \in \mathbb{F}[X]$ be a degree-$(B-1)$ polynomial for each $i \in [1, B]$, which is referred to as a Lagrange polynomial.

Initialization. $\mathcal{P}$ and $\mathcal{V}$ send (init) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns a uniform $\Delta \in \mathbb{F}$ to $\mathcal{V}$.

Circuit evaluation. For B executions of circuit $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{V}$ pack B same-type gates into a group in a straightforward way. In particular, for an index i, the parties pack the i-th input/output/multiplication/addition gates from all B executions of circuit $\mathcal{C}$ into a group.

1. $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ shown in Figure 3 to create a uniform key $\Lambda \in \mathbb{F}$.
2. The parties execute the following steps to process the inputs:
   (a) $\mathcal{P}$ and $\mathcal{V}$ send (extend, mB) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns IT-MACs $\{[a_{i,j}]\}_{i \in [1,B], j \in [1,m]}$ to the parties.
   (b) For $i \in [1, B], j \in [1, m]$ (i.e., corresponding to the j-th circuit-input wire of the i-th execution of circuit $\mathcal{C}$), $\mathcal{P}$ sends $b_{i,j} := w_{i,j} - a_{i,j}$ to $\mathcal{V}$, and then both parties locally compute $[w_{i,j}] := [a_{i,j}] + b_{i,j}$.
   (c) For $j \in [1, m]$, for the j-th group of B input gates with input vector $(w_{1,j}, \ldots, w_{B,j})$, $\mathcal{P}$ defines a degree-$(B-1)$ polynomial $u_j(\cdot)$ such that $u_j(\alpha_i) = w_{i,j}$ for $i \in [1, B]$.
   (d) $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ to generate IT-PACs $[u_1(\cdot)], \ldots, [u_m(\cdot)]$.
3. Following a predetermined topological order for circuit $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{V}$ execute as follows:
   (a) For each group of addition gates with input IT-PACs $[f(\cdot)]$ and $[g(\cdot)]$, both parties locally compute an output IT-PAC $[h(\cdot)] := [f(\cdot)] + [g(\cdot)]$.
   (b) For the j-th group of multiplication gates with input IT-PACs $[f_j(\cdot)]$ and $[g_j(\cdot)]$ where $j \in [1, n]$, $\mathcal{P}$ computes a degree-$(2B-2)$ polynomial $\widetilde{h}_j(\cdot) := f_j(\cdot) \cdot g_j(\cdot) \in \mathbb{F}[X]$ and a degree-$(B-1)$ polynomial $h_j(\cdot)$ such that $h_j(\alpha_i) = \widetilde{h}_j(\alpha_i)$ for all $i \in [1, B]$. Then, $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ to generate two IT-PACs $[h_j(\cdot)]$ and $[\widetilde{h}_j(\cdot)]$.

Correctness check of multiplication gates. Prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that the multiplication gates are evaluated correctly.

4. $\mathcal{P}$ and $\mathcal{V}$ execute the linear-combination phase of the $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$ procedure on inputs $\{[h_1(\cdot)], \ldots, [h_n(\cdot)]\}$ and $\{[\widetilde{h}_1(\cdot)], \ldots, [\widetilde{h}_n(\cdot)]\}$ to check that $h_j(\alpha_i) = \widetilde{h}_j(\alpha_i)$ for all $i \in [1, B], j \in [1, n]$.
5. $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ to open Λ to $\mathcal{P}$, and then $\mathcal{V}$ can compute the local keys on all IT-PACs.
6. Then, $\mathcal{P}$ and $\mathcal{V}$ execute the check phase of $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$ to complete the above check. If the check fails, $\mathcal{V}$ aborts.
7. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof $\mathsf{DVZK}\left\{([f_j(\Lambda)], [g_j(\Lambda)], [\widetilde{h}_j(\Lambda)])_{j \in [1,n]} \mid \forall j \in [1, n],\ \widetilde{h}_j(\Lambda) = f_j(\Lambda) \cdot g_j(\Lambda)\right\}$.

Consistency check of input gates and output gates. $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that $[u_j(\cdot)]$ is consistent to $([w_{1,j}], \ldots, [w_{B,j}])$ for $j \in [1, m]$, and the values on all output gates are 0.

8. For each $j \in [1, m]$, $\mathcal{P}$ and $\mathcal{V}$ locally compute an IT-MAC $[z_j] := \sum_{i \in [1,B]} \delta_i(\Lambda) \cdot [w_{i,j}]$, and then run $\mathsf{CheckZero}([u_j(\Lambda)] - [z_j])$ to check $u_j(\Lambda) = z_j$. If the check fails, $\mathcal{V}$ aborts.
9. Let $[v(\cdot)]$ be the IT-PAC associated with the output values of B executions of circuit $\mathcal{C}$. The parties $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{CheckZero}([v(\Lambda)])$ to check $v(\Lambda) = 0$, and $\mathcal{V}$ aborts if the check fails.
10. If $\mathcal{V}$ will abort in some step, then $\mathcal{V}$ outputs false and aborts, else it outputs true.

Figure 4: Sublinear zero-knowledge protocol for SIMD circuits in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.

Theorem 1. *If the* AHE *scheme satisfies the CPA security, degree-restriction and circuit privacy along with* G *is a pseudorandom generator, protocol* $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ *shown in Figure 4 UC-realizes functionality* $\mathcal{F}_{\mathsf{ZK}}^{B}$ *on* $(B, |\mathcal{C}|)$*-SIMD circuits in the* $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$*-hybrid model and the random oracle model with soundness error at most* $\frac{2B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Streaming ZK proofs. For a very large circuit, we need to prove it batch-by-batch, i.e., a batch of gates are proved each time. For every execution of $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, the key Λ is opened, and thus the IT-PAC commitments, which are generated after Λ was opened, are not binding any more. Thus, for the next execution, the verifier has to sample a fresh key Λ' by running the sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$. Then, two parties $\mathcal{P}$ and $\mathcal{V}$ need to convert the IT-PACs $[v_1(\cdot)]_\Lambda, \ldots, [v_\ell(\cdot)]_\Lambda$ that commit to the output values in the current batch into the IT-PACs $[v_1(\cdot)]_{\Lambda'}, \ldots, [v_\ell(\cdot)]_{\Lambda'}$ on the same polynomials, which are used as the input IT-PACs for the next batch. To realize the conversion of IT-PACs $[v_1(\cdot)]_\Lambda, \ldots, [v_\ell(\cdot)]_\Lambda$ from the key Λ to a new key Λ', both parties generate the IT-PACs $[v_1(\cdot)]_{\Lambda'}, \ldots, [v_\ell(\cdot)]_{\Lambda'}$ by running sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ before the polynomial keys Λ and Λ' are opened, and then check the consistency of polynomials between two sets of IT-PACs $\{[v_i(\cdot)]_\Lambda\}_{i\in[1,\ell]}$ and $\{[v_i(\cdot)]_{\Lambda'}\}_{i\in[1,\ell]}$. Specifically, the consistency check of polynomial-key conversion of IT-PACs is done as follows:

- LINEAR COMBINATION PHASE: Before both polynomial keys Λ and Λ' are opened, two parties $\mathcal{P}$ and $\mathcal{V}$ do the following:

  1. $\mathcal{P}$ and $\mathcal{V}$ generate two random IT-PACs $[r(\cdot)]_\Lambda$ and $[r(\cdot)]_{\Lambda'}$ under different polynomial keys by running sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$, where $r(\cdot)$ is a random polynomial and has the same degree as $v_i(\cdot)$ for $i \in [1, \ell]$.
  2. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$ and sends it to $\mathcal{P}$. Then, both parties compute $(\chi_1, \ldots, \chi_\ell) := \mathsf{H}(\mathsf{seed}) \in \mathbb{F}^\ell$.
  3. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[v(\cdot)]_\Lambda := \sum_{i=1}^{\ell} \chi_i \cdot [v_i(\cdot)]_\Lambda + [r(\cdot)]_\Lambda$ and $[v(\cdot)]_{\Lambda'} := \sum_{i=1}^{\ell} \chi_i \cdot [v_i(\cdot)]_{\Lambda'} + [r(\cdot)]_{\Lambda'}$. Then, $\mathcal{P}$ sends the polynomial $v(\cdot) = \sum_{i=1}^{\ell} \chi_i \cdot v_i(\cdot) + r(\cdot)$ to $\mathcal{V}$.

- CHECK PHASE:

  4. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[\epsilon] := [v(\Lambda)] - v(\Lambda)$ and $[\sigma] := [v(\Lambda')] - v(\Lambda')$. Then, both parties run $\mathsf{CheckZero}([\epsilon], [\sigma])$ to check that $\epsilon = 0$ and $\sigma = 0$. If the check fails, $\mathcal{V}$ aborts.

$\mathcal{V}$ is able to execute the above check phase after both keys Λ and Λ' were opened in which the local keys are obtained by $\mathcal{V}$. The linear combination of $v_1(\cdot), \ldots, v_\ell(\cdot)$ is masked by a random polynomial $r(\cdot)$, which assures the zero-knowledge property. Following the proof of Lemma 1, the soundness error is at most $\frac{2B}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$, as all IT-PACs are generated before the public coefficients $\chi_1, \ldots, \chi_\ell$ are determined and both polynomial keys are opened, and *a single* polynomial $v(\cdot)$ is opened for guaranteeing consistency.

Integrating AntMan with VOLE-based ZK proofs. While the recent VOLE-based ZK proofs [WYKW21, DIO21, BMRS21, YSWW21, WYX+21, BBMH+21, DILO22] have a communication *linear* to the circuit size for proving a single generic circuit, our ZK protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ shown in Figure 4 achieves the *sublinear* communication for proving SIMD circuits. We can seamlessly integrate $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ into a VOLE-based ZK protocol to obtain better communication efficiency for proving a single generic circuit. In particular, $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ is used to prove the sub-circuits that have the SIMD structure, and the VOLE-based ZK protocol is used to prove the remaining parts of the circuit. While protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ evaluates the circuit using IT-PACs, the VOLE-based ZK protocol adopts IT-MACs to perform the circuit evaluation. Therefore, we need to give efficient procedures that allow to convert between IT-PACs and IT-MACs. Protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ shown in Figure 4 has implied the conversion procedure from IT-MACs to IT-PACs. We only need to present the conversion procedure from IT-PACs to IT-MACs, which is totally similar. Specifically, let $[v_1(\cdot)], \ldots, [v_\ell(\cdot)]$ be the

IT-PACs generated by protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, and $y_{j,i} = v_j(\alpha_i)$ for $i \in [1, B], j \in [1, \ell]$ be the output values that need to be committed by IT-MACs. For each $i \in [1, B], j \in [1, \ell]$, $\mathcal{P}$ and $\mathcal{V}$ can generate an IT-MAC $[y_{j,i}]$ before the polynomial key Λ is opened. In particular, for $i \in [1, B], j \in [1, \ell]$, the parties call functionality $\mathcal{F}_{\mathsf{VOLE}}$ to generate a random IT-MAC $[r_{j,i}]$, and then $\mathcal{P}$ sends $d_{j,i} = y_{j,i} - r_{j,i}$ to $\mathcal{V}$ and both parties compute $[y_{j,i}] := [r_{j,i}] + d_{j,i}$. After Λ was opened, $\mathcal{P}$ and $\mathcal{V}$ execute the verification procedure described in Figure 4 to check the consistency between IT-PAC $[v_j(\cdot)]$ and IT-MACs $([y_{j,1}], \ldots, [y_{j,B}])$ for $j \in [1, \ell]$. That is, for each $j \in [1, \ell]$, $\mathcal{P}$ and $\mathcal{V}$ locally compute an IT-MAC $[y_j] := \sum_{i \in [1,B]} \delta_i(\Lambda) \cdot [y_{j,i}]$, and then run $\mathsf{CheckZero}([v_j(\Lambda)] - [y_j])$ to check $v_j(\Lambda) = y_j$.

5.2 Sublinear ZK Proof for Generic Circuits

Below, we show how to extend the ZK protocol on SIMD circuits (as shown in Figure 4) into a ZK protocol that proves satisfiability of a single generic circuit with *sublinear* communication. To do this, we first compile a generic circuit $\mathcal{C}$ into another equivalent circuit $\mathcal{C}'$ with $|\mathcal{C}'| = |\mathcal{C}| + O(B)$, where B is the number of wire values committed by a single IT-PAC. The new circuit $\mathcal{C}'$ needs to satisfy the following properties:

1. For each input $\boldsymbol{w}$, $\mathcal{C}(\boldsymbol{w}) = \mathcal{C}'(\boldsymbol{w})$.
2. The number of input gates, addition gates and multiplication gates is the multiple of B. There are at least B output gates.
3. Every B same-type gates are divided into a group, where the B related wire values in a group will be committed by an IT-PAC.

This is easy to be realized by adding "dummy" wires and gates following the approach [YW22], where the values on the "dummy" wires are set as 0. For completeness, we describe the circuit-transformation procedure $\mathsf{PrepCircuit}$ in Appendix D.

Then, we need to deal with the case that the input wires of B gates in a group are *not* corresponding to the output wires of B gates in any group of previous layers. In this case, the values on these input wires have *not* been committed by existing IT-PACs. Thus, we let the prover and verifier generate such an IT-PAC by running sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$. However, for a malicious prover, the values committed by the IT-PAC may be *inconsistent* with the values on the output wires of B gates from different groups in previous layers. Therefore, we give an efficient consistency check to detect such malicious behavior based on the $\mathsf{BatchCheck}$ procedure shown in Figure 2. Specifically, for each input IT-PAC $[\hat{g}(\cdot)]$, if the j-th wire value $\hat{g}(\alpha_j)$ comes from the i-th wire value $\hat{f}(\alpha_i)$ committed by an output IT-PAC $[\hat{f}(\cdot)]$, we need to check $\hat{f}(\alpha_i) = \hat{g}(\alpha_j)$. This corresponds to the wire that carries the value $\hat{g}(\alpha_j) = \hat{f}(\alpha_i)$ in the circuit. Following the previous work [GPS21, YW22], we refer to a tuple $([\hat{f}(\cdot)], [\hat{g}(\cdot)], i, j)$ as a wire tuple. In Section 2.4, we present the approach how to check the consistency of wire tuples in a batch. In the following, we directly give the concrete construction.

In detail, the sublinear ZK protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ for proving a single generic circuit can be constructed by extending the protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ on SIMD circuits (as described in Figure 4) in the following way.

- **Preprocess circuit.** $\mathcal{P}$ and $\mathcal{V}$ execute the $\mathsf{PrepCircuit}$ shown in Figure 8 of Appendix D to preprocess a generic circuit $\mathcal{C}$, and obtain an equivalent circuit $\mathcal{C}'$ with the same output. Let $\boldsymbol{w} \in \mathbb{F}^{mB}$ be the input of circuit $\mathcal{C}'$ that consists of an actual witness and "dummy" zero values.
- **Circuit evaluation.** The input preprocessing is the same as that of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, except for the following differences:
  - $\boldsymbol{w}$ is written as $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$ where $w_{1,j}, \ldots, w_{B,j}$ for each $j \in [1, m]$ are packed in a group and will be committed by an IT-PAC. Meanwhile, $\boldsymbol{w}$ will also be committed by mB IT-MACs. Note that $(w_{1,j}, \ldots, w_{B,j})$ for $j \in [1, m]$ corresponds to B input gates in the j-th group.

– If $w_{i,j}$ for $i \in [1, B], j \in [1, m]$ corresponds to a "dummy" zero value, the IT-MAC $[w_{i,j}] = [0]$ can be generated locally by $\mathcal{P}$ and $\mathcal{V}$. If $(w_{1,j}, \ldots, w_{B,j})$ for $j \in [1, m]$ corresponds to B "dummy" zero values, the IT-PAC $[u_j(\cdot)]$ with $u_j(\alpha_i) = w_{i,j} = 0$ for $i \in [1, B]$ can also be generated locally by both parties.

Following a predetermined topological order, $\mathcal{P}$ and $\mathcal{V}$ can evaluate the addition and multiplication gates as in protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, except for the following differences:

– If the input wires of B gates in a group are not corresponding to the output wires of B gates in any group of previous layers, then $\mathcal{P}$ computes a degree-$(B-1)$ polynomial $\hat{g}(\cdot)$ such that $\hat{g}(\alpha_i) = z_i$ for all $i \in [1, B]$, where $\{z_i\}_{i \in [1,B]}$ are the values on these input wires.

– Then, $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ shown in Figure 3 to generate an IT-PAC $[\hat{g}(\cdot)]$.

- **Consistency check of wire tuples.** Let $L(i, j)$ be a set of all wire tuples $\{([\hat{f}_h(\cdot)], [\hat{g}_h(\cdot)], i, j)\}$ such that $\hat{f}_h(\alpha_i) = \hat{g}_h(\alpha_j)$. For each $i, j \in [1, B]$, $\mathcal{P}$ and $\mathcal{V}$ check the consistency of the wire tuples in $L(i, j)$ as follows:

  1. Let ℓ be the size of $L(i, j)$, and $([\hat{f}_1(\cdot)], [\hat{g}_1(\cdot)], i, j), \ldots, ([\hat{f}_\ell(\cdot)], [\hat{g}_\ell(\cdot)], i, j)$ be the wire tuples in $L(i, j)$.
  2. The parties $\mathcal{P}$ and $\mathcal{V}$ execute the linear-combination phase of the $\mathsf{BatchCheck}_{(B-1),(B-1),1}$ procedure (as shown in Figure 2) on inputs $\{[\hat{f}_1(\cdot)], \ldots, [\hat{f}_\ell(\cdot)]\}$ and $\{[\hat{g}_1(\cdot)], \ldots, [\hat{g}_\ell(\cdot)]\}$ to check that $\hat{f}_h(\alpha_i) = \hat{g}_h(\alpha_j)$ for all $h \in [1, \ell]$ before the polynomial key Λ is opened.
  3. $\mathcal{P}$ and $\mathcal{V}$ execute the check phase of $\mathsf{BatchCheck}_{(B-1),(B-1),1}$ to complete the above check, where $\mathcal{V}$ can perform the check after the key Λ was opened and local keys on the IT-PACs were computed. If the check fails, $\mathcal{V}$ aborts.

A direct optimization for the consistency check as describe above is that $\mathcal{V}$ sends only one random seed to $\mathcal{P}$ for all B^2 executions of $\mathsf{BatchCheck}$, and then both parties can use the seed and a random oracle to generate the public coefficients for all $\mathsf{BatchCheck}$ executions. Note that all B^2 executions of $\mathsf{BatchCheck}$ can be run in parallel. The above consistency check can be executed in parallel with the correctness check of multiplication gates. Therefore, the ZK protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ has the same rounds as protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, and thus is constant-round. The communication complexity of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ is $O(|\mathcal{C}|/B + B^3)$, and becomes $O(|\mathcal{C}|^{3/4})$ sublinear to the circuit size if $B = |\mathcal{C}|^{1/4}$. The communication for addition gates is only from the possible IT-PAC generation when the values on the input wires of these gates are not committed by any existing IT-PAC, and thus depends on the circuit structure.

The security proof of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ is similar to that of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$, where the simulation of sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ for generating IT-PACs $\{[\hat{g}(\cdot)]\}$ is the same as that of $\Pi_{\mathsf{PAC}}^{(2B-2)}$ for generating IT-PACs $\{([h_j(\cdot)], [\widetilde{h}_j(\cdot)])\}$. Furthermore, the consistency-check procedure for wire tuples is easy to be simulated by invoking the simulator for the $\mathsf{BatchCheck}$ procedure involved in the proof of Theorem 1. Therefore, for the security proof of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$, we only need to analyze the soundness error of the consistency-check procedure on wire tuples, and the other part of the proof just follows the proof of Theorem 1. In particular, we have the following lemma.

Lemma 2. *For a malicious prover $\mathcal{P}$ and an honest verifier $\mathcal{V}$, if there exists an inconsistent wire tuple, $\mathcal{V}$ aborts in the execution of protocol $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ except with probability at most $\frac{2B}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.*

Based on Lemma 1 and Lemma 4 of Appendix C, the proof of the above lemma is straightforward. Particularly, if there exists some $h \in [1, \ell]$ such that $\hat{f}_h(\alpha_i) \neq \hat{g}_h(\alpha_j)$ for some $i, j \in [1, B]$, the $\mathsf{BatchCheck}$ procedure will abort except with probability $\frac{2B}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Combining the proof of Theorem 1 with Lemma 2, we can obtain the following corollary.

Corollary 1. *Protocol* $\Pi_{\mathsf{ZK}}^{\mathsf{generic}}$ *UC-realizes functionality* $\mathcal{F}_{\mathsf{ZK}}^{1}$ *on a single generic circuit in the* $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$*-hybrid model with soundness error at most* $\frac{4B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

6 Implementation and Benchmarking

6.1 Practical Optimizations

Faster polynomial interpolation and multiplication. For every multiplication gate, we have the input polynomials $f(\cdot), g(\cdot) \in \mathbb{F}[X]$ of degree $(B-1)$, and need to compute the polynomial $\widetilde{h}(\cdot) = f(\cdot) \cdot g(\cdot)$ and a degree-B polynomial $h(\cdot)$ such that $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for $i \in [1, B]$. To maximize the performance, we represent every degree-$(B-1)$ polynomial by the polynomial values evaluated at all B-th roots of unity in $\mathbb{F}$. This representation brings a lot of benefits: 1) we can directly use number theoretic transformation (NTT) to switch between the representation of polynomial values and the representation of polynomial coefficients; 2) the set of polynomial values representing $\widetilde{h}(\cdot)$ evaluated at all $(2B-1)$-th roots of unity contains the set of polynomial values representing $h(\cdot)$ evaluated at all B-th roots of unity. In this way, the polynomial-value representation of $h(\cdot)$ can be directly computed by B field multiplications. The computation of polynomial $\widetilde{h}(\cdot)$ only requires additionally applying inverse NTT and NTT to switch between different representations of the polynomials $f(\cdot)$ and $g(\cdot)$, followed by B field multiplications.

Better utilization of plaintext slots in AHE. The AHE schemes based on ring-LWE such as [BGV12, Bra12, FV12] support the efficient computation over multiple independent slots, and allow to manipulate multiple plaintexts at once using SIMD operations. Taking advantage of multiple plaintext slots is the key to obtain high efficiency in the AHE schemes. In our IT-PAC generation protocol with simplified notation, the verifier has a vector $\boldsymbol{a} \in \mathbb{F}^k$ (where $k = 2B-2$ when applying this protocol into our ZK protocol), and the prover has a matrix $\mathbf{M} \in \mathbb{F}^{\ell \times k}$ and is desired to get a vector $\mathbf{M} \cdot \boldsymbol{a} \in \mathbb{F}^{\ell}$. Let S be the number of slots (which is 8192 in our implementation). One way to accomplish this task is to have the prover obtaining the ciphertexts on plaintext vectors $\hat{\boldsymbol{a}}_i = (a_i, \ldots, a_i) \in \mathbb{F}^S$ for all $i \in [1, k]$. However, the communication cost in this case is $O(BS)$. To reduce the communication, we adopt the "diagonal method" in prior work [HS14]. In particular, the verifier fills S slots with copies of $\boldsymbol{a}$ and sends a rotation key to the prover. Then, the prover uses the key to rotate the ciphertexts on vector $\boldsymbol{a}$ and obtains the encryption of a cyclic matrix $\mathbf{A}$ defined by $\boldsymbol{a}$. The rest of the computation remains the same except for homomorphically operating on the ciphertext of matrix $\mathbf{A}$.

6.2 Parameters and Testbed Configuration

We implemented our ZK protocol AntMan for proving SIMD circuits and demonstrate its performance by a series of experiments. We use two Amazon EC2 `m5.8xlarge` instances located in the same region to act as the prover and verifier. We use 4 threads and throttle the bandwidth to 1 Gbps unless otherwise specified. Our protocol AntMan adopts the BGV homomorphic encryption with a single level [BGV12] to instantiate the AHE scheme, and uses the LPN-based VOLE protocol [WYKW21] to realize functionality $\mathcal{F}_{\mathsf{VOLE}}$. In our ZK protocol, addition gates are free and the performance is dominated by evaluating and checking multiplication gates.

We use a finite field $\mathbb{F}$ defined by a prime $p = 2^{59} - 2^{28} + 1$, which guarantees that $2N$-th roots of unity exist for any $N = 2^k$ and $k < 28$. We set the computational security parameter $\lambda = 128$ and statistical security parameter $\rho = \log|\mathbb{F}| - \log(2B+3) > 40$, where B is the number of executions of a circuit and chosen from 16 to 2048. We often refer to B as a batch size as well. In our implementation, we consider a setup phase to generate the AHE ciphertexts on the powers of a polynomial key Λ and rotate these ciphertexts as described in Section 6.1. The setup cost is independent of the circuit size and only

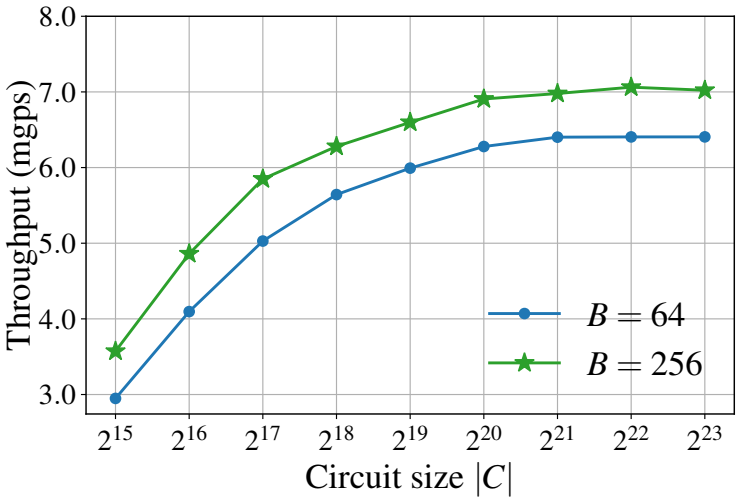

Figure 5: **The throughput of our ZK protocol with respect to the circuit size.** The performance is measured as million MULT gates per second (mgps), and is shown for proving $(B, |\mathcal{C}|)$-SIMD circuits where $B \in \{64, 256\}$ and $|\mathcal{C}| \in \{2^{15}, \ldots, 2^{23}\}$.

depends on the parameter B. Our experimental results involve the running time to execute the setup phase. For performance evaluation, we compare AntMan with the state-of-the-art VOLE-based ZK implementation QuickSilver [YSWW21].

6.3 Performance for Circuit Size and Batch Size

The performance of our ZK protocol AntMan does not depend on the circuit structure, and is only related to the circuit size $|\mathcal{C}|$ and the batch size B. Therefore, we evaluate the performance using random circuits.

Performance vs. circuit size. In Figure 5, we analyze how the throughput of AntMan changes over different circuit sizes. We study the performance on proving $(B, |\mathcal{C}|)$-SIMD circuits with $B \in \{64, 256\}$ and that $|\mathcal{C}|$ is chosen from 2^{15} to 2^{23}. From this figure, we observe that the overall performance of our ZK protocol increases as the circuit size becomes larger. This is because the setup cost of $O(B)$ is amortized over the circuit, and thus a larger circuit leads to a smaller amortized cost. The highest throughput is reached when the circuit size is larger than 2^{20}, after that the throughput is stable. Indeed, the throughput in this case is roughly the throughput without counting the setup cost.

Performance vs. batch size. Now we fix the circuit size $|\mathcal{C}|$ to be 2^{20} and study how different batch sizes (i.e., B) impacts the performance of our ZK protocol. In Table 1, we report the running time and communication cost of AntMan with batch sizes B changed from 16 to 2048.

Because our AHE parameters support up to 8192 slots, the communication cost for the setup phase is 5.1 MB for all choices of B. This consists of the ciphertexts encrypting the powers of a polynomial key (0.4 MB) as well as rotation keys (4.7 MB). When B is greater than 2048, more ciphertexts need to be sent but it does not impact the overall setup communication by much. The setup computation cost is mostly brought from rotation operations.

As for the performance without the setup cost, we observe that we are able to achieve better than one-field-element per multiplication gate once $B \geq 16$. The amortized running time per multiplication gate of AntMan decreases as the number of circuit executions increases, and stays stable when the batch size is greater than 128. However, it is still 30% slower than QuickSilver under the 1Gbps network bandwidth. In terms of communication cost, the communication per multiplication gate for AntMan is reduced by half every time B doubles. When $B = 2048$, the communication cost per multiplication gate of our ZK

B	Running time		Communication
	Setup (ms)	Per gate (μs)	Per gate (field elements)
16	138	0.241	0.82
32	179	0.181	0.41
64	263	0.156	0.205
128	430	0.144	0.1
256	761	0.142	0.051
512	1430	0.142	0.0256
1024	2743	0.141	0.0127
2048	5445	0.141	0.0064
QuickSilver	0	0.107	1

Table 1: **The communication and running time of our ZK protocol.** The running time is benchmarked with 4 threads and 1 Gbps bandwidth. The circuit size $|\mathcal{C}| = 2^{20}$ for the whole table. The setup communication cost for all B in the table is 5.1 MB.

Protocol-thread	Network Bandwidth				
	10 Mbps	50 Mbps	100 Mbps	500 Mbps	1 Gbps
AntMan-1	1.79	2.00	2.05	2.08	2.09
AntMan-2	3.02	3.78	3.91	3.99	4.26
AntMan-4	4.86	6.88	6.69	6.99	7.01
AntMan-8	6.30	10.06	10.79	11.67	11.64
AntMan-16	7.56	14.07	15.86	17.51	17.74
QuickSilver-∞	0.17	0.85	1.7	8.47	16.95

Table 2: **The performance of our ZK protocol subject to the bandwidth and the number of threads.** The benchmark results are the number of million MULT gates per second (mgps). QuickSilver-∞ refers to the theoretical performance of QuickSilver with infinity computational power and thus the running time is solely determined by the communication.

protocol is about 0.0064 field elements, which improves the state-of-the-art ZK implementation QuickSilver by a factor of more than $150\times$. This means that when the network bandwidth is low, our ZK protocol is significantly better.

6.4 Performance for Threads and Bandwidthes

In Table 2, we show how multi-threading and different bandwidthes affect the performance of the ZK protocol AntMan. In particular, we fix the batch size $B = 1024$ and the circuit size $|\mathcal{C}| = 2^{21}$. Our ZK protocol is computationally heavy, and thus the multi-threading boosts the efficiency significantly. In a high bandwidth setting (1 Gbps), the throughput of AntMan increases from 2.09 to 17.74 mgps when the number of threads increases from 1 to 16. On the other hand, AntMan is highly communication-efficient, and thus performs well in low bandwidth settings. The throughput of AntMan does not significantly benefit from the increase of network bandwidthes when it is higher than 50 Mbps. It is due to the fact that the online communication cost is reduced by a factor of B asymptotically, compared to the ZK protocol QuickSilver [YSWW21].

Operations	Running Time (ns)
BGV homomorphic evaluation	84.53
BGV decryption	1.5
Polynomial multiplication	24.84
Hashing (SHA-256)	0.98
Check of MULT gates	7.74
Others	22.47
Total	142.06

Table 3: **The microbenchmark of our ZK protocol.** The running time is the amortized time for proving one multiplication gate. The communication time is involved in the "Others" part.

6.5 Microbenchmarking

To understand the slowest part of our ZK protocol, we conduct a microbenchmark of the protocol execution in Table 3. In this experiment, we synthesize a random circuit with $|\mathcal{C}| = 2^{22}$ multiplication gates, and prove $B = 256$ executions of a circuit (and thus 2^{30} multiplication gates in total). Other choices of B and $|\mathcal{C}|$ have a similar proportion for microbenchmark. In this table, 5 major expensive operations are counted. The computation cost mainly comes from the homomorphic operations of AHE and NTT operations. The homomorphic evaluation of BGV ciphertexts dominates the whole computation cost, and takes around 60% of the total running time. The polynomial multiplication (involving the operations of NTT and inverse NTT) takes about 18% of the total running time. Note that the BGV encryption and rotation operations are performed only once at the setup phase, and the setup cost can be amortized to negligible when proving a large circuit.

7 Conclusion

This paper made the first step toward sublinear communication for VOLE-based ZK proofs, and achieved high concrete efficiency for SIMD circuits. The result shows a new direction to design efficient ZK proofs, and many open questions deserve further study: 1) design a protocol of generating IT-PACs with similar efficiency to realize some weak functionality that can be used in ZK protocols, and make the design of ZK protocols modular; 2) improve the asymptotic and concrete efficiencies of our ZK protocol with communication of $O(|\mathcal{C}|^{3/4})$ for proving a single execution of an arbitrary circuit; 3) improve the computational complexity of the protocol to something linear to the circuit size. 4) develop automatic tools to adopt different techniques for proving different parts of the statement in zero-knowledge.

Acknowledgements

Work of Kang Yang is supported by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019, 62022018). Work of Xiao Wang is supported in part by DARPA under Contract No. HR001120C0087, NSF award #2016240, and research awards from Facebook and Google. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[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.

[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.

[BdMW16] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 62–89, Santa Barbara, CA, USA, August 14–18, 2016. 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.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. ACM.

[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.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[dCJV21] Leo de Castro, Chiraag Juvekar, and Vinod Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography – WAHC'21*, pages 29–41. 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 *ACM CCS 2022*. ACM Press, November 15–19, 2022.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography*, 2021.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 167–187, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[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.

[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.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 275–304, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[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.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[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.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[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.

[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.

[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.

[YW22] Kang Yang and Xiao Wang. Non-interactive zero-knowledge proofs to multiple verifiers. Cryptology ePrint Archive, Report 2022/063, 2022. https://eprint.iacr.org/2022/063.

[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.

A More Preliminaries

A.1 Security Model and Functionalities

We use the universal composability (UC) framework [Can01] to prove security of our ZK protocol in the presence of a malicious, static adversary. We say that a protocol Π *UC-realizes* an ideal functionality $\mathcal{F}$ if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for

Functionality $\mathcal{F}_{\mathsf{Com}}$

This functionality runs with two parties $\mathsf{P_A}$ and $\mathsf{P_B}$ as follows:

Commit: Upon receiving $(\mathsf{Commit}, x, \tau_x)$ from $\mathsf{P_A}$, store (x, τ_x) and output τ_x to $\mathsf{P_B}$, where τ_x is a handle that is used to identify the message x.

Open: Upon receiving (Open, τ_x) from $\mathsf{P_A}$, if a tuple (x, τ_x) was previously stored, output (x, τ_x) to $\mathsf{P_B}$.

Figure 6: Two-party commitment functionality.

Functionality $\mathcal{F}_{\mathsf{VOLE}}$

This functionality works over a field $\mathbb{F}$, and interacts with two parties $\mathcal{P}$ and $\mathcal{V}$ as well as an adversary as follows:

Initialize: Upon receiving (init) from $\mathcal{P}$ and $\mathcal{V}$, if $\mathcal{V}$ is honest, then sample $\Delta \leftarrow \mathbb{F}$, else receive $\Delta \in \mathbb{F}$ from the adversary. Store Δ and ignore all subsequent (init) commands.

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

- If $\mathcal{V}$ is honest, sample $\boldsymbol{v} \leftarrow \mathbb{F}^n$. Otherwise, receive $\boldsymbol{v} \in \mathbb{F}^n$ from the adversary.
- If $\mathcal{P}$ is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}^n$ and compute $\boldsymbol{w} := \boldsymbol{v} + \boldsymbol{u} \cdot \Delta \in \mathbb{F}^n$. Otherwise, receive $\boldsymbol{u} \in \mathbb{F}^n$ and $\boldsymbol{w} \in \mathbb{F}^n$ from the adversary, and then recompute $\boldsymbol{v} := \boldsymbol{w} - \boldsymbol{u} \cdot \Delta \in \mathbb{F}^n$.
- Output $(\boldsymbol{u}, \boldsymbol{w})$ to $\mathcal{P}$ and $\boldsymbol{v}$ to $\mathcal{V}$.

Figure 7: Functionality for VOLE correlations.

every PPT environment $\mathcal{Z}$, the output distribution of $\mathcal{Z}$ in the *real-world* execution where the parties interact with $\mathcal{A}$ and execute Π is computationally indistinguishable from the one in the *ideal-world* execution where the parties interact with $\mathcal{S}$ and $\mathcal{F}$.

Our protocol will invoke a commitment functionality $\mathcal{F}_{\mathsf{Com}}$ that is reviewed in Figure 6 and can be UC-realized by defining $\mathsf{H}(m, r)$ as a commitment on a message m, where H is a random oracle and r is a randomness.

A.2 Generation of IT-MACs from VOLE

The IT-MAC commitments on random messages can be efficiently generated by the recent LPN-based VOLE protocols with *sublinear* communication in the malicious setting [BCGI18, YWL+20, WYKW21]. We model random VOLE in the standard functionality shown in Figure 7. Two parties $\mathcal{P}$ and $\mathcal{V}$ can transform the IT-MAC commitments on random messages into that on chosen messages in the standard way by sending an additional element for each commitment (see our ZK protocols shown in Section 5 or [WYKW21, DIO21, BMRS21, YSWW21, WYX+21, BBMH+21] for details).

A.3 More Details for AHE Schemes

An additively homomorphic encryption (AHE) scheme $\mathsf{AHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, this algorithm outputs a set of public parameters par.
- $\mathsf{KeyGen}(\mathsf{par})$: On input the set of public parameters par, this algorithm outputs a secret key sk.
- $\mathsf{Enc}(\mathsf{sk}, m)$: On input the secret key sk and a message $m \in \mathbb{F}$ where par is assumed to be an implicit input, this algorithm outputs a ciphertext c.

- $\mathsf{Dec}(\mathsf{sk}, c)$: On input the secret key sk and a ciphertext c where par is an implicit input, this algorithm outputs a message m.

- *Additively homomorphic operations:* Given the set of public parameters par and a ciphertext $c = \mathsf{Enc}(\mathsf{sk}, x)$, a party $\mathcal{P}$ can compute a ciphertext $c' = y \cdot c - b$ without knowing secret key sk, where $y, b \in \mathbb{F}$ are known by $\mathcal{P}$. The party $\mathcal{V}$ owning sk can decrypt c' to obtain $a = \mathsf{Dec}(\mathsf{sk}, c') = x \cdot y - b \in \mathbb{F}$. Here, we require that the AHE scheme achieves *circuit privacy*, which guarantees that c' does not leak any information about y and b even to the owner of sk. We refer the reader to [IP07, BdMW16, dCJV21] for the formal definition of circuit privacy. Furthermore, given par and two ciphertexts $c_1 = \mathsf{Enc}(\mathsf{sk}, m_1)$ and $c_2 = \mathsf{Enc}(\mathsf{sk}, m_2)$, one can compute $c_1 + c_2 = \mathsf{Enc}(\mathsf{sk}, m_1 + m_2)$ without knowing sk.

Informally, we say that the scheme AHE is correct, if the ciphertext c on a message m, which is obtained by $\mathsf{Enc}(\mathsf{sk}, m)$ or the additively homomorphic operations of some ciphertexts generated by $\mathsf{Enc}(\mathsf{sk}, \cdot)$, can be decrypted to m via $\mathsf{Dec}(\mathsf{sk}, c)$ with probability $1 - \mathsf{negl}(\lambda)$.

Our implementation adopts the BGV homomorphic encryption with a single level [BGV12] to instantiate the scheme AHE. In the BGV-AHE scheme, while the ciphertexts are defined over a ring $R_q = R/qR$, the plaintexts are lied in a ring $R_p = R/pR$, where $R = \mathbb{Z}[X]/(X^N + 1)$ is a polynomial ring with integer coefficients modulo $X^N + 1$, N is a power-of-two integer and $p, q \in \mathbb{N}$ are co-prime. Using the packing technique, we can pack multiple values in a single ciphertext and support parallel computation in the single instruction multiple data (SIMD) way. In particular, we can set $\mathbb{F} = \mathbb{F}_p$ for a prime $p = 1 \pmod{2N}$ and consider an element $a \in R_p$ as a vector in $\mathbb{F}^N$. Although the BGV-AHE scheme supports N plaintext slots (i.e., allowing to encrypt N messages from $\mathbb{F}$ in a single ciphertext), we still use the notation $\mathsf{Enc}(\mathsf{sk}, m)$ to denote the encryption of a single message m for the sake of simplicity. The circuit privacy of the BGV-AHE scheme is achieved using the noise-flooding technique [Gen09]. We refer the reader to [BGV12, KPR18] for details of the BGV-AHE scheme. Besides, we can also use the BFV homomorphic encryption with a single level [Bra12, FV12] to instantiate the AHE scheme, where the circuit privacy can be guaranteed in the more efficient way using the recent rounding technique [dCJV21].

B Proof of Lemma 1

In this section, we prove Lemma 1 that is restated as follows.

Lemma 3 (Lemma 1, restated)**.** *Let $\mathcal{P}$ be a malicious party who interacts with an honest verifier $\mathcal{V}$ during the execution of* $\mathsf{BatchCheck}$*. Let H be a random oracle. If there exists some $i \in [1, t], j \in [1, \ell]$ such that $f_j(\alpha_i) \neq g_j(\beta_i)$, then the probability that $\mathcal{V}$ accepts at the end of* $\mathsf{BatchCheck}$ *is at most* $\frac{\max\{k,m\}+2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Proof. Let $f(\cdot)$ and $g(\cdot)$ be the polynomials committed in IT-PACs $[f(\cdot)]$ and $[g(\cdot)]$ respectively, where $[f(\cdot)] = \sum_{j\in[1,\ell]} \chi_j \cdot [f_j(\cdot)] + [r(\cdot)]$ and $[g(\cdot)] = \sum_{j\in[1,\ell]} \chi_j \cdot [g_j(\cdot)] + [s(\cdot)]$. This means that $f(\cdot) = \sum_{j\in[1,\ell]} \chi_j \cdot f_j(\cdot) + r(\cdot) \in \mathbb{F}[X]$ and $g(\cdot) = \sum_{j\in[1,\ell]} \chi_j \cdot g_j(\cdot) + s(\cdot) \in \mathbb{F}[X]$. Let $f'(\cdot)$ and $g'(\cdot)$ be two polynomials opened by the malicious $\mathcal{P}$ in the step 3 of the $\mathsf{BatchCheck}$ procedure.

Let E_1 be the event that $f'(\Lambda) \neq f(\Lambda)$ or $g'(\Lambda) \neq g(\Lambda)$ but $\mathcal{V}$ accepts in the $\mathsf{CheckZero}$ procedure. According to the security of $\mathsf{CheckZero}$, we have that $\Pr[E_1] \leq \frac{1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Let E_2 be the event that $f'(\Lambda) = f(\Lambda)$ and $g'(\Lambda) = g(\Lambda)$ but $f'(\cdot) \neq f(\cdot)$ or $g'(\cdot) \neq g(\cdot)$. Note that the polynomials $f'(\cdot)$ and $g'(\cdot)$ are opened by malicious $\mathcal{P}$ before the polynomial key Λ known by $\mathcal{P}$. At least one of two polynomials $f'(\cdot) - f(\cdot)$ and $g'(\cdot) - g(\cdot)$ is non-zero, and the degrees of $f'(\cdot) - f(\cdot)$ and $g'(\cdot) - g(\cdot)$ are bounded by k and m respectively. Therefore, for a uniform $\Lambda \in \mathbb{F}$, the probability that $f'(\Lambda) - f(\Lambda) = 0$ and $g'(\Lambda) - g(\Lambda) = 0$ is at most $\frac{\max\{k,m\}}{|\mathbb{F}|}$. In other words, $\Pr[E_2] \leq \frac{\max\{k,m\}}{|\mathbb{F}|}$.

Let E_3 be the event that there exists some $i \in [1,t], j \in [1,\ell]$ such that $f_j(\alpha_i) \neq g_j(\beta_i)$ but $\mathcal{V}$ accepts at the end of BatchCheck. We assume that both E_1 and E_2 do not happen. Thus, we have that $f'(\cdot) = \sum_{j\in[1,\ell]} \chi_j \cdot f_j(\cdot) + r(\cdot)$ or $g'(\cdot) = g(\cdot) = \sum_{j\in[1,\ell]} \chi_j \cdot g_j(\cdot) + s(\cdot)$. Since $\mathcal{V}$ accepts, we obtain that $f'(\alpha_i) = g'(\alpha_i)$ for $i \in [1,t]$. Therefore, $\sum_{j\in[1,\ell]} \chi_j \cdot (f_j(\alpha_i) - g_j(\alpha_i)) + (r(\alpha_i) - s(\alpha_i)) = 0$ for each $i \in [1,t]$. If the malicious $\mathcal{P}$ does not make a query seed to random oracle H before receiving seed, then $\chi_1, \ldots, \chi_\ell$ are determined after the polynomials $\{f_j(\cdot), g_j(\cdot)\}_{j=1}^{\ell}$, $r(\cdot)$ and $s(\cdot)$ have already been defined, and thus are independent of these polynomials. In this case we have that $\Pr[E_3 \,|\, \neg(E_1 \cup E_2)] \leq \frac{1}{|\mathbb{F}|}$. The probability that $\mathcal{P}$ queried seed to random oracle H before $\mathcal{V}$ sends seed to $\mathcal{P}$ is at most $\frac{q}{2^\lambda}$, where q is the number of queries to random oracle H. Together, we obtain that $\Pr[E_3 \,|\, \neg(E_1 \cup E_2)] \leq \frac{1}{|\mathbb{F}|} + \frac{q}{2^\lambda} = \frac{1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Overall, we have the following:

$$\begin{aligned}
\Pr[E_3] &= \Pr[E_3 \,|\, E_1 \cup E_2] \cdot \Pr[E_1 \cup E_2] + \Pr[E_3 \,|\, \neg(E_1 \cup E_2)] \cdot \Pr[\neg(E_1 \cup E_2)] \\
&\leq \Pr[E_1 \cup E_2] + \Pr[E_3 \,|\, \neg(E_1 \cup E_2)] \\
&\leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3 \,|\, \neg(E_1 \cup E_2)] \\
&\leq \frac{\max\{k,m\}+2}{|\mathbb{F}|} + \mathsf{negl}(\lambda),
\end{aligned}$$

which completes the proof. □

C Proof of Theorem 1

In this section, we give the formal proof of Theorem 1. First of all, we extend Lemma 1 from the information-theoretic setting to the computational setting. While Lemma 1 assumes that the polynomial key Λ is information-theoretically secure, we prove that the result claimed in the lemma still holds, even if the malicious $\mathcal{P}$ is given the AHE ciphertexts $\mathsf{Enc}(\Lambda), \ldots, \mathsf{Enc}(\Lambda^h)$ with $h = \max\{k,m\}$ and the input IT-PACs of the BatchCheck procedure are generated by executing the protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$. In particular, we have the following lemma:

Lemma 4. *Let the IT-PACs used in the* BatchCheck *procedure be generated by two parties $\mathcal{P}$ and $\mathcal{V}$ by running protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$ shown in Figure 3 where $h = \max\{k,m\}$. Let H be a random oracle and G be a pseudorandom generator. If the* AHE *scheme is CPA secure and satisfies the degree-restriction property, then the probability that there exists some $i \in [1,t], j \in [1,\ell]$ such that $f_j(\alpha_i) \neq g_j(\beta_i)$ but an honest verifier $\mathcal{V}$ accepts at the end of* BatchCheck *is bounded by $\frac{\max\{k,m\}+2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.*

Proof. We consider that $\mathcal{P}$ is corrupted by a PPT malicious adversary $\mathcal{A}$ and $\mathcal{V}$ is honest. In this proof, we will reuse the notation used in the proof of Lemma 1. For example, the events E_1, E_2, E_3 and the polynomials $f'(\cdot), g'(\cdot)$ opened by $\mathcal{P}$ during the BatchCheck procedure. In the protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$, if $\mathcal{A}$ sends a ciphertext evaluated on a polynomial of degree $h' > h$ to the honest verifier $\mathcal{V}$, then it is straightforward to construct an algorithm who breaks the degree-restriction property of the AHE scheme. In particular, the algorithm obtains the ciphertexts $\langle\Lambda\rangle, \ldots, \langle\Lambda^h\rangle$ from the degree-restriction game, and then forwards them to $\mathcal{A}$. Then, the algorithm randomly chooses one from the ciphertexts sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$ as its ciphertext sent to the degree-restriction game. Therefore, the probability that $\mathcal{A}$ sends a ciphertext evaluated on a polynomial of degree $h' > h$ is bound by the advantage of the adversary to break the degree-restriction property, and thus is negligible in λ. In the following proof, we always assume that the degree of the polynomials committed in the IT-PACs generated by protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$ is bounded by $h = \max\{k,m\}$.

We consider a hybrid-check procedure denoted by HybridCheck, which is the same as BatchCheck except that the verification of $\mathsf{CheckZero}([\mu],[\nu])$ is modified as follows:

- Let M_μ and M_ν be the MACs involved in the IT-MACs $[\mu]$ and $[\nu]$ respectively, where $[\mu] = [f(\Lambda)] - f'(\Lambda)$ and $[\nu] = [g(\Lambda)] - g'(\Lambda)$. Use M_μ and M_ν to check the correctness of the hash value sent by $\mathcal{P}$.
- Check that $z_1 = f'(\Lambda)$ and $z_2 = g'(\Lambda)$, where z_1, z_2 are the values committed in the IT-MACs $[f(\Lambda)]$ and $[g(\Lambda)]$.

It is easy to see that the output of $\mathsf{HybridCheck}$ is identical to that of $\mathsf{CheckZero}$, unless event E_1 occurs with probability at most $\frac{1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Note that global key $\Delta \in \mathbb{F}$ is uniformly random in the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model. Below, we assume that E_1 does not occur.

Based on the $\mathsf{HybridCheck}$ procedure, we construct the following hybrid protocol that is executed between adversary $\mathcal{A}$ and a PPT simulator $\mathcal{S}_{\mathsf{hybrid}}$:

1. On behalf of an honest verifier, $\mathcal{S}_{\mathsf{hybrid}}$ simulates the phase to create and encrypt a uniform polynomial key Λ following the protocol description, except that the secret key sk and randomness used in the AHE ciphertexts are now sampled at random rather than being generated from a random seed. The resulting ciphertexts $\langle \Lambda \rangle, \ldots, \langle \Lambda^h \rangle$ are sent to $\mathcal{A}$.
2. $\mathcal{S}_{\mathsf{hybrid}}$ emulates functionality $\mathcal{F}_{\mathsf{VOLE}}$ by recording the vectors $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{F}^{2\ell+2}$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{VOLE}}$, where there are $2\ell + 2$ IT-PACs that need to be generated. Then, $\mathcal{S}_{\mathsf{hybrid}}$ sets $\mathsf{M}_i = w_i$ for $i \in [1, 2\ell + 2]$ as the MACs in the IT-PACs $\{([f_j(\cdot)], [g_j(\cdot)])\}_{j\in[1,\ell]}$ and $(r[\cdot], s[\cdot])$.
3. $\mathcal{S}_{\mathsf{hybrid}}$ emulates functionality $\mathcal{F}_{\mathsf{Com}}$ by receiving the ciphertexts $\langle b_1 \rangle, \ldots, \langle b_{2\ell+2} \rangle$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$ and sending a handle τ_2 to $\mathcal{A}$. Then, $\mathcal{S}_{\mathsf{hybrid}}$ obtains b_i by running $\mathsf{Dec}(\mathsf{sk}, \langle b_i \rangle)$ for $i \in [1, 2\ell + 2]$. Simulator $\mathcal{S}_{\mathsf{hybrid}}$ computes $y_i := b_i + u_i$ for all $i \in [1, 2\ell + 2]$ as the values $\{f_j(\Lambda), g_j(\Lambda)\}_{j\in[1,\ell]}$, $r(\Lambda)$ and $s(\Lambda)$, where the underlying polynomials are unknown for $\mathcal{S}_{\mathsf{hybrid}}$.
4. Using (y_i, M_i) for all $i \in [1, 2\ell + 2]$, $\mathcal{S}_{\mathsf{hybrid}}$ runs the linear-combination phase of the $\mathsf{HybridCheck}$ procedure with $\mathcal{A}$, and then obtains the values $\mathsf{M}_\mu, \mathsf{M}_\nu, z_1, z_2 \in \mathbb{F}$ and the opened polynomials $f'(\cdot), g'(\cdot)$. Then, $\mathcal{S}_{\mathsf{hybrid}}$ checks that $f'(\alpha_i) = g'(\beta_i)$ for $i \in [1, t]$ and aborts if the check fails.
5. $\mathcal{S}_{\mathsf{hybrid}}$ sends Λ to $\mathcal{A}$, and emulates $\mathcal{F}_{\mathsf{Com}}$ by receiving a handle τ_2 from $\mathcal{A}$.
6. Using the key Λ, $(\mathsf{M}_\mu, \mathsf{M}_\nu, z_1, z_2)$ and $(f'(\cdot), g'(\cdot))$, $\mathcal{S}_{\mathsf{hybrid}}$ runs the check phase of the $\mathsf{HybridCheck}$ procedure with $\mathcal{A}$.

For the case of a malicious prover $\mathcal{P}$, we only need to consider the soundness. In this case, it is unnecessary to commit and open a random seed that is used to generate the secret key and randomness, as the verifier is always honest. Together with that the output of G is pseudorandom, the above hybrid protocol is computationally indistinguishable from the protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$ combining with $\mathsf{BatchCheck}$, when only considering the soundness.

In the following, we assume that event E_2 happens in the above hybrid protocol, meaning that $z_1 = f'(\Lambda)$ and $z_2 = g'(\Lambda)$ but $f'(\cdot) \neq f(\cdot)$ or $g'(\cdot) \neq g(\cdot)$ always hold, where z_1 and z_2 are computed by $\mathcal{S}_{\mathsf{hybrid}}$ as described above. Specifically, for a PPT adversary $\mathcal{A}$ who makes E_2 happen, we construct a PPT algorithm $\mathcal{B}$ who breaks the CPA security of the AHE scheme. Algorithm $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{B}$ samples two random values $\Lambda, \Lambda^* \leftarrow \mathbb{F}$, and then sends h message pairs $(\Lambda^i, (\Lambda^*)^i)$ for $i \in [1, h]$ to the CPA game. Then, $\mathcal{B}$ obtains h challenge ciphertexts $c_1, \ldots, c_h$ from the CPA game, where c_i is the encryption of either Λ^i or $(\Lambda^*)^i$ for $i \in [1, h]$. Algorithm $\mathcal{B}$ sends $c_1, \ldots, c_h$ to $\mathcal{A}$.
2. $\mathcal{B}$ emulates functionality $\mathcal{F}_{\mathsf{VOLE}}$ by recording the vectors $\boldsymbol{u}, \boldsymbol{w}$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{VOLE}}$, and then sets $\mathsf{M}_i = w_i$ for $i \in [1, 2\ell + 2]$.

3. $\mathcal{B}$ emulates $\mathcal{F}_{\mathsf{Com}}$ by receiving the ciphertexts $\langle b_1\rangle, \ldots, \langle b_{2\ell+2}\rangle$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$ and sending a handle τ_2 to $\mathcal{A}$.

4. Using $\{\mathsf{M}_i\}_{i\in[1,2\ell+2]}$, $\mathcal{B}$ runs the linear-combination phase of the HybridCheck procedure with $\mathcal{A}$, and then obtains the values $\mathsf{M}_\mu, \mathsf{M}_\nu \in \mathbb{F}$ and the opened polynomials $f'(\cdot), g'(\cdot)$. Then, $\mathcal{B}$ checks that $f'(\alpha_i) = g'(\beta_i)$ for $i \in [1, t]$ and aborts if the check fails.

5. Using Λ, $(\mathsf{M}_\mu, \mathsf{M}_\nu)$ and $(f'(\cdot), g'(\cdot))$, $\mathcal{B}$ runs the check phase of the HybridCheck procedure with $\mathcal{A}$. In particular, $\mathcal{B}$ always uses Λ to check that $z_1 = f'(\Lambda)$ and $z_2 = g'(\Lambda)$, ignoring which key (Λ or Λ^*) is used to compute $\{c_i\}_{i\in[1,h]}$. Actually, this check is *implicit* and assumed to be passed as E_2 occurs. Additionally, $\mathcal{B}$ uses the MACs M_μ and M_ν to check the correctness of the hash value sent by $\mathcal{A}$ in the CheckZero procedure.

In the case that event E_2 occurs, $\mathcal{B}$ behaves just like as $\mathcal{S}_{\mathsf{hybrid}}$, except for the encryption of a polynomial key. If the challenge ciphertexts $c_1, \ldots, c_h$ are the encryption of the powers of Λ, then the protocol execution simulated by $\mathcal{B}$ is the same as the above hybrid protocol. Otherwise, the key Λ used in the HybridCheck procedure is independent of the adversary's view, and thus is information-theoretically secure. From the proof of Lemma 1, we have that the event E_2 occurs with probability at most $\frac{\max\{k,m\}}{|\mathbb{F}|}$ in this case. Therefore, the successful probability of adversary $\mathcal{A}$ in the above hybrid protocol is bounded by $\frac{\max\{k,m\}}{|\mathbb{F}|} + \epsilon_{\mathsf{cpa}}$, where ϵ_{cpa} is the advantage of a PPT adversary to break the CPA security of the AHE scheme. Overall, the probability, that event E_2 occurs in the BatchCheck procedure for the IT-PACs generated by protocol $\Pi^h_{\mathsf{IT\text{-}PAC}}$, is at most $\frac{\max\{k,m\}}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Given the probabilities that the events E_1 and E_2 happen, we can easily obtain the probability that event E_3 occurs following the proof of Lemma 1. In particular, we have that $\Pr[E_3] \leq \frac{\max\{k,m\}+2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$, which completes the proof. □

Theorem 2 (Theorem 1, restated)**.** *If the* AHE *scheme satisfies the CPA security, degree-restriction and circuit privacy along with* G *is a pseudorandom generator, protocol* $\Pi^{\mathsf{SIMD}}_{\mathsf{ZK}}$ *shown in Figure 4 UC-realizes functionality* $\mathcal{F}^B_{\mathsf{ZK}}$ *on* $(B, |\mathcal{C}|)$*-SIMD circuits in the* $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$*-hybrid model and the random oracle model with soundness error at most* $\frac{2B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Proof. We first consider the case of a malicious prover (i.e., soundness) and then consider the case of a malicious verifier (i.e., zero knowledge). In each case, we construct a PPT simulator $\mathcal{S}_{\mathsf{ZK}}$ given access to functionality $\mathcal{F}^B_{\mathsf{ZK}}$, and running a PPT adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}_{\mathsf{VOLE}}$ and $\mathcal{F}_{\mathsf{Com}}$ for $\mathcal{A}$. For each case, we show that no PPT environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution. We always implicitly assume that $\mathcal{S}_{\mathsf{ZK}}$ passes all communication between adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Let $\mathcal{S}_{\mathsf{DVZK}}$ be a PPT simulator for the underlying VOLE-based ZK proof DVZK.

Malicious prover. Firstly, we construct a PPT simulator $\mathcal{S}_{\mathsf{PAC}}$ to simulate the adversary's view in the execution of sub-protocol $\Pi^{(2B-2)}_{\mathsf{IT\text{-}PAC}}$. Specifically, $\mathcal{S}_{\mathsf{PAC}}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}_{\mathsf{PAC}}$ samples a random seed $\in \{0,1\}^\lambda$, and then emulates the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ by sending a handle τ_1 to $\mathcal{A}$.

2. Following the protocol specification, $\mathcal{S}_{\mathsf{PAC}}$ samples $\Lambda \leftarrow \mathbb{F}$, and simulates the encryption of polynomial key Λ honestly where a secret key sk is derived from seed. Then, $\mathcal{S}_{\mathsf{PAC}}$ sends the ciphertexts $\langle\Lambda\rangle, \ldots, \langle\Lambda^{(2B-2)}\rangle$ to $\mathcal{A}$.

3. $\mathcal{S}_{\mathsf{PAC}}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ by sampling uniform $\Delta \in \mathbb{F}$ and recording the vectors $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{F}^\ell$ sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{VOLE}}$, where $\ell = 2n + m + 2$ when applying sub-protocol $\Pi^{(2B-2)}_{\mathsf{IT\text{-}PAC}}$ into protocol $\Pi^{\mathsf{SIMD}}_{\mathsf{ZK}}$. Then, $\mathcal{S}_{\mathsf{PAC}}$ computes $\boldsymbol{v} := \boldsymbol{w} - \boldsymbol{u} \cdot \Delta$.

4. By emulating the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}_{\mathsf{PAC}}$ receives the ciphertexts $\langle b_1\rangle, \ldots, \langle b_\ell\rangle$ from $\mathcal{A}$, and sends a handle τ_2 to $\mathcal{A}$. For each $j \in [1, \ell]$, $\mathcal{S}_{\mathsf{PAC}}$ computes b_j by decrypting the ciphertext $\langle b_j\rangle$ with secret key sk, and then computes $\mathsf{K}_j := v_j - b_j \cdot \Delta$ following the protocol description.

5. $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Open) command of functionality $\mathcal{F}_{\mathsf{Com}}$ by sending (seed, τ_1) to $\mathcal{A}$, and also sends Λ to $\mathcal{A}$. In addition, $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ by receiving τ_2 from adversary $\mathcal{A}$.

By invoking $\mathcal{S}_{\mathsf{PAC}}$ and $\mathcal{S}_{\mathsf{DVZK}}$, the simulator $\mathcal{S}_{\mathsf{ZK}}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of $\Pi_{\mathsf{ZK}}^{\mathsf{SIMD}}$ as follows:

1. $\mathcal{S}_{\mathsf{ZK}}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ by invoking $\mathcal{S}_{\mathsf{PAC}}$ to generate a uniform global key $\Delta \in \mathbb{F}$.

2. $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to generate a uniform polynomial key $\Lambda \in \mathbb{F}$.

3. $\mathcal{S}_{\mathsf{ZK}}$ emulates functionality $\mathcal{F}_{\mathsf{VOLE}}$ by recording the values and MACs on $\{[a_{i,j}]\}_{i\in[1,B],j\in[1,m]}$ received from $\mathcal{A}$, and then computes the corresponding local keys in the natural way.

4. After receiving $b_{i,j} \in \mathbb{F}$ for $i \in [1, B], j \in [1, m]$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{ZK}}$ computes $w_{i,j} := a_{i,j} + b_{i,j} \in \mathbb{F}$ for each $i \in [1, B], j \in [1, m]$, and then defines $\boldsymbol{w}_i = (w_{i,1}, \ldots, w_{i,m}) \in \mathbb{F}^m$ as a witness for each $i \in [1, m]$.

5. Simulator $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to simulate the execution of sub-protocol $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$ for generating the IT-PACs $\{[u_j(\cdot)]\}_{j\in[1,m]}$, $\{([h_j(\cdot)], [\widetilde{h}_j(\cdot)])\}_{j\in[1,n]}$ and $([r(\cdot)], [s(\cdot)])$. Following the topological order, $\mathcal{S}_{\mathsf{ZK}}$ computes the local key on the output wire of every addition gate. Now, $\mathcal{S}_{\mathsf{ZK}}$ holds the local keys of the IT-PAC on every wire and the local keys of the IT-MAC on every circuit-input wire.

6. Following the protocol specification, the simulator $\mathcal{S}_{\mathsf{ZK}}$ executes the linear-combination phase of the $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$ procedure with $\mathcal{A}$. Then, $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to open the key Λ to $\mathcal{A}$. Next, $\mathcal{S}_{\mathsf{ZK}}$ uses the Δ and local keys to execute the check phase of $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$ with $\mathcal{A}$.

7. $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{DVZK}}$ to perform the verification of the VOLE-based ZK proof DVZK using the global key Δ and local keys.

8. Following the protocol description, $\mathcal{S}_{\mathsf{ZK}}$ executes the CheckZero procedure with $\mathcal{A}$ for the consistency check of input gates and output gates.

9. If $\mathcal{S}_{\mathsf{ZK}}$ who acts as an honest verifier will abort in some check step, $\mathcal{S}_{\mathsf{ZK}}$ sends $\boldsymbol{w}_i = \perp$ for $i \in [1, B]$ along with a circuit $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}^B$ and then aborts. Otherwise, $\mathcal{S}_{\mathsf{ZK}}$ sends the extracted witnesses $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$ and $\mathcal{C}$ to $\mathcal{F}_{\mathsf{ZK}}^B$.

It is easy to see that the simulation of $\mathcal{S}_{\mathsf{ZK}}$ is perfect, where recall that the DVZK simulation of $\mathcal{S}_{\mathsf{DVZK}}$ invoked by $\mathcal{S}_{\mathsf{ZK}}$ is perfect. Whenever the honest verifier outputs false in the real-world execution, the verifier also outputs false in the ideal-world execution, as $\mathcal{S}_{\mathsf{ZK}}$ sends $\perp$ to $\mathcal{F}_{\mathsf{ZK}}^B$ in this case. Therefore, it only remains to bound the probability that the verifier in the real-world execution outputs true, but there exists some $i \in [1, B]$ such that $\mathcal{C}(\boldsymbol{w}_i) \neq 0$ where $\boldsymbol{w}_i$ is extracted by $\mathcal{S}_{\mathsf{ZK}}$. Below, we show if $\mathcal{C}(\boldsymbol{w}_i) \neq 0$ for some $i \in [1, B]$, then the probability that the verifier outputs true in the real protocol execution is at most $\frac{2B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

By induction, we prove that all addition and multiplication gates in all B executions of circuit $\mathcal{C}$ are evaluated correctly. It is trivial that all addition gates in the $(B, |\mathcal{C}|)$-SIMD circuit are computed correctly from the additively homomorphic property of IT-PACs. Thus, we focus on analyzing the correctness of computing multiplication gates. From the soundness of the DVZK proof, we obtain that $\widetilde{h}_j(\Lambda) = f_j(\Lambda) \cdot g_j(\Lambda)$ for all $j \in [1, n]$, except with probability at most $\epsilon_{\mathsf{dvzk}} = \frac{3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Thus, we can replace the IT-PAC $[\widetilde{h}_j(\cdot)]$ with $[f_j(\cdot) \cdot g_j(\cdot)]$ for each $j \in [1, n]$. From Lemma 4 for the soundness of

$\mathsf{BatchCheck}_{(B-1),(2B-2),B}$, we have that $h_j(\alpha_i) = f_j(\alpha_i) \cdot g_j(\alpha_i)$ for all $i \in [1, B], j \in [1, n]$ hold for the IT-PACs $\{([h_j(\cdot)], [f_j(\cdot) \cdot g_j(\cdot)])\}_{j\in[1,n]}$, except with probability at most $\frac{2B}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Together, for each $j \in [1, n]$, we have that the j-th multiplication gate in all B executions of circuit $\mathcal{C}$ is evaluated correctly, except with probability at most $\frac{2B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

In the following, we prove that the IT-PACs on the input gates and output gates in the $(B, |\mathcal{C}|)$-SIMD circuit are computed in the way consistent to the witnesses and 0 respectively. Specifically, for each $j \in [1, m]$, we define a polynomial $u_j^*(\cdot)$ such that $u_j^*(\alpha_i) = w_{i,j}$ for all $i \in [1, B]$, where $w_{i,j} = a_{i,j} + b_{i,j}$ for $i \in [1, B], j \in [1, m]$. From the definition of $[z_j] = \sum_{i\in[1,B]} \delta_i(\Lambda) \cdot [w_{i,j}]$, we have that $[z_j] = [u_j^*(\Lambda)]$. The probability that there exists some $j \in [1, m]$ such that $u_j(\Lambda) \neq u_j^*(\Lambda)$ but the verifier accepts in the $\mathsf{CheckZero}([u_j(\Lambda)] - [u_j^*(\Lambda)])$ procedure is at most $\frac{1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$, where $[u_j(\cdot)]$ is generated by running sub-protocol $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$. Let E_4 be the event that there exists some $j \in [1, m]$ such that $u_j(\Lambda) = u_j^*(\Lambda)$ but $u_j(\cdot) \neq u_j^*(\cdot)$. According to the proof of Lemma 4, we know that $\Pr[E_4] \leq \frac{2B-2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Therefore, the witnesses are consistent to the IT-PACs on the input gates, except with probability at most $\frac{2B-1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. Since the verifier accepts in the $\mathsf{CheckZero}([v(\Lambda)])$ procedure, we have that $v(\Lambda) = 0$ except with probability at most $\frac{1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$, where $[v(\cdot)]$ be the IT-PAC committing to the values on the output gates. Let E_5 be the event that $v(\Lambda) = 0$ but $v(\cdot) \neq 0$. Again, according to the proof of Lemma 4, we obtain that $\Pr[E_5] \leq \frac{2B-2}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

Overall, we conclude that if $\mathcal{C}(\boldsymbol{w}_i) \neq 0$ for some $i \in [1, B]$, then the probability that the verifier outputs true in the real-world execution is bounded by $\frac{2B+3}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$, where the repeated computation of probabilities is merged.

Malicious verifier. As such, we first construct a PPT simulator $\mathcal{S}_{\mathsf{PAC}}$ to simulate the adversary's view in the execution of sub-protocol $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$. In particular, $\mathcal{S}_{\mathsf{PAC}}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}_{\mathsf{PAC}}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ by recording $\Delta \in \mathbb{F}$ and a vector $\boldsymbol{v} \in \mathbb{F}^\ell$ that are sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{VOLE}}$.

2. $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ by receiving a seed from $\mathcal{A}$ and sending a handle τ_1 to $\mathcal{A}$. Then $\mathcal{S}_{\mathsf{PAC}}$ computes the secret key sk and randomness $r_0, r_1, \ldots, r_{(2B-2)}$ with seed following the protocol specification.

3. On behalf of an honest verifier, $\mathcal{S}_{\mathsf{PAC}}$ receives the AHE ciphertexts $c_1, \ldots, c_{(2B-2)}$ from $\mathcal{A}$.

4. For $j \in [1, \ell]$, $\mathcal{S}_{\mathsf{PAC}}$ samples $b_j \leftarrow \mathbb{F}$ and encrypts b_j as $\langle b_j \rangle$ using secret key sk. Then, $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ by sending a handle τ_2 to $\mathcal{A}$.

5. $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Open) command of functionality $\mathcal{F}_{\mathsf{Com}}$ by receiving a handle τ_1 from $\mathcal{A}$. After receiving $\Lambda \in \mathbb{F}$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{PAC}}$ verifies the correctness of the ciphertexts sent by $\mathcal{A}$ by checking $c_i = \mathsf{Enc}(\mathsf{sk}, \Lambda^i; r_i)$ for $i \in [1, 2B-2]$. If the check fails, $\mathcal{S}_{\mathsf{PAC}}$ aborts. Otherwise, $\mathcal{S}_{\mathsf{PAC}}$ computes $\mathsf{K}_j := v_j - b_j \cdot \Delta$ for each $j \in [1, \ell]$.

6. $\mathcal{S}_{\mathsf{PAC}}$ emulates the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ by sending the ciphertexts $\langle b_1 \rangle, \ldots, \langle b_\ell \rangle$ as well as τ_2 to adversary $\mathcal{A}$.

For the $\mathsf{CheckZero}$ procedure, the verifier checks that the hash value sent by the prover is identical to that computed from the local keys. Therefore, the simulator can use the local keys held by the verifier to simulate $\mathsf{CheckZero}$ by computing the hash value from the local keys. If $\mathcal{S}_{\mathsf{ZK}}$ receives false from functionality $\mathcal{F}_{\mathsf{ZK}}^B$, then it simply aborts. Otherwise, by invoking $\mathcal{S}_{\mathsf{PAC}}$ and $\mathcal{S}_{\mathsf{DVZK}}$, $\mathcal{S}_{\mathsf{ZK}}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}_{\mathsf{ZK}}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ by invoking $\mathcal{S}_{\mathsf{PAC}}$ to receive a global key $\Delta \in \mathbb{F}$ from $\mathcal{A}$, and stores Δ.

2. $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to simulate the execution about creating a polynomial key Λ.

3. $\mathcal{S}_{\mathsf{ZK}}$ emulates $\mathcal{F}_{\mathsf{VOLE}}$ by recording the local keys on IT-MACs $[a_{i,j}]$ for $i \in [1, B], j \in [1, m]$ that are sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{VOLE}}$.

4. For each $i \in [1, B], j \in [1, m]$, $\mathcal{S}_{\mathsf{ZK}}$ samples $b_{i,j} \leftarrow \mathbb{F}$ and sends it to $\mathcal{A}$, and then computes the local key on the IT-MAC $[w_{i,j}]$.

5. $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to simulate the execution of $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$ for generating the IT-PACs $\{[u_j(\cdot)]\}_{j\in[1,m]}$, $\{([h_j(\cdot)], [\widetilde{h}_j(\cdot)])\}_{j\in[1,n]}$ and $([r(\cdot)], [s(\cdot)])$. $\mathcal{S}_{\mathsf{ZK}}$ also computes the local key on the output wire of every addition gate. Now, $\mathcal{S}_{\mathsf{ZK}}$ holds the local keys on all IT-PACs and IT-MACs.

6. For the linear-combination phase of $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$, $\mathcal{S}_{\mathsf{ZK}}$ receives a seed from $\mathcal{A}$, and then samples two random polynomials $f(\cdot)$ and $g(\cdot)$ in $\mathbb{F}[X]$ such that $f(\alpha_i) = g(\alpha_i)$ for all $i \in [1, B]$, where the degrees of $f(\cdot)$ and $g(\cdot)$ are $B-1$ and $2B-2$ respectively. Then, $\mathcal{S}_{\mathsf{ZK}}$ sends $f(\cdot)$ and $g(\cdot)$ to $\mathcal{A}$.

7. $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{PAC}}$ to simulate the execution of receiving the key Λ from $\mathcal{A}$. For the check phase of $\mathsf{BatchCheck}_{(B-1),(2B-2),B}$, $\mathcal{S}_{\mathsf{ZK}}$ computes the local keys on IT-MACs $[f(\Lambda)] - f(\Lambda)$ and $[g(\Lambda)] - g(\Lambda)$, and then uses them to execute the $\mathsf{CheckZero}$ procedure with $\mathcal{A}$, where note that $f(\Lambda)$ and $g(\Lambda)$ can be computed by $\mathcal{S}_{\mathsf{ZK}}$.

8. Using Δ and the local keys, simulator $\mathcal{S}_{\mathsf{ZK}}$ invokes $\mathcal{S}_{\mathsf{DVZK}}$ to simulate the protocol execution of DVZK without knowing the underlying witness.

9. Following the protocol specification, $\mathcal{S}_{\mathsf{ZK}}$ computes the local keys on IT-MACs $[u_j(\Lambda)] - [z_j]$ for all $j \in [1, m]$, and then uses them to execute the $\mathsf{CheckZero}$ procedure with $\mathcal{A}$.

10. $\mathcal{S}_{\mathsf{ZK}}$ uses the local key on $[v(\cdot)]$ to run the $\mathsf{CheckZero}([v(\Lambda)])$ procedure with $\mathcal{A}$, where the local key in IT-MAC $[v(\Lambda)]$ is equal to that in IT-PAC $[v(\cdot)]$.

Clearly, the simulation of functionalities $\mathcal{F}_{\mathsf{VOLE}}$ and $\mathcal{F}_{\mathsf{Com}}$ is perfect. In the real execution of sub-protocol $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$, the ciphertexts $\langle\Lambda\rangle, \ldots, \langle\Lambda^{(2B-2)}\rangle$ sent by $\mathcal{A}$ are guaranteed to be computed correctly by the opening and checking step. In the $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model, for each $j \in [1, \ell]$, u_j is uniform in $\mathbb{F}$, and thus $b_j = f(\Lambda) - u_j$ is also uniform in $\mathbb{F}$. For the j-th ciphertext $\langle b_j\rangle$ with $j \in [1, \ell]$ in the execution of sub-protocol $\Pi_{\mathsf{IT\text{-}PAC}}^{(2B-2)}$, while $\langle b_j\rangle$ is computed as $\langle b_j\rangle = \sum_{i=1}^{k} f_{j,i} \cdot \langle\Lambda^i\rangle + f_{j,0} - u_j$ in the real-world execution, $\langle b_j\rangle$ is computed as $\mathsf{Enc}(\mathsf{sk}, b_j)$ for a random $b_j \in \mathbb{F}$ in the ideal-world execution. For each $j \in [1, \ell]$, the message b_j has the identical distribution in both worlds, and the only difference between the real-world execution and ideal-world execution is the computation method of ciphertext $\langle b_j\rangle$. The difference is easy to be bounded by a reduction to the circuit privacy of the underlying AHE scheme, and thus is negligible in λ.

For the input processing, for each $i \in [1, B], j \in [1, m]$, while $b_{i,j} = w_{i,j} - a_{i,j} \in \mathbb{F}$ in the real-world execution, $b_{i,j} \in \mathbb{F}$ is sampled uniformly at random in the ideal-world execution. Due to the uniformity of $a_{i,j}$ for $i \in [1, B], j \in [1, m]$, the distribution of $\{b_{i,j}\}_{i\in[1,B],j\in[1,m]}$ is identical in both worlds. In the real protocol execution, the random linear combination of polynomials $h_j(\cdot), \widetilde{h}_j(\cdot)$ for $j \in [1, n]$ is masked by two random polynomials $r(\cdot), s(\cdot)$ with $r(\alpha_i) = s(\alpha_i)$ for $i \in [1, B]$. Therefore, the resulting polynomials $f(\cdot) = \sum_{j=1}^{\ell} \chi_j \cdot h_j(\cdot) + r(\cdot)$ and $g(\cdot) = \sum_{j=1}^{\ell} \chi_j \cdot \widetilde{h}_j(\cdot) + s(\cdot)$ are uniform in $\mathbb{F}[X]$ such that $f(\alpha_i) = g(\alpha_i)$ for all $i \in [1, B]$. Therefore, the polynomials $f(\cdot)$ and $g(\cdot)$ simulated by $\mathcal{S}_{\mathsf{ZK}}$ have the identical distribution as that in the real protocol execution. Since $\mathcal{S}_{\mathsf{ZK}}$ computes the local keys in the same way as that done by verifier $\mathcal{V}$, the simulation of $\mathsf{CheckZero}$ has the identical distribution as the $\mathsf{CheckZero}$ execution in the real world. From the zero-knowledge property of DVZK, we have that the simulation of protocol DVZK is indistinguishable from the real protocol execution.

Procedure PrepCircuit

Inputs: A prover $\mathcal{P}$ and a verifier $\mathcal{V}$ hold a generic circuit $\mathcal{C}$. Let C_M (resp., C_A) be the number of multiplication (resp., addition) gates in the circuit. Let B be the number of wire values committed in a single IT-PAC.

Circuit compilation: $\mathcal{P}$ and $\mathcal{V}$ transform a generic circuit $\mathcal{C}$ into another equivalent circuit $\mathcal{C}'$ as follows:

1. Insert $\lceil \frac{C_M}{B} \rceil \cdot B - C_M$ multiplication gates and $\lceil \frac{C_A}{B} \rceil \cdot B - C_A$ addition gates into the circuit. Specifically, each of these inserted gates takes two new input wires as the inputs of the gate, and creates one new output wire as its output. The values on these new wires are set as 0. The new input wires create two new input gates, and the new output wire also creates a new output gate.
2. If the number C_O of all output gates (including these new inserted output gates from the previous step) is less than B, insert $B - C_O$ new input wires that set 0 as the wire values, and then define these input wires as output wires directly. As such, the new input and output wires create new input and output gates.
3. Let C_I be the number of all input gates (including these new inserted input gates). Insert $\lceil \frac{C_I}{B} \rceil \cdot B - C_I$ new input wires which set 0 as the wire values, and also define these input wires as output wires directly. The new wires create new input and output gates.

Commit to wire values: Now, the number of same-type gates (except for the output gates) in the circuit are multiple of B, and there are at least B output gates. Two parties $\mathcal{P}$ and $\mathcal{V}$ do the following:

- Divide every B input gates into a group, and the input values in a group will be committed by a single IT-PAC.
- Following a predetermined topological order, execute the following:
  1. Collect B unused multiplication gates into a group. For the group of multiplication gates, the values on the first input wires and the second input wires of these gates will be respectively committed by two IT-PACs, and the values on the output wires of these gates will be committed by a single IT-PAC.
  2. Collect B unused addition gates into a group. For the group, commit to the input and output values by IT-PACs as above.
- Divide the single actual output gate along with $B - 1$ "dummy" output gates into a group. The values on the B output gates will be committed by a single IT-PAC.

Figure 8: Procedure for preprocessing a circuit.

Overall, any PPT environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, which completes the proof. □

D Preprocess Circuits

In Figure 8, we show how to compile a circuit $\mathcal{C}$ into an equivalent circuit $\mathcal{C}'$ with the same output by adding "dummy" gates and wires, where the values on the new inserted wires are set as 0. Furthermore, we describe how to divide every B same-type gates into a group and commit to the corresponding values using an IT-PAC. In particular, by scanning the circuit, one can dynamically maintain a list, which adds the groups of wires having IT-PACs and removes the groups of output wires that will not be used as the input wires of subsequent gates. We note that the preprocessing of a circuit shown in Figure 8 does not introduce any communication.