# An Efficient ZK Compiler from SIMD Circuits to General Circuits

Dung Bui* Haotian Chu†‡ Geoffroy Couteau* Xiao Wang† Chenkai Weng†

Kang Yang§ Yu Yu‡

**Abstract**

We propose a generic compiler that can convert any zero-knowledge proof for SIMD circuits to general circuits efficiently, and an extension that can preserve the space complexity of the proof systems. Our compiler can immediately produce new results improving upon state of the art.

- By plugging in our compiler to Antman, an interactive sublinear-communication protocol, we improve the overall communication complexity for general circuits from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. Our implementation shows that for a circuit of size $2^{27}$, it achieves up to 83.6× improvement on communication compared to the state-of-the-art implementation. Its end-to-end running time is at least 70% faster in a 10Mbps network.
- Using recent results on compressed Σ-protocol theory, we obtain a discrete-log-based constant-round zero-knowledge argument with $\mathcal{O}(C^{1/2})$ communication and common random string length, improving over the state of the art that has linear-size common random string and requires heavier computation.
- We improve the communication of a designated $n$-verifier zero-knowledge proof from $\mathcal{O}(nC/B + n^2B^2)$ to $\mathcal{O}(nC/B + n^2)$.

To demonstrate the scalability of our compilers, we were able to extract a commit-and-prove SIMD ZK from Ligero and cast it in our framework. We also give one instantiation derived from LegoSNARK, demonstrating that the idea of CP-SNARK also fits in our methodology.

## 1 Introduction

Assume that the verification of a statement is represented as a public circuit $C : \{0,1\}^n \to \{0,1\}$. A zero-knowledge proof (ZKP) allows a prover to convince a verifier that it possesses a witness $w$ such that $C(w) = 0$, without the verifier learning any information beyond the circuit output. The commit-and-prove zero-knowledge (CP-ZK) paradigm is among the most flexible and modular design mechanisms for constructing ZKP. For instance, a CP-SNARKs allows a prover to commit to a batch of secrets via a commitment scheme (e.g. vector commitment or polynomial commitment), then prove relations between the committed values in ZK [CFQ19, CFF+21, Lip16]. A small

*Université Paris Cité, CNRS, IRIF {bui,couteau}@irif.fr

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

‡Shanghai Jiao Tong University {chtvii,yuyu}@sjtu.edu.cn

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

communication footprint is achieved when the commitment is compressing and the proof is succinct. On the other hand, schemes like VOLE-based ZKPs [BMRS21, WYKW21, YSWW21, DIO20] rely on efficient interactive commitment scheme that separately commits to wire values in the circuit, then prove the consistency between committed wire values with constant overhead. Though general VOLE-ZKs incur communication complexity linear to the circuit size, they achieve high throughput owing to the lightweight operations.

Generally, CP-ZK proof systems with sublinear communication involve two components after the batch commitment of witnesses: (1) Hadamard product of committed vectors, (2) equality of individual wires across different committed vectors. The former is used to demonstrate the correct computation of multiplication gates and the latter is used to show that the committed wire values are consistent with the circuit topology.

**From SIMD-ZK to general ZK.** From another perspective, the above approach can be viewed as a conversion from commit-and-prove SIMD-ZK to general ZK. Define $(B, \mathcal{C})$-SIMD circuit which contains $B$ identical components of the circuit $\mathcal{C}$. A *SIMD-ZK* proves that for input witnesses $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$, $\mathcal{C}(\boldsymbol{w}_i) = 0$ for $i \in [B]$. By exploiting the fact that operations are identical across $B$ components, SIMD-ZK schemes typically utilize vector commitments and batch proofs to achieve communication sublinear in $B \cdot |\mathcal{C}|$. In more detail, denote by $[\![\boldsymbol{w}]\!]$ a commitment to a vector $\boldsymbol{w}$. Define a witness matrix $\boldsymbol{W} = (\boldsymbol{w}_1 \| \ldots \| \boldsymbol{w}_B)$. Instead of viewing the $i$th column as the witness to the $i$th evaluation of $\mathcal{C}$, a prover commits to each row vector and lets the verifier obtain $([\![\boldsymbol{w}^1]\!], \ldots, [\![\boldsymbol{w}^{|\mathcal{C}|}]\!])$. In this way, for any gate $(\alpha, \beta, \gamma, \diamond)$ in $\mathcal{C}$ and $\diamond \in \{\mathsf{Add}, \mathsf{Mult}\}$, the prover only needs to prove that $\boldsymbol{w}^\gamma = \boldsymbol{w}^\alpha \diamond \boldsymbol{w}^\beta$. ZKP schemes achieve $\mathcal{O}(|\mathcal{C}|)$ proof size if both the vector commitment and batch proof of additions and multiplications incur constant size.

Most of priors work on different proof systems indeed take this approach by first implementing batch commitment and proof of multiplication gates, which are followed by a wiring consistency check [GWC19, CHM+20, CFQ19, CFF+21, AHIV17, WYY+22, YW22]. However, they take divergent paths to tackle the latter problem. A popular approach is to compile the circuit into an algebraic format via a constraint system, e.g. rank-1 constraint system (R1CS) [GGPR13]. Define $\boldsymbol{z} := (1, \boldsymbol{x}, \boldsymbol{w})$ in which $\boldsymbol{x}$ and $\boldsymbol{w}$ are the public and private inputs of the circuit. Denote $(\mathbf{L}, \mathbf{R}, \mathbf{O})$ as the matrices that represent the map from $\boldsymbol{z}$ to the vectors of the left, right and output wires of multiplication gates $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$. Then the relation $\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{c} = \mathbf{0}$ can be expressed as $(\mathbf{L} \cdot \boldsymbol{z}) * (\mathbf{R} \cdot \boldsymbol{z}) - (\mathbf{O} \cdot \boldsymbol{z}) = \mathbf{0}$. In this way, the ZKP is reduced to proving matrix-vector products on committed values. On the other hand, some ZKPs like [WYY+22] and [YW22] proceed differently: they individually prove that $\boldsymbol{w}^\alpha[i] = \boldsymbol{w}^\beta[j]$ for any $i, j \in [B]$. Although this approach yields better scalability for the ZKP, it results in worse communication complexity, usually with a $B^2$ factor.

An interesting question is whether we can design a generic compiler that translates any commit-and-prove SIMD-ZK (CP-SIMD-ZK) into a general CP-ZK with sublinear communication. It would facilitate the design of communication-efficient ZKP because it allows the focus to be shifted to the design of SIMD-ZK primitives, which are generally easier than general-purpose ZKP.

**From SIMD-ZK to scalable ZK.** It is common for ZKPs to trade off scalability against succinctness. On the one hand, although zk-SNARKs generate proofs of constant size or size sublinear to $|\mathcal{C}|$, their memory overhead is at least $\mathcal{O}(|\mathcal{C}|)$. The constant factor is large when public-key operations are involved. This prevents them from being applied to large statements: prior benchmarks only focus on statements represented by less than $2^{25}$ constraints [CBBZ22]. Efforts are made to distribute the zk-SNARK proof generation among a set of provers [WZC+18, OB22, SVdV16, KZGM21, BG22], however, the overall computational and memory overhead is still prohibitive. They either need to disclose secret input to all provers, or only aim to delegate computation to more powerful workers but not to reduce the computational cost of them. Another line of work focuses on re-

cursive SNARKs [KST22, KS22, BGH19] that allow a statement that can be divided into multiple steps to be proven step-by-step, but they require the statement to be structured, i.e., each step is represented by identical constraints. On the other hand, interactive ZKPs such as VOLE-ZK [BMRS21, WYKW21, YSWW21, DIO20] achieve high scalability by "streaming" the circuit evaluation. They evaluate the circuit gate-by-gate and only incur memory overhead linear in the current gates that are evaluated. Neither the witness nor the circuit structure for future gates are required to be known in advance. Hence these types of ZKPs scale to large circuits with billions of gates. However, their drawback is the $\mathcal{O}(|\mathcal{C}|)$ communication complexity and lack of public verifiability.

Naturally, it would be interesting to study how to achieve scalability and succinctness at the same time. Specifically, can we obtain efficient ZKPs with proof size sublinear to the circuit size, without the memory overhead being lower bounded by the circuit size?

### 1.1 Our Contributions

In this work, we start from SIMD-ZK schemes and aim to obtain efficient general ZK and scalable ZK. We first extend the SIMD-ZK functionality by adding a proof of linear map, which is easily realized by most SIMD-ZK schemes. Then we design two compilers. The first one converts a wide range of extended SIMD-ZK to general ZK, and the second one further converts it to scalable ZK for memory-constrained provers to prove large statements. For both constructions, we also demonstrate the generality of the compilers, i.e., our methods promote any SIMD-ZK to general and possibly scalable ZK so that attention can be paid only to the design of the efficient SIMD-ZK, instead of more complicated generic primitives. Our contributions are fourfold.

**Extended SIMD-ZK.** We propose a functionality that extends the SIMD-ZK functionality $\mathcal{F}_{\mathsf{SIMDZK}}$ and denote it as $\mathcal{F}_{\mathsf{eSIMDZK}}$. In addition to the subroutines *commit*, *open* and *prove* that are commonly supported by SIMD-ZK schemes, it also contains a proof of linear map that checks the relation $\boldsymbol{x} = \mathbf{M}\boldsymbol{y}$ for committed vectors $(\boldsymbol{x}, \boldsymbol{y})$. The functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ is the fundamental building block of our constructions. Additionally, we observe that special attention needs to be paid to the security of commit-and-prove procedures when designing a general framework for scalable ZK. Some commitment schemes may put a restriction on its proving phase. The security consideration will be reflected as a counter in $\mathcal{F}_{\mathsf{SIMDZK}}$ and analyzed when such a commitment scheme is encountered.

**Compiling SIMD-ZK to general ZK.** Based on the extended SIMD-ZK, we design a SIMD compiler that allows a wide spectrum of SIMD-ZK to work for general circuits. To do so, it first converts the general circuit into a SIMD circuit by ignoring the circuit connectivity, and proves its satisfiability via a SIMD proof. This only utilizes the *commit*, *open* and *prove* thus can be handled by the underlying SIMD-ZK. Then the compiler represents the wiring as a linear mapping of committed wire values, and proves the wiring consistency by the proof of linear map from $\mathcal{F}_{\mathsf{eSIMDZK}}$. Our compiler is a generalization of a few works including Ligero [AHIV17, BFH+20] and LegoSNARK [CFQ19], which utilize R1CS-style representations for the wiring of circuits and reduce the statement to relations that can be better handled by the extended SIMD-ZK.

**ZKP for large statements.** Except for VOLE-based ZKP, most practical ZKPs incur large RAM consumption, often linear to the circuit size. To relax the memory overhead, we propose a framework for memory bounded provers to prove the correctness of large statements. It also relies on $\mathcal{F}_{\mathsf{eSIMDZK}}$ and can easily achieve sublinear communication complexity for arbitrary large circuits by properly instantiating the underlying SIMD-ZK. Particularly, it utilizes the proving technique in our SIMD compiler to evaluate a circuit segment by segment and prove the connectivity of wires

between these segments. Similar to the current scalable interactive ZK, it does not require the whole circuit structure or the witness to be known in advance, hence allowing streaming.

**Instantiation for various proof systems.** To demonstrate the generality of our compiler, we describe and analyze the detailed instantiation of our compiler with various CP-ZK that inherently work well for SIMD circuits, including VOLE-based ZK [WYY+22], constant-round sublinear ZK from Σ-protocol [AC20], designated multi-verifier ZK from packed Shamir sharing [YW22], MPC-in-the-Head [AHIV17] and zk-SNARK from pairing [CFQ19]. We show how to adapt these work for general ZK and scalable ZK by merely satisfying the minimum requirement, that is, realizing the SIMD-ZK functionality. We emphasize that the transformation may affect the security guarantee of the underlying SIMD-ZK, and extra security analysis will be provided in that case.

In many cases, applying our compiler yields concrete efficiency improvements over the state of the art in various settings. We list our results in Section 2.2. Furthermore, we implement the SIMD compiler and evaluate the compilation of a VOLE-based ZK [WYY+22] that is previously designed for SIMD circuits. For a circuit of size $|\mathcal{C}| = 2^{27}$, it shows up to 83.6× improvement on communication, compared to the general VOLE-ZK Quicksilver [YSWW21]. In terms of running time, it is 70% faster when bandwidth is 10Mbps and 30% faster when bandwidth increases to 25Mbps using the same set of parameters. Its running time can be further improved if sacrificing communication by reducing batch size.

### 1.2 Related Work

Previous work on complexity-preserving zero-knowledge proofs study efficient proof generation with constrained space or time budget [BHR+20, BHR+21, EFKP20, HR18, BC12, BCCT13]. Bootle et al. propose elastic SNARKs that can either achieve linear time and space complexity, or reduce the RAM consumption to $\mathcal{O}(\log C)$ with $\mathcal{O}(C \log^2 C)$ computational complexity [BCHO22]. Assume an NP relation that can be verified in time $T$ and space $S$ by a RAM program, Bangalore et al. [BBHV22] propose a public-coin ZKP based on collision-resistant hash functions that allows the prover to run in time $\tilde{\mathcal{O}}(T)$ and space $\tilde{\mathcal{O}}(S)$, with proof size $\tilde{\mathcal{O}}(T/S)$. Their space-preserving ZKP is converted from Ligero [AHIV17].

Recent recursive zk-SNARK and incremental verifiable computation (IVC) propose succinct arguments for composed circuits, which can be evaluated step by step [KST22, KS22, KS23, STW23, BGH19, BCL+21]. These techniques increase the scalability of the prover, who separately generates proof for each step while simultaneously proves its consistency with all previous steps without going over the history data. They can potentially support streaming proofs in a way that the input and witness for future steps are not necessary known until those steps are reached. However, many of them only support structured circuit which are divided into a sequence of components that share the same structure. More advanced IVCs cross this barrier, however they reveal the output of each step thus does not provide the zero-knowledge guarantee when they are treated as general ZK [KST22, KS22].

### 1.3 Notation and Functionalities

**Notation.** Denote $\lambda$ as the computational security parameters and $[1, m]$ as a set $\{1, 2, \ldots, m\}$. For a vector $\mathbf{x}$ we define its $i$-th coordinate by $x_i$. Given distribution ensembles $\{X_n\}, \{Y_n\}$, we write $X_n \approx Y_n$ to denote that $X_n$ is computationally indistinguishable to $Y_n$. $\mathsf{negl}()$ is defined as a negligible function such that $\mathsf{negl}(\lambda) = o(\lambda^{-c})$ for any 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. $|\mathcal{C}| = C$

is the number of multiplication gates in the circuit $\mathcal{C}$. Define $(B, \mathcal{C})$-SIMD circuit as a circuit that contains $B$ copies of $\mathcal{C}$.

**Zero-knowledge proof $\mathcal{F}_{\mathsf{ZK}}$:**

- Upon receiving $(\mathsf{prove}, \mathcal{C}, \boldsymbol{w})$ from prover $\mathcal{P}$ and $(\mathsf{verify}, \mathcal{C})$ from verifier $\mathcal{V}$, if $\mathcal{C}(\boldsymbol{w}) = 0$, then output $(\mathsf{true})$ to $\mathcal{V}$, else output $(\mathsf{false})$ to $\mathcal{V}$.

**Vector oblivious linear evaluation $\mathcal{F}_{\mathsf{VOLE}}$.** This functionality works over a field $\mathbb{F}$, and upon receiving $(\mathsf{init})$ 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 $\Delta$ and ignore all subsequent $(\mathsf{init})$ commands. Upon receiving $(\mathsf{extend}, n)$ 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 $\mathcal{V}$.

**Commitment $\mathcal{F}_{\mathsf{Com}}$.** Similar to the functionality of $\mathsf{Commit}$ command in $\mathcal{F}_{\mathsf{SIMDZK}}$:

- Upon receiving input $(\mathsf{Commit}, \boldsymbol{w})$ from $\mathcal{P}$ and $(\mathsf{Commit})$ from $\mathcal{V}$, pick a tag $[\![\boldsymbol{w}]\!]$ and store $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ in the memory. Return $[\![\boldsymbol{w}]\!]$ to both parties.
- Upon receiving $(\mathsf{Open}, [\![\boldsymbol{w}]\!])$, if a tuple $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ was previously stored, output $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ to $\mathcal{V}$; otherwise abort.

The descriptions of special honest-verifier ZK argument are deferred to Supplementary Material A.1.

# 2 Technical Overview

## 2.1 From SIMD to General Circuit in ZK

Denote the prover as $\mathcal{P}$ and verifier as $\mathcal{V}$. Define $(B, \mathcal{C})$-SIMD circuit as $B$ identical repetitions of a circuit $\mathcal{C}$ with size $|\mathcal{C}| = C$. SIMD-ZK is designed for such circuits. First, we would like to focus on converting SIMD-ZK to general ZK that works for arbitrary circuits. The functionality of ZKP for SIMD circuits is shown in figure 1. $\mathcal{P}$ first groups and commits to the vectors of witnesses. Then it use the underlying ZKP to prove the relation of committed values by directly operating on commitments. Since elements in each vector are committed in a batch, the operations on the commitment apply to all of the committed elements. For a SIMD-ZK to be interesting, it usually costs less than separately evaluating $\mathcal{C}$ for $B$ times. For example, AntMan [WYY+22] has a complexity of $\mathcal{O}(B+C)$ for $(B, \mathcal{C})$-SIMD circuits, which shows significant saving on communication compared with its non-SIMD opponents [YSWW21, DIO20] that incur $\mathcal{O}(BC)$ complexity.

There are multiple ways to conduct the transformation from SIMD-ZK to general ZK. As discussed in Section 1, such constructions usually need a wire consistency check on top of SIMD-ZK. Taking AntMan [WYY+22] as an example, one can first arrange all gates in batches, commit to their input and output wire values, then utilize a SIMD-ZK to prove that all batches of gates are computed correctly. Then an extra protocol is invoked to prove the consistency of each individual wire value that is repeatedly packed in multiple commitments, E.g. for batched wire values $\boldsymbol{w}_1, \boldsymbol{w}_2 \in$

**Functionality $\mathcal{F}_{\mathsf{SIMDZK}}$**

**Public parameter:** Define $B$ to be the batch size and $\tau_{\mathsf{max}}$ to be the maximum time that a commitment can be used in the proof.

**Commit:** Upon receiving input $(\mathsf{Commit}, \boldsymbol{w} \in \mathbb{F}^B)$ from $\mathcal{P}$ and $(\mathsf{Commit})$ from $\mathcal{V}$, pick a tag $[\![\boldsymbol{w}]\!]$ and store $([\![\boldsymbol{w}]\!], \boldsymbol{w}, \mathsf{ctr}_{\boldsymbol{w}} = 0)$ in the memory. Return $[\![\boldsymbol{w}]\!]$ to both parties.

**Open:** Upon receiving $(\mathsf{Open}, [\![\boldsymbol{w}]\!])$, if a tuple $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ was previously stored, output $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ to $\mathcal{V}$; otherwise abort.

**Prove:** Upon receiving $(\mathsf{Prove}, \mathcal{C}, [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$, where the circuit $\mathcal{C} : \{0,1\}^m \to \{0,1\}$, fetch $\boldsymbol{w}_i$ from the memory, for $i \in [m]$. If for any $\boldsymbol{w}_i$ that $[\![\boldsymbol{w}_i]\!]$ does not exist or its counter $\mathsf{ctr}_{\boldsymbol{w}_i} \geq \tau_{\mathsf{max}}$, abort. Check $\mathcal{C}(\boldsymbol{w}_1[i], \ldots, \boldsymbol{w}_m[i]) = 0$ for all $i \in [B]$. If any check fails, abort; otherwise, return $\mathsf{Pass}$. For $i \in [m]$, set $\mathsf{ctr}_{\boldsymbol{w}_i} = \mathsf{ctr}_{\boldsymbol{w}_i} + 1$.

Figure 1: Functionality of SIMD ZK.

$\mathbb{F}^B$ and wire indices $i, j \in [B]$, it aims to check whether they satisfy $\boldsymbol{w}_1[i] = \boldsymbol{w}_2[j]$. AntMan requires $\mathcal{O}(B^3)$ complexity for checking all combinations of $(i, j) \in [B] \times [B]$, which leads to a total communication complexity of $\mathcal{O}(B^3 + C/B)$. This translates to a $\mathcal{O}(C^{3/4})$ cost when setting $B = C^{1/4}$. The designated multi-verifier ZK from [YW22] also uses a similar wire consistency check, which incurs $\mathcal{O}(n^2 B^2)$ among $n$ verifiers.

**A better wire consistency check.** We follow an idea similar to the above but manage to improve the complexity from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. As in AntMan [WYY+22], we ignore the wiring of the circuit and pack the multiplication gates in blocks of size $B$, which results in $C/B$ batches. The SIMD proof is invoked to first commit to the input and output wires of the packed multiplication gates, then prove the SIMD circuit satisfiability. They totally incur communication complexity $\mathcal{O}(C/B)$. Then, we manage to perform the wire consistency check with cost $\mathcal{O}(B)$ rather than $\mathcal{O}(B^3)$.

Instead of considering the wire consistency among each pair of commitments that contain values from the same wire as done in AntMan, we consider how they are all consistent with a global vector $\boldsymbol{w}$ that contains all wire values in the circuit. Taking the left input wire of all multiplication gates as an example. Define a circuit $\mathcal{C}$ that has a total of $Bm$ wire values and $Bn$ multiplication gates. Assume global wire values $\boldsymbol{w} \in \mathbb{F}^{Bm}$ and the values of left input wires across all multiplication gates $\boldsymbol{l} \in \mathbb{F}^{Bn}$. For any $i \in [Bn]$, the left wire of the $i$-th multiplication gate must be associated a wire index $\alpha_i \in [Bm]$ such that $\boldsymbol{l}[i] = \boldsymbol{w}[\alpha_i]$. Alternatively, one can define a mapping matrix $\mathbf{L} \in \{0,1\}^{Bn \times Bm}$ such that the $i$-th row $\mathbf{L}_i$ is all-zero except at the entry $\mathbf{L}_i[\alpha_i]$. In this way, the wire consistency check boils down to check $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$, where $\mathbf{L}$ is public and parties have commitments $\{[\![\boldsymbol{l}_i]\!]\}_{i \in [n]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [m]}$. In the context of SIMD-ZK protocols, values in $\boldsymbol{l}$ and $\boldsymbol{w}$ are batch-committed, meaning that operations on them are applied to every element in the vector. As a result, it is not straightforward to use SIMD-ZK to prove wire consistency which intuitively involves operations for separate elements.

We sketch our idea below. First, let $\mathcal{V}$ send a challenge vector $\boldsymbol{r} \in \mathbb{F}^{Bn}$ and convert the check of $\boldsymbol{l} \stackrel{?}{=} \mathbf{L}\boldsymbol{w}$ to the check of $\boldsymbol{r}^\intercal \boldsymbol{l} \stackrel{?}{=} \boldsymbol{r}^\intercal \mathbf{L}\boldsymbol{w}$. This reduces the proof of a matrix-vector multiplication to a proof of two inner products, with an increase in soundness error depending on the distribution of $\boldsymbol{r}$. To simplify the notation, we define a public vector $\boldsymbol{v}^\intercal = \boldsymbol{r}^\intercal \mathbf{L}$, then rewrite the above relation

as $\boldsymbol{r}^\intercal \boldsymbol{l} \stackrel{?}{=} \boldsymbol{v}^\intercal \boldsymbol{w}$. If we define a circuit $\mathcal{C} : \mathbb{F}^{2n+2m+1} \to \mathbb{F}$ such that

$$\mathcal{C}(r_1, \ldots, r_n, l_1, \ldots, l_n, v_1, \ldots, v_m, w_1, \ldots, w_m, q) : \sum_{i\in[n]} r_i \cdot l_i - \sum_{j\in[m]} v_j \cdot w_j - q,$$

then $\mathcal{P}$ can prove the above statement by: 1) Divide each of the vectors in $(\boldsymbol{r}, \boldsymbol{l}, \boldsymbol{v}, \boldsymbol{w})$ into length-$B$ segments. Compute and commit to $\boldsymbol{q} := \sum_{i\in[n]} \boldsymbol{r}_i * \boldsymbol{l}_i - \sum_{j\in[m]} \boldsymbol{v}_j * \boldsymbol{w}_j \in \mathbb{F}^B$. Prove the consistency between $(\boldsymbol{r}, \boldsymbol{l}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{q})$ by using a SIMD-ZK composed of $B$ evaluations of the circuit $\mathcal{C}$. 2) prove that $\sum_i \boldsymbol{q}[i] = 0$. This is not obvious, as it involves the computation of the sum of values in one commitment. A naive way is for $\mathcal{P}$ to open the commitment to $\boldsymbol{q}$, but it compromises the zero-knowledge requirements because $\boldsymbol{q}$ is the linear combination of private circuit wire values. To tackle the problem, $\mathcal{P}$ instead commits to a uniform vector $\boldsymbol{r}^* \in \mathbb{F}^B$ under the constraint that $\sum_{i\in[B]} \boldsymbol{r}^*[i] = 0$. It should be done before $\mathcal{V}$ samples $\boldsymbol{r}$ (else $\mathcal{P}$ can break soundness). After $\mathcal{P}$ commits to the mask vector, $\mathcal{V}$ sends the challenge $\boldsymbol{r}$ and the new SIMD circuit is defined to be

$$\begin{aligned}&\mathcal{C}'(r_1, \ldots, r_n, l_1, \ldots, l_n, v_1, \ldots, v_m, w_1, \ldots, w_m, q, r^*)\\ &= \sum_{i\in[n]} r_i \cdot l_i - \sum_{j\in[m]} v_j \cdot w_j - q - r^*\end{aligned}$$

$\mathcal{P}$ computes and commits to $\boldsymbol{q} \in \mathbb{F}^B$ such that

$$\boldsymbol{q} = \sum_{i\in[n]} \boldsymbol{r}_i * \boldsymbol{l}_i - \sum_{j\in[m]} \boldsymbol{v}_j * \boldsymbol{w}_j - \boldsymbol{r}^*.$$

The parties can now use the SIMD-ZK to prove $B$ number of instances of $\mathcal{C}'$ with committed inputs $[\![\boldsymbol{r}_1]\!], \ldots, [\![\boldsymbol{r}_n]\!]$, $[\![\boldsymbol{l}_1]\!], \ldots, [\![\boldsymbol{l}_n]\!]$, $[\![\boldsymbol{v}_1]\!], \ldots, [\![\boldsymbol{v}_m]\!], [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!]$, $[\![\boldsymbol{q}]\!]$ and $[\![\boldsymbol{r}^*]\!]$. Finally, the proof of $\sum_i \boldsymbol{q}[i] = 0$ is specific to the underlying commitment schemes. The naive way is to let $\mathcal{P}$ fully open $\boldsymbol{q}$ to $\mathcal{V}$ who verifies its sum locally. This would generally require $\mathcal{O}(B)$ communication complexity.

Soundness comes from the randomness of the challenge vector $\boldsymbol{r}$ that is sampled after $\mathcal{P}$ commits to $\boldsymbol{r}^*$. Assume that $\mathbb{F}$ is an exponentially large field and a cheating prover commits to $(\boldsymbol{l}, \boldsymbol{w})$ such that $\boldsymbol{l} - \mathbf{L}\boldsymbol{w} \neq \mathbf{0}^{Bn}$. By Schwarz-Zippel, the probability that the erroneous values happen to be corrected by $\boldsymbol{r}$ during the check of $\sum_i \boldsymbol{q}[i] \stackrel{?}{=} 0$ where $\boldsymbol{q} := \boldsymbol{r}^\intercal \boldsymbol{l} - \boldsymbol{r}^\intercal \mathbf{L}\boldsymbol{w}$ is $1/|\mathbb{F}|$, which is negligible.

**Functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$**

**Public parameter:** batch size $B$.
$\mathcal{F}_{\mathsf{eSIMDZK}}$ supports all that $\mathcal{F}_{\mathsf{SIMDZK}}$ supports and the following instruction.

**Linear map:** Upon receiving input $(\mathsf{LinearMap}, [\![\boldsymbol{x}_1]\!], \ldots, [\![\boldsymbol{x}_n]\!], [\![\boldsymbol{y}_1]\!], \ldots, [\![\boldsymbol{y}_n]\!], \mathbf{M})$, check if tuple $([\![\boldsymbol{x}_i]\!], \boldsymbol{x}_i)$ and $([\![\boldsymbol{y}_i]\!], \boldsymbol{y}_i)$ exists for $i \in [n]$ and that $\boldsymbol{x} = \mathbf{M}\boldsymbol{y}$. If any check fails, abort; otherwise, return $\mathsf{Pass}$.

Figure 2: Functionality of extended SIMD zero-knowledge.

**Plugging in the protocol.** For a general circuit with a total of $|\boldsymbol{w}| = Bm$ wire values and $C = Bn$ multiplication gates, the above approach leads to a zero-knowledge proof of linear map that can be instantiated by any SIMD-ZK. The actual communication complexity depends on the cost of proving the inner product argument by the underlying SIMD-ZK, plus the opening cost of the commitment scheme. Let $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}) \in \mathbb{F}^{nB}$ be the batched wire values of left, right and output of multiplication gates in the circuit. The wire consistency can be proven by checking

$(\boldsymbol{l} \stackrel{?}{=} \mathbf{L}\boldsymbol{w}, \boldsymbol{r} \stackrel{?}{=} \mathbf{R}\boldsymbol{w}, \boldsymbol{o} \stackrel{?}{=} \mathbf{O}\boldsymbol{w})$, where $(\mathbf{L}, \mathbf{R}, \mathbf{O}) \in \mathbb{F}^{nB \times mB}$ are public maps that describe the circuit connectivity. Furthermore, the SIMD-ZK protocol handles the rest of the multiplicative relation check $\boldsymbol{o} \stackrel{?}{=} \boldsymbol{l} * \boldsymbol{r}$. This scheme is captured in the extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ shown in Figure 2. Compared to the common SIMD-ZK functionality shown in Figure 1, it additionally supports the proof of linear map between committed vectors. Based on this extended SIMD-ZK, we propose a compiler that compiles any SIMD-ZK into general ZK. By plugging this compiler to AntMan [WYY+22], it improves its communication complexity from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. In another case, for compressed $\Sigma$-protocols [AC20], this yields a reduction of the CRS size from $\mathcal{O}(C)$ to $\mathcal{O}(\sqrt{C})$ for constant-round sublinear ZK. Eventually, the multi-verifier ZK [YW22] can be improved from $\mathcal{O}(nC/B + n^2B^2)$ to $\mathcal{O}(nC/B + n^2)$.

**Memory constrained prover.** The above construction can be viewed as a compiler that enables a SIMD-ZK to handle arbitrary circuits $\mathcal{C}$, where all wire values fit in a vector $\boldsymbol{w}$ of size $\mathcal{O}(C)$. Assume the linear mapping matrices use succinct representation, the proof requires memory overhead $\mathcal{O}(C)$, which upper bounds the largest circuit that the scheme can prove. We propose a second compiler that further extends the previous idea to the streaming setting, in which the memory overhead is proportional to the plaintext evaluation of the circuit. Furthermore, the whole circuit structure and the witnesses are not required to be known until they are reached. Instead, $\mathcal{P}$ proves the circuit segment-by-segment and only needs to evaluate the current and the previous one at a time: the circuit $\mathcal{C}$ is split into segments $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_{n'})$. For any consecutive segments $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$, let $(\boldsymbol{w}_j, \boldsymbol{l}_j, \boldsymbol{r}_j, \boldsymbol{o}_j)$ and $(\boldsymbol{w}_{j+1}, \boldsymbol{l}_{j+1}, \boldsymbol{r}_{j+1}, \boldsymbol{o}_{j+1})$ be the witness and the input and output wire values of multiplication gates for each segment. $\mathcal{P}$ first uses a commit-and-prove SIMD-ZK to prove the internal satisfiability of $\mathcal{C}_j$ including the linear and multiplicative relations of $(\boldsymbol{w}_j, \boldsymbol{l}_j, \boldsymbol{r}_j, \boldsymbol{o}_j)$. Then $\mathcal{P}$ proves that the output wires of $\mathcal{C}_j$ correctly link to some input wires of $\mathcal{C}_{j+1}$. Namely, it additionally invokes the check of linear map to prove $\mathbf{M}\boldsymbol{w}_j = \tilde{\boldsymbol{w}}_{j+1}$, in which $\mathbf{M}$ is a map that indicates the connectivity between $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ and $\tilde{\boldsymbol{w}}_{j+1}$ are the input wire values of $\mathcal{C}_{j+1}$. After this, $\mathcal{P}$ and $\mathcal{V}$ discard everything for segment $\mathcal{C}_j$ and carry on with the check of internal circuit satisfiability of $\mathcal{C}_{j+1}$. The above step incurs memory overhead $\mathcal{O}(|\boldsymbol{w}_j| + |\tilde{\boldsymbol{w}}_{j+1}|)$. Based on this framework, $\mathcal{P}$ is able to prove the satisfiability of a large circuit by separately evaluating a sequence of smaller circuits.

## 2.2 Improved Commit-and-Prove ZK via SIMD Compiler

Now we show three commit-and-prove SIMD ZK protocols that take advantage of our compilers to perform general ZKP with either reduced online communication complexity or reduced setup cost:

- The aforementioned AntMan [WYY+22] requires $\mathcal{O}(B + C)$ communication for $(B, |\mathcal{C}|)$-SIMD circuit and at least $\mathcal{O}(C^{3/4})$ for a general circuit. Our compiler transforms it into a general VOLE-ZK with communication $\mathcal{O}(C/B + B)$, which is $\mathcal{O}(C^{1/2})$ when $B = \mathcal{O}(C^{1/2})$.

- A constant-round SHVZK argument of knowledge for NP from the discrete logarithm assumption with sublinear communication $\mathcal{O}(C/B + B) = \mathcal{O}(C^{1/2})$ and a CRS of size $\mathcal{O}(B) = \mathcal{O}(C^{1/2})$, where the computation is dominated by $\mathcal{O}(C^{1/2})$ $C^{1/2}$-size Fast Fourier Transforms (FFT). It builds upon the techniques from Attema et al. [AC20] (denoted as AC20) and is combined with a 2-round SHVZK for Hadamard product of [Gro09]. It improves upon a protocol of AC20 which has a CRS of size $\mathcal{O}(C)$ and requires $\mathcal{O}(1)$ $C$-sized FFT. For $(B, \mathcal{C})$-SIMD circuit, our protocol has $\mathcal{O}(C + \sqrt{B}) = \mathcal{O}(C^{1/2})$ communication.

- A non-interactive designated $n$-verifiers ZK based on the packed Shamir secret sharing [YW22, FY92]. Restricting $B < n - 2t$ where $t$ is the number of corrupted verifiers, it incurs $\mathcal{O}(nC)$

communication overhead for $(B, \mathcal{C})$-SIMD circuits and $\mathcal{O}(nC/B + n^2B^2)$ for arbitrary circuit of size $C$, The cost is optimized to $\mathcal{O}(nC/B + n^2)$ with the help of our compiler.

Additionally, we also demonstrate that Ligero [AHIV17] and its follow-up work [BBHV22] perfectly fit our compilers. Although there is no improvement in terms of the proof size or computational complexity, casting Ligero in our framework and using it as a commit-and-prove ZK allows us to identify an important security consideration that would affect both the soundness and zero-knowledge properties.

**Compiling AntMan SIMD-ZK.** The AntMan SIMD-ZK protocol consists of the following key components: 1) a constant-size additive-homomorphic polynomial commitment scheme, 2) a proof of multiplicative relation on committed polynomials, i.e. prove that $f_0(\cdot) = f_1(\cdot) \cdot f_2(\cdot)$. and 3) a proof of degree reduction, i.e. for two polynomials $(f(\cdot), \hat{f}(\cdot))$ with degrees $d_1 < d_2$, $f(i) = \hat{f}(i)$ for $i \in [d_1 + 1]$. We write $[\![f]\!]$ for a commitment to the polynomial $f(\cdot)$. The AntMan protocol realizes $\mathcal{F}_{\mathsf{SIMDZK}}$ as follows:

1. For each batch of $B$ private inputs $\boldsymbol{w}_{\boldsymbol{\alpha}} \in \mathbb{F}^B$, $\mathcal{P}$ computes a degree-$(B-1)$ polynomial $f_{\boldsymbol{\alpha}}$ such that $f_{\boldsymbol{\alpha}}(i) = \boldsymbol{w}_{\boldsymbol{\alpha}}[i]$. $\mathcal{P}$ commits to $f_{\boldsymbol{\alpha}}$ so that $\mathcal{P}$ and $\mathcal{V}$ obtain $[\![f_{\boldsymbol{\alpha}}]\!]$.

2. The parties process the circuit in topological order. For any batch of $k$ addition gates with commitments to input wires $([\![f_{\boldsymbol{\alpha}}]\!], [\![f_{\boldsymbol{\beta}}]\!])$, $\mathcal{P}$ and $\mathcal{V}$ locally computes the commitment to output wires by $[\![f_{\boldsymbol{\gamma}}]\!] = [\![f_{\boldsymbol{\alpha}}]\!] + [\![f_{\boldsymbol{\beta}}]\!]$. For multiplication gates with input commitments $[\![f_{\boldsymbol{\alpha}}]\!]$ and $[\![f_{\boldsymbol{\beta}}]\!]$, $\mathcal{P}$ computes $\boldsymbol{w}_{\boldsymbol{\gamma}} = \boldsymbol{w}_{\boldsymbol{\alpha}} * \boldsymbol{w}_{\boldsymbol{\alpha}}$ and a degree-$(B-1)$ polynomial $f_{\boldsymbol{\gamma}}$ such that $f_{\boldsymbol{\gamma}}(i) = \boldsymbol{w}_{\boldsymbol{\gamma}}[i], i \in [B]$. $\mathcal{P}$ also computes $\hat{f}_{\boldsymbol{\gamma}}(\cdot) = f_{\boldsymbol{\alpha}}(\cdot) \cdot f_{\boldsymbol{\beta}}(\cdot)$. $\mathcal{P}$ commits to them by generating $[\![f_{\boldsymbol{\gamma}}]\!]$ and $[\![\hat{f}_{\boldsymbol{\gamma}}]\!]$.

3. For each multiplication gates with input and output wires $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$, $\mathcal{P}$ proves that $([\![f_{\boldsymbol{\alpha}}]\!], [\![f_{\boldsymbol{\beta}}]\!], [\![\hat{f}_{\boldsymbol{\gamma}}]\!])$ is a multiplication triple and $\hat{f}_{\boldsymbol{\gamma}}(i) = f_{\boldsymbol{\gamma}}(i)$ for $i \in [B]$.

4. When a batch of $k$ output wires $\boldsymbol{\alpha}$, $\mathcal{P}$ opens the commitment to $f_{\boldsymbol{\alpha}}$, from which $\mathcal{V}$ reconstructs $\boldsymbol{w}_{\boldsymbol{\alpha}}$.

The overhead of AntMan SIMD-ZK lies in the commitment of batch circuit intermediate wire values at Step 2, which takes $\mathcal{O}(C)$ for a $(B, \mathcal{C})$-SIMD circuit. The proof of multiplication and degree reduction only incurs $\mathcal{O}(B)$ with random linear combination.

When applying the SIMD compiler to the AntMan SIMD-ZK, it takes $\mathcal{O}(C/B)$ to prove all multiplicative relations for a general circuit of size $C$. Namely, it checks $C$ multiplication triples $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ via SIMD-ZK. Additionally, it invokes the proof of linear map to check the wire consistency between $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ and $\boldsymbol{w}$, which contains intermediate wire values in the circuit. This procedure incurs $\mathcal{O}(B)$ communication overhead at the final commitment opening. Hence, it takes $\mathcal{O}(C/B + B) \geq \mathcal{O}(C^{1/2})$ in total to prove the satisfiability of arbitrary circuits. This protocol is referred as AntMan++. We implemented the AntMan++ and evaluate its performance on proving general circuits of size up to $C = 2^{27}$. It is compared with the prior practical VOLE-based ZK QuickSilver [YSWW21], which requires $\mathcal{O}(C)$ communication overhead. More details are shown in Section 4.1.

**SIMD-ZK based on Pedersen commitment.** We briefly present a SHVZK argument of knowledge for $(B, \mathcal{C})$-SIMD circuits which relies on the techniques of AC20 [AC20]. The key construction of AC20 is a compression mechanism to handle ZK proof for general linear relations (the prover wants to prove the correction of evaluation of a linear form over a committed vector). We expand this technique to obtain a constant-round DLOG-based ZK proof for $(B, \mathcal{C})$-SIMD circuits with $\mathcal{O}(C + \sqrt{B})$ communication. When plugged into our compiler, we get a constant-round circuit ZK with $\mathcal{O}(C^{1/2})$ communication, $\mathcal{O}(C^{1/2})$ CRS size, and with computation dominated by $\mathcal{O}(C^{1/2})$

FFTs of size $\mathcal{O}(C^{1/2})$. It improves over AC20 in both CRS size (from linear to square root) and computation time.

Specifically, for a group of $B$ multiplication gates, we encode the values over all $B$ evaluations on left wire values i.e $\mathbf{x} \in \mathbb{F}^B$ into one polynomial $f$ using pack secret sharing such that $f(0) \xleftarrow{\$} \mathbb{F}, f(i) = x_i$ for $i \in [1, B]$ and commit to it using Pedersen commitment to obtain $[\![f]\!] = \mathbf{g}^{\mathbf{x}'} h^r$ where $\mathbf{x}' := (f(0), \mathbf{x}) \in \mathbb{F}^{B+1}$. The vector of right wire values $\mathbf{y}$ is committed in the same way as $\mathbf{x}$ to get $[\![g]\!]$. For the vector of output wire values $\mathbf{z}$, we define $h(X) := f(X)g(X)$ and $[\![h]\!] = \mathbf{g}^{\mathbf{z}'} h^r$ where $\mathbf{z}' := (h(0), \mathbf{z}, h(B+1), \ldots, h(2B)) \in \mathbb{F}^{2B+1}$. $\mathcal{P}$ can convince $\mathcal{V}$ that $z_i = x_i y_i$ for all $i \in [1, s]$ by revealing $f(c), g(c)$ and $h(c)$ where the challenge $c$ is randomly picked by $\mathcal{V}$. $\mathcal{V}$ now checks $f(c)g(c) \stackrel{?}{=} h(c)$ while $\mathcal{P}$ needs to prove that the revealed values are correct evaluations of $f(X), g(X)$ and $h(X)$ at $c$. This can be handled by using a ZK proof for linear relations since by Langrage formula, $f(c), g(c)$ and $h(c)$ can be expressed as linear form on the committed vectors $\mathbf{x}', \mathbf{y}'$ and $\mathbf{z}'$. Observe that this way, $\mathcal{P}$ can prove correctness of a batch of $B$-tuples of multiplication gates, by showing that the evaluation of many different committed polynomials at a given challenge $c$ is correctly computed. This can be done using an amortized check over many executions, with cost identical to that of a single execution. Using a *sublinear* argument for a batch of $B$-tuples of multiplications, the circuit ZK can be obtained by combing our compiler with an amortized nullity check over one commitment scheme which is used to check the consistency of output gates and also of two different commitments of two vectors of the form $\mathbf{x} \in \mathbb{F}^{B+1}$ and $(\mathbf{x}, \mathsf{aux}) \in \mathbb{F}^{2B+1}$. The details of construction are shown in section 4.2.

**Compiling multi-verifier ZK.** Yang et al. [YW22] proposed an non-interactive designated multi-verifier zero-knowledge proof (MVZK) that allows a prover to prove the correctness of a statement to a set of $n$ honest-majority verifiers. It leverages packed Shamir secret sharing (PSS) [FY92] to support SIMD statements. At a high-level, $\mathcal{P}$ first distributes the witnesses to $\mathcal{V}$ in the form of PSS, then utilize a polynomial compression protocol [GSZ20, BGIN21, BBC+19] to reduce the check of all multiplications into a single multiplication triple. The PSS of witnesses serves as commitments among all $\mathcal{V}$, thus it can be viewed as a commit-and-prove SIMD ZK. Effort is made in [YW22] to convert its SIMD-ZK to general ZK by arranging all wire connection as a tuple $([\![\boldsymbol{w}_1]\!], [\![\boldsymbol{w}_2]\!], i, j)$, indicating that $\boldsymbol{w}_1[i] = \boldsymbol{w}_2[j]$. All tuples with the same $(i, j)$ can be checked in a batch with commitment-opening cost $\mathcal{O}(n^2)$ by a random linear combination. Since $i, j \in [B]$, the total wire consistency check incurs $\mathcal{O}(n^2 B^2)$. However, by applying our SIMD compiler, the overhead for the check is reduced to $\mathcal{O}(n^2)$. The cost to prove multiplicative relations remains $\mathcal{O}(nC/B)$. One caveat is that this protocol is not flexible in choosing the batch size $B$. Assume the maximum number of corrupted verifiers $t < n/2$, it requires that $2t + B < n$ to ensure that honest verifiers have enough shares to determine the result.

**SIMD-ZK from Ligero.** Our compiler is partially inspired by Ligero [AHIV17], an MPC-in-the-head based ZKP [IKOS07] that works for general circuits. At the core of Ligero, $\mathcal{P}$ batch encodes the witness using the Reed-Solomon (RS) coding scheme and commits to each entry of the codewords. $\mathcal{V}$ chooses a subset of entries in codewords, and applies the interleaved RS test, linear constraint test and quadratic constraint test to verify the correctness of encoding, wiring consistency and multiplicative consistency.

We first extract a commit-and-prove SIMD-ZK from Ligero and prove that it realizes $\mathcal{F}_{\mathsf{SIMDZK}}$. Applying our SIMD compiler would result in the original Ligero. We then identify a security issue when applying SIMD Ligero to our memory-constrained framework designed for scalable-ZK. Namely, although Ligero can be turned into a commit-and-prove ZK, its commitment only supports a pre-determined limited number of invocations from the proving procedure. Following

the MPC-in-the-head paradigm, the committing phase mentioned above is equivalent to emulating a $n$-party computation of such circuit, then separately commit to the view of each party among $(P_1, \ldots, P_n)$. During the proving phase, $\mathcal{P}$ opens a subset of $t < n$ views to $\mathcal{V}$, who applies the above mentioned tests. This is fine for one-shot proofs. However, general commit-and-prove ZK does not restrict the number of times that the proving procedure is applied to a commitment. The *zero-knowledge* property can be compromised if the number of opened views exceeds the degree parameter of Reed-Solomon encoding. Although refreshing the commitment solves this issue, a proof of equality across the obsolete and new commitments does not come for free. Our framework covers this issue by adding a counter in $\mathcal{F}_{\mathsf{SIMDZK}}$ to check the usage of each commitment, and abort the proving phase when an input commitment is overused.

# 3 Generic Compiler of ZK Proofs from SIMD Circuits to Arbitrary Circuits

In this section, we first present a construction for extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ which supports the proof of linear map, in addition to the normal SIMD-ZK functionality $\mathcal{F}_{\mathsf{SIMDZK}}$. Based on the extended SIMD-ZK, we describe our compiler that enables a SIMD-ZK scheme to work for general circuits. At last, we present a framework that allows SIMD-ZK schemes to prove large statements with small memory footprints.

## 3.1 Extended SIMD-ZK

The protocol for extended SIMD-ZK is shown in Figure 3, which realizes the functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$. It is based on the $\mathcal{F}_{\mathsf{SIMDZK}}$ functionality to perform the committing and opening of batched wire values, as well as prove the element-wise multiplicative relations between these batches. It takes input a public matrix $\boldsymbol{M} \in \mathbb{F}^{Bn \times Bk}$ and two vectors $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{F}^{Bn}$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k) \in \mathbb{F}^{Bk}$ from $\mathcal{P}$, outputs 1-bit information to $\mathcal{V}$ indicating whether $\boldsymbol{x} = \boldsymbol{M}\boldsymbol{y}$. Essentially, it is a proof of linear map. The first step is to reduce the proof of linear map to a proof of inner products, which is achieved by a random linear combination: $\mathcal{V}$ uniformly samples $\boldsymbol{r} \in \mathbb{F}^{Bn}$ and converts the check of $\boldsymbol{x} \stackrel{?}{=} \boldsymbol{M}\boldsymbol{y}$ into $\boldsymbol{r}^\intercal \boldsymbol{x} \stackrel{?}{=} \boldsymbol{v}^\intercal \boldsymbol{y}$, where $\boldsymbol{v}^\intercal = \boldsymbol{r}^\intercal \boldsymbol{M}$. After dividing these vectors into length-$B$ segments, $\mathcal{P}$ and $\mathcal{V}$ invoke the $\mathcal{F}_{\mathsf{SIMDZK}}$ functionality of batch size $B$. $\mathcal{P}$ inputs $\boldsymbol{q}$ and proves the correctness of $\boldsymbol{q} = \sum_{i=1}^{n} \boldsymbol{r}_i * \boldsymbol{x}_i - \sum_{j=1}^{k} \boldsymbol{v}_i * \boldsymbol{y}_i \in \mathbb{F}^B$. Eventually it opens the commitment to $\boldsymbol{q}$ and let $\mathcal{V}$ check $\sum_{i=1}^{B} \boldsymbol{q}[i] \stackrel{?}{=} 0$. To ensure the privacy of $\mathcal{P}$, it needs to make sure that only opened commitment to $\boldsymbol{q}$ does not reveal information of $\boldsymbol{x}$ and $\boldsymbol{y}$. It does so by the random mask $\tilde{\boldsymbol{r}}$. The impact of this mask on soundness is negligible since it is committed before $\boldsymbol{r}$ is sampled.

In terms of the cost, the protocol $\Pi_{\mathsf{eSIMDZK}}$ takes input $k + n$ vector commitments. During the protocol execution, it additionally commits to $k + n + 1$ size-$B$ vectors. If element-wise product between a public vector and a committed vector is supported the underlying $\mathcal{F}_{\mathsf{SIMDZK}}$, the number of commitments is reduced to 1 size-$B$ vector commitment. Parties invoke the Prove procedure from $\mathcal{F}_{\mathsf{SIMDZK}}$ to prove a $(B, n + k)$-SIMD circuit. $\mathcal{P}$ also opens a size-$B$ vector to $\mathcal{V}$ with cost at most $\mathcal{O}(B)$. The cost is reduced if the underlying SIMD-ZK protocol provides an easier way to prove $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ for a committed vector $\boldsymbol{q}$ without opening the commitment.

**Theorem 1.** *Protocol $\Pi_{\mathsf{eSIMDZK}}$ (Figure 3) securely realizes the Functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{SIMDZK}}$-hybrid model, with soundness error $|\mathbb{F}|^{-1}$.*

*Proof.* We first consider the case of a malicious prover and then the case of a malicious verifier. In each case, we construct a PPT simulator $\mathcal{S}$ given access to functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$, and running

**Protocol $\Pi_{\mathsf{eSIMDZK}}$**

**Inputs:** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold a public matrix $\mathbf{M} \in \mathbb{F}^{Bn \times Bk}$ for some integers $n$ and $k$. Commitments $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{y}]\!]$ are public, where $\boldsymbol{x} \in \mathbb{F}^{Bn}$ and $\boldsymbol{y} \in \mathbb{F}^{Bk}$.
**Protocol:**

1. $\mathcal{P}$ uniformly samples a vector $\tilde{\boldsymbol{r}} \in \mathbb{F}^B$ such that $\sum_{i=1}^{B} \tilde{\boldsymbol{r}}[i] = 0$. Then $\mathcal{F}_{\mathsf{SIMDZK}}$ is invoked to obtain its commitment $[\![\tilde{\boldsymbol{r}}]\!]$.
2. $\mathcal{V}$ uniformly samples a vector $\boldsymbol{r} \in \mathbb{F}^{Bn}$ and sends it to $\mathcal{P}$. Everyone computes $\boldsymbol{v} = \boldsymbol{r}^T\mathbf{M} \in \mathbb{F}^{Bk}$. Then for $i \in [k]$, $\mathcal{F}_{\mathsf{SIMDZK}}$ is invoked to construct $[\![\boldsymbol{v}_i]\!]$, where $\boldsymbol{v}_i$ is the $i$-th B-sized vector of $\boldsymbol{v}$. In the same way, everyone can have access to $\{[\![\boldsymbol{r}_i]\!]\}_{i\in[n]}$.
3. $\mathcal{P}$ computes $\boldsymbol{q} \in \mathbb{F}^B$, such that $\boldsymbol{q}[i] = \sum_{j=1}^{n} \boldsymbol{r}_j[i]\boldsymbol{x}_j[i] - \sum_{j=1}^{k} \boldsymbol{v}_j[i]\boldsymbol{y}_j[i] + \tilde{\boldsymbol{r}}[i]$. $\mathcal{P}$ invokes $\mathcal{F}_{\mathsf{SIMDZK}}$ to obtain $[\![\boldsymbol{q}]\!]$.
4. Define circuit
$$\begin{aligned}&\mathcal{C}_{\mathsf{Lin}}(a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_k, d_1, \ldots, d_k, e, f)\\ &:= \sum_{i=1}^{n} a_i \cdot b_i - \sum_{i=1}^{k} c_i \cdot d_i + e - f,\end{aligned}$$
then call $\mathcal{F}_{\mathsf{SIMDZK}}.\mathsf{Prove}(\mathcal{C}_{\mathsf{Lin}}, [\![\boldsymbol{r}_1]\!], \ldots, [\![\boldsymbol{r}_n]\!], [\![\boldsymbol{x}_1]\!], \ldots, [\![\boldsymbol{x}_n]\!],$
$[\![\boldsymbol{v}_1]\!], \ldots, [\![\boldsymbol{v}_k]\!], [\![\boldsymbol{y}_1]\!], \ldots, [\![\boldsymbol{y}_k]\!], [\![\tilde{\boldsymbol{r}}]\!], [\![\boldsymbol{q}]\!])$.
5. $\mathcal{V}$ sends $(\mathsf{Open}, [\![\boldsymbol{q}]\!])$ to $\mathcal{F}_{\mathsf{SIMDZK}}$, which returns $\boldsymbol{q}$ to $\mathcal{V}$; $\mathcal{V}$ checks $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ and aborts if the check fails.

Figure 3: The protocol for extended SIMD ZK from SIMD ZK.

a PPT adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}_{\mathsf{SIMDZK}}$ for $\mathcal{A}$. We show that no PPT environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

**Malicious prover.** The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of $\Pi_{\mathsf{eSIMDZK}}$ as follows:

1. By emulating the $(\mathsf{Commit})$ command of $\mathcal{F}_{\mathsf{SIMDZK}}$, $\mathcal{S}$ receives $\tilde{\boldsymbol{r}}$ from $\mathcal{A}$ and sends a handler $[\![\tilde{\boldsymbol{r}}]\!]$ to $\mathcal{A}$.
2. $\mathcal{S}$ uniformly samples $\boldsymbol{r} \in \mathbb{F}^{Bn}$ and sends to $\mathcal{A}$. For $i \in [k]$, after receiving $(\mathsf{Commit}, \boldsymbol{v}_i)$ from $\mathcal{A}$, $\mathcal{S}$ sends a handler $[\![\boldsymbol{v}_i]\!]$ to $\mathcal{A}$. Similarly, $\mathcal{S}$ sends $[\![\boldsymbol{r}_i]\!]$ to $\mathcal{A}$ for $i \in [n]$.
3. After receiving $(\mathsf{Commit}, \boldsymbol{q})$ from $\mathcal{A}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{SIMDZK}}$ by sending $\mathcal{A}$ another handler $[\![\boldsymbol{q}]\!]$.
4. $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \ldots, \tau_{2n+2k+2})$ from $\mathcal{A}$ and checks whether $\{\tau_i\}_{i\in[2n+2k+2]}$ match their corresponding tags. Then, for $i \in [B]$, $\mathcal{S}$ checks whether $\sum_{j=1}^{n} \boldsymbol{r}_j[i]\boldsymbol{x}_j[i] - \sum_{j=1}^{k} \boldsymbol{v}_j[i]\boldsymbol{y}_j[i] + \tilde{\boldsymbol{r}}[i] - \boldsymbol{q}[i]$ equals to 0 or not. If any check fails, $\mathcal{S}$ aborts; otherwise sends $\mathsf{Pass}$ to $\mathcal{A}$.
5. $\mathcal{S}$ emulates the $(\mathsf{Open})$ command of $\mathcal{F}_{\mathsf{SIMDZK}}$ and receives a handler $\tau$ from $\mathcal{A}$. If $\tau$ does not match $[\![\boldsymbol{q}]\!]$ or the vector $\boldsymbol{q}$ previously sent by $\mathcal{A}$ does not satisfy $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$, $\mathcal{S}$ aborts.

Define $E$ to be the event that a cheating prover $\mathcal{A}$ successfully convinces $\mathcal{V}$ in the real world.

This happens when $\boldsymbol{r}$ accidentally corrects the wrong input of $\mathcal{A}$. Define $\boldsymbol{z} = \mathbf{M}\boldsymbol{y}$ and

$$f(x_1, \ldots, x_{Bn}) = \sum_{i=1}^{Bn} x_i(\boldsymbol{x}[i] - \boldsymbol{z}[i]) + \sum_{i=1}^{B} \tilde{\boldsymbol{r}}[i].$$

With fixed $\boldsymbol{x}, \boldsymbol{z}, \tilde{\boldsymbol{r}}$ and uniformly sampled $\boldsymbol{r}$, we have

$$\Pr[E|\boldsymbol{x} \neq \mathbf{M}\boldsymbol{y}] = \Pr[f(\boldsymbol{r}) = 0|\boldsymbol{x} \neq \mathbf{M}\boldsymbol{y}] = |\mathbb{F}|^{-1}.$$

since $f(x_1, \ldots, x_{Bn})$ is a $Bn$-variate degree-1 polynomial. Hence we conclude that $\mathcal{A}$ cannot distinguish between the real and ideal world except with probability $|\mathbb{F}|^{-1}$.

**Malicious verifier.** Similarly in this case, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. To emulate the (Commit) command, $\mathcal{S}$ sends a handler $[\![\tilde{\boldsymbol{r}}]\!]$ to $\mathcal{A}$.
2. $\mathcal{S}$ recieves $\boldsymbol{r}$ and $(\mathsf{Commit}, \boldsymbol{v}_i)$ from $\mathcal{A}$ for $i \in [k]$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{SIMDZK}}$ by sending $\mathcal{A}$ a handler $[\![\boldsymbol{v}_i]\!]$ for $i \in [k]$. In the same way, $\mathcal{S}$ sends $\mathcal{A}$ handlers $\{[\![\boldsymbol{r}_i]\!]\}_{i\in[n]}$.
3. Then, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{SIMDZK}}$ and sends a handler $[\![\boldsymbol{q}]\!]$ to $\mathcal{A}$.
4. $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \ldots, \tau_{2n+2k+2})$ from $\mathcal{A}$ and checks whether $\{\tau_i\}_{i\in[2n+2k+2]}$ match their corresponding tags. Then $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{eSIMDZK}}$. If check fails or $\mathcal{F}_{\mathsf{eSIMDZK}}$ aborts, $\mathcal{S}$ aborts; otherwise sends Pass to $\mathcal{A}$.
5. By emulating the (Open) command of $\mathcal{F}_{\mathsf{SIMDZK}}$, $\mathcal{S}$ uniformly samples a vector $\boldsymbol{q} \in \mathbb{F}^B$ such that $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ and sends $\boldsymbol{q}$ to $\mathcal{A}$.

The only difference between reality and the ideal world is the method of calculating vector $\boldsymbol{q}$. Following the constraint $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$, $\mathcal{S}$ uniformly samples vector $\boldsymbol{q}$. While in reality, each entry of $\boldsymbol{q}$ is masked by vector $\tilde{\boldsymbol{r}}$ chosen by $\mathcal{P}$. As a result, in both worlds, all entries except one of $\boldsymbol{q}$ are information-theoretic secure, so no one can distinguish one from another.

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

### 3.2 Compiling Extended SIMD-ZK

The general approach to compile a SIMD protocol into a generic protocol is to supplement it with an additional proof of wiring consistency. Namely, denote $\boldsymbol{w}$ as a vector that includes all the wire values in a circuit, then any input wire of a multiplication gate can be represented as the linear combination of a series of values in $\boldsymbol{w}$, who are the wire values that connect from the circuit inputs or the output of other gates. This relation can be generally represented as a linear map $\boldsymbol{M}$ between a vector of wire values $\boldsymbol{x}$, and $\boldsymbol{w}$, which should satisfy $\boldsymbol{x} = \boldsymbol{M}\boldsymbol{w}$. As shown in Figure 4, along with the vector $\boldsymbol{w}$, $\mathcal{P}$ also commits to $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ which are the batches of input and output wire values of multiplication gates. Showing that $\boldsymbol{o} = \boldsymbol{l} * \boldsymbol{r}$ is enough to prove that all multiplication gates are computed correctly. Additionally, $\mathcal{P}$ also proves the correctness of $(\boldsymbol{l} = \boldsymbol{L}\boldsymbol{w}, \boldsymbol{r} = \boldsymbol{R}\boldsymbol{w}, \boldsymbol{o} = \boldsymbol{O}\boldsymbol{w})$, in which $(\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O})$ are the linear maps that defines the routing of wires that connects to the input and output wires of multiplication gates. Additionally, the proof of $\mathbf{0} = \boldsymbol{A}\boldsymbol{w}$ shows the correct computation of all addition gates.

To handle a general circuit $\mathcal{C}$, our compiler fully depends on the extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$. Regarding the cost analysis, $\mathcal{P}$ commits to a total of $k + 3n_2$ size-$B$ vectors to $\mathcal{V}$. They

**Protocol $\Pi_{\mathsf{compiler}}$**

**Inputs:** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold an arbitrary circuit $\mathcal{C}$ over a large field $\mathbb{F}$, where $\mathcal{C}$ contains $N_1 = Bn_1$ addition gates, $N_2 = Bn_2$ multiplication gates and $K = Bk$ wires for some $n_1, n_2$ and $k$.
**Protocol:**

1. Set $c = 1$. For each gate in the form $(i, \alpha, \beta, \gamma, T)$
   - If $T = ADD$, set $\mathbf{A}_i := \mathbf{I}_\alpha + \mathbf{I}_\beta - \mathbf{I}_\gamma$; $\mathcal{P}$ sets $\boldsymbol{w}[\gamma] := \boldsymbol{w}[\alpha] + \boldsymbol{w}[\beta]$
   - If $T = MULT$, set $(\mathbf{L}_c, \mathbf{R}_c, \mathbf{O}_c) := (\mathbf{I}_\alpha, \mathbf{I}_\beta, \mathbf{I}_\gamma)$; $\mathcal{P}$ sets $(\boldsymbol{l}[c], \boldsymbol{r}[c], \boldsymbol{o}[c]) =: (\boldsymbol{w}[\alpha], \boldsymbol{w}[\beta], \boldsymbol{w}[\alpha] \cdot \boldsymbol{w}[\beta])$. Increase $c$ by 1.

   After the circuit is processed, matrix $\mathbf{L}, \mathbf{R}, \mathbf{O} \in \mathbb{F}^{N_2 \times K}$, and $\mathbf{A} \in \mathbb{F}^{N_1 \times K}$ are public; $\mathcal{P}$ has $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}, \boldsymbol{w}) \in \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^{K}$.
2. $\mathcal{P}$ splits wire values ($\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{o}$, $\boldsymbol{w}$) into chunks of size $B$, i.e., $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i\in[n_2]}$ and $\{\boldsymbol{w}_i\}_{i\in[k]}$, such that each element is in $\mathbb{F}^B$. $\mathcal{F}_{\mathsf{eSIMDZK}}$ is invoked to obtain commitments $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i\in[n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$.
3. Then, $(\mathsf{LinearMap}, \{[\![\boldsymbol{l}_i]\!]\}_{i\in[n_2]}, \{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$ to check that $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$; similarly check that $\boldsymbol{r} = \mathbf{R}\boldsymbol{w}$, $\boldsymbol{o} = \mathbf{O}\boldsymbol{w}$, and that $\mathbf{0} = \mathbf{A}\boldsymbol{w}$.
4. Let circuit $\mathcal{C}_{\mathsf{Mult}} : \mathbb{F}^3 \to \mathbb{F}$ such that $\mathcal{C}_{\mathsf{Mult}}(x, y, z) := xy - z$. For $i \in [n_2]$, send $(\mathsf{Prove}, \mathcal{C}_{\mathsf{Mult}}, [\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!])$ to $\mathcal{F}_{\mathsf{eSIMDZK}}$.

Figure 4: Generic ZK in the $\mathcal{F}_{\mathsf{eSIMDZK}}$ hybrid.

invoke the proof of linear map for 4 times to prove the wiring consistency, and the proof of element-wise multiplication to prove the correctness of $n_2$ batches of multiplication gates. An optimization to reduce the cost for the proof of linear map is to combine the 4 of them into 1. Namely, define $\boldsymbol{w}'$ to be the wire values excluding the input and output wires of multiplication gates. Construct the witness vector $\boldsymbol{w} = (\boldsymbol{w}' \| \boldsymbol{l} \| \boldsymbol{r} \| \boldsymbol{o})$ and prove the wiring consistency by proving $\mathbf{0} = \mathbf{A}'\boldsymbol{w}$, in which $\mathbf{A}' \in \mathbb{F}^{K \times K}$ is a map that describes the circuit wire connectivity.

**Theorem 2.** *The Protocol $\Pi_{\mathsf{compiler}}$ (Figure 4) securely realizes the Functionality $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{eSIMDZK}}$-hybrid model, with 0 soundness error.*

*Proof.* Similarly, we construct a PPT simulator in two cases and argue that no PPT environment $\mathcal{Z}$ can distinguish reality and the ideal world.

**Malicious prover.** The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of $\Pi_{\mathsf{compiler}}$ as follows:

1. Following the protocol specification, $\mathcal{S}$ obtain matrix $\mathbf{L}, \mathbf{R}, \mathbf{O}$ and $\mathbf{A}$ from circuit $\mathcal{C}$.
2. By emulating the ($\mathsf{Commit}$) command of $\mathcal{F}_{\mathsf{eSIMDZK}}$, $\mathcal{S}$ receives $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i\in[n_2]}$ and $\{\boldsymbol{w}_i\}_{i\in[k]}$ from $\mathcal{A}$ and sends $\mathcal{A}$ handlers $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i\in[n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$.
3. After receiving $(\mathsf{LinearMap}, \{\tau_i\}_{i\in[n_2+k]}, \mathbf{L})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether $\{\tau_i\}_{i\in[n_2]}$ match $\{[\![\boldsymbol{l}_i]\!]\}_{i\in[n_2]}$ and $\{\tau_i\}_{i\in[n_2+1,n_2+k]}$ matches $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$. Then, $\mathcal{S}$ checks whether $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$. If any check fails, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends $\mathsf{Pass}$ to $\mathcal{A}$. Similarly, $\mathcal{S}$ handles other three ($\mathsf{LinearMap}$) commands from $\mathcal{A}$.
4. For $i \in [n_2]$, $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \tau_2, \tau_3)$ from $\mathcal{A}$ and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}$. In each round, $\mathcal{S}$ also checks that $\boldsymbol{l}_i[j] \cdot \boldsymbol{r}_i[j] = \boldsymbol{o}_i[j]$ for $j \in [B]$. If any

check fails, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$.

It is trivial that $\mathcal{S}$ is perfect, since whenever an ideal functionality is called in the protocol, $\mathcal{S}$ acts exactly the same as the definition of the functionality. On the other hand, if the witness indeed satisfies linear as well as the multiplication constraints, we can conclude that it satisfies circuit $\mathcal{C}$. Given the perfectness of the ideal functionality, we can conclude that the soundness error is 0.

**Malicious verifier.** The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of $\Pi_{\mathsf{compiler}}$ as follows:

1. $\mathcal{S}$ follows the protocol specification and obtain matrix $\mathbf{L}, \mathbf{R}, \mathbf{O}$ and $\mathbf{A}$ from circuit $\mathcal{C}$.
2. By emulating the (Commit) command of $\mathcal{F}_{\mathsf{eSIMDZK}}$, $\mathcal{S}$ sends $\mathcal{A}$ handlers $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i\in[n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$.
3. After receiving $(\mathsf{LinearMap}, \{\tau_i\}_{i\in[n_2+k]}, \mathbf{L})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether $\{\tau_i\}_{i\in[n_2]}$ match $\{[\![\boldsymbol{l}_i]\!]\}_{i\in[n_2]}$ and $\{\tau_i\}_{i\in[n_2+1,n_2+k]}$ matches $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$. Then, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{ZK}}$. If check fails or $\mathcal{F}_{\mathsf{ZK}}$ aborts, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$. Similarly, $\mathcal{S}$ handles other three (LinearMap) command from $\mathcal{A}$.
4. For $i \in [n_2]$, $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \tau_2, \tau_3)$ from $\mathcal{A}$ and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}$. In each round, $\mathcal{S}$ also queries $\mathcal{F}_{\mathsf{ZK}}$. If any check fails or $\mathcal{F}_{\mathsf{ZK}}$ aborts, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$.

Similarly, since $\mathcal{S}$ acts according to the definition of the ideal functionality and there is no commitment opening during the protocol, the simulation is perfect.

As a result, no PPT environment $\mathcal{Z}$ can distinguish between the real-world scenario and the ideal-world execution, which completes the proof.

### 3.3 Generic ZK for Limited-Memory

Besides a basic-version compiler, we also present another compiler that can deal with a situation where the prover's memory is limited. Although a similar question has already been proposed before [BCCT13, GGPR13, PS04, KST22], our construction does not rely on any complicated assumption other than the realizatin of $\mathcal{F}_{\mathsf{eSIMDZK}}$ with the parameter $\tau_{\mathsf{max}} > 1$. The protocol is shown in Figure 5. We take the advantage of the commit-and-prove paradigm: instead of proving the whole circuit at one time, circuit can be "partially" proved. The value of wires that connect between different parts of the circuit can be reserved as commitments and used for the proof of connectivity. Specifically, prover will clarify a space threshold parameter $S$ before the proof, and the original circuit $\mathcal{C}$ will be divided into $\lceil |\mathcal{C}|/S \rceil$ parts (denoted as $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{\lceil |\mathcal{C}|/S \rceil}$), where each part contains at most $S$ gates. In each round, $S$ gates of $\mathcal{C}_i$ will be read and processed in the memory, and $\mathcal{P}$ generates the proof for $\mathcal{C}_i$. At the end of each round, $\mathcal{P}$ commits to a vector which contains all the wire values that are still active in $\mathcal{C}_{i+1}$, and discards those that won't be used in the remaining circuit.

To support this pruning operation, we add a $DEL$ gate to the encoding of the circuit. $\mathcal{P}$ reads the circuit from a stream of $(\alpha, \beta, \gamma, T)$, where $T \in \{ADD, MULT, DEL\}$. If $T \in \{ADD, MULT\}$, $\mathcal{P}$ processes gates $\alpha, \beta, \gamma$ similarly as the previous compiler. If $T = DEL$, $\mathcal{P}$ adds gate $\alpha$ to the set $\mathcal{D}$, which contains all the wire values that no longer appear in the next segment of the circuit. After the proof of consistency inside $\mathcal{C}_i$, $\mathcal{P}$ forms a new commitment to wire values that are not in the set $\mathcal{D}$. By applying $\mathcal{F}_{\mathsf{eSIMDZK}}$.LinearMap, $\mathcal{P}$ proves that the committed wire values belongs

**Protocol $\Pi_{\mathsf{small-space}}$**

**Inputs:** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold an arbitrary circuit $\mathcal{C}$ over a large field $\mathbb{F}$, and a space threshold parameter $S = sB$ for some integer $s$. $\mathcal{P}$ holds the secret input $\boldsymbol{x}$ such that $\mathcal{C}(\boldsymbol{x}) = 0$.
**Protocol:**

1. Let $h() : \mathbb{Z} \to \mathbb{Z}$ be a function map wire indices to physical indices and $\boldsymbol{w}$ be a dynamic list storing wire value to be dealt with in the current round. Initially, set $h(i) = i$ and $\boldsymbol{w}[i] = \boldsymbol{x}[i]$ for all $i \in [|\boldsymbol{x}|]$. Define function $\mathsf{Im}() : f \to \mathbb{Z}$, returning the maximum index that $f$ has the definition.
2. Let $\mathcal{D} = \emptyset$ and $W = \mathsf{Im}(h) + S$. Initialize **L, R, O, A** to be empty matrices. Read the next $S$ gates to the memory (or until the last gate). For each in the form $(\alpha, \beta, \gamma, T)$:
   - If $T = ADD$, set $h(\gamma) := \mathsf{Im}(h) + 1$, compute $\boldsymbol{r} := \mathbf{I}_{h(\alpha)} + \mathbf{I}_{h(\beta)} - \mathbf{I}_{h(\gamma)} \in \mathbb{F}^W$ and append $\boldsymbol{r}$ to $\mathbf{A}$. $\mathcal{P}$ sets $\boldsymbol{w}[h(\gamma)] := \boldsymbol{w}[h(\alpha)] + \boldsymbol{w}[h(\beta)]$
   - If $T = MULT$, set $h(\gamma) := \mathsf{Im}(h) + 1$, append rows in $\mathbb{F}^W$ $\mathbf{I}_{h(\alpha)}, \mathbf{I}_{h(\beta)}, \mathbf{I}_{h(\gamma)}$ to matrices $\mathbf{L}, \mathbf{R}, \mathbf{O}$ respectively. $\mathcal{P}$ sets $\boldsymbol{w}[h(\gamma)] := \boldsymbol{w}[h(\alpha)] \cdot \boldsymbol{w}[h(\beta)]$ and append values $\boldsymbol{w}[h(\alpha)], \boldsymbol{w}[h(\beta)], \boldsymbol{w}[h(\gamma)]$ to vectors $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}$ respectively.
   - If $T = DEL$, add $\alpha$ to $\mathcal{D}$.

   Suppose that there are $S_1 = s_1 B$ addition gates and $S_2 = s_2 B$ multiplication gates ($S = S_1 + S_2$), and after processing $S$ gates, $|\boldsymbol{w}| = kB$. $\mathbf{A} \in \mathbb{F}^{S_1 \times W}$ and $\mathbf{L}, \mathbf{R}, \mathbf{O} \in \mathbb{F}^{S_2 \times W}$ are public.
3. $\mathcal{P}$ splits wire values ($\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{o}$, $\boldsymbol{w}$) into chunks of size $B$, i.e., $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i \in [s_2]}$ and $\{\boldsymbol{w}_i\}_{i \in [k]}$, such that each element is in $\mathbb{F}^B$. $\mathcal{F}_{\mathsf{eSIMDZK}}$ is invoked to obtain commitments $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i \in [s_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$.
4. Then, $(\mathsf{LinearMap}, \{[\![\boldsymbol{l}_i]\!]\}_{i \in [n_2]}, \{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$ to check that $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$; similarly check that $\boldsymbol{r} = \mathbf{R}\boldsymbol{w}$, $\boldsymbol{o} = \mathbf{O}\boldsymbol{w}$, and that $\mathbf{0} = \mathbf{A}\boldsymbol{w}$.
5. Let circuit $\mathcal{C}_{\mathsf{Mult}} : \mathbb{F}^3 \to \mathbb{F}$ such that $\mathcal{C}_{\mathsf{Mult}}(x, y, z) := xy - z$. For $i \in [s_2]$, $(\mathsf{Prove}, \mathcal{C}_{\mathsf{Mult}}, [\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!])$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$.
6. Let $\mathcal{R} = \mathsf{Domain}(h) \setminus \mathcal{D}$. Suppose that $|\mathcal{R}| = k'$. For the $i$-th element in $\mathcal{R}$, let $h'(\mathcal{R}[i]) = i$, and set the $i^{th}$ row of $\mathbf{H}$ as $\mathbf{I}_{h(\mathcal{R}[i])}$.
7. $\mathcal{P}$ computes $\boldsymbol{w}'$ such that for each $\boldsymbol{w}'[h'(i)] = \boldsymbol{w}[h(i)]$. Append 0 to $\boldsymbol{w}'$ and $\mathbf{0}$ to $\mathbf{H}$ until the size of $\boldsymbol{w}'$ becomes a multiple of $B$. Supppose that $|\boldsymbol{w}'| = k'B$, and then $\mathcal{P}$ calls $\mathsf{Commit}$ to obatin $\{[\![\boldsymbol{w}'_i]\!]\}_{i \in [k']}$. Update $(h, \boldsymbol{w}) := (h', \boldsymbol{w}')$.
8. Both parties call $(\mathsf{LinearMap}, \{\boldsymbol{w}'_i\}_{i \in [k']}, \{\boldsymbol{w}_i\}_{i \in [k]}, \mathbf{H})$ to check the consistency between $\boldsymbol{w}$ and $\boldsymbol{w}'$.
9. If more gates need to be processed, jump to step 2.

Figure 5: Generic ZK in limited-memory scenario.

to the output wires of $\mathcal{C}_i$, which are also the input of $\mathcal{C}_{i+1}$. $\mathcal{P}$ and $\mathcal{V}$ repeat this procedure for the proof of each segment.

Now we claim that if the plaintext evaluation of circuit $\mathcal{C}$ requires memory space $M$, then in our protocol, the prover's space complexity is $\mathcal{O}(M)$. Denote $\boldsymbol{o}_i$ as the output of subcircuit $\mathcal{C}_i$, and circuit input $\boldsymbol{x}$ is denoted as $\boldsymbol{o}_0$. In each round, we call $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{Prove}$ to complete the proof for $\mathcal{C}_i$ and $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$ to prove the transformation between $[\![\boldsymbol{o}_{i-1}]\!]$ and $[\![\boldsymbol{o}_i]\!]$. As each subcircuit contains at most $S$ gates, proving $\mathcal{C}_i$ requires $\mathcal{O}(S)$ space. And also, using $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$ to prove the consistency between $[\![\boldsymbol{o}_{i-1}]\!]$ and $[\![\boldsymbol{o}_i]\!]$ requires $\mathcal{O}(|\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|)$ space, so the space complexity of each round is $\mathcal{O}(S + |\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|)$. As a result, the overall space complexity is $\mathcal{O}(S + \max\{|\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|\}_{i \in [\lceil |\mathcal{C}|/S \rceil]})$. Since in the plaintext evaluation of $\mathcal{C}$, only active wire value

**Protocol $\Pi_{\mathsf{AntMan}}$**

**Public inputs.** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold a general circuit $\mathcal{C}$ over a large field $\mathbb{F}$, where $\mathcal{C}$ contains $n = |\mathcal{C}|$ multiplication gates and $m$ input gates. Let $\alpha_1, \ldots, \alpha_B \in \mathbb{F}$ be $B$ distinct elements that are fixed for the whole protocol execution. Both parties invoke $\mathsf{Initialize}()$ in IT-PAC to obtain $\tau_1$.

**Private input.** $\mathcal{P}$ holds $m$ witnesses $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m \in \mathbb{F}^B$ such that $\mathcal{C}(\boldsymbol{w}_1[i], \ldots, \boldsymbol{w}_m[i]) = 0$ for all $i \in [B]$.

**Commit:** On input $\boldsymbol{w} \in \mathbb{F}^B$, $\mathcal{P}$ computes polynomial $f(\cdot) = \sum_{i=0}^{B-1} f_i \cdot X^i$ such that for $i \in [B]$, $f(\alpha_i) = w_i$. Both parties invoke $(\langle b \rangle, \tau_2) \leftarrow \mathsf{PreGen}(f)$. $\mathcal{P}$ obtains $\langle b \rangle$ and $\mathcal{V}$ obtains $\tau_2$. If $\Lambda$ has been revealed, invoke $(\mathsf{M}, \mathsf{K}) \leftarrow \mathsf{Gen}(\tau_1, \tau_2)$. $\mathcal{P}$ holds $\mathsf{M}$ and $\mathcal{V}$ holds $\mathsf{K}$.

**Open:** On input $([\![f(\cdot)]\!], f(\cdot), \Lambda)$, both parties compute that $[\![\mu]\!] := [\![f(\Lambda)]\!] - f(\Lambda)$. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. $\mathcal{P}$ sends $\mathsf{H}(\mathsf{M}_\mu)$ to $\mathcal{V}$ who checks whether $\mathsf{H}(\mathsf{M}_\mu) = \mathsf{H}(\mathsf{K}_\mu)$.

Figure 6: The protocol of $\mathsf{SIMDZK}$ from AntMan.

needs to be read into the memory, memory upper bound $M \geq \max\{|\boldsymbol{o}_0|, |\boldsymbol{o}_1|, |\boldsymbol{o}_2|, \ldots, |\boldsymbol{o}_{\lceil |\mathcal{C}|/S \rceil}|\}$. By choosing $S < M$, we can conclude that the space complexity of the protocol is $\mathcal{O}(M)$.

# 4 Efficient Instantiations of Our Compiler

The only assumption that our general compiler described in Section 3.2 makes is that the underlying ZKP realizes the extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$. The compiler for scalable ZK described in Section 3.3 only additionally requires the parameter $\tau_{max} > 1$ for $\mathcal{F}_{\mathsf{eSIMDZK}}$. In this section, we show three instantiations of SIMD-ZK that benefits from these compilers, including

- A ZKP based on vector oblivious linear evaluation [WYY+22].
- A zk-SNARK from $\Sigma$-protocol [AC20].
- A designated multi-verifier ZKP based on packed Shamir secret sharing and recursive inner product check [YW22].

All of these works are in the form of SIMD-ZK and originally require significant extra effort to be converted into a general ZK. Our compilers are able to transform them into general ZK with decrease in their proof size or setup cost. The only exception is the AntMan [WYY+22] which is restricted by $\tau_{max} = 1$ thus does not fit into the second compiler for scalable ZK. In Supplementary Material A.2 and A.3, we additionally describe a SIMD-ZK that is extracted from an MPC-in-the-head scheme Ligero [AHIV17], and a construction from LegoSNARK [CFQ19]. Both our compilers are generalizations of them and a follow-up work [BBHV22]. We show that Ligero SIMD-ZK perfectly fits our compilers and discuss extra caution that need to take when compiling Ligero.

## 4.1 AntMan++: Sublinear Designated-Verifier ZK

AntMan [WYY+22] is a sublinear VOLE-based ZK proof for SIMD circuits, which only requires communicating $\mathcal{O}(B + |\mathcal{C}|)$ field elements to prove a $(B, \mathcal{C})$-SIMD circuit. It also presents a construction for proving a single execution of an arbitrary circuit, by breaking down the circuits into individual gates and batching them as SIMD circuits. The proving of SIMD circuits requires sending $\mathcal{O}(|\mathcal{C}|/B + B)$ field elements, and the cost to check the wire-value consistency is $\mathcal{O}(B^3)$, which leads to $\mathcal{O}(|\mathcal{C}|^{3/4})$ communication complexity in optimal. It is the only sublinear-communication VOLE-ZK protocol for proving an arbitrary circuit. In AntMan [WYY+22], the information-theoretic

polynomial authentication code $\Pi^k_{\mathsf{IT\text{-}PAC}}$ servers as a polynomial commitment scheme. For arbitrary degree-$k$ polynomial $f(\cdot)$ known by $\mathcal{P}$, an IT-PAC $[\![f(\cdot)]\!]$ consists of a MAC $M \in \mathbb{F}$ known by $\mathcal{P}$ and a tuple of keys $(K, \Delta, \Lambda) \in \mathbb{F}^3$ known by $\mathcal{V}$, such that $M = K + f(\Lambda) \cdot \Delta$. In the following, we first detail the commitment scheme used in the AntMan protocol, then discuss how to enable AntMan to prove arbitrary circuits.

**Information-theoretic polynomial authentication code $\Pi_{\mathsf{IT-PAC}}$.** As shown in Figure 7, the protocol is designed in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model. It adopts additively homomorphic encryption (AHE) scheme to obliviously evaluate a polynomial, where the polynomial is known by $\mathcal{P}$ and the secret point $\Lambda$ is known by $\mathcal{V}$. Then VOLE correlations further transform such oblivious polynomial evaluation (OPE) into IT-PACs. A critical issue is to guarantee that the HE ciphertext which encodes the evaluation point $\Lambda$ is correct. Instead of using the zero-knowledge proof of knowledge for the proof of validity (as done in several MPC protocols [KPR18, DPSZ12]), AntMan utilizes a simple commit-and-open approach. Specifically, $\mathcal{V}$ first commits to the randomness that are used to generate the HE ciphertexts $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$. After receiving HE ciphertexts from $\mathcal{V}$, $\mathcal{P}$ performs the homomorphic evaluation and commits to all of HE ciphertexts $\langle b \rangle$ that it should send to $\mathcal{V}$ for OPE. Then $\mathcal{V}$ opens the randomness and let $\mathcal{P}$ check the correctness of $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$. If they are valid, $\mathcal{P}$ opens $\langle b \rangle$ to continue with the execution of OPE. This allows the AntMan protocol to remove the possible leakage of secret polynomials, which is incurred by homomorphically performing polynomial evaluation upon incorrect ciphertexts.

**Compiling AntMan.** The AntMan protocol that realizes $\mathcal{F}_{\mathsf{SIMDZK}}$ is shown in Figure 6 (derived from the original AntMan [WYY+22]). Below we show how to compile it into a ZK protocol that proves the satisfiability of a single general circuit with sublinear communication, following the protocol of our compiler $\Pi_{\mathsf{compiler}}$.

Given a public circuit $\mathcal{C}$, both parties scan the circuit following the procedure in figure 4 to calculate wire rearrangement matrices $\mathbf{L}, \mathbf{R}, \mathbf{O}$, as well as matrix $\mathbf{A}$ that describes addition gates. Then $\mathcal{P}$ splits wire values $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}, \boldsymbol{w})$ into chunks of size $B$, and calls $\mathsf{Commit}$ to obtain $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i \in [n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$. To prove $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$:

1. $\mathcal{P}$ uniformly samples a vector $\tilde{\boldsymbol{r}} \in \mathbb{F}^B$ such that $\sum_{i=1}^{B} \tilde{\boldsymbol{r}}[i] = 0$, and calls $\mathsf{Commit}$ to obtain $[\![\tilde{\boldsymbol{r}}]\!]$.

2. $\mathcal{V}$ uniformly samples a vector $\boldsymbol{r} \in \mathbb{F}^{Bn_2}$ and sends it to $\mathcal{P}$. Both parties compute $\boldsymbol{v} = \boldsymbol{r}^T \mathbf{L} \in \mathbb{F}^{Bk}$ and call $\mathsf{Commit}$ to obtain $\{[\![\boldsymbol{v}_i]\!]\}_{i \in [k]}$ and $\{[\![\boldsymbol{r}_i]\!]\}_{i \in [n_2]}$.

3. $\mathcal{P}$ computes $\boldsymbol{q} \in \mathbb{F}^B$, such that

$$\boldsymbol{q}[i] = \sum_{j=1}^{n_2} \boldsymbol{r}_j[i]\boldsymbol{l}_j[i] - \sum_{j=1}^{k} \boldsymbol{v}_j[i]\boldsymbol{w}_j[i] + \tilde{\boldsymbol{r}}[i].$$

   $\mathcal{P}$ calls $\mathsf{Commit}$ to obtain $[\![\boldsymbol{q}]\!]$.

4. Define circuit

$$\begin{aligned} &\mathcal{C}_{\mathsf{Lin}}(a_1, \ldots, a_{n_2}, b_1, \ldots, b_{n_2}, c_1, \ldots, c_k, d_1, \ldots, d_k, e, f) \\ &:= \sum_{i=1}^{n_2} a_i \cdot b_i - \sum_{i=1}^{k} c_i \cdot d_i + e - f, \end{aligned}$$

   then call $\mathsf{Prove}(\mathcal{C}_{\mathsf{Lin}}, [\![\boldsymbol{r}_1]\!], \ldots, [\![\boldsymbol{r}_{n_2}]\!], [\![\boldsymbol{l}_1]\!], \ldots, [\![\boldsymbol{l}_{n_2}]\!], [\![\boldsymbol{v}_1]\!], \ldots,$ $[\![\boldsymbol{v}_k]\!], [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_k]\!], [\![\tilde{\boldsymbol{r}}]\!], [\![\boldsymbol{q}]\!])$.

**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 $\mathsf{par} = \mathsf{Setup}(1^\lambda)$ and global key $\Delta \in \mathbb{F}$. Let $\mathsf{G}$ be a PRG. Let $k$ be the maximum degree of the polynomials committed in each IT-PAC.

**Initialize.**

1. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$, and then $\mathcal{V}$ and $\mathcal{P}$ call the ($\mathsf{Commit}$) command of $\mathcal{F}_{\mathsf{Com}}$ with input $\mathsf{seed}$, which returns a handle $\tau_1$ 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-Gen.** On input $f$,

3. $\mathcal{P}$ and $\mathcal{V}$ sends ($\mathsf{extend}$) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns $u, w$ to $\mathcal{P}$ and $v$ to $\mathcal{V}$, such that $w = v + u \cdot \Delta$.
4. On input polynomial $f(\cdot) = \sum_{i=0}^k f_i \cdot X^i \in \mathbb{F}[X]$, $\mathcal{P}$ computes a ciphertext $\langle b \rangle$ with $u + b = f(\Lambda)$ via $\langle b \rangle = \sum_{i=1}^k f_i \cdot \langle \Lambda^i \rangle + f_0 - u$.
5. $\mathcal{P}$ and $\mathcal{V}$ call the ($\mathsf{Commit}$) command of $\mathcal{F}_{\mathsf{Com}}$ with inputs $\langle b \rangle$, which returns a handle $\tau_2$ to $\mathcal{V}$.

**Gen.** On input $(\tau_1, \tau_2)$,

6. $\mathcal{V}$ and $\mathcal{P}$ call the ($\mathsf{Open}$) command of $\mathcal{F}_{\mathsf{Com}}$ on input $\tau_1$, which returns $(\mathsf{seed}, \tau_1)$ to $\mathcal{P}$. In parallel, $\mathcal{V}$ sends $\Lambda$ 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. $\mathcal{P}$ sets $\mathsf{M} := w$.
7. $\mathcal{P}$ and $\mathcal{V}$ call the ($\mathsf{Open}$) command of $\mathcal{F}_{\mathsf{Com}}$ on input $\tau_2$, which returns $(\langle b_1 \rangle, \ldots, \langle b_\ell \rangle, \tau_2)$ to $\mathcal{V}$. Then, $\mathcal{V}$ runs $b \leftarrow \mathsf{Dec}(\mathsf{sk}, \langle b \rangle)$, and then computes $\mathsf{K} := v - b \cdot \Lambda \in \mathbb{F}$.
8. Two parties obtain an IT-PAC $[f(\cdot)]$, where $\mathcal{P}$ holds $(f(\cdot), \mathsf{M})$ and $\mathcal{V}$ holds $\mathsf{K}$.

Figure 7: Protocol for generating IT-PACs without ZK proofs in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.

In the same way, check that $\boldsymbol{r} = \mathbf{R}\boldsymbol{w}$, $\boldsymbol{o} = \mathbf{O}\boldsymbol{w}$, and that $\mathbf{0} = \mathbf{A}\boldsymbol{w}$. After wire consistency check, we check the correctness of multiplication gates. Define circuit $\mathcal{C}_{\mathsf{Mult}}(\{\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i\}_{i\in[n_2]}) := \bigvee_{i\in[n_2]}(\boldsymbol{x}_i\boldsymbol{y}_i - \boldsymbol{z}_i)$, and call $\mathsf{Prove}(\mathcal{C}_{\mathsf{Mult}}, \{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i\in[n_2]})$. Then, $\mathcal{V}$ calls $(\mathsf{Open}, [\![\boldsymbol{q}]\!])$ and checks $\sum_{i=1}^B \boldsymbol{q}[i] = 0$. $\mathcal{V}$ aborts if any check fails. The full description of SIMD AntMan is shown in Protocol 6 and 8.

**AntMan++.** By applying our SIMD compiler to the original SIMD AntMan, we propose AntMan++, which is a more efficient VOLE-based ZK proof for arbitrary circuits. Similar to the original AntMan, we first batch arithmetic gates and prove their correctness. The generation of IT-PACs of all the wire values incurs $\mathcal{O}(|\mathcal{C}|/B)$ communication communication complexity. Additionally, checking the correctness of multiplication gates requires an opening of size $B$.

The improvement of AntMan++ lies in the proof of wire consistency. As shown in $\Pi_{\mathsf{compiler}}$, this problem is transferred into proof of linear map. And we use a random vector to further transfer linear-mapping proof into inner-product proof. In AntMan, we observe that the proof of inner product between public and private vectors takes only $\mathcal{O}(B)$ communication overhead. Suppose the challenge vector $\boldsymbol{r}$ is public and witness $\boldsymbol{x}$ is private, and the IT-PACs of two vectors are known to both parties. After the secret evaluation point $\Lambda$ is revealed, both parties can locally calculate $f_{\boldsymbol{r}}(\Lambda)$ because $\boldsymbol{r}$ is known. Via the additively homomorphic property of IT-PACs, both parties

**Protocol $\Pi_{\mathsf{AntMan}}$ (Cont.)**

**Prove:** On input $(\mathcal{C}, [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$, $\mathcal{P}$ and $\mathcal{V}$ do:

1. For each gate $(\alpha, \beta, \gamma, T)$ in $\mathcal{C}$, two parties holds IT-PAC of input wire vectors $[\![f]\!]$ and $[\![g]\!]$:
   - If $T = ADD$, both parties locally compute output IT-PAC $[\![h]\!] = [\![f]\!] + [\![g]\!]$.
   - If $T = MULT$, $\mathcal{P}$ computes a degree-$(2B-2)$ polynomial $\widetilde{h}(\cdot) := f(\cdot) \cdot g(\cdot) \in \mathbb{F}[X]$ and a degree-$(B-1)$ polynomial $h(\cdot)$ such that $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [B]$. Then, $\mathcal{P}$ and $\mathcal{V}$ run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ to generate two IT-PACs $[\![h(\cdot)]\!]$ and $[\![\widetilde{h}(\cdot)]\!]$.

   As there are $n_2$ multiplication gates, the commitments of their outputs are denoted as $[\![h_1]\!], \ldots, [\![h_{n_2}]\!]$. Consequently, their degree-$(2B-2)$ polynomials are denoted as $[\![\widetilde{h}_1]\!], \ldots, [\![\widetilde{h}_{n_2}]\!]$.
2. $\mathcal{P}$ samples two random polynomials $r(\cdot)$ and $s(\cdot)$ of respective degrees $B-1$ and $2B-2$ in $\mathbb{F}[X]$ such that $r(\alpha_i) = s(\alpha_i)$ for $i \in [1, t]$. Then, $\mathcal{P}$ and $\mathcal{V}$ generate the corresponding IT-PACs $[\![r(\cdot)]\!]$ and $[\![s(\cdot)]\!]$.
3. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$ and sends it to $\mathcal{P}$. Then, two parties compute $(\chi_1, \ldots, \chi_{n_2}) := \mathsf{Hash}(\mathsf{seed}) \in \mathbb{F}^{n_2}$.
4. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[\![h(\cdot)]\!] := \sum_{j=1}^{n_2} \chi_j \cdot [\![h_j(\cdot)]\!] + [\![r(\cdot)]\!]$ and $[\![\widetilde{h}(\cdot)]\!] := \sum_{j=1}^{n_2} \chi_j \cdot [\![\widetilde{h}_j(\cdot)]\!] + [\![s(\cdot)]\!]$. Then, $\mathcal{P}$ sends the polynomial pair $(h(\cdot), \widetilde{h}(\cdot))$ to $\mathcal{V}$, who checks that $h(\cdot), \widetilde{h}(\cdot)$ have the degrees $B-1$ and $2B-2$ respectively and $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [1, t]$.
5. $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{Gen}(\tau_1, \tau_2)$ to open $\Lambda$ to $\mathcal{P}$, and then $\mathcal{V}$ can compute the local keys on all IT-PACs.
6. $\mathcal{P}$ and $\mathcal{V}$ run a VOLE-based zero-knowledge proof
$$\mathsf{DVZK}\left\{([\![f_j(\Lambda)]\!], [\![g_j(\Lambda)]\!], [\![\widetilde{h}_j(\Lambda)]\!])_{j\in[n_2]} \mid \forall j \in [n_2], \widetilde{h}_j(\Lambda) = f_j(\Lambda) \cdot g_j(\Lambda)\right\}.$$
7. $\mathcal{P}$ and $\mathcal{V}$ locally compute $[\mu] := [h(\Lambda)] - h(\Lambda)$ and $[\nu] := [\widetilde{h}(\Lambda)] - \widetilde{h}(\Lambda)$. Then, two parties run $\mathsf{Open}$ to check that $\mu = 0$ and $\nu = 0$.
8. Let $[\![v(\cdot)]\!]$ be the IT-PAC associated with the output values circuit $\mathcal{C}$. $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{Open}$ to check $v(\Lambda) = 0$.

If any check fails, $\mathcal{V}$ aborts.

Figure 8: The protocol of $\mathsf{SIMDZK}$ from AntMan (Cont.).

compute $f_{\boldsymbol{r}}(\Lambda) \cdot [\![\boldsymbol{x}]\!]$, which is also the IT-PAC of Hadamard product of $\boldsymbol{r}$ and $\boldsymbol{x}$. In this way, both parties compute $n + k$ IT-PACs and add them up to obtain $[\![\boldsymbol{q}]\!]$. In the end, according to the protocol in figure 3, both parties open the vector of size $B$ and check whether their sum equals to 0. As a result, the communication cost of AntMan++ is $\mathcal{O}(|\mathcal{C}|/B + B)$. When setting $B = |\mathcal{C}|^{1/2}$, it results in $\mathcal{O}(|\mathcal{C}|^{1/2})$.

**Performance evaluation.** We implement the AntMan++ protocol and benchmark its performance. Its homomorphic encryption (HE) is supported by the Microsoft SEAL [SEA22] and other cryptographic building blocks are from EMP-toolkits [WMK16]. Two Amazon EC2 `m5.8xlarge` instances located in the same region are running as $\mathcal{P}$ and $\mathcal{V}$. We manually throttle the network to simulate low-bandwidth settings. We use the same 59-bit FFT-friendly field as the AntMan [WYY+22]. The performance of AntMan++ is not affected by the circuit structure and we benchmark with layered circuits for convenience. In all experiments, we randomly sample a circuit with $2^{16}$ input wires, $2^{27}$ addition gates and $2^{27}$ multiplication gates distributed at $2^{12}$ layers. We compare AntMan++ with the prior general VOLE-ZK Quicksilver [YSWW21].

| $\log_2 B$ | Comm. (MB) | | Running time (s) | |
|---|---|---|---|---|
| | Setup | Online | Setup | Online |
| 9 | 4.6 | 60.13 | 6.84 | 377.3 |
| 10 | 4.6 | 30.54 | 14.7 | 380.7 |
| 11 | 4.6 | 15.78 | 38.72 | 407.83 |
| 12 | 6.7 | 8.82 | 144.75 | 438.19 |
| QS [YSWW21] | 1087.23 | | 185.43 | |

Table 1: **Performance of AntMan++ with variable batch size.** Benchmarked with 1 thread, 50 Mbps bandwidth and circuit size $C = 2^{27}$.

| Scheme-Threads | Bandwidth (Mbps) | | | |
|---|---|---|---|---|
| | 10 | 25 | 50 | 100 |
| AM-1 | 461.71 | 449.75 | 446.55 | 444.17 |
| AM-4 | 292.61 | 280.53 | 277.43 | 275.28 |
| AM-8 | 263.55 | 249.89 | 248.24 | 246.07 |
| QS-8 [YSWW21] | 900.47 | 361.29 | 181.63 | 91.9 |

Table 2: **Performance of AntMan++ with variable threads and bandwidth.** Benchmarked with circuit size $C = 2^{27}$ and batch size $B = 2^{11}$.

We first benchmark the running time and communication overhead with variable batch size $\log_2 B \in [9, 12]$. AntMan++ is split into the input-independent setup phase and online phase, and their performance are reported separately. As shown in Table 1, the increase of $B$ leads to the significant reducing of the online communication overhead. The setup communication is dominated by HE ciphertexts and rotation keys. For the security of HE, the ciphertext size is fixed for all $\log_2 B \leq 11$ and start to increase when $B \geq 12$. The running time for both setup and online phases increase with $B$. The overhead mainly comes from the ciphertext rotation during the setup phase as well as the HE evaluation and polynomial multiplication during the online phase. Although its running time is $2.1\times \sim 3.2\times$ longer than Quicksilver, the bandwidth usage is $17.5\times \sim 83.6\times$ smaller.

Then we show the running time with the variable network bandwidth and the number of threads (Table 2). The batch size is fixed to be $B = 2^{11}$. AntMan++ is highly efficient in terms of network communication with asymptotically $\mathcal{O}(C/B)$ overhead. Its running time does not significantly deteriorate with the decreasing of bandwidth. On the another hand, AntMan++ is computationally heavy but fully parallelable, thus multi-threading is effective on increasing its throughput. When the number of threading is increased from 1 to 4, the running time is decreased by $36\% \sim 38\%$. Compared to Quicksilver, it requires 70% less running time when bandwidth is 10Mbps and 30% less when bandwidth is 25Mbps.

### 4.2 Constant-round SIMD-ZK in the Discrete Logarithm setting

Turning the attention to the circuit ZK in the discrete logarithm setting which traditionally can be achieved with the communication cost linear to the circuit size, recently there are some efficient works [Gro09, Seo11, BCC+16, AC20] significantly reducing the communication complexity from

linear to sublinear in constant rounds or to logarithm with logarithm round complexity. While the discrete logarithm-based zero-knowledge arguments in [Gro09, Seo11] have constant moves and sublinear communication, [BCC+16] based on [Gro09] and Bulletproof proposes an ingenious recursive approach to achieve a communication and round complexity proportional to the logarithm of the circuit size. Currently, the work of Cramer *et al.* [AC20] obtains the same result as [BCC+16] by using a different approach i.e they firstly construct a compressed pivot for ZK proof of arbitrary linear form (inspired by Bulletproof) then use this pivot as a black box combining with pack secret sharing polynomial. When considering the circuit ZK of [AC20] in constant moves and without the Fiat-Shamir heuristic, it achieves a sublinear communication complexity with the size of CRS to be $\mathcal{O}(n)$ and the dominant computation cost to be equivalent to $\mathcal{O}(1)$-interpolation of polynomials of degree $\mathcal{O}(n)$ where $n$ is the size of the circuit.
We present an SHVZK argument of knowledge for $(B, C)$- SIMD circuits and then for a general circuit based on Pedersen's commitment. We firstly state the compression and the amortization techniques used in [AC20] by a formal construction. We show in detail that, a sublinear argument of knowledge for linear form evaluation is actually achieved in constant-move with specific explanations about communication and computation complexity while in [AC20] it only appears in the remark. We then use these techniques to build batch check multiplications (section 4.2.3). Note that, the batch check for multiplication gates is non-trivial, it is realized by carefully using the amortization for many checks of different committed vectors over the same linear form. Finally, the construction of SIMD ZK and the concrete efficiency of circuit ZK are described in section 4.2.4. Compared with circuit ZK of [AC20], we obtained a better CRS size and computation complexity by taking advantage of our compiler and the approach of dividing circuit into smaller ones, i.e our CRS size is $\mathcal{O}(B)$ instead of $\mathcal{O}(|\mathcal{C}|)$, our dominant computation cost is $\mathcal{O}(|\mathcal{C}|/B)$ interpolations of polynomial of degree $\mathcal{O}(B)$ while it is $\mathcal{O}(1)$ interpolations of polynomial of degree $\mathcal{O}(|\mathcal{C}|)$ in [AC20].

### 4.2.1 Sublinear ZK Argument for Linear Form Evaluation

Given a linear form $L : \mathbb{Z}_p^n \to \mathbb{Z}_p$, consider the relation

$$\mathcal{R} = \{(P \in \mathbb{G}, L, y \in \mathbb{Z}_p; \mathbf{x} \in \mathbb{Z}_p^n, r \in \mathbb{Z}_p) : P = \mathbf{g}^{\mathbf{x}} h^r, y = L(\mathbf{x})\}$$

We have commitment $P \in \mathbb{G}$ to $\mathbf{x} \in Z_p^n$ and the prover wants to convince the verifier that the correctness of evaluation of $L$ on the committed vector i.e $y = L(\mathbf{x})$, while the information of $\mathbf{x}$ is kept secret.
Without the loss of generality, for a dimension $m$ and a vector $\mathbf{g} \in G^m$, given $k|m$ where $k$ is the number of parts to divide $\mathbf{g}$ into, if this is not the case the vector $\mathbf{g}$ can be appended with zeros, then $\mathbf{g} := \mathbf{g}_1|\mathbf{g}_2|\ldots|\mathbf{g}_k \in \mathbb{Z}_p^m$ and $\mathbf{g}_i \in \mathbb{Z}_p^{m/k}$ for $i \in [1, k]$. Given a linear form $L : \mathbb{Z}_p^m \to \mathbb{Z}_p$, let us define $k$ sub-linear forms $L_i : \mathbb{Z}_p^{m/k} \to \mathbb{Z}_p$ of $L$ as $\mathbf{x} \to L(\mathbf{x}')$ where vector $\mathbf{x}' \in Z_p^m$ such that block $i$ of $\mathbf{x}'$ equals to $\mathbf{x}$ and other $k-1$ blocks equal to $0s$. As a result, $L(\mathbf{x}) = \sum_{i=1}^{k} L_i(\mathbf{x}_i)$. The notation $[\![\mathbf{x}]\!]$ is a Pedersen commitment of vector $\mathbf{x}$.

While in [AC20] divides the witness vector $\mathbf{x} \in Z_p^n$ into 2 parts then recursive run protocol $\log(n) - 1$ times and get an honest-verifier proof of knowledge for relation $\mathcal{R}$ with $\mathcal{O}(\log n)$ bits communication in $\mathcal{O}(\log n)$ moves. Observing from [Gro09], there is a trade-off between communication cost and the number of rounds, so we can generalize by dividing the witness into $k = \mathcal{O}(\sqrt{n})$ parts resulting in a protocol in constant-round with sublinear communication. This observation is indicated informally in [AC20], we now state it in a formal protocol as Figure 9 with the detail of complexity cost.

**Protocol $\Pi([\![\mathbf{x}]\!], L, y; \mathbf{x})$**

**Public parameters.** $(\mathbf{g}, h, k) \in \mathbb{G}^{n+2}, L : \mathbb{Z}_p^n \to \mathbb{Z}_p, P = \mathbf{g}^{\mathbf{x}} h^r, y = L(\mathbf{x})$.
**Protocol.**

1. $\mathbf{P} \to \mathbf{V}$. Prover picks randomly $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n, \beta \xleftarrow{\$} \mathbb{Z}_p$. Prover sends $t = L(\boldsymbol{\alpha}), A = \mathbf{g}^{\boldsymbol{\alpha}} h^{\beta}$ to verifier.

2. $\mathbf{V} \to \mathbf{P}$. Verifier chosen randomly $c_0, c_1 \xrightarrow{\$} \mathbb{Z}_p$ and sends them to prover.
   Prover defines $\mathbf{z} = c_0 \mathbf{x} + \boldsymbol{\alpha}$, $\gamma = c_0 r + \beta$ and $\hat{\mathbf{z}} := (\mathbf{z}, \gamma)$.
   Define $\hat{\mathbf{g}} := (\mathbf{g}, h) \in \mathbb{G}^{n+1}, Q := AP^{c_0} k^{c_1(c_0 y + t)}$ and $\tilde{L}(\mathbf{z}, \gamma) := c_1 L(\mathbf{z})$. Note that $Q := \hat{\mathbf{g}}^{\hat{\mathbf{z}}} k^{\tilde{L}(\hat{\mathbf{z}})}$.

3. $\mathbf{P} \to \mathbf{V}$. For $l \in [0, 2k-2]$, prover computes:

$$m_l := \prod_{i,j,l=k+i-j-1} \hat{\mathbf{g}}_i^{\hat{\mathbf{z}}_j} k^{\tilde{L}_i(\hat{\mathbf{z}}_j)}$$

   Observe by construction $m_{k-1} = Q$.
   Prover sends $m_i$ to verifier for $i \in [0, 2k-2]$.

4. $\mathbf{V} \to \mathbf{P}$. Verifier picks randomly $c \xrightarrow{\$} \mathbb{Z}_p$ and sends $c$ to prover.
   Define $\mathbf{g}' := \hat{\mathbf{g}}_1 * \hat{\mathbf{g}}_2^{c} * \cdots * \hat{\mathbf{g}}_{k-1}^{c^{k-2}} * \hat{\mathbf{g}}_k^{c^{k-1}} \in \mathbb{G}^{n+1/k}$ ($*$ is denoted as component-wise product), $Q' := \prod_{l=0}^{2k-2} m_l^{c^l}$, $L' := \sum_{i=1}^{k} c^{i-1} \tilde{L}_i$.

5. $\mathbf{P} \to \mathbf{V}$. Prover defines and sends $\mathbf{z}' := \sum_{i=1}^{k} c^{k-i} \hat{\mathbf{z}}_i \in \mathbb{Z}_p^{n+1/k}$ to verifier.
   Verifier checks $\mathbf{g}'^{\mathbf{z}'} k^{L'(\mathbf{z}')} \stackrel{?}{=} Q'$.

Figure 9: HVZK argument $\Pi([\![\mathbf{x}]\!], L, y; \mathbf{x})$ for relation $\mathcal{R}$

**Theorem 1.** *$\Pi$ (Figure 9) is a 5-move protocol for relation $\mathcal{R}$. This argument is perfect completeness, special HVZK, and computationally special soundness. The total communication costs include:*

- ***P*** $\to$ ***V:*** *$2k-1$ elements of $\mathbb{G}$ and $(n+1)/k+1$ element of $\mathbb{Z}_p$.*
- ***V*** $\to$ ***P:*** *3 elements of $Z_p$.*

The completeness comes from:

$$\begin{aligned} Q' := \prod_{l=0}^{2k-2} m_l^{c^l} &= \prod_{l=0}^{2k-2} \left( \prod_{i,j,l=k+i-j-1} \hat{\mathbf{g}}_i^{\hat{\mathbf{z}}_j} k^{\tilde{L}_i(\hat{\mathbf{z}}_j)} \right)^{c^l} = \prod_{l=0}^{2k-2} \left( \prod_{i,j,l=k+i-j-1} \hat{\mathbf{g}}_i^{\hat{\mathbf{z}}_j} k^{\tilde{L}_i(\hat{\mathbf{z}}_j)} \right)^{c^{k+i-j-1}} \\ &= \prod_{i=1}^{k} \left( \hat{\mathbf{g}}_i^{c^{i-1}} \right)^{\sum_{j=1}^{k} c^{k-j} \hat{\mathbf{z}}_j} \prod_{i=1}^{k} \left( k^{c^{i-1}} \right)^{\tilde{L}_i(\sum_{j=1}^{k} c^{k-j} \hat{\mathbf{z}}_j)} \\ &= \left( \hat{\mathbf{g}}_1 * \hat{\mathbf{g}}_2^{c} * \cdots * \hat{\mathbf{g}}_{k-1}^{c^{k-2}} * \hat{\mathbf{g}}_k^{c^{k-1}} \right)^{\mathbf{z}'} k^{\sum_{i=1}^{k} c^{i-1} \tilde{L}_i(\mathbf{z}')} = \mathbf{g}'^{\mathbf{z}'} k^{L'(\mathbf{z}')} \end{aligned}$$

The remaining proof of this theorem is directly obtained from [AC20], [Gro09].

### 4.2.2 Amortization

In this section, we present some amortizations which allow the prover to prove that 1) many nullity checks of *different* linear forms over the same committed vector $\mathbf{x}$ and 2) many evaluations of the

same linear form over many *different* committed vectors in the 5-move protocols by the trade-off of an insignificant communication cost compared to single check $\Pi$.

**Compressed many nullity checks as one** Given $P = \mathbf{g}^{\mathbf{x}}h^r$, $s$ linear functions $L_i : \mathbb{Z}_p^n \to Z_p$, the prover want to prove that $L_i(\mathbf{x}) = 0$ for all $i \in [1, s]$. By the well-known Schwartz-Zippel lemma, the prover can do many nullity checks at the cost of one single check plus one more element of $Z_p$ (challenge) from the verifier to prover. Especially, the verifier sends a challenge $\varphi \xleftarrow{\$} \mathbb{Z}_p$ to prover and both of them then define the new linear form $L(\mathbf{x}) := \sum_{i=1}^{s} L_i(\mathbf{x})\varphi^{i-1}$. Observe that $L(\mathbf{x}) = 0$ implies $L_i(\mathbf{x}) = 0$ for all $i \in [1, s]$ with the probability at least $1 - (s-1)/p \approx 1$ when $s$ is polynomial of $\lambda$ and $p$ is exponential. So, it requires only one single nullity check plus sending one more challenge $\varphi$ to do $s$ nullity checks. instead of running sequentially $s$- $\Pi([\![\mathbf{x}]\!], L_i, 0; \mathbf{x})$ for $i \in [1, s]$.

**Protocol** $\Pi_{\mathsf{zeros}}([\![\mathbf{x}]\!], L_1, L_2, \ldots, L_s, 0; \mathbf{x})$

**Public parameters.** $(\mathbf{g}, h) \in \mathbb{G}^{n+1}, L_i : \mathbb{Z}_p^n \to \mathbb{Z}_p$ for $i \in [1, s]$, $P = \mathsf{com}_{\mathsf{ck}}(\mathbf{x}, r) = \mathbf{g}^{\mathbf{x}}h^r$.
**Protocol.**

1. $\mathbf{V} \to \mathbf{P}$. Verifier chosen randomly $\varphi \xrightarrow{\$} \mathbb{Z}_p$ and sends to prover.
2. Define $L := \sum_{i=1}^{s} L_i\varphi^{i-1}$.
3. Prove and verifier runs protocol $\Pi(P, L, 0; \mathbf{x})$ for checking the evaluation of $L(\mathbf{x}) \stackrel{?}{=} 0$.

Figure 10: Many nullity checks $\Pi_{\mathsf{zeros}}([\![\mathbf{x}]\!], L_1, L_2, \ldots, L_s, 0; \mathbf{x})$

**Amortized over many commitments** Given $P_i = \mathbf{g}^{\mathbf{x}_i}h^{r_i}$ for $i \in [1, s]$, the prover wants to convince that the evaluation of the same linear form $L$ on many committed vectors is correct i.e $y_i = L(\mathbf{x}_i)$. Intuitively, a prover can do this batch evaluation checks of $L$ over many committed vectors $\mathbf{x}_i$ at the same cost of evaluation checks of $L$ over only one committed vector. We can see that $\tilde{P} := A\prod_{i=1}^{s} P_i^{c^i}$ is the Pedersen commitment of vector $\mathbf{z} = \sum_{i=1}^{i} \mathbf{x}_i c_0^i + \boldsymbol{\alpha}$ where $A$ is commitment of $\boldsymbol{\alpha}$ then $L(\mathbf{z}) := L(\boldsymbol{\alpha}) + \sum_{i=1}^{s} c_0^i y_i$. Note that, assume $c_0 \xleftarrow{\$} \mathbb{Z}_p$, if the prover knows the open of $\tilde{P}$, it means the prover has to know the open of each $P_j$ with a probability of almost 1. Indeed, if there exists an $P_j$ such that prover does not know the opening $\mathbf{x}_j$ then prover can cheat when having a correct opening of $\tilde{P}$ with at most probability of $s/p$ (a cheating prover succeeds when $c_0$ is the zero of some polynomial of degree at most $s$).
Following the protocol $\Pi$ (Figure 9), we modify the definition as $\mathbf{z} = \sum_{i=1}^{i} \mathbf{x}_i c_0^i + \boldsymbol{\alpha}$, $\gamma = \sum_{i=1}^{s} r_i c_0^i + \beta$ and $Q := A\prod_{i=1}^{s} P_i^{c_0^i} k^{c_1(\sum_{i=1}^{s} c_0^i y_i + t)}$. So, prover and verifier now can interact following the same last 3-move in $\Pi$.

**Protocol $\Pi_{\mathsf{Am}}([\![\mathbf{x}_i]\!], L, y_i; \mathbf{x}_i)_{i\in[1,s]}$**

**Public parameters.** $(\mathbf{g}, h, k) \in \mathbb{G}^{n+2}, L : \mathbb{Z}_p^n \to \mathbb{Z}_p, P_i = \mathbf{g}^{\mathbf{x}_i} h^{r_i}, y_i = L(\mathbf{x}_i)$ for $i \in [1, s]$.
**Protocol.**

1. $\mathbf{P} \to \mathbf{V}$. Prover picks randomly $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n, \beta \xleftarrow{\$} \mathbb{Z}_p$. Prover sends $t = L(\boldsymbol{\alpha}), A = \mathbf{g}^{\boldsymbol{\alpha}} h^{\beta}$ to verifier.
2. $\mathbf{V} \to \mathbf{P}$. Verifier chosen randomly $c_0, c_1 \xrightarrow{\$} \mathbb{Z}_p$ and sends them to prover.
   Prover defines $\mathbf{z} = \sum_{i=1}^{i} \mathbf{x}_i c_0^i + \boldsymbol{\alpha}$, $\gamma = \sum_{i=1}^{s} r_i c_0^i + \beta$ and $\hat{\mathbf{z}} := (\mathbf{z}, \gamma)$.
   Define $\hat{\mathbf{g}} := (\mathbf{g}, h) \in \mathbb{G}^{n+1}, Q := A \prod_{i=1}^{s} P_i^{c_0^i} k^{c_1(\sum_{i=1}^{s} c_0^i y_i + t)}$ and $\tilde{L}(\mathbf{z}, \gamma) := c_1 L(\mathbf{z})$. Note that $Q := \hat{\mathbf{g}}^{\hat{\mathbf{z}}} k^{\tilde{L}(\hat{\mathbf{z}})}$.
3. The last 3-move is the same as the protocol in the Figure 9.

Figure 11: Amortized over many commitments $\Pi_{\mathsf{Am}}([\![\mathbf{x}_i]\!], L, y_i; \mathbf{x}_i)_{i\in[1,s]}$

### 4.2.3 Batch argument for multiplication gates

Let's consider $m$ tuples of $B$ multiplications $(x_{j,i}, y_{j,i}, z_{j,i} = x_{j,i} y_{j,i})_{i\in[1,B]}$ for each $j \in [1, m]$. This approach follows the commit and prove paradigm, i.e., the prover commits to the witness and subsequently proves that it satisfies the required relation. The batch argument is based on algebraic interpolation polynomial and the ZK proof $\Pi$ which proves the correct evaluation of linear form (consider $\Pi$ as a black box).

- For $j \in [1, m]$, $\mathcal{P}$ defines 2 random polynomials $f_j, g_j$ of degree at most $B$ such that $f_j(i) = x_{j,i}$ and $g_j(i) = y_{j,i}$ for all $i \in [1, B]$. By Lagrange-interpolation $f_j, g_j$ are well-defined from $\mathbf{x}_j := (f_j(0), x_{j,1}, x_{j,2}, \ldots, x_{j,B}) \in \mathbb{Z}_p^{B+1}$ and $\mathbf{y}_j := (g_j(0), y_{j,1}, y_{j,2}, \ldots, y_{j,B}) \in \mathbb{Z}_p^{B+1}$. Define $h_j := f_j g_j$, observe that degree of $h_j$ is at most $2B$, $h_j(i) = z_{j,i}$ for $i \in [1, B]$ and $h_j$ is well-defined from $\mathbf{z}_j := (h(0), z_{j,1}, z_{j,2}, \ldots, z_{j,B}, h(B+1), \ldots, h(2B)) \in \mathbb{Z}_p^{2B+1}$. $\mathcal{P}$ commits $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)_{j\in[1,m]}$.
- $\mathcal{V}$ pick randomly $c \xrightarrow{\$} \mathbb{Z}_p \setminus [1, B]$ and sends it to $\mathcal{P}$.
- $\mathcal{P}$ reveals $(f_j(c), g_j(c), h_j(c))_{j\in[1,m]}$. Verifier now then checks $h_j(c) \stackrel{?}{=} f_j(c) g_j(c)$. $\mathcal{P}$ can cheat with negligible probability at most $(2B)/(p - B)$.
  Denote $L_c : \mathbb{Z}_p^{B+1} \to \mathbb{Z}_p, L'_c : \mathbb{Z}_p^{2B+1} \to \mathbb{Z}_p$ are public linear forms by Lagrange formula such that $L_c(\mathbf{x}_j) = f_j(c), L_c(\mathbf{y}_j) = g_j(c)$ and $L'_c(\mathbf{z}_j) = h_j(c)$ ($f_j, g_j$ are corresponding to the same linear form).
- $\mathcal{P}$ runs in parallel $\Pi_{\mathsf{Am}}([\![\mathbf{x}_j]\!], [\![\mathbf{y}_j]\!], L_c, f_j(c), g_j(c); \mathbf{x}_j, \mathbf{y}_j)_{j\in[1,m]}$ and $\Pi_{\mathsf{Am}}([\![\mathbf{z}_j]\!], L'_c, h_j(c); \mathbf{z}_j)_{j\in[1,m]}$.

We obtain an argument for $n$ multiplications with *sublinear* communication cost when choosing $B = \mathcal{O}(\sqrt{n})$.

**Theorem 2.** *There is an argument for showing the corrections of n multiplications with sublinear communication cost in constant-round. This argument has the below properties:*

- *Perfect completeness, computational special soundness, and special HVZK.*
- *The size of CRS is $2B + 3$ random elements of $\mathbb{G}$.*
- *The total communication cost includes*

- $P \to V$ : $3n/B + \mathcal{O}(\sqrt{B})$ *elements of* $\mathbb{G}$ *and* $(\mathcal{O}(\sqrt{B}) + 3n/B)$*elements of* $\mathbb{Z}_p$.
- $V \to P$ : 4 *elements of* $\mathbb{Z}_p$.

- *The computation complexity is* $\mathcal{O}(n/B)$ *interpolations of polynomial of degree* $\mathcal{O}(B)$ *over* $\mathbb{Z}_p$.

#### 4.2.4 Instantiation of SIMD ZK

The prover P and verifier V hold a general circuit $\mathcal{C}$ over $\mathbb{Z}_p$, where $\mathcal{C}$ contains $k$ addition gates, $n$ multiplication gates and $m$ input gates. P holds $B$ witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_B \in \mathbb{Z}_p^m$ such that $\mathcal{C}(\mathbf{w}_i) = 0$ for all $i \in [1, B]$. For $B$ executions of circuit $\mathcal{C}$, P and 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 $C$ into a group.

1. **Commitment of gates**
   For each group $j \in [1, |\mathcal{C}|]$, the prover defines commitments for tuple of left wire values $(l_{1,j}, l_{2,j}, \ldots, l_{B,j})$, right wire values $(r_{1,j}, r_{2,j}, \ldots, r_{B,j})$ and output wire values $(o_{1,j}, o_{2,j}, \ldots, o_{B,j})$ in the following way

   - For left wire values, the prover selects random polynomial $f_j(X)$ that defines a packed secret sharing of the vector $(l_{1,j}, l_{2,j}, \ldots, l_{B,j})$ i.e $f_j(i) = l_{i,j}$ for $i \in [1, B]$, prover then commit vector $\mathbf{l}_j := (f_j(0), l_{1,j}, l_{2,j}, \ldots, l_{B,j})$ as $[\![l_j]\!]$. Similarly for the right wire values with random polynomial $g_j(X)$.
   - If the group contains addition gates then prover sends $[\![l_j]\!], [\![r_j]\!]$ to verifier and the commitment of output wire values $[\![o_j]\!] := [\![l_j]\!][\![r_j]\!]$ is non-interactive computed by both prover and verifier.
   - Otherwise, if the group contains multiplication gates $[\![o_j]\!]$ is defined by the commitment of vector $\mathbf{o}_j := (h(0), o_{1,j}, o_{2,j}, \ldots, o_{B,j}, h(B+1), \ldots, h(2B))$ where $h(X) = f(X)g(X)$. Prover then sends $[\![l_j]\!], [\![r_j]\!]$ and $[\![o_j]\!]$ to verifier.

2. **Correctness check of multiplication gates**
   Prover convinces verifier that the evaluation of multiplication gates is correct using the batch arguments in section 4.2.3.

3. **Consistency check of output gates**
   For output gates, prover wants to show that the value of all output gates is 0s. It means that when $j = |\mathcal{C}|$, $[\![o_j]\!]$ is the commitment of vector $\mathbf{o}_j = (r, 0, 0, \ldots, 0) \in \mathbb{Z}_p^{B+1}$ (w.r.t addition output gates) or $\mathbf{o}_j = (h_j(0), 0, 0, \ldots, 0, h_j(B+1), \ldots, h_j(2B)) \in \mathbb{Z}_p^{2B+1}$ (multiplication output gates). Consider $\mathsf{crs} = (g_1, g_2, \ldots, g_{2B+3})$ then prover has to convince that in $[\![o_j]\!]$ the powers of set of generator $\{g_2, g_3, \ldots, g_{B+1}\}$ are zeros. This constraint actually is a batch of $B$-nullity checks for different linear forms over one committed vector. Prover and verifier then run $\Pi_{\mathsf{zeros}}$ for many nullity checks.

Informally, we achieve a ZK proof for $(B,\mathcal{C})$-SIMD circuits with communication $\mathcal{O}(|\mathcal{C}| + \sqrt{B})$. Especially, sending commitments of gates costs at most $3|\mathcal{C}|$ elements of $\mathbb{G}$. For checking the correction of multiplication gates and consistency of output gates, the communication cost is less than 3 times the instantiation of $\Pi$ over the committed vector of length $2B$ which has $\mathcal{O}(\sqrt{B})$ communication cost.

**Sublinear circuit satisfiability** Given an arbitrary circuit $\mathcal{C}$, we present a ZK proof for circuit satisfiability with *sublinear* communication cost based on generalized Pedersen commitments. Intuitively, the circuit $\mathcal{C}$ is divided into $B$ smaller sub-circuits having the same number of input gates,

addition gates, and multiplication gates then we do ZK proof in the batch of tuple $B$ elements corresponding with the same type of gate in $B$ sub-circuits. Without loss of generality, assume that the number of input gates, addition gates, and multiplication gates of circuit $\mathcal{C}$ are multiple of $B$. If not the compiler to transfer general circuits in detail is explained in [WYY+22].

Since Pedersen's commitment is homomorphic, additions are free in our system, there are two constraints needed to prove 1) that multiplication gates are correctly computed and 2) the consistency of wires between layers. The former constraints is handled by the batch multiplication argument and the latter is proven using our compiler (section 3). Note here, we apply our compiler to prove in ZK the consistency of wires between layers (this is essentially a proof of a linear map) and combine the high-level intuition underlying the analysis of our compiler with the analysis of Attema et al. to obtain a direct security proof. Concretely, we carefully combine two works ([3] and [28]) to obtain an SHVZK for SIMD circuit.

These two proofs use different commitments for two vectors which present for the same tuple of output wire values of multiplication gates so then it requires to prove the consistency. Specifically, for $j$ group of multiplication gates, we have two commitments $[\![o'_j]\!]$, $[\![o_j]\!]$ which committed of two vectors $\mathbf{o}'_j$ and $\mathbf{o}_j$. While $\mathbf{o}'_j := (r, o_{1,j}, o_{2,j}, \ldots, o_{B,j}) \in \mathbb{Z}_p^{B+1}$ and $\mathbf{o}_j := (h_j(0), o_{1,j}, o_{2,j}, \ldots, o_{B,j}, h_j(B+1), \ldots, h_j(2B)) \in \mathbb{Z}_p^{2B+1}$ where $r \in \mathbb{Z}_p$. Prover therefore needs to prove that $[\![o'_j]\!]/[\![o_j]\!]$ is the commitment of vector which having the power 0s of the set of generators $\{g_2, g_3, \ldots, g_{B+1}\}$. By the method described earlier, this is handled by a $\Pi_{\mathsf{Am}}$ for checking many nullities. The protocol $\Pi_{\mathsf{ZKPed}}$ of our sublinear ZK is shown in the Figure 12.

**Theorem 3.** *There is a sublinear argument of knowledge for circuit satisfiability in constant-round with the following properties:*

- *Perfect completeness, computational special soundness, and special HVZK.*
- *The number of rounds is 7.*
- *The size of CRS is $\mathcal{O}(\sqrt{|\mathcal{C}|})$ random elements of $\mathbb{G}$.*
- *The dominant computation cost is equivalent to $\mathcal{O}(\sqrt{|\mathcal{C}|})$-interpolations of polynomial of degree $\mathcal{O}(\sqrt{|\mathcal{C}|})$.*

Note that in our sublinear ZK based on DLOG setting, we do not directly derive the result from the generic UC proof of security of the abstract compiler, and in particular, do not achieve UC security. This would require the commitments to be extractable. The proof of the consistency of wires follows our compiler, but there is some extra work needed to prove the consistency of commitments (described above). As for the security analysis, the analysis of our ZK based on DLOG is not directly inherited from the real-ideal security proof of instantiation of eSIMDZK, but rather follows directly from the security analysis of the two works [AC20] and [Gro09]. Note that we can define $[\![x]\!]$ in functionality 2 being Perdersen commitment of $x$, but in DLOG-setting, we never need to extract the commitments, and the proofs of soundness and ZK are not in the UC model.

**Comparison with the Work of Attema–Cramer.** The work of [AC20] described a logarithmic-round and logarithmic communication protocol from the discrete logarithm assumption, and mentioned in a remark that their protocol can be made constant-round, at the cost of increasing the communication to $\mathcal{O}(\sqrt{C})$. This builds upon a "direct" reduction from proving satisfiability of an arithmetic circuit to batch Hadamard arguments and proofs for linear relations. Working out the details, the protocol of Attema–Cramer enjoys

**Protocol $\Pi_{\mathsf{ZKPer}}$**

- **Preprocessing circuit** Every $B$ same-type gate is divided into a group, the prover commits all left wire and right wire values of $B$ gates as in the SIDM-ZK framework. Then for every $j$-th group, we have 2 commitments $[\![l_j]\!], [\![r_j]\!]$.

  - If the $j$-th group includes addition gates, prover commits output wire values as in SIDM, i.e $[\![o_j]\!] := [\![l_j]\!][\![r_j]\!]$.
  - Otherwise, if this group contains multiplication gates, prover computes two commitments $[\![o_j]\!]$ and $[\![o'_j]\!]$ while $[\![o_j]\!]$ is defined in SIMDZK for output wire values of multiplication gates and $[\![o'_j]\!]$ is defined as the same of $[\![l_j]\!], [\![r_j]\!]$. Note that $[\![o'_j]\!]$, $[\![o_j]\!]$ are used for showing the correction of routing and multiplications respectively.

  All the wires in circuit $\mathcal{C}$ are also divided into groups of $B$ wires and each $j$-th group is committed to getting the commitments $[\![w_j]\!]$ as $[\![l_j]\!], [\![r_j]\!]$.

- **Correctness check of multiplication gates** is presented in section 4.2.

- **Consistency check of wire routing** We use high-level intuition of the compiler described in the section 3 to prove the corresponding left wire, right wire, and output wire values of all gates to the wire values of circuit. As described, the core idea of our compiler is that we transfer the proof of consistency into a SIMD ZK which can be achieved with sublinear communication.

- **Consistency of output wire values of multiplication gates** Observe that for each group of multiplication gates, we have two commitments $[\![o'_j]\!]$, $[\![o_j]\!]$ which committed of two vectors $\mathbf{o}'_j$ and $\mathbf{o}_j$. While $\mathbf{o}'_j := (r, o_{1,j}, o_{2,j}, \ldots, o_{B,j}) \in \mathbb{Z}_p^{B+1}$ and $\mathbf{o}_j := (h_j(0), o_{1,j}, o_{2,j}, \ldots, o_{B,j}, h_j(B+1), \ldots, h_j(2B)) \in \mathbb{Z}_p^{2B+1}$ where $r \in \mathbb{Z}_p$. Prover therefore need to prove that $[\![o'_j]\!]/[\![o_j]\!]$ is the commitment of vector which having the power 0s of the set of generators $\{g_2, g_3, \ldots, g_{B+1}\}$. By the method described earlier, this is handled by a $\Pi_{\mathsf{Am}}$ for checking many nullities.

If any check fails, V aborts.

Figure 12: The protocol of SHVZK for circuit satisfiability from Pedersen commitment.

- A constant number of rounds,
- $\mathcal{O}(\sqrt{C})$ communication,
- $\mathcal{O}(C)$ CRS size,
- A computation dominated by $\mathcal{O}(1)$ executions of an FFT on degree-$C$ polynomials (as well as $\mathcal{O}(C)$ exponentiations).

In contrast, when using our approach (which proceeds by first building a ZK proof for SIMD circuits via the techniques of Attema–Cramer combined with a careful batch checking argument, then applying our general compile) also has constant round complexity and $\mathcal{O}(\sqrt{C})$ communication, but additionally achieves a sublinear CRS size $\mathcal{O}(\sqrt{C})$, and the computation is dominated by $\mathcal{O}(\sqrt{C})$ executions of an FFT on degree-$\sqrt{C}$ polynomials (as well as $\mathcal{O}(C)$ exponentiations). For

large circuits, due to the polylogarithmic overhead of FFTs, this translates to a lower computational overhead.

### 4.3 Multi-Verifier ZK

Yang et al. propose a non-interactive designated multi-verifier ZK (MVZK) proof [YW22]. In this protocol, a prover $\mathcal{P}$ validates a statement to $n$ verifiers $(\mathcal{V}_1, \ldots, \mathcal{V}_n)$ with its private input. Verifiers are assumed honest-majority with adversary threshold $t < n(1/2 - \varepsilon)$ for $0 < \varepsilon < 1/2$. $\mathcal{P}$ distributes packed Shamir secret shares (PSS) of all batched circuit wire values to $(\mathcal{V}_1, \ldots, \mathcal{V}_n)$, who jointly execute a distributed ZK to verify the correctness of the circuit evaluation [GSZ20, BGIN21, BBC+19], with the assistance of $\mathcal{P}$. A special coin tossing protocol is designed to maintain non-interactiveness. This MVZK protocol can be viewed as a commit-and-prove ZK, in which the circuit wire values are committed by the PSS. The *hiding* property is ensured by the privacy property of PSS, for which a collusion of $\leq t$ parties can not reconstruct the secret values. The *binding* property holds by the fact that any $(n - t) > t + 1$ honest parties' shares define the secret values.

In addition to proving the satisfiability of SIMD circuits, [YW22] also proposes a protocol for the check of wiring consistency, which enables the PSS-based MVZK to work for general circuits. Define a packing parameter $B$, for any indices $i, j \in [B]$ and PSS $[\boldsymbol{w}_1], [\boldsymbol{w}_2]$, the protocol proves that $\boldsymbol{w}_1[i] = \boldsymbol{w}_2[j]$. Overall, the checking procedure incurs communication complexity $\mathcal{O}(n^2B^2)$. Our compiler reduces it to $\mathcal{O}(n^2)$. In the following, we first introduce a protocol that verifies the multiplications, before delving into the multi-verifier zero-knowledge proof protocol for SIMD circuits. Then we describe the realization of the functionality $\mathcal{F}_{\mathsf{verifyprod}}$, which recursively compress the check of an inner product to a single multiplication triple.

**SIMD-ZK from [YW22].** The protocol is shown in Figure 13. A commitment in MVZK is a PSS for a vector of $B$ values. The opening of a commitment is done by each verifier sending its PSS share to all other verifiers, followed by all of them validating the shares and decoding the committed values. The proving procedure takes the input of commitments to batch circuit input wires and output wires of all multiplication gates. They are precomputed by $\mathcal{P}$ and distributed to verifiers via PSS. At Step 1, verifiers locally arrange $([\![\boldsymbol{w}_\alpha]\!], [\![\boldsymbol{w}_\beta]\!], [\![\boldsymbol{w}_\gamma]\!])$ for the indices of all batch multiplication triples $(\alpha, \beta, \gamma)$. The PSS for the input wires of batch multiplication gates are obtained locally by linearly combining the PSS for previous batches. Next, parties invoke a Fiat-Shamir procedure at Step 2 to sample a random coin $\chi \in \mathbb{K}$. The property of non-interactiveness forbids the verifiers from sending messages to $\mathcal{P}$. In this protocol, $\mathcal{P}$ computes the input to the Fiat-Shamir transformation from the shares of parties and let verifiers verify their correctness, namely, $(com_1, \ldots, com_n)$. In this way, verifiers are able to compute $\chi$ by hashing these commitments. In the end, parties convert the multiplication triples to an inner product triple, which is verified by $\mathcal{F}_{\mathsf{verifyprod}}$.

**Inner product verification $\mathcal{F}_{\mathsf{verifyprod}}$.** The functionality $\mathcal{F}_{\mathsf{verifyprod}}$ takes input $m\ell$ multiplication triples and first convert it into a dimension-$m\ell$ inner product triple $\{([\![\boldsymbol{x}_i]\!], [\![\boldsymbol{y}_i]\!]\}_{i=1}^{m\ell}$ and $[\![\boldsymbol{z}]\!]$. Then it verifies whether $\boldsymbol{z} = \sum_{i=1}^{m\ell} \boldsymbol{x}_i * \boldsymbol{y}_i$. It incurs only logarithmic communication and round-trip complexity via a compression procedure. Define $\mathbb{K}$ to be a field extension of $\mathbb{F}$ and is exponentially large to achieve negligible soundness error. The compression is done in the following steps.

1. Divide the triple into $m$ inner product triples of dimension-$\ell$

$$\{([\![\boldsymbol{x}_{i,1}]\!], \ldots, [\![\boldsymbol{x}_{i,\ell}]\!]), ([\![\boldsymbol{y}_{i,1}]\!], \ldots, [\![\boldsymbol{y}_{i,\ell}]\!]), [\![\boldsymbol{z}_i]\!]\}_{i=1}^{m}$$

The shares $\{[\![\boldsymbol{z}_i]\!]\}_{i=1}^{m-1}$ are directly distributed by $\mathcal{P}$, and $[\![\boldsymbol{z}_m]\!] = [\![\boldsymbol{z}]\!] - \sum_{i=1}^{m-1} [\![\boldsymbol{z}_i]\!]$

2. interpolate $2\ell$ degree-$(m-1)$ polynomials

$$(\llbracket \boldsymbol{f}_1(\cdot) \rrbracket, \ldots, \llbracket \boldsymbol{f}_\ell(\cdot) \rrbracket), (\llbracket \boldsymbol{g}_1(\cdot) \rrbracket, \ldots, \llbracket \boldsymbol{g}_\ell(\cdot) \rrbracket)$$

such that $\llbracket \boldsymbol{f}_j(i) \rrbracket = \llbracket \boldsymbol{x}_{i,j} \rrbracket$ and $\llbracket \boldsymbol{g}_j(i) \rrbracket = \llbracket \boldsymbol{y}_{i,j} \rrbracket$, for $i \in [m], j \in [\ell]$.

3. $\mathcal{P}$ computes $\boldsymbol{z}_i = \sum_{j=1}^{\ell} \boldsymbol{f}_j(i) * \boldsymbol{g}_j(i)$ for $i \in [m+1, 2m-1]$. It distributes $\{\llbracket \boldsymbol{z}_i \rrbracket\}_{i=m+1}^{2m-1}$ to parties.

4. Interpolate a degree-$2(m-1)$ polynomial $\llbracket \boldsymbol{h}(\cdot) \rrbracket$ such that $\llbracket \boldsymbol{h}(i) \rrbracket = \llbracket \boldsymbol{z}_i \rrbracket$ for $i \in [2m-1]$.

5. Jointly sample a random element $\beta \in \mathbb{K}$, and evaluate these $2\ell+1$ polynomials to $\beta$. Output a dimension-$\ell$ inner product triple

$$(\llbracket \boldsymbol{f}_1(\beta) \rrbracket, \ldots, \llbracket \boldsymbol{f}_\ell(\beta) \rrbracket), (\llbracket \boldsymbol{g}_1(\beta) \rrbracket, \ldots, \llbracket \boldsymbol{g}_\ell(\beta) \rrbracket), \llbracket \boldsymbol{h}(\beta) \rrbracket.$$

The above compression is executed recursively for logarithmic rounds to reduce the dimension of inner product triple to constant. The last round of compression outputs a single multiplication triple $(\tilde{\boldsymbol{f}}(\tilde{\beta}), \tilde{\boldsymbol{g}}(\tilde{\beta}), \tilde{\boldsymbol{h}}(\tilde{\beta}))$, whose correctness will be checked by all parties. Two facts are omitted in the above description, and we refer the readers to [YW22].

- The sampling of random point $\beta \in \mathbb{K}$ is done by Fiat-Shamir heuristic. It requires a public message known by all parties. At step 3, instead of $\mathcal{P}$ distributing a PSS $\llbracket \boldsymbol{z} \rrbracket$, parties first hold a random PSS $\llbracket \boldsymbol{r} \rrbracket$, then $\mathcal{P}$ broadcasts $\boldsymbol{u} = \boldsymbol{z} + \boldsymbol{r}$. Then parties not only derive $\llbracket \boldsymbol{z} \rrbracket = \llbracket \boldsymbol{r} \rrbracket - \boldsymbol{u}$, but also attain the public message $\boldsymbol{u}$.
- To achieve zero-knowledge, the multiplication triple output at the final round should not reveal any private information. Some independent randomness will be added into the final round of compression to mask the information computed from the witness.

## 5 Acknowledgement

Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award #2016240, #2318974, #2310927 and research awards from Meta and Google. Work of Geoffroy Couteau is supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE) and the France 2030 ANR Project ANR-22-PECY-003 SecureCompute. Work of Dung Bui is supported by Dim Math Innov funding from the Paris Mathematical Sciences Foundation (FSMP) funded by the Paris Ile-de-France Region. 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

[AC20] Thomas Attema and Ronald Cramer. Compressed $\Sigma$-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

**Protocol $\Pi_{\mathsf{MVZK}}$**

**Setup:** Define the number of verifiers $n$, packing parameter $B$ and adversary threshold $t < n(1/2 - \epsilon)$ for $0 < \epsilon < 1/2$, which satisfy $t + 2B - 1 \leq n$. $\alpha_1, \ldots, \alpha_n, \gamma_1, \ldots, \gamma_B \in \mathbb{F}$ are $n + B$ evaluation points. Let $H_1 : \{0,1\}^* \to \{0,1\}^\kappa$ and $H_2 : \{0,1\}^* \to \mathbb{K}$ be two random oracles.

**Commit:** On input degree $d$ and a vector $\boldsymbol{w} \in \mathbb{F}^B$, $\mathcal{P}$ uniformly samples $\boldsymbol{r} \in \mathbb{F}^{d-B+1}$ and interpolates a degree-$d$ polynomial $f(\cdot)$ such that $f(\gamma_i) = w_i$ for $i \in [B]$ and $f(\alpha_i) = r_i$ for $i \in [d - B + 1]$. The commitment is defined as $[\![\boldsymbol{w}]\!] = \{f(\alpha_i)\}_{i=1}^n$, in which $f(i)$ is held by $\mathcal{V}_i$.
The communication cost can be reduced from $n|\mathbb{F}|$ to $(n - d + B - 1)|\mathbb{F}|$ if for $i \in [d - B + 1]$, $\mathcal{P}$ and $\mathcal{V}_i$ hold a shared PRG $\mathsf{PRG}_i$. The share can be computed locally as $f(\alpha_i) \leftarrow \mathsf{PRG}_i.\mathsf{Next}()$.

**Open:** On input degree $d$, and commitment $[\![\boldsymbol{w}]\!]$, each verifier in $(\mathcal{V}_1, \ldots, \mathcal{V}_n)$ sends its share to all other verifiers. They reconstruct the degree-$d$ polynomial $f(\cdot)$ and derive $\boldsymbol{w} = (f(\gamma_1), \ldots, f(\gamma_B))$. If the reconstruction fails, the verifier aborts.

**Prove:** On input the commitment to circuit input wires and output wires of multiplications $([\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$,

1. $\mathcal{P}$ and $(\mathcal{V}_1, \ldots, \mathcal{V}_n)$ scan the circuit in topological order. For each batch of multiplication gates $(\alpha, \beta, \gamma)$ for which they already have $[\![\boldsymbol{w}_\gamma]\!]$, compute $([\![\boldsymbol{w}_\alpha]\!], [\![\boldsymbol{w}_\beta]\!])$ which are linear combinations of commitments for previous wires.
2. Denote $\hat{w}^i$ as the share that $\mathcal{V}_i$ holds for a PSS $[\![\boldsymbol{w}]\!]$. For $i \in [n]$, $\mathcal{P}$ samples $r_i \leftarrow \{0,1\}^\kappa$, sends $r_i$ to $\mathcal{V}_i$ and computes
$$com_i = H_1(\hat{w}_1^i, \ldots, \hat{w}_m^i, r_i).$$
$\mathcal{P}$ broadcasts $(com_1, \ldots, com_n)$ to all verifiers. Each verifier $\mathcal{V}_i$ checks whether $com_i$ is consistent with its local shares and $r_i$. It aborts if the check fails. Otherwise, $\mathcal{P}$ and all verifiers compute $\chi := H_2(com_1, \ldots, com_i)$.
3. Denote $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i \in [n]}$ to be commitments of all left, right and output wires of batched multiplication gates. Construct $[\tilde{\boldsymbol{x}}_i] := \chi^{i-1} \cdot [\boldsymbol{l}_i]$, $[\tilde{\boldsymbol{y}}_i] := [\boldsymbol{r}_i]$ for $i \in [n]$ and $[\tilde{\boldsymbol{z}}] := \sum_{i \in [n]} \chi^{i-1} \cdot [\boldsymbol{o}_i]$. Invoke $\mathcal{F}_{\mathsf{verifyprod}}$ with input $(\{[\tilde{\boldsymbol{x}}_i]\}_{i \in [n]}, \{[\tilde{\boldsymbol{y}}_i]\}_{i \in [n]}, [\tilde{\boldsymbol{z}}])$ and parties output $\mathsf{reject}$ if $\mathcal{F}_{\mathsf{verifyprod}}$ outputs $\mathsf{reject}$.

Figure 13: The protocol of $\mathsf{SIMDZK}$ from designated multi-verifier ZK.


[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. ACM Press, October / November 2017.

[AHIV22] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. Cryptology ePrint Archive, Report 2022/1608, 2022. https://eprint.iacr.org/2022/1608.

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

[BBHV22] Laasya Bangalore, Rishabh Bhadauria, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. On black-box constructions of time and space efficient sublinear arguments from symmetric-key primitives. In Eike Kiltz and Vinod Vaikuntanathan,

editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 417–446. Springer, Heidelberg, November 2022.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 255–272. Springer, Heidelberg, August 2012.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 427–457. Springer, Heidelberg, May / June 2022.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 681–710, Virtual Event, August 2021. Springer, Heidelberg.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020.

[BG22] Alexander R. Block and Christina Garman. Honest majority multi-prover interactive arguments. Cryptology ePrint Archive, Report 2022/557, 2022. https://eprint.iacr.org/2022/557.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 457–485, Virtual Event, August 2021. Springer, Heidelberg.

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

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

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

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Report 2022/1355, 2022. https://eprint.iacr.org/2022/1355.

[CFF+21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[DIO20] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446, 2020. https://eprint.iacr.org/2020/1446.

[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. Springer, Heidelberg, August 2012.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. SPARKs: Succinct parallelizable arguments of knowledge. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 707–737. Springer, Heidelberg, May 2020.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

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

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 192–208. Springer, Heidelberg, August 2009.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Heidelberg, August 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In Mikkel Thorup, editor, *59th FOCS*, pages 124–135. IEEE Computer Society Press, October 2018.

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

[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. Springer, Heidelberg, April / May 2018.

[KS22] Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. https://eprint.iacr.org/2022/1758.

[KS23] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023. https://eprint.iacr.org/2023/573.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Heidelberg, August 2022.

[KZGM21] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly auditable mpc-as-a-service with succinct verification and universal setup. *CoRR*, abs/2107.04248, 2021.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990.

[Lip16] Helger Lipmaa. Prover-efficient commit-and-prove zero-knowledge SNARKs. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 185–206. Springer, Heidelberg, April 2016.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 4291–4308. USENIX Association, August 2022.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PS04] Boaz Patt-Shamir. A note on efficient aggregate queries in sensor networks. In Soma Chaudhuri and Shay Kutten, editors, *23rd ACM PODC*, pages 283–289. ACM, July 2004.

[SEA22] Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL, March 2022. Microsoft Research, Redmond, WA.

[Seo11] Jae Hong Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 387–402. Springer, Heidelberg, March 2011.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552, 2023. https://eprint.iacr.org/2023/552.

[SVdV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 346–366. Springer, Heidelberg, June 2016.

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

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

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

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018.

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

[YW22] Kang Yang and Xiao Wang. Non-interactive zero-knowledge proofs to multiple verifiers. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 517–546. Springer, Heidelberg, December 2022.

# A Supplementary Material

## A.1 Additional Preliminaries

The following definitions are provided for the instantiation of SHVZK argument of knowledge.

**Definition 1(Generalized Pedersen Commitment [Ped92]).** Given a Abelian group $\mathbb{G}$ of prime order $p$. The Pedersen commitment works as follow:

- Set up a commitment key $\mathsf{ck} = (\mathbf{g}, h) = \{g_1, g_2, \ldots, g_n, h\} \xleftarrow{\$} \mathbb{G}^{n+1}$.
- Commitment of a vector $\mathbf{x} \in \mathbb{Z}_p^n$ is defined by $\mathsf{com}_{\mathsf{ck}}(\mathbf{x}, r) = h^r \mathbf{g}^{\mathbf{x}} = h^r \prod_{i=1}^{n} g_i^{x_i}$ where $r \xleftarrow{\$} \mathbb{Z}_p$.

Pedersen commitment is perfectly hiding, computationally binding under the DDH assumption. It satisfies homomorphic property, for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}_p^n$, $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$:

$$\mathsf{com}_{\mathsf{ck}}(\mathbf{x}_1, r_1).\mathsf{com}_{\mathsf{ck}}(\mathbf{x}_2, r_2) = \mathsf{com}_{\mathsf{ck}}(\mathbf{x}_1 + \mathbf{x}_2, r_1 + r_2)$$

Let $\mathcal{R}$ be an efficiently decidable binary relation for an **NP** language $\mathscr{L}$. If $x \in \mathscr{L}$ and $(x, w) \in \mathcal{R}$ then $x$ is a statement and $w$ is a witness. An interactive argument for $\mathcal{R}$ is a tuple of three probabilistic polynomial time interactive algorithms $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ called the common reference string generator, the prover and the verifier. with the following properties:

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$. On input $1^\lambda$ generates public parameters *par* (such as group parameters), a $\mathsf{crs}$. For simplicity of notation, we assume that any group parameters are implicitly included in the $\mathsf{crs}$.
- We write $tr \leftarrow \langle \mathsf{P}(x), \mathsf{V}(y) \rangle$ for the public transcript produced by $\mathsf{P}$ and $\mathsf{V}$ when interacting on inputs $x$ and $y$. This transcript ends with $\mathsf{V}$ either accepting or rejecting. We sometimes shorten the notation by saying $\langle \mathsf{P}(x), \mathsf{V}(y) \rangle = b$, where $b = 0$ corresponds to $\mathsf{V}$ rejecting and $b = 1$ corresponds to $\mathsf{V}$ accepting.

**Definition 2(Perfect completeness).** A proof system $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ for $\mathcal{R}$ is perfectly complete, if

$$\Pr\left[\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, x) \rangle = 1 \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ (x, w) \in \mathcal{R} \end{array}\right] = 1$$

**Definition 3(Computationally soundness).** A proof system $\Pi$ is computational sound if for every efficient adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}, \mathsf{V}(\mathsf{crs}, x) \rangle = 1 \\ x \notin \mathscr{L} \end{array} \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ x \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

An argument $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is public coin if the verifier's messages are chosen uniformly at random independently of the messages sent by the prover, i.e the challenges correspond to the verifier's randomness $\rho$. If there is a polynomial time algorithm that, given a statement $x$ and a $(k_1, \ldots, k_\mu)$-tree of accepted transcripts, produces a witness $w$ for $x$, then the public coin protocol is said to be (unconditionally) $(k_1 \ldots, k_\mu)$-special sound. Under the DL assumption, we state that the protocol is *computationally* special sound if there exists an efficient algorithm that either extracts a witness or finds a non-trivial DL relation between $g_1, \ldots, g_n, h$.

**Protocol $\Pi_{\mathsf{RSEncode}}$**

**Parameters.** Define parameters $n, B$ such that $B < n$. $L^d_{\mathsf{RS}}$ is the set of all valid Reed-Solomon codeword. $\alpha_1, \ldots, \alpha_n, \gamma_1, \ldots, \gamma_B \in \mathbb{F}$ are $n + B$ evaluation points. A pseudorandom generator $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^*$.

**Input.** Parameter $d$ satisfying $B \leq d + 1 < n$, vector $\boldsymbol{x} \in \mathbb{F}^B$ and a randomness $r \in \{0,1\}^\kappa$.

**Encode.** If $B < d + 1$, compute $(\bar{x}_1, \ldots, \bar{x}_{d-B+1}) \leftarrow \mathsf{PRG}(r)$. Interpolate a degree-$d$ polynomial $f_{\boldsymbol{x}}(\cdot)$ such that $f_{\boldsymbol{x}}(\gamma_i) = x_i$ for $i \in [B]$ and and set $f_{\boldsymbol{x}}(\alpha_i) = \bar{x}_i$ for $i \in [d-B+1]$. Output $(f_{\boldsymbol{x}}(\alpha_1), \ldots, f_{\boldsymbol{x}}(\alpha_n))$.

Figure 14: Reed-Solomon Encoding.

**Definition 4(Special honest-verifier zero-knowledge (SHVZK)).** A public coin argument $\Pi$ is a SHVZK if if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\left[\begin{array}{l|c} \mathcal{A}(tr) = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ (x,w) \in \mathcal{R} & (x,w,\rho) \leftarrow \mathcal{A}(\mathsf{crs}); tr \leftarrow \langle \mathsf{P}(\mathsf{crs},x,w), \mathsf{V}(\mathsf{crs},x)\rangle \end{array}\right]$$
$$\approx \Pr\left[\begin{array}{l|c} \mathcal{A}(tr) = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ (x,w) \in \mathcal{R} & (x,w,\rho) \leftarrow \mathcal{A}(\mathsf{crs}); tr \leftarrow \mathcal{S}(\mathsf{crs},x,\rho) \end{array}\right]$$

where $\rho$ is the public coin randomness used by the verifier.

## A.2 SIMD ZK from Ligero

We present the SIMD ZK protocol modified from Ligero [AHIV17] with its security analysis. As shown in Figure 16, the Protocol $\Pi_{\mathsf{LigeroSIMD}}$ is an instantiation of $\mathcal{F}_{\mathsf{SIMDZK}}$. Its two building blocks are the Reed-Solomon (RS) encoding described in Figure 14 and the Merkle commitment scheme described in Figure 15.

**Protocol $\Pi_{\mathsf{MerkleCnP}}$**

Define $\tau_{\mathsf{max}}$ to be the maximum time a commitment is used in the MerkleProve procedure.

**MerkleCommit.** On input a vector $\boldsymbol{x}$ of size $|\boldsymbol{x}| = n$, Build a Merkle tree with $2^{\lceil \log_2 n \rceil}$ leaf nodes, where the first $n$ leaf nodes are elements in $\boldsymbol{x}$ and the rest are dummy. Output the root node $\tau \in \{0,1\}^{2\kappa}$. Initiate a counter $\mathsf{ctr} = 0$ and associate it with $\tau$.

**MerkleProve.** On input $(\boldsymbol{x}, Q) \in \mathbb{F}^n \times \mathbb{N}^t$, first check the counter ctr that associates with the commitment to $\boldsymbol{x}$. If $\mathsf{ctr} \geq \tau_{\mathsf{max}}$, abort. Recompute the Merkle tree as described in MerkleCommit. Construct $\boldsymbol{\sigma}$ which contains all sibling nodes that are on the path to the leaves in set $\{\boldsymbol{x}[i]\}_{i \in Q}$. Output $\boldsymbol{\sigma}$. Set $\mathsf{ctx} = \mathsf{ctr} + 1$.

**MerkleVerify.** On input $(\tau, \{\boldsymbol{x}[i]\}_{i \in Q}, \boldsymbol{\sigma})$, reconstruct the Merkle tree path from $\{\boldsymbol{x}[i]\}_{i \in Q}, \boldsymbol{\sigma}$. Define the Merkle tree root $\tau'$. If $\tau \neq \tau'$, output fail.

Figure 15: Merkle tree vector commitment.

*Commit and open procedures.* As described in Figure 14, to commit to a vector of field elements $\boldsymbol{w} \in \mathbb{F}^B$, the prover first pad it into a length-$(d+1)$ vector with randomly sampled $d - B + 1$

elements, then encode them into a Reed-Solomon codeword $\boldsymbol{u} \in \mathbb{F}^n$. Eventually, the codeword is committed by the Merkle commitment scheme specified in Figure 15. Assume a set of $n + B$ distinct evaluation points $\alpha_1, \ldots, \alpha_n, \gamma_1, \ldots, \gamma_B \in \mathbb{F}$. The $(n, d+1)$-RS codes naturally determines a degree-$d$ polynomial $f(\cdot)$ where $f(\alpha_i) = u_i$ for $i \in [n]$ and $f_j(\gamma_i) = w_i$ for $i \in [B]$. To open a commitment, the prover simply reveal the vector $\boldsymbol{w}$ and the randomness used for padding.

Assume that during the commitment phase, $\mathcal{P}$ commits to $m$ batches of wire values $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$ to $\mathcal{V}$, which are taken as inputs during the proving phase to convince $\mathcal{V}$ of the relation $\mathcal{C}(\llbracket \boldsymbol{w}_1 \rrbracket, \ldots, \llbracket \boldsymbol{w}_m \rrbracket) = 0$. The proving phase (in Figure 16) for SIMD circuits inherits the tests of interleaved linear codes and quadratic constraints over interleaved linear codes from the Ligero [AHIV17] protocol. Throughout these steps, the challenges sampled and sent by $\mathcal{V}$ include the coefficients $(\boldsymbol{r}, \bar{\boldsymbol{r}})$ for random linear combination, and the set $Q$ indicating the subset of $t$ elements to be opened among a length-$n$ codeword.

*Testing interleaved Reed-Solomon Codes.* $\mathcal{P}$ additionally samples and commits to a random vector $\boldsymbol{w}_{m+1}$ as a mask. Upon receiving the coefficients $\bar{\boldsymbol{r}} \in \mathbb{F}^{m+1}$, $\mathcal{P}$ responses with the polynomial $h(\cdot) := \sum_{j=1}^{m+1} \bar{r}_j f_j(\cdot)$, in which $\{f_j(\cdot)\}_{j\in[m]}$ are RS polynomials for committed wire values $\{\boldsymbol{w}_j\}_{j\in[m]}$ and $f_{m+1}(\cdot)$ is for $\boldsymbol{w}_{m+1}$. $\mathcal{V}$ first checks whether $h(\cdot)$ is a degree-$(k-1)$ polynomial. This is equivalent to check whether $(h(\alpha_1), \ldots, h(\alpha_n))$ is a valid Reed-Solomon codeword. After $\mathcal{P}$ reveals $\{\boldsymbol{u}_j[i]\}_{i\in Q}$ and proves the correctness of Merkle opening, $\mathcal{V}$ checks whether $h(\alpha_i) := \sum_{j=1}^{m+1} \bar{r}_j \boldsymbol{u}_j[i]$ for $i \in Q$. This step validates whether the openings of $t$ views are consistent with the previously claimed $h(\cdot)$. Since $h(\cdot)$ is the linear combination of committed codewords, the passing of above steps validates the correctness of RS encoding for all committed wire values. Intuitively, the zero-knowledge property is preserved by the mask $f_{\ell+1}(\cdot)$ corresponding to the randomly sampled $\boldsymbol{w}_{m+1}$. The soundness requires that $\bar{\boldsymbol{r}}$ is uniformly sampled after $\mathcal{P}$ committing to $\boldsymbol{w}_{m+1}$ and it is hard for $\mathcal{P}$ to guess correctly the subset $Q$ of size $t$.

*Testing quadratic constraints over interleaved Reed-Solomon Codes.* Assume that there are $\ell$ batches of multiplication gates in the SIMD circuit. $\mathcal{P}$ constructs and commits to $\boldsymbol{w}_{m+2} := \boldsymbol{0}^B$. Unlike the $(n, k)$-RS encoding used during previous the commitment phase, $\mathcal{P}$ needs to perform a $(n, 2k-1)$-RS encoding for $\boldsymbol{w}_{m+2}$, which results in a degree-$(2k-2)$ polynomial $f_{m+2}$. Upon receiving the challenge $\boldsymbol{r} \in \mathbb{F}^m$, $\mathcal{P}$ responses with the degree-$(2k-2)$ polynomial $g(\cdot) := f_{m+2}(\cdot) + \sum_{j=1}^{\ell} r_j \left(f_{\alpha_j}(\cdot) f_{\beta_j}(\cdot) - f_{\gamma_j}(\cdot)\right)$ where $(\alpha_j, \beta_j, \gamma_j)$ are the indices of input and output wires of a batch of multiplication gates. It becomes clear at this point that the reason why $f_{m+2}$ has degree $2(k-1)$ instead of $k-1$ is to disguise the combination of circuit wire encodings, which is a degree-$(2k-2)$ polynomial. If all multiplication gates are computed correctly, it should satisfy that $g(\gamma_i) = 0$ for $i \in [B]$. After $\mathcal{P}$ reveals $\{\boldsymbol{u}_j[i]\}_{i\in Q}$ and proves the correctness of Merkle opening, $\mathcal{V}$ checks their consistency with $g(\cdot)$. In terms of the security, the zero-knowledge is guaranteed by the mask $f_{m+2}$. Similar to the previous test, the soundness requires that $\boldsymbol{r}$ is uniformly sampled after $\mathcal{P}$ committing to $\boldsymbol{w}_{m+2}$ and it is hard for $\mathcal{P}$ to guess correctly the subset $Q$ of size $t$.

**Security Analysis.** We provide formal security analysis of the SIMD Ligero and pay attention to the case then compiling SIMD Ligero to general ZK and scalable ZK. We claim that the protocol $\Pi_{\mathsf{LigeroSIMD}}$ shown in Figure 16 securely realizes the functionality $\mathcal{F}_{\mathsf{SIMDZK}}$. The proof separately considers the case of corrupted verifier and prover. In each case, a PPT simulator $\mathcal{S}$ is constructed to simulate the view of adversaries.

**Malicious Verifier $\mathcal{V}^*$.** We construct a PPT simulator $\mathcal{S}$ to simulate the view of $\mathcal{V}^*$ executing the Protocol 16. $\mathcal{S}$ interacts with $\mathcal{A}$ in the following way.

1. $\mathcal{S}$ emulates a random oracle $\mathcal{O}_H$.

**Protocol $\Pi_{\mathsf{LigeroSIMD}}$**

**Public inputs.** Define Batch size $B$ and Reed-Solomon parameters $n, k, t$ such that $k < n$ and $B+t-1 \leq k$. $\alpha_1, \ldots, \alpha_n, \gamma_1, \ldots, \gamma_B \in \mathbb{F}$ are $n + B$ evaluation points.

**Private inputs.** $\mathcal{P}$ holds witness $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$ such that $\mathcal{C}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m) = \mathbf{0}^B$.

**Commit.** On input $\boldsymbol{w} \in \mathbb{F}^B$ and degree $d$ from $\mathcal{P}$, parties proceed as follows: (By default, $d = k - 1$, unless otherwise mentioned.)

1. $\mathcal{P}$ samples a uniform $r \in \{0,1\}^\kappa$ and invokes $(\boldsymbol{u}, f(\cdot)) \leftarrow \mathsf{RSEncode}(\boldsymbol{w}, r, d+1)$. The codeword $\boldsymbol{u}$ and degree-$d$ polynomial $f(\cdot)$ satisfy that $\boldsymbol{u} \in L^d_{\mathsf{RS}}$ and $f(\gamma_i) = \boldsymbol{w}[i]$ for $i \in [B]$.
2. $\mathcal{P}$ invokes $\tau \leftarrow \mathsf{MerkleCommit}(\boldsymbol{u})$ and sends $\tau$ to $\mathcal{V}$.
3. Output $[\![\boldsymbol{w}]\!] := ((\boldsymbol{w}, r, \boldsymbol{u}, f(\cdot)), \tau)$ such that $\mathcal{P}$ holds $(\boldsymbol{w}, r, \boldsymbol{u}, f(\cdot))$ and $\mathcal{V}$ holds $\tau$.

**Open.** On input $[\![\boldsymbol{w}]\!] := ((\boldsymbol{w}, r, \boldsymbol{u}, f(\cdot)), \tau)$ from $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{P}$ sends $(\boldsymbol{w}, r)$ to $\mathcal{V}$. $\mathcal{V}$ recomputes $(\boldsymbol{u}', f'(\cdot)) \leftarrow \mathsf{RSEncode}(\boldsymbol{w}, r)$ then $\tau' \leftarrow \mathsf{MerkleCommit}(\boldsymbol{u}')$. If $\tau \neq \tau'$, $\mathcal{V}$ aborts.

**Prove.** On input $([\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$, parties proceed as follows:

1. $\mathcal{P}$ uniformly samples $\boldsymbol{w}_{m+1} \in \mathbb{F}^B$ and constructs $\boldsymbol{w}_{m+2} := \mathbf{0}^B$. $\mathcal{P}$ and $\mathcal{V}$ invoke the above Commit procedure to obtain $[\![\boldsymbol{w}_{m+1}]\!]$. They also invoke Commit with input $d = 2(k-1)$ to obtain $[\![\boldsymbol{w}_{m+2}]\!]$.
2. $\mathcal{V}$ uniformly samples $\boldsymbol{r} \in \mathbb{F}^m, \bar{\boldsymbol{r}} \in \mathbb{F}^{m+1}$ and sends them to $\mathcal{P}$.
3. $\mathcal{P}$ holds $\{[\![\boldsymbol{w}_j]\!]\}_{j\in[m+2]} := \{(\boldsymbol{w}_j, r_j, \boldsymbol{u}_j, f_j(\cdot))\}_{j\in[m+2]}$. $\mathcal{P}$ defines:
   (a) $g(\cdot) := f_{m+2}(\cdot) + \sum_{j=1}^{\ell} r_j \left(f_{\alpha_j}(\cdot) f_{\beta_j}(\cdot) - f_{\gamma_j}(\cdot)\right)$ where $(\alpha_j, \beta_j, \gamma_j)$ are the indices of input and output wires of $j$-th batch of multiplication gates.
   (b) $h(\cdot) := \sum_{j=1}^{m+1} \bar{r}_j f_j(\cdot)$.

   $\mathcal{P}$ sends the degree-$(2k-2)$ polynomial $g(\cdot)$ and degree-$(k-1)$ polynomial $h(\cdot)$ to $\mathcal{V}$. $\mathcal{V}$ checks whether $g(\gamma_i) = 0$ for $i \in [B]$. It also checks whether the codeword $(h(\alpha_1), \ldots, h(\alpha_n)) \in L_{RS}$. If not, it aborts.
4. $\mathcal{V}$ uniformly samples a set $Q \subset [n]$ of size $|Q| = t$, and sends it to $\mathcal{P}$. For $j \in [m+2]$, $\mathcal{P}$ invokes $\boldsymbol{\sigma}_j \leftarrow \mathsf{MerkleProve}([\![\boldsymbol{w}_j]\!].\boldsymbol{u}, Q)$. It sends $\{\boldsymbol{u}_j[i]\}_{i\in Q}$ and the proof of opening $\boldsymbol{\sigma}_j$ to $\mathcal{V}$. $\mathcal{V}$ invokes $\mathsf{MerkleVerify}([\![\boldsymbol{w}_j]\!].\tau, \{\boldsymbol{u}_j[i]\}_{i\in Q}, \boldsymbol{\sigma}_j)$. If the verification fails, $\mathcal{V}$ aborts.
5. $\mathcal{V}$ checks the followings:
   (a) For any batches of addition gates indexed by $(\alpha, \beta, \gamma)$, it satisfies that $\boldsymbol{u}_\alpha[i] + \boldsymbol{u}_\beta[i] - \boldsymbol{u}_\gamma[i] = 0$ for $i \in Q$.
   (b) $g(\alpha_i) := \boldsymbol{u}_{m+2}[i] + \sum_{j=1}^{\ell} r_j \left(\boldsymbol{u}_{\alpha_j}[i] \cdot \boldsymbol{u}_{\beta_j}[i] - \boldsymbol{u}_{\gamma_j}[i]\right)$ for $i \in Q$.
   (c) $h(\alpha_i) := \sum_{j=1}^{m+1} \bar{r}_j \boldsymbol{u}_j[i]$ for $i \in Q$.

   If any check fails, $\mathcal{V}$ aborts.

Figure 16: The protocol of SIMD ZK from Ligero.

2. $\mathcal{S}$ simulates the procedure $\mathsf{MerkleCommit}$ as an honest prover do. It stores the commitment $\tau_j$ for each committed vector $\boldsymbol{w}_j$.
3. On simulating the $\mathsf{Prove}$, $\mathcal{S}$ first commits to random $(\boldsymbol{w}_{m+1}, \boldsymbol{w}_{m+2})$. Upon receiving $\boldsymbol{r}, \bar{\boldsymbol{r}}$ from $\mathcal{A}$, $\mathcal{S}$ samples a random degree-$(2k-2)$ polynomial $\tilde{g}(\cdot)$ such that $\tilde{g}(\gamma_i) = 0$ for $i \in [B]$. It also

samples a random degree-$(k-1)$ polynomial $\tilde{h}(\cdot)$. It sends $(\tilde{g}(\cdot), \tilde{h}(\cdot))$ to $\mathcal{A}$.

4. On receiving $Q$ from $\mathcal{A}$, $\mathcal{S}$ constructs $\{\{\boldsymbol{u}_j[i]\}_{i\in Q}\}_{j=1}^{m+2}$ in the following ways:
   (a) Construct random $\{\{\boldsymbol{u}_j[i]\}_{i\in Q}\}_{j=1}^{m}$ that can pass the check at step 5a.
   (b) Randomly sample the rest of values except for $\boldsymbol{u}_{\ell+1}$ and $\boldsymbol{u}_{\ell+2}$.
   (c) For $i \in Q$, set
   $$\boldsymbol{u}_{m+2}[i] := \tilde{g}(\alpha_i) - \sum_{j=1}^{\ell} r_j \left(\boldsymbol{u}_{\alpha_j}[i] \cdot \boldsymbol{u}_{\beta_j}[i] - \boldsymbol{u}_{\gamma_j}[i]\right)$$
   .
   (d) For $i \in Q$, set $\boldsymbol{u}_{m+1}[i] := \tilde{h}(\alpha_i) - \sum_{j=1}^{m} \boldsymbol{u}_j[i]$.

5. $\mathcal{S}$ simulates the procedure $\mathsf{MerkleProve}$. It aborts if the counter of a commitment reaches the maximum. Otherwise it samples random sibling nodes $\{\boldsymbol{\sigma}_j\}_{j\in[m+2]}$ and sends $\{\boldsymbol{\sigma}_j\}_{j\in[m+2]}$ to $\mathcal{A}$. $\mathcal{S}$ acts as an honest prover to compute Merkle trees from the above constructed codewords. It constructs a list $\mathcal{L}$ which records all inputs to the random oracle $\mathcal{O}_H$ when computing the root node.

6. $\mathcal{S}$ simulates the procedure $\mathsf{MerkleVerify}$ by monitoring the random oracle query to $\mathcal{O}_H$. For any query $q = \mathcal{L}[j]$ where $j \in [m]$, answer the query with $\tau_j$ (stored when simulating $\mathsf{MerkleCommit}$). Otherwise sample a uniform answer $\bar{\tau} \in \{0,1\}^{2\kappa}$.

We argue the indistinguishability between the real and ideal world from the view of $\mathcal{A}$. $\mathcal{S}$ emulates a programmable random oracle $\mathcal{O}_H$ to handle oracle queries and program the oracle output for certain inputs. At step 2, $\mathcal{S}$ simulates the Merkle tree commitment by a random value $\tau_j$ of which the length is the same as a hash output. The polynomials $g(\cdot)$ and $h(\cdot)$ sent by $\mathcal{S}$ is indistinguishable from their distribution in real-world because: $(i)$ In the real world, degree-$(2k-2)$ polynomial satisfies $g(\gamma_i) = 0$ for $i \in [B]$ and $g(\cdot)$ is masked by a random polynomial $f_{m+2}(\cdot)$ such that $f_{m+2}(\gamma_i) = 0$ for $i \in [B]$. It is perfectly simulated by $\tilde{g}(\cdot)$ in the ideal world. $(ii)$ In the real world, $h(\cdot)$ is masked by a random polynomial $f_{m+1}(\cdot)$. It is simulated by the random $\tilde{h}(\cdot)$ of the same degree. At step 4, $\{\{\boldsymbol{u}_j[i]\}_{i\in Q}\}_{j\in[\ell+2]}$ constructed by $\mathcal{S}$ follows the real-world distribution as long as the Reed-Solomon encoding has degree $k > B + t - 1$. In this case any $t$-out-of-$n$ views are random and independent of the encoded vector, thus indistinguishable from what are sampled by $\mathcal{S}$. At last, $\mathcal{S}$ monitors the random oracle and programs any oracle query in $\mathcal{L}$ for the computing of the Merkle tree root. $\Pi_{\mathsf{LigeroSIMD}}$ requires that the randomness of the commitment contains enough entropy so that there is negligible probability for $\mathcal{A}$ to make queries whose results match elements in $\mathcal{L}$.

**Notes on achieving zero-knowledge.** The protocol $\Pi_{\mathsf{LigeroSIMD}}$ shown in Figure 16 turns Ligero into a commit-and-prove SIMD ZK. It comes with restriction such that a commitment can only be used in a limited number of proofs. Also, this maximum number of usage $\tau_{\mathsf{max}}$ is determined upon the committing phase. This has been reflected in the functionality $\mathcal{F}_{\mathsf{SIMDZK}}$: a counter $\mathsf{ctr}_{\boldsymbol{w}}$ is attached to the commitment to vector $\boldsymbol{w}$. Each time a $[\![\boldsymbol{w}]\!]$ contributes to a proof, $\mathsf{ctr}_{\boldsymbol{w}}$ increments. A commitment should never be used again unless in the opening phase, if its counter reaches $\tau_{\mathsf{max}}$. In Ligero, it is related to the fact that each proving phase exposes $t$-out-of-$n$ committed views, which are $t$ elements in each of size-$n$ codeword. Once $(k+1)$-out-of-$n$ elements in the codeword is disclosed, the commitment will be fully opened. Hence, the parameters chosen during the initial committing phase fully determine $\tau_{\mathsf{max}}$ and it should guarantee $t \cdot \tau_{\mathsf{max}} \leq k$.

**Malicious Prover $\mathcal{P}^*$.** We analyze the soundness error when $\mathcal{A}$ (as a cheating prover) causes $\mathcal{S}$ to abort in the ideal world, but successfully cheats in the real world. Ligero follows the framework of MPC-in-the-Head in a way that $\mathcal{P}$ first commits to $n$ views to $\mathcal{V}$ by the Reed-Solomon code with length-$n$ codeword. Then $\mathcal{V}$ randomly chooses $t$ of them to open and check. A cheating prover $\mathcal{P}^*$ is not caught if the opened $t$ parties are among the honest parties that $\mathcal{P}^*$ emulates. The probability of such event is required to be negligible. Define $\Pr[\mathsf{succ}]$ to be the probability that the cheating prover $\mathcal{A}$ succeeds. We adapt the optimized soundness analysis in the full version of Ligero [AHIV22]. We have

$$\Pr[\mathsf{succ}] \leq \frac{\binom{k+e}{t} + \binom{2k-2}{t}}{\binom{n}{t}} + \frac{n+2}{|\mathbb{F}|}$$

in which the error $e < (n-k+1)/2$.

### A.3 Succinct Non-Interactive Arguments from Pairing

Campanelli et al. propose LegoSNARK [CFQ19], a framework for commit-and-prove zk-SNARKs (CP-SNARKs). In Figure 17, we present one of the detailed construction mentioned in the paper, which naturally fits the intuition of $\mathcal{F}_{\mathsf{eSIMDZK}}$. It has to be mentioned that in SNARK paradigm, to achieve succinctness, batch size $B$ is fixed as circuit size $\mathcal{C}$. Basically what we are doing is to show that SNARK can also be constructed via our methodology: parallel proof of multiplication gate ($\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{Prove}$) and wire consistency check ($\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$).

In this construction, $\mathcal{P}$ commits vector to its multilinear extension (MLE), which is the (unique) multilinear polynomial $f_{\boldsymbol{w}} : \mathbb{F}^d \to \mathbb{F}$ such that $\boldsymbol{w}(\boldsymbol{b}) = f_{\boldsymbol{w}}(\boldsymbol{b})$ for all $\boldsymbol{b} \in \{0,1\}^d$. Formally, it is defined as:

$$f_{\boldsymbol{w}}(X_1, \ldots, X_d) = \sum_{\boldsymbol{b} \in \{0,1\}^d} \chi_{\boldsymbol{b}}(X_1, \ldots, X_d) \cdot \boldsymbol{w}(\boldsymbol{b}).$$

where $\chi_{\boldsymbol{b}}(X_1, \ldots, X_d) = \prod_{i=1}^{d} \chi_{b_i}(X_i), \chi_1(X) = X$ and $\chi_0(X) = 1 - X$.

Besides MLE, the protocol realizes our functionality based on two gadgets $\mathcal{F}_{\mathsf{poly}}$ and $\mathcal{F}_{\mathsf{sc}}$. $\mathcal{F}_{\mathsf{poly}}$ checks the relation $R^{\mathsf{poly}}$ over $\mathbb{F}^d \times \mathcal{F} \times \mathbb{F}$, where $R^{\mathsf{poly}}(\boldsymbol{x}, f, y) := y \stackrel{?}{=} f(\boldsymbol{x})$. $\mathcal{F}_{\mathsf{sc}}$ enables a prover to convince a verifier of the validity of a statement of the form $t = \sum_{\boldsymbol{b} \in \{0,1\}^d} g(\boldsymbol{b})$ where $g : \mathbb{F}^d \to \mathbb{F}$. It has already been realized in paper [LFKN90]. To achieve zero-knowledge, $g$ is defined in the form $\prod_{i=0}^{2} g_i(\boldsymbol{S})$, that all the $g_i$'s, except $g_0$, are committed. Namely, $\mathcal{F}_{\mathsf{sc}}$ checks the relation $R^{\mathsf{sc}}(\boldsymbol{x}, \boldsymbol{u})$, with $\boldsymbol{x} \in \mathcal{F}$ and $\boldsymbol{u} \in \mathbb{F} \times \mathcal{F}^2$, that is formally defined as:

$$R^{\mathsf{sc}}(g_0, (t, g_1, g_2)) = 1 \Leftrightarrow g(S) = \prod_{i=0}^{2} g_i(\boldsymbol{S}) \wedge t = \sum_{\boldsymbol{b} \in \{0,1\}^d} g(\boldsymbol{b}).$$

The construction in Figure 17 is secure in the GGM and random oracle model, and it has only linear proving time. The size of the proof in this protocol is $\mathcal{O}(\log n)$, while it can be shrink to $\mathcal{O}(1)$ by applying another method of proving Hadamard product [Lip16], with a $\log n$ blow-up in prover's computation cost.

**Protocol $\Pi_{\mathsf{LegoSNARK}}$**

**Setup:** Upon receiving maximum degree $d$, sample a bilinear group and define $\langle \mathsf{group} \rangle := (\mathbb{G}_1, \mathbb{G}_2, q, g, e)$ where $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ and $g$ is a generator of $\mathbb{G}_1$. Sample random elements $s_1, \ldots, s_{d+1} \in \mathbb{F}_q$ and compute vector

$$\boldsymbol{\Sigma} := \{g^{\Pi_{i \in W} s_i}\}_{W \in 2^{[d]}},$$

and then output the public parameters $(\langle \mathsf{group} \rangle, \boldsymbol{\Sigma}, g^{s_{d+1}})$.

**Commit:** On input $\boldsymbol{w}$, $\mathcal{P}$ computes its multilinear extension $f_{\boldsymbol{w}}$ and uniformly samples $\tilde{r} \in \mathbb{F}$, and then sends $\mathcal{V}$ the commitment $[\![f_{\boldsymbol{w}}]\!] = g^{f_{\boldsymbol{w}}(s_1,\ldots,s_d)+\tilde{r}s_{d+1}}$; To commit a single value $w$, the commitment $[\![w]\!]$ is simply $g^{w+\tilde{r}s_{d+1}}$.

**Open:** On input $([\![w]\!], w, \tilde{r})$, $\mathcal{V}$ checks whether $[\![w]\!] = g^{w+\tilde{r}s_{d+1}}$.

**Prove:** For each gate $(\alpha, \beta, \gamma, T)$ in the public circuit $\mathcal{C}$, commitments to input wire vectors $[\![f_\alpha]\!]$ and $[\![f_\beta]\!]$ are also public:

- If $T = ADD$, every party locally computes $[\![f_\gamma]\!] = [\![f_\alpha]\!] \cdot [\![f_\beta]\!]$.
- If $T = MULT$:
  1. $\mathcal{P}$ computes $f_\gamma$ such that $f_\gamma[\boldsymbol{i}] = f_\alpha[\boldsymbol{i}] \cdot f_\beta[\boldsymbol{i}]$ for $\boldsymbol{i} \in \{0,1\}^d$. Then $\mathcal{P}$ calls $\mathsf{Commit}$ to obtain $[\![f_\gamma]\!]$.
  2. $\mathcal{V}$ picks a random point $\boldsymbol{r}$ and sends it to $\mathcal{P}$.
  3. $\mathcal{P}$ computes $t = f_\gamma(\boldsymbol{r})$ and calls $\mathsf{Commit}$ to obtain $[\![t]\!]$. Then $\mathcal{F}_{\mathsf{poly}}$ is invoked to check that $f_\gamma(\boldsymbol{r}) = t$.
  4. $\mathcal{P}$ calls $\mathcal{F}_{\mathsf{sc}}$ to prove that $t = \sum_{\boldsymbol{b} \in \{0,1\}^d} \widetilde{eq}(\boldsymbol{r}, \boldsymbol{b}) \cdot f_\alpha(\boldsymbol{b}) \cdot f_\beta(\boldsymbol{b})$, where $\widetilde{eq}(\boldsymbol{r}, \boldsymbol{b}) = 1$ when $\boldsymbol{r} = \boldsymbol{b}$, otherwise 0.

If any check fails, $\mathcal{V}$ aborts.

**Linear map:** Upon receiving $([\![f_{\boldsymbol{x}}]\!], [\![f_{\boldsymbol{y}}]\!], \mathbf{M})$, to prove that $\boldsymbol{x} = \mathbf{M}\boldsymbol{y}$:

1. Trusted parties compute the MLE of matrix $\mathbf{M}$, and form its commitment $[\![f_{\mathbf{M}}]\!]$, as well as corresponding proving and verification key.
2. $\mathcal{V}$ samples a random point $\boldsymbol{r}$ and sends to $\mathcal{P}$. Then $\mathcal{P}$ uses $\mathsf{Commit}$ to obtain $[\![g]\!]$, where $g(\boldsymbol{S}) = f_{\mathbf{M}}(\boldsymbol{r}, \boldsymbol{S})$.
3. $\mathcal{V}$ picks another random point $\boldsymbol{\sigma}$ and sends to $\mathcal{P}$. $\mathcal{P}$ computes $t = g(\boldsymbol{\sigma}) = f_{\mathbf{M}}(\boldsymbol{r}, \boldsymbol{\sigma})$ and commit $t$.
4. $\mathcal{F}_{\mathsf{poly}}$ is invoked to check that $g(\boldsymbol{\sigma}) = t$ and $f_{\mathbf{M}}(\boldsymbol{r}, \boldsymbol{\sigma}) = t$, and $\mathcal{V}$ calls $\mathsf{Open}$ to check the correctness of $[\![t]\!]$.
5. $\mathcal{P}$ computes $f_{\boldsymbol{x}}(\boldsymbol{r}) = k$ and calls $\mathsf{Commit}$ to obtain $[\![k]\!]$, and $\mathcal{V}$ calls $\mathsf{Open}$ to check its correctness.
6. $\mathcal{P}$ calls $\mathcal{F}_{\mathsf{sc}}$ to prove that $k = \sum_{\boldsymbol{b} \in \{0,1\}^d} g(\boldsymbol{b}) \cdot f_{\boldsymbol{y}}(\boldsymbol{b})$.

If any check fails, $\mathcal{V}$ aborts.

Figure 17: The protocol of $\mathsf{eSIMDZK}$ from LegoSNARK.