



Research Article

An Efficient ZK Compiler from SIMD Circuits to General Circuits*

Dung Bui

Université Paris Cité, CNRS, IRIF, Paris, France
bui@irif.fr

Haotian Chu

Shanghai Jiao Tong University, Shanghai, China
chtvii@sjtu.edu.cn

Geoffroy Couteau

Université Paris Cité, CNRS, IRIF, Paris, France
couteau@irif.fr

Xiao Wang

Northwestern University, Evanston, USA
wangxiao@northwestern.edu

Chenkai Weng

Arizona State University, Tempe, USA
chenkai.weng@asu.edu

Kang Yang

State Key Laboratory of Cryptology, Beijing, China
yangk@sklc.org

Yu Yu

Shanghai Jiao Tong University, Shanghai, China
yyuu@sjtu.edu.cn

Communicated by Benoit Libert and Carla Rafols Salvador

Received 21 February 2024 / Revised 28 October 2024 / Accepted 29 October 2024

Abstract. We propose a generic compiler that can convert any zero-knowledge (ZK) 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.

Electronic supplementary material The online version of this article (<https://doi.org/10.1007/s00145-024-09531-4>) contains supplementary material, which is available to authorized users.

*This paper was reviewed by Cyprien Delpech de Saint-Guilhem and Daniel Demmler

© The Author(s) 2024

Published online: 05 December 2024

- 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 $83.6\times$ 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 the 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 Liger 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.

Keywords. Zero-Knowledge Proof, General Compiler, SIMD ZK, VOLE-based ZK, Σ -protocol.

1. Introduction

Assume that the verification of a statement is represented as a public circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}$. 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-SNARK 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 [17, 18, 38]. A small communication footprint is achieved when the commitment is compressing and the proof is succinct. On the other hand, schemes like VOLE-based ZKPs [5, 23, 49, 52] 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, C) -SIMD circuit which contains B identical components of the circuit C . A SIMD-ZK proves that for input witnesses (w_1, \dots, w_B) , $C(w_i) = 0$ 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 $B \cdot |C|$. In more detail, denote by $\llbracket w \rrbracket$ a commitment to a vector w . Define a witness matrix $W = (w_1 \parallel \dots \parallel w_B)$. 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 $(\llbracket w^1 \rrbracket, \dots, \llbracket w^{|\mathcal{C}|} \rrbracket)$. In this way, for any gate $(\alpha, \beta, \gamma, \diamond)$ in \mathcal{C} and $\diamond \in \{\text{Add}, \text{Mult}\}$, the prover only needs to prove that $w^\gamma = w^\alpha \diamond w^\beta$. 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 [2, 17, 18, 20, 26, 50, 53]. 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) [27]. Define $z := (1, x, w)$ in which x and 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 z to the vectors of the left, right and output wires of multiplication gates a, b, c . Then the relation $a * b - c = 0$ can be expressed as $(\mathbf{L} \cdot z) * (\mathbf{R} \cdot z) - (\mathbf{O} \cdot z) = 0$. In this way, the ZKP is reduced to proving matrix-vector products on committed values. On the other hand, some ZKPs like [50, 53] proceed differently: they individually prove that $w^\alpha[i] = w^\beta[j]$ 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 [19]. Efforts are made to distribute the zk-SNARK proof generation among a set of provers [9, 33, 40, 45, 51], 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 recursive SNARKs [14, 35, 37] 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 [5, 23, 49, 52] 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 $\mathcal{F}_{\text{SIMDZK}}$ and denote it as $\mathcal{F}_{\text{eSIMDZK}}$. 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 $\mathbf{x} = \mathbf{M}\mathbf{y}$ for committed vectors (\mathbf{x}, \mathbf{y}) . The functionality $\mathcal{F}_{\text{eSIMDZK}}$ 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 $\mathcal{F}_{\text{SIMDZK}}$ 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 $\mathcal{F}_{\text{eSIMDZK}}$. Our compiler is a generalization of a few works including Liger [2,6] and LegoSNARK [18], which utilize RICS-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 $\mathcal{F}_{\text{eSIMDZK}}$ 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 [50], constant-round sublinear ZK from Σ -protocol [3], designated multi-verifier ZK from packed Shamir sharing [53], MPC-in-the-Head [2] and zk-SNARK from pairing [18]. 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 Sect. 2.2. Furthermore, we implement the SIMD compiler and evaluate the compilation of a VOLE-based ZK [50] that is previously designed for SIMD circuits. For a circuit of size $|\mathcal{C}| = 2^{27}$, it shows up to $83.6\times$ improvement on communication, compared to the general VOLE-ZK Quicksilver [52]. 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 [7, 8, 10, 11, 24, 31]. 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 [13]. Assume an NP relation that can be verified in time T and space S by a RAM program, Bangalore et al. [4] 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 [2].

Recent recursive zk-SNARK and incremental verifiable computation (IVC) propose succinct arguments for composed circuits, which can be evaluated step by step [14, 16, 35–37, 47]. 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 [35, 37].

1.3. Notation and Functionalities

Notation. Denote λ as the computational security parameters and $[1, m]$ as a set $\{1, 2, \dots, m\}$. For a vector $\mathbf{x} \in \mathbb{F}^B$ we define its i -th coordinate by x_i , and a vector $\mathbf{x}' := (f(0), \mathbf{x}) \in \mathbb{F}^{B+1}$ as the concatenation of a value $f(0) \in \mathbb{F}$ and the vector \mathbf{x} . 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 . $\text{negl}()$ is defined as a negligible function such that $\text{negl}(\lambda) = o(\lambda^{-c})$ 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, C) -SIMD circuit as a circuit that contains B copies of \mathcal{C} .

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

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

Vector oblivious linear evaluation $\mathcal{F}_{\text{VOLE}}$. This functionality works over a field \mathbb{F} , and upon receiving (**init**) from \mathcal{P} and \mathcal{V} , if \mathcal{V} is honest, then sample $\Delta \leftarrow \mathbb{F}$, else receive $\Delta \in \mathbb{F}$ from the adversary. Store Δ and ignore all subsequent (**init**) commands. Upon receiving (**extend**, n) from \mathcal{P} and \mathcal{V} , execute:

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

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

- Upon receiving input (**Commit**, \mathbf{w}) from \mathcal{P} and (**Commit**) from \mathcal{V} , pick a tag $\llbracket \mathbf{w} \rrbracket$ and store $(\llbracket \mathbf{w} \rrbracket, \mathbf{w})$ in the memory. Return $\llbracket \mathbf{w} \rrbracket$ to both parties.
- Upon receiving (**Open**, $\llbracket \mathbf{w} \rrbracket$), if a tuple $(\llbracket \mathbf{w} \rrbracket, \mathbf{w})$ was previously stored, output $(\llbracket \mathbf{w} \rrbracket, \mathbf{w})$ 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 Fig. 1. \mathcal{P} first groups and commits to the vectors of witnesses. Then it uses 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 [50] 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 [23, 52] that incur $\mathcal{O}(BC)$ complexity.

There are multiple ways to conduct the transformation from SIMD-ZK to general ZK. As discussed in Sect. 1, such constructions usually need a wire consistency check on top of SIMD-ZK. Taking AntMan [50] 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 $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{F}^B$ and wire indices $i, j \in [B]$, it aims to check whether they satisfy $\mathbf{w}_1[i] = \mathbf{w}_2[j]$. AntMan requires $\mathcal{O}(B^3)$ complexity for checking all combinations of $(i, j) \in [B] \times [B]$, which leads to a total communication

Functionality $\mathcal{F}_{\text{SIMDZK}}$

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

Commit: Upon receiving input (Commit, $\mathbf{w} \in \mathbb{F}^B$) from \mathcal{P} and (Commit) from \mathcal{V} , pick a tag $\llbracket \mathbf{w} \rrbracket$ and store $(\llbracket \mathbf{w} \rrbracket, \mathbf{w}, \text{ctr}_{\mathbf{w}} = 0)$ in the memory. Return $\llbracket \mathbf{w} \rrbracket$ to both parties.

Open: Upon receiving (Open, $\llbracket \mathbf{w} \rrbracket$), if a tuple $(\llbracket \mathbf{w} \rrbracket, \mathbf{w})$ was previously stored, output $(\llbracket \mathbf{w} \rrbracket, \mathbf{w})$ to \mathcal{V} ; otherwise abort.

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

Fig. 1. Functionality of SIMD ZK .

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 [53] 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 [50], 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 \mathbf{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 $\mathbf{w} \in \mathbb{F}^{Bm}$ and the values of left input wires across all multiplication gates $\mathbf{l} \in \mathbb{F}^{Bn}$. 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 $\mathbf{l}[i] = \mathbf{w}[\alpha_i]$. 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 $\mathbf{l} = \mathbf{L}\mathbf{w}$, where \mathbf{L} is public and parties have commitments $\{\llbracket \mathbf{L}_i \rrbracket\}_{i \in [n]}$ and $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [m]}$. In the context of SIMD-ZK protocols, values in \mathbf{l} and \mathbf{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 $\hat{\mathbf{r}} \in \mathbb{F}^{Bn}$ and convert the check of $\mathbf{l} \stackrel{?}{=} \mathbf{L}\mathbf{w}$ to the check of $\hat{\mathbf{r}}^\top \mathbf{l} \stackrel{?}{=} \hat{\mathbf{r}}^\top \mathbf{L}\mathbf{w}$. 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 $\hat{\mathbf{r}}$. To simplify the notation, we define a public vector $\mathbf{v}^\top = \hat{\mathbf{r}}^\top \mathbf{L}$, then rewrite the above relation as $\hat{\mathbf{r}}^\top \mathbf{l} \stackrel{?}{=} \mathbf{v}^\top \mathbf{w}$. If we define a circuit $\mathcal{C} : \mathbb{F}^{2n+2m+1} \rightarrow \mathbb{F}$ such that $\mathcal{C}(\hat{r}_1, \dots, \hat{r}_n, l_1, \dots, l_n, v_1, \dots, v_m, w_1, \dots, w_m, q) :$

$$\sum_{i \in [n]} \hat{r}_i \cdot l_i - \sum_{j \in [m]} v_j \cdot w_j - q,$$

then \mathcal{P} can prove the above statement by: (1) Divide each of the vectors in $(\hat{\mathbf{r}}, \mathbf{l}, \mathbf{v}, \mathbf{w})$ into length- B segments. Compute and commit to $\mathbf{q} := \sum_{i \in [n]} \hat{r}_i * l_i - \sum_{j \in [m]} v_j * w_j \in \mathbb{F}^B$. Prove the consistency between $(\hat{\mathbf{r}}, \mathbf{l}, \mathbf{v}, \mathbf{w}, \mathbf{q})$ by using a SIMD-ZK composed of B evaluations of the circuit \mathcal{C} . (2) prove that $\sum_i \mathbf{q}[i] = 0$. 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 \mathbf{q} , but it compromises the zero-knowledge requirements because \mathbf{q} is the linear combination of private circuit wire values. To tackle the problem, \mathcal{P} instead commits to a uniform vector $\tilde{\mathbf{r}} \in \mathbb{F}^B$ under the constraint that $\sum_{i \in [B]} \tilde{\mathbf{r}}[i] = 0$. It should be done before \mathcal{V} samples $\hat{\mathbf{r}}$ (else \mathcal{P} can break soundness). After \mathcal{P} commits to the mask vector, \mathcal{V} sends the challenge $\hat{\mathbf{r}}$ and the new SIMD circuit is defined to be

$$\begin{aligned} \mathcal{C}'(\hat{r}_1, \dots, \hat{r}_n, l_1, \dots, l_n, v_1, \dots, v_m, w_1, \dots, w_m, q, \tilde{r}) \\ = \sum_{i \in [n]} \hat{r}_i \cdot l_i - \sum_{j \in [m]} v_j \cdot w_j - q - \tilde{r} \end{aligned}$$

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

$$\mathbf{q} = \sum_{i \in [n]} \hat{\mathbf{r}} * l_i - \sum_{j \in [m]} v_j * w_j - \tilde{\mathbf{r}}.$$

The parties can now use the SIMD-ZK to prove B number of instances of \mathcal{C}' with committed inputs $[\hat{\mathbf{r}}_1], \dots, [\hat{\mathbf{r}}_n], [\mathbf{l}_1], \dots, [\mathbf{l}_n], [\mathbf{v}_1], \dots, [\mathbf{v}_m], [\mathbf{w}_1], \dots, [\mathbf{w}_m], [\mathbf{q}]$ and $[\tilde{\mathbf{r}}]$. Finally, the proof of $\sum_i \mathbf{q}[i] = 0$ is specific to the underlying commitment schemes. The naive way is to let \mathcal{P} fully open \mathbf{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 \mathbf{r} that is sampled after \mathcal{P} commits to $\tilde{\mathbf{r}}$. Assume that \mathbb{F} is an exponentially large field and a cheating prover commits to (\mathbf{l}, \mathbf{w}) such that $\mathbf{l} - \mathbf{L}\mathbf{w} \neq \mathbf{0}^{Bn}$. By Schwarz-Zippel, the probability that the erroneous values happen to be corrected by $\hat{\mathbf{r}}$ during the check of $\sum_i \mathbf{q}[i] \stackrel{?}{=} 0$ where $\mathbf{q} := \hat{\mathbf{r}}^\top \mathbf{l} - \hat{\mathbf{r}}^\top \mathbf{L}\mathbf{w}$ is $1/|\mathbb{F}|$, which is negligible.

Plugging in the protocol. For a general circuit with a total of $|\mathbf{w}| = Bm$ 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 $(\mathbf{l}, \mathbf{r}, \mathbf{o}) \in \mathbb{F}^{nB}$ be the batched wire values of left, right and output of multiplication gates in the circuit. The wire consistency can be proven by checking $(\mathbf{l} \stackrel{?}{=} \mathbf{L}\mathbf{w}, \mathbf{r} \stackrel{?}{=} \mathbf{R}\mathbf{w}, \mathbf{o} \stackrel{?}{=} \mathbf{O}\mathbf{w})$, where $(\mathbf{L}, \mathbf{R}, \mathbf{O}) \in$

Functionality $\mathcal{F}_{\text{eSIMDZK}}$

Public parameter: batch size B .

$\mathcal{F}_{\text{eSIMDZK}}$ supports all that $\mathcal{F}_{\text{SIMDZK}}$ supports and the following instruction.

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

Fig. 2. Functionality of extended SIMD zero-knowledge.

$\mathbb{F}^{nB \times mB}$ are public maps that describe the circuit connectivity. Furthermore, the SIMD-ZK protocol handles the rest of the multiplicative relation check $\mathbf{o} \stackrel{?}{=} \mathbf{l} * \mathbf{r}$. This scheme is captured in the extended SIMD-ZK functionality $\mathcal{F}_{\text{eSIMDZK}}$ shown in Fig. 2. Compared to the common SIMD-ZK functionality shown in Fig. 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 [50], it improves its communication complexity from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. In another case, for compressed Σ -protocols [3], 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 [53] can be improved from $\mathcal{O}(nC/B + n^2 B^2)$ 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 \mathbf{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 $\mathcal{C} = (\mathcal{C}_1, \dots, \mathcal{C}_{n'})$ using existing circuit partition methods [1, 43, 44]. For any consecutive segments \mathcal{C}_j and \mathcal{C}_{j+1} , let $(\mathbf{w}_j, \mathbf{l}_j, \mathbf{r}_j, \mathbf{o}_j)$ and $(\mathbf{w}_{j+1}, \mathbf{l}_{j+1}, \mathbf{r}_{j+1}, \mathbf{o}_{j+1})$ 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 $(\mathbf{w}_j, \mathbf{l}_j, \mathbf{r}_j, \mathbf{o}_j)$. 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 $\mathbf{M}\mathbf{w}_j = \tilde{\mathbf{w}}_{j+1}$, in which \mathbf{M} is a map that indicates the connectivity between \mathcal{C}_j and \mathcal{C}_{j+1} and $\tilde{\mathbf{w}}_{j+1}$ 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 $\mathcal{O}(|\mathbf{w}_j| + |\tilde{\mathbf{w}}_{j+1}|)$. 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 [50] 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. [3] (denoted as AC20) and is combined with a 2-round SHVZK for Hadamard product of [30]. 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, 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 [25, 53]. Restricting $B < n - 2t$ where t is the number of corrupted verifiers, it incurs $\mathcal{O}(nC)$ communication overhead for (B, C) -SIMD circuits and $\mathcal{O}(nC/B + n^2 B^2)$ 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 [2] and its follow-up work [4] 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 $\llbracket f \rrbracket$ for a commitment to the polynomial $f(\cdot)$. The AntMan protocol realizes $\mathcal{F}_{\text{SIMDZK}}$ as follows:

1. For each batch of B private inputs $\mathbf{w}_\alpha \in \mathbb{F}^B$, \mathcal{P} computes a degree- $(B - 1)$ polynomial f_α such that $f_\alpha(i) = \mathbf{w}_\alpha[i]$. \mathcal{P} commits to f_α so that \mathcal{P} and \mathcal{V} obtain $\llbracket f_\alpha \rrbracket$.
2. The parties process the circuit in topological order. For any batch of k addition gates with commitments to input wires $(\llbracket f_\alpha \rrbracket, \llbracket f_\beta \rrbracket)$, \mathcal{P} and \mathcal{V} locally computes the commitment to output wires by $\llbracket f_\gamma \rrbracket = \llbracket f_\alpha \rrbracket + \llbracket f_\beta \rrbracket$. For multiplication gates with input commitments $\llbracket f_\alpha \rrbracket$ and $\llbracket f_\beta \rrbracket$, \mathcal{P} computes $\mathbf{w}_\gamma = \mathbf{w}_\alpha * \mathbf{w}_\beta$ and a degree- $(B - 1)$ polynomial f_γ such that $f_\gamma(i) = \mathbf{w}_\gamma[i]$, $i \in [B]$. \mathcal{P} also computes $\hat{f}_\gamma(\cdot) = f_\alpha(\cdot) \cdot f_\beta(\cdot)$. \mathcal{P} commits to them by generating $\llbracket f_\gamma \rrbracket$ and $\llbracket \hat{f}_\gamma \rrbracket$.
3. For each multiplication gates with input and output wires (α, β, γ) , \mathcal{P} proves that $(\llbracket f_\alpha \rrbracket, \llbracket f_\beta \rrbracket, \llbracket \hat{f}_\gamma \rrbracket)$ is a multiplication triple and $\hat{f}_\gamma(i) = f_\gamma(i)$ for $i \in [B]$.
4. When a batch of k output wires α , \mathcal{P} opens the commitment to f_α , from which \mathcal{V} reconstructs \mathbf{w}_α .

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, 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 $(\mathbf{l}, \mathbf{r}, \mathbf{o})$ via SIMD-ZK. Additionally, it invokes the proof of linear map to check the wire consistency between $(\mathbf{l}, \mathbf{r}, \mathbf{o})$ and \mathbf{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 [52], which requires $\mathcal{O}(C)$ communication overhead. More details are shown in Sect. 4.1.

SIMD-ZK based on Pedersen commitment. We briefly present a SHVZK argument of knowledge for (B, C) -SIMD circuits which relies on the techniques of AC20 [3]. 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, 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 $\llbracket f \rrbracket = \mathbf{g}^{\mathbf{x}'} h^r$ 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 $\llbracket g \rrbracket$. For the vector of output wire values \mathbf{z} , we define $h(X) := f(X)g(X)$ and $\llbracket h \rrbracket = \mathbf{g}^{\mathbf{z}'} h^r$ where $\mathbf{z}' := (h(0), \mathbf{z}, h(B+1), \dots, h(2B)) \in \mathbb{F}^{2B+1}$. \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 Lagrange 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 $(\mathbf{x}, \mathbf{aux}) \in \mathbb{F}^{2B+1}$. The details of construction are shown in Sect. 4.2.

Compiling multi-verifier ZK. Yang et al. [53] proposed an non-interactive designed 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) [25] 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 [12, 15, 29] 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 [53] to convert its SIMD-ZK to general ZK by arranging all wire connection as a tuple $(\llbracket \mathbf{w}_1 \rrbracket, \llbracket \mathbf{w}_2 \rrbracket, i, j)$, indicating that $w_1[i] = w_2[j]$. 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 Ligerio. Our compiler is partially inspired by Ligerio [2], an MPC-in-the-head-based ZKP [32] that works for general circuits. At the core of Ligerio, \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 Ligerio and prove that it realizes $\mathcal{F}_{\text{SIMDZK}}$. Applying our SIMD compiler would result in the original Ligerio. We then identify a security issue when applying SIMD Ligerio to our memory-constrained framework designed for scalable ZK. Namely, although Ligerio 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, \dots, 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 $\mathcal{F}_{\text{SIMDZK}}$ 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 $\mathcal{F}_{\text{eSIMDZK}}$ which supports the proof of linear map, in addition to the normal SIMD-ZK functionality $\mathcal{F}_{\text{SIMDZK}}$. Based on the extended SIMD-ZK, we describe our compiler that

Protocol Π_{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 $\llbracket \mathbf{x} \rrbracket$ and $\llbracket \mathbf{y} \rrbracket$ are public, where $\mathbf{x} \in \mathbb{F}^{Bn}$ and $\mathbf{y} \in \mathbb{F}^{Bk}$.

Protocol:

1. \mathcal{P} uniformly samples a vector $\tilde{\mathbf{r}} \in \mathbb{F}^B$ such that $\sum_{i=1}^B \tilde{\mathbf{r}}[i] = 0$. Then $\mathcal{F}_{\text{SIMDZK}}$ is invoked to obtain its commitment $\llbracket \tilde{\mathbf{r}} \rrbracket$.
2. \mathcal{V} uniformly samples a vector $\hat{\mathbf{r}} \in \mathbb{F}^{Bn}$ and sends it to \mathcal{P} . Everyone computes $\mathbf{v} = \hat{\mathbf{r}}^T \mathbf{M} \in \mathbb{F}^{Bk}$. Then for $i \in [k]$, $\mathcal{F}_{\text{SIMDZK}}$ is invoked to construct $\llbracket \mathbf{v}_i \rrbracket$, where \mathbf{v}_i is the i -th B -sized vector of \mathbf{v} . In the same way, everyone can have access to $\{\llbracket \hat{\mathbf{r}}_i \rrbracket\}_{i \in [n]}$.
3. \mathcal{P} computes $\mathbf{q} \in \mathbb{F}^B$, such that $\mathbf{q}[i] = \sum_{j=1}^n \hat{\mathbf{r}}_j[i] \mathbf{x}_j[i] - \sum_{j=1}^k \mathbf{v}_j[i] \mathbf{y}_j[i] + \tilde{\mathbf{r}}[i]$. \mathcal{P} invokes $\mathcal{F}_{\text{SIMDZK}}$ to obtain $\llbracket \mathbf{q} \rrbracket$.
4. Define circuit

$$\begin{aligned} & \mathcal{C}_{\text{Lin}}(a_1, \dots, a_n, b_1, \dots, b_n, c_1, \dots, c_k, d_1, \dots, d_k, e, f) \\ & := \sum_{i=1}^n a_i \cdot b_i - \sum_{i=1}^k c_i \cdot d_i + e - f, \end{aligned}$$

then call $\mathcal{F}_{\text{SIMDZK}}.\text{Prove}(\mathcal{C}_{\text{Lin}}, \llbracket \hat{\mathbf{r}}_1 \rrbracket, \dots, \llbracket \hat{\mathbf{r}}_n \rrbracket, \llbracket \mathbf{x}_1 \rrbracket, \dots, \llbracket \mathbf{x}_n \rrbracket, \llbracket \mathbf{v}_1 \rrbracket, \dots, \llbracket \mathbf{v}_k \rrbracket, \llbracket \mathbf{y}_1 \rrbracket, \dots, \llbracket \mathbf{y}_k \rrbracket, \llbracket \tilde{\mathbf{r}} \rrbracket, \llbracket \mathbf{q} \rrbracket)$.

5. \mathcal{V} sends $(\text{Open}, \llbracket \mathbf{q} \rrbracket)$ to $\mathcal{F}_{\text{SIMDZK}}$, which returns \mathbf{q} to \mathcal{V} ; \mathcal{V} checks $\sum_{i=1}^B \mathbf{q}[i] = 0$ and aborts if the check fails.

Fig. 3. The protocol for extended SIMD ZK from SIMD ZK.

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 Fig. 3, which realizes the functionality $\mathcal{F}_{\text{eSIMDZK}}$. It is based on the $\mathcal{F}_{\text{SIMDZK}}$ 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 $\mathbf{M} \in \mathbb{F}^{Bn \times Bk}$ and two vectors $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{F}^{Bn}$ and $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_k) \in \mathbb{F}^{Bk}$ from \mathcal{P} , outputs 1-bit information to \mathcal{V} indicating whether $\mathbf{x} = \mathbf{M}\mathbf{y}$. 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 $\hat{\mathbf{r}} \in \mathbb{F}^{Bn}$ and converts the check of $\mathbf{x} \stackrel{?}{=} \mathbf{M}\mathbf{y}$ into $\hat{\mathbf{r}}^T \mathbf{x} \stackrel{?}{=} \mathbf{v}^T \mathbf{y}$, where $\mathbf{v}^T = \hat{\mathbf{r}}^T \mathbf{M}$. After dividing these vectors into length- B segments, \mathcal{P} and \mathcal{V} invoke the $\mathcal{F}_{\text{SIMDZK}}$ functionality of batch size B . \mathcal{P} inputs \mathbf{q} and proves the correctness of $\mathbf{q} = \sum_{i=1}^n \hat{\mathbf{r}}_i * \mathbf{x}_i - \sum_{j=1}^k \mathbf{v}_i * \mathbf{y}_i \in \mathbb{F}^B$.

Eventually it opens the commitment to \mathbf{q} and let \mathcal{V} check $\sum_{i=1}^B \mathbf{q}[i] \stackrel{?}{=} 0$. To ensure the privacy of \mathcal{P} , it needs to make sure that only opened commitment to \mathbf{q} does not reveal information of \mathbf{x} and \mathbf{y} . It does so by the random mask $\tilde{\mathbf{r}}$. The impact of this mask on soundness is negligible since it is committed before $\hat{\mathbf{r}}$ is sampled.

In terms of the cost, the protocol Π_{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 by the underlying $\mathcal{F}_{\text{SIMDZK}}$, the number of commitments is reduced to 1 size- B vector commitment. Parties invoke the **Prove** procedure from $\mathcal{F}_{\text{SIMDZK}}$ 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 $\sum_{i=1}^B \mathbf{q}[i] = 0$ for a committed vector \mathbf{q} without opening the commitment.

Theorem 1. *Protocol Π_{eSIMDZK} (Fig. 3) securely realizes the Functionality $\mathcal{F}_{\text{eSIMDZK}}$ (Fig. 2) in the $\mathcal{F}_{\text{SIMDZK}}$ -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 $\mathcal{F}_{\text{eSIMDZK}}$, and running a PPT adversary \mathcal{A} as a subroutine while emulating $\mathcal{F}_{\text{SIMDZK}}$ 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 Π_{eSIMDZK} as follows:

1. By emulating the (**Commit**) command of $\mathcal{F}_{\text{SIMDZK}}$, \mathcal{S} receives $\tilde{\mathbf{r}}$ from \mathcal{A} and sends a handler $\llbracket \tilde{\mathbf{r}} \rrbracket$ to \mathcal{A} .
2. \mathcal{S} uniformly samples $\hat{\mathbf{r}} \in \mathbb{F}^{Bn}$ and sends to \mathcal{A} . For $i \in [k]$, after receiving (**Commit**, \mathbf{v}_i) from \mathcal{A} , \mathcal{S} sends a handler $\llbracket \mathbf{v}_i \rrbracket$ to \mathcal{A} . Similarly, \mathcal{S} sends $\llbracket \hat{\mathbf{r}}_i \rrbracket$ to \mathcal{A} for $i \in [n]$.
3. After receiving (**Commit**, \mathbf{q}) from \mathcal{A} , \mathcal{S} emulates $\mathcal{F}_{\text{SIMDZK}}$ by sending \mathcal{A} another handler $\llbracket \mathbf{q} \rrbracket$.
4. \mathcal{S} receives (**Prove**, \mathcal{C} , $\tau_1, \dots, \tau_{2n+2k+2}$) from \mathcal{A} , and then checks whether τ_i for all $i \in [2n+2k+2]$ match their corresponding tags. For $i \in [B]$, \mathcal{S} checks whether $\sum_{j=1}^n \hat{\mathbf{r}}_j[i] \mathbf{x}_j[i] - \sum_{j=1}^k \mathbf{v}_j[i] \mathbf{y}_j[i] + \tilde{\mathbf{r}}[i] - \mathbf{q}[i]$ equals to 0 or not. If any check fails, \mathcal{S} aborts; otherwise sends **Pass** to \mathcal{A} .
5. \mathcal{S} emulates the (**Open**) command of $\mathcal{F}_{\text{SIMDZK}}$ and receives a handler τ from \mathcal{A} . If τ does not match $\llbracket \mathbf{q} \rrbracket$ or the vector \mathbf{q} previously sent by \mathcal{A} does not satisfy $\sum_{i=1}^B \mathbf{q}[i] = 0$, \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 \mathbf{r} accidentally corrects the wrong input of \mathcal{A} . Define $\mathbf{z} = \mathbf{M}\mathbf{y}$ and

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

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

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

since $f(x_1, \dots, x_{Bn})$ 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 $\llbracket \tilde{\mathbf{r}} \rrbracket$ to \mathcal{A} .
2. \mathcal{S} receives $\hat{\mathbf{r}}$ and (Commit, \mathbf{v}_i) from \mathcal{A} for $i \in [k]$. Then \mathcal{S} emulates $\mathcal{F}_{\text{SIMDZK}}$ by sending \mathcal{A} a handler $\llbracket \mathbf{v}_i \rrbracket$ for $i \in [k]$. In the same way, \mathcal{S} sends \mathcal{A} handlers $\{\llbracket \hat{\mathbf{r}}_i \rrbracket\}_{i \in [n]}$.
3. Then, \mathcal{S} plays the role of $\mathcal{F}_{\text{SIMDZK}}$ and sends a handler $\llbracket \mathbf{q} \rrbracket$ to \mathcal{A} .
4. \mathcal{S} receives (Prove, \mathcal{C} , $\tau_1, \dots, \tau_{2n+2k+2}$) from \mathcal{A} and checks whether $\{\tau_i\}_{i \in [2n+2k+2]}$ match their corresponding tags. Then \mathcal{S} queries $\mathcal{F}_{\text{eSIMDZK}}$. If check fails or $\mathcal{F}_{\text{eSIMDZK}}$ aborts, \mathcal{S} aborts; otherwise sends Pass to \mathcal{A} .
5. By emulating the (Open) command of $\mathcal{F}_{\text{SIMDZK}}$, \mathcal{S} uniformly samples a vector $\mathbf{q} \in \mathbb{F}^B$ such that $\sum_{i=1}^B \mathbf{q}[i] = 0$ and sends \mathbf{q} to \mathcal{A} .

The only difference between reality and the ideal world is the method of calculating vector \mathbf{q} . Following the constraint $\sum_{i=1}^B \mathbf{q}[i] = 0$, \mathcal{S} uniformly samples vector \mathbf{q} . While in reality, each entry of \mathbf{q} is masked by vector $\tilde{\mathbf{r}}$ chosen by \mathcal{P} . As a result, in both worlds, all entries except one of \mathbf{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. \square

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 \mathbf{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 \mathbf{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 \mathbf{M} between a vector of wire values \mathbf{x} , and \mathbf{w} , which should satisfy $\mathbf{x} = \mathbf{M}\mathbf{w}$. As shown in Fig. 4, along with the vector \mathbf{w} , \mathcal{P} also commits to $(\mathbf{l}, \mathbf{r}, \mathbf{o})$ which are the batches of input and output wire values of multiplication gates. Showing that $\mathbf{o} = \mathbf{l} * \mathbf{r}$ is enough to prove that all multiplication gates are computed correctly. Additionally, \mathcal{P} also proves the correctness of $(\mathbf{l} = \mathbf{L}\mathbf{w}, \mathbf{r} = \mathbf{R}\mathbf{w}, \mathbf{o} = \mathbf{O}\mathbf{w})$, in which $(\mathbf{L}, \mathbf{R}, \mathbf{O})$ 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 $\mathbf{0} = \mathbf{A}\mathbf{w}$ 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 $\mathcal{F}_{\text{eSIMDZK}}$. Regarding the cost analysis, \mathcal{P} commits to a total of $k + 3n_2$ size- B vectors to \mathcal{V} . They 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 \mathbf{w}' to be the wire values excluding the input and output wires of multiplication gates. Construct the witness

Protocol Π_{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 $T = \text{ADD}$, set $\mathbf{A}_i := \mathbf{I}_\alpha + \mathbf{I}_\beta - \mathbf{I}_\gamma$; \mathcal{P} sets $\mathbf{w}[\gamma] := \mathbf{w}[\alpha] + \mathbf{w}[\beta]$
 - If $T = \text{MULT}$, set $(\mathbf{L}_c, \mathbf{R}_c, \mathbf{O}_c) := (\mathbf{I}_\alpha, \mathbf{I}_\beta, \mathbf{I}_\gamma)$; \mathcal{P} sets $(\mathbf{l}[c], \mathbf{r}[c], \mathbf{o}[c]) := (\mathbf{w}[\alpha], \mathbf{w}[\beta], \mathbf{w}[\alpha] \cdot \mathbf{w}[\beta])$. 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 $(\mathbf{l}, \mathbf{r}, \mathbf{o}, \mathbf{w}) \in \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^K$.
2. \mathcal{P} splits wire values $(\mathbf{l}, \mathbf{r}, \mathbf{o}, \mathbf{w})$ into chunks of size B , i.e., $\{\mathbf{l}_i, \mathbf{r}_i, \mathbf{o}_i\}_{i \in [n_2]}$ and $\{\mathbf{w}_i\}_{i \in [k]}$, such that each element is in \mathbb{F}^B . $\mathcal{F}_{\text{eSIMDZK}}$ is invoked to obtain commitments $\{\llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}_{i \in [n_2]}$ and $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$.
3. Then, $(\text{LinearMap}, \{\llbracket \mathbf{l}_i \rrbracket\}_{i \in [n_2]}, \{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\text{eSIMDZK}}$ to check that $\mathbf{l} = \mathbf{L}\mathbf{w}$; similarly check that $\mathbf{r} = \mathbf{R}\mathbf{w}$, $\mathbf{o} = \mathbf{O}\mathbf{w}$, and that $\mathbf{0} = \mathbf{A}\mathbf{w}$.
4. Let circuit $\mathcal{C}_{\text{Mult}} : \mathbb{F}^3 \rightarrow \mathbb{F}$ such that $\mathcal{C}_{\text{Mult}}(x, y, z) := xy - z$. For $i \in [n_2]$, send $(\text{Prove}, \mathcal{C}_{\text{Mult}}, \llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket)$ to $\mathcal{F}_{\text{eSIMDZK}}$.

Fig. 4. Generic ZK in the $\mathcal{F}_{\text{eSIMDZK}}$ hybrid.

vector $\mathbf{w} = (\mathbf{w}' \parallel \mathbf{l} \parallel \mathbf{r} \parallel \mathbf{o})$ and prove the wiring consistency by proving $\mathbf{0} = \mathbf{A}'\mathbf{w}$, in which $\mathbf{A}' \in \mathbb{F}^{K \times K}$ is a map that describes the circuit wire connectivity.

Theorem 2. *The Protocol Π_{compiler} (Fig. 4) securely realizes the Functionality \mathcal{F}_{ZK} in the $\mathcal{F}_{\text{eSIMDZK}}$ -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 Π_{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 (Commit) command of $\mathcal{F}_{\text{eSIMDZK}}$, \mathcal{S} receives $\{\mathbf{l}_i, \mathbf{r}_i, \mathbf{o}_i\}_{i \in [n_2]}$ and $\{\mathbf{w}_i\}_{i \in [k]}$ from \mathcal{A} and sends \mathcal{A} handlers $\{\llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}_{i \in [n_2]}$ and $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$.
3. After receiving (LinearMap, $\{\tau_i\}_{i \in [n_2+k]}, \mathbf{L})$ from \mathcal{A} , \mathcal{S} checks whether $\{\tau_i\}_{i \in [n_2]}$ match $\{\llbracket \mathbf{l}_i \rrbracket\}_{i \in [n_2]}$ and $\{\tau_i\}_{i \in [n_2+1, n_2+k]}$ matches $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$. Then, \mathcal{S} checks whether $\mathbf{l} = \mathbf{L}\mathbf{w}$. If any check fails, \mathcal{S} aborts; otherwise, \mathcal{S} sends Pass to \mathcal{A} . Similarly, \mathcal{S} handles other three (LinearMap) commands from \mathcal{A} .
4. For $i \in [n_2]$, \mathcal{S} receives (Prove, \mathcal{C} , τ_1, τ_2, τ_3) from \mathcal{A} and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{\llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}$. In each round, \mathcal{S} also checks that $\mathbf{l}_i[j] \cdot \mathbf{r}_i[j] = \mathbf{o}_i[j]$ 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 Π_{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 functionality $\mathcal{F}_{\text{eSIMDZK}}$, \mathcal{S} sends \mathcal{A} handlers $\{\llbracket \mathbf{I}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}_{i \in [n_2]}$ and $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$.
3. After receiving (LinearMap, $\{\tau_i\}_{i \in [n_2+k]}$, \mathbf{L}) from \mathcal{A} , \mathcal{S} checks whether $\{\tau_i\}_{i \in [n_2]}$ match $\{\llbracket \mathbf{I}_i \rrbracket\}_{i \in [n_2]}$ and $\{\tau_i\}_{i \in [n_2+1, n_2+k]}$ matches $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$. Then, \mathcal{S} queries \mathcal{F}_{ZK} . If check fails or \mathcal{F}_{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 (Prove, \mathcal{C} , τ_1 , τ_2 , τ_3) from \mathcal{A} and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{\llbracket \mathbf{I}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}$. In each round, \mathcal{S} also queries \mathcal{F}_{ZK} . If any check fails or \mathcal{F}_{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. \square

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 [7, 27, 37, 41], our construction does not rely on any complicated assumption other than the realization of $\mathcal{F}_{\text{eSIMDZK}}$ with the parameter $\tau_{\max} > 1$. The protocol is shown in Fig. 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 $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{\lceil |\mathcal{C}|/S \rceil}$), 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 α, β, γ similarly as the previous compiler. If $T = DEL$, \mathcal{P} adds gate α 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

Protocol $\Pi_{\text{small-space}}$

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 \mathbf{x} such that $\mathcal{C}(\mathbf{x}) = 0$.

Protocol:

1. Let $h() : \mathbb{Z} \rightarrow \mathbb{Z}$ be a function map wire indices to physical indices and \mathbf{w} be a dynamic list storing wire value to be dealt with in the current round. Initially, set $h(i) = i$ and $\mathbf{w}[i] = \mathbf{x}[i]$ for all $i \in [|\mathbf{x}|]$. Define function $\text{lm}() : f \rightarrow \mathbb{Z}$, returning the maximum index that f has the definition.
2. Let $\mathcal{D} = \emptyset$ and $W = \text{lm}(h) + S$. Initialize $\mathbf{L}, \mathbf{R}, \mathbf{O}, \mathbf{A}$ 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 $T = \text{ADD}$, set $h(\gamma) := \text{lm}(h) + 1$, compute $\mathbf{r} := \mathbf{I}_{h(\alpha)} + \mathbf{I}_{h(\beta)} - \mathbf{I}_{h(\gamma)} \in \mathbb{F}^W$ and append \mathbf{r} to \mathbf{A} . \mathcal{P} sets $\mathbf{w}[h(\gamma)] := \mathbf{w}[h(\alpha)] + \mathbf{w}[h(\beta)]$
 - If $T = \text{MULT}$, set $h(\gamma) := \text{lm}(h) + 1$, 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 $\mathbf{w}[h(\gamma)] := \mathbf{w}[h(\alpha)] \cdot \mathbf{w}[h(\beta)]$ and append values $\mathbf{w}[h(\alpha)], \mathbf{w}[h(\beta)], \mathbf{w}[h(\gamma)]$ to vectors $\mathbf{l}, \mathbf{r}, \mathbf{o}$ respectively.
 - If $T = \text{DEL}$, add α to \mathcal{D} .
 Suppose that there are $S_1 = s_1B$ addition gates and $S_2 = s_2B$ multiplication gates ($S = S_1 + S_2$), and after processing S gates, $|\mathbf{w}| = kB$. $\mathbf{A} \in \mathbb{F}^{S_1 \times W}$ and $\mathbf{L}, \mathbf{R}, \mathbf{O} \in \mathbb{F}^{S_2 \times W}$ are public.
3. \mathcal{P} splits wire values $(\mathbf{l}, \mathbf{r}, \mathbf{o}, \mathbf{w})$ into chunks of size B , i.e., $\{\mathbf{l}_i, \mathbf{r}_i, \mathbf{o}_i\}_{i \in [s_2]}$ and $\{\mathbf{w}_i\}_{i \in [k]}$, such that each element is in \mathbb{F}^B . $\mathcal{F}_{\text{eSIMDZK}}$ is invoked to obtain commitments $\{\llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}_{i \in [s_2]}$ and $\{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}$.
4. Then, $(\text{LinearMap}, \{\llbracket \mathbf{l}_i \rrbracket\}_{i \in [n_2]}, \{\llbracket \mathbf{w}_i \rrbracket\}_{i \in [k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\text{eSIMDZK}}$ to check that $\mathbf{l} = \mathbf{L}\mathbf{w}$; similarly check that $\mathbf{r} = \mathbf{R}\mathbf{w}$, $\mathbf{o} = \mathbf{O}\mathbf{w}$, and that $\mathbf{0} = \mathbf{A}\mathbf{w}$.
5. Let circuit $\mathcal{C}_{\text{Mult}} : \mathbb{F}^3 \rightarrow \mathbb{F}$ such that $\mathcal{C}_{\text{Mult}}(x, y, z) := xy - z$. For $i \in [s_2]$, $(\text{Prove}, \mathcal{C}_{\text{Mult}}, \llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket)$ is sent to $\mathcal{F}_{\text{eSIMDZK}}$.
6. Let $\mathcal{R} = \text{Domain}(h) \setminus \mathcal{D}$. 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 \mathbf{w}' such that for each $\mathbf{w}'[h'(i)] = \mathbf{w}[h(i)]$. Append 0 to \mathbf{w}' and $\mathbf{0}$ to \mathbf{H} until the size of \mathbf{w}' becomes a multiple of B . Suppose that $|\mathbf{w}'| = k'B$, and then \mathcal{P} calls Commit to obtain $\{\llbracket \mathbf{w}'_i \rrbracket\}_{i \in [k']}$. Update $(h, \mathbf{w}) := (h', \mathbf{w}')$.
8. Both parties call $(\text{LinearMap}, \{\llbracket \mathbf{w}'_i \rrbracket\}_{i \in [k]}, \{\mathbf{w}_i\}_{i \in [k]}, \mathbf{H})$ to check the consistency between \mathbf{w} and \mathbf{w}' .
9. If more gates need to be processed, jump to step 2.

Fig. 5. Generic ZK in limited-memory scenario.

$\mathcal{F}_{\text{eSIMDZK}}.\text{LinearMap}$, \mathcal{P} proves that the committed wire values belongs 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 \mathbf{o}_i as the output of subcircuit \mathcal{C}_i , and circuit input \mathbf{x} is denoted as \mathbf{o}_0 . In each round, we call $\mathcal{F}_{\text{eSIMDZK}}.\text{Prove}$ to complete the proof for \mathcal{C}_i and $\mathcal{F}_{\text{eSIMDZK}}.\text{LinearMap}$ to prove the transformation between $\llbracket \mathbf{o}_{i-1} \rrbracket$ and $\llbracket \mathbf{o}_i \rrbracket$. As each subcircuit contains at most S gates, proving \mathcal{C}_i requires $\mathcal{O}(S)$ space. And also, using $\mathcal{F}_{\text{eSIMDZK}}.\text{LinearMap}$ to prove the consistency between $\llbracket \mathbf{o}_{i-1} \rrbracket$ and $\llbracket \mathbf{o}_i \rrbracket$ requires $\mathcal{O}(|\mathbf{o}_{i-1}| + |\mathbf{o}_i|)$ space, so the space complexity of each round

is $\mathcal{O}(S + |\mathbf{o}_{i-1}| + |\mathbf{o}_i|)$. As a result, the overall space complexity is $\mathcal{O}(S + \max\{|\mathbf{o}_{i-1}| + |\mathbf{o}_i|\}_{i \in [|C|/S]})$. Since in the plaintext evaluation of \mathcal{C} , only active wire value needs to be read into the memory, memory upper bound $M \geq \max\{|\mathbf{o}_0|, |\mathbf{o}_1|, |\mathbf{o}_2|, \dots, |\mathbf{o}_{\lceil |C|/S} |\}$. By choosing $S < M$, we can conclude that the space complexity of the protocol is $\mathcal{O}(M)$ (Fig. 6).

4. Efficient Instantiations of Our Compiler

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

- A ZKP based on vector oblivious linear evaluation [50].
- A zk-SNARK from Σ -protocol [3].
- A designated multi-verifier ZKP based on packed Shamir secret sharing and recursive inner product check [53].

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 [50] which is restricted by $\tau_{\max} = 1$ thus does not fit into the second compiler for scalable ZK. In Supplementary Material A.5 and A.6, we additionally describe a SIMD-ZK that is extracted from an MPC-in-the-head scheme Ligerio [2], and a construction from LegoSNARK [18]. Both our compilers are generalizations of them and a follow-up work [4]. We show that Ligerio SIMD-ZK perfectly fits our compilers and discuss extra caution that need to take when compiling Ligerio.

4.1. AntMan++: Sublinear Designated-Verifier ZK

AntMan [50] is a sublinear VOLE-based ZK proof for SIMD circuits, which only requires communicating $\mathcal{O}(B + |C|)$ field elements to prove a (B, 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 $\mathcal{O}(|C|/B + B)$ field elements, and the cost to check the wire-value consistency is $\mathcal{O}(B^3)$, which leads to $\mathcal{O}(|C|^{3/4})$ communication complexity in optimal. It is the only sublinear-communication VOLE-ZK protocol for proving an arbitrary circuit. In AntMan [50], the information-theoretic polynomial authentication code $\Pi_{\text{IT-PAC}}^k$ serves as a polynomial commitment scheme. For arbitrary degree- k polynomial $f(\cdot)$ known by \mathcal{P} , an IT-PAC $\llbracket f(\cdot) \rrbracket$ 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 $\Pi_{\text{IT-PAC}}$. As shown in Fig. 7, the protocol is designed in the $(\mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{Com}})$ -hybrid model. It adopts additively homomorphic encryption (AHE) scheme to obliviously evaluate a polynomial, where the

Protocol Π_{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 $\alpha_1, \dots, \alpha_B \in \mathbb{F}$ be B distinct elements that are fixed for the whole protocol execution. Both parties invoke `Initialize()` in IT-PAC to obtain τ_1 .

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

Commit: On input $\mathbf{w} \in \mathbb{F}^B$, \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 $(\langle b \rangle, \tau_2) \leftarrow \text{PreGen}(f)$. \mathcal{P} obtains $\langle b \rangle$ and \mathcal{V} obtains τ_2 . If Λ has been revealed, invoke $(M, K) \leftarrow \text{Gen}(\tau_1, \tau_2)$. \mathcal{P} holds M and \mathcal{V} holds K .

Open: On input $(\llbracket f(\cdot) \rrbracket, f(\cdot), \Lambda)$, both parties compute that $\llbracket \mu \rrbracket := \llbracket f(\Lambda) \rrbracket - f(\Lambda)$. Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ be a random oracle. \mathcal{P} sends $H(M_\mu)$ to \mathcal{V} who checks whether $H(M_\mu) = H(K_\mu)$.

Fig. 6. The protocol of SIMDZK from AntMan.

polynomial is known by \mathcal{P} and the secret point Λ 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 Λ is correct. Instead of using the zero-knowledge proof of knowledge for the proof of validity (as done in several MPC protocols [21, 34]), AntMan utilizes a simple commit-and-open approach. Specifically, \mathcal{V} first commits to the randomness that are used to generate the HE ciphertexts $\langle \Lambda^1 \rangle, \dots, \langle \Lambda^k \rangle$. 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 $\langle \Lambda^1 \rangle, \dots, \langle \Lambda^k \rangle$. 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.

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 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 Π_{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 \mathbf{r} is public and witness \mathbf{x} is private, and the IT-PACs of two vectors are known to both parties. After the secret evaluation point Λ is revealed, both parties can locally calculate $f_{\mathbf{r}}(\Lambda)$ because \mathbf{r} is

Protocol $\Pi_{\text{IT-PAC}}^k$

Let $\text{AHE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ be an additively homomorphic encryption scheme. Suppose that two parties \mathcal{P} and \mathcal{V} have already agreed a set of public parameters $\text{par} = \text{Setup}(1^\lambda)$ and global key $\Delta \in \mathbb{F}$. Let \mathbf{G} be a PRG. Let k be the maximum degree of the polynomials committed in each IT-PAC.

Initialize.

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

Pre-Gen. On input f ,

3. \mathcal{P} and \mathcal{V} sends (extend) to $\mathcal{F}_{\text{Vole}}$, 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 (Commit) command of \mathcal{F}_{Com} with inputs $\langle b \rangle$, which returns a handle τ_2 to \mathcal{V} .

Gen. On input (τ_1, τ_2) ,

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

Fig. 7. Protocol for generating IT-PACs without ZK proofs in the $(\mathcal{F}_{\text{Vole}}, \mathcal{F}_{\text{Com}})$ -hybrid model .

known. Via the additively homomorphic property of IT-PACs, both parties compute $f_r(\Lambda) \cdot \llbracket \mathbf{x} \rrbracket$, which is also the IT-PAC of Hadamard product of \mathbf{r} and \mathbf{x} . In this way, both parties compute $n + k$ IT-PACs and add them up to obtain $\llbracket \mathbf{q} \rrbracket$. In the end, according to the protocol in Fig. 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 $\mathcal{O}(|\mathcal{C}|^{1/2})$.

The full description of SIMD AntMan is shown in Protocol Figs. 6 and 8.

Performance evaluation. We implement the AntMan++ protocol and benchmark its performance. Its homomorphic encryption (HE) is supported by the Microsoft SEAL [46] and other cryptographic building blocks are from EMP-toolkits [48]. 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 [50]. The performance of AntMan++ is not affected by the circuit structure and we benchmark with layered circuits for convenience. In all

Protocol Π_{AntMan} (Cont.)

Prove: On input $(\mathcal{C}, \llbracket \mathbf{w}_1 \rrbracket, \dots, \llbracket \mathbf{w}_m \rrbracket)$, \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 $\llbracket f \rrbracket$ and $\llbracket g \rrbracket$:
 - If $T = \text{ADD}$, both parties locally compute output IT-PAC $\llbracket h \rrbracket = \llbracket f \rrbracket + \llbracket g \rrbracket$.
 - If $T = \text{MULT}$, \mathcal{P} computes a degree- $(2B-2)$ polynomial $\tilde{h}(\cdot) := f(\cdot) \cdot g(\cdot) \in \mathbb{F}[X]$ and a degree- $(B-1)$ polynomial $h(\cdot)$ such that $h(\alpha_i) = \tilde{h}(\alpha_i)$ for all $i \in [B]$. Then, \mathcal{P} and \mathcal{V} run sub-protocol $\Pi_{\text{PAC}}^{(2B-2)}$ to generate two IT-PACs $\llbracket h(\cdot) \rrbracket$ and $\llbracket \tilde{h}(\cdot) \rrbracket$.

As there are n_2 multiplication gates, the commitments of their outputs are denoted as $\llbracket h_1 \rrbracket, \dots, \llbracket h_{n_2} \rrbracket$. Consequently, their degree- $(2B-2)$ polynomials are denoted as $\llbracket \tilde{h}_1 \rrbracket, \dots, \llbracket \tilde{h}_{n_2} \rrbracket$.
 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 $\llbracket r(\cdot) \rrbracket$ and $\llbracket s(\cdot) \rrbracket$.
 3. \mathcal{V} samples $\text{seed} \leftarrow \{0, 1\}^\lambda$ and sends it to \mathcal{P} . Then, two parties compute $(\chi_1, \dots, \chi_{n_2}) := \text{Hash}(\text{seed}) \in \mathbb{F}^{n_2}$.
 4. \mathcal{P} and \mathcal{V} locally compute $\llbracket h(\cdot) \rrbracket := \sum_{j=1}^{n_2} \chi_j \cdot \llbracket h_j(\cdot) \rrbracket + \llbracket r(\cdot) \rrbracket$ and $\llbracket \tilde{h}(\cdot) \rrbracket := \sum_{j=1}^{n_2} \chi_j \cdot \llbracket \tilde{h}_j(\cdot) \rrbracket + \llbracket s(\cdot) \rrbracket$. Then, \mathcal{P} sends the polynomial pair $(h(\cdot), \tilde{h}(\cdot))$ to \mathcal{V} , who checks that $h(\cdot), \tilde{h}(\cdot)$ have the degrees $B-1$ and $2B-2$ respectively and $h(\alpha_i) = \tilde{h}(\alpha_i)$ for all $i \in [1, t]$.
 5. \mathcal{P} and \mathcal{V} run $\text{Gen}(\tau_1, \tau_2)$ to open A 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 $\text{DVZK} \left\{ (\llbracket f_j(A) \rrbracket, \llbracket g_j(A) \rrbracket, \llbracket \tilde{h}_j(A) \rrbracket)_{j \in [n_2]} \mid \forall j \in [n_2], \tilde{h}_j(A) = f_j(A) \cdot g_j(A) \right\}$.
 7. \mathcal{P} and \mathcal{V} locally compute $[\mu] := [h(A)] - h(A)$ and $[\nu] := [\tilde{h}(A)] - \tilde{h}(A)$. Then, two parties run Open to check that $\mu = 0$ and $\nu = 0$.
 8. Let $\llbracket v(\cdot) \rrbracket$ be the IT-PAC associated with the output values circuit \mathcal{C} . \mathcal{P} and \mathcal{V} run Open to check $v(A) = 0$.
- If any check fails, \mathcal{V} aborts.

Fig. 8. The protocol of SIMDZK from AntMan (Cont.).

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 [52] and use its default parameter setting in [48]. We do not compare with AntMan [50] because it only proves SIMD circuits.

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.

Table 1. Performance of AntMan++ with variable batch size. .

$\log_2 B$	Communication (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 [52]	1087.23		185.43	

Benchmarked with 1 thread, 50 Mbps bandwidth and circuit size $C = 2^{27}$

Table 2. Performance of AntMan++ with variable threads and bandwidth .

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 [52]	900.47	361.29	181.63	91.9

Benchmarked with circuit size $C = 2^{27}$ and batch size $B = 2^{11}$. Numbers are in seconds

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

To showcase the versatility of our framework, we present an SHVZK argument of knowledge for (B, C) - SIMD circuits, and compile it to a constant-round square-root size argument for general circuit. This construction is based on the work of Attema–Cramer [3]. We note that [3] also mention (in a remark) that their techniques yield a constant-round sublinear argument; however, our approach achieves better parameters. Our SIMD-ZK has 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 polynomials of degree $\mathcal{O}(B)$ while it is $\mathcal{O}(1)$ interpolations of polynomials of degree $\mathcal{O}(|\mathcal{C}|)$ in [3]. Moreover, for a special type of circuit, i.e for (B, C) -SIMD circuit, our protocol has only

$\mathcal{O}(\mathcal{C} + \sqrt{B})$ communication. We provide a more in-depth comparison in Supplementary Material A.3.

Sublinear ZK Argument for Linear Form Evaluation. Given a linear form $L : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p$, consider the relation

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

Let $P \in \mathbb{G}$ be a commitment to $\mathbf{x} \in \mathbb{Z}_p^n$. The prover wants to convince the verifier that $y = L(\mathbf{x})$, while keeping \mathbf{x} secret. To achieve this, [3] divides the witness vector $\mathbf{x} \in \mathbb{Z}_p^n$ into 2 parts then recursively run the 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. It is known [30] that there is a trade-off between communication cost and the number of rounds, and this protocol can be generalized by dividing the witness into $k = \mathcal{O}(\sqrt{n})$ parts, yielding a 5-round protocol with sublinear communication (this was also mentioned in [3]). We denote the 5-round sublinear ZK argument for linear form evaluation by $\Pi(\llbracket \mathbf{x} \rrbracket, L_i, 0; \mathbf{x})$. We provide a formal description of this protocol in the Supplementary Material A.3.

Amortization. We now overview some amortizations techniques of [3] which allow the prover to prove correctness of (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, with negligible communication overhead compared to a single check Π .

Compressing many nullity checks $\Pi_{\text{zeros}}(\llbracket \mathbf{x} \rrbracket, L_1, L_2, \dots, L_s, 0; \mathbf{x})$: Given $P = \mathbf{g}^{\mathbf{x}} h^r$, s linear functions $L_i : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p$, \mathcal{P} can show that $L_i(\mathbf{x}) = 0$ for all $i \in [1, s]$ at the cost of one single check plus one \mathbb{Z}_p challenge from \mathcal{V} to \mathcal{P} .

Amortizing over many commitments $\Pi_{\text{Am}}(\llbracket \mathbf{x}_i \rrbracket, L, y_i; \mathbf{x}_i)_{i \in [1, s]}$: Given $P_i = \mathbf{g}^{\mathbf{x}_i} h^{r_i}$ for $i \in [1, s]$, the prover wants to show 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.

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 $j \in [1, m]$. The batch argument is based on algebraic interpolation polynomial and the ZK proof Π which proves the correct evaluation of linear form (consider Π 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 $\mathbf{x}_j := (f_j(0), x_{j,1}, x_{j,2}, \dots, x_{j,B}) \in \mathbb{Z}_p^{B+1}$ and $\mathbf{y}_j := (g_j(0), y_{j,1}, y_{j,2}, \dots, y_{j,B}) \in \mathbb{Z}_p^{B+1}$. 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 $\mathbf{z}_j := (h(0), z_{j,1}, z_{j,2}, \dots, z_{j,B}, h(B+1), \dots, h(2B)) \in \mathbb{Z}_p^{2B+1}$. \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 to prover.
- \mathcal{P} reveals $(f_j(c), g_j(c), h_j(c))_{j \in [1, m]}$. \mathcal{V} now then checks $h_j(c) \stackrel{?}{=} f_j(c)g_j(c)$. \mathcal{P} can cheat with probability at most $(2B)/(p - B)$. Denote $L_c : \mathbb{Z}_p^{B+1} \rightarrow \mathbb{Z}_p$, $L'_c :$

- $\mathbb{Z}_p^{2B+1} \rightarrow \mathbb{Z}_p$ 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 $\Pi_{\text{Am}}(\llbracket \mathbf{x}_j \rrbracket, \llbracket \mathbf{y}_j \rrbracket, L_c, f_j(c), g_j(c); \mathbf{x}_j, \mathbf{y}_j)_{j \in [1, m]}$ and $\Pi_{\text{Am}}(\llbracket \mathbf{z}_j \rrbracket, L'_c, h_j(c); \mathbf{z}_j)_{j \in [1, m]}$.

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

Instantiation of SIMD ZK. We achieve a ZK proof for (B, \mathcal{C}) -SIMD circuits with communication $\mathcal{O}(|\mathcal{C}| + \sqrt{B})$. Concretely, 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 cost is less than 3 times the instantiation of Π over the committed vector of length $2B$ which has $\mathcal{O}(\sqrt{B})$ communication (Fig. 9).

Sublinear Circuit Satisfiability. 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 (Sect. 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 [2, 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 $\llbracket o'_j \rrbracket, \llbracket o_j \rrbracket$ which committed of two vectors \mathbf{o}'_j and \mathbf{o}_j . While $\mathbf{o}'_j := (r, o_{1,j}, o_{2,j}, \dots, o_{B,j}) \in \mathbb{Z}_p^{B+1}$ and $\mathbf{o}_j := (h_j(0), o_{1,j}, o_{2,j}, \dots, o_{B,j}, h_j(B+1), \dots, h_j(2B)) \in \mathbb{Z}_p^{2B+1}$ where $r \in \mathbb{Z}_p$. Prover therefore needs to prove that $\llbracket o'_j \rrbracket / \llbracket o_j \rrbracket$ is the commitment of vector which having the power 0s of the set of generators $\{g_2, g_3, \dots, g_{B+1}\}$. By the method described earlier, this is handled by a Π_{Am} for checking many nullities. The protocol Π_{ZKPed} of our sublinear ZK is shown in Figure 14 of Supplementary Material.

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 under the discrete logarithm assumption.*
- *The number of rounds is 7.*
- *The size of CRS is $\mathcal{O}(\sqrt{|\mathcal{C}|})$ random elements of \mathbb{G} .*
- *Computation is dominated by $\mathcal{O}(\sqrt{|\mathcal{C}|})$ 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

Protocol $\Pi_{\text{SIMD}-\text{Ped}}$

Public inputs. A general circuit \mathcal{C} over \mathbb{Z}_p , where \mathcal{C} contains k addition gates, n multiplication gates and m input gates. For B executions of circuit \mathcal{C} , \mathcal{P} and \mathcal{V} pack B same-type gates into a group in a straightforward way. In particular, for an index i , the parties pack the i -th input/output/multiplication/addition gates from all B executions of circuit \mathcal{C} into a group.

Private input. \mathcal{P} holds B witnesses $\mathbf{w}_1, \dots, \mathbf{w}_B \in \mathbb{Z}_p^m$ such that $\mathcal{C}(\mathbf{w}_i) = 0$ for all $i \in [1, B]$.

Commit: For each group $j \in [1, |\mathcal{C}|]$, \mathcal{P} defines commitments for tuple of left wire values $(l_{1,j}, l_{2,j}, \dots, l_{B,j})$, right wire values $(r_{1,j}, r_{2,j}, \dots, r_{B,j})$ and output wire values $(o_{1,j}, o_{2,j}, \dots, o_{B,j})$ in the following way

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

Prove:

- **Correctness check of multiplication gates** is proven by using the batch multiplication argument (described above).
- **Consistency check of output gates** for output gates, \mathcal{P} wants to show that the value of all output gates is 0s. It means that when $j = |\mathcal{C}|$, $\llbracket o_j \rrbracket$ is the commitment of vector $\mathbf{o}_j = (r, 0, 0, \dots, 0) \in \mathbb{Z}_p^{B+1}$ (w.r.t addition output gates) or $\mathbf{o}_j = (h_j(0), 0, 0, \dots, 0, h_j(B+1), \dots, h_j(2B)) \in \mathbb{Z}_p^{2B+1}$ (multiplication output gates). Consider $\text{crs} = (g_1, g_2, \dots, g_{2B+3})$ then \mathcal{P} has to convince that in $\llbracket o_j \rrbracket$ the powers of set of generator $\{g_2, g_3, \dots, g_{B+1}\}$ are zeros. This constraint actually is a batch of B -nullity checks for different linear forms over one committed vector. \mathcal{P} and \mathcal{V} then run Π_{zeros} for many nullity checks.

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

Fig. 9. The protocol of SIMDZK from [3].

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 [3,30]. Note that we can define $\llbracket x \rrbracket$ in functionality Fig. 2 being Pedersen 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.

4.3. Multi-Verifier ZK

Yang et al. propose a non-interactive designated multi-verifier ZK (MVZK) proof [53]. In this protocol, a prover \mathcal{P} validates a statement to n verifiers $(\mathcal{V}_1, \dots, \mathcal{V}_n)$ with its private

Protocol Π_{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 $t + 2B - 1 \leq n$. $\alpha_1, \dots, \alpha_n, \gamma_1, \dots, \gamma_B \in \mathbb{F}$ are $n + B$ evaluation points. Let $H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$ and $H_2 : \{0, 1\}^* \rightarrow \mathbb{K}$ be two random oracles.

Commit: On input degree d and a vector $\mathbf{w} \in \mathbb{F}^B$, \mathcal{P} uniformly samples $\mathbf{r} \in \mathbb{F}^{d-B+1}$ 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 $\llbracket \mathbf{w} \rrbracket = \{f(\alpha_i)\}_{i=1}^n$, 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 PRG_i . The share can be computed locally as $f(\alpha_i) \leftarrow \text{PRG}_i.\text{Next}()$.

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

Prove: On input the commitment to circuit input wires and output wires of multiplications $(\llbracket \mathbf{w}_1 \rrbracket, \dots, \llbracket \mathbf{w}_m \rrbracket)$,

1. \mathcal{P} and $(\mathcal{V}_1, \dots, \mathcal{V}_n)$ scan the circuit in topological order. For each batch of multiplication gates (α, β, γ) for which they already have $\llbracket \mathbf{w}_\gamma \rrbracket$, compute $(\llbracket \mathbf{w}_\alpha \rrbracket, \llbracket \mathbf{w}_\beta \rrbracket)$ 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 $\llbracket \mathbf{w} \rrbracket$. For $i \in [n]$, \mathcal{P} samples $r_i \leftarrow \{0, 1\}^\kappa$, sends r_i to \mathcal{V}_i and computes

$$\text{com}_i = H_1(\hat{w}_1^i, \dots, \hat{w}_m^i, r_i).$$

\mathcal{P} broadcasts $(\text{com}_1, \dots, \text{com}_n)$ 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 $\chi := H_2(\text{com}_1, \dots, \text{com}_i)$.

3. Denote $\{\llbracket \mathbf{l}_i \rrbracket, \llbracket \mathbf{r}_i \rrbracket, \llbracket \mathbf{o}_i \rrbracket\}_{i \in [n]}$ to be commitments of all left, right and output wires of batched multiplication gates. Construct $[\tilde{\mathbf{x}}_i] := \chi^{i-1} \cdot [\mathbf{l}_i]$, $[\tilde{\mathbf{y}}_i] := [\mathbf{r}_i]$ for $i \in [n]$ and $[\tilde{\mathbf{z}}] := \sum_{i \in [n]} \chi^{i-1} \cdot [\mathbf{o}_i]$. Invoke $\mathcal{F}_{\text{verifyprod}}$ with input $(\{[\tilde{\mathbf{x}}_i]\}_{i \in [n]}, \{[\tilde{\mathbf{y}}_i]\}_{i \in [n]}, [\tilde{\mathbf{z}}])$ and parties output reject if $\mathcal{F}_{\text{verifyprod}}$ outputs reject.

Fig. 10. The protocol of SIMDZK from designated multi-verifier ZK .

input. Verifiers are assumed honest-majority with adversary threshold $t < n(1/2 - \epsilon)$ for $0 < \epsilon < 1/2$. \mathcal{P} distributes packed Shamir secret shares (PSS) of all batched circuit wire values to $(\mathcal{V}_1, \dots, \mathcal{V}_n)$, who jointly execute a distributed ZK to verify the correctness of the circuit evaluation [12, 15, 29], 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, [53] 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

$[w_1], [w_2]$, the protocol proves that $w_1[i] = w_2[j]$. Overall, the checking procedure incurs communication complexity $\mathcal{O}(n^2 B^2)$. Our compiler reduces it to $\mathcal{O}(n^2)$. In Supplementary Material A.4, we introduce an inner product verification protocol for the check of multiplication gates, and its functionality is denoted as $\mathcal{F}_{\text{verifyprod}}$.

SIMD-ZK from [53]. The protocol is shown in Fig. 10. 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 $([w_\alpha], [w_\beta], [w_\gamma])$ for the indices of all batch multiplication triples (α, β, γ) . 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, \dots, com_n) . In this way, verifiers are able to compute χ by hashing these commitments. In the end, parties convert the multiplication triples to an inner product triple, which is verified by $\mathcal{F}_{\text{verifyprod}}$.

Acknowledgements

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 Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 92270201 and 62125204). Yu Yu's work has also been supported by the New Corner-stone Science Foundation through the XPLOER PRIZE. 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. Work of Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award # 2016240, # 2236819, # 2318975, # 2310927 and research awards from Meta and Google. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the US Government.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- [1] C.J. Alpert, J.H. Huang, A.B. Kahng, Multilevel circuit partitioning, in *Proceedings of the 34th Annual Design Automation Conference* (1997), pp. 530–533
- [2] S. Ames, C. Hazay, Y. Ishai, M. Venkitasubramaniam, Ligerio: Lightweight sublinear arguments without a trusted setup, in B.M. Thuraisingham, D. Evans, T. Malkin, D. Xu (eds.) *ACM CCS 2017* (ACM Press, 2017), pp. 2087–2104. <https://doi.org/10.1145/3133956.3134104>
- [3] T. Attema, R. Cramer, Compressed Σ -protocol theory and practical application to plug & play secure algorithmics, in D. Micciancio, T. Ristenpart (eds.) *CRYPTO 2020, Part III. LNCS*, vol. 12172 (Springer, Heidelberg, 2020), pp. 513–543. https://doi.org/10.1007/978-3-030-56877-1_18
- [4] L. Bangalore, R. Bhadauria, C. Hazay, M. Venkitasubramaniam, On black-box constructions of time and space efficient sublinear arguments from symmetric-key primitives, in E. Kiltz, V. Vaikuntanathan (eds.) *TCC 2022, Part I. LNCS*, vol. 13747. (Springer, Heidelberg, 2022), pp. 417–446. https://doi.org/10.1007/978-3-031-22318-1_15
- [5] C. Baum, A.J. Malozemoff, M.B. Rosen, P. Scholl, Mac’n’cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions, in T. Malkin, C. Peikert (eds.) *CRYPTO 2021, Part IV. LNCS*, vol. 12828 (Springer, Heidelberg, Virtual Event, 2021), pp. 92–122. https://doi.org/10.1007/978-3-030-84259-8_4
- [6] R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, Y. Zhang, Ligerio++: a new optimized sublinear IOP, in J. Ligatti, X. Ou, J. Katz, G. Vigna (eds.) *ACM CCS 2020* (ACM Press, 2020). pp. 2025–2038. <https://doi.org/10.1145/3372297.3417893>
- [7] N. Bitansky, R. Canetti, A. Chiesa, E. Tromer, Recursive composition and bootstrapping for SNARKS and proof-carrying data, in D. Boneh, T. Roughgarden, J. Feigenbaum (eds.) *45th ACM STOC* (ACM Press, 2013), pp. 111–120. <https://doi.org/10.1145/2488608.2488623>
- [8] N. Bitansky, A. Chiesa, Succinct arguments from multi-prover interactive proofs and their efficiency benefits, in R. Safavi-Naini, R. Canetti (eds.) *CRYPTO 2012. LNCS*, vol. 7417 (Springer, Heidelberg, 2012), pp. 255–272. https://doi.org/10.1007/978-3-642-32009-5_16
- [9] Block, A.R., Garman, C.: Honest majority multi-prover interactive arguments. Cryptology ePrint Archive, Report 2022/557 (2022), <https://eprint.iacr.org/2022/557>
- [10] A.R. Block, J. Holmgren, A. Rosen, R.D. Rothblum, P. Soni, Public-coin zero-knowledge arguments with (almost) minimal time and space overheads, in R. Pass, K. Pietrzak (eds.) *TCC 2020, Part II. LNCS*, vol. 12551 (Springer, Heidelberg, 2020), pp. 168–197. https://doi.org/10.1007/978-3-030-64378-2_7
- [11] A.R. Block, J. Holmgren, A. Rosen, R.D. Rothblum, P. Soni, Time- and space-efficient arguments from groups of unknown order, in T. Malkin, C. Peikert (eds.) *CRYPTO 2021, Part IV. LNCS*, vol. 12828 (Springer, Heidelberg, Virtual Event, 2021), pp. 123–152. https://doi.org/10.1007/978-3-030-84259-8_5
- [12] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, Y. Ishai, Zero-knowledge proofs on secret-shared data via fully linear PCPs, in A. Boldyreva, D. Micciancio (eds.) *CRYPTO 2019, Part III. LNCS*, vol. 11694 (Springer, Heidelberg, 2019), pp. 67–97. https://doi.org/10.1007/978-3-030-26954-8_3
- [13] J. Bootle, A. Chiesa, Y. Hu, M. Orrù, Gemini: Elastic SNARKs for diverse environments, in O. Dunkelman, S. Dziembowski (eds.) *EUROCRYPT 2022, Part II. LNCS*, vol. 13276 (Springer, Heidelberg, 2022), pp. 427–457. https://doi.org/10.1007/978-3-031-07085-3_15
- [14] S. Bowe, J. Grigg, D. Hopwood, Halo: recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019). <https://eprint.iacr.org/2019/1021>
- [15] E. Boyle, N. Gilboa, Y. Ishai, A. Nof, Sublinear GMW-style compiler for MPC with preprocessing, in T. Malkin, C. Peikert (eds.) *CRYPTO 2021, Part II. LNCS*, vol. 12826 (Springer, Heidelberg, Virtual Event, 2021), pp. 457–485. https://doi.org/10.1007/978-3-030-84245-1_16
- [16] B. Bünz, A. Chiesa, W. Lin, P. Mishra, N. Spooner, Proof-carrying data without succinct arguments, in T. Malkin, C. Peikert (eds.) *CRYPTO 2021, Part I. LNCS*, vol. 12825 (Springer, Heidelberg, Virtual Event, 2021), pp. 681–710. https://doi.org/10.1007/978-3-030-84242-0_24
- [17] M. Campanelli, A. Faonio, D. Fiore, A. Querol, H. Rodríguez, Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. in M. Tibouchi, H. Wang (eds.) *ASIACRYPT 2021, Part III. LNCS*, vol. 13092 (Springer, Heidelberg, 2021), pp. 3–33. https://doi.org/10.1007/978-3-030-92078-4_1

- [18] M. Campanelli, D. Fiore, A. Querol, LegoSNARK: modular design and composition of succinct zero-knowledge proofs, in L. Cavallaro, J. Kinder, X. Wang, J. Katz (eds.) *ACM CCS 2019* (ACM Press, 2019), pp. 2075–2092. <https://doi.org/10.1145/3319535.3339820>
- [19] B. Chen, B. Bünz, D. Boneh, Z. Zhang, HyperPlonk: plonk with linear-time prover and high-degree custom gates, in *EUROCRYPT 2023, Part II*. LNCS (Springer, Heidelberg, 2023), pp. 499–530. https://doi.org/10.1007/978-3-031-30617-4_17
- [20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, N.P. Ward, Marlin: preprocessing zkSNARKs with universal and updatable SRS, in A. Canteaut, Y. Ishai (eds.) *EUROCRYPT 2020, Part I*. LNCS, vol. 12105 (Springer, Heidelberg, 2020), pp. 738–768. https://doi.org/10.1007/978-3-030-45721-1_26
- [21] I. Damgård, V. Pastro, N.P. Smart, S. Zakarias, Multiparty computation from somewhat homomorphic encryption, in R. Safavi-Naini, R. Canetti (eds.) *CRYPTO 2012*. LNCS, vol. 7417 (Springer, Heidelberg, 2012), pp. 643–662. https://doi.org/10.1007/978-3-642-32009-5_38
- [22] C. Delpech de Saint Guilhem, E. Orsini, T. Tanguy, Limbo: efficient zero-knowledge MPCitH-based arguments, in G. Vigna, E. Shi (eds.) *ACM CCS 2021* (ACM Press, 2021), pp. 3022–3036. <https://doi.org/10.1145/3460120.3484595>
- [23] S. Dittmer, Y. Ishai, R. Ostrovsky, Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446 (2020). <https://eprint.iacr.org/2020/1446>
- [24] N. Ephraim, C. Freitag, I. Komargodski, R. Pass, SPARKs: succinct parallelizable arguments of knowledge, in A. Canteaut, Y. Ishai (eds.) *EUROCRYPT 2020, Part I*. LNCS, vol. 12105 (Springer, Heidelberg, 2020), pp. 707–737. https://doi.org/10.1007/978-3-030-45721-1_25
- [25] M.K. Franklin, M. Yung, Communication complexity of secure computation (extended abstract), in *24th ACM STOC* (ACM Press, 1992), pp. 699–710. <https://doi.org/10.1145/129712.129780>
- [26] A. Gabizon, Z.J. Williamson, O. 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>
- [27] R. Gennaro, C. Gentry, B. Parno, M. Raykova, Quadratic span programs and succinct NIZKs without PCPs, in T. Johansson, P.Q. Nguyen (eds.) *EUROCRYPT 2013*. LNCS, vol. 7881 (Springer, Heidelberg, 2013), pp. 626–645. https://doi.org/10.1007/978-3-642-38348-9_37
- [28] V. Goyal, Y. Song, Malicious security comes free in honest-majority MPC. Cryptology ePrint Archive, Report 2020/134 (2020). <https://eprint.iacr.org/2020/134>
- [29] V. Goyal, Y. Song, C. Zhu, Guaranteed output delivery comes free in honest majority MPC, in D. Micciancio, T. Ristenpart (eds.) *CRYPTO 2020, Part II*. LNCS, vol. 12171 (Springer, Heidelberg, 2020), pp. 618–646. https://doi.org/10.1007/978-3-030-56880-1_22
- [30] J. Groth, Linear algebra with sub-linear zero-knowledge arguments, in S. Halevi (ed.) *CRYPTO 2009*. LNCS, vol. 5677 (Springer, Heidelberg, 2009), pp. 192–208. https://doi.org/10.1007/978-3-642-03356-8_12
- [31] J. Holmgren, R. Rothblum, Delegating computations with (almost) minimal time and space overhead, in M. Thorup (ed.) *59th FOCS* (IEEE Computer Society Press, 2018), pp. 124–135. <https://doi.org/10.1109/FOCS.2018.00021>
- [32] Y. Ishai, E. Kushilevitz, R. Ostrovsky, A. Sahai, Zero-knowledge from secure multiparty computation, in D.S. Johnson, U. Feige (eds.) *39th ACM STOC* (ACM Press, 2007), pp. 21–30. <https://doi.org/10.1145/1250790.1250794>
- [33] S. Kanjalkar, Y. Zhang, S. Gandlur, A. Miller, Publicly auditable mpc-as-a-service with succinct verification and universal setup. CoRR abs/2107.04248 (2021). <https://arxiv.org/abs/2107.04248>
- [34] M. Keller, V. Pastro, D. Rotaru, Overdrive: making SPDZ great again, in J.B. Nielsen, V. Rijmen (eds.) *EUROCRYPT 2018, Part III*. LNCS, vol. 10822 (Springer, Heidelberg, 2018), pp. 158–189. https://doi.org/10.1007/978-3-319-78372-7_6
- [35] A. Kothapalli, S. Setty, SuperNova: proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758 (2022). <https://eprint.iacr.org/2022/1758>
- [36] A. Kothapalli, S. Setty, Hypernova: recursive arguments for customizable constraint systems. Cryptology ePrint Archive (2023)
- [37] A. Kothapalli, S. Setty, I. Tzialla, Nova: recursive zero-knowledge arguments from folding schemes, in Y. Dodis, T. Shrimpton (eds.) *CRYPTO 2022, Part IV*. LNCS, vol. 13510 (Springer, Heidelberg, 2022), pp. 359–388. https://doi.org/10.1007/978-3-031-15985-5_13

- [38] H. Lipmaa, Prover-efficient commit-and-prove zero-knowledge SNARKs, in D. Pointcheval, A. Nitaj, T. Rachidi (eds.) *AFRICACRYPT 16*. LNCS, vol. 9646 (Springer, Heidelberg, 2016), pp. 185–206. https://doi.org/10.1007/978-3-319-31517-1_10
- [39] C. Lund, L. Fortnow, H.J. Karloff, N. Nisan, Algebraic methods for interactive proof systems, in *31st FOCS* (IEEE Computer Society Press, 1990), pp. 2–10. <https://doi.org/10.1109/FSCS.1990.89518>
- [40] A. Ozdemir, D. Boneh, Experimenting with collaborative zk-SNARKs: zero-knowledge proofs for distributed secrets, in K.R.B. Butler, K. Thomas (eds.) *USENIX Security 2022* (USENIX Association, 2022), pp. 4291–4308
- [41] B. Patt-Shamir, A note on efficient aggregate queries in sensor networks, in S. Chaudhuri, S. Kuten (eds.) *23rd ACM PODC* (ACM, 2004), pp. 283–289. <https://doi.org/10.1145/1011767.1011809>
- [42] T.P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in J. Feigenbaum (ed.) *CRYPTO '91*. LNCS, vol. 576 (Springer, Heidelberg, 1992), pp. 129–140. https://doi.org/10.1007/3-540-46766-1_9
- [43] Y. Perl, M. Snir, Circuit partitioning with size and connection constraints. *Networks* **13**(3), 365–375 (1983)
- [44] R. Rohrer, Circuit partitioning simplified. *IEEE Trans. Circuits Syst.* **35**(1), 2–5 (1988)
- [45] B. Schoenmakers, M. Veeningen, N. de Vreede, Trinocchio: privacy-preserving outsourcing by distributed verifiable computation, in M. Manulis, A.R. Sadeghi, S. Schneider (eds.) *ACNS 16*. LNCS, vol. 9696 (Springer, Heidelberg, 2016), pp. 346–366. https://doi.org/10.1007/978-3-319-39555-5_19
- [46] A. Microsoft SEAL (release 4.0). <https://github.com/Microsoft/SEAL> (2022), microsoft Research, Redmond, WA.
- [47] S. Setty, J. Thaler, R. Wahby, Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552 (2023). <https://eprint.iacr.org/2023/552>
- [48] X. Wang, A.J. Malozemoff, J. Katz, EMP-toolkit: efficient MultiParty computation toolkit. <https://github.com/emp-toolkit> (2016)
- [49] C. Weng, K. Yang, J. Katz, X. Wang, Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits, in *2021 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, 2021), pp. 1074–1091. <https://doi.org/10.1109/SP40001.2021.00056>
- [50] C. Weng, K. Yang, Z. Yang, X. Xie, X. Wang, AntMan: interactive zero-knowledge proofs with sublinear communication, in H. Yin, A. Stavrou, C. Cremers, E. Shi (eds.) *ACM CCS 2022* (ACM Press, 2022), pp. 2901–2914. <https://doi.org/10.1145/3548606.3560667>
- [51] H. Wu, W. Zheng, A. Chiesa, R.A. Popa, I. Stoica, DIZK: a distributed zero knowledge proof system, in W. Enck, A.P. Felt (eds.) *USENIX Security 2018* (USENIX Association, 2018), pp. 675–692
- [52] K. Yang, P. Sarkar, C. Weng, X. Wang, QuickSilver: efficient and affordable zero-knowledge proofs for circuits and polynomials over any field, in G. Vigna, E. Shi (eds.) *ACM CCS 2021* (ACM Press, 2021), pp. 2986–3001. <https://doi.org/10.1145/3460120.3484556>
- [53] K. Yang, X. Wang, Non-interactive zero-knowledge proofs to multiple verifiers, in S. Agrawal, D. Lin (eds.) *ASIACRYPT 2022, Part III*. LNCS, vol. 13793 (Springer, Heidelberg, 2022), pp. 517–546. https://doi.org/10.1007/978-3-031-22969-5_18