An Efficient SNARK for Field-Programmable and RAM Circuits

Jehyuk Jang* and Jamie Judd§
Tokamak Network

March 2024

Abstract – The advancement of succinct non-interactive argument of knowledge (SNARK) with constant proof size has significantly enhanced the efficiency and privacy of verifiable computation. Verifiable computation finds applications in distributed computing networks, particularly in scenarios where nodes cannot be generally trusted, such as blockchains. However, fully harnessing the efficiency of SNARK becomes challenging when the computing targets in the network change frequently, as the SNARK verification can involve some untrusted preprocess of the target, which is expected to be reproduced by other nodes. This problem can be addressed with two approaches: One relieves the reproduction overhead by reducing the dimensionality of preprocessing data; The other utilizes *verifiable machine computation*, which eliminates the dependency on preprocess at the cost of increased overhead to SNARK proving and verification. In this paper, we propose a new SNARK with constant proof size applicable to both approaches. The proposed SNARK combines the efficiency of Groth16 protocol, albeit lacking universality for new problems, and the universality of PlonK protocol, albeit with significantly larger preprocessing data dimensions. Consequently, we demonstrate that our proposed SNARK maintains the efficiency and the universality while significantly reducing the dimensionality of preprocessing data. Furthermore, our SNARK can be seamlessly applied to the verifiable machine computation, requiring a proof size smaller about four to ten times than other related works.

1 Introduction

Succinct non-interactive arguments of knowledge (SNARK) are protocols for practical verifiable computation [1] of general programs, translated into statements in NP languages [2]. The protocols include at least two parties including a prover and a verifier. A verifier outsources the execution of a computationally intensive program to a prover and gets back the execution result along with a proof of its correctness. A SNARK verifier should be able to decide the validity of proof in a way more efficient than reproducing the execution result. A SNARK is said zero-knowledge (zk-) SNARK, if a prover cannot obtain any information from a valid proof other than the correctness of computation. For the recent decade, zk-SNARKs with constant-length proofs of which length is independent of the program size have been proposed [3–8].

SNARKs, for instance, can find application in blockchain. A blockchain operates as a chain of signatures for correct computation of user transactions, which transcribe programs with specific input and output. Trustless validators, referred to as full nodes, are randomly selected to sign the correctness of transactions accumulated to date. Blockchain security relies on the number of validations to each transaction, indicating the probability of involving at least one honest validator [9]. While traditional validation entails reproducing the same transaction results, verifiable computation with SNARKs may provide more efficient validation [10–12] of a large accumulation of signatures [13]. Furthermore, zk-SNARKs contributes to preserving privacy [14].

As instantiation of a SNARK with constant proof size, Parno and Gentry in [3] have proposed *Pinocchio*, which is the first practical SNARK for general programs. The efficiency of Pinocchio in terms of the succinctness of verifying a computation comes from the use of an offline compiler that translates a program along with specific input and output into an NP statement, expressed in a deterministic *circuit*. SNARKs after Pinocchio have also

* Corresponding author, cjhyuck213@gmail.com

§ jamie@tokamak.network

inherited the use of such compilers. For example, in [4], Groth has proposed a nearly-optimal SNARK called *Groth16*, whose proof length and proving overhead are the smallest to date.

Security of the constant-length proof SNARKs usually depends on a *trusted setup*. For example, trusted setups in Pinocchio and Groth16 encodes and compresses a circuit into a *common reference string* (CRS) in a cryptographic way. Provers and verifiers are enforced to use a CRS, which ensures they argue a common circuit. Although trust in the setup can be altered by a multi-party computation (MPC) manner [15–17], researchers have pointed out that in this model, for every new circuit, the problem of composing a multi-party and renewed opportunity of adversarial subversion remain unresolved [18]. For example, MPC takes a long time more than a month to organize a multi-party [19–21].

More recently, SNARKs in [5–8] have incorporated *updatable and universal setups* [18], which produce *structured reference strings* (SRS). These setups reduce dependence on trust or MPC, as SRS can be updated by variable participant group to thwart adversarial subversion and are applicable universally across all circuits. Notably, Groth *et al.* in [18] have shown that the setups for CRS, such as those used in Pinocchio and Groth16, are not updatable. Maller *et al*. proposed *Sonic* [5], which stands out as the first practical SNARK featuring updatable and universal setup. *Marlin* [6] and *PlonK* [7], proposed by Chiesa *et al.* and Gabizon *et al.*, respectively, have improved communication and computation efficiency. Lipmaa et. al. [8] have proposed *Vampire*, which further optimized Marlin by adopting a simplified system of constraints referred to as R1CSlite that only has linear constraints to variables. Despite of the reduced reliance on the setup, the security of these works relies on an assumption of *preprocessed verifiers*, where the online verifier is required to preprocess encoding of some or the entire circuit.

Applying the above SNARKs to a blockchain would be crucially tackled, if the blockchain operates on a random-access machine (RAM), such as Ethereum [22]. In RAM, an initial sequence of instructions that defines a program can be modified into *unrolled* instructions by input during execution. Thus, with a CRS-based SNARK, for every transaction, which describes a program with input, issued, a new circuit must be generated, requiring a trusted setup or an MPC for a fresh CRS. Even with an SRS-based SNARK, a new transaction invokes new verifier preprocess, which must be verified or reproduced by other verifiers afterward, as blockchain verifiers are not generally trustworthy[1,2]. Based on this observation, we claim that the more SNARK security relies on verifier preprocess, the higher overhead to communication is imposed for the blockchain security. This problem is not limited to blockchain applications but potentially affects any verifiable computation of RAM through the distributed computing networks.

Works in [24–31] have addressed this problem by utilizing a *universal circuit*, which encompasses a portion or the entirety of computation in a RAM, rather than program-specific circuits. Subsequently, a setup then processes a universal circuit into a CRS that is reusable for general programs, unless there is a change in the machine specification. There exists a trade-off between the portion of RAM computation covered by a universal circuit and the complexity of a SNARK [29]. For instance, to emulate the entire computation of a RAM using a universal circuit, the circuit size must be proportional to the register size of the RAM, affecting both the overhead to the SNARK prover and the communication complexity with the verifier. Conversely, allowing the SNARK verifier to take over a portion of RAM's computation, specifically unrolling instructions, and preprocess the corresponding data, we can maintain the SNARK complexity closely proportional to that of the program-specific circuits.

In this paper, we propose a SNARK that effectively manages the dependency on verifier preprocess. We utilize *field-programmable* circuit derivation: Starting from our universal circuit, defined as a set of subcircuits, a

---

[1] The examples can be found from zk-rollups [23]. In a rollup manner, each transaction in a second layer (as an execution layer) of a blockchain network can be verified in a succinct way based on the trust in each verifier's preprocess, but to upload to the first layer (as a validation layer) a rollup of those for a specific period, the preprocesses must be validated.

[2] Fortunately, CRS or SRS generated by a trusted setup are not the case, as multi-party computation can replace the trust.

program-specific circuit can be derived by placing copies of subcircuits and establishing wiring between them. While our setup does not support updatability, the field-programmable derivation reduces the data dimensionality to be preprocessed by the verifier, focusing primarily on the wiring of subcircuits. As discussed above, we expect that this reduced dependency on the verifier preprocess would address the high communication complexity encountered when implementing verifiable RAM computation in a distributed computing network of untrusted nodes. Additionally, the verifier preprocess can be eliminated without modifying the SNARK itself; rather, by enhancing the complexity of subcircuit designs to handle unrolling instructions.

Our contributions are summarized as follows:

- We demonstrate a method to transform a SNARK with a common reference string, like Groth16, into one with a universal setup by integrating a permutation argument [32, 33, 5, 7]. This integration divides the responsibility of configuring a circuit into two algorithms: the setup and verifier, unlike the previous SNARKs where one algorithm handled both tasks. This integration allows for adjusting the dependency of both parties on maintaining security. An overview of our methodology is provided in subsection 1.2.

- The proposed SNARK is efficient. When used with verifier preprocessing, its communication and computation efficiency is asymptotically comparable to the state-of-the-art universal SNARKs. Moreover, when the verifier preprocessing is eliminated, the communication efficiency of our result surpasses that of other related works. A detailed comparison is provided in subsection 1.1.

- We provide a rigorous security analysis of the proposed SNARK.

The remainder of this paper is organized as follows. Section 2 defines preliminaries. In Section 3, we define a system of constraints, followed by construction of the proposed SNARK in Section 4. In Section 5, we provide security analysis of our SNARK, including adding zero-knowledge. Section 6 illustrates the elimination of verifier preprocess. Section 7 concludes the paper.

1.1 Comparison with related works

For the comparison, we informally use two models of circuits. One represents the size of a circuit in terms of the maximum numbers of addition and multiplication gates and wiring between them, denoted by N_+, $N_\times$, and $N_=$, respectively. In the other model, a circuit is placement and wiring of at most $s_{\max}$ copies of s_D subcircuits, where each subcircuit has at most n_+ addition and $n_\times$ multiplication gates and $n_=$ wires connecting them. Assuming that the subcircuits in the latter model are optimized, we can write $(n_+ + n_\times + n_=)s_{\max} = N_+ + N_\times + N_=$.

As a SNARK with a universal setup. SNARKs with universal setups in common have an intermediate process of deriving a circuit specific to a program and input from URS. We refer to the intermediate outputs as *feature polynomials*, which involve data about the entirety or a part of circuit description. Given the feature polynomials, the SNARKs convince the verifier of the correctness of a circuit evaluation, whereas correctness of the feature polynomials is left to be preprocessed by the verifier. In Figure 1, we summarize a comparison of communication and computation efficiency, including the dimensionality of data that must be preprocessed by the verifier, denoted by $\dim(\text{pre-input})$. Technically, our work is differentiated by the other works in the choice of feature polynomials.

Protocol	Setup			Prove	Verify
	Universality	Updatability	$\lvert$reference string$\rvert$	$\lvert$proof$\rvert$	dim(pre-input)
Groth16 [3]	no	no	$O(m_{\times,+})\ \mathbb{G}$	$3\ \mathbb{G}$	-
Sonic [4]	yes	yes	$O(N_{\times})\ \mathbb{G}$	$20\ \mathbb{G}\ +16\ \mathbb{F}$	$O(N_{\times}N_{+,=})\ \mathbb{F}$
PlonK [6]	yes	yes	$O(N_{\times,+})\ \mathbb{G}$	$9\ \mathbb{G}\ +6\ \mathbb{F}$	$O(N_{\times,+})\ \mathbb{F}$
Marlin [5]	yes	yes	$O(N_{\times,+})\ \mathbb{G}$	$13\ \mathbb{G}\ +8\ \mathbb{F}$	$O(N_{\times,+})\ \mathbb{F}$
This work	yes	no	$O(s_D s_{\max} n_{\times,+})\ \mathbb{G}$	$12\ \mathbb{G}\ +3\ \mathbb{F}$	$O(s_D s_{\max})\ \mathbb{F}$

Protocol	Prove time	Verify time
Groth16 [3]	$O(m_{\times,+})\ \mathbb{G}_{Ex.}\ +\ O(m_{\times}\log m_{\times})\ \mathbb{F}_{Arit.}$	$l\ \mathbb{G}_{Ex.}\ +3\ \mathrm{P}$
Sonic [4]	$O(N_{\times})\ \mathbb{G}_{Ex.}\ +\ O(N_{\times}N_{+,=}\log N_{\times})\ \mathbb{F}_{Arit.}$	$O(l)\ \mathbb{F}_{Arith.}\ +13\ \mathrm{P}$
PlonK [6]	$O(N_{\times,+})\ \mathbb{G}_{Ex.}\ +\ O(N_{\times,+}\log N_{\times,+})\ \mathbb{F}_{Arit.}$	$O(l)\ \mathbb{F}_{Arith.}\ +2\ \mathrm{P}$
Marlin [5]	$O(N_{\times,+})\ \mathbb{G}_{Ex.}\ +\ O(N_{\times,+}\log N_{\times,+})\ \mathbb{F}_{Arit.}$	$O(l+\log N_{\times,+})\ \mathbb{F}_{Arith.}\ +2\ \mathrm{P}$
This work	$O\big((s_D+n_{\times,+})s_{\max}\big)\ \mathbb{G}_{Ex.}\ +\ O\big((s_D+n_{\times})s_{\max}\log s_D s_{\max} n_{\times}\big)\ \mathbb{F}_{Arit.}$	$l\ \mathbb{G}_{Ex.}\ +16\ \mathrm{P}$

Figure 1. Efficiency comparison of SNARKs with universal (and updatable) setups, including Groth16 as a reference state-of-the-art SNARK with a circuit-specific setup. Tables depict the numbers of elements in $\mathbb{G}$ and $\mathbb{F}$, denoted as $\mathbb{G}_1$ or $\mathbb{G}_2$ and $\mathbb{F}$, respectively. $\mathbb{G}_{Ex.}$ and $\mathbb{F}_{Arith.}$ represent the numbers of exponentiations in $\mathbb{G}_1$ or $\mathbb{G}_2$ and arithmetic operations in $\mathbb{F}$, respectively. Parameters $M_{\times}$, M_{+}, and $M_{=}$ for $M \in \{m, N, n\}$ denote the numbers of multiplication constraints, linear constraints, and equal constraints, respectively. Additionally, $M_{\times,+} = M_{\times} + M_{+}$ and $M_{+,=} = M_{+} + M_{=}$. When $M = m$, constraints are counted from an optimized circuit; when $M = N$, it represents the maximum number of constraints a circuit can contain; when $M = n$, it indicates the maximum number of constraints in a subcircuit. In our work, a circuit is placement of at most $s_{\max}$ copies of s_D subcircuits.

In [18], Groth *et al.* proposed a SNARK with updatable and universal setups. The protocol works with a circuit representation referred to as rank 1 constraint system (R1CS), which is of three matrices of size $N_{\times} \times (N_{\times} + N_{+})$ to represent the relationship how each wire contributes to each gate and the wiring. The authors encoded a kernel vector that is orthogonal to all columns of the R1CS matrices into a feature polynomial. This naturally leads that $\dim(\text{pre-input}) = O(N_{\times}(N_{\times} + N_{+}))$. In addition, computing the kernel costs $O(N_{\times}^3)$ field operations.

Maller *et al.* in [5] proposed Sonic, which is known as the first practical SNARK with updatable and universal setups. Sonic is combination of a polynomial commitment scheme referred to as a KZG scheme [34] and a permutation argument [32, 33]. Sonic represents a circuit with three vectors of wires and three matrices of gate configuration. The wire vectors contain values to be assigned to the wires, and therefore an equation of Hadamard product between them specifies nonlinear (multiplication) gates in a circuit. The gate configuration matrices specify linear relationship between the wires, such as addition and wiring. The authors chose to encode the gate configuration matrices into a single feature polynomial, leading that $\dim(\text{pre-input}) = O(N_{\times}(N_{+} + N_{=}))$.

PlonK proposed by Gabizon *et al.* in [7] further optimized the circuit representation of Sonic. PlonK is known as one of the most efficient protocols in terms of communication and computation efficiency. Besides theory, the protocol has been implemented by Iden3 in practice [35] with applying computing acceleration techniques in [36]. Circuits of PlonK, referred to as Plonkish circuits, are restricted to have at most two input wires and an output wire[3]. Gates are configured by five selector vectors, which enable or disable the input and output wires and set the gate operation as either addition or multiplication. What comparable to the circuits of Sonic is that the multiplication between wires is not constrained by default. Also, wiring between gates is described by

[3] This restriction does not compromise generality, as Plonkish circuits can encompass any circuit by increasing the number of gates.

separated three vectors that form a permutation map. The authors encoded the eight vectors into eight feature polynomials, respectively, leading that $\dim(\text{pre-input}) = O(N_\times + N_+)$.

Chiesa *et al.* in [6] proposed Marlin by combining holographic proof system [37] with an optimized KZG scheme. Marlin works with the R1CS circuit representation. The authors utilized a fact that R1CS matrices are sparse, if there are small number of addition gates in a circuit. The sparsity can be preserved even for additionally dense circuits by further splitting constraints. As a result, the sparsity equals the number of wires involved in multiplication and addition gates. The verifier of Marlin encodes the sparse representation of R1CS matrices into nine feature polynomials, leading that $\dim(\text{pre-input}) = O(N_\times + N_+)$.

Our SNARK combines the R1CS and the Plonkish circuit representation. Circuits of our interest are placement of at most $s_{\max}$ copies of predefined s_D subcircuits. Each subcircuit is represented by three R1CS matrices. Wiring between subcircuits is not established by R1CS but by a permutation map. As the subcircuits are predefined and committed by the setup, the permutation is the only feature that uniquely specifies a derived circuit *on-the-fly*. Letting the number of input and output wires in each subcircuit be less than a constant c, the data dimensionality to define a permutation map is $\dim(\text{pre-input}) = O(s_D s_{\max})$. For applications where $n_\times + n_+ \gg s_D$, our SNARK has the reduced $\dim(\text{pre-input})$ compared to that of PlonK or Marlin. This advantage costs that our setup is not updatable, as it outputs a CRS for the R1CS [18]. Though, the subversion of CRS still can be prevented by MPC in weaker sense [15, 16].

As a machine computation eliminating verifier preprocess. In [24–28], various universal circuits for RAM computation have been introduced to eliminate the need for the verifier preprocess. These circuits can be structured into $s_{\max}$ layers, each comprising s_D subcircuits corresponding to instructions of a RAM. Each layer disputes instruction execution at each machine step, including unrolling the next step instruction. Data transfer is restricted to occur only between adjacent layers, ensuring the circuit's layered structure remains independent of RAM programs and input. We refer to such applications of SNARKs as *machine computation*. While the deterministic nature of universal circuits frees the verifier from reproducing feature polynomials, the size of subcircuits remains proportional to the number of slots in the register and memory of a RAM, which is necessary for tracking the machine's internal state changes. This requisite poses implementation challenges, especially for large-scale machines like the Ethereum virtual machine [22].

In the earlier design of the universal circuit in [24], multiplexer (MUX) components were used within each layer to choose a single output from all subcircuit outputs, depending on the input instruction provided to the layer. This design inherently resulted in an asymptotic prover overhead of $O((n_\times + n_+)s_D s_{\max})$. Subsequent works [25, 26] addressed the redundant structure within the universal circuit, which involved replicating identical layers. Instead, the data transfer between adjacent machine steps was argued externally to the circuit, employing arguments such as recursive proof composition [38] or folding schemes [39].

Another simplifications were made in [27, 28] where the universal circuit was presented as a set of s_D subcircuits. The authors proposed a protocol where the prover initially derives a program-specific circuit based on unrolled instructions using a lookup argument [40], followed by disputing the program execution. Specifically, *MUX-Marlin* proposed by Di *et al.* in [27] combined Marlin for program execution verification, a permutation argument for verifying the layered structure in a derived circuit, and a variant of the lookup argument for ensuring the correct copying of subcircuits from the predefined universal circuit. This design can be seen as replacing the role of MUX components with Marlin and the lookup argument compared to the design in [24]. Independently, *SublonK*, proposed by Choudhuri *et al.* in [28], adopted a similar approach but with PlonK instead of Marlin. These works resulted in reducing the size of universal circuits and the asymptotic prover overhead. Figure 2 provides further details.

Protocol	RAM-specific setup	Prove		Verify
	| reference string |	| proof |	Time	Time
MUX-Marlin [27]	$O\left((s_D + s_{\max})n\right)$ $\mathbb{G}$	125 $\mathbb{G}$ + 116 $\mathbb{F}$	$O(s_{\max} n + s_{\max} s_D + s_D n)$ $\mathbb{G}_{Ex.}$ $+ O\left((s_{\max} \log^2 s_{\max} n + s_D \log s_D n)n\right)$ $\mathbb{F}_{Arith.}$	$O(l + \log s_{\max})$ $\mathbb{F}_{Arith.}$ + 2 P
SublonK [28]	$O\left((s_D + s_{\max} \log s_D)n\right)$ $\mathbb{G}$	42 $\mathbb{G}$ + 12 $\mathbb{F}$	$O\left((s_D + s_{\max} \log s_D)n\right)$ $\mathbb{G}_{Ex.}$ $+ O\left((s_D \log s_D n + s_{\max} \log s_{\max} n)n\right)$ $\mathbb{F}_{Arith.}$	$O(l + \log s_{\max})$ $\mathbb{F}_{Arith.}$ + 23 P
This work	$O(s_D s_{\max} n)$ $\mathbb{G}$	12 $\mathbb{G}$ + 3 $\mathbb{F}$	$O((s_D + n)s_{\max})$ $\mathbb{G}_{Ex.}$ $+ O(s_{\max}(s_D \log s_D s_{\max} + n \log s_{\max} n))$ $\mathbb{F}_{Arith.}$	l $\mathbb{G}_{Ex.}$ $+ O(\log s_D)$ $\mathbb{F}_{Arith.}$ + 16 P

Figure 2. Efficiency comparison of machine computation protocols. Tables depict the numbers of elements in $\mathbb{G}$ and $\mathbb{F}$, denoted as $\mathbb{G}_1$ or $\mathbb{G}_2$ and $\mathbb{F}$, respectively. $\mathbb{G}_{Ex.}$ and $\mathbb{F}_{Arith.}$ represent the numbers of exponentiations in $\mathbb{G}_1$ or $\mathbb{G}_2$ and arithmetic operations in $\mathbb{F}$, respectively. Setup builds s_D library subcircuits, each can have at most n constraints. A RAM circuit can consist of at most $s_{\max}$ copies of the library subcircuits.

Our SNARK also supports the universal circuits of MUX-Marlin [27] and SublonK [28]. In terms of argument composition, we also utilize a permutation argument to verify the layered structure in a derived circuit, similar to these works. However, we use Groth16 instead of Marlin or PlonK for program execution verification and introduce an inner-product argument for connecting the other two arguments. As shown in Figure 2, our SNARK achieves remarkable efficiency in proof size.

As a RAM-to-circuit reduction. To alleviate the high prover complexity arising from machine computation of large-scaled RAMs, the works in [29–31] have explored *RAM-to-circuit reduction* [30]. This approach allows the verifier to preprocess feature polynomials containing essential features for unrolling instructions. Consequently, the prover can focus on their program-specific computation rather than the entire machine. Their universal circuits no longer need to represent all RAM states but capture minimal structure of a machine that can be efficiently derived into an unrolled program-specific circuit. However, this reduction necessitates the verifier's knowledge of non-deterministic behavior of a RAM during unrolling program instructions, which includes the entire [29] or the multiplicity [30] of unrolled program instructions, or input and output values at intermediate machine steps [31].

Our SNARK can be seen as a form of RAM-to-circuit reduction, since the feature polynomial requires the knowledge of how instructions are unrolled. *Mirage*, proposed by Kosba *et al.* in [31], shares a similar approach with ours, where a derived circuit is represented by the placement of predefined subcircuits and wiring between them. Thus, the circuit derivation can be uniquely specified by data of the dimensionality $O(s_D s_{\max})$. The key technical distinction lies in the verification of the wiring between subcircuits: In Mirage, it is verified by a predefined permutation subcircuit, whereas in our approach, it is done by the permutation and inner-product arguments, externally to the circuit. As a result, Mirage's verifier requires preprocessed data containing the input and output values to each subcircuit, while our approach does not. This feature gives our SNARK the option to eliminate the verifier preprocess through machine computation, depending on the application.

1.2 Technical overview

A universal circuit we consider is defined as a library $\mathcal{L}$ of s_D subcircuits. A circuit can be derived from the universal circuit as placement of at most $s_{\max}$ copies of the subcircuits and specified by a wire map that describes wiring between the copies. Figure 3 and Figure 4 illustrate the derivation. Analogously, a system of constraints for a circuit derivation can be split into two subsystems, one argues arithmetic constraints inside each subcircuit copy, and the other argues copy constraints that the wires connecting two or more subcircuit copies must share the same values. We represent variables of our constraint system as a triple of vectors $(\mathbf{a}, \mathbf{b}, \mathbf{c})$, where $\mathbf{a}$ is public instance to a derived circuit, $\mathbf{b}$ is private and of the values shared by the connecting wires, and $\mathbf{c}$ is of the values assigned to the internal wires inside each subcircuit copy. The arithmetic constraints check whether all the variables $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ satisfy the constraints of all subcircuit copies placed in the circuit. The copy constraints checks whether $\mathbf{b}$ satisfies a permutation, which is defined by the wire map.

Our SNARK shown in Figure 5 consists of three arguments: an arithmetic argument based on Groth16 [4] for the arithmetic constraints, a permutation argument based on [32, 33, 5, 7] for the copy constraints, and an inner-product argument that connects the two arguments. More specifically, the inner-product argument convinces a verifier that proofs of the other two arguments are generated from the same witness $\mathbf{b}$.

The arithmetic argument is a modification of Groth16. Originally, Groth16 worked with a univariate polynomial $p_A(X)$ and a finite range $\mathcal{X}$ such that $p_A(x_i)=0$ for $x_i \in \mathcal{X}$, if and only if the i-th constraint in a circuit is satisfied. On the contrary, as we consider $s_{\max}$ copies of subcircuits, we extend the protocol to work with a bivariate polynomial $p_A(X,Y)$ and two finite ranges $\mathcal{X}$ and $\mathcal{Y}$, where $\mathcal{X}$ indicates the constraint indices, and $\mathcal{Y}$ indicates the copy indices. In other words, $p_A(x_h, y_i)=0$ for $(x_h, y_i) \in \mathcal{X}\times\mathcal{Y}$, if and only if the h-th constraint of the i-th subcircuit copy in a derived circuit is satisfied.

The permutation argument is a modification of that used in PlonK [7]. The original argument worked with a univariate permutation polynomial $p_C(X)$ and a finite range $\mathcal{X}$ such that $p_C(x_i)=x_k$ for $x_i, x_k \in \mathcal{X}$ holds if and only if the two values to be assigned respectively to the i-th wire and the k-th wire in a circuit are the same. On the contrary, we introduce a bivariate permutation polynomial $p_C(Y,Z)$ over finite ranges $\mathcal{Y}$ and $\mathcal{Z}$, where $\mathcal{Y}$ indicates the copy indices, and $\mathcal{Z}$ indicates the wire indices. For example, $p_C(y_h, z_i)=y_j\theta+z_k$ for $(y_h, z_i),(y_j, z_k)\in\mathcal{Y}\times\mathcal{Z}$ holds true for any θ, if and only if the two values to be assigned respectively to the i-th wire of the h-th subcircuit copy and the k-th wire of the j-th subcircuit copy must be the same.

As the last argument, the inner-product argument encloses the two other arguments. Let $b_i(Y)$ and $b_i'(Y)$ denote polynomial encodings of the witness $\mathbf{b}$ and $\mathbf{b}'$ to the arithmetic and permutation arguments over $\mathcal{Y}$. The arithmetic argument produces a proof polynomial involving $\sum_i b_i(Y)o_i(X)$, which is a part of $p_A(X,Y)$, where basis polynomials $o_i(X)$ represent the feature of wires (they are not orthonormal to each other on $\mathcal{X}$ in general). The permutation argument produces a proof polynomial involving $\sum_i b_i'(Y)K_i(Z)$, which will be combined with $p_C(Y,Z)$, where $K_i(Z)$ are Lagrange bases over $\mathcal{Z}$ and therefore orthonormal to each other on $\mathcal{Z}$. The goal of inner-product argument is to identify $b_i(Y)=b_i'(Y)$, and to achieve this we make use of the orthonormality of $K_i(Z)$. Taking an honestly computed polynomial $\sum_i o_i(X)K_i(Z)$ as a precomputable input, the verifier of inner-product argument accepts the proof polynomials of the other arguments only if

$$\left(\sum_i b_i'(Y)K_i(Z)\right)\left(\sum_i o_i(X)K_i(Z)\right)-\sum_i b_i(Y)o_i(X)\equiv 0 \mod t_{\mathcal{Z}}(Z),$$

where $t_{\mathcal{Z}}(Z)$ is a polynomial that vanishes on $\mathcal{Z}$.

The remaining aspect involves configuring the setup to efficiently compress the precomputable polynomials forming the constraint system, while maintaining universality. Specifically, the setup output, a CRS, must remain reusable for new circuits and instances $\mathbf{a}$. To achieve this, the setup publishes commitments to randomized monomials and polynomials, which consistently include the wire polynomials of each subcircuit in the library to form $p_A(X,Y)$. However, including a permutation polynomial $p_C(Y,Z)$ into the CRS could be done cautiously. Since wire maps are typically circuit-specific, including it in the CRS might compromise the universality of the setup. Nevertheless, for specific applications such as machine computation, which have fixed wiring structures, the permutation can also be committed by the setup (will be further discussed in Section 6).

2 Preliminaries

2.1 Notations

All sets throughout this paper are regarded as multisets unless otherwise specified. We write $\{x_{i_1,i_2,\cdots,i_n}\}_{i_1=m_1,i_2=m_2,\cdots,i_n=m_n}^{l_1,l_2,\cdots,l_n}$ as a set-builder notation for indexed elements $x_{i_1,i_2,\cdots,i_n}$ for all

$(i_1, i_2, \cdots, i_n) \in \prod_{k=1}^{n}\{m_k, \cdots, l_k\}$. Similarly, given an algebraic structure $\mathbb{A}$, we write $\mathbf{x} = (x_{i_1,i_2,\cdots,i_n})_{i_1=m_1,i_2=m_2,\cdots,i_n=m_n}^{l_1,l_2,\cdots,l_n}$ to denote a vector or a tuple of the indexed elements $x_{i_1,i_2,\cdots,i_n} \in \mathbb{A}$ in arbitrary order. Given a field $\mathbb{F}$, we write $\mathbb{F}_{d_1,d_2,\cdots,d_n}[X_1, X_2, \cdots, X_n]$ as a set of polynomials with degree less than d_i respectively in X_i for all $i \in \{1, \cdots, n\}$ and with total degree less than $\sum_{i=1}^{n} d_i$. We write $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ as a concatenation of two vectors $\mathbf{x}$ and $\mathbf{y}$.

2.2 Cryptographic definitions

Let $pp_\lambda = (\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ be a bilinear group generated from a security parameter $\lambda \in \mathbb{N}$. $\mathbb{G}_1, \mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a multiplicative group defined over a field $\mathbb{F}$. Letting $G_i \in \mathbb{G}_i$ for $i \in \{1,2\}$ denote the generators, we write group encodings $[\mathbf{x}]_i := (x_0 G_i, \cdots, x_{n-1} G_i)$ for $\mathbf{x} \in \mathbb{F}^n$. We define $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as a non-degenerative bilinear map that holds $e([x]_1, [y]_2) = e([1]_1, [1]_2)^{xy}$ for $x, y \in \mathbb{F}$. It is deduced that $e([1]_1, [1]_2)$ is the generator of $\mathbb{G}_T$. We say a function $\epsilon : \mathbb{N} \to [0,1]$ is *negligible* in λ, shortly $negl(\lambda)$, if there exists a constant c such that for all λ, $\epsilon(\lambda) < \lambda^{-c}$.

We consider generic group model (GGM) with affine prover strategy, where generic polynomial-time adversaries $\mathcal{A}$ have no direct access to the group operations in pp_λ. General group model has been defined in [41] by a random injective encoding $[\cdot]_i$ from a field $\mathbb{F}$ to a group $\mathbb{G}_i$ for $i \in \{1,2\}$. As $\mathcal{A}$ has no access to the randomness of $[\cdot]_i$, group operations can be handled only through an oracle. This implies that every $\mathbf{y} \in \mathbb{G}_i^k$ produced by $\mathcal{A}$ there is an affine strategy $\mathbf{P} \in \mathbb{F}^{k \times l}$ such that $\mathbf{y} = \mathbf{P}\mathbf{x}$ for given random encoding $\mathbf{x} \in \mathbb{G}_i^l$.

2.3 Useful lemmas about polynomials

Lemma 1 (Schwartz-Zippel (SZ) Lemma). *Let $p \in \mathbb{F}[X_1, \cdots, X_n]$ be a non-zero n-variate polynomial with total degree not greater than d. Let $\mathcal{X} \subseteq \mathbb{F}$ and $(x_1, \cdots, x_n)$ be picked at random independently and uniformly from $\mathcal{X}^n$. Then,* $\Pr[f(x_1, \cdots, x_n) = 0] \le d\,|\mathcal{X}|^{-1}$.

Lemma 2 (Ben-Sasson and Sudan [42]). *Let $\mathcal{X} \subseteq \mathbb{F}$ with $|\mathcal{X}| = n$. A polynomial $p \in \mathbb{F}_N[X]$ with degree $N-1 \ge n$ vanishes on $\mathcal{X}$ if and only if the vanishing polynomial $t_{\mathcal{X}}(X) := \prod_{x \in \mathcal{X}}(X - x)$ divides p, i.e., if and only if there exists a quotient polynomial $h \in \mathbb{F}[X]$ (i.e., $h(X)$ does not involve negative powers of X) such that $p(X) = t_{\mathcal{X}}(X) h(X)$.*

Proof. See Appendix A.

Corollary 1 (An extension of Lemma 2 to bivariate polynomials). *Let $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{F}$ with $|\mathcal{X}| = n$ and $|\mathcal{Y}| = s$. A polynomial $p \in \mathbb{F}_{N,S}[X,Y]$ with degree $N \ge n$ in X and $S \ge s$ in Y vanishes on $\mathcal{X} \times \mathcal{Y}$ if and only if, given two vanishing polynomials $t_{\mathcal{X}}(X) := \prod_{x \in \mathcal{X}}(X - x)$ and $t_{\mathcal{Y}}(Y) := \prod_{y \in \mathcal{Y}}(Y - y)$, there exist quotient polynomials $h_{\mathcal{X}}, h_{\mathcal{Y}} \in \mathbb{F}[X,Y]$ such that $p(X,Y) = t_{\mathcal{X}}(X) h_{\mathcal{X}}(X,Y) + t_{\mathcal{Y}}(Y) h_{\mathcal{Y}}(X,Y)$.*

Proof. See Appendix B.

2.4 Circuit, rank-1 constraint system, and quadratic arithmetic program

An arithmetic circuit comprises multiplication gates, addition gates, and wires. Each gate has two input wires and one output wire, with some gates possibly have wiring connections. For a given list of values assigned to the wires, the circuit is considered satisfied if these values satisfy to the input-output relationships of all gates.

A rank-1 constraint system (R1CS) is a matrix representation of a circuit. R1CS is a set of matrices $\mathbf{U},\mathbf{V},\mathbf{W} \in \mathbb{F}^{n\times m}$, where the triple of column vectors $(\mathbf{u}_k = (u_{i,k})_{i=0}^{n-1}, \mathbf{v}_k = (v_{i,k})_{i=0}^{n-1}, \mathbf{w}_k = (w_{i,k})_{i=0}^{n-1})$ for each $k \in \{0,\cdots,m-1\}$ represents a wire. Given assignments to the wires, denoted by a vector $\mathbf{d} = (d_0,\cdots,d_{m-1})$, a circuit is satisfied if and only if the following n equations for $i=0,\cdots,n$ holds simultaneously,

$$\left(\sum_{k=0}^{m-1} d_k u_{i,k}\right)\cdot\left(\sum_{k=0}^{m-1} d_k v_{i,k}\right)-\left(\sum_{k=0}^{m-1} d_k w_{i,k}\right)=0. \tag{1}$$

Each equation is referred to as a *constraint*, which represents the relationship between a left input, a right input, and an output within a multiplication gate. In R1CS, addition gates are typically not considered independent constraints but instead merged into the n multiplication gates as summations of the inputs or outputs, except for special uses such as Marlin [6], where n is expanded as increase of the addition gates to preserve sparsity of the R1CS matrices. It is also notable that R1CS does not count equalities between the wires as constraints, unlike constraint systems of PlonK [7] or Sonic [5].

Quadratic arithmetic program (QAP) is a polynomial representation of rank-1 constraint systems (R1CS) [43]. We first define a vanishing set $\mathcal{X} \subseteq \mathbb{F}$ of $|\mathcal{X}| = n$. It is preferred to pick $\mathcal{X}$ as a group of order n generated by an n-th root of unity ω_X (i.e., $\omega_X^n = 1$) for computation efficiency. A vanishing polynomial is then defined as $t_{\mathcal{X}}(X) := \prod_{x\in\mathcal{X}}(X-x) = \omega_X^n - 1$. A QAP is defined as a set of polynomials, $\mathcal{Q} \subseteq \mathbb{F}_n[X]$ such that

$$\mathcal{Q} := \left\{\left(u_k(X), v_k(X), w_k(X)\right)\right\}_{k=0}^{m-1},$$

where $u_k(X)$, $v_k(X)$, and $w_k(X)$ encodes the column vectors $\mathbf{u}_k$, $\mathbf{v}_k$, $\mathbf{w}_k$ over $\mathcal{X}$, respectively. In other words, $u_k(\omega_X^i) = u_{k,i}$, $v_k(\omega_X^i) = v_{k,i}$, $w_k(\omega_X^i) = w_{k,i}$ for $i = 0,\cdots,n-1$ and $k = 0,\cdots,m-1$. The left-hand side of equation (1) can be equivalently represented a circuit polynomial $p(X)$ as defined by

$$p(X) := \left(\sum_{k=0}^{m-1} d_k u_k(X)\right)\cdot\left(\sum_{k=0}^{m-1} d_k v_k(X)\right)-\left(\sum_{k=0}^{m-1} d_k w_k(X)\right).$$

By Lemma 2, a circuit is satisfied if and only if there exists $q \in \mathbb{F}_{n-1}[X]$ such that

$$p(X) = q(X)t_{\mathcal{X}}(X). \tag{2}$$

QAP is useful to define NP language statements. A statement can be a tuple $(\mathbf{a},\mathbf{c})$ of instance and witness, where the instance $\mathbf{a} = (d_0,\cdots,d_{l-1})$ would incorporate public input and output of a circuit, and the witness $\mathbf{c} = (d_l,\cdots,d_{m-1})$ would involve the intermediate outputs of the gates. We specify a relation generator $\mathcal{R}$ for a security parameter λ that outputs a polynomial-time decidable binary relation R_λ, which is a set of the instance and witness tuples that satisfy a given QAP. For notational simplicity, we elide λ from R_λ and denote it by R. Formally, we can define

$$R := \left\{(\mathbf{a},\mathbf{c})\middle|\begin{array}{l}\mathbf{a} = \left(d_0,\cdots,d_{l-1}\right) \in \mathbb{F}^l, \mathbf{c} = \left(d_l,\cdots,d_{m-1}\right) \in \mathbb{F}^{m-l}, \\ \exists q \in \mathbb{F}_{n-1}[X]: p(X) = q(X)t_{\mathcal{X}}(X)\end{array}\right\}.$$

2.5 Groth16: Non-interactive linear proof system for QAP-based R

Groth16, proposed by J. Groth [4], is a non-interactive system to argue a statement in R, known as the most succinct protocol under generic group model. The system is a quadruple of polynomial-time algorithms $(\texttt{Setup},\texttt{Prove},\texttt{Verify},\texttt{Sim})$. In detail, given a relation R based on a QAP $\mathcal{Q}$ with parameters l,m,n, each algorithm is defined as follows:

- $\texttt{Setup}:(pp_\lambda,\mathcal{Q})\mapsto(\boldsymbol{\tau},\boldsymbol{\sigma})$ is a probabilistic polynomial-time (PPT) algorithm that takes as input the bilinear group pp_λ and a QAP $\mathcal{Q}$ and outputs a simulation trapdoor $\boldsymbol{\tau}\in\mathbb{F}^5$ and a common reference string (CRS) $\boldsymbol{\sigma}\in\mathbb{G}_1^{m+2n+3}\times\mathbb{G}_2^{n+5}$,
- $\texttt{Prove}:(\mathcal{Q},\boldsymbol{\sigma},\mathbf{a},\mathbf{c})\mapsto\boldsymbol{\pi}$ is a deterministic polynomial-time (DPT) algorithm that takes as input statement $\mathbf{a}\in\mathbb{F}^l$ and witness $\mathbf{c}\in\mathbb{F}^{m-l}$ and outputs a proof $\boldsymbol{\pi}\in\mathbb{G}_1{}^2\times\mathbb{G}_2$,
- $\texttt{Verify}:(\boldsymbol{\sigma},\mathbf{a},\boldsymbol{\pi})\mapsto 0/1$ is a DPT algorithm that takes as input a proof $\boldsymbol{\pi}$ and returns 0 (reject) or 1 (accept),
- $\texttt{Sim}:(\mathcal{Q},\boldsymbol{\tau},\mathbf{a})\mapsto\boldsymbol{\pi}^\star$ is a PPT algorithm that outputs a simulated proof $\boldsymbol{\pi}^\star$.

Definition 1 (Perfect completeness). *A proof system* $(\texttt{Setup},\texttt{Prove},\texttt{Verify})$ *for* R *is perfect complete, if, for all* $(\mathbf{a},\mathbf{c})\in R$,

$$\Pr\left[(\boldsymbol{\sigma},\boldsymbol{\tau})\leftarrow\texttt{Setup}(pp_\lambda,\mathcal{Q});\boldsymbol{\pi}\leftarrow\texttt{Prove}(\mathcal{Q},\boldsymbol{\sigma},\mathbf{a},\mathbf{c})\quad:\texttt{Verify}(\boldsymbol{\sigma},\mathbf{a},\boldsymbol{\pi})=1\right]=1.$$

Definition 2 (Statistical knowledge soundness). *A proof system* $(\texttt{Setup},\texttt{Prove},\texttt{Verify})$ *for* R *is statistical knowledge sound, if for all polynomial-time generic adversaries* $\mathcal{A}$, *there exists a polynomial-time extractor* $\mathcal{X}_\mathcal{A}$ *such that*

$$\Pr\left[\begin{array}{c}(\boldsymbol{\sigma},\boldsymbol{\tau})\leftarrow\texttt{Setup}(pp_\lambda,\mathcal{Q});\\(\mathbf{a},\boldsymbol{\pi})\leftarrow\mathcal{A}(\mathcal{Q},\boldsymbol{\sigma});\mathbf{c}\leftarrow\mathcal{X}_\mathcal{A}(R,\boldsymbol{\sigma},\mathbf{a},\boldsymbol{\pi})\end{array}:\begin{array}{c}\texttt{Verify}(\boldsymbol{\sigma},\mathbf{a},\boldsymbol{\pi})=1\wedge\\(\mathbf{a},\mathbf{c})\notin R\end{array}\right]=negl(\lambda).$$

We introduce the Groth16 algorithms that satisfy the above definitions.

$\texttt{Setup}(pp_\lambda,\mathcal{Q})$ picks $\boldsymbol{\tau}=(x,\alpha,\beta,\gamma,\delta)$ uniformly from $(\mathbb{F}^*)^5$ at random, defines

$$o_i(X):=\beta u_i(X)+\alpha v_i(X)+w_i(X),$$

computes

$$\begin{aligned}\boldsymbol{\sigma}_1&=\left(\alpha,\beta,\delta,\left\{x^h\right\}_{h=0}^{n-1},\left\{\gamma^{-1}o_i(x)\right\}_{i=0}^{l-1},\left\{\delta^{-1}o_i(x)\right\}_{i=l}^{m-1},\left\{\delta^{-1}x^h t_\mathcal{X}(x)\right\}_{h=0}^{n-2}\right),\\\boldsymbol{\sigma}_2&=\left(\beta,\gamma,\delta,\left\{x^h\right\}_{h=0}^{n-1}\right),\end{aligned}$$

and returns $(\boldsymbol{\tau},\boldsymbol{\sigma})$, where $\boldsymbol{\sigma}=([\boldsymbol{\sigma}_1]_1,[\boldsymbol{\sigma}_2]_2)$.

$\texttt{Prove}(\mathcal{Q},\boldsymbol{\sigma},\mathbf{a},\mathbf{c})$ parses $\mathbf{a}=(d_0,\cdots,d_{l-1})$ and $\mathbf{c}=(d_l,\cdots,d_{m-1})$, computes $q\in\mathbb{F}_{n-1}[X]$ defined in (2), computes

$$
\begin{aligned}
[A]_1 &= [\alpha]_1 + \left[\sum_{i=0}^{m-1} d_i u_i(x)\right]_1, \\
[B]_k &= [\beta]_k + \left[\sum_{i=0}^{m-1} d_i v_i(x)\right]_k \text{ for } k \in \{0,1\}, \\
[C]_1 &= \sum_{i=l}^{m-1} d_i \left[\delta^{-1} o_i(x)\right]_1 + \left[\delta^{-1} q(x) t_{\mathcal{X}}(x)\right]_1,
\end{aligned}
\tag{3}
$$

and returns $\boldsymbol{\pi} = ([A]_1, [B]_2, [C]_1)$.

$\texttt{Verify}(\boldsymbol{\sigma}, \mathbf{a}, \boldsymbol{\pi})$ parses $\mathbf{a} = (d_0, \cdots, d_{l-1})$, $\boldsymbol{\pi} = ([A]_1, [B]_2, [C]_1)$ and returns $\texttt{accept}$ if and only if the following equation holds,

$$
e\left([A]_1, [B]_2\right) = e\left([\alpha]_1, [\beta]_2\right) e\left(\sum_{i=0}^{l-1} d_i \left[\gamma^{-1} o_i(x)\right]_1, [\gamma]_2\right) e\left([C]_1, [\delta]_2\right).
$$

Theorem 1 (Groth [4]). *The Groth16 system* $(\texttt{Setup}, \texttt{Prove}, \texttt{Verify})$ *for* R *constructed in Section 2.5 is perfect complete and statistical knowledge soundness in generic group model.*

We omit the proof of Theorem 1, as further discussion on completeness and knowledge soundness will be provided in Section 5.

The above construction of $\texttt{Prove}$ as a DPT algorithm can be converted into a PPT algorithm $\texttt{Prove}^\star$ to have one additional security property, zero-knowledge as defined below.

Definition 3 (Perfect zero-knowledge). A proof system $(\texttt{Setup}, \texttt{Prove}^\star, \texttt{Verify}, \texttt{Sim})$ for R has perfect zero-knowledge if for all $(\mathbf{a}, \mathbf{c}) \in R$ and all adversaries $\mathcal{A}$

$$
\begin{aligned}
&\Pr\left[(\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \texttt{Setup}(pp_\lambda, \mathcal{Q}); \boldsymbol{\pi} \leftarrow \texttt{Prove}(\mathcal{Q}, \boldsymbol{\sigma}, \mathbf{a}, \mathbf{c}) \quad : \mathcal{A}(R, \boldsymbol{\sigma}, \boldsymbol{\tau}, \mathbf{a}, \boldsymbol{\pi}) = 1\right] = \\
&\Pr\left[(\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \texttt{Setup}(pp_\lambda, \mathcal{Q}); \boldsymbol{\pi} \leftarrow \texttt{Sim}(\mathcal{Q}, \boldsymbol{\tau}, \mathbf{a}) \quad : \mathcal{A}(R, \boldsymbol{\sigma}, \boldsymbol{\tau}, \mathbf{a}, \boldsymbol{\pi}) = 1\right].
\end{aligned}
$$

The perfect zero-knowledge means that the proof generated from a valid statement $(\mathbf{a}, \mathbf{c}) \in R$ is probabilistically indistinguishable from a simulated proof $\boldsymbol{\pi}^\star$. For the Groth16 system to have the perfect zero-knowledge, a simulation algorithm has been constructed as follow.

$\texttt{Sim}(\mathcal{Q}, \boldsymbol{\tau}, \mathbf{a})$ parses $\mathbf{a} = (d_0, \cdots, d_{l-1})$, picks $(A^\star, B^\star)$ uniformly from $(\mathbb{F}^*)^2$ at random, computes

$$
[C^\star]_1 = \left[\delta^{-1}\left(A^\star B^\star - \alpha\beta - \sum_{i=0}^{l-1} d_i \gamma^{-1} o_i(x)\right)\right]_1,
$$

and returns $\boldsymbol{\pi}^\star = ([A^\star]_1, [B^\star]_2, [C^\star]_1)$.

It is straightforward to see that the simulated proof $\boldsymbol{\pi}^\star$ is always accepted by $\texttt{Verify}$. All that remains is to construct a PPT algorithm $\texttt{Prove}^\star$ that outputs a proof $\boldsymbol{\pi}$ with the identical probability distribution as the simulated proof $\boldsymbol{\pi}^\star$:

$\texttt{Prove}^\star(\mathcal{Q}, \boldsymbol{\sigma}, \mathbf{a}, \mathbf{c})$ is a modification of $\texttt{Prove}$, where it additionally picks (r, s) uniformly from $(\mathbb{F}^*)^2$ at random and modifies the computation (3) as $[\tilde{A}]_1 = [A]_1 + r[\delta]_1$, $[\tilde{B}]_2 = [B]_2 + s[\delta]_2$, $[\tilde{C}]_1 = [C]_1 + s[A]_1 + r[B]_1$

$+rs[\delta]_1$. The output is $\boldsymbol{\pi} = ([\tilde{A}]_1, [\tilde{B}]_2, [\tilde{C}]_1)$.

Theorem 2 (Groth [4]). *The Groth16 system* $(\mathtt{Setup}, \mathtt{Prove}^{\star}, \mathtt{Verify}, \mathtt{Sim})$ *for* R *constructed in Section 2.5 has perfect zero-knowledge.*

Circuit-specific setup of the Groth16 system. The setup algorithm allows the prover to compress a statement into three group elements. However, the security relies on the assumption that any intermediate output, such as the trapdoor $\boldsymbol{\tau}$ or the randomized monomials $\{[\delta^{-1}x^h]_1, [\gamma^{-1}x^h]_1\}_h$, other than the final output $\boldsymbol{\sigma}$, is kept secret. Some real-world applications can realize this assumption based on trust. In this case, since the setup algorithm requires a QAP as input, each new circuit requires new trust.

2.6 KZG polynomial commitment scheme and witness-extended emulation

To reduce dependence on the trusted setup of succinct proof systems like Groth16, interactive public-coin protocols for R such as Sonic [5], PlonK [7], and Marlin [6] have introduced universal setups that are independent of specific circuits. Instead, these protocols have incorporated polynomial commitment schemes as sub-protocols to ensure that the parties—the prover and verifier—agree on the same circuit or polynomials before commencing larger protocols. Specifically, in a larger protocol, the verifier is convinced of the prover's knowledge given circuit polynomials that were previously committed to.

KZG polynomial commitment scheme, proposed by Kate *et al.* in [34], is an efficient polynomial scheme that utilizes the following algebraic behavior: Let $p \in \mathbb{F}_n[X]$ be a polynomial to be committed. By Lemma 2, for every $\xi \in \mathbb{F}$, there exists a pair of a quotient polynomial $q \in \mathbb{F}[X]$ and the evaluation $p(\xi)$ such that $p(X) - p(\xi) = q(X)(X - \xi)$. Suppose there is a structured reference string (SRS) $\boldsymbol{\sigma} = ([(x^i)_{i=0}^{n-1}]_1, [1]_2, [x]_2)$ with a random trapdoor $x \in \mathbb{F}^*$. We let $[f(x)]_1$ denote a commitment to a polynomial $f \in \mathbb{F}_n[X]$. The protocol proceeds as follows: *1)* prover commits to $p(X)$; *2)* verifier picks ξ at random from $\mathbb{F}$; *3)* prover evaluates $p(\xi)$ and commits to $q(X)$ that satisfies $p(X) - p(\xi) = q(X)(X - \xi)$; *4)* verifier accepts the transcript $\mathbf{t} = ([p(x)]_1, \xi, p(\xi), [q(x)]_1)$, if and only if

$$e\big([p(x)]_1 - p(\xi)[1]_1, [1]_2\big) = e\big([q(x)]_1, [x]_2 - \xi[1]_2\big).$$

As a sub-protocol, an advantage of the KZG commitment scheme is that a larger protocol can manipulate the evaluation $p(\xi)$ in place of the commitment $[p(x)]_1$, which is referred to as *evaluation binding*. Manipulating the additive group elements is costly and inefficient, and is only allowed with limited operations, such as linear operations or bilinear pairings. Technically, the verifier can be convinced of the prover's knowledge of the coefficients of $p(x)$, once an acceptable evaluation $p(\xi)$ is provided, through *witness-extended emulation* [44] (refer to Definition 5 for a formal definition) with high probability in λ . Below, we outline how the extraction works.

Consider a PPT adversary $\mathcal{A}$. Suppose that a transcript $\mathbf{t} = ([a]_1, \xi, b, [c]_1)$, generated between $\mathcal{A}$ and the verifier, is acceptable. In generic group model, the verification equation can be rewritten into $a - b = c(x - \xi)$, where the transcript components can be rewritten into $a = \sum_{i=0}^{n-1} a_i x^i$ and $c = \sum_{i=0}^{n-1} c_i x^i$. The SZ lemma allows to rewrite the verification equation into a polynomial equation $a(X) - b = c(X)(X - \xi)$, with high probability greater than $1 - n|\mathbb{F}|^{-1}$. Suppose this case has happened. Then, by Lemma 2, we can imply that $a(\xi_i) = \sum_{i=0}^{n-1} a_i \xi^i = b$. Suppose there exists a DPT emulator that accesses all the internal states of $\mathcal{A}$, including coin-tossing, used to generate $\mathbf{t}$. This emulator thus can interact with the verifier by mimicking the behavior of $\mathcal{A}$. By emulating the protocol to collect n acceptable transcripts with fresh (distinct) randomness $\xi_1, \cdots \xi_n$, the verifier can obtain n simultaneous linear equations $a(\xi_i) = b$, where the linear map is a Vandermonde matrix, so it is invertible. As a result, the coefficients $(a_0,\cdots,a_{n-1})$ can be extractable from the emulated transcripts, with probability greater than $1-n^2\,|\,\mathbb{F}\,|$. Lindel in [44] has shown that the emulation can be done expectedly in n runs.

2.7 PlonK's permutation argument

We revisit a permutation argument based on the KZG commitment scheme, refined by Gabizon *et al.* in **[7]**. Recall a circuit represented by a constraint system R. Suppose there are wiring constraints in the circuit that necessitate repetitions in some of the wire assignments within a statement $\mathbf{d}=(d_0,\cdots,d_{m-1})$. The wiring constraints, referred to as *copy constraints* throughout the rest of this paper, indicate that two wires attached to different gates share the same assignment. As an interactive public-coin protocol, the objective of a permutation argument is to convince the verifier of the prover's knowledge of $\mathbf{d}$ that satisfies the copy constraints.

Define a vanishing set $\mathcal{Z}:=\{\omega_Z^i\}_{i=0}^{m-1}$ of the m-th root of unity and the corresponding vanishing polynomial $t_{\mathcal{Z}}(Z)$. Formally, copy constraints in a statement $\mathbf{d}=(d_0,\cdots,d_{m-1})$ can be defined using a permutation polynomial $s:\mathcal{Z}\to\mathcal{Z}$ such that $d_i=d_k$, if $s(\omega_Z^i)=\omega_Z^k$. Encoding the wire assignments into a polynomial $b:\mathcal{Z}\to\mathbb{F}$ such that $b(\omega_Z^i)=d_i$, the following claim can be used to equivalently check the copy constraints $s(Z)$ with respect to $b(Z)$.

Claim. *Denoting indeterminates $\boldsymbol{\theta}=(\theta_0,\theta_1)$, let $f(Z,\boldsymbol{\theta}):=b(Z)+\theta_0 s(Z)+\theta_1$ and $g(Z,\boldsymbol{\theta}):=b(Z)+\theta_0 Z+\theta_1$. The wire assignments $b(Z)$ satisfies copy constraints of $s(Z)$, if and only if the equation $\prod_{z\in\mathcal{Z}} f(z,\boldsymbol{\theta})=\prod_{z\in\mathcal{Z}} g(z,\boldsymbol{\theta})$ holds.*

We omit the proof of this claim. Instead, we will provide an extended version of this claim for bivariate polynomials in the next section with a formal proof (See Lemma 3).

The sufficient and necessary condition of the above claim can be equivalently satisfied by showing a recursion polynomial $r\in\mathbb{F}_{m,2,2}[Z,\boldsymbol{\theta}]$, constructed as

$$\begin{cases} r(\omega_Z^0,\boldsymbol{\theta})=1, \\ r(\omega_Z^{i+1},\boldsymbol{\theta})g(\omega_Z^i,\boldsymbol{\theta})=r(\omega_Z^i,\boldsymbol{\theta})f(\omega_Z^i,\boldsymbol{\theta}), \quad \text{for } 0\le i\le m-1. \end{cases} \tag{4}$$

In a permutation argument, the prover is requested to show the knowledge of $r(Z,\boldsymbol{\theta})$. Suppose the verifier can access $f(Z,\boldsymbol{\theta})$ and $g(Z,\boldsymbol{\theta})$. The verifier accepts a claim polynomial $r(Z,\boldsymbol{\theta})$ if and only if two polynomial $p_1(Z,\boldsymbol{\theta}):=(r(Z,\boldsymbol{\theta})-1)K_0(Z)$ and $p_2(Z,\boldsymbol{\theta}):=r(\omega_Z Z,\boldsymbol{\theta})g(Z,\boldsymbol{\theta})-r(Z,\boldsymbol{\theta})f(Z,\boldsymbol{\theta})$ vanishes on $\mathcal{Z}$, where $K_0(Z)$ vanishes on $\mathcal{Z}-\{\omega_Z^0\}$ and $K_0(\omega_Z^0)=1$. It is easy to see that the two polynomials p_1 and p_2 are to check the first and second conditions of the above claim, respectively.

By Lemma 2, the constraints on $r(Z,\boldsymbol{\theta})$ are satisfied if and only if there exists quotient polynomials $q_1,q_2\in\mathbb{F}[Z,\boldsymbol{\theta}]$ of degree less than $m-1$ in Z such that, for each $i\in\{0,1\}$,

$$p_i(Z,\boldsymbol{\theta})=q_i\left(Z,\boldsymbol{\theta}\right)t_{\mathcal{Z}}\left(Z\right). \tag{5}$$

To argue the copy constraints in (5) efficiently, Gabizon *et al.* in [7] have introduced a highly optimized protocol. Suppose there is an SRS $\boldsymbol{\sigma}=([(z^i)_{i=0}^{m-1}]_1,[1]_2,[z]_2)$ with a random trapdoor $z\in\mathbb{F}^*$. The protocol proceeds as follows: *1)* prover commits to $b(Z)$; *2)* verifier picks $\boldsymbol{\theta}\in(\mathbb{F}^*)^2$ at random; *3)* prover commits to $r(Z,\boldsymbol{\theta})$; *4)* verifier picks $\kappa_0 \in \mathbb{F}^*$ at random; *5)* prover commits to a quotient polynomial $q(Z,\boldsymbol{\theta}) := q_1(Z,\boldsymbol{\theta}) + \kappa_0 q_2(Z,\boldsymbol{\theta})$; *6)* verifier picks $\xi \in \mathbb{F}$ at random; *7)* prover computes the evaluations $r(\xi,\boldsymbol{\theta})$ and $r(\omega_Z\xi,\boldsymbol{\theta})$; *8)* verifier picks $\kappa_1 \in \mathbb{F}^*$ at random; *9)* prover commits to two quotient polynomials $h_0, h_1 \in \mathbb{F}[Z]$ that satisfies, respectively,

$$\begin{pmatrix} \big(r(\xi,\boldsymbol{\theta})-1\big)K_0(Z)+\kappa_0\big(r(\omega_Z\xi,\boldsymbol{\theta})g(Z,\boldsymbol{\theta})-r(\xi,\boldsymbol{\theta})f(Z,\boldsymbol{\theta})\big) \\ -q(Z)t_{\mathcal{Z}}(\xi)+\kappa_1\big(r(Z,\boldsymbol{\theta})-r(\xi,\boldsymbol{\theta})\big) \end{pmatrix} = h_0(Z)(Z-\xi),$$
$$r(Z,\boldsymbol{\theta})-r(\omega_Z\xi,\boldsymbol{\theta}) = h_1(Z)(Z-\omega_Z\xi);$$

10) verifier finally accepts the transcript $\mathbf{t} = ([b(z)]_1,\ \boldsymbol{\theta},\ [r(z)]_1,\ \kappa_0,\ [q(z)]_1,\ \xi,\ r(\xi),\ r(\omega_Z\xi),\ \kappa_1,\ [h_0(z)]_1,\ [h_1(z)]_1)$, if and only if the following two equations hold:

$$e\begin{pmatrix} \big(r(\xi,\boldsymbol{\theta})-1\big)[K_0(z)]_1+\kappa_0\big(r(\omega_Z\xi,\boldsymbol{\theta})[g(z,\boldsymbol{\theta})]_1-r(\xi,\boldsymbol{\theta})[f(z,\boldsymbol{\theta})]_1\big) & ,[1]_2 \\ -t_{\mathcal{Z}}(\xi)[h(z)]_1+\kappa_1\big([r(z,\boldsymbol{\theta})]_1-r(\xi,\boldsymbol{\theta})[1]_1\big) & \end{pmatrix} = e\big([h_0(z)]_1,[z]_2-\xi[1]_2\big), \tag{6}$$
$$e\big([r(z,\boldsymbol{\theta})]_1-r(\omega_Z\xi,\boldsymbol{\theta})[1]_1 \quad ,[1]_2\big) = e\big([h_1(z)]_1,[z]_2-\omega_Z\xi[1]_2\big),$$

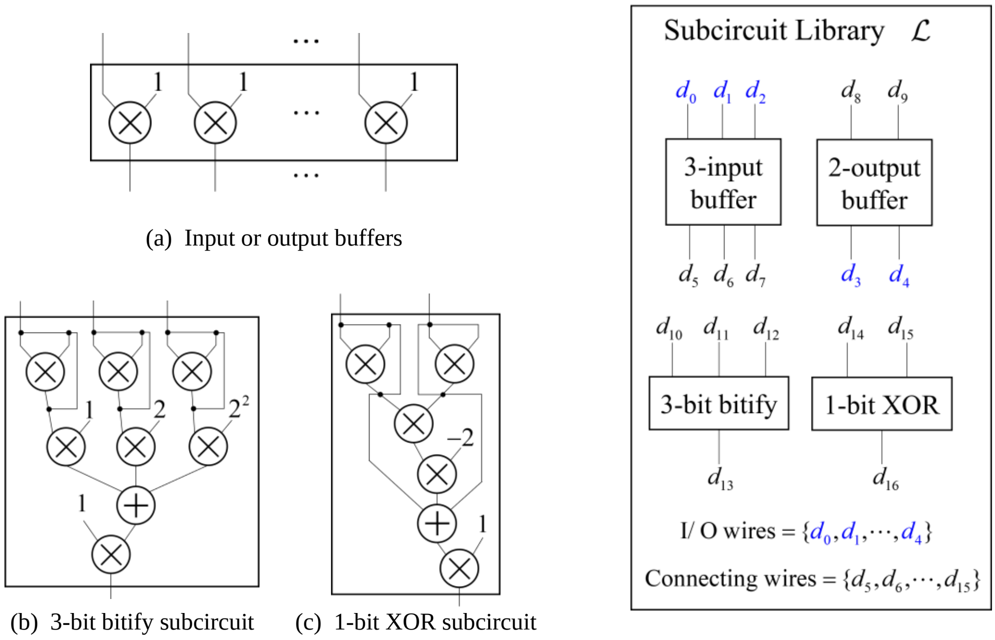
where $[f(z,\boldsymbol{\theta})]_1 = [b(z)]_1+\theta_0[s(z)]_1+\theta_1[1]_1$ and $[g(z,\boldsymbol{\theta})]_1 = [b(z)]_1+\theta_0[z]_1+\theta_1[1]_1$. It was assumed that the verifier can access $[s(z)]_1$, which is an honest commitment to $s(Z)$.

It is straightforward to see the completeness that if the prover knows the recursion polynomial r that satisfies (4), the verifier accepts the transcript (with assuming $\boldsymbol{\theta}$ is of indeterminates). Conversely, by recalling the evaluation binding of KZG commitment scheme, we can see this protocol is knowledge sound (with assuming that κ_0 and κ_1 are indeterminates): By the evaluation binding, satisfying (6) implies satisfying (5). We will provide rigorous proofs for completeness, knowledge soundness, and zero-knowledge of an extended protocol in Section 5.

3 Front-end preprocess: System of constraints and setup algorithm

We define a system of constraints based on a *subcircuit library* $\mathcal{L}$. Our system is parameterized by the number of subcircuits in $\mathcal{L}$, denoted by s_D, the maximum number of constraints that a subcircuit can contain, denoted by n, and the maximum number of subcircuit copies that can be placed in a circuit, denoted by $s_{\max}$.

Subcircuit library. A subcircuit library is defined as $\mathcal{L} := \bigcup_{k=0}^{s_D-1} \mathsf{c}_k$, where c_0 is an input buffer circuit, c_{s_D-1} is an output buffer circuit, and $\mathsf{c}_1,\cdots,\mathsf{c}_{s_D-2}$ are custom circuits. All the circuits in $\mathcal{L}$ are referred to as *subcircuits* and defined in the QAP representation. We let $m^{(k)}$ denote the total number of wires that compose the subcircuit c_k, and out of those we let $l^{(k)}$ denote the number of input and output wires. The input buffer c_0 can be placed in a circuit to take as input instance of length l_{in}, and the output buffer c_{s_D-1} returns output instance of length l_{out}. In other words, $l^{(0)} = 2l_{in}$ and $l^{(s_D-1)} = 2l_{out}$. We let $l := l_{in} + l_{out}$ denote the length of the input and output of a circuit. Figure 3 depicts an example of the subcircuit library $\mathcal{L}$.



(a) Input or output buffers

(b) 3-bit bitify subcircuit

(c) 1-bit XOR subcircuit

(d) An example of subcircuit library

Figure 3. An example of subcircuit library $\mathcal{L}$, which consists of I/O buffers, 3-bit bitify subcircuit, 1-bit XOR subcircuit. (a) I/O buffers check consistency between the input and output. (b) Bitify subcircuit verifies first whether each input is binary and then the binary representation of the output. (c) XOR subcircuit verifiers first whether each input is binary and then XOR of the input. (d) In $\mathcal{L}$, I/O wires and connecting wires are separately listed.

Circuit derivation. A system of constraints constraint we consider are a class of field-programmable circuits. In this class, a circuit $\mathcal{C}$ is programed by a *placement* of at most $s_{\max}$ copies of the subcircuits in $\mathcal{L}$, denoted by $\mathbf{Q} = (\mathtt{d}^{(0)}, \cdots, \mathtt{d}^{(s_{\max}-1)})$, where $\mathtt{d}^{(i)} \in \{\mathtt{c}_k\}_{k=0}^{s_D-1}$ denotes an *active subcircuit*, and wiring between the copies. The wiring is encoded into a permutation ρ, called a *wire map*, which entails data transfer only between subcircuits but does not involve data transfer between internal gates within each subcircuit. In other words, given a fixed $\mathcal{L}$, $\mathcal{C} := (\mathbf{Q}, \rho)$. The activation of subcircuits is determined by ρ. Figure 4 exemplifies how a wire map programs a circuit.

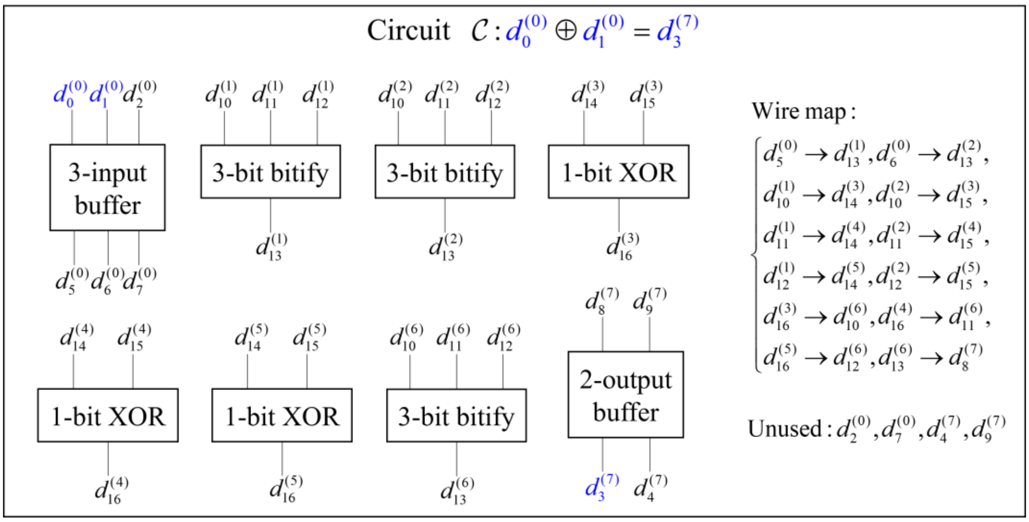


Figure 4. An example circuit $\mathcal{C}$, where the program takes two inputs of 3-bits and returns the XOR of them. Subcircuits that make up this circuit are copied from $\mathcal{L}$ in Figure 3. Each wire is represented as $d_j^{(i)}$, which denotes the i -th copy of $\mathcal{L}$'s j -th wire. In the wire map, $a \to b$ denotes that an input b is driven by an output a .

3.1 Compilers

QAP compiler. QAP compiler outputs the QAP of $\mathcal{L}$. According to the QAP representation, we define $\mathrm{c}_k \subset \mathcal{L}$ for each $k = 0, \cdots, s_D - 1$ as

$$\mathrm{c}_k := \left\{ u_0^{(k)}, \cdots, u_{m^{(k)}-1}^{(k)}, v_0^{(k)}, \cdots, v_{m^{(j)}-1}^{(k)}, w_0^{(k)}, \cdots, w_{m^{(k)}-1}^{(k)} \right\} \subset \mathbb{F}_n[X].$$

A library $\mathcal{L}$ is defined as $\mathcal{L} := \bigcup_{k=0}^{s_D-1} \mathrm{c}_k$. Denoting $l_D := \sum_{k=0}^{s_D-1} l^{(k)}$ and $m_D := \sum_{k=0}^{s_D-1} m^{(k)}$, $\mathcal{L}$ contains $3m_D$ polynomials in total, and out of those $3l_D$ polynomials are related to the input and output wires of each subcircuit. We sort the elements of $\mathcal{L}$ as

$$\mathcal{L} = \left\{ \begin{matrix} u_0, \cdots, u_{l_{in}-1}, u_{l_{in}}, \cdots, u_{l-1}, u_l, \cdots, u_{l_D-1}, u_{l_D}, \cdots, u_{m_D-1}, \\ v_0, \cdots, v_{l_{in}-1}, v_{l_{in}}, \cdots, v_{l-1}, v_l, \cdots, v_{l_D-1}, v_{l_D}, \cdots, v_{m_D-1}, \\ w_0, \cdots, w_{l_{in}-1}, w_{l_{in}}, \cdots, w_{l-1}, w_l, \cdots, w_{l_D-1}, w_{l_D}, \cdots, w_{m_D-1} \end{matrix} \right\} \subset \mathbb{F}_n[X].$$

Each polynomial $o_j \in \{u_j, v_j, w_j\} \subset \mathcal{L}$ is picked from $z_j^{(k)} \in \{u_j^{(k)}, v_j^{(k)}, w_j^{(k)}\} \subset \mathrm{c}_k$ by the following rules:

$$\begin{aligned} &(o_0, \cdots, o_{l-1}) = (z_0^{(0)}, \cdots, z_{l_{in}-1}^{(0)}, z_{l_{out}}^{(s_D-1)}, \cdots, o_{2l_{out}-1}^{(s_D-1)}), \\ &(o_l, \cdots, o_{l_D-1}) = (z_{l_{in}}^{(0)}, \cdots, z_{l^{(0)}-1}^{(0)}, z_0^{(s_D-1)}, \cdots, z_{l_{out}-1}^{(s_D-1)}, z_0^{(1)}, \cdots, z_{l^{(1)}-1}^{(1)}, z_0^{(2)}, \cdots, z_{l^{(2)}-1}^{(2)}, \cdots, z_0^{(s_D-2)}, \cdots, z_{l^{(s_D-2)}-1}^{(s_D-2)}), \\ &(o_{l_D}, \cdots, o_{m_D-1}) = (z_{l^{(0)}}^{(0)}, \cdots, z_{m^{(0)}}^{(0)}, z_{l^{(1)}}^{(1)}, \cdots, z_{m^{(1)}}^{(1)}, \cdots, z_{l^{(s_D-1)}}^{(s_D-1)}, \cdots, z_{m^{(s_D-1)}}^{(s_D-1)}). \end{aligned} \tag{7}$$

The polynomials are arranged by the role of the corresponding wires. The first polynomials o_j for $j \in \{0, \cdots, l-1\}$ represent *I/O wires*, which are used only for taking and returning the circuit input and output. We put a restriction on $\mathcal{C}$ that the first and last slots in a placement must be occupied by the input and output buffers c_0 and c_{s_D-1} , respectively. The next polynomials o_j for $j \in \{l, \cdots, l_D - 1\}$ represent *connecting wires*, which are used only for transferring data from one subcircuit to other subcircuits. The last polynomials o_j for

$j \in \{l_D, \cdots, m_D - 1\}$ represent *internal wires* inside subcircuits.

Synthesizer. Given $\mathcal{L}$ as input, synthesizer outputs $\mathbf{Q}$ and ρ, those together form a circuit $\mathcal{C} = (\mathbf{Q}, \rho)$. By setting the wire assignments in all inactive subcircuits to zero, we can rewrite $\mathbf{Q} = (\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(s_{\max}-1)})$, where $\mathcal{L}^{(i)}$ is the i-th copy of $\mathcal{L}$. Synthesis draws a wire map that describes how the connecting wires in each copy are connected to those in another copy. Wires that are connected to each other must share the same value assignment.

Formally, motivated by [5, 7], we define a wire map as a permutation ρ. Let $d_j^{(i)} \in \mathbb{F}$ denote a *value assignment* to the wire $u_j, v_j, w_j \in \mathcal{L}^{(i)}$ for $i \in \{0, \cdots, s_{\max} - 1\}$. We collect and write $b_{j-l}^{(i)} = d_j^{(i)}$ for $j \in \{l, \cdots, l_D - 1\}$, which are the value assignments to the connecting wires of $\mathcal{L}^{(i)}$. As an index set for $b_j^{(i)}$, we define $\mathcal{N} := \{0, \cdots, s_{\max} - 1\} \times \{0, \cdots, l_D - l - 1\}$. Then, a wire map is defined as a permutation $\rho : \mathcal{N} \to \mathcal{N}$. To construct the mapping rule of ρ, we divide $\mathcal{N}$ into M disjoint subsets $\mathcal{N}_k$ for $k = 0, \cdots, M-1$ so that all $b_j^{(i)}$ indexed by $(i, j) \in \mathcal{N}_k$ share the same value. We define cycles $\rho_k : \mathcal{N}_k \to \mathcal{N}_k$ for $k = 0, \cdots, M-1$. The permutation ρ is finally constructed as an integration of the cycles

$$\rho = (\rho_0) \cdots (\rho_{M-1}). \tag{8}$$

In the example circuit in Figure 4, a permutation ρ can be defined over $M = 12$ disjoint subsets, each with $|\mathcal{N}_k| = 2$.

3.2 System of constraints

Given a circuit $\mathcal{C} = (\mathbf{Q}, \rho)$, a system of constraints contains two subsystems: arithmetic constraints and copy constraints. Arithmetic constraints checks whether all wire assignments $d_j^{(i)}$ for $h \in \{0, \cdots, s_{\max} - 1\}$ and $j \in \{0, \cdots, m_D\}$ satisfy the QAP of $\mathbf{Q}$. Copy constraints checks the correctness of the connection between subcircuits in the placement by checking whether the wire assignments to the connecting wires, i.e., $b_j^{(i)}$ for $(i, j) \in \mathcal{N}$ satisfy a permutation ρ.

For the construction of a constraint system, we need Lagrange bases $L_i \in \mathbb{F}_{s_{\max}}[Y]$ and $K_j \in \mathbb{F}_{l_D - l}[Z]$. Given vanishing sets $\mathcal{Y} = \{\omega_Y^i\}_{i=0}^{s_{\max}-1}$ and $\mathcal{Z} = \{\omega_Z^j\}_{j=0}^{l_D - l}$, the Lagrange bases L_i for $i \in \{0, \cdots, s_{\max} - 1\}$ satisfy $L_i(\omega_Y^i) = 1$ and $L_i(\omega_Y^k) = 0$ for every $k \neq i$. The other Lagrange bases $K_i(Z)$ for $i = 0, \cdots, l_D - l - 1$ are defined in the same way with $\mathcal{Z}$. We also define vanishing polynomials $t_{\mathcal{Y}} \in \mathbb{F}_{s_{\max}+1}[Y]$ and $t_{\mathcal{Z}} \in \mathbb{F}_{l_D - l + 1}[Z]$ corresponding to $\mathcal{Y}$ and $\mathcal{Z}$, respectively.

We encode some of the wire assignments $\{d_j^{(i)}\}_{i=0}^{s_{\max}-1}$ for each $j \in \{0, \cdots, m_D - 1\}$ into polynomials according to the roles of wires as discussed in (7):

$$\sum_{i=0}^{s_{\max}-1} d_j^{(i)} L_i(Y) =: \begin{cases} b_{j-l}(Y), & \text{for } j = l, \cdots, l_D - 1, \\ c_{j-l_D}(Y), & \text{for } j = l_D, \cdots, m_D - 1. \end{cases} \tag{9}$$

1) *Arithmetic constraints*

The arithmetic constraints can be represented by a set of equations: for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$,

$$p_0\left(x,y\right)=0, \tag{10}$$

where

$$p_0(X,Y):=U(X,Y)V(X,Y)-W(X,Y),$$

and for $O\in\{U,V,W\}$,

$$O(X,Y):=\sum_{j=0}^{l_{in}-1}d_j^{(0)}L_0(Y)o_j(X)+\sum_{j=l_{in}}^{l-1}d_j^{(s_D-1)}L_{-1}(Y)o_j(X)+\sum_{j=l}^{l_D-1}b_{j-l}(Y)o_j(X)+\sum_{j=l_D}^{m_D-1}c_{j-l_D}(Y)o_j(X), \tag{11}$$

where $o_i=u_i$, if $O=U$; $o_i=v_i$, if $O=V$; and $o_i=w_i$, if $O=W$.

Applying Corollary 1 to the $ns_{\max}$ equations of (10), the arithmetic constraints are satisfied if and only if

$$\exists q_0,q_1\in\mathbb{F}[X,Y]:\ p_0(X,Y)=q_0(X,Y)t_{\mathcal{X}}(X)+q_1(X,Y)t_{\mathcal{Y}}(Y), \tag{12}$$

where $\mathcal{X}$ is a vanishing set of size n and $t_{\mathcal{X}}$ is the corresponding vanishing polynomial.

2) Copy constraints

The copy constraints check whether $B(Y,Z)$ satisfies a permutation ρ, where

$$B(Y,Z):=\sum_{j=0}^{l_D-l-1}b_j(Y)K_j(Y). \tag{13}$$

For $\rho(i,j)\mapsto(h,k)$, we write $\rho(i,j)_1=h$ and $\rho(i,j)_2=k$. We say the copy constraints are satisfied if and only if $B(\omega_Y^i,\omega_Z^j)=B(\omega_Y^{\rho(i,j)_1},\omega_Z^{\rho(i,j)_2})$, i.e., $b_j^{(i)}=b_{\rho(i,j)_2}^{(\rho(i,j)_1)}$ for every connecting wire index $(i,j)\in\mathcal{N}$.

Motivated by [5, 7, 33], we construct a permutation check algorithm for the copy constraints. We first encode ρ into permutation polynomials $s^{(0)},s^{(1)},s^{(2)}\in\mathbb{F}[Y,Z]$ such that for $(i,j)\in\mathcal{N}$,

$$\begin{aligned}
&s^{(0)}(\omega_Y^i,\omega_Z^j)=\omega_Y^{\rho(i,j)_1},\\
&s^{(1)}(\omega_Y^i,\omega_Z^j)=\omega_Z^{\rho(i,j)_2},\\
&s^{(2)}(\omega_Y^i,\omega_Z^j)=\omega_Z^j \Leftrightarrow s^{(2)}(Y,Z)=Z.
\end{aligned} \tag{14}$$

With introducing indeterminates $\boldsymbol{\theta}=(\theta_0,\theta_1,\theta_2)$, we also define

$$\begin{aligned}
f\left(Y,Z,\boldsymbol{\theta}\right)&:=B\left(Y,Z\right)+\theta_0 s^{(0)}\left(Y,Z\right)+\theta_1 s^{(1)}\left(Y,Z\right)+\theta_2,\\
g\left(Y,Z,\boldsymbol{\theta}\right)&:=B\left(Y,Z\right)+\theta_0 Y+s^{(2)}\left(Y,Z\right)\theta_1+\theta_2.
\end{aligned} \tag{15}$$

Lemma 3 below is useful for checking the copy constraints.

Lemma 3. *Given polynomials f,g defined in (15), $B(Y,Z)$ satisfies copy constraints with ρ, if and only if the following equation holds*

$$\prod_{y\in\mathcal{Y},z\in\mathcal{Z}} f(y,z,\boldsymbol{\theta}) = \prod_{y\in\mathcal{Y},z\in\mathcal{Z}} g(y,z,\boldsymbol{\theta}). \tag{16}$$

Proof. For the simplicity of expression, we denote $b_{i,j} = b(\omega_Y^i, \omega_Z^j)$ for $(i,j)\in\mathcal{N}$. If $b_{i,j} = b_{\rho(i,j)}$ then the factors on both sides are the same, just in a different order, so the equation holds. Conversely, suppose the equation (16) holds. Consider the roots of θ_0 on both sides given by

$$\left\{-\frac{b_{i,j} + \omega_Z^{\rho(i,j)_2}\theta_1 + \theta_2}{\omega_Y^{\rho(i,j)_1}} : (i,j)\in\mathcal{N}\right\}, \left\{-\frac{b_{h,k} + \omega_Z^{k}\theta_1 + \theta_2}{\omega_Y^{h}} : (h,k)\in\mathcal{N}\right\}.$$

The equation implies that two sets of roots must be the same, i.e., for every $(i,j)\in\mathcal{N}$, there must exist $(h,k)\in\mathcal{N}$ such that

$$\omega_Y^h\left(b_{i,j} + \omega_Z^{\rho(i,j)_2}\theta_2 + \theta_3\right) = \omega_Y^{\rho(i,j)_1}\left(b_{h,k} + \omega_Z^{k}\theta_2 + \theta_3\right).$$

Since there is the unique pair $((h,i),(j,k))\in\mathcal{N}$ such that $\omega_Y^j = \omega_Y^{\rho(h,i)_1}$ and $\omega_Z^k = \omega_Z^{\rho(h,i)_2}$ by the definition of ρ, we conclude that on those indices it holds $b_{h,i} = b_{j,k}$. In other words, $b_{i,j} = b_{\rho(i,j)}$. □

For an efficient utilization of Lemma 3, we define a recursion polynomial $r\in\mathbb{F}[Y,Z,\boldsymbol{\theta}]$ such that

$$\begin{cases} r(\omega_Y^{s_{\max}-1}, \omega_Z^{l_D-l-1}, \boldsymbol{\theta}) = 1, \\ r(\omega_Y^i, \omega_Z^j, \boldsymbol{\theta}) g(\omega_Y^i, \omega_Z^j, \boldsymbol{\theta}) = r(\omega_Y^i, \omega_Z^{j-1}, \boldsymbol{\theta}) f(\omega_Y^i, \omega_Z^j, \boldsymbol{\theta}), \quad \text{for } 0\le i\le s_{\max}-1, 0< j\le l_D-l-1, \\ r(\omega_Y^i, \omega_Z^0, \boldsymbol{\theta}) g(\omega_Y^i, \omega_Z^0, \boldsymbol{\theta}) = r(\omega_Y^{i-1}, \omega_Z^{l_D-l-1}, \boldsymbol{\theta}) f(\omega_Y^i, \omega_Z^0, \boldsymbol{\theta}), \quad \text{for } 0\le i\le s_{\max}-1. \end{cases} \tag{17}$$

It is straightforward to see that there exists $r(Y,Z,\boldsymbol{\theta})$ that holds (17), if and only if the equation (16) holds.

By Corollary 1, the polynomial $r(Y,Z,\boldsymbol{\theta})$ satisfies (17), if and only if there exist $q_i\in\mathbb{F}[Y,Z,\boldsymbol{\theta}]$ for $i\in\{2,\cdots,7\}$ such that

$$\begin{aligned} p_1(Y,Z,\boldsymbol{\theta}) &= q_2(Y,Z,\boldsymbol{\theta})t_{\mathcal{Y}}(Y) + q_3(Y,Z,\boldsymbol{\theta})t_{\mathcal{Z}}(Z), \\ p_2(Y,Z,\boldsymbol{\theta}) &= q_4(Y,Z,\boldsymbol{\theta})t_{\mathcal{Y}}(Y) + q_5(Y,Z,\boldsymbol{\theta})t_{\mathcal{Z}}(Z), \\ p_3(Y,Z,\boldsymbol{\theta}) &= q_6(Y,Z,\boldsymbol{\theta})t_{\mathcal{Y}}(Y) + q_7(Y,Z,\boldsymbol{\theta})t_{\mathcal{Z}}(Z), \end{aligned} \tag{18}$$

where

$$\begin{aligned} p_1(Y,Z,\boldsymbol{\theta}) &:= \left(r(Y,Z,\boldsymbol{\theta})-1\right)L_{-1}(Y)K_{-1}(Z), \\ p_2(Y,Z,\boldsymbol{\theta}) &:= \left(Z-1\right)\left(r(Y,Z,\boldsymbol{\theta})g(Y,Z,\boldsymbol{\theta}) - r(Y,\omega_Z^{-1}Z,\boldsymbol{\theta})f(Y,Z,\boldsymbol{\theta})\right), \\ p_3(Y,Z,\boldsymbol{\theta}) &:= K_0(Z)\left(r(Y,Z,\boldsymbol{\theta})g(Y,Z,\boldsymbol{\theta}) - r(\omega_Y^{-1}Y,\omega_Z^{-1}Z,\boldsymbol{\theta})f(Y,Z,\boldsymbol{\theta})\right). \end{aligned} \tag{19}$$

3) Integrating all the constraints

With introducing an indeterminate κ, we integrate all the constraint polynomials $p_0,\cdots,p_3$ into a single polynomial $p\in\mathbb{F}[X,Y,Z,\boldsymbol{\theta},\kappa]$ as follow:

$$p\left(X,Y,Z,\boldsymbol{\theta},\kappa\right) := p_0\left(X,Y\right)+\sum_{i=1}^{3}\kappa^i p_i\left(Y,Z,\boldsymbol{\theta}\right). \quad (20)$$

We finally define the constraint system as a relation generator $\mathcal{R}$. The relation generator takes as input a security parameter λ, a subcircuit library $\mathcal{L} \subset \mathbb{F}[X]$, and permutation polynomials $s^{(0)}, s^{(1)}, s^{(2)} \in \mathbb{F}[Y,Z]$ and generates a polynomial-time decidable binary relation R, which is a set of pairs of instance and witness $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y))$, where $\mathbf{a}^{(in)} = (d_0^{(0)},\cdots,d_{l_{in}-1}^{(0)})$, $\mathbf{a}^{(out)} = (d_{l_{in}}^{(s_D-1)},\cdots,d_{l-1}^{(s_D-1)})$, $\mathbf{b}(Y) = (b_0(Y),\cdots,b_{l_D-l-1}(Y))$, $\mathbf{c}(Y) = (c_0(Y),\cdots,c_{m_D-l_D-1}(Y))$, such that the polynomial $p(X,Y,Z,\boldsymbol{\theta},\kappa)$ vanishes on $\mathcal{X}\times\mathcal{Y}\times\mathcal{Z}$. Formally, R is constructed as

$$R = \left\{ ((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)) \in \mathbb{F}^l \times \mathbb{F}^{m_D-l}[Y] \left| \begin{array}{r} \exists q_{\mathcal{X}}, q_{\mathcal{Y}}, q_{\mathcal{Z}} \in \mathbb{F}[X,Y,Z,\boldsymbol{\theta},\kappa]: \\ \left[\begin{array}{r} p(X,Y,Z,\boldsymbol{\theta},\kappa) = q_{\mathcal{X}}(X,Y,Z,\boldsymbol{\theta},\kappa)t_{\mathcal{X}}(X) \\ +q_{\mathcal{Y}}(X,Y,Z,\boldsymbol{\theta},\kappa)t_{\mathcal{Y}}(Y) \\ +q_{\mathcal{Z}}(X,Y,Z,\boldsymbol{\theta},\kappa)t_{\mathcal{Z}}(Z) \end{array}\right] \end{array} \right. \right\}. \quad (21)$$

3.3 Setup of subcircuit library

Our back-end protocol that will be defined in the next section relies on a probabilistic algorithm `Setup` for R that produces an encoded reference string $\boldsymbol{\sigma}$ of the library subcircuit polynomials in $\mathcal{L}$. Parties of the back-end protocol will be enforced to use $\boldsymbol{\sigma}$, by which a prover can compress a claim statement $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y))$ for R into proof of a small size. A verifier can be convinced by $\boldsymbol{\sigma}$ that the counterparty is disputing a circuit derived from the same library $\mathcal{L}$. Also, randomness in $\boldsymbol{\sigma}$ keeps $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y))$ extractable from the compression.

However, `Setup` may not include permutation polynomials $\{s^{(0)},s^{(1)},s^{(2)}\}$, when leaving the parties to commit to them by themselves grants universality to the back-end protocol. In special cases where universality is guaranteed even if the permutation polynomials are fixed, we can consider appending them to the reference string. Section 6 will illustrate one of these cases, verifiable machine computation.

`Setup`$(pp_\lambda,\mathcal{L})$ takes as input the bilinear pairing group pp_λ and the subcircuit library $\mathcal{L} = \{u_j(X),v_j(X),w_j(X)\}_{j=0}^{m_D-1}$, picks a trapdoor

$$\boldsymbol{\tau} := \left(x,y,z,\alpha,\beta,\gamma,\delta,\eta_0,\eta_1,\mu,\nu,\psi_0,\psi_1,\psi_2,\psi_3,\kappa\right) \overset{\$}{\leftarrow} (\mathbb{F}^*)^{16}, \quad (22)$$

computes $L_i(y)o_j(x)$ for every $j \in \{0,\cdots,m_D-1\}$ and $i \in \{0,\cdots,s_{\max}-1\}$, where

$$o_j(X) := \beta u_j(X) + \alpha v_j(X) + w_j(X),$$

and $M_j(x,z)$ for $j \in \{l,\cdots,l_D-1\}$ as defined

$$M_j(X,Z) := \sum_{k=l,k\neq j}^{l_D-1} \frac{o_k(X)}{l_D-l}\left(\frac{\omega_Z^k K_{j-l}(Z) - \omega_Z^j K_{k-l}(Z)}{\omega_Z^j - \omega_Z^k}\right), \quad (23)$$

and returns $\boldsymbol{\sigma} = ([\boldsymbol{\sigma}_{A,I}]_1,[\boldsymbol{\sigma}_C]_1,[\boldsymbol{\sigma}_{zk}]_1,[\boldsymbol{\sigma}_V]_2)$, where

$$\boldsymbol{\sigma}_{A,I} := \left(\begin{array}{l} \alpha, \left(x^h y^i\right)_{h=0,i=0}^{n-1,s_{\max}-1}, \\ \left(\gamma^{-1} L_0(y) o_j(x)\right)_{j=0}^{l_{in}-1}, \left(\gamma^{-1} L_{-1}(y) o_j(x)\right)_{j=l_{in}}^{l-1}, \left(\eta_1^{-1} L_i(y) o_j(x)\right)_{i=0,\, j=l}^{s_{\max}-1, l_D-1}, \left(\delta^{-1} L_i(y) o_j(x)\right)_{i=0,\, j=l_D}^{s_{\max}-1, m_D-1}, \\ \left(\eta_0^{-1} L_i(y) o_j(x) \left(K_{j-l}^2(z) - 1\right)\right)_{i=0,\, j=l}^{s_{\max}-1, l_D-1}, \\ \left(\delta^{-1} x^h y^i t_{\mathcal{X}}(x)\right)_{h=0,i=0}^{n-2,s_{\max}-1}, \left(\delta^{-1} x^h y^i t_{\mathcal{Y}}(y)\right)_{h=0,i=0}^{2n-2,s_{\max}-2}, \left(\eta_0^{-1} L_i(y) M_j(x,z) t_{\mathcal{Z}}(z)\right)_{i=0,\, j=l}^{s_{\max}-1, l_D-1} \end{array} \right),$$

$$\boldsymbol{\sigma}_{C} := \left(\begin{array}{l} \left(\mu^{-1} L_i(y) K_j(z)\right)_{i=0,\, j=0}^{s_{\max}-1, l_D-l-1}, \\ \left(\nu^{-1} y^i z^j t_{\mathcal{Y}}(y)\right)_{i=0,\, j=0}^{s_{\max}-2, 2l_D-2l-2}, \left(\nu^{-1} y^i z^j t_{\mathcal{Z}}(z)\right)_{i=0,\, j=0}^{i=2s_{\max}-2,\, j=2(l_D-l)-3}, \\ \left(\psi_0^{-1} \kappa^h y^i z^j\right)_{h=0,i=0,\, j=0}^{1,2s_{\max}-3,3(l_D-l-1)}, \left(\psi_1^{-1} z^j\right)_{j=0}^{3(l_D-l)-4}, \left(\psi_2^{-1} \kappa^2 y^i z^j\right)_{i=0,\, j=0}^{s_{\max}-2, l_D-l-1}, \left(\psi_3^{-1} \kappa^h z^j\right)_{h=1,\, j=0}^{2, l_D-l-2}, \end{array} \right),$$

$$\boldsymbol{\sigma}_{zk} := \left(\begin{array}{l} \beta, \delta, \eta_1, \left(\mu^{-1} y^i t_{\mathcal{Y}}(y)\right)_{i=0}^{1}, \eta_0^{-1} t_{\mathcal{Y}}(y) \sum_{j=l}^{l_D-1} M_j(x,z) t_{\mathcal{Z}}(z), \\ \eta_1^{-1} t_{\mathcal{Y}}(y) \sum_{j=l}^{l_D-1} o_j(x), \eta_0^{-1} t_{\mathcal{Y}}(y) \sum_{j=l}^{l_D-1} o_j(x) \left(K_{j-l}^2(z) - 1\right), \\ \left(\nu^{-1} y^i z^j t_{\mathcal{Y}}(y)\right)_{i=s_{\max}-1,\, j=0}^{s_{\max}+1, 2l_D-2l-2}, \left(\psi_0^{-1} \kappa^h y^i z^j\right)_{h=0,i=2s_{\max}-2,\, j=0}^{1,2s_{\max},3(l_D-l-1)}, \left(\psi_2^{-1} \kappa^2 y^i z^j\right)_{i=s_{\max}-1,\, j=0}^{s_{\max}, l_D-l-1} \end{array} \right),$$

$$\boldsymbol{\sigma}_{\mathcal{V}} := \left(\begin{array}{l} \beta, \gamma, \delta, \eta_1, \mu\eta_0, \mu\eta_1, \left(x^h y^i\right)_{h=0,i=0}^{n-1,s_{\max}-1}, \mu^2 \sum_{j=l}^{l_D-1} o_j(x) K_{j-l}(z), \\ \mu^3 \nu, \left(\mu^4 \kappa^h\right)_{h=0}^{2}, \left(\mu^3 \psi_h y^i z^j\right)_{h=0,i=0,\, j=0}^{3,1,1} \end{array} \right).$$

In the next section, the three vectors $\boldsymbol{\sigma}_{A,I}$, $\boldsymbol{\sigma}_{C}$, and $\boldsymbol{\sigma}_{zk}$ will be used for different purposes: $\boldsymbol{\sigma}_{A,I}$ is for *arithmetic constraint and inner-product arguments*, $\boldsymbol{\sigma}_{C}$ is for *copy constraint argument*, and $\boldsymbol{\sigma}_{zk}$ is for adding zero-knowledge.

4 Back-end Protocol: A SNARK for R

We construct an interactive protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ for the relation R, as shown in Figure 5. The protocol consists of a tuple of prover algorithms $\mathcal{P} = (\mathtt{Prove}_0, \mathtt{Commit}_1, \mathtt{Prove}_2, \mathtt{Eval}_3, \mathtt{Prove}_4)$ and a tuple of verifier algorithms $\mathcal{V} = (\mathtt{Open}_0, \mathtt{Open}_1, \mathtt{Open}_2, \mathtt{Open}_3, \mathtt{Verifiy})$. We sometimes write $\mathtt{IP}_{\mathcal{P},\mathcal{V}} = \langle \mathcal{P}, \mathcal{V} \rangle$. The algorithms of $\mathcal{P}$ and $\mathcal{V}$ will be defined in subsection 4.4.

The protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ consists of three arguments: the arithmetic constraint argument, the copy constraint argument, and the inner product argument. The arithmetic constraint argument argues the arithmetic constraints, while the copy constraint argument argues the copy constraints. The inner product argument connects these two by ensuring that the witness provided as proofs of both arguments are identical. We explain each argument, followed by presenting the integrated protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$.

The construction of $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ in this section excludes the zero-knowledge. Instead, in the next section, we will illustrate how $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ can restore the zero-knowledge.

	Protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ for R	
	$\mathcal{P}$	$\mathcal{V}$
1:	$\mathtt{Prove}_0\left(\mathcal{L},\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{b}(Y),\mathbf{c}(Y)\right)$ $\mapsto \left([U]_1,[V]_2,[W]_1,[A]_1,[B]_1,[C]_1\right)$	
2:		$\mathtt{Open}_0\left([U]_1,[V]_2,[W]_1,[A]_1,[B]_1,[C]_1\right)\mapsto \boldsymbol{\theta}$
3:	$\mathtt{Commit}_1\left(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\mathbf{b}(Y)\right)\mapsto \left([R]_1\right)$	
4:		$\mathtt{Open}_1\left([R]_1\right)\mapsto \kappa_0$
5:	$\mathtt{Prove}_2\left(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\kappa_0,\mathbf{b}(Y)\right)\mapsto [Q]_1$	
6:		$\mathtt{Open}_2\left([Q]_1\right)\mapsto \left(\zeta,\xi\right)$
7:	$\mathtt{Eval}_3\left(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\zeta,\xi,\mathbf{b}(Y)\right)\mapsto \left(R'_{y,z},R''_{y,z},B_{y,z}\right)$	
8:		$\mathtt{Open}_3\left(R'_{y,z},R''_{y,z},B_{y,z}\right)\mapsto \kappa_1$
9:	$\mathtt{Prove}_4\left(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\kappa_0,\zeta,\xi,\kappa_1,\mathbf{b}(Y)\right)$ $\mapsto \left([\Pi_0]_1,[\Pi_1]_1,[\Pi_2]_1,[\Pi_3]_1\right)$	
		$Let\ \mathbf{t}=\begin{pmatrix}[U]_1,[V]_2,[W]_1,[A]_1,[B]_1,[C]_1,\\ \boldsymbol{\theta},[R]_1,\kappa_0,[Q]_1,\zeta,\xi,R'_{y,z},R''_{y,z},B_{y,z},\\ \kappa_1,[\Pi_0]_1,[\Pi_1]_1,[\Pi_2]_1,[\Pi_3]_1\end{pmatrix}$
10:		$\mathtt{Verify}\left(pp_\lambda,\mathcal{L},[S^{(0)}]_1,[S^{(1)}]_1,s^{(2)}(Y,Z),\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{t}\right)\mapsto 0/1$

Figure 5. An interactive protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ for R. In the protocol, every output produced by $\mathcal{P}$ or $\mathcal{V}$ is sent to the other party, immediately.

We assume $\mathcal{V}$ is given preprocessed commitments $[S^{(i)}]_1$ for $i\in\{0,1\}$ to the permutation polynomials $s^{(i)}(Y,Z)$ in (14), where, given $\boldsymbol{\tau}$, $S^{(i)}=\mu^{-1}s^{(i)}(y,z)$. Also, according to (14), $s^{(2)}(Y,Z)=Z$. We write $\mathbf{s}(Y,Z)=(s^{(i)}(Y,Z))_{i=0}^{2}$.

4.1 Arithmetic constraint argument

The arithmetic constraint argument is based on Groth16 [4], with modifications for bivariate polynomials. In this argument, the prover claims instance and witness that satisfies the arithmetic constraints (12). Let $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)))$ denote the claimed pair of instance and witness. We write a concatenation $\mathbf{d}(Y)=\mathbf{a}^{(in)}(Y)\,|\,\mathbf{a}^{(out)}(Y)\,|\,\mathbf{b}(Y)\,|\,\mathbf{c}(Y)$, where $a_i^{(in)}(Y)=a_i^{(in)}L_0(Y)$ and $a_i^{(out)}(Y)=a_i^{(out)}L_1(Y)$.

Given a reference string $\boldsymbol{\sigma}$, the argument is $\langle \mathtt{Prove}_0,\mathtt{Verify}\rangle$ and produces a transcript $([U]_1,[V]_2,[W]_1,[C]_1)$. Each transcript component is produced by $\mathtt{Prove}_0$ as follows,

$$\begin{aligned}
[U]_1 &= [\alpha]_1+\left[\sum_{j=0}^{m_D-1} d_j(y)u_j(x)\right]_1,\\
[V]_2 &= [\beta]_2+\left[\sum_{j=0}^{m_D-1} d_j(y)v_j(x)\right]_2,\\
[W]_1 &= \sum_{j=l}^{l_D-1}\left[\eta_1^{-1}b_{j-l}(y)o_j(x)\right]_1,\\
[C]_1 &= \sum_{j=l_D}^{m_D-1}\left[\delta^{-1}c_{j-l_D}(y)o_j(x)\right]_1+\left[\delta^{-1}q_0(x,y)t_{\mathcal{X}}(x)\right]_1+\left[\delta^{-1}q_1(x,y)t_{\mathcal{Y}}(y)\right]_1,
\end{aligned}\tag{24}$$

where $q_0(X,Y)$ and $q_1(X,Y)$ are computed as defined in (12). The verifier algorithm $\mathtt{Verify}$ accepts the transcript, given public input $\mathbf{a}(Y)$, only if $E_A = 1$, where

$$E_A = e^{-1}\left([U]_1,[V]_2\right)e\left([\alpha]_1,[\beta]_2\right)e\left([W]_1,[\eta_1]_2\right)e\left([C]_1,[\delta]_2\right) \\ \cdot e\left(\sum_{j=0}^{l_{in}-1} a_j^{(in)}\left[\gamma^{-1}L_0(y)o_j(x)\right]_1 + \sum_{j=l_{in}}^{l-1} a_{j-l_{in}}^{(out)}\left[\gamma^{-1}L_{-1}(y)o_j(x)\right]_1,[\gamma]_2\right). \tag{25}$$

It is straightforward to see that if $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)))$ satisfies the copy constraint (12) and if the computations (24) and (25) are strictly followed, $E_A = 1$.

Efficiency of arithmetic constraint argument. Computing $[U]_1$ and $[V]_2$ is done in $O(ns_{\max})$ exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. According to (9), $[W]_1$ and the most left them of $[C]_1$ can be computed by $\sum_{j=l}^{l_D-1}\sum_{i=0}^{s_{\max}-1} d_j^{(i)}[\eta_1^{-1}L_i(y)o_j(x)]_1$ and $\sum_{j=l_D}^{m_D-1}\sum_{i=0}^{s_{\max}-1} d_j^{(i)}[\delta^{-1}L_i(y)o_j(x)]_1$, respectively. Since there are at most $s_{\max}$ subcircuits placed in a circuit and each subcircuit has $O(n)$ wires, computing $[W]_1$ and the most left term of $[C]_1$ is therefore done by $O(ns_{\max})$ exponentiations in $\mathbb{G}_1$. The rest two terms of $[C]_1$ are also computed by $O(ns_{\max})$ exponentiations in $\mathbb{G}_1$ by the degree bounds of $q_0(X,Y)$ and $q_1(X,Y)$. The complexity of finding the quotients $q_0(X,Y)$ and $q_1(X,Y)$ is given by $O(ns_{\max}\log ns_{\max})$ operations in $\mathbb{F}$. Finally, the verifier needs to compute l exponentiations in $\mathbb{G}_1$.

4.2 Copy constraint argument

The copy constraint argument is based on a permutation argument in [33] that was also used in Sonic [5] and PlonK [7]. We modify the permutation argument to work with bivariate polynomials and reduced interactions. In the argument, the prover claims a polynomial $B(Y,Z) := \sum_{j=0}^{l_D-l-1} b_j(Y)K_j(Z)$ that satisfies the copy constraints in Lemma 3.

Given a reference string $\boldsymbol{\sigma}$, the argument is $\langle\mathcal{P},\mathcal{V}\rangle$ and produces a transcript

$$\left([B]_1,\boldsymbol{\theta},[R]_1,\kappa_0,[Q]_1,\zeta,\xi,R'_{y,z},R''_{y,z},B_{y,z},\kappa_1,[\Pi_0]_1,[\Pi_1]_1,[\Pi_2]_1,[\Pi_3]_1\right).$$

We explain how each transcript component is computed in sequence.

Given $B(Y,Z)$ as input, the first component $[B]_1$ is produced by $\texttt{Prove}_0$ as

$$[B]_1 = \left[\mu^{-1}B(y,z)\right]_1, \tag{26}$$

which provokes the verifier algorithm $\texttt{Open}_0$ to pick a challenge $\boldsymbol{\theta}$ uniformly from $(\mathbb{F}^*)^3$ at random.

Given $\boldsymbol{\theta}$ as input, $[R]_1$ is produced by $\texttt{Commit}_1$ as

$$[R]_1 = \left[\mu^{-1}r(y,z,\boldsymbol{\theta})\right]_1, \tag{27}$$

where $r(Y,Z,\boldsymbol{\theta})$ is the recursion polynomial defined in (17). Then, a challenge κ_0 is picked uniformly from $\mathbb{F}^*$ at random by $\texttt{Open}_1$.

Given all the previous challenges as input, $[Q]_1$ is computed by $\texttt{Prove}_2$ as

$$
\begin{aligned}
[Q]_1 &= \left[\nu^{-1}\left(q_2\left(y,z,\boldsymbol{\theta}\right)+\kappa_0 q_4\left(y,z,\boldsymbol{\theta}\right)+\kappa_0^2 q_6\left(y,z,\boldsymbol{\theta}\right)\right)t_{\mathcal{Y}}\left(y\right)\right]_1 \\
&+\left[\nu^{-1}\left(q_3\left(y,z,\boldsymbol{\theta}\right)+\kappa_0 q_5\left(y,z,\boldsymbol{\theta}\right)+\kappa_0^2 q_7\left(y,z,\boldsymbol{\theta}\right)\right)t_{\mathcal{Z}}\left(z\right)\right]_1,
\end{aligned}
\tag{28}
$$

where $q_2(Y,Z,\boldsymbol{\theta}),\cdots,q_7(Y,Z,\boldsymbol{\theta})$ are defined (18). Then, the verifier algorithm $\texttt{Open}_2$ picks two challenges ζ,ξ uniformly from $\mathbb{F}^*$ at random.

Given $\boldsymbol{\theta},\zeta,\xi$, $\texttt{Eval}_3$ produces $R'_{y,z},R''_{y,z},B_{y,z}$ as follows,

$$
B_{y,z}=B(\zeta,\xi),\quad R'_{y,z}=r(\zeta,\omega_Z^{-1}\xi,\boldsymbol{\theta}),\quad R''_{y,z}=r(\omega_Y^{-1}\zeta,\omega_Z^{-1}\xi,\boldsymbol{\theta}),
\tag{29}
$$

which provokes $\texttt{Open}_3$ to pick a challenge κ_1 uniformly from $\mathbb{F}^*$ at random .

Taking all the previous challenges as input, $\texttt{Prove}_4$ computes the last transcript components $[\Pi_0]_1,\cdots,[\Pi_3]_1$ as the following procedure:

1. Define $P(Y,Z,\boldsymbol{\theta})$ as

$$
\begin{aligned}
P(Y,Z,\boldsymbol{\theta}) \coloneqq &\left(r(Y,Z,\boldsymbol{\theta})-1\right)L_{-1}(\zeta)K_{-1}(\xi)-p_1(Y,Z,\boldsymbol{\theta}) \\
&+\kappa_0\left(\left(\xi-1\right)\left(g(\zeta,\xi)r(Y,Z,\boldsymbol{\theta})-r(\zeta,\omega_Z^{-1}\xi)f(Y,Z,\boldsymbol{\theta})\right)-p_2(Y,Z,\boldsymbol{\theta})\right) \\
&+\kappa_0^2\left(K_0(\xi)\left(g(\zeta,\xi)r(Y,Z,\boldsymbol{\theta})-r(\omega_Y^{-1}\zeta,\omega_Z^{-1}\xi)f(Y,Z,\boldsymbol{\theta})\right)-p_3\left(Y,Z,\boldsymbol{\theta}\right)\right) \\
&+\kappa_1\left(B(Y,Z)-B(\zeta,\xi)\right),
\end{aligned}
\tag{30}
$$

   where $f(Y,Z,\boldsymbol{\theta})$ and $g(Y,Z,\boldsymbol{\theta})$ are given in (15) and $p_1(Y,Z,\boldsymbol{\theta}),\cdots,p_3(Y,Z,\boldsymbol{\theta})$ are given in (19).

2. Compute quotient polynomials $\pi_0,\cdots,\pi_5$ that satisfies

$$
\begin{aligned}
P(Y,Z,\boldsymbol{\theta}) &= \pi_0(Y,Z,\boldsymbol{\theta})\left(Y-\zeta\right)+\pi_1(Z,\boldsymbol{\theta})\left(Z-\xi\right), \\
r(Y,Z,\boldsymbol{\theta})-r(\zeta,\omega_Z^{-1}\xi,\boldsymbol{\theta}) &= \pi_2(Y,Z,\boldsymbol{\theta})\left(Y-\zeta\right)+\pi_3(Z,\boldsymbol{\theta})\left(Z-\omega_Z^{-1}\xi\right), \\
r(Y,Z,\boldsymbol{\theta})-r(\omega_Y^{-1}\zeta,\omega_Z^{-1}\xi,\boldsymbol{\theta}) &= \pi_4(Y,Z,\boldsymbol{\theta})\left(Y-\omega_Y^{-1}\zeta\right)+\pi_5(Z,\boldsymbol{\theta})\left(Z-\omega_Z^{-1}\xi\right).
\end{aligned}
\tag{31}
$$

3. Return $[\Pi_0]_1,\cdots,[\Pi_3]_1$, where

$$
\begin{aligned}
[\Pi_0]_1 &= \left[\psi_0^{-1}\left(\pi_0(y,z,\boldsymbol{\theta})+\kappa\pi_2(y,z,\boldsymbol{\theta})\right)\right]_1, \\
[\Pi_1]_1 &= \left[\psi_1^{-1}\pi_1(z,\boldsymbol{\theta})\right]_1, \\
[\Pi_2]_1 &= \left[\psi_2^{-1}\kappa^2\pi_4(y,z,\boldsymbol{\theta})\right]_1, \\
[\Pi_3]_1 &= \left[\psi_3^{-1}\left(\kappa\pi_3(z,\boldsymbol{\theta})+\kappa^2\pi_5(z,\boldsymbol{\theta})\right)\right]_1.
\end{aligned}
\tag{32}
$$

Finally, $\texttt{Verify}$ with the public input $[S^{(0)}]_1,[S^{(1)}]_1,s^{(2)}(Y,Z)$ computes

$$
\begin{aligned}
[P]_1 &= L_{-1}(\zeta)K_{-1}(\xi)\left([R]_1-[\mu^{-1}]_1\right)+\kappa_0\left(\xi-1\right)\left(G[R]_1-R'_{y,z}[F]_1\right)\\
&+\kappa_0^2K_0(\xi)\left(G[R]_1-R''_{y,z}[F]_1\right)+\kappa_1\left([B]_1-B_{y,z}[\mu^{-1}]_1\right),
\end{aligned}
$$

where

$$
\begin{aligned}
G &= B_{y,z}+\theta_0\zeta+\theta_1 s^{(2)}(\zeta,\xi)+\theta_2,\\
[F]_1 &= [B]_1+\theta_0[S^{(0)}]_1+\theta_1[S^{(1)}]_1+\theta_2[\mu^{-1}]_1,
\end{aligned}
$$

and accepts the transcript only if $E_C E_O = E_\Pi$, where

$$
\begin{aligned}
E_C &= e\left([P]_1,[\mu^4]_2\right)e^{-1}\left([Q]_1,[\mu^3\nu]_2\right),\\
E_O &= e\left([R]_1-R'_{y,z}[\mu^{-1}]_1,[\mu^4\kappa]_2\right)e\left([R]_1-R''_{y,z}[\mu^{-1}]_1,[\mu^4\kappa^2]_2\right),\\
E_\Pi &= e\left([\Pi_0]_1,[\mu^3\psi_0 y]_2-\zeta[\mu^3\psi_0]_2\right)e\left([\Pi_1]_1,[\mu^3\psi_1 z]_2-\xi[\mu^3\psi_1]_2\right)\\
&\cdot e\left([\Pi_2]_1,[\mu^3\psi_2 y]_2-\omega_Y^{-1}\zeta[\mu^3\psi_2]_2\right)e\left([\Pi_3]_1,[\mu^3\psi_3 z]_2-\omega_Z^{-1}\xi[\mu^3\psi_3]_2\right).
\end{aligned}
\tag{33}
$$

We can see that if $B(Y,Z)$ satisfies the copy constraint (12) and if the computations (26)-(33) are strictly followed, it holds true that $E_C E_O = E_\Pi$ with high probability: Suppose $B(Y,Z)$ satisfies the copy constraint, then by Lemma 3, $f(Y,Z,\boldsymbol{\theta})$ and $g(Y,Z,\boldsymbol{\theta})$ satisfy (16) for an indeterminant $\boldsymbol{\theta}$. However, as we sample $\boldsymbol{\theta}$ to replace the indeterminants, the equation (16) breaks if at least one factor happens to be zero, in which cases $r(Y,Z,\boldsymbol{\theta})$ cannot be defined as well. This probability is by the SZ lemma not greater than $3|\mathcal{Y}||\mathcal{Z}|/|\mathbb{F}^*|$. With supposing that $r(Y,Z,\boldsymbol{\theta})$ is well defined, Corollary 1 leads to $E_C E_O = E_\Pi$.

Efficiency of copy constraint argument. Computing each group component of the transcript is done in $O(s_{\max}(l_D-l))$ exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, where $l_D = O(s_D)$. Each polynomial evaluation of the transcript is found in $O(\log s_{\max}s_D)$ operations in $\mathbb{F}$. The prover also computes multiplication of some polynomials at the cost of $O(s_{\max}s_D\log s_{\max}s_D)$ operations in $\mathbb{F}$. The verifier evaluates $s^{(2)}(Y,Z)$ at (ζ,ξ), of which the cost is negligible, when $s^{(2)}(Y,Z)=Z$ as defined in Section 3. In Section 6, we will also consider $s^{(2)}(Y,Z)=Z^{O(s_D)}$, where the evaluation costs $O(\log s_D)$ operations in $\mathbb{F}$.

4.3 Inner product argument

We construct the *inner product argument* that connects the arithmetic and copy constraint arguments: It argues the relationship between the two transcript components $[W]_1$ and $[B]_1$, which are respectively produced by the two arguments, that both components are made of the same witness $\mathbf{b}(Y)$.

Given a reference string $\boldsymbol{\sigma}$, the argument is $\langle \texttt{Prove}_0, \texttt{Verify}\rangle$ and produces a transcript $([W]_1,[A]_1,[B]_1)$, where $[A]_1$ is computed by $\texttt{Prove}_0$ as

$$
[A]_1 = \left[\eta_0^{-1}\sum_{j=l}^{l_D-1} b_{j-l}(y)o_j(x)\left(K_{j-l}^2(z)-1\right)\right]_1-\left[\eta_0^{-1}q_8(x,y,z)t_{\mathcal{Z}}(z)\right]_1, \tag{34}
$$

with a quotient polynomial q_8 that satisfies

$$\sum_{j=l}^{l_D-1}\sum_{k=l,k\neq j}^{l_D-1} b_{j-l}(Y)K_{j-l}(Z)o_k(X)K_{k-l}(Z)=q_8(X,Y,Z)t_{\mathcal{Z}}(Z). \tag{35}$$

The verifier algorithm `Verify` accepts the transcript only if $E_I=1$, where

$$E_I=e\left([B]_1,\left[\mu^2\sum\nolimits_{j=l}^{l_D-1}o_j(x)K_{j-l}(z)\right]_2\right)e^{-1}\left([A]_1,[\mu\eta_0]_2\right)e^{-1}\left([W]_1,[\mu\eta_1]_2\right). \tag{36}$$

It is straightforward to see that if both $[W]_1,[B]_1$ are made of $\mathbf{b}(Y)$ and if the computations (34)-(36) are strictly followed, it holds true that $E_I=1$. The only thing to be careful is the calculation of q_8. Proposition 1 below shows that $[q_8(x,y,z)t_{\mathcal{Z}}(z)]_1$ can be efficiently computed from $\boldsymbol{\sigma}$.

Proposition 1. *A polynomial* $q_8(X,Y,Z)$ *that satisfies (35) can be expressed as a linear combination*

$$q_8(X,Y,Z)t_{\mathcal{Z}}(Z)=\sum_{j=l}^{l_D-1}b_{j-l}(Y)M_j(X,Z)t_{\mathcal{Z}}(Z), \tag{37}$$

where M_j *are defined in (23).*

Proof. See Appendix C

The polynomials $M_j(X,Z)$ in Proposition 1 can be seen as cached quotients [45]. They can be preprocessed by `Setup` as they remain independent of any specific statement.

Efficiency of inner-product argument. We can rewrite $l_D\le cs_D$ for a constant c, i.e., each subcircuit can have at most c input and output wires. Since there are at most $s_{\max}$ subcircuits placed in a circuit, the computation of $[A]_1$ in (34), which involves $b_{j-l}(Y)=\sum_{i=0}^{s_{\max}-1}d_j^{(i)}L_i(Y)$ (see (9)), is done in $O(s_{\max})$ exponentiations in $\mathbb{G}_1$. Also, by Proposition 1, the prover does not need to run polynomial multiplication or division when computing $q_8(X,Y,Z)$.

4.4 The final protocol as the integration of three arguments

We integrate the three arguments constructed above. There is no change in the algorithms of $\mathcal{P}$ and $\mathcal{V}$ other than `Verify` after the integration. We therefore just summarize input and output of each algorithm.

We first list the prover algorithms $\mathcal{P}=(\texttt{Prove}_0,\texttt{Commit}_1,\texttt{Prove}_2,\texttt{Eval}_3,\texttt{Prove}_4)$.

$\texttt{Prove}_0(\mathcal{L},\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{b}(Y),\mathbf{c}(Y))$ takes as input a subcircuit library $\mathcal{L}$, a reference string $\boldsymbol{\sigma}$, and a claimed pair $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)))$ of instance and witness and returns $([U]_1,[V]_2,[W]_1,[A]_1,[B]_1,[C]_1)$ as given in (24),(26),(34).

$\texttt{Commit}_1(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\mathbf{b}(Y))$ takes as input public permutation polynomials $\mathbf{s}(Y,Z)$ and challenges $\boldsymbol{\theta}\in(\mathbb{F}^*)^3$ and returns $[R]_1$ as given in (27).

$\texttt{Prove}_2(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\kappa_0,\mathbf{b}(Y))$ takes as input a challenge $\kappa_0\in\mathbb{F}^*$ and returns $[Q]_1$ as given in (28).

$\texttt{Eval}_3(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\zeta,\xi,\mathbf{b}(Y))$ takes as input challenges $\zeta,\xi\in\mathbb{F}^*$ and returns $(R'_{y,z},R''_{y,z},B_{y,z})$ as given in (29).

$\texttt{Prove}_4(\mathbf{s}(Y,Z),\boldsymbol{\sigma},\boldsymbol{\theta},\kappa_0,\zeta,\xi,\kappa_1,\mathbf{b}(Y))$ takes as input a challenge $\kappa_1\in\mathbb{F}^*$ and returns $([\Pi_0]_1,\cdots,[\Pi_3]_1)$ as given in (32).

The verifier algorithm $\texttt{Open}_i$ for each $i=0,\cdots,3$ in $\mathcal{V}$ is a probabilistic algorithm, which takes as input the specific number of group or field elements as specified in Figure 5 and returns challenges picked uniformly from $\mathbb{F}^*$ at random. We finally define $\texttt{Verify}$ as follow:

$\texttt{Verify}(pp_\lambda,\mathcal{L},[S^{(0)}]_1,[S^{(1)}]_1,s^{(2)}(Y,Z),\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{t})$ takes as input a bilinear group pp_λ, commitments to the permutation polynomials $s^{(i)}(Y,Z)$ for $i=0,1$, and a transcript $\mathbf{t}$ and returns true if and only if

$$E_A E_C E_O E_I = E_\Pi, \tag{38}$$

where each factor in (38) is computed by (25), (33), (36).

Overall efficiency of the protocol. The protocol $\texttt{IP}_{\mathcal{P},\mathcal{V}}$ consists of 10 rounds of interaction. The prover sends 11, 1, and 3 elements respectively in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{F}$. The verifier picks 7 challenges from $\mathbb{F}^*$. The prover, in total, performs $O(ns_{\max})$ and $O(s_{\max}s_D)$ exponentiations in $\mathbb{G}_1$, $O(ns_{\max})$ exponentiations in $\mathbb{G}_2$, and $O((n+s_D)s_{\max}\log ns_{\max}s_D)$ operations in $\mathbb{F}$. The verifier computes l exponentiations in $\mathbb{G}_1$ and 16 pairings. When $s^{(2)}(Y,Z)=Z^{O(s_D)}$ as defined in Section 6, the verifier additionally costs $O(\log s_D)$ operations in $\mathbb{F}$.

5 Protocol Security

5.1 Completeness and knowledge-soundness

We define security properties: completeness and knowledge-soundness. For the completeness, we follow the perfect completeness in Definition 1, with modifications for interactive protocols and allowing the statistical imperfection.

Definition 4 (Statistical completeness). *Given* $\texttt{Setup}$ *for* R*, an interactive protocol* $\texttt{IP}_{\mathcal{P},\mathcal{V}}=\langle\mathcal{P},\mathcal{V}\rangle$ *with preprocessed input* $\mathbf{z}$ *is statistically complete, if given a valid pair of instance and witness as input to* $\mathcal{P}$*, a transcript produced by the protocol is acceptable by* $\mathcal{V}$*, with high probability in* λ*. In other words, for every* $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)))\in R$*, it holds that*

$$\Pr\left[\begin{array}{l}\boldsymbol{\sigma}\leftarrow\texttt{Setup}(R);\\ \mathbf{t}\leftarrow\left\langle\mathcal{P}\left(\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{b}(Y),\mathbf{c}(Y),\mathbf{z}\right),\mathcal{V}\left(\boldsymbol{\sigma},\mathbf{a}^{(in)},\mathbf{a}^{(out)},\mathbf{z}\right)\right\rangle:\\ \mathbf{t}\ \textit{is accepted by}\ \mathcal{V}\end{array}\right]\geq 1-\epsilon(\lambda)$$

for a failure probability ϵ *negligible in* λ*.*

Proposition 2. *Given* $\texttt{Setup}$ *for* R*, the interactive protocol* $\texttt{IP}_{\mathcal{P},\mathcal{V}}=<\mathcal{P},\mathcal{V}>$ *constructed in Section 4 is statistical complete with a failure probability of*

$$\epsilon(\lambda) \le \frac{3s_{\max}(l_D - l)}{|\mathbb{F}^*|}.$$

We omit the proof of Proposition 2, as it has been already explained in Section 4. The protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}} = \langle \mathcal{P}, \mathcal{V} \rangle$ is imperfect in completeness due to the copy constraint argument. Noticeably, it has been discussed in [7] that the copy constraint argument can achieve perfect completeness at the cost of adding $\epsilon(\lambda)$ to knowledge soundness error discussed below, by forcing $\mathcal{V}$ to accept an incomplete transcript immediately whenever a bad challenge $\boldsymbol{\theta}$ has been chosen.

For the knowledge soundness, instead of direct extraction of witness in Definition 2, we utilize *witness-extended emulation*. Informally, a protocol for a relation is said to have knowledge sound if a valid pair of instance and witness that resides in the relation can be extractable from an acceptable transcript with high probability. However, the relation R we consider is a collection of polynomials with the large degree $s_{\max}$. Thus, instead of knowledge of the valid polynomials, $\mathcal{V}$ in the copy constraint argument queries $\mathcal{P}$ an evaluation of every relevant polynomial at a challenged point that cannot be predicted by $\mathcal{P}$ in advance. Since there is no efficient algorithm to extract a polynomial from a single evaluation, we need a special soundness that extracts a valid witness from two or more distinct acceptable transcripts **[2]**. In the same context, we follow the definition of witness-extended emulation **[5, 44, 46]**, which is a general framework to define a special soundness for any public coin interactive protocols.

Definition 5 (Witness-extended emulation against affine prover strategy). *Consider a public coin interactive protocol* $\mathtt{IP}_{\mathcal{P},\mathcal{V}} =< \mathcal{P}, \mathcal{V} >$ *with preprocessed input* $\mathbf{z}$, *given* $\mathtt{Setup}$ *for* R. *The protocol satisfies witness-extended emulation, if, for all deterministic polynomial time* $\mathcal{P}^*$, *there exists an expected polynomial-time emulator* $\mathcal{X}$ *such that for all generic adversaries* $\mathcal{A}$ *we have*

$$\Pr\left[\begin{array}{l}\boldsymbol{\sigma} \leftarrow \mathtt{Setup}(R); \\ \left(\mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{b}(Y), \mathbf{c}(Y)\right) \leftarrow \mathcal{A}(\boldsymbol{\sigma}, \mathbf{z}); \\ \mathbf{t} \xleftarrow{\$} \left\langle \mathcal{P}^*\left(\boldsymbol{\sigma}, \mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{b}(Y), \mathbf{c}(Y), \mathbf{z}\right), \mathcal{V}\left(\boldsymbol{\sigma}, \mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{z}\right)\right\rangle : \\ \mathcal{A}\left(\mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{t}, \mathbf{z}\right) = 1\end{array}\right] \approx$$

$$\Pr\left[\begin{array}{l}\boldsymbol{\sigma} \xleftarrow{\$} \mathtt{Setup}(\mathbf{b}, \mathcal{Q}_A); \\ \left((\mathbf{a}^{(in)}, \mathbf{a}^{(out)}), \mathbf{b}(Y), \mathbf{c}(Y)\right) \leftarrow \mathcal{A}(\boldsymbol{\sigma}, [s_Y]_1, [s_Z]_1); \\ \left(\mathbf{t}, \mathbf{b}(Y), \mathbf{c}(Y)\right) \xleftarrow{\$} \mathcal{X}^{\left\langle \mathcal{P}^*\left(\boldsymbol{\sigma}, \mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{b}(Y), \mathbf{c}(Y), \mathbf{z}\right), \mathcal{V}\left(\boldsymbol{\sigma}, \mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{z}\right)\right\rangle} : \\ \mathcal{A}\left(\mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{t}, \mathbf{z}\right) = 1 \wedge \left(\begin{array}{l}\mathbf{t} \textit{ is acceptable} \Rightarrow \\ \left((\mathbf{a}^{(in)}, \mathbf{a}^{(out)}), (\mathbf{b}(Y), \mathbf{c}(Y))\right) \in R\end{array}\right)\end{array}\right],$$

where $\mathcal{X}$ *has access to repeatedly rewind* $< \mathcal{P}^*, \mathcal{V} >$ *to a particular round for fresh randomness of* $\mathcal{V}$ *and produce the corresponding transcript.*

Proposition 3 (Witness-extended emulation). *Given* $\mathtt{Setup}$ *for* R, *the interactive protocol* $\mathtt{IP}_{\mathcal{P},\mathcal{V}} =< \mathcal{P}, \mathcal{V} >$ *constructed in Section 4 satisfies witness-extended emulation with an emulator that extracts* $((\mathbf{a}^{(in)}, \mathbf{a}^{(out)}), (\mathbf{b}(Y), \mathbf{c}(Y))) \in R$ *from an acceptable transcript* $\mathbf{t} \leftarrow \mathtt{IP}_{\mathcal{A},\mathcal{V}}$ *expectedly in* $(s_{\max} + 2)(l_D - l)$ *runs with probability bounded below by*

$$\frac{s_{\max}(l_D-l)\left(\deg_{\boldsymbol{\sigma}}(\boldsymbol{\tau})+3\right)+5s_{\max}+(l_D-l)\left(2\deg_{\boldsymbol{\sigma}}(\boldsymbol{\tau})+8\right)+2}{\left|\mathbb{F}^*\right|}.$$

Proof. See Appendix D.

We consider GGM, which is a stronger assumption than algebraic group model (AGM). Unlike generic adversaries, an algebraic adversary has direct access to the group operations. We do not provide AGM analysis for the simplicity of proof, but interested readers are referred to [47, 48]. In [47], it has been shown that Groth16, which is the original version of our arithmetic constraint argument, satisfies the knowledge soundness against computationally bounded algebraic adversaries. Also, under AGM, a general framework for online witness-extended emulation, where the emulator does not rewind the protocol, for public-coin interactive protocols has been introduced in [48].

5.2 Adding Zero-knowledge

The protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ in Section 4 does not provide zero-knowledge (e.g., Definition 3), as all algorithms of $\mathcal{P}$ are deterministic. In this section, we modify each argument of $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ to add zero-knowledge.

Perfect zero-knowledge for the arithmetic constraint and inner product arguments. Consider a non-interactive argument $(\mathtt{Setup},\mathtt{Prove}^{\star},\mathtt{Verify},\mathtt{Sim})$ and the perfect zero-knowledge in Definition 3. We collect and modify the algorithms of the arithmetic constraint and inner product arguments to form the argument that satisfies the perfect zero-knowledge.

We first construct $\mathtt{Sim}$ as follow,

$\mathtt{Sim}\left(\mathcal{L}, pp_{\lambda}, \boldsymbol{\tau}, \mathbf{a}^{(in)}, \mathbf{a}^{(out)}\right) \mapsto \pi^{\star}$: It picks $U^{\star}, V^{\star}, C^{\star}, B^{\star}$ uniformly from $\mathbb{F}$ at random, computes

$$W^{\star}=\eta_1^{-1}\left(U^{\star}V^{\star}-\alpha\beta-C^{\star}\delta-\sum_{j=0}^{l_{in}-1}a_j^{(in)}L_0(y)o_j(x)-\sum_{j=l_{in}}^{l-1}a_{j-l_{in}}^{(out)}L_{-1}(y)o_j(x)\right),$$
$$A^{\star}=\eta_0^{-1}\left(B^{\star}\mu\sum_{j=l}^{l_D-1}o_j(x)K_{j-l}(z)-W^{\star}\eta_1\right),$$

and returns a simulated proof $\pi^{\star}=([U^{\star}]_1,[V^{\star}]_2,[W^{\star}]_1,[A^{\star}]_1,[B^{\star}]_1,[C^{\star}]_1)$.

We construct $\mathtt{Prove}^{\star}$ as a modification of $\mathtt{Prove}_0$. Unlike $\mathtt{Prove}_0$, $\mathtt{Prove}^{\star}$ is a PPT algorithm:

- Pick random mixers r_0, r_1, r_2, r_3 from $\mathbb{F}$.
- Let $\tilde{b}_i(Y)=b_i(Y)+r_3t_{\mathcal{Y}}(Y)$ for $i=0,\cdots,l_D-l-1$, and $\tilde{B}(Y,Z)=\sum_{i=l}^{l_D-1}\tilde{b}_{i-l}(Y)K_{i-l}(Z)$.
- Replace $b_{i-l}(Y)$ in (37) with $\tilde{b}_{i-l}(Y)$ to compute the corresponding quotient polynomial $\tilde{q}_8(X,Y,Z)$.
- Modify (24), (26), and (34) as

$$[\tilde{U}]_1=[U]_1+r_0[\eta_1]_1,$$
$$[\tilde{V}]_2=[V]_2+r_1[\eta_1]_2,$$
$$[\tilde{W}]_1=[W]_1+r_3\left[\eta_1^{-1}t_{\mathcal{Y}}(y)\sum_{i=l}^{l_D-1}o_i(x)\right]_1+r_1[U]_1+r_0[V]_1+r_0r_1[\eta_1]_1-r_2[\delta]_1,$$

$$[\tilde{A}]_1 = [A]_1 + r_3\left[\eta_0^{-1}t_{\mathcal{Y}}(y)\sum_{i=l}^{l_D-1}o_i(x)\left(K_{i-l}^2(z)-1\right)\right]_1 - r_3\left[\eta_0^{-1}t_{\mathcal{Y}}(y)\sum_{i=l}^{l_D-1}M_i(x,z)t_{\mathcal{Z}}(z)\right]_1,$$
$$[\tilde{B}]_1 = [B]_1 + r_3\left[\mu^{-1}t_{\mathcal{Y}}(y)\right]_1,$$
$$[\tilde{C}]_1 = [C]_1 + r_2[\eta_1]_1.$$

Finally, `Verify` accepts the proof $\pi = ([\tilde{U}]_1, [\tilde{V}]_2, [\tilde{W}]_1, [\tilde{A}]_1, [\tilde{B}]_1, [\tilde{C}]_1)$ if and only if $E_A E_I = 1$, where E_A and E_I are computed by (25) and (36), respectively.

By the modification $\texttt{Prove}^\star$, the distribution of π becomes identical with that of the simulated proof $\pi^\star$. Also, the argument satisfies the perfect completeness in Definition 1: For the arithmetic constraint argument, $[C]_1$ compensates all aliasing terms in the verification equation incurred by the random mixers; and for the inner product argument, the verifier accepts the proof as long as $[W]_1$, $[A]_1$, and $[B]_1$ are made of the same $\tilde{\mathbf{b}}(Y)$.

Honest-verifier zero-knowledge for the copy constraint argument. It would be challenging for an interactive protocol to have the perfect zero-knowledge in Definition 3. Instead, we can consider honest-verifier zero-knowledge [49]. Informally, an interactive protocol that produces a transcript $\mathbf{t}$ is said to have honest-verifier (statistical) zero-knowledge, if there exists a simulator that produces a simulated transcript $\mathbf{t}^\star$ such that the distributions of $\mathbf{t}$ and $\mathbf{t}^\star$ are identical (or statistically indistinguishable) given that the randomized verifier strictly follows the protocol. In [5, 7], the authors have exemplified adding honest-verifier zero-knowledge to an interactive protocol. We can apply a similar approach with them to our copy constraint argument.

We start by constructing a simulator that accesses the coin-tossing of $\mathcal{V}$ as follow,

$\texttt{Sim}_{\mathcal{V}}\left(pp_\lambda, \boldsymbol{\tau}, \mathbf{s}(Y,Z)\right) \mapsto \mathbf{t}^\star$: It picks $B^\star, R^\star, Q^\star, R'^\star_{y,z}, R''^\star_{y,z}, B^\star_{y,z}, \pi_1^\star, \pi_3^\star, \pi_5^\star$ uniformly from $\mathbb{F}$ at random, takes random challenges $\boldsymbol{\theta}, \kappa_0, \zeta, \xi, \kappa_1 \in \mathbb{F}$ from $\mathcal{V}$, computes

$$\begin{aligned}
P^\star &= L_{-1}(\zeta)K_{-1}(\xi)\left(R^\star - \mu^{-1}\right) + \kappa_1\left(B^\star - B^\star_{y,z}\mu^{-1}\right)\\
&\quad + \kappa_0\left(\xi - 1\right)\left(\left(B^\star_{y,z} + \theta_0\zeta + \theta_1 s^{(2)}(\zeta,\xi) + \theta_2\right)R^\star - R'^\star_{y,z}\left(B^\star + \theta_0 s^{(0)}(y,z) + \theta_1 s^{(1)}(y,z) + \theta_2\right)\right)\\
&\quad + \kappa_0^2 K_0(\xi)\left(\left(B^\star_{y,z} + \theta_0\zeta + \theta_1 s^{(2)}(\zeta,\xi) + \theta_2\right)R^\star - R''^\star_{y,z}\left(B^\star + \theta_0 s^{(0)}(y,z) + \theta_1 s^{(1)}(y,z) + \theta_2\right)\right),
\end{aligned}$$

$$\begin{aligned}
\pi_0^\star &= \left(P^\star\mu^4 - Q^\star\mu^3\nu - \pi_1^\star\left(z - \xi\right)\right)\left(y - \zeta\right)^{-1},\\
\pi_2^\star &= \left(\mu^4\left(R^\star - R'^\star_{y,z}\mu^{-1}\right) - \pi_3^\star\left(z - \omega_Z^{-1}\xi\right)\right)\left(y - \zeta\right)^{-1},\\
\pi_4^\star &= \left(\mu^4\left(R^\star - R''^\star_{y,z}\mu^{-1}\right) - \pi_5^\star\left(z - \omega_Z^{-1}\xi\right)\right)\left(y - \omega_Y^{-1}\zeta\right)^{-1},
\end{aligned}$$

and returns $\mathbf{t}^\star = ([B^\star]_1, \boldsymbol{\theta}, [R^\star]_1, \kappa_0, [Q^\star]_1, \zeta, \xi, R'^\star_{y,z}, R''^\star_{y,z}, B^\star_{y,z}, \kappa_1, [\Pi_0^\star]_1, [\Pi_1^\star]_1, [\Pi_2^\star]_1, [\Pi_3^\star]_1)$, where

$$\begin{aligned}
\Pi_0^\star &= \mu^{-3}\psi_0^{-1}\left(\pi_0^\star + \kappa\pi_2^\star\right),\\
\Pi_1^\star &= \mu^{-3}\psi_1^{-1}\pi_1^\star,\\
\Pi_2^\star &= \mu^{-3}\psi_2^{-1}\kappa^2\pi_4^\star,\\
\Pi_3^\star &= \mu^{-3}\psi_3^{-1}\left(\kappa\pi_3^\star + \kappa^2\pi_5^\star\right).
\end{aligned}$$

We now construct $\mathcal{P}^\star$ by modifying $\mathcal{P}$ of $\texttt{IP}_{\mathcal{P},\mathcal{V}}$ as follows:

- Pick random mixers r_3, r_4, r_5 from $\mathbb{F}$.
- Let $\tilde{B}(Y,Z) = B(Y,Z) + r_3 t_{\mathcal{Y}}(Y)$ and $\tilde{r}(Y,Z,\boldsymbol{\theta}) = r(Y,Z,\boldsymbol{\theta}) + (r_4 + r_5 Y) t_{\mathcal{Y}}(Y)$.
- Modify (27) and (29) as

$$
\begin{aligned}
[\tilde{R}]_1 &= [R]_1 + [\mu^{-1}(r_4 + r_5 y) t_{\mathcal{Y}}(y)]_1, \\
\tilde{B}_{y,z} &= B_{y,z} + r_3 t_{\mathcal{Y}}(\zeta), \\
\tilde{R}'_{y,z} &= R'_{y,z} + (r_4 + r_5 \zeta) t_{\mathcal{Y}}(\zeta), \\
\tilde{R}''_{y,z} &= R''_{y,z} + (r_4 + r_5 \omega_Y^{-1} \zeta) t_{\mathcal{Y}}(\zeta).
\end{aligned}
$$

- Replace $B(Y)$ with $\tilde{B}(Y)$ in (15) and (19) to compute $(\tilde{f}(Y,Z,\boldsymbol{\theta}), \tilde{g}(Y,Z,\boldsymbol{\theta}))$ and $(\tilde{p}_1(Y,Z,\boldsymbol{\theta}), \cdots, \tilde{p}_3(Y,Z,\boldsymbol{\theta}))$, respectively.
- To compute $[\tilde{Q}]_1$, modify (28) as

$$
\begin{aligned}
\tilde{q}_2(y,z,\boldsymbol{\theta}) &= q_2(y,z,\boldsymbol{\theta}) + (r_4 + r_5 y) L_{-1}(y) K_{-1}(z), \\
\tilde{q}_4(y,z,\boldsymbol{\theta}) &= q_4(y,z,\boldsymbol{\theta}) + (z-1) \begin{pmatrix} r_3 r(y,z,\boldsymbol{\theta}) + (r_4 + r_5 y) g(y,z,\boldsymbol{\theta}) \\ -r_3 r(y, \omega_Z^{-1} z, \boldsymbol{\theta}) - (r_4 + r_5 y) f(y,z,\boldsymbol{\theta}) \end{pmatrix}, \\
\tilde{q}_6(y,z,\boldsymbol{\theta}) &= q_6(y,z,\boldsymbol{\theta}) + K_0(z) \begin{pmatrix} r_3 r(y,z,\boldsymbol{\theta}) + (r_4 + r_5 y) g(y,z,\boldsymbol{\theta}) + (1 - \omega_Y^{-1}) r_3 r_5 y t_{\mathcal{Y}}(y) \\ -r_3 r(\omega_Y^{-1} y, \omega_Z^{-1} z, \boldsymbol{\theta}) - (r_4 + r_5 \omega_Y^{-1} y) f(y,z,\boldsymbol{\theta}) \end{pmatrix}.
\end{aligned}
$$

- Compute $\tilde{P}(Y,Z,\boldsymbol{\theta})$ and the corresponding quotient polynomials $\tilde{\pi}_0, \cdots, \tilde{\pi}_5$, according to (30) and (31) with $\tilde{r}(Y,Z,\boldsymbol{\theta})$, $\tilde{f}(Y,Z,\boldsymbol{\theta})$, and $\tilde{g}(Y,Z,\boldsymbol{\theta})$.
- Replace $\pi_0, \cdots, \pi_5$ with $\tilde{\pi}_0, \cdots, \tilde{\pi}_5$ in (32) to compute $[\tilde{\Pi}_0]_1, \cdots, [\tilde{\Pi}_3]_1$.

Finally, our verification algorithm accepts the transcript,

$$
\mathbf{t} = \left([\tilde{B}]_1, \boldsymbol{\theta}, [\tilde{R}]_1, \kappa_0, [\tilde{Q}]_1, \zeta, \xi, \tilde{R}'_{y,z}, \tilde{R}''_{y,z}, \tilde{B}_{y,z}, \kappa_1, [\tilde{\Pi}_0]_1, [\tilde{\Pi}_1]_1, [\tilde{\Pi}_2]_1, [\tilde{\Pi}_3]_1\right),
$$

if and only if $E_C E_O = E_\Pi$, where E_C, E_O, and E_Π are computed according to (33).

The statistical completeness of $\langle \mathcal{P}^\star, \mathcal{V} \rangle$ is still satisfied: If $B(Y,Z)$ satisfies the copy constraints of Lemma 3, so does $\tilde{B}(Y,Z)$ (evaluations of the two polynomials on $\mathcal{Y}$ are consistent); And if $r(Y,Z,\boldsymbol{\theta})$ is a well-defined recursion polynomial, $\tilde{P}(Y,Z,\boldsymbol{\theta})$ vanishes on $\mathcal{Y} \times \mathcal{Z}$, as the additional mixing terms in both $\tilde{B}(Y,Z)$ and $\tilde{r}(Y,Z,\boldsymbol{\theta})$ always vanish on $\mathcal{Y}$.

The simulated transcript $\mathbf{t}^\star$ is drawn from 9 independent coin tosses, whereas the real transcript $\mathbf{t}$ involved only 3 independent coin tosses (r_3, r_4, r_5). Fortunately, if the verifier's challenges are picked from public coin tosses, their combination with the prover's coin tosses brings the effect of additional independent (not necessarily identical) coin tosses. We thus conjecture that given honest verifier, a simulated transcript $\mathbf{t}^\star$ and a real transcript $\mathbf{t}$ would be statistically indistinguishable.

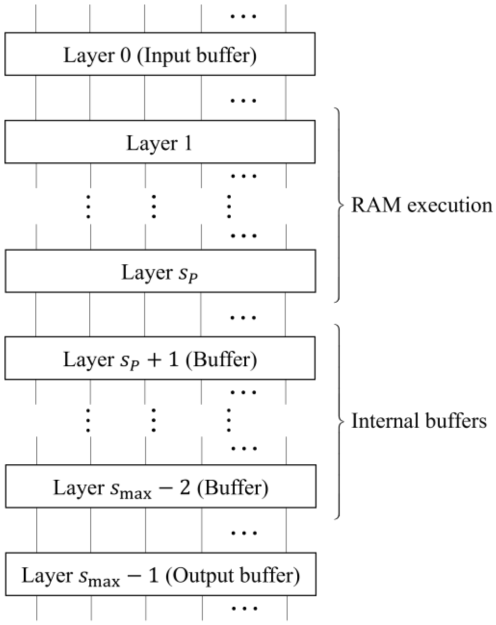


Figure 6. RAM circuit illustration: This circuit comprises $s_{\max}$ layers. The initial and final layers are dedicated to the input and output buffers, respectively. Among the $s_{\max} - 2$ intermediate layers, s_P layers handle RAM execution, with each activation subcircuit is dynamically determined by the input. The remaining layers are activated by internal buffers.

6 Elimination of the verifier preprocess with machine computation

The protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ for R in Section 4 still requires the verifier $\mathcal{V}$ to preprocess wiring of a circuit, represented by $s^{(0)}, s^{(1)}, s^{(2)} \in \mathbb{F}_{s_{\max}, l_D - l}[Y, Z]$. In this section, motivated by verifiable machine computations in [26–28], we eliminate the dependency on the verifier preprocess.

Machine model. We define a subcircuit library $\mathcal{L}$ that is specific to a random-access machine (RAM). Given the number of instructions, s_D, the subcircuit library is defined as $\mathcal{L} := \bigcup_{k=0}^{s_D+2} \mathtt{c}_k$. Each subcircuit $\mathtt{c}_k$ has l_K input wires and l_K outputs wires, where K denotes a length of variables long enough to represent the RAM states.

Out of the $s_D + 3$ subcircuits, subcircuits $\mathtt{c}_0, \mathtt{c}_{s_D+1}, \mathtt{c}_{s_D+2}$ are buffers, which are specialized to pass data to or retrieve data from other circuit components. Specifically, $\mathtt{c}_0$ is an input buffer to transfer the initial machine state $\mathbf{x}$ to other subcircuits, $\mathtt{c}_{s_D+2}$ is an output buffer to return the resulting machine state $\mathbf{y}$, and $\mathtt{c}_{s_D+1}$ is an internal buffer to transfer an intermediate machine state from one subcircuit to another subcircuit.

The rest subcircuits, $\mathtt{c}_i$ for $i = 1, \cdots, s_D$, take as input a machine state and execute the i-th instruction of $\mathcal{M}$. In addition to simply executing each instruction, the subcircuit $\mathtt{c}_i$ for each $i = 1, \cdots, s_D$ checks *1)* whether an input instruction to a subcircuit matches with the subcircuit index, *2)* whether the next program counter is correctly computed, and *3)* whether the next instruction is correctly retrieved from P according to the next program counter.

As a result, a program P with an initial state $\mathbf{x}$, which returns the resulting state $\mathbf{y}$ in s_P machine steps, can be validated by a circuit $\mathcal{C}$ of $s_P + 2$ layers, each comprising $s_D + 3$ subcircuit branches (the additional two layers are for the input and output buffers). However, instead of s_P, we use a constant $s_{\max}$ such that $s_P + 2 \le s_{\max}$ to

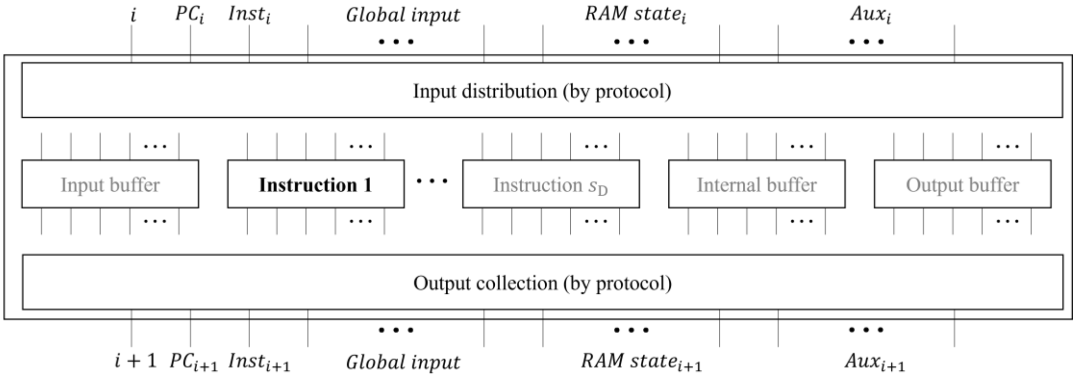


Figure 7. Layer composition example: Each layer comprises $s_D + 3$ subcircuits, with only one activated based on the layer's input. The layer input is distributed to the active subcircuit, and the subcircuit output is collected and returned as the layer output. The input distribution and output collection blocks are driven the back-end protocol rather than implemented subcircuits.

keep the setup algorithm for $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ independent of P. Also, we have structured the setup to ensure that the I/O buffers are consistently placed in the first and last layers, respectively (see $\boldsymbol{\sigma}$ in Section 3.3, where $l_{in} = l_{out} = l_K$ in this model). Except for the I/O buffers, in $\mathbf{Q}$, there are s_P layers for the program execution, and the remaining $s_{\max} - s_P - 2$ layers are filled with the internal buffers $\mathtt{c}_{s_D+1}$ (see Figure 6).

By the constraints in $\mathcal{L}$, only one subcircuit in each layer can be activated (see Figure 7). However, in the view of verifier, the activation of intermediate layers remains nondeterministic. A straightforward effort to resolve this is to make up a large deterministic circuit by injecting multiplexer (MUX) components in each layer that selects one of the outputs from the $s_D + 3$ branches [31], which results in the prover time complexity $O(s_D s_{\max} n)$. Instead, it has been shown that a back-end protocol can play the role of MUX [27, 28]. We will show that our protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ also replaces the MUX, which results in the prover complexity $O((s_D + s_{\max})n)$.

QAP compiler. We rewrite the subcircuit library $\mathcal{L}$ in the QAP representation. Recall $z_j^{(k)} \in \{u_j^{(k)}, v_j^{(k)}, w_j^{(k)}\}$ for $j \in \{0, \cdots, l^{(k)}, \cdots, m^{(k)}\}$ represents the j-th wire polynomials of a subcircuit $\mathtt{c}_k \subset \mathcal{L}$. Since all subcircuits $\mathtt{c}_k$ have l_K input wires and l_K output wires, we can fix the number of connecting wires in the subcircuit $\mathtt{c}_k$, $l^{(k)} = 2l_K$ for all k. As $\mathcal{L}$ is a union of $\mathtt{c}_k$ for all k. Thus, the number of wires in $\mathcal{L}$ is parameterized by $l = 2l_K$, $l_D = 2l_K(s_D + 3)$, and $m_D = \sum_{k=0}^{s_D+2} m^{(k)}$. In addition to the definitions in Section 3.1, we use an index set of *connecting position*, $\mathcal{I} := \{0, \cdots, 2l_K - 1\}$, and disjoint subsets of it, $\mathcal{I}_{in} := \{0, \cdots, l_K - 1\}$ and $\mathcal{I}_{out} := \{l_K, \cdots, 2l_K - 1\}$.

We write the elements of $\mathcal{L}$ as

$$\mathcal{L} = \{o_j\}_{j=0}^{m_D - 1},$$

where $o_j \in \{u_j, v_j, w_j\} \subset \mathbb{F}[X]$. Each element o_j is picked from $\mathtt{c}_k$ according to the position indices as follows:

$$\begin{aligned}
(o_j)_{j \in \{0, \cdots, l-1\}} &= \left((z_j^{(0)})_{j \in \mathcal{I}_{in}}, (z_j^{(s_D+2)})_{j \in \mathcal{I}_{out}} \right), \\
(o_j)_{j \in \{l, \cdots, l_D - 1\}} &= \left((z_j^{(s_D+2)})_{j \in \mathcal{I}_{in}}, (z_j^{(0)})_{j \in \mathcal{I}_{out}}, (z_j^{(1)})_{j \in \mathcal{I}}, \cdots, (z_j^{(s_D+1)})_{j \in \mathcal{I}} \right), \\
(o_j)_{j \in \{l_D, \cdots, m_D - 1\}} &= \left(z_{l^{(0)}}^{(0)}, \cdots, z_{m^{(0)}}^{(0)}, z_{l^{(1)}}^{(1)}, \cdots, z_{m^{(1)}}^{(1)}, \cdots, z_{l^{(s_D+2)}}^{(s_D+2)}, \cdots, z_{m^{(s_D+2)}}^{(s_D+2)} \right).
\end{aligned}$$

For the connecting wires, represented by o_j for $j \in \{l, \cdots, l_D - 1\}$, we say two wires o_{j_1} and o_{j_2} are in the same *position*, if $j_1 \equiv j_2 \pmod{l_K}$. We put a restriction on the wiring of a circuit such that only two connecting wires at the same position can be connected to each other, as shown in Figure 6.

Recall $\mathcal{C} = (\mathbf{Q}, \rho)$ with $\mathbf{Q} = (\mathtt{d}^{(0)}, \mathtt{d}^{(1)}, \cdots, \mathtt{d}^{(s_{\max}-1)})$, where $\mathtt{d}^{(i)} \in \{\mathtt{c}_0, \cdots, \mathtt{c}_{s_D+2}\}$ indicates the *active subcircuit* of the i-th layer. By the virtue of the enhanced constraints in the subcircuits, unlike the constraint system in Section 3, the activation is determined by the input to each layer. We again set the wire assignments in all inactive subcircuits to zero.

Synthesizer. When defining wiring of $\mathcal{C}$, the data transfer occurs only between two wires on the same position as shown in Figure 6. We let "$\mathtt{d}^{(g)} \to \mathtt{d}^{(h)}$" for $g \neq h$ denote the data transfer from the g-th layer to the h-th layer. Let $k^{(i)} \in \{1, \cdots, s_D + 2\}$ be the index of active subcircuit in the i-th layer so that $d^{(i)}_{2k^{(i)}l_K + i}$ for $i \in \mathcal{I}$ indicate the assignments to the connecting wires of an active subcircuit. Then, $\mathtt{d}^{(g)} \to \mathtt{d}^{(h)}$ is formally defined as copy constraints between $d^{(g)}_{2k^{(g)}l_K + i_{out}} = d^{(h)}_{2k^{(h)}l_K + i_{in}}$ for all index pairs $(i_{out}, i_{in}) \in \mathcal{I}_{out} \times \mathcal{I}_{in}$ such that $i_{out} \equiv i_{in} \pmod{l_K}$. We also allow multiple compositions of "$\to$", e.g., the wiring of $\mathcal{C}$ in Figure 6 is defined as

$$\mathtt{d}^{(0)} \to \mathtt{d}^{(1)} \to \cdots \to \mathtt{d}^{(s_{\max}-1)} \Leftrightarrow$$
$$(\mathtt{d}^{(0)} \to \mathtt{d}^{(1)}) \wedge (\mathtt{d}^{(1)} \to \mathtt{d}^{(2)}) \wedge \cdots \wedge (\mathtt{d}^{(s_{\max}-2)} \to \mathtt{d}^{(s_{\max}-1)}).$$

Copy constraints. To make the copy constraints $\mathtt{d}^{(g)} \to \mathtt{d}^{(h)}$ deterministic, the following Corollary 2 modifies Lemma 3.

Corollary 2. Assume the zero wire assignments for inactive subcircuits. Given the generators ω_Y and ω_Z of the vanishing sets $\mathcal{Y}$ and $\mathcal{Z}$, define $\omega := \omega_Z^{s_D+2}$ so that $\omega^{2l_K} = 1$. The copy constraints $\mathtt{d}^{(g)} \to \mathtt{d}^{(h)}$ holds if and only if the following equation holds:

$$P_{g\to h}(\boldsymbol{\theta}) = 1, \tag{39}$$

where $P_{g\to h}$ is a Laurent polynomial of indeterminates $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$ in $\mathbb{F}$ defined as

$$P_{g\to h}(\boldsymbol{\theta}) := \frac{\prod_{k=1}^{s_D+2}\left(\prod_{j\in\mathcal{I}_{out}}\left(d^{(g)}_{2kl_K+j} + \omega_Y^h\theta_0 + \omega^{j-l_K}\theta_1 + \theta_2\right)\prod_{j\in\mathcal{I}_{in}}\left(d^{(h)}_{2kl_K+j} + \omega_Y^g\theta_0 + \omega^{j+l_K}\theta_1 + \theta_2\right)\right)}{\prod_{k=1}^{s_D+2}\left(\prod_{j\in\mathcal{I}_{out}}\left(d^{(g)}_{2kl_K+j} + \omega_Y^g\theta_0 + \omega^{j}\theta_1 + \theta_2\right)\prod_{j\in\mathcal{I}_{in}}\left(d^{(h)}_{2kl_K+j} + \omega_Y^h\theta_0 + \omega^{j}\theta_1 + \theta_2\right)\right)}. \tag{40}$$

Proof. Due to the zero wire assignments for inactive subcircuits, equation (39) can be reduced to

$$\begin{aligned}&\prod_{j\in\mathcal{I}_{out}}\left(d^{(g)}_{2k^{(g)}l_K+j} + \omega_Y^h\theta_0 + \omega^{j-l_K}\theta_1 + \theta_2\right)\prod_{j\in\mathcal{I}_{in}}\left(d^{(h)}_{2k^{(h)}l_K+j} + \omega_Y^g\theta_0 + \omega^{j+l_K}\theta_1 + \theta_2\right) = \\ &\prod_{j\in\mathcal{I}_{out}}\left(d^{(g)}_{2k^{(g)}l_K+j} + \omega_Y^g\theta_0 + \omega^{j}\theta_1 + \theta_2\right)\prod_{j\in\mathcal{I}_{in}}\left(d^{(h)}_{2k^{(h)}l_K+j} + \omega_Y^h\theta_0 + \omega^{j}\theta_1 + \theta_2\right)\end{aligned} \tag{41}$$

By the definition, it is straightforward to see that if $\mathtt{d}^{(g)} \to \mathtt{d}^{(h)}$, equation (41) holds. To see the converse, suppose (41) holds. The sets of roots of θ_0 on both sides are given by, respectively,

$$\left\{-\frac{b^{(g)}_{2k^{(g)}l_K+j}+\omega_Z^{j-l_K}\theta_1+\theta_2}{\omega_Y^h}\right\}_{j\in\mathcal{I}_{out}}\cup\left\{-\frac{b^{(h)}_{2k^{(h)}l_K+j}+\omega_Z^{j+l_K}\theta_1+\theta_2}{\omega_Y^g}\right\}_{j\in\mathcal{I}_{in}},$$

$$\left\{-\frac{b^{(h)}_{2k^{(h)}l_K+j}+\omega_Z^{j}\theta_1+\theta_2}{\omega_Y^h}\right\}_{j\in\mathcal{I}_{in}}\cup\left\{-\frac{b^{(g)}_{2k^{(g)}l_K+j}+\omega_Z^{j}\theta_1+\theta_2}{\omega_Y^g}\right\}_{j\in\mathcal{I}_{out}}.$$

Since $\{j+l_K,\forall j\in\mathcal{I}_{in}\}=\mathcal{I}_{out}$, equating the two sets of roots implies $\mathtt{d}^{(g)}\to\mathtt{d}^{(h)}$. ☐

By extending Corollary 2, we can define a sufficient and necessary condition for $\mathtt{d}^{(0)}\to\mathtt{d}^{(1)}\to\cdots\to\mathtt{d}^{(s_{\max}-1)}$ as

$$\prod_{h=0}^{s_{\max}-2}P_{h\to h+1}(\boldsymbol{\theta})=1. \tag{42}$$

Permutation polynomials. Our protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ is useful to argue (42) without further modification. All we need to do is replacing the permutation polynomials $s^{(0)},s^{(1)},s^{(2)}\in\mathbb{F}_{s_{\max},l_D-l}[Y,Z]$ for the construction of a recursion polynomial $r\in\mathbb{F}[Y,Z,\boldsymbol{\theta}]$ in (17) with the following new definitions:

$$\begin{aligned}
s^{(0)}\left(\omega_Y^h,\omega_Z^i\right)&:=\begin{cases}\omega_Y^{h-1}, & \text{if } (h, i \bmod 2l_K)\in\{1,\cdots,s_{\max}-1\}\times\mathcal{I}_{in},\\ \omega_Y^{h+1}, & \text{if } (h, i \bmod 2l_K)\in\{0,\cdots,s_{\max}-2\}\times\mathcal{I}_{out},\\ \omega_Y^h, & \textit{otherwise.}\end{cases}\\
s^{(1)}\left(\omega_Y^h,\omega_Z^i\right)&:=\begin{cases}\omega^{i+l_K}, & \text{if } (h, i \bmod 2l_K)\in\{1,\cdots,s_{\max}-1\}\times\mathcal{I}_{in},\\ \omega^{i-l_K}, & \text{if } (h, i \bmod 2l_K)\in\{0,\cdots,s_{\max}-2\}\times\mathcal{I}_{out},\\ \omega^i, & \textit{otherwise.}\end{cases}\\
s^{(2)}\left(\omega_Y^h,\omega_Z^i\right)&:=\omega^i \Leftrightarrow s^{(2)}(Y,Z)=Z^{s_D+2},
\end{aligned} \tag{43}$$

where $\omega:=\omega_Z^{s_D+2}$. Then, given $B(Y,Z)$ as an encoding of $d_j^{(i)}$ (see (13)), the copy constraints in (42) holds if and only if

$$\prod_{h=0}^{s_{\max}-2}P_{h\to h+1}(\boldsymbol{\theta})=\frac{\prod_{y\in\mathcal{Y},z\in\mathcal{Z}}B(y,z)+s^{(0)}(y,z)\theta_0+s^{(1)}(y,z)\theta_1+\theta_2}{\prod_{y\in\mathcal{Y},z\in\mathcal{Z}}B(y,z)+y\theta_0+s^{(2)}(y,z)\theta_1+\theta_2}=1.$$

It is clear to see that the equality of this equation can be argued by the copy constraint argument of $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$ along with the newly constructed recursion polynomial r.

Efficiency of machine computation. The permutation polynomials $s^{(0)}(Y,Z)$ and $s^{(1)}(Y,Z)$ in this verifiable machine computation model are independent of programs and the input instance but only parameterized by the maximum number of machine steps, $s_{\max}$. Thus, in the protocol $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$, appending commitments to the permutation polynomials into the reference string $\boldsymbol{\sigma}$ resolves the reliance of the verifier preprocessing. This approach only adds the verifier time complexity $O(\log s_D)$ for the evaluation of $s^{(2)}(Y,Z)$. The rest factors such as the prover time complexity and the proof size directly inherit those of $\mathtt{IP}_{\mathcal{P},\mathcal{V}}$.

7 Conclusion

In this paper, we have proposed a SNARK with universal setups. We showed that our SNARK satisfies completeness and knowledge soundness, and it can be further enhanced with zero-knowledge. In our SNARK, the prover the complexity is $O((s_D + s_{\max})n \log s_D s_{\max} n)$, with proof size of 12 group elements and 5 field elements, and the verifier time complexity is $O(1)$. Here, s_D and $s_{\max}$ can be considered complementary to each other [26], and in extreme cases where $s_D = O(1)$ with maximized $s_{\max}$, the efficiency of our SNARK asymptotically comparable to other state-of-the-art SNARKs with updatable and universal setups in [6, 7]. Additionally, compared to the other SNARKs, we have reduced the dimensionality of verifier preprocessing data from $O(ns_{\max})$ to $O(s_{\max})$, albeit at the cost of sacrificing updatability of the setup.

Furthermore, we have demonstrated the applicability of our SNARK in verifiable machine computation. Unlike recent machine computation protocols such as MUX-Marlin [27] and SublonK [28], our approach achieves machine computation by merely changing the verifier preprocessed inputs to deterministic ones, without requiring additional auxiliary protocols. This results in proof sizes four to ten times smaller. However, while the CRS sizes of the other two protocols are $O((s_D + s_{\max} \log s_D)n)$, offering an advantage over our CRS size of $O(s_D s_{\max} n)$.

Our SNARK can effectively support efficient verifiable computation in distributed computing networks, particularly in environments like blockchains where nodes are generally untrusted. In such networks, the preprocessing generated by one verifier node must be verified by others, consuming network resources over time as more preprocesses accumulate. Future research could investigate how the reduced data dimensionality in verifier preprocessing could alleviate this burden, thus improving network efficiency.

References

1. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. pp. 465–482. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25.

2. Thaler, J.: Proofs, Arguments, and Zero-Knowledge. Found. Trends® Priv. Secur. 4, 117–660 (2022). https://doi.org/10.1561/3300000030.

3. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE, Berkeley, CA (2013). https://doi.org/10.1109/SP.2013.47.

4. Groth, J.: On the Size of Pairing-Based Non-interactive Arguments. In: Fischlin, M. and Coron, J.-S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11.

5. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2111–2128. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3339817.

6. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In: Canteaut, A. and Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 738–768. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26.

7. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge, https://eprint.iacr.org/2019/953, (2019).

8. Lipmaa, H., Siim, J., Zajac, M.: Counting Vampires: From Univariate Sumcheck to Updatable ZK-SNARK, https://eprint.iacr.org/2022/406, (2022).

9. Jang, J., Lee, H.-N.: Profitable Double-Spending Attacks. Appl. Sci. 10, 8477 (2020). https://doi.org/10.3390/app10238477.

10. GitHub - privacy-scaling-explorations/zkevm-specs, https://github.com/privacy-scaling-explorations/zkevm-specs, last accessed 2023/03/16.

11. zkEVM - Scroll, https://scroll.io/blog/zkEVM, last accessed 2023/03/19.

12. Bégassat, O., Belling, A., Chapuis-Chkaiban, T., Delehelle, F., Kolad, B., Liochon, N.: A ZK-EVM specification - Part 2 - Layer 2, https://ethresear.ch/t/a-zk-evm-specification-part-2/13903, (2022).

13. Bez, M., Fornari, G., Vardanega, T.: The scalability challenge of ethereum: An initial quantitative analysis. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). pp. 167–176 (2019). https://doi.org/10.1109/SOSE.2019.00031.

14. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized Anonymous Payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE, San Jose, CA (2014). https://doi.org/10.1109/SP.2014.36.

15. Bowe, S., Gabizon, A., Miers, I.: Scalable Multi-party Computation for zk-SNARK Parameters in the

Random Beacon Model, https://eprint.iacr.org/2017/1050, (2017).

16. Kohlweiss, M., Maller, M., Siim, J., Volkhov, M.: Snarky Ceremonies. In: Tibouchi, M. and Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 98–127. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-92078-4_4.

17. Episode 133: Trusted Setup Ceremonies Explored, https://zeroknowledge.fm/133-2/, last accessed 2023/03/17.

18. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III. pp. 698–728. Springer (2018).

19. Parameter Generation, https://z.cash/technology/paramgen/, last accessed 2023/04/06.

20. Cash, T.: Tornado.cash Trusted Setup Ceremony, https://tornado-cash.medium.com/tornado-cash-trusted-setup-ceremony-b846e1e00be1, last accessed 2023/04/06.

21. Labs, M.: Security, https://docs.zksync.io/userdocs/security/, last accessed 2023/04/06.

22. Wood, D.G.: ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. Available Online Httpsethereumgithubioyellowpaperpaperpdf. 39 (2013).

23. An Incomplete Guide to Rollups, https://vitalik.eth.limo/general/2021/01/05/rollup.html, last accessed 2024/01/12.

24. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: 23rd ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 14). pp. 781–796 (2014).

25. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable Zero Knowledge via Cycles of Elliptic Curves. In: Garay, J.A. and Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 276–294. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_16.

26. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits, https://eprint.iacr.org/2022/1758, (2022).

27. Di, Z., Xia, L., Nguyen, W., Tyagi, N.: MUXProofs: Succinct Arguments for Machine Computation from Tuple Lookups, https://eprint.iacr.org/2023/974, (2023).

28. Choudhuri, A.R., Garg, S., Goel, A., Sekar, S., Sinha, R.: SublonK: Sublinear Prover PlonK, https://eprint.iacr.org/2023/902, (2023).

29. Wahby, R.S., Setty, S., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. Presented at the Network & Distributed System Security Symposium (NDSS) February 1 (2015).

30. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster Verifiable RAM with Program-Independent Preprocessing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 908–925 (2018). https://doi.org/10.1109/SP.2018.00013.

31. Kosba, A., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs, https://eprint.iacr.org/2020/278, (2020).

32. Bayer, S., Groth, J.: Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In: Pointcheval, D. and Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 263–280. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_17.

33. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability. In: Takagi, T. and Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 336–365. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_12.

34. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. pp. 177–194. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11.

35. snarkjs, https://github.com/iden3/snarkjs, (2023).

36. Vezenov, M.: Accelerating zkSNARKs on Modern Architectures, (2022).

37. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the twenty-third annual ACM symposium on Theory of Computing. pp. 21–32. Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/103418.103428.

38. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Proceedings of the forty-fifth annual ACM symposium on Theory of Computing. pp. 111–120. Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2488608.2488623.

39. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: Dodis, Y. and Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 359–388. Springer Nature Switzerland, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_13.

40. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables, https://eprint.iacr.org/2020/315, (2020).

41. Shoup, V.: Lower Bounds for Discrete Logarithms and Related Problems. In: Fumy, W. (ed.) Advances in Cryptology — EUROCRYPT ’97. pp. 256–266. Springer, Berlin, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18.

42. Ben-Sasson, E., Sudan, M.: Short PCPs with Polylog Query Complexity. SIAM J. Comput. 38, 551–607 (2008). https://doi.org/10.1137/050646445.

43. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: Johansson, T. and Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37.

44. Lindell, Y.: Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation, https://eprint.iacr.org/2001/107, (2001).

45. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups, https://eprint.iacr.org/2022/1763, (2022).

46. Groth, J., Ishai, Y.: Sub-linear Zero-Knowledge Argument for Correctness of a Shuffle. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. pp. 379–396. Springer (2008). https://doi.org/10.1007/978-3-540-78967-3_22.

47. Fuchsbauer, G., Kiltz, E., Loss, J.: The Algebraic Group Model and its Applications. In: Shacham, H. and

Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 33–62. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2.

48. Ghoshal, A., Tessaro, S.: Tight State-Restoration Soundness in the Algebraic Group Model. In: Malkin, T. and Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 64–93. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_3.

49. Goldreich, O., Sahai, A., Vadhan, S.: Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 399–408. Association for Computing Machinery, New York, NY, USA (1998). https://doi.org/10.1145/276698.276852.

Appendix A – Proof of Lemma 2

It is straightforward to see that the existence of $h(X)$ such that $p(X)=h(X)t_{\mathcal{X}}(X)$ implies $p(x)=0$ for every $x\in\mathcal{X}$. We show the converse, if $p(x)=0$ for every $x\in\mathcal{X}$ then there exists such a $h(X)$, by contradiction. Pick any $x_k\in\mathcal{X}$, and suppose that $p(x_k)\neq 0$ but there exists such a $h(X)$. Let $h^{(k)}(X)=h(X)\prod_{x\in\mathcal{X},x\neq x_k}(X-x)$ so that $p(X)=h^{(k)}(X)(X-x_k)$. Since p and $h^{(k)}$ are polynomials, we can express them as $\sum_{i=0}^{d}p_iX^i$ and $\sum_{i=0}^{d-1}h_i^{(k)}X^i$, respectively. Then, we have $\sum_{i=0}^{d}p_iX^i=h_{d-1}^{(k)}X^d+\sum_{i=1}^{d-1}(h_{i-1}^{(k)}-x_kh_i^{(k)})X^i-x_kh_0^{(k)}$. This implies a recurrence $p_i=h_{i-1}^{(k)}-x_kh_i^{(k)}$ for $i=1,\cdots,d-1$ with initial conditions $h_{d-1}^{(k)}=p_d$ and $-x_kh_0^{(k)}=p_0$. Solving the recurrence gives us $\sum_{i=0}^{d}p_ix_k^i=0$, i.e., $p(x_k)=0$, which contradicts the supposition $p(x_k)\neq 0$. □

Appendix B – Proof of Corollary 1

It is straightforward to see that if there are such $h_{\mathcal{X}}$ and $h_{\mathcal{Y}}$, $p(X,Y)$ vanishes on $\mathcal{X}\times\mathcal{Y}$. To see the converse, we suppose $p(X,Y)$ vanishes on $\mathcal{X}\times\mathcal{Y}$ and then show there is a unique representation $h_0,h_1,h_2\in\mathbb{F}[X,Y]$ such that $p(X,Y)=h_0(X,Y)t_{\mathcal{X}}(X)+h_1(X,Y)t_{\mathcal{Y}}(X)+h_2(X,Y)t_{\mathcal{X}}(X)t_{\mathcal{Y}}(X)$.

We first construct h_0. By Lemma 2, $p(X,y)$ for every $y\in\mathcal{Y}$ has a quotient polynomial $h_0^{(y)}$ such that $p(X,y)=h_0^{(y)}(X)t_{\mathcal{X}}(X)$. Let $h_0(X,Y)$ be an interpolating polynomial of data points $(y,h_0^{(y)}(X))$ for all $y\in\mathcal{Y}$ so that $h_0(X,y)=h_0^{(y)}(X)$. We next construct h_1. Let $q(X,Y):=p(X,Y)-h_0(X,Y)t_{\mathcal{X}}(X)$. By Lemma 2, $q(x,Y)$ for every $x\in\mathcal{X}$ has a quotient polynomial $h_1^{(x)}$ such that $q(x,Y)=h_1^{(x)}(Y)t_{\mathcal{Y}}(Y)$. We can obtain an interpolating polynomial $h_1(X,Y)$ of data points $(x,h_1^{(x)}(Y))$ for all $x\in\mathcal{X}$.

Given h_0 and h_1, we find h_2. Let $r(X,Y):=p(X,Y)-h_0(X,Y)t_{\mathcal{X}}(X)-h_1(X,Y)t_{\mathcal{Y}}(Y)$. We can see that $r(X,y)=0$ for every $y\in\mathcal{Y}$, since $p(X,y)=h_0^{(y)}(X)t_{\mathcal{X}}(X)$ and $h_0(X,y)=h_0^{(y)}(X)$. Applying Lemma 2 to $r(X,y)=0$ implies there exists h_3 such that $r(X,Y)=h_3(X,Y)t_{\mathcal{Y}}(Y)$. Similarly, as it holds true that $r(x,Y)=0$ for every $x\in\mathcal{X}$, there also exists h_4 such that $h_3(X,Y)=h_4(X,Y)t_{\mathcal{X}}(X)$. Combining them we finally obtain $r(X,Y)=h_2(X,Y)t_{\mathcal{X}}(X)t_{\mathcal{Y}}(Y)$, where $h_2(X,Y)=h_3(X,Y)h_4(X,Y)$.

Letting $h_{\mathcal{X}}(X,Y)=h_0(X,Y)$ and $h_{\mathcal{Y}}(X,Y)=h_1(X,Y)+h_2(X,Y)t_{\mathcal{X}}(X)$ concludes the proof. □

Appendix C – Proof of Proposition 1

The left-hand side of (35) can be rewritten as

$$\sum_{j=l}^{l_D-1}b_{j-l}(Y)\sum_{k=l,k\neq j}^{l_D-1}o_k(X)K_{j-l}(Z)K_{k-l}(Z).$$

Our goal is to rewrite $K_j(Z)K_k(Z)=(c_kK_j(Z)-c_jK_k(Z))t_{\mathcal{Z}}(Z)$ for some coefficients c_j and c_k. Let $h=l_D-l$. Then, a Lagrange basis $K_j(Z)$ can be expressed as $K_j(Z)=c_j(Z^h-1)/(Z-\omega_Z^i)$, where $c_j:=\omega_Z^{-j(h-1)}h^{-1}$. This allows us to write

$$D_{j,k}(Z) := \frac{K_j(Z)K_k(Z)}{t_{\mathcal{Z}}(Z)} = c_j c_k \frac{(Z^h - 1)}{(Z-\omega_Z^j)(Z-\omega_Z^k)} = \frac{c_k K_j(Z)}{(Z-\omega_Z^k)} = \frac{c_j K_k(Z)}{(Z-\omega_Z^j)}.$$

Meanwhile, we define a polynomial $\tilde{D}_{i,k}(Z)$ as

$$\tilde{D}_{j,k}(Z) := \frac{c_k K_j(Z) - c_j K_k(Z)}{\omega_Z^j - \omega_Z^k}.$$

We can observe that both $D_{i,k}$ and $\tilde{D}_{i,k}$ are of degree $h-2$ and that they return the same evaluation on $Z = \omega_Z^k$ for every $k = 0, \cdots, h-1$. Thus, $D_{i,k}(Z) = \tilde{D}_{i,k}(Z)$. By letting

$$M_j(X,Z) = \sum_{k=l, k\neq j}^{l_D - l} o_k(X) D_{j,k}(Z),$$

we can conclude that the expressions (35) and (37) are identical. ☐

Appendix D– Proof of Proposition 3

We start the proof by rewriting components in a transcript **t** into the affine prover strategy of generic adversaries as defined below, where

$$\mathbf{t} = \begin{pmatrix} [U]_1, [V]_2, [W]_1, [A]_1, [B]_1, [C]_1, \boldsymbol{\theta}, [R]_1, \kappa_0, [Q]_1, \\ \zeta, \xi, R'_{y,z}, R''_{y,z}, B_{y,z}, \kappa_1, [\Pi_0]_1, [\Pi_1]_1, [\Pi_2]_1, [\Pi_3]_1 \end{pmatrix}.$$

Definition 6 (Affine prover strategy). *Let $\mathcal{A}$ be a polynomial-time adversary. In generic group model, any group elements $[G]_1$ or $[V]_2$ returned by $\mathcal{A}$ are expressed as linear combinations of $(\boldsymbol{\sigma}_{A,I}, \boldsymbol{\sigma}_C, \boldsymbol{\sigma}_{zk})$ or $\boldsymbol{\sigma}_V$, respectively. Formally, let*

$$\mathbf{T} := (X, Y, Z, \mathrm{A}, \mathrm{B}, \Gamma, \Delta, \mathrm{H}_0, \mathrm{H}_1, \mathrm{M}, \mathrm{N}, \Psi_0, \Psi_1, \Psi_2, \Psi_3, \mathrm{K}) \tag{44}$$

be a vector of 16 indeterminates. There are degree-bounded Laurent polynomials $G(\mathbf{T})$ and $V(\mathbf{T})$ such that $G = G(\boldsymbol{\tau})$ and $V = V(\boldsymbol{\tau})$, where

$$\begin{aligned} G(\boldsymbol{\tau}) &= G^{(\alpha)}\alpha + G^{(\beta)}\beta + G^{(r_0)}\delta + G^{(r_1)}\eta_1 + G^{(xy)}(x,y) + \mu^{-1}\left(G^{(yz)}(y,z) + G^{(\mu_{zk})}(y)t_{\mathcal{Y}}(y)\right) \\ &+ \gamma^{-1}\sum_{i=0}^{l-1} G_i^{(\gamma)}(y)o_i(x) + \eta_1^{-1}\left(\sum_{i=l}^{l_D-1} G_i^{(\eta_1)}(y)o_i(x) + G^{(\eta_{1,zk})}t_{\mathcal{Y}}(y)\sum_{j=l}^{l_D-1} o_j(x)\right) \\ &+ \delta^{-1}\left(G^{(q_0)}(x,y)t_{\mathcal{X}}(x) + G^{(q_1)}(x,y)t_{\mathcal{Y}}(y) + \sum_{i=l_D}^{m_D-1} G_i^{(\delta)}(y)o_i(x)\right) \\ &+ \nu^{-1}\left(G^{(q_y)}(y,z)t_{\mathcal{Y}}(y) + G^{(q_z)}(y,z)t_{\mathcal{Z}}(z)\right) \\ &+ \psi_0^{-1}\sum_{i=0}^{1}\kappa^i G_i^{(\psi_0)}(y,z) + \psi_1^{-1}G^{(\psi_1)}(z) + \psi_2^{-1}\kappa^2 G^{(\psi_2)}(y,z) + \psi_3^{-1}\sum_{i=1}^{2}\kappa^i G_i^{(\psi_3)}(z) \\ &+ \eta_0^{-1}\left(\sum_{i=l}^{l_D-1} G_i^{(\eta_0)}(y)o_i(x)\left(K_{i-l}^2(z) - 1\right) + \sum_{i=l}^{l_D-1} G_i^{(\mu\eta_1)}(y)M_i(x,z)t_{\mathcal{Z}}(z)\right), \end{aligned}$$

and

$$
\begin{aligned}
V(\boldsymbol{\tau}) &= V^{(\beta)}\beta + V^{(\gamma)}\gamma + V^{(\delta)}\delta + V^{(\eta_1)}\eta_1 + V^{(xy)}(x,y) \\
&+ V^{(\mu)}\mu^3\nu + \mu^4\sum_{i=0}^{2} V_i^{(\kappa)}\kappa^i + \mu^3\sum_{i=0}^{3}\psi_i V_i^{(yz)}(y,z) \\
&+ V^{(\eta_0)}\mu\eta_0 + V^{(\mu\eta_1)}\mu\eta_1 + V^{(o)}\mu^2\sum_{i=l}^{l_D-1} o_i(x) K_{i-l}(z).
\end{aligned}
$$

We write a tensor product of the two vectors $(\boldsymbol{\sigma}_{A,I}, \boldsymbol{\sigma}_C, \boldsymbol{\sigma}_{zk})$ *and* $\boldsymbol{\sigma}_V$ *as* $\tilde{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}_{A,I}, \boldsymbol{\sigma}_C, \boldsymbol{\sigma}_{zk}) \otimes \boldsymbol{\sigma}_V$ *. We let* $\deg_{\tilde{\boldsymbol{\sigma}}}(\boldsymbol{\tau})$ *denote an upper bound on the sum of the highest exponent subtracted by the lowest negative exponent of each component of* $\boldsymbol{\tau}$ *over all components of* $\tilde{\boldsymbol{\sigma}}$ *(i.e., the total degree of* $G(\mathbf{T})V(\mathbf{T})$ *with the negative degree compensated). There is a constant* c *such that* $\deg_{\tilde{\boldsymbol{\sigma}}}(\boldsymbol{\tau}) \le c(n + s_{\max} + l_D - l)$ *.*

By Definition 6 and the bilinearity of pairing, we can rewrite the verification equation (38) as $P(\boldsymbol{\tau}) = 0$, where $P(\mathbf{T})$ is a degree-bounded Laurent polynomial of $\mathbf{T}$ (the indeterminates are listed in (44)), evaluated as

$$
\begin{aligned}
&P(\boldsymbol{\tau}) := P_0(\boldsymbol{\tau}) + \mu P_1(\boldsymbol{\tau}) + \mu^3 P_2(\boldsymbol{\tau}), \\
&P_0(\boldsymbol{\tau}) := -U(\boldsymbol{\tau})V(\boldsymbol{\tau}) + \alpha\beta + \eta_1 W(\boldsymbol{\tau}) + \delta C(\boldsymbol{\tau}) + \sum_{i=0}^{l_{in}-1} a_i^{(in)} L_0(y) o_i(x) + \sum_{i=l_{in}}^{l-1} a_i^{(out)} L_{s_{\max}-1}(y) o_i(x), \\
&P_1(\boldsymbol{\tau}) := \mu B(\boldsymbol{\tau}) \sum_{i=l}^{l_D-1} o_i(x) K_i(z) - \eta_0 A(\boldsymbol{\tau}) - \eta_1 W(\boldsymbol{\tau}), \\
&P_2(\boldsymbol{\tau}) := P_3(\boldsymbol{\tau}) + \kappa_1\left(\mu B(\boldsymbol{\tau}) - B_{y,z}\right) + \kappa\left(\mu R(\boldsymbol{\tau}) - R'_{y,z}\right) + \kappa^2\left(\mu R(\boldsymbol{\tau}) - R''_{y,z}\right) - \nu Q(\boldsymbol{\tau}) - P_4(\boldsymbol{\tau}),
\end{aligned}
$$

$$
\begin{aligned}
P_3(\boldsymbol{\tau}) &:= \left(\mu R(\boldsymbol{\tau}) - 1\right) L_{-1}(\zeta) K_{-1}(\xi) \\
&+ \kappa_0(\xi - 1)\begin{pmatrix} \mu R(\boldsymbol{\tau})\left(B_{y,z} + \theta_0\zeta + \theta_1 s^{(2)}(\zeta,\xi) + \theta_2\right) \\ -R'_{y,z}\left(\mu B(\boldsymbol{\tau}) + \theta_0 s^{(0)}(Y,Z) + \theta_1 s^{(1)}(Y,Z) + \theta_2\right) \end{pmatrix} \\
&+ \kappa_0^2 K_0(\xi)\begin{pmatrix} \mu R(\boldsymbol{\tau})\left(B_{y,z} + \theta_0\zeta + \theta_1 s^{(2)}(\zeta,\xi) + \theta_2\right) \\ -R''_{y,z}\left(\mu B(\boldsymbol{\tau}) + \theta_0 s^{(0)}(y,z) + \theta_1 s^{(1)}(y,z) + \theta_2\right) \end{pmatrix},
\end{aligned}
$$

$$
\begin{aligned}
P_4(\boldsymbol{\tau}) &:= \psi_0\left(Y - \zeta\right)\Pi_0(\boldsymbol{\tau}) + \psi_1\left(Z - \xi\right)\Pi_1(\boldsymbol{\tau}) \\
&+ \psi_2\left(Y - \omega_Y^{-1}\zeta\right)\Pi_2(\boldsymbol{\tau}) + \psi_3\left(Z - \omega_Z^{-1}\xi\right)\Pi_3(\boldsymbol{\tau}).
\end{aligned}
$$

For simplicity, we denote $(a_0(y), \cdots, a_{l-1}(y)) = (a_0^{(in)} L_0(y), \cdots, a_{l_{in}-1}^{(in)} L_0(y), a_{l_{in}}^{(out)} L_{s_{\max}-1}(y), \cdots, a_{l-1}^{(out)} L_{s_{\max}-1}(y))$.

1) Witness-extended emulation

For the copy constraints, we run a witness-extended emulation on $\mathtt{IP}_{\mathcal{P}^*,\mathcal{V}}$, which runs a deterministic $\mathcal{P}^*$ on the fixed input of $\mathcal{A}$'s affine strategy $(\mathbf{a}^{(in)}, \mathbf{a}^{(out)}, \mathbf{b}(Y), \mathbf{c}(Y))$, with fresh randomness of ζ, ξ picked by $\mathcal{V}$. In specific, given a leading part

$$
\left([U]_1, [V]_2, [W]_1, [A]_1, [B]_1, [C]_1, \boldsymbol{\theta}, [R]_1, \kappa_0, [Q]_1\right)
$$

of the transcript $\mathbf{t}$ submitted by $\mathcal{P}^*$, the emulator runs $\mathtt{IP}_{\mathcal{P}^*,\mathcal{V}}$ with fresh randomness of ζ,ξ,κ_0 to complete the rest part of the transcript

$$\left(\zeta,\xi,R'_{y,z},R''_{y,z},B_{y,z},\kappa_1,[\Pi_0]_1,[\Pi_1]_1,[\Pi_2]_1,[\Pi_3]_1\right).$$

We run the emulation until collecting $s_{\max}(l_D-l)$ pairs $((\zeta,\xi,\kappa_1),\mathbf{t})$ of emulated challenges and the corresponding acceptable transcripts. Let $\mathcal{S}$ denote the collection of emulation results. Per every challenge $((\zeta,\xi,\kappa_1),\mathbf{t})\in\mathcal{S}$, the construction of function P can vary, and therefore given all emulated transcripts in $\mathcal{S}$ are acceptable, the emulator can collect $|\mathcal{S}|$ instances of $P(\boldsymbol{\tau})=0$. Throughout the rest part of proof, we will show that $|\mathcal{S}|=(|\mathcal{Y}|+2)|\mathcal{Z}|$ (where the constant 2 is needed for adding zero-knowledge) is sufficient to succeed the extraction of a valid witness in R.

Before the extraction, we first count the number of expected runs for the successful witness extraction. Suppose $\mathcal{P}^*$ returns an acceptable transcript with an unknown probability ϵ. The emulator repeats the protocol until collecting $|\mathcal{S}|$ acceptable transcripts. When $\mathcal{P}^*$ fails to produce an acceptable transcript, which contradicts the assumption $P(\boldsymbol{\tau})=0$, the emulator stops the extraction procedure, immediately. Given every transcript is acceptable, the emulator is thus expected to run in a polynomial time $\epsilon|\mathcal{S}|/\epsilon=|\mathcal{S}|$.

2) Polynomial reconstruction from randomness in $\boldsymbol{\tau}$

Suppose $P(\boldsymbol{\tau})=0$ for all emulated transcripts in $\mathcal{S}$. We apply the SZ lemma to $P(\boldsymbol{\tau})=0$ with respect to $\boldsymbol{\tau}$. Let $\mathcal{E}_{\boldsymbol{\tau}}\subset(\mathbb{F}^*)^{13}$ be the set of roots of $P(\mathbf{T})$, where the coefficients of P vary by the emulation. By the SZ lemma, we have $\Pr[\boldsymbol{\tau}\in\mathcal{E}_{\boldsymbol{\tau}}]\le|\mathcal{S}|\deg_{\bar{\boldsymbol{\sigma}}}(\boldsymbol{\tau})/\mathbb{F}^*$.

Suppose $\boldsymbol{\tau}\in(\mathcal{E}_{\boldsymbol{\tau}})^C$. Under this case, for all transcripts in $|\mathcal{S}|$, $P(\boldsymbol{\tau})=0$ implies $P(\mathbf{T})=0$, which further implies $P_0(\mathbf{T})=P_1(\mathbf{T})=P_2(\mathbf{T})=0$, and vice versa. Thus, throughout the rest of proof, notations $P_0(\boldsymbol{\tau})=0$, $P_1(\boldsymbol{\tau})=0$, and $P_2(\boldsymbol{\tau})=0$ are equivalently used as $P_0(\mathbf{T})=0$, $P_1(\mathbf{T})=0$, and $P_2(\mathbf{T})=0$, respectively.

3) Extraction of arithmetic constraints

We extract the arithmetic constraints from $P_0(\boldsymbol{\tau})=0$. To this end, we first show that to refine the expression of $U(\boldsymbol{\tau})$ can be reduced into

$$U(\boldsymbol{\tau})=U^{(\alpha)}\alpha+U^{(xy)}(x,y)+U^{(r_0)}\delta+U^{(r_1)}\eta_1.$$

In $P_0(\boldsymbol{\tau})=0$ for each transcript in $\mathcal{S}$, collecting the terms of $\alpha\beta$ forms an equation $U^{(\alpha)}V^{(\beta)}\alpha\beta-\alpha\beta=0$, which implies that

$$U^{(\alpha)}V^{(\beta)}=1 \tag{45}$$

and that $U^{(\alpha)}\neq 0$ and $V^{(\beta)}\neq 0$. Collecting terms of β^2 forms $U^{(\beta)}V^{(\beta)}\beta^2=0$, which implies $U^{(\beta)}=0$. Collecting the terms of $\{x^iy^j\}_{i,j}$ scaled by $\beta^2\gamma^{-1}$, $\alpha\beta\gamma^{-1}$, or $\beta\gamma^{-1}$ can be found only in $V^{(\beta)}\beta\delta^{-1}\sum_{i=0}^{l-1}U_i^{(\gamma)}(y)o_i(x)$, which, given $V^{(\beta)}\neq 0$, implies $\sum_{i=0}^{l-1}U_i^{(\gamma)}(y)o_i(x)=0$. By repeating this investigation for the terms of $\{x^iy^j\}_{i,j}$ scaled by β^2s^{-1}, $\alpha\beta s^{-1}$, or βs^{-1} for $s\in\{\delta,\eta_0,\eta_1,\mu,\nu\}\cup\{\psi_i\kappa^j\}_{i=0,j=0}^{3,2}$, we obtain the above reduced expression of $U(\boldsymbol{\tau})$.

We move on to the terms of $\{\alpha x^i y^j\}_{i,j}$ in $P_0(\boldsymbol{\tau})=0$. Then, we obtain

$$\alpha U^{(\alpha)}V^{(xy)}(x,y)=\alpha\left(\sum_{i=0}^{l-1}a_i(y)v_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)v_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)v_i(x)\right). \tag{46}$$

Similarly, looking into the terms of $\{\beta x^i y^j\}_{i,j}$ gives

$$\beta V^{(\beta)}U^{(xy)}(x,y)=\beta\left(\sum_{i=0}^{l-1}a_i(y)u_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)u_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)u_i(x)\right). \tag{47}$$

From $P_0(\boldsymbol{\tau})=0$, we extract some of the terms of $\{x^i y^j\}_{i,j}$ and of those scaled by α or β and define

$$\begin{aligned}
P_0^{(xy)}(x,y) &:= \sum_{i=0}^{l-1}a_i(y)o_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)o_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)o_i(x)\\
&\quad -\alpha U^{(\alpha)}V^{(xy)}(x,y)-\beta V^{(\beta)}U^{(xy)}(x,y)-U^{(xy)}(x,y)V^{(xy)}(x,y)\\
&=\left(\sum_{i=0}^{l-1}a_i(y)u_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)u_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)u_i(x)\right)\\
&\quad\cdot\left(\sum_{i=0}^{l-1}a_i(y)v_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)v_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)v_i(x)\right)\\
&\quad-\left(\sum_{i=0}^{l-1}a_i(y)w_i(x)+\sum_{i=l}^{l_D-1}W_i^{(\eta_1)}(y)w_i(x)+\sum_{i=l_D}^{m_D-1}C_i^{(\delta)}(y)w_i(x)\right),
\end{aligned}$$

where the equation is resulted from (45)-(47).

Finally, by collecting all of the terms of $\{x^i y^j\}_{i,j}$ from $P_0(\boldsymbol{\tau})=0$, we obtain

$$P_0^{(xy)}(X,Y)=C^{(q_0)}(X,Y)t_{\mathcal{X}}(X)+\left(C^{(q_1)}(X,Y)+W_i^{(\eta_{1,zk})}\sum_{j=l}^{l_D-1}o_j(X)\right)t_{\mathcal{Y}}(Y). \tag{48}$$

Since $C^{(q_0)}(X,Y)$ and $C^{(q_1)}(X,Y)+W_i^{(\eta_{1,zk})}\sum_{j=l}^{l_D-1}o_j(X)$ are polynomials of X,Y (i.e., they do not involve negative powers of X and Y), the polynomial $P_0^{(xy)}(X,Y)$ vanishes on $\mathcal{X}\times\mathcal{Y}$. Letting $b_i(Y)=W_{i-l}^{(\eta_1)}(Y)$ and $c_i(Y)=C_{i-l_D}^{(\delta)}(Y)$, we conclude that $((\mathbf{a}^{(in)},\mathbf{a}^{(out)}),(\mathbf{b}(Y),\mathbf{c}(Y)))$ satisfies the arithmetic constraints.

4) Extraction of inner product constraints

We extract the inner product constraints from $\mu P_1(\boldsymbol{\tau})=0$. To this end, we collect the terms of $\{\mu x^h y^i z^j\}_{h,i,j}$ including those scaled by α or β. Consider a polynomial $P_1^{(z)}(Z)$, which is a part of the collection, defined as

$$P_1^{(z)}(Z):=\left(\begin{array}{l}B^{(\star)}(Y,Z)\sum_{i=l}^{l_D-1}o_i(X)K_{i-l}(Z)\\ -\sum_{i=l}^{l_D-1}A_i^{(\eta_0)}(Y)o_i(X)\left(K_{i-l}^2(Z)-1\right)-\sum_{i=l}^{l_D-1}W_i^{(\star)}(Y)o_i(X)\end{array}\right),$$

where $B^{(\star)}(Y,Z) := B^{(yz)}(Y,Z) + B^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y)$, and $W_i^{(\star)} := W_i^{(\eta_1)}(Y) + W^{(\eta_{1,zk})}t_{\mathcal{Y}}(Y)$. The equation $\mu P_1(\boldsymbol{\tau}) = 0$ implies that

$$P_1^{(z)}(Z) = \sum_{i=l}^{l_D-1} A_i^{(\mu\eta_1)}(Y)M_i(X,Z)t_{\mathcal{Z}}(Z).$$

Since $\sum_{i=l}^{l_D-1} A_i^{(M)}(Y)M_i(X,Z)$ on the right-hand side is a polynomial of Z, the polynomial $P_1^{(z)}(Z)$ vanishes on $\mathcal{Z}$. In other words, for every $k = 0,\cdots,l_D - l - 1$ it holds true that

$$B^{(\star)}(Y,\omega_z^k)o_{k+l}(X) + \sum_{h=0,h\neq k}^{l_D-l-1} A_{h+l}^{(\eta_0)}(Y)o_{h+l}(X) = \sum_{h=0}^{l_D-l-1} W_{h+l}^{(\star)}(Y)o_{h+l}(X).$$

Note that the right-hand side of this equation is independent of ω_Z^k. Thus, collecting the equations by changing k and solving them simultaneously imply $A_k^{(\eta_0)}(y)o_k(x) = B^{(\star)}(y,\omega_Z^{k-l})o_k(x)$ for every k, and therefore $A_k^{(\eta_0)}(y) = B^{(\star)}(y,\omega_Z^{k-l})$. So, we can rewrite the above equation as

$$\sum_{k=0}^{l_D-l-1} B^{(\star)}(y,\omega_z^k)o_{k+l}(x) = \sum_{k=0}^{l_D-l-1} W_{k+l}^{(\star)}(y)o_{k+l}(x).$$

By the linear independency of the Lagrange bases $\{K_i(Z)\}_{k=0}^{l_D-l-1}$, we can decompose $B^{(yz)}(Y,Z) = \sum_{k=0}^{l_D-l-1} B_{k+l}^{(y)}(Y)K_h(Z)$. Then, the above equation can be rewritten as

$$\sum_{k=0}^{l_D-l-1} \left(W_{k+l}^{(\eta_1)}(Y) + W^{(\eta_{1,zk})}t_{\mathcal{Y}}(Y)\right)o_{k+l}(X) = \sum_{k=0}^{l_D-l-1} \left(B_{k+l}^{(y)}(Y) + B^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y)\right)o_{k+l}(X).$$

We finally conclude that $W^{(\eta_{1,zk})} = B^{(\mu_{zk})}(Y)$ and that

$$\sum_{k=0}^{l_D-l-1} W_{k+l}^{(\eta_1)}(Y)o_{k+l}(X) = \sum_{k=0}^{l_D-l-1} B_{k+l}^{(y)}(Y)o_{k+l}(X). \tag{49}$$

5) Extraction of copy constraints

What left to see is that $B^{(yz)}(Y,Z)$ also satisfies the copy constraints. The copy constraints can be extracted from $\mu^3 P_2(\boldsymbol{\tau}) = 0$, where $P_2(\boldsymbol{\tau})$ involves $P_3(\boldsymbol{\tau})$ and $P_4(\boldsymbol{\tau})$. Let $R^{(\star)}(Y,Z) := R^{(yz)}(Y,Z) + R^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y)$. We collect the terms of $\{\kappa^h y^i z^j\}_{h,i,j}$ from $P_3(\boldsymbol{\tau})$ and $P_4(\boldsymbol{\tau})$ and define

$$P_3^{(\star)}(Y,Z) := \left(R^{(\star)}(Y,Z)-1\right)L_{-1}(\zeta)K_{-1}(\xi)$$
$$+\kappa_0\left(\xi-1\right)\begin{pmatrix} R^{(\star)}(Y,Z)\left(B_{y,z}+\theta_0\zeta+\theta_1 s^{(2)}(\zeta,\xi)+\theta_2\right) \\ -R'_{y,z}\left(B^{(\star)}(Y,Z)+\theta_0 s^{(0)}(Y,Z)+\theta_1 s^{(1)}(Y,Z)+\theta_2\right)\end{pmatrix} \tag{50}$$
$$+\kappa_0^2 K_0(\xi)\begin{pmatrix} R^{(\star)}(Y,Z)\left(B_{y,z}+\theta_0\zeta+\theta_1 s^{(2)}(\zeta,\xi)+\theta_2\right) \\ -R''_{y,z}\left(B^{(\star)}(Y,Z)+\theta_0 s^{(0)}(Y,Z)+\theta_1 s^{(1)}(Y,Z)+\theta_2\right)\end{pmatrix},$$

$$P_4^{(\kappa yz)}(\mathrm{K},Y,Z) := \begin{pmatrix}\left(Y-\zeta\right)\Pi_{0,0}^{(\psi_0)}(Y,Z) \\ +\left(Z-\xi\right)\Pi_1^{(\psi_1)}(Z)\end{pmatrix} + \mathrm{K}\begin{pmatrix}\left(Y-\zeta\right)\Pi_{0,1}^{(\psi_0)}(Y,Z) \\ +\left(Z-\omega_Z^{-1}\xi\right)\Pi_{3,1}^{(\psi_3)}(Z)\end{pmatrix} + \mathrm{K}^2\begin{pmatrix}\left(Y-\omega_Y^{-1}\zeta\right)\Pi_2^{(\psi_2)}(Y,Z) \\ +\left(Z-\omega_Z^{-1}\xi\right)\Pi_{3,2}^{(\psi_3)}(Z)\end{pmatrix}.$$

The equation $P_2(\boldsymbol{\tau})=0$ then implies

$$\begin{pmatrix} P_3^{(yz)}(Y,Z)+\kappa_1\left(B^{(\star)}(Y,Z)-B_{y,z}\right)-Q^{(q_y)}(Y,Z)t_{\mathcal{Y}}(Y)-Q^{(q_z)}(Y,Z)t_{\mathcal{Z}}(Z) \\ +\mathrm{K}\left(R^{(\star)}(Y,Z)-R'_{y,z}\right)+\mathrm{K}^2\left(R^{(\star)}(Y,Z)-R''_{y,z}\right)\end{pmatrix} = P_4^{(\kappa yz)}(\mathrm{K},Y,Z).$$

Since $\Pi_{i,j}^{(\psi_i)}$ in $P_4^{(\kappa yz)}(\mathrm{K},Y,Z)$ are polynomials of Y and Z, we obtain

$$P_3^{(\star)}(\zeta,\xi)+\kappa_1\left(B^{(\star)}(\zeta,\xi)-B_{y,z}\right)=Q^{(q_y)}(\zeta,\xi)t_{\mathcal{Y}}(\zeta)+Q^{(q_z)}(\zeta,\xi)t_{\mathcal{Z}}(\xi), \tag{51}$$

$$\begin{aligned} R^{(\star)}(\zeta,\omega_Z^{-1}\xi) &= R'_{y,z}, \\ R^{(\star)}(\omega_Y^{-1}\zeta,\omega_Z^{-1}\xi) &= R''_{y,z}. \end{aligned} \tag{52}$$

6) Equation separation from randomness of κ_1

We must consider the possibility that each emulated transcript for (51) held true only for the specific samples of κ_1 (The other interim implications in (52) are independent of κ_1). By the SZ lemma, the probability that at least one transcripts of (51) did not identically hold true with respect to κ_1 but did only for the specific κ_1 is upper bounded to $|\mathcal{S}|/|\mathbb{F}^*|$. We define the set $\mathcal{E}_{\kappa_1}$ of such samples κ_1.

Suppose $\kappa_1 \in (\mathcal{E}_{\kappa_1})^C$. Under this case, we can split (51) into two equations,

$$B^{(\star)}(\zeta,\xi)=B_{y,z}, \tag{53}$$

$$P_3^{(\star)}(\zeta,\xi)=Q^{(q_y)}(\zeta,\xi)t_{\mathcal{Y}}(\zeta)+Q^{(q_z)}(\zeta,\xi)t_{\mathcal{Z}}(\xi). \tag{54}$$

7) Polynomial reconstruction from emulated samples in $\mathcal{S}$

The extraction of copy constraints requires to reconstruct a polynomial that emulates the equations in (54). To this end, we first interpolate three polynomials $\tilde{B}^{(\star)}, \tilde{R}'^{(\star)}, \tilde{R}''^{(\star)} \in \mathbb{F}_{s_{\max}+2, l_D-l}[Y,Z]$ based on the data $((\zeta,\xi),B_{y,z})$, $((\zeta,\xi),R'_{y,z})$, and $((\zeta,\xi),R''_{y,z})$ obtained from $\mathcal{S}$, respectively. Then, by equations (52)-(53), the interpolated polynomials hold the follows,

$$\begin{aligned} B^{(\star)}(Y,Z) &= \tilde{B}^{(\star)}(Y,Z) := \tilde{B}^{(yz)}(Y,Z) + \tilde{B}^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y), \\ R^{(\star)}(Y,\omega_Z^{-1}Z) &= \tilde{R}'^{(\star)}(Y,Z) := \tilde{R}'^{(yz)}(Y,Z) + \tilde{R}'^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y), \\ R^{(\star)}(\omega_Y^{-1}Y,\omega_Z^{-1}Z) &= \tilde{R}''^{(\star)}(Y,Z) := \tilde{R}''^{(yz)}(Y,Z) + \tilde{R}''^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y). \end{aligned} \tag{55}$$

We then reconstruct a polynomial $\tilde{P}_3^{(\star)}(Y,Z)$ that emulates $P_3^{(\star)}(Y,Z)$, by replacing $B_{y,z}$, $R'_{y,z}$, and $R''_{y,z}$ with $\tilde{B}^{(yz)}$, $\tilde{R}'^{(yz)}$, and $\tilde{R}''^{(yz)}$ into the definition (50), respectively. By the results in (55), it holds that $\tilde{P}_3^{(\star)}(\zeta,\xi) = P_3^{(\star)}(\zeta,\xi)$ for all $(\zeta,\xi)\in\mathcal{S}$ and that, by the result in (54),

$$\begin{aligned} &\tilde{P}_3^{(\star)}(\zeta,\xi) = Q^{(q_y)}(\zeta,\xi)t_{\mathcal{Y}}(\zeta) + Q^{(q_z)}(\zeta,\xi)t_{\mathcal{Z}}(\xi) \\ &\Leftrightarrow \tilde{P}_3^{(yz)}(\zeta,\xi) = Q^{(\star)}(\zeta,\xi)t_{\mathcal{Y}}(\zeta) + Q^{(q_z)}(\zeta,\xi)t_{\mathcal{Z}}(\xi) = 0, \end{aligned} \tag{56}$$

where

$$\begin{aligned} \tilde{P}_3^{(yz)}(Y,Z) := &\left(R^{(yz)}(Y,Z)-1\right)L_{-1}(Y)K_{-1}(Z) \\ &+\kappa_0\left(Z-1\right)\begin{pmatrix} R^{(yz)}(Y,Z)\tilde{G}(Y,Z) \\ -\tilde{R}'^{(yz)}(Y,Z)F(Y,Z) \end{pmatrix} \\ &+\kappa_0^2 K_0(Z)\begin{pmatrix} R^{(yz)}(Y,Z)\tilde{G}(Y,Z) \\ -\tilde{R}''^{(yz)}(Y,Z)F(Y,Z) \end{pmatrix}, \end{aligned}$$

$$\begin{aligned} \tilde{G}(Y,Z) &:= \tilde{B}^{(yz)}(Y,Z) + \theta_0 Y + \theta_1 s^{(2)}(Y,Z) + \theta_2, \\ F(Y,Z) &:= B^{(yz)}(Y,Z) + \theta_0 s^{(0)}(Y,Z) + \theta_1 s^{(1)}(Y,Z) + \theta_2, \end{aligned}$$

$$\begin{aligned} Q^{(\star)}(Y,Z) := &\, Q^{(q_y)}(Y,Z) + R^{(\mu_{zk})}(Y,Z)L_{-1}(Y)K_{-1}(Z) \\ &+\kappa_0\left(Z-1\right)\begin{pmatrix} R^{(\mu_{zk})}(Y)\tilde{G}(Y,Z) + R^{(yz)}(Y,Z)\tilde{B}^{(\mu_{zk})}(Y) + R^{(\mu_{zk})}(Y)\tilde{B}^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y) \\ -\tilde{R}'^{(\mu_{zk})}(Y)F(Y,Z) + \tilde{R}'^{(yz)}(Y,Z)B^{(\mu_{zk})}(Y) + \tilde{R}'^{(\mu_{zk})}(Y)B^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y) \end{pmatrix} \\ &+\kappa_0^2 K_0(Z)\begin{pmatrix} R^{(\mu_{zk})}(Y)\tilde{G}(Y,Z) + R^{(yz)}(Y,Z)\tilde{B}^{(\mu_{zk})}(Y) + R^{(\mu_{zk})}(Y)\tilde{B}^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y) \\ -\tilde{R}''^{(\mu_{zk})}(Y)F(Y,Z) + \tilde{R}''^{(yz)}(Y)(Y,Z)B^{(\mu_{zk})}(Y) + \tilde{R}''^{(\mu_{zk})}(Y)B^{(\mu_{zk})}(Y)t_{\mathcal{Y}}(Y) \end{pmatrix}. \end{aligned}$$

We apply the SZ lemma to (56) with respect to ζ,ξ. Let $D(Y,Z) := \tilde{P}_3^{(yz)}(Y,Z) - Q^{(\star)}(Y,Z)t_{\mathcal{Y}}(Y) - Q^{(q_z)}(Y,Z)t_{\mathcal{Z}}(Z)$. By the SZ lemma, the probability that $D(Y,Z)\neq 0$ and that equation (56) held only for the specific ζ,ξ is upper bounded to $(2|\mathcal{Y}|+3|\mathcal{Z}|+2)/|\mathbb{F}^*|$ (where the numerator comes from the total degree of $\tilde{P}_3^{(\star)}(Y,Z)$, which is $2s_{\max}+2$ in Y and $3(l_D-l-1)$ in Z). We define the set $\mathcal{E}_{\zeta,\xi}$ of the roots of $D(Y,Z)$.

Suppose $(\zeta,\xi)\in(\mathcal{E}_{\zeta,\xi})^C$. Under this case, equation (56) implies that for all $(y^*,z^*)\in\mathcal{Y}\times\mathcal{Z}$,

$$\tilde{P}_3^{(yz)}(y^*,z^*) = 0. \tag{57}$$

8) Equation separation from randomness of κ_0

As equations (57) for all $(y^*, z^*) \in \mathcal{Y} \times \mathcal{Z}$ involve κ_0, we must check if they held true only for the specific choice of κ_0. By the SZ lemma, the probability that at least one of the equations did not identically hold true but did only for the specific κ_0 is upper bounded to $2|\mathcal{Y}||\mathcal{Z}|/|\mathbb{F}^*|$ (where the scaling factor 2 comes from the degree of κ_0). We define the set $\mathcal{E}_{\kappa_0}$ of such samples κ_0.

Suppose $\kappa_0 \in (\mathcal{E}_{\kappa_0})^C$. Under this case, the equations (57) can be converted into the following recurrence relation,

$$\begin{cases} r_{s_{\max}-1, l_D-l-1} = 1, \\ r_{j,k} g_{j,k} = r_{j,k-1} f_{j,k}, \quad \text{for } 0 \le j \le s_{\max}-1, 0 < k \le l_D - l - 1, \\ r_{j,0} g_{j,0} = r_{j-1, l_D-l-1} f_{j,0}, \quad \text{for } 0 \le j \le s_{\max}-1, \end{cases} \tag{58}$$

where $f_{j,k} = F(\omega_Y^j, \omega_Z^k)$, $g_{j,k} = \tilde{G}(\omega_Y^j, \omega_Z^k)$, $r_{j,k} = R^{(yz)}(\omega_Y^j, \omega_Z^k)$. By solving this recursion, we obtain,

$$\prod_{j=0}^{s_{\max}-1} \prod_{k=0}^{l_D-l-1} f_{j,k} = \prod_{j=0}^{s_{\max}-1} \prod_{k=0}^{l_D-l-1} g_{j,k}. \tag{59}$$

9) Polynomial reconstruction from randomness of $\boldsymbol{\theta}$

As equation (59) involves $\boldsymbol{\theta}$, we must check if it held true only for the specific choice of $\boldsymbol{\theta}$. By the SZ lemma, the probability that this equation did not identically hold true with respect to $\boldsymbol{\theta}$ but did only for a specific $\boldsymbol{\theta}$ is upper bounded to $3(|\mathcal{Y}|+|\mathcal{Z}|)/|\mathbb{F}^*|$ (where the numerator comes from the total degree of $\boldsymbol{\theta}$). We define the set $\mathcal{E}_{\boldsymbol{\theta}}$ of such samples $\boldsymbol{\theta}$.

Suppose $\boldsymbol{\theta} \in (\mathcal{E}_{\boldsymbol{\theta}})^C$. Under this case, by Lemma 3, $\mathbf{b}(Y)$ satisfies the copy constraint.

10) Concluding the extraction

So far, we have extracted $b_i(Y) = W_{i-l}^{(\eta_1)}(Y)$ and $c_i(Y) = C_{i-l_D}^{(\delta)}(Y)$ such that $((\mathbf{a}^{(in)}, \mathbf{a}^{(out)}), (\mathbf{b}(Y), \mathbf{c}(Y))) \in R$ with assuming that $\boldsymbol{\tau} \in (\mathcal{E}_{\boldsymbol{\tau}})^C$, $\kappa_1 \in (\mathcal{E}_{\kappa_1})^C$, $(\zeta, \xi) \in (\mathcal{E}_{\zeta,\xi})^C$, $\kappa_0 \in (\mathcal{E}_{\kappa_0})^C$, and $\boldsymbol{\theta} \in (\mathcal{E}_{\boldsymbol{\theta}})^C$. The extraction fails when the challenges are picked from the complement sets, of which the probability is bounded above by

$$\begin{aligned} \Pr[\textit{Extraction failed}] &\le \frac{|\mathcal{S}|\deg_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) + |\mathcal{S}| + (2|\mathcal{Y}| + 3|\mathcal{Z}| + 2) + 2|\mathcal{Y}||\mathcal{Z}| + 3(|\mathcal{Y}| + |\mathcal{Z}|)}{|\mathbb{F}^*|} \\ &= \frac{s_{\max}(l_D - l)(\deg_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) + 3) + 5 s_{\max} + (l_D - l)(2\deg_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) + 8) + 2}{|\mathbb{F}^*|}. \end{aligned}$$