Succinct Non-Interactive Arguments from Quadratic Arithmetic Programs

Alisa Pankova

University of Tartu, Cybernetica AS

Abstract. Verifiable computation in zero knowledge allows the verifier to prove that he performed the computation of a certain functionality correctly without having to repeat the entire computational process, and without revealing any details of the computation. Any program can be represented by a circuit, and verifying the correctness of the computation is equivalent to checking circuit satisfiability on given input and output. There exist cryptographic solutions to this task, based on reduction to different mathematical problems. This survey gives an overview of verifying circuit satisfiability (Circuit-SAT) in zero knowledge using quadratic arithmetic programs (QAP).

1 Introduction

Suppose that a user does not have enough resources to perform the computation of some functionality by himself. He may want to give it to some cloud that would perform that computation instead of him. At the same time, the user may have no reason to trust the cloud. He may want to verify that the obtained result is indeed correct, but of course without repeating the entire computation by himself. In this case, some quicker way to verify the computation is needed. Also, if the cloud generates some randomness or other non-deterministic input, it may want to keep the details of the computation private.

Consider the case where several parties securely compute some functionality, where the input and the output are shared amongst the parties according to some scheme. If the protocol is secure only against passive adversaries, then some party might have cheated in the computation. Therefore any party should be able to prove to any other party that it performed its computation correctly, but without providing any extra information about its own shares and the randomness.

The previous problems have cryptographic solutions. Any functionality can be represented as an arithmetic or boolean circuit. Given the public input and the public output of the circuit, the task of the prover is to convince any verifier that he performed the computation correctly, but without revealing anything except the public data that is already known.

There can be different requirements for proofs. One is the *designated verifier* proof, which is intended for some particular verifier who knows the secret verification key that the prover should not know. The *public verifier* proof is meant to be provable to anyone, and there is no need for some secret key that should be unknown to the prover. This survey is about public verification.

Notation In this survey, we will use the following notation:

- bold lower case letters $\boldsymbol{x}$ represent vectors;
- upper case letters A represent matrices;
- AB, $A\boldsymbol{x}$ represent multiplication of a matrix by another matrix/vector;
- $\circ$ represents pointwise multiplication of two vectors;
- $\cdot$ is an ordinary field multiplication.

2 Arithmetic Circuit-SAT to Quadratic Arithmetic Program

Any program can be reduced to an arithmetic circuit (a circuit that contains only addition and multiplication gates). A particular reduction can be found for example in [BSCG+13]. Given a (possibly partial) circuit evaluation, the task of the prover is to convince the verifier that there exist valuations of intermediate gates (and possibly some additional inputs) such that the circuit indeed produces such an output with such an input.

This section introduces the notion of quadratic arithmetic program (QAP) and shows one possible reduction from Arithmetic Circuit-SAT to QAP. The main idea is to represent each gate input and each gate output as a variable. Then we may rewrite each gate as an equation. For example, an addition gate can be written as $x + y = z$, where x and y correspond to the inputs and z to to the output. If the output is in turn used as an input for some other gate, we use the same variable there. The construction will in fact be a bit more compact, since an entire subcircuit that represents some linear combination will be represented by a single equation. In this way, the prover needs to convince the verifier that all the equations are satisfiable.

Let us now state it more formally. Here are used the definition and reduction proposed in [Lip13], a similar construction is being used in [BSCG+13], but the representation itself looks a bit different. The main difference is that the target vector is included into the matrix, and that multiplication is written as a matrix times a vector.

Definition 1 (Quadratic Arithmetic Program). *Consider some integers m, n, k such that $n - 1 \geq k$. A* strong quadratic arithmetic program (QAP) *over a field $\mathbb{F}$, denoted $\mathsf{P}(A, B, C)$, consists of three $m \times n$ matrices A, B, C over a field $\mathbb{F}$. P accepts a vector $\boldsymbol{x} \in \mathbb{F}^k$ iff there exists a vector $\boldsymbol{w} = (1, w_1, \ldots, w_{n-1})$ such that $(w_1, \ldots, w_k) = \boldsymbol{x}$ and $A\boldsymbol{w} \circ B\boldsymbol{w} = C\boldsymbol{w}$.*

- *The* dimension *of P is $\mathsf{sdim}(\mathsf{P}) = m$ (the number of rows in A, B, C).*
- *The* size *of P is $\mathsf{size}(\mathsf{P}) = n$ (the number of columns in A, B, C).*
- *The* support *of P, denoted $\mathsf{supp}(\mathsf{P})$, is the number of non-zero elements in A,B,C together.*

The relation $R_{\mathsf{P}(A,B,C)}$ is defined as

$$(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)} \iff \mathsf{P}(A, B, C) \text{ accepts on input } \boldsymbol{x} \ .$$

Since in this survey we will use only strong QAP, we will refer to a strong QAP just as QAP. We want to use QAP to describe function computations.

Definition 2. *We say that the quadratic arithmetic program* P computes a function $f : \mathbb{F}^s \to \mathbb{F}^t$ *if for all* $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}^s \times \mathbb{F}^t$ *it is true that* $f(\boldsymbol{x}) = \boldsymbol{y}$ *iff* P *accepts* $(\boldsymbol{x}||\boldsymbol{y})$*, where* $||$ *denotes concatenation.*

The proposed way of composing a QAP for a function f is based on the arithmetic circuit that describes f.

Definition 3. *An* arithmetic circuit *consists of addition, multiplication by scalar, multiplication of two variables, and constant gates only. A* multiplication subcircuit *is an arithmetic circuit that has exactly one two-variable multiplication gate which is the last gate.*

The main reason why the notion of a multiplication circuit is introduced is that we first describe how to compose a QAP for a multiplication subcircuit, and then how to recursively compose QAP-s for multiplication subcircuits to a single QAP for the entire arithmetic circuit, which takes the outputs of the multiplication subcircuits as inputs. If the final output gate of the arithmetic circuit is an addition gate, then a dummy multiplication by 1 has to be performed in the end.

Multiplication subcircuit If C is a multiplication subcircuit with input $\boldsymbol{x} = (x_1, \ldots, x_m)$ and output $\boldsymbol{y} = (y)$, then the function it computes can be written as

$$y = \rho_1(x_1, \ldots, x_m) \cdot \rho_2(x_1, \ldots, x_m)$$

for some affine functions ρ_1, ρ_2 (an affine function is of the form $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$ for some matrix A and vector $\boldsymbol{b}$). Since they are affine, we may write

$$\rho_i(x_1, \ldots, x_m) = \boldsymbol{d_i} \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}$$

for an appropriate vector $\boldsymbol{d_i}$, and hence the corresponding equality looks like

$$(0, \ldots, 0, 1) \begin{pmatrix} 1 \\ \boldsymbol{x} \\ y \end{pmatrix} = (\boldsymbol{d_1}||0) \begin{pmatrix} 1 \\ \boldsymbol{x} \\ y \end{pmatrix} \cdot (\boldsymbol{d_2}||0) \begin{pmatrix} 1 \\ \boldsymbol{x} \\ y \end{pmatrix} \quad .$$

This is an instance of QAP with $A = (\boldsymbol{d_1}||0)$, $B = (\boldsymbol{d_2}||0)$, and $C = (0, \ldots, 0, 1)$. Here we have $\boldsymbol{w} = \begin{pmatrix} 1 \\ \boldsymbol{x} \\ y \end{pmatrix}$.

Composing subcircuits A general arithmetic circuit can be composed from its multiplication subcircuits using compositions.

Let $\mathsf{P}_1(A_1, B_1, C_1)$ and $\mathsf{P}_2(A_2, B_2, C_2)$ be two QAP-s that compute some functions f_1 and f_2 respectively (the exact definitions of f_1 and f_2 do not matter).

Let some output variables of f_1 act as some input variables of f_2. Let I_1 be the variables used by f_1, and I_2 the variables used by f_2 (both the input and the output). In general $I_1 \cap I_2 \neq \emptyset$. Each variable in I_i corresponds to the i-th column of each of the matrices A_i, B_i, C_i. Extend A_i, B_i, C_i over the entire vector of variables $I_1 \cup I_2$ by introducing zero columns for the unused variables, obtaining the matrices A'_i, B'_i, C'_i of $|I_1 \cup I_2|$ columns, where again each variable is represented by exactly one column. The QAP that computes the composition $f(\boldsymbol{x_1}, \boldsymbol{x_2}) = f_2(\boldsymbol{x_2}, (f_1(\boldsymbol{x_1})))$ ($\boldsymbol{x_i} \in I_i$), denoted $\mathsf{P}(A, B, C)$, is defined as follows: $A := \begin{pmatrix} A'_1 \\ A'_2 \end{pmatrix}$, $B := \begin{pmatrix} B'_1 \\ B'_2 \end{pmatrix}$, $C := \begin{pmatrix} C'_1 \\ C'_2 \end{pmatrix}$. By definition, this composition performs all the checks that both P_1 and P_2 do. The size of the obtained QAP is $\mathsf{size}(\mathsf{P}_1)+\mathsf{size}(\mathsf{P}_2)-|I_1 \cap I_2|$, and $\mathsf{sdim}(\mathsf{P}) = \mathsf{sdim}(\mathsf{P}_1)+\mathsf{sdim}(\mathsf{P}_2)$.

In this way, each multiplication node of the circuit contributes a row to A, B, C. In addition to the initial input and output vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, there will be some new variables $\boldsymbol{z}$ that represent the values of the intermediate multiplication gate outputs, so in general $\boldsymbol{w} = (1||\boldsymbol{x}||\boldsymbol{y}||\boldsymbol{z})$ (if the variables come in a different order, we may always reorder the columns of A,B,C if necessary).

Soundness By construction, if $f(\boldsymbol{x}) = \boldsymbol{y}$ then $\mathsf{P}(A, B, C)$ accepts $(\boldsymbol{x}||\boldsymbol{y})$. It remains to show that if $\mathsf{P}(A, B, C)$ accepts $(\boldsymbol{x}||\boldsymbol{y})$, then $f(\boldsymbol{x}) = \boldsymbol{y}$. Straightforwardly from the definition, accepting means satisfying each row constraint, what in turn means that each multiplication subcircuit is computed correctly, and due to the facts that the entries that correspond to the same variable are located in the same column, and the same vector $\boldsymbol{w} = (1||\boldsymbol{x}||\boldsymbol{y}||\boldsymbol{z})$ is used in $A\boldsymbol{w}$, $B\boldsymbol{w}$, and $C\boldsymbol{w}$, there are no variable inconsistencies.

Example: A circuit that computes $2x_1 * x_2 + (x_1 + x_2) * x_2 + 1$ can be represented by a QAP of the form

$$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ y \\ z_1 \\ z_2 \end{pmatrix} \circ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ y \\ z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ y \\ z_1 \\ z_2 \end{pmatrix},$$

where z_1, z_2 are intermediate variables such that $z_1 = 2x_1 * x_2$, $z_2 = (x_1 + x_2) * x_2$, and y is the final result.

Complexity: Given an arithmetic circuit C with $s_m \in O(|C|)$ multiplication gates that computes a function $f : \mathbb{F}^s \to \mathbb{F}^t$, we may construct a QAP P with the following properties:

- $\mathsf{size}(\mathsf{P}) = s + t + s_m = s + t + O(|C|) = O(|C|)$ since the QAP uses all the input and the output variables, and additionally introduces a new variable for each intermediate multiplication gate.
- $\mathsf{sdim}(\mathsf{P}) = s_m = O(|C|)$ since each multiplication subcircuit is represented by one matrix row.

The relation $R_{\mathsf{P}(A,B,C)}$ of such a program is defined over pairs $(\boldsymbol{x}, \boldsymbol{w}) = \mathbb{F}^{s+t} \times \mathbb{F}^{s+t+s_m}$, where $\boldsymbol{x}$ (and hence the first $s+t$ entries of $\boldsymbol{w}$) contains information about the input and the output whose values need to be verified, and the rest s_m entries of $\boldsymbol{w}$ are the intermediate values that the prover would not like to show to the verifier. Therefore we need to construct a proof in which the verifier is convinced in the existence of $\boldsymbol{w}$, but does not see its value.

In general, the idea is that the prover wants to convince the verifier that he knows $\boldsymbol{w}$ such that $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$. A verifier sends to the prover a challenge that does not depend on the output. The prover has to respond, and the response should not reveal any private information contained in $\boldsymbol{w}$ (the $\boldsymbol{z}$ part in $(1||\boldsymbol{x}||\boldsymbol{y}||\boldsymbol{z})$). The requirement that $\boldsymbol{z}$ should remain private depends on the task. Hiding it would be definitely senseless if it contained only the intermediate gate valuations since the verifier could compute these by himself. However, since $\boldsymbol{z}$ may contain some additional input (used for example in solving NP-complete or privacy-preserving tasks), in general we would like to have a zero-knowledge proof: a proof that does not reveal *anything* except that $\boldsymbol{z}$ does exist.

We will do the following steps:

1. **Quadratic arithmetic program → linear probabilistically checkable proof:** allows to prove the knowledge of $\boldsymbol{w}$ without revealing anything about $\boldsymbol{z}$ on the assumption that for each query $\boldsymbol{q_1}, \ldots, \boldsymbol{q_k}$ sent by the verifier, the prover answers with $\langle \boldsymbol{\pi}, \boldsymbol{q_i} \rangle$ for the same vector $\boldsymbol{\pi}$. If we do not take into account this assumption, then the prover may cheat.
2. **Linear probabilistically checkable proof → linear interactive proof:** weakens the assumption and extends the prover's power to $\Pi(\boldsymbol{q_1}, \ldots, \boldsymbol{q_k})$ for an arbitrary affine function Π. The prover is however still linearly bounded, and if he uses something beyond affine functions, he may still cheat.
3. **Linear interactive proof → succinct non-interactive argument of knowledge:** weakens the assumption that the prover should be linear and allows him to respond with $f(\boldsymbol{q_1}, \ldots, \boldsymbol{q_k})$ for an arbitrary polynomial-time computable f. The polynomial-time bounded prover cannot cheat anymore.

3 QAP to Linear Probabilistically Checkable Proofs

A *probabilistically checkable proof* is something that can be used as a quicker way to verify the computation without repeating it from the beginning to the end, and without revealing the values of the intermediate gates. The idea is that the verifier sends a couple of randomly generated queries that do not depend on the input, and the prover has to answer to these queries. The queries are generated in such a way that if the prover does not know the proof, then he can give the correct answers with a very small probability.

In our settings, it is possible to do the verification in such a way that it is sufficient for an honest prover to respond with a certain *linear combination* of the obtained query elements. This makes it possible to make the construction simpler. Hence we will use the notion of *linear probabilistically checkable proof*. Its difference from the general definition of a probabilistically checkable proof is

that here the proof $\boldsymbol{\pi}$ provided by the prover has to be a vector over some field $\mathbb{F}$, but in general π could be any functionality.

Definition 4 (Linear Probabilistically Checkable Proof (PCP)). *[BCI+13] Let R be a binary relation, $\mathbb{F}$ a finite field, P a deterministic prover algorithm, and V a probabilistic oracle verifier algorithm. We say that the pair (P, V) is a k-query linear PCP for R over $\mathbb{F}$ with knowledge error ε and query length m if it satisfies the following requirements:*

Syntax: *On any input $\boldsymbol{x}$ and oracle access to a vector $\boldsymbol{\pi}$, the verifier $V^{\boldsymbol{\pi}}(\boldsymbol{x})$ makes k queries (that do not depend on $\boldsymbol{x}$) to $\boldsymbol{\pi}$ and then decides whether to accept or reject. More precisely, V consists of a probabilistic query algorithm Q and a deterministic decision algorithm D working as follows. Based on its internal randomness, and independently of $\boldsymbol{x}$, Q generates k queries $\boldsymbol{q_1}, \ldots, \boldsymbol{q_k} \in \mathbb{F}^m$ to $\boldsymbol{\pi}$ and state information $\boldsymbol{u}$. Then, given $\boldsymbol{x}$, $\boldsymbol{u}$, and the k oracle answers $\boldsymbol{a} = (a_1, \ldots, a_k) = (\langle \boldsymbol{\pi}, \boldsymbol{q_1} \rangle, \ldots, \langle \boldsymbol{\pi}, \boldsymbol{q_k} \rangle)$, D accepts or rejects.*

Completeness: *For every $(\boldsymbol{x}, \boldsymbol{w}) \in R$, the output of $\mathsf{P}(\boldsymbol{x}, \boldsymbol{w})$ is a description of a vector $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$ such that $V^{\boldsymbol{\pi}}(\boldsymbol{x})$ accepts with probability 1.*

Knowledge: *There exists a knowledge extractor E such that for every vector $\boldsymbol{\pi}^* : \mathbb{F}^m \to \mathbb{F}$, if the probability that $V^{\boldsymbol{\pi}^*}(\boldsymbol{x})$ accepts is greater than ε, then $E^{\boldsymbol{\pi}^*}(\boldsymbol{x})$ outputs $\boldsymbol{w}$ such that $(\boldsymbol{x}, \boldsymbol{w}) \in R$.*

Degree: *The pair (P, V) has* degree (d_Q, d_D) *if:*

1. *The query algorithm Q is computed by a degree d_Q arithmetic circuit (i.e there are k polynomials $p_1, \ldots, p_k$: $\mathbb{F}^\mu \to \mathbb{F}^m$ and state polynomial $p : \mathbb{F}^\mu \to \mathbb{F}^{m'}$, all of degree d_Q, such that the queries are $\boldsymbol{q_1} = p_1(r), \ldots, \boldsymbol{q_k} = p_k(r)$ and the state is $\boldsymbol{u} = p(r)$ for a random $r \in \mathbb{F}_\mu$, where m is the length of one query $\boldsymbol{q_i}$, and m' is the length of $\boldsymbol{u}$).*
2. *The decision algorithm D is computed by a degree d_D arithmetic circuit (i.e., for every input x there is a test polynomial $\boldsymbol{t} : \mathbb{F}^{m'+k} \to \mathbb{F}^\eta$ of degree d_D such that $\boldsymbol{t}(\boldsymbol{u}||\boldsymbol{a}) = 0^\eta$ iff $D(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a})$ accepts).*

The degree of the polynomials is a very important parameter for complexity since we do not want to have to compute large powers. In particular for our task, bounding the degree becomes essential when the multiplication has to be performed in an encrypted domain.

In our context, the prover will compute all the values $a_i = \langle \boldsymbol{\pi}, \boldsymbol{q_i} \rangle$ by itself. This means that the construction relies on assumption that the prover indeed applies the same vector $\boldsymbol{\pi}$ to each $\boldsymbol{q_i}$, and the only place where he may cheat is generating a malicious $\boldsymbol{\pi^*}$. This is the reason why it will have to be extended further.

3.1 The Linear PCP Construction

There can be various constructions of linear PCP algorithms. Here is presented the linear PCP from [BSCG+13]. Let $\mathsf{P}(A, B, C)$ be a QAP over $\mathbb{F}$ of size n and dimension m that accepts vectors $\boldsymbol{x} \in \mathbb{F}^k$. We will show that there exists a linear PCP $(P, (Q, D))$ for $R_{\mathsf{P}(A,B,C)}$ with the following properties:

- The number of queries is 5.
- The query length is $4 + m + n$.
- The state length is $k + 2$.
- The degree is $(d_Q, d_D) = (m, 2)$.
- The Q needs to sample only a single random element of $\mathbb{F}$ to generate the output queries and state.
- The knowledge error is $\frac{2m}{|\mathbb{F}|}$, and moreover, it is $\frac{m}{|\mathbb{F}|}$-statistical Honest Verifier Zero Knowledge (HVZK).

The main idea here is to define a *polynomial* QAP to which a common QAP can be easily extended.

Definition 5. *Consider some integers m, n, k such that $n - 1 \geq k$. A polynomial quadratic arithmetic program defined by vectors of $(m-1)$-degree polynomials $\boldsymbol{A}(x) = (A_0(x), \ldots, A_{n-1}(x))$, $\boldsymbol{B}(x) = (B_0(x), \ldots, B_{n-1}(x))$, $\boldsymbol{C}(x) = (C_0(x), \ldots, C_{n-1}(x))$ and an m-degree polynomial $Z(x)$ accepts a vector $\boldsymbol{x} \in \mathbb{F}^k$ iff there exists a vector $\boldsymbol{w} = (1, w_1, \ldots, w_{n-1})$ such that $(w_1, \ldots, w_k) = \boldsymbol{x}$ and*

$$Z(x) | \langle \boldsymbol{A}(x), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(x), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(x), \boldsymbol{w} \rangle \ ,$$

where $|$ denotes not pointwise divisibility, but divisibility of polynomials as formal structures.

The relation $R_{\mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x))}$ is defined as

$$(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x))} \iff \mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x)) \text{ accepts on } \boldsymbol{x} \ .$$

Proposition 1. *For any QAP $\mathsf{P}'(A, B, C)$ there exists a polynomial QAP $\mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x))$ such that*

$$(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}'(A,B,C)} \text{ iff } (\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x))} \ .$$

Proof. A polynomial quadratic arithmetic program $\mathsf{P}(\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x), Z(x))$ can be constructed from $\mathsf{P}'(A, B, C)$ as follows. Generate an arbitrary set of m points $S \subseteq \mathbb{F}$. Construct $\boldsymbol{A}(x), \boldsymbol{B}(x), \boldsymbol{C}(x)$ in such a way that $\boldsymbol{A}(s_i), \boldsymbol{B}(s_i), \boldsymbol{C}(s_i)$ are the i-th rows of A,B,C respectively. The $Z(x)$ is defined in such a way that $\forall s \in S : \ Z(s) = 0$.

$\Rightarrow$ If $A\boldsymbol{w} \circ B\boldsymbol{w} = C\boldsymbol{w}$, then $\langle \boldsymbol{A}(s), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(s), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(s), \boldsymbol{w} \rangle = 0$ for each $s \in S$, and hence $\langle \boldsymbol{A}(x), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(x), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(x), \boldsymbol{w} \rangle$ trivially divides $Z(x)$ since $Z(x)$ by definition has exactly the roots that are elements of S.

$\Leftarrow$ If $Z(x) | \langle \boldsymbol{A}(x), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(x), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(x), \boldsymbol{w} \rangle$, then $\langle \boldsymbol{A}(x), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(x), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(x), \boldsymbol{w} \rangle$ contains the same roots as $Z(x)$ does. Hence $\forall s \in S : \langle \boldsymbol{A}(s), \boldsymbol{w} \rangle \cdot \langle \boldsymbol{B}(s), \boldsymbol{w} \rangle - \langle \boldsymbol{C}(s), \boldsymbol{w} \rangle = 0$, what implies $A\boldsymbol{w} \circ B\boldsymbol{w} = C\boldsymbol{w}$. □

The polynomial representation is definitely more compact since it suffices to evaluate the polynomial on the points $s \in S$ to obtain various matrix rows. It allows to use probabilistic arguments of polynomial properties.

We will first show how the linear PCP algorithm is constructed, and then prove its properties one by one. The algorithms P,Q,D are presented separately. Additionally, there is a preprocessing phase that has to be performed once by some trusted third party (this is like key distribution before running the protocol).

Preprocessing: Denote $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ for $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, n-1\}$. Let $S = \{s_0, \ldots, s_{m-1}\} \subseteq \mathbb{F}$ be an arbitrary chosen subset of m points of $\mathbb{F}$.

Let A_j, B_j, C_j for $j \in \{0, \ldots, n-1\}$ be polynomials of degree $m-1$ defined in such a way that $A_j(s_i) = a_{ij}$, $B_j(s_i) = b_{ij}$, $C_j(s_i) = c_{ij}$. The coefficients of these polynomials can be computed by interpolation, and they are of degree $m-1$ since they are defined on m points. Let $\boldsymbol{A}(x) := (A_0(x), \ldots, A_{n-1}(x))$, $\boldsymbol{B}(x) := (B_0(x), \ldots, B_{n-1}(x))$, $\boldsymbol{C}(x) := (C_0(x), \ldots, C_{n-1}(x))$ denote the vectors of corresponding polynomials.

Let Z_S be an m-degree polynomial over $\mathbb{F}$ such that $\forall s \in S: \ Z_S(s) = 0$. The simplest solution is to define $Z_S(x) = \prod_{s \in S}(x - s)$. In this way, $Z_S(x)$ has exactly m roots which are the elements of S.

The set S and the coefficients of $A_j(x)$, $B_j(x)$, $C_j(x)$, $Z_S(x)$ are published.

Linear PCP Prover $\boldsymbol{P(x, w)}$: Let $\boldsymbol{x} \in \mathbb{F}^k$, $\boldsymbol{w} \in \mathbb{F}^n$. The prover works as follows:

- Let $\delta_A, \delta_B, \delta_C \in \mathbb{F}$ be random field elements.
- Let $A(x)$, $B(x)$, $C(x)$ be polynomials of degree m such that:

$$A(x) := \langle \boldsymbol{w}, \boldsymbol{A}(x) \rangle + \delta_A Z_S(x) \ ;$$

$$B(x) := \langle \boldsymbol{w}, \boldsymbol{B}(x) \rangle + \delta_B Z_S(x) \ ;$$

$$C(x) := \langle \boldsymbol{w}, \boldsymbol{C}(x) \rangle + \delta_C Z_S(x) \ ;$$

  where the degree m comes from the fact that the degree of each polynomial in $\boldsymbol{A}(x)$, $\boldsymbol{B}(x)$, $\boldsymbol{C}(x)$ is $m-1$, and the degree of $Z_S(x)$ is m.
- Let $\boldsymbol{h} = (h_0, \ldots, h_m)$ be the coefficients of the polynomial

$$H(x) := \frac{A(x)B(x) - C(x)}{Z_S(x)} \ .$$

The algorithm returns the proof $\boldsymbol{\pi} = ((\delta_A, \delta_B, \delta_C) || \boldsymbol{w} || \boldsymbol{h})$.

The question is whether $H(x)$ exists at all since it may be that $A(x)B(x) - C(x)$ is not divisible by $Z_S(x)$. We need to show that at least for $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$, such an $H(x)$ exists. By definition,

$$\begin{aligned} A(x)B(x) - C(x) &= (\langle \boldsymbol{w}, \boldsymbol{A}(x) \rangle + \delta_A Z_S(x))(\langle \boldsymbol{w}, \boldsymbol{B}(x) \rangle + \delta_B Z_S(x)) - \langle \boldsymbol{w}, \boldsymbol{C}(x) \rangle - \delta_C Z_S(x) \\ &= \langle \boldsymbol{w}, \boldsymbol{A}(x) \rangle \langle \boldsymbol{w}, \boldsymbol{B}(x) \rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x) \rangle + Z_S(x) D \ , \end{aligned}$$

where $D = \delta_B \langle \boldsymbol{w}, \boldsymbol{A}(x) \rangle + \delta_A \langle \boldsymbol{w}, \boldsymbol{B}(x) \rangle + \delta_A \delta_B Z_S(x) - \delta_C$. It suffices to prove that $\langle \boldsymbol{w}, \boldsymbol{A}(x) \rangle \langle \boldsymbol{w}, \boldsymbol{B}(x) \rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x) \rangle$ divides $Z_S(x)$.

- For $s \in S$, by definition of $\boldsymbol{A}(s)$, $\boldsymbol{B}(s)$, $\boldsymbol{C}(s)$, it suffices to prove that $A\boldsymbol{w} \circ B\boldsymbol{w} - C\boldsymbol{w} = \boldsymbol{0}$, and this is indeed true if $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$.

– For $s \notin S$ we cannot claim that straightforwardly. However, since $Z_S(x)$ is an m-degree polynomial that is not 0 everywhere, it has exactly m roots, and it can be written as $c\prod_{s\in S}(x-s)$ for some constant c. For $s \in S$ we have $\langle \boldsymbol{w}, \boldsymbol{A}(s)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(s)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(s)\rangle = 0$, so $\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle$ divides $(x-s)$. Since $(x-s)$ are all coprime, we may as well say that $\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle$ divides the product $\prod_{s\in S}(x-s)$, and hence divides $Z_S(x)$.

Linear PCP Verifier $\boldsymbol{V = (Q(), D(x, u, a))}$: The work of the verifier is split into two parts: query algorithm $Q()$ and decision algorithm D.

– $Q()$: First of all, a random element $\tau \in \mathbb{F}$ is generated. Then the following queries $\boldsymbol{q_i} \in \mathbb{F}^{3+m+(n+1)=4+m+n}$ are computed:
  1. $\boldsymbol{q_1} = ((Z_S(\tau), 0, 0)||\boldsymbol{A}(\tau)||(0, 0, \ldots, 0))$;
  2. $\boldsymbol{q_2} = ((0, Z_S(\tau), 0)||\boldsymbol{B}(\tau)||(0, 0, \ldots, 0))$;
  3. $\boldsymbol{q_3} = ((0, 0, Z_S(\tau))||\boldsymbol{C}(\tau)||(0, 0, \ldots, 0))$;
  4. $\boldsymbol{q_4} = ((0, 0, 0)||(0, 0, \ldots, 0, 0, \ldots, 0)||(1, \tau, \ldots, \tau^m))$;
  5. $\boldsymbol{q_5} = ((0, 0, 0)||(1, \tau, \ldots, \tau^k, 0, \ldots, 0)||(0, 0, \ldots, 0))$.

  The state information is $\boldsymbol{u} := (1, \tau, \tau^2, \ldots, \tau^k, Z_S(\tau))$. The query results are $a_i = \langle \boldsymbol{\pi}, \boldsymbol{q_i}\rangle$ for $i \in \{1, \ldots, 5\}$.
  According to the description, the degree of Q is $d_Q = m$ since it deals with polynomials of degree at most m.
– $D(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a})$: Let $x = (x_1, \ldots, x_k)$, $u = (u_1, \ldots, u_{k+2})$, $\boldsymbol{a} = (a_1, \ldots, a_5)$. The algorithm accepts iff:
  1. $a_1 a_2 - a_3 - a_4 u_{k+2} = 0$,
  2. $a_5 - u_1 - \langle \boldsymbol{x}, (u_2, \ldots, u_{k+1})\rangle = 0$.

According to the description, the degree of D is $d_D = 2$ since it has to deal with multiplications of at most two values.

3.2 Properties

Since it may be unclear what the verifier does exactly, we will prove correctness and knowledge properties in more details.

Correctness: Let $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$. We need to show that in this case the verifier accepts with probability 1. Let $\boldsymbol{\tau} = (\tau, \ldots, \tau^k)$.

If the input is provided by a honest verifier, then $w_0 = 1$, $w_i = x_i$ for $i \in \{1, \ldots, k\}$, and the answers to the queries are constructed as follows:

– $a_1 = \langle \boldsymbol{\pi}, \boldsymbol{q_1}\rangle = (\delta_A Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{A}(\tau)\rangle)$;
– $a_2 = \langle \boldsymbol{\pi}, \boldsymbol{q_2}\rangle = (\delta_B Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{B}(\tau)\rangle)$;
– $a_3 = \langle \boldsymbol{\pi}, \boldsymbol{q_3}\rangle = (\delta_C Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{C}(\tau)\rangle)$;
– $a_4 = \langle \boldsymbol{\pi}, \boldsymbol{q_4}\rangle = h_0 + h_1\tau + h_2\tau^2 + \ldots + h_m\tau^m = H(\tau)$;
– $a_5 = \langle \boldsymbol{\pi}, \boldsymbol{q_5}\rangle = w_0 + w_1\tau + w_2\tau^2 + \ldots + w_k\tau^k = 1 + \langle \boldsymbol{x}, \boldsymbol{\tau}\rangle$.

Let us consider both conditions that the verifier checks:

$$\begin{aligned}
(1) &= a_1 a_2 - a_3 - a_4 u_{k+2} \\
&= \langle \boldsymbol{\pi}, \boldsymbol{q_1} \rangle \langle \boldsymbol{\pi}, \boldsymbol{q_2} \rangle - \langle \boldsymbol{\pi}, \boldsymbol{q_3} \rangle - \langle \boldsymbol{\pi}, \boldsymbol{q_4} \rangle Z_S(\tau) \\
&= (\delta_A Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{A}(\tau) \rangle)(\delta_B Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{B}(\tau) \rangle) - (\delta_C Z_S(\tau) + \langle \boldsymbol{w}, \boldsymbol{C}(\tau) \rangle) - H(\tau) Z_S(\tau) \\
&= H(\tau) Z_S(\tau) - H(\tau) Z_S(\tau) \\
&= 0 \ ; \\
(2) &= a_5 - u_1 - \langle \boldsymbol{x}, (u_2, \ldots, u_{k+1}) \rangle \\
&= \langle \boldsymbol{\pi}, \boldsymbol{q_5} \rangle + 1 - 1 - \langle \boldsymbol{x}, \boldsymbol{\tau} \rangle \\
&= \langle \boldsymbol{x}, \boldsymbol{\tau} \rangle + 1 - 1 - \langle \boldsymbol{x}, \boldsymbol{\tau} \rangle \\
&= 0 \ .
\end{aligned}$$

Hence if the prover is honest, then the verifier definitely accepts.

Knowledge: Let $(\boldsymbol{x}, \boldsymbol{w}) \notin R_{\mathsf{P}(A,B,C)}$. We need to show that in this case the verifier accepts with probability at most $\frac{2m}{|\mathbb{F}|}$. In other words, we need to show that, for any $\boldsymbol{\pi^*}$, if the verifier accepts with probability greater than $\frac{2m}{|\mathbb{F}|}$, then there exists some $(\boldsymbol{x}, \boldsymbol{w^*}) \in R_{\mathsf{P}(A,B,C)}$ from which $\boldsymbol{\pi^*}$ can be extracted, what implies that the $\boldsymbol{\pi^*}$ is actually correct, and there indeed exists a witness $\boldsymbol{w^*}$.

Suppose the prover has come up with $\boldsymbol{\pi^*} = ((\delta_A^*, \delta_B^*, \delta_C^*) || \boldsymbol{w^*} || \boldsymbol{h^*})$ such that

$$\Pr_{\tau \leftarrow \mathbb{F}}[\text{Verifier accepts}] > \frac{2m}{|\mathbb{F}|} \ .$$

This means that the verifier accepts with probability $> \frac{2m}{|\mathbb{F}|}$ the following two conditions:

$$\begin{aligned}
0 &= a_1 a_2 - a_3 - a_4 u_{k+2} \\
&= \langle \boldsymbol{\pi^*}, \boldsymbol{q_1} \rangle \langle \boldsymbol{\pi^*}, \boldsymbol{q_2} \rangle - \langle \boldsymbol{\pi^*}, \boldsymbol{q_3} \rangle - \langle \boldsymbol{\pi^*}, \boldsymbol{q_4} \rangle Z_S(\tau) \\
&= (\delta_A^* Z_S(\tau) + \langle \boldsymbol{w^*}, \boldsymbol{A}(\tau) \rangle)(\delta_B^* Z_S(\tau) + \langle \boldsymbol{w^*}, \boldsymbol{B}(\tau) \rangle) - (\delta_C^* Z_S(\tau) + \langle \boldsymbol{w^*}, \boldsymbol{C}(\tau) \rangle) - H^*(\tau) Z_S(\tau) \ , \\
0 &= a_5 - u_1 - \langle \boldsymbol{x}, (u_2, \ldots, u_{k+1}) \rangle \\
&= \langle \boldsymbol{w^*}, (1 || \boldsymbol{\tau}) \rangle - 1 - \langle \boldsymbol{x}, \boldsymbol{\tau} \rangle \ .
\end{aligned}$$

We may consider both these expressions as evaluations of polynomials on the point τ. Here we use Schwartz-Zippel lemma, which states that the probability of choosing an appropriate τ for an arbitrary polynomial is bounded by $\frac{2m}{|\mathbb{F}|}$, where $2m$ comes from the the upper bound of degree of $Z_S(x)H(x)$. The condition $\deg(H) \leq m$ is guaranteed by the length of the proof π (the prover can increase the degree just by sending more coefficients and hence increasing the proof length), and $\deg(Z_S) = m$ since it was chosen in such a way in the preprocessing phase by a Trusted Third Party. Since we assumed that the probability is strictly larger than $\frac{2m}{|\mathbb{F}|}$, both polynomials should be zero polynomials, which implies

$$\begin{aligned}
(\delta_A^* Z_S(z) + \langle \boldsymbol{w^*}, \boldsymbol{A}(z) \rangle)(\delta_B^* Z_S(z) + \langle \boldsymbol{w^*}, \boldsymbol{B}(z) \rangle) - (\delta_C^* Z_S(z) + \langle \boldsymbol{w^*}, \boldsymbol{C}(z) \rangle) - H^*(z) Z_S(z) = 0 \ , \\
\langle \boldsymbol{w^*}, (1 || (z, z^2, \ldots, z^k)) \rangle - 1 - \langle \boldsymbol{x}, (z, z^2, \ldots, z^k) \rangle = 0 \ ,
\end{aligned}$$

where z is a formal variable. Now we can conclude that the equalities hold coefficientwise. It immediately follows that $w_0^* = 1$ and $(w_1^*, \ldots, w_k^*) = \boldsymbol{x}$.

Consider evaluation on any $s \in S$. Since by definition $Z_S(s) = 0$, it follows that

$$\langle \boldsymbol{w^*}, \boldsymbol{A}(s)\rangle\langle \boldsymbol{w^*}, \boldsymbol{B}(s)\rangle - \langle \boldsymbol{w^*}, \boldsymbol{C}(s)\rangle = 0 \ .$$

This holds for any $s \in S$. Recall that S was defined in a way that $\boldsymbol{A}(s)$, $\boldsymbol{B}(s)$, $\boldsymbol{C}(s)$ correspond to some i-th row of matrices A,B,C, and the entire S defines all the rows. Since $\langle \boldsymbol{w^*}, \boldsymbol{A}(s)\rangle\langle \boldsymbol{w^*}, \boldsymbol{B}(s)\rangle - \langle \boldsymbol{w^*}, \boldsymbol{C}(s)\rangle = 0$ for all $s \in S$, we get $\boldsymbol{w^*}^{\mathrm{T}} A^{\mathrm{T}} \circ \boldsymbol{w^*}^{\mathrm{T}} B^{\mathrm{T}} - \boldsymbol{w^*}^{\mathrm{T}} C^{\mathrm{T}} = \boldsymbol{0}$. We may as well write $A\boldsymbol{w^*} \circ B\boldsymbol{w^*} - C\boldsymbol{w^*} = \boldsymbol{0}$. Hence $(\boldsymbol{x}, \boldsymbol{w^*}) \in R_{\mathsf{P}(A,B,C)}$.

Honest Verifier Zero-Knowledge Intuitively, we need to show that the answers $a_1, \ldots, a_5$ sent by the prover do not reveal any information about $\boldsymbol{w}$. Suppose that the verifier is honest and indeed chooses τ randomly. According to Schwartz-Zippel lemma, for a random τ it happens that $Z_S(\tau) \neq 0$ with probability $\frac{m}{|\mathbb{F}|}$. Since δ_A, δ_B, δ_C are selected uniformly and independently at random from $\mathbb{F}$, the values a_1, a_2, a_3 are uniform and independent field elements in $\mathbb{F}$. Also, a_4 is determined by a_1, a_2, a_3, u_{n+2}, neither of which is dependent on $\boldsymbol{w}$. The last element a_5 contains information only about the part of $\boldsymbol{w}$ that is equal to $\boldsymbol{x}$, which is known to the verifier anyway.

Let us now prove it more formally. We need to show that the proof reveals nothing more than the correctness of the statement. For this purpose we show that it is possible to simulate the proof for a valid statement without knowing a witness (but knowing some certain trapdoor). The formal definition of zero-knowledge proof (based on [GGPR13] definition) is the following.

Definition 6. *There exist a polynomial-time simulator S and a trapdoor* trap *such that for any distinguishing algorithm A and auxiliary input z,*

$$\Pr\left[\begin{matrix}(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)} \\ A(\boldsymbol{a}) = 1\end{matrix}\middle| \begin{matrix}(\boldsymbol{x}, \boldsymbol{w}) \leftarrow A(z) \\ \boldsymbol{a} \leftarrow P(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{q_1}, \ldots, \boldsymbol{q_5})\end{matrix}\right] =$$

$$\Pr\left[\begin{matrix}(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)} \\ A(\boldsymbol{a}) = 1\end{matrix}\middle| \begin{matrix}(\boldsymbol{x}, \boldsymbol{w}) \leftarrow A(z) \\ \boldsymbol{a} \leftarrow S(\boldsymbol{x}, \mathsf{trap}, \boldsymbol{q_1}, \ldots, \boldsymbol{q_5})\end{matrix}\right]$$

According to the previous discussion, we may define S that takes $\mathsf{trap} = \tau$ which allows him to generate the entire $\boldsymbol{u}$. It draws a_1, a_2, a_3 randomly from $\mathbb{F}$, without even looking at $\boldsymbol{q_i}$. He then computes $a_4 = (a_1 a_2 - a_3)/u_{k+2}$ and $a_5 = u_1 + \langle \boldsymbol{x}, (u_2, \ldots, u_{k+1})\rangle$. By definition of a_4 and a_5, both checks pass with the same probability as for a real prover P as the elements of $\boldsymbol{a}$ come from exactly the same distribution. The only statistical difference comes from the fact that with probability $\frac{m}{|\mathbb{F}|}$ it may happen that $\tau \in S$, then $u_{k+2} = Z_S(\tau) = 0$, and the division $(a_1 a_2 - a_3)/u_{k+2}$ fails. Hence the proposed linear PCP is an $\frac{m}{|\mathbb{F}|}$ honest verifier zero knowledge.

3.3 Computational Algorithms and Their Complexities

The particular implementation of the previously defined Linear PCP is also proposed in [BSCG+13], together with complexity estimations.

- $P(\boldsymbol{x}, \boldsymbol{w})$: $O(\mathsf{sdim}(\mathsf{P}) \log(\mathsf{sdim}(\mathsf{P})) + \mathsf{supp}(\mathsf{P}))$.
- $Q()$: $O(\mathsf{sdim}(\mathsf{P}) + \mathsf{supp}(\mathsf{P}))$
- $D(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a})$: $O(|\boldsymbol{x}|)$

The efficiency enhancements are based on Fast Fourier Transform for polynomial multiplication. Hence we assume for simplicity that m is a power of 2 (additional dummy easily satisfiable constraints may be added to A,B,C), and there exists a principal m-th root of unity ω. Let $S = (1, \omega, \ldots, \omega^{m-1})$ and $Z_S(x) = \prod_{i=0}^{m-1}(x - \omega^i) = x^m - 1$. This choice of $Z_S(x)$ is correct since for any $i \in \{0, \ldots, m-1\}$: $(\omega^i)^m = 1$, and it is simple enough to be easily computed.

$\boldsymbol{P(x, w)}$: the main overhead comes from finding the coefficients of $H(x)$. This algorithm uses Fast Fourier Transform for both evaluation and interpolation. Let FFT denote the evaluation, and FFT^{-1} interpolation. Define an additional set of points $T = tS = \{t, t\omega, \ldots, t\omega^{m-1}\}$ for some $t \in \mathbb{F} \setminus S$. The idea is to do do the computation in the following order:

1. Evaluate (straightforwardly from A,B,C) $\langle \boldsymbol{w}, \boldsymbol{A}(\omega^i)\rangle$, $\langle \boldsymbol{w}, \boldsymbol{B}(\omega^i)\rangle$, $\langle \boldsymbol{w}, \boldsymbol{C}(\omega^i)\rangle$ for $i \in \{0, \ldots, m-1\}$. The coordinates whose values are 0 can be omitted since their locations are known by construction of QAP (can be precomputed in the preprocessing phase if necessary), and hence the complexity of this step is $O(\mathsf{supp}(\mathsf{P}))$.
2. Compute the coefficients of $\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle$, $\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle$, $\langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle$ using FFT^{-1} on the evaluations on S that we have computed in the previous step. It requires $O(m \log m)$ steps.
3. Evaluate $\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle$,$\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle$,$\langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle$,$Z_S(x)$ on T. A suitable choice of $Z_S(x)$ is $x^m - 1$, so its evaluation by fast exponentiation requires $\log m + 1$ steps. The other three polynomials can all be evaluated using FFT in $O(m \log m)$ steps.
4. Compute the coefficients of $\frac{\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle}{Z_S(x)}$ by using FFT^{-1} on the evaluations on T. Note that it is the place where we actually need T since we could not divide by $Z_S(s)$ using $s \in S$ due to the definition of $Z_S(x)$. Since $T = tS$, the FFT used for T differs from S just by an extra multiplication by a diagonal matrix whose diagonal consists of the elements $(1, t, \ldots, t^{m-1})$, so although the elements of T are not roots of unity, the FFT still takes $O(m \log m)$ steps.
5. Compute straightforwardly

$$
\begin{aligned}
H(x) &= \frac{\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle}{Z_S(x)} + \delta_A\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle + \delta_B\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle + \delta_A\delta_B Z_S(x) - \delta_C \\
&= \frac{\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle + \delta_A\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle Z_S(x) + \delta_B\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle Z_S(x) + \delta_A\delta_B Z_S(x)^2 - \delta_C Z_S(x)}{Z_S(x)} \\
&= \frac{(\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle + \delta_A Z_S(x))(\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle + \delta_B Z_S(x)) - (\langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle + \delta_C Z_S(x))}{Z_S(x)} \ .
\end{aligned}
$$

This requires $O(m \log m)$ steps since the degrees of all the polynomials are at most m, and we have already shown that computing $\frac{\langle \boldsymbol{w}, \boldsymbol{A}(x)\rangle\langle \boldsymbol{w}, \boldsymbol{B}(x)\rangle - \langle \boldsymbol{w}, \boldsymbol{C}(x)\rangle}{Z_S(x)}$ requires $O(m \log m)$ steps.

In total, the prover works in time $O(m \log m) + O(\mathsf{supp}(\mathsf{P})) = O(\mathsf{sdim}(\mathsf{P}) \log(\mathsf{sdim}(\mathsf{P}))) + O(\mathsf{supp}(\mathsf{P}))$.

$\boldsymbol{Q}()$: the complexity is dominated by evaluating each $\boldsymbol{A}(\tau)$, $\boldsymbol{B}(\tau)$, $\boldsymbol{C}(\tau)$. Using FFT efficiently is not possible since τ should be random and we cannot force it to be a root of unity or something similar. However, an algorithm of just $m + \log m + 3 \cdot \mathsf{supp}(\mathsf{P})$ steps is proposed.

In general, the coefficients of any $A_i(x) \in \boldsymbol{A}(x)$ can be computed by Lagrange interpolation:

$$A_i(x) = \sum_{j=1}^{m} A_i(\alpha_j) \prod_{k \neq j} \frac{x - \alpha_k}{\alpha_j - \alpha_k} \ .$$

Since $Z_S(x) = x^m - 1 = (x-1)(x-\omega)\dots(x-\omega^{m-1}) = \prod_{j=1}^{n}(x - \alpha_j)$, we may as well write

$$A_i(x) = \sum_{j=1}^{m} A_i(\alpha_j) \frac{Z_S(x)}{(x-\alpha_j)\prod_{k\neq j}(\alpha_j - \alpha_k)} \ .$$

If we order S for example in a way $\alpha_i = \omega^{i-1}$, then:

$$A_i(x) = \sum_{j=1}^{m} a_{ij} \frac{Z_S(x)}{(x-\omega^{j-1})\prod_{k\neq j}(\omega^{j-1} - \omega^{k-1})} \ .$$

The enhancement comes from the fact that $\frac{Z_S(x)}{\prod_{k\neq j}(\omega^{j-1} - \omega^{k-1})} =: L_j(x)$ does not have to be computed separately for each j. Note that $\alpha_{j+1} = \omega\alpha_j$. We have

$$\prod_{k\neq(j+1)} (\omega^j - \omega^{k-1}) = \prod_{k\neq(j+1)} \omega(\omega^{j-1} - \omega^k) = \omega^{m-1} \prod_{k\neq j} (\omega^{j-1} - \omega^{k-1})$$

Hence $L_{j+1}(x) = \omega^{-1} L_j(x)$, and it is sufficient to compute only $L_1(x)$ and then spend additional $m-1$ multiplications by ω^{-1}.

The value $L_1(x)$ can also be computed efficiently. Here we use the fact that

$$(1 + x + \ldots + x^{m-1}) = \frac{x^m - 1}{x - 1} = (x-\omega)\dots(1-\omega^{m-1}) \ .$$

Hence if $x = 1$, then $m = 1^0 + 1^1 + 1^2 + \ldots + 1^{m-1} = (1-\omega)\ldots(1-\omega^{m-1})$.

$$\begin{aligned} L_1(x) &= \frac{Z_S(x)}{\prod_{k\neq 1}(\omega^0 - \omega^{k-1})} \\ &= \frac{Z_S(x)}{\prod_{k\neq 1}(1 - \omega^{k-1})} \\ &= \frac{Z_S(x)}{(1-\omega)\ldots(1-\omega^{m-1})} \\ &= \frac{Z_S(x)}{m} = Z_S(x)\cdot m^{-1} \ . \end{aligned}$$

Hence all the $L_j(\tau)$ can be computed in $\log m + m$ steps (one evaluation of $Z_S(\tau)$ and m steps to generate all the $L_j(\tau)$ from $L_1(\tau)$). Each $A_i(\tau)$ is a sum of m such terms, but we may omit the multiplications with zeroes since their locations are known. Since $B_i(\tau)$ and $C_i(\tau)$ use the same $L_j(\tau)$, we need $3 \cdot \mathsf{supp}(\mathsf{P})$ steps to compute A_i,B_i,C_i for all i. The total complexity is $O(m + \mathsf{supp}(\mathsf{P})) = O(\mathsf{sdim}(\mathsf{P}) + \mathsf{supp}(\mathsf{P}))$.

$\boldsymbol{D(x, u, a)}$**:** performs only $2k + 9$ group operations and hence is already very efficient. Since $k = |\boldsymbol{x}|$, the complexity is $O(|\boldsymbol{x}|)$.

4 From HVZK Linear PCP to HVZK Two-Message Linear Interactive Proof

The transition from linear PCP to a linear interactive proof is described according to [BCI+13]. The description is no longer dependent on the particular Linear PCP presented in the previous subsection, and is more general. In this subsection we will use k to denote the number of queries of Linear PCP (in the previous construction it was 5). We will use m to denote the length of a single query (in the previous construction it was $4 + \mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P})$). The values $\alpha_1, \ldots, \alpha_k, \alpha_{k+1}$ will denote scalar coefficients.

Definition 7 (Two-Message Linear Interactive Proof). *A two-message linear interactive proof is a pair (P, V) defined similarly to* PCP*, but where the verifier is not given access to $\boldsymbol{\pi}$. Instead, the verifier sends $\boldsymbol{q} = (\boldsymbol{q_1}||\ldots||\boldsymbol{q_k})$ to the prover as a single message. The prover responds with $\boldsymbol{a} = (a_1, \ldots, a_k) = \Pi(\boldsymbol{q_1}||\ldots||\boldsymbol{q_k})$ that is also sent as a single message, where $\Pi : \mathbb{F}^{km} \to \mathbb{F}^k$ is an affine function.*

The Linear Interactive Proof definition is stronger than Linear PCP. In the Linear PCP definition, we assumed that the prover applies the same $\boldsymbol{\pi}$ to each $\boldsymbol{q_i}$ since formally the verifier V made an oracle query to $\boldsymbol{\pi}$. Actually, it may happen that the malicious verifier cheats by applying different $\boldsymbol{\pi_i}$ to different $\boldsymbol{q}_i$. In general, he may apply an arbitrary affine transformation Π to $\boldsymbol{q}$ and return $\boldsymbol{a} = \Pi(\boldsymbol{q})$ (he cannot do anything beyond affine since by assumption he is linearly bounded).

Therefore an additional consistency check is needed to transform a linear PCP to a linear interactive proof. There is a small modification that checks this.

- The Verifier additionally generates $\boldsymbol{\alpha} := (\alpha_1, \ldots, \alpha_k) \xleftarrow{\$} \mathbb{F}^k$, and an additional query $\boldsymbol{q_{k+1}} := \alpha_1 \boldsymbol{q_1} + \ldots + \alpha_k \boldsymbol{q_k}$. It sends $(\boldsymbol{q} || \boldsymbol{q_{k+1}})$ to the prover.
- The prover computes $a_i = \langle \boldsymbol{\pi}, \boldsymbol{q_i} \rangle$ for $i \in \{1, \ldots, k, k+1\}$. Let $\boldsymbol{a} = (a_1, \ldots, a_k)$. The prover sends $(\boldsymbol{a} || a_{k+1})$ to the verifier.
- In addition to the Linear PCP verification described in the previous subsection, the verifier checks that $a_{k+1} = \langle \boldsymbol{\alpha}, \boldsymbol{a} \rangle$.

The idea is that it is difficult for the prover to succeed in the last verification unless it applies the same vector $\boldsymbol{\pi}$ to each query. Let us prove it more formally.

Suppose $(P, (Q, D))$ is a k-query Linear PCP for a relation R over $\mathbb{F}$ with query length m and knowledge error ε. Then the previously described Linear Interactive Proof for R over $\mathbb{F}$ $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ has the following properties:

- the knowledge error of $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ is $\varepsilon + \frac{2}{|\mathbb{F}|}$;
- if $(P, (Q, D))$ is an HVZK, then $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ is also HVZK;
- if $(P, (Q, D))$ has a verifier of degree (d_Q, d_D), then $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ has one with degree $(d_Q, \max(2, d_D))$.

Completeness: If the prover is honest, then:

$$\begin{aligned} \langle \pi, \boldsymbol{q_{k+1}} \rangle &= \langle \pi, \alpha_1 \boldsymbol{q_1} + \ldots + \alpha_k \boldsymbol{q_k} \rangle \\ &= \alpha_1 \langle \pi, \boldsymbol{q_1} \rangle + \ldots + \alpha_k \langle \pi, \boldsymbol{q_k} \rangle \\ &= \langle \boldsymbol{\alpha}, \boldsymbol{a} \rangle \ . \end{aligned}$$

Knowledge: Suppose that the malicious prover does not use a single $\boldsymbol{\pi}$. Instead, he has some affine function Π such that $\boldsymbol{a} = \Pi(\boldsymbol{q})$. Since the function is affine, we may as well write that there are some $(k+1)$ vectors $\boldsymbol{\pi_{i,j}} \in \mathbb{F}^m$ and $\gamma_i \in \mathbb{F}$ for $i, j \in \{1, \ldots, k+1\}$ such that $a_i = \sum_{j=1}^{k+1} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i$.

Now suppose that the prover has come up with some $\boldsymbol{\pi_{i,j}}$, γ_i such that

$$Pr_{(\alpha_1, \ldots, \alpha_k) \xleftarrow{\$} \mathbb{F}^k}[\text{Verifier accepts the last check}] > \frac{2}{\mathbb{F}} \ .$$

We show that in this case $a_i = \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_i} \rangle$ for any $i \in \ldots 1, \ldots, k$. Suppose by contrary that there exists some i^* such that $a_{i^*} \neq \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_{i^*}} \rangle$. If the consistency check has passed, it means that

$$\sum_{i=1}^{k} \alpha_i \left(\sum_{j=1}^{k+1} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i \right) = \sum_{j=1}^{k+1} \langle \boldsymbol{\pi_{k+1,j}}, \boldsymbol{q_j} \rangle + \gamma_{k+1} \ ,$$

or in other words

$$\sum_{i=1}^{k} \alpha_i \left(\sum_{j=1}^{k+1} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i \right) - \sum_{j=1}^{k+1} \langle \boldsymbol{\pi_{k+1,j}}, \boldsymbol{q_j} \rangle - \gamma_{k+1} = 0 \ .$$

We may now consider this expression as a multivariate two-degree polynomial with the variables $\alpha_1, \ldots, \alpha_k$. By using the equality $\boldsymbol{q_{k+1}} = \sum_{i=1}^{k} \alpha_i \boldsymbol{q_i}$, the previous expression can be rewritten as

$$\sum_{i=1}^{k} \alpha_i \left(\sum_{j=1}^{k} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \langle \boldsymbol{\pi_{k+1,k+1}}, \sum_{j=1}^{k} \alpha_j \boldsymbol{q_j} \rangle + \gamma_i \right) - \sum_{j=1}^{k} \langle \boldsymbol{\pi_{k+1,j}}, \boldsymbol{q_j} \rangle - \langle \boldsymbol{\pi_{k+1,k+1}}, \sum_{i=1}^{k} \alpha_i \boldsymbol{q_i} \rangle - \gamma_{k+1} = 0 \ ,$$

which is equivalent to

$$\sum_{i=1}^{k} \alpha_i \left(\sum_{j=1}^{k} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \sum_{j=1}^{k} \alpha_j \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_j} \rangle + \gamma_i \right) - \sum_{j=1}^{k} \langle \boldsymbol{\pi_{k+1,j}}, \boldsymbol{q_j} \rangle - \sum_{i=1}^{k} \alpha_i \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_i} \rangle - \gamma_{k+1} = 0 \ .$$

Separating the groups $\alpha_i \alpha_j$ and α_i from the constant terms, we get

$$\sum_{i,j=1}^{k} \alpha_i \alpha_j \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_j} \rangle + \sum_{i=1}^{k} \alpha_i \left(\sum_{j=1}^{k} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i - \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_i} \rangle \right) - \sum_{j=1}^{k} \langle \boldsymbol{\pi_{k,j}}, \boldsymbol{q_j} \rangle - \gamma_{k+1} = 0 \ .$$

From this equation we can see that the coefficient of each α_i in this polynomial is $\sum_{j=1}^{k} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i - \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_i} \rangle$. This value equals 0 iff $\sum_{j=1}^{k} \langle \boldsymbol{\pi_{i,j}}, \boldsymbol{q_j} \rangle + \gamma_i = \langle \boldsymbol{\pi_{k+1,k+1}}, \boldsymbol{q_i} \rangle$. Hence the coefficient of α_{i^*} for an incorrectly defined a_{i^*} is not 0, so the entire polynomial is not a zero polynomial, and by Schwartz-Zippel lemma the probability that the expression equals 0 is $\frac{2}{\mathbb{F}}$.

If the prover is dishonest, then he fools either the Linear PCP verifier or this last check of consistency of $\boldsymbol{\pi}$, Hence:

$$\begin{aligned} \Pr[\text{Dishonest prover succeeds}] &= \Pr[\text{Succeeds in PCP or Succeeds in the } \boldsymbol{\pi} \text{ check}] \\ &\leq \Pr[\text{Succeeds in the PCP check}] + \Pr[\text{Succeeds in the } \boldsymbol{\pi} \text{ check}] \\ &\leq \varepsilon + \frac{2}{\mathbb{F}} \ . \end{aligned}$$

Honest Verifier Zero Knowledge: comes from the fact that the corresponding Linear PCP was a HVZK.

Complexity overheads: Let us now see how the obtained Linear Interactive Proof is related to the initial QAP P. Since $\tilde{Q}$ generates just one more query as a linear combination of the existing 5 queries of length $4 + \mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P})$, its asymptotic complexity does not change. In [BSCG+13] it was stated that we may omit $O(\mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P}))$ since the value $O(\mathsf{supp}(\mathsf{P}) \log \mathsf{supp}(\mathsf{P}))$ is larger for the proposed QAP for circuits, that's why it has not been included into complexities of P and Q before. The proof construction algorithm $\tilde{P}_1$ does not change at all, it works in the same way as P. Now we have a new proof oracle evaluation algorithm $\tilde{P}_2$, it needs to compute 6 scalar products of length $4 + \mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P})$, so its complexity is $O(\mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P}))$. The verifier does just one more quadratic check that depends on the number of queries, which is constant. In total we get:

- $\tilde{P}_1(\boldsymbol{x}, \boldsymbol{w})$: $O(\mathsf{sdim}(\mathsf{P}) \log(\mathsf{sdim}(\mathsf{P})) + \mathsf{supp}(\mathsf{P}))$;
- $\tilde{P}_2(\boldsymbol{q})$: $O(\mathsf{sdim}(\mathsf{P}) + \mathsf{size}(\mathsf{P}))$;
- $\tilde{Q}()$: $O(\mathsf{sdim}(\mathsf{P}) + \mathsf{supp}(\mathsf{P}))$;
- $\tilde{D}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a})$: $O(|\boldsymbol{x}|)$.

Degrees: The additional query $\boldsymbol{q_k}$ computed by $\tilde{Q}$ is just a linear combination of the already produced queries, so $d_{\tilde{Q}} = d_Q$. The algorithm $\tilde{D}$ does a single additional quadratic test, so $d_{\tilde{D}} = \max(2, d_D)$.

5 From HVZK Linear Interactive Proof to Zero-Knowledge SNARK

This transition is again taken from [BCI+13]. In this subsection m will denote the total length of $\boldsymbol{q}$, and m' the total length of $\boldsymbol{u}$. The value k is now not necessarily the number of queries themselves (since they come as a single vector), but it is still the length of the prover's answer to queries. Here it is assumed more generally that the prover applies a matrix $\Pi \in \mathbb{F}^{k \times m}$ to the query $\boldsymbol{q}$, obtaining all the answers at once. This place might be simplified and adjusted to the fact that the QAP-related prover applies just the same vector $\boldsymbol{\pi}$ everywhere, so he does not have to encode the entire $\Pi \in \mathbb{F}^{k \times m}$.

In the Linear Interactive Proof we assumed that the prover is restricted to linear operations. Since we cannot in general assume that the malicious prover will perform only linear operations, we need to enforce this requirement. The *Succinct non-interactive argument of knowledge* (SNARK) enhances the Linear Interactive Proof with some encoding scheme that allows to restrict the prover to linear operations.

Definition 8. *A triple of algorithms (G, P, V) is a SNARK for the relation R if the following conditions are satisfied:*

1. ***Completeness:*** *for large enough $\lambda \in \mathbb{N}$, for any $(y, w) \in R$:*

$$\Pr\left[V(\tau, y, \pi) = 1 \middle| \begin{matrix}(\sigma, \tau) \leftarrow G(1^\lambda) \\ \pi \leftarrow P(\sigma, y, w)\end{matrix}\right] = 1 \ .$$

2. ***Adaptive proof of knowledge:*** *for any polynomial-size prover P^* there exists a polynomial-time extractor E such that for every large enough $\lambda \in \mathbb{N}$, every auxiliary input $z \in \{0,1\}^{poly(\lambda)}$:*

$$\Pr\left[\begin{matrix}V(\tau, y, \pi) = 1 \\ (y, w) \notin R\end{matrix} \middle| \begin{matrix}(\sigma, \tau) \leftarrow G(1^\lambda) \\ (y, \pi) \leftarrow P^*(z, \sigma) \\ w \leftarrow E(z, \sigma)\end{matrix}\right] \leq negl(\lambda) \ .$$

One way to achieve SNARK is to use some encoding scheme such that (τ, σ) are the private and the secret keys, and all the computation is performed in encrypted domain. Additionally, we want the computation to be publicly verifiable, so the verifier should be able to do his job without having access to the secret key. In order to make it possible, the scheme has to satisfy the following properties:

- It allows public testing of quadratic predicates on encoded elements.
- It provides a certain notion of one-way security to encoded elements.

– It ensures that any polynomial-size prover can only perform linear operations on the encoded elements, up to information leaked by the encoding.

There are some definitions that the scheme should satisfy formally.

Definition 9 (Linear-only encoding scheme). *A linear-only encoding scheme is a tuple of algorithms* $(Gen, Enc, SEnc, Test, Add, ImVer)$ *with the following syntax and correctness properties:*

- *Given a security parameter* 1^λ*, Gen generates a public key pk. The public key pk also includes a description of a field* $\mathbb{F}_\lambda$ *representing the plaintext space.*
- *Encoding can be performed in two modes:* Enc_{pk} *works in linear-only mode, and* $SEnc_{pk}$ *is a deterministic encoding algorithm that works in standard mode.*
- *As in linear-only encryption,* $Add(pk, Enc_{pk}(a_1), \ldots, Enc_{pk}(a_m), \alpha_1, \ldots, \alpha_m)$ *is a homomorphic evaluation algorithm for linear combinations. Namely, it computes an evaluated encoding* $c \in Enc_{pk}(\sum_i \alpha_i a_i)$*.*
- $ImVer_{pk}(c)$ *tests whether a given candidate encoding* c *is in the image of* Enc_{pk}*.*
- $Test(pk, \boldsymbol{t}, Enc_{pk}(a_1), \ldots, Enc_{pk}(a_m), SEnc_{pk}(b_1), \ldots, SEnc_{pk}(b_n)$ *is a public test for zeros of a multivariate polynomial* $\boldsymbol{t}$*. The test succeeds iff*

$$\boldsymbol{t}(a_1, \ldots, a_m, b_1, \ldots, b_n) = 0 \ .$$

The reason why non-linear-mode encryption $SEnc_{pk}$ is introduced is that the homomorphic property is not always needed, and some simpler encryption is sufficient.

Since the Π should remain unrevealed, it may happen that the Prover cheats and does not use a linear Π as he is supposed to. In this way, the encoding scheme should be chosen in such a way that the prover may apply only linear operations to the encrypted values, otherwise he should obtain some rubbish.

Definition 10 (Linear-onlyness). *A linear encoding scheme has the linear-only property if for any polynomial-size adversary A there is a polynomial-size extractor E such that for sufficiently large security parameter* $\lambda \in \mathbb{N}$*, an auxiliary input* $z \in \{0,1\}^{poly(\lambda)}$*, and any plaintext vectors* $\boldsymbol{q}$*,* $\tilde{\boldsymbol{q}}$*:*

$$\Pr\left[\begin{array}{l|c} \exists i^* : ImVer(\hat{c}_{i*}) = 1, & pk \leftarrow \hat{G}(1^\lambda) \\ \text{but } \hat{c}_{i^*} \neq Enc_{pk}(\hat{a}_{i^*}) & \boldsymbol{c} \leftarrow (Enc_{pk}(q_1), \ldots, Enc_{pk}(q_m)) \\ & \tilde{\boldsymbol{c}} \leftarrow (SEnc_{pk}(\tilde{q}_1), \ldots, SEnc_{pk}(\tilde{q}_m)) \\ & (\hat{c}_1, \ldots, \hat{c}_k) \leftarrow A(pk, z, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \\ & (\Pi, \boldsymbol{b}) \leftarrow E(pk, z, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \\ & (\hat{a}_1, \ldots, \hat{a}_k)^{\mathrm{T}} \leftarrow \Pi \cdot (q_1, \ldots, q_m)^{\mathrm{T}} + \boldsymbol{b} \end{array}\right] \leq negl(\lambda) \ ,$$

where $\Pi \in \mathbb{F}^{k \times m}$ *and* $\boldsymbol{b} \in \mathbb{F}^k$*.*

One more thing that we need to ensure is that the encoding scheme does not provide any means of getting $Test$ passed without $\boldsymbol{t}$ actually being a zero polynomial. By Schwartz-Zippel lemma, for a random evaluation this probability is small with respect to degree of $\boldsymbol{t}$ and the field size, and now we need to ensure that it is small with respect to the security parameter λ.

Definition 11 (Multivariate Δ-power one-wayness). *A linear encoding scheme satisfies multivariate Δ-power one-wayness if, for every polynomial-size adversary A, large enough security parameter $\lambda \in \mathbb{N}$ and μ-variate polynomials $(p_1, \ldots, p_\ell)$ of total degree at most Δ:*

$$\Pr\left[p^* \neq 0, p^*(\boldsymbol{s}) = 0 \;\middle|\; \begin{array}{c} pk \leftarrow \hat{G}(1^\lambda) \\ \boldsymbol{s} \leftarrow \mathbb{F}^\mu \\ \boldsymbol{c} \leftarrow (Enc_{pk}(p_1(\boldsymbol{s})), \ldots, Enc_{pk}(p_\ell(\boldsymbol{s}))) \\ \tilde{\boldsymbol{c}} \leftarrow (SEnc_{pk}(p_1(\boldsymbol{s})), \ldots, SEnc_{pk}(p_\ell(\boldsymbol{s}))) \\ p^* \leftarrow A(pk, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \end{array} \right] \leq negl(\lambda) \ ,$$

where p^* is μ-variate and Δ, ℓ, μ are poly(λ).

5.1 Construction

Let $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ be a HVZK Linear Interactive Proof System. Let λ be a security parameter. Let all the computations be done over a field $\mathbb{F}_\lambda$ (the size of the field depends on the security parameter). Formally, a $SNARK(\hat{G}, \hat{P}, \hat{V})$ consists of a key generator $\hat{G}$, a prover $\hat{P}$, and a verifier $\hat{V}$. They are defined as follows.

- $\hat{G}(\lambda)$ invokes $\tilde{Q}()$ and generates $\boldsymbol{q} \in \mathbb{F}_\lambda^m$, $\boldsymbol{u} \in \mathbb{F}_\lambda^{m'}$. It selects an appropriate *linear-only one-way* encoding scheme and generates the public and the secret key pair of that scheme (pk, sk). Let $c_i \leftarrow Enc_{pk}(\boldsymbol{q_i})$, $\tilde{c_i} \leftarrow SEnc_{pk}(\boldsymbol{q_i})$. The algorithm outputs $(\sigma, \tau) = ((pk, c_1, \ldots, c_m), (pk, \tilde{c_1}, \ldots, \tilde{c_{m'}}))$, which are called the *reference string* and the *verification state*.
- The prover $\hat{P}(\boldsymbol{x}, \boldsymbol{w}, \sigma)$ refers to the algorithm $\tilde{P}_1(\boldsymbol{x}, \boldsymbol{w})$ to generate $\boldsymbol{\pi}$. As we discussed above, in general the Linear Interactive Prover may generate a $(k \times m)$ matrix Π which in our particular construction will be

$$\Pi = \begin{pmatrix} \boldsymbol{\pi}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & \ldots & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0}^{\mathrm{T}} & \boldsymbol{\pi}^{\mathrm{T}} & \ldots & \mathbf{0}^{\mathrm{T}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & \ldots & \boldsymbol{\pi}^{\mathrm{T}} \end{pmatrix} \ .$$

  Let Π_i denote the i-th row vector of Π. In general, the prover computes $\bar{a}_i = Enc_{pk}(\Pi_i \cdot \boldsymbol{q})$ by using homomorphic properties of the encoding scheme. This can be done by computing

$$\bar{a}_i = Add(pk, Enc_{pk}(q_1), \ldots, Enc_{pk}(q_m), \Pi_{i1}, \ldots, \Pi_{im}) = \sum_{j=1}^{m} \Pi_{ij} q_j = Enc_{pk}(\langle \Pi_i, \boldsymbol{q} \rangle) \ .$$

In particular, in our PCP construction $\boldsymbol{q} = (\hat{\boldsymbol{q}}_1 || \ldots || \hat{\boldsymbol{q}}_k)$ consists of subqueries of some length m/k, each of whom should be multiplied by $\boldsymbol{\pi}$. Hence an honest prover computes

$$\bar{a}_i = Add(pk, Enc_{pk}(\hat{\boldsymbol{q}}_{i1}), \ldots, Enc_{pk}(\hat{\boldsymbol{q}}_{im/k}), \pi_1, \ldots, \pi_{m/k}) = \sum_{j=1}^{m/k} \pi_j \hat{\boldsymbol{q}}_{ij} = Enc_{pk}(\langle \boldsymbol{\pi}, \hat{\boldsymbol{q}}_i \rangle) \ .$$

– The verifier $\hat{V}(\boldsymbol{x}, \bar{\boldsymbol{a}}, \tau)$ cannot decrypt the values $\bar{a}_i$, but due to the properties of the encoding scheme he is able to do the verification. First of all, it checks if $ImVer_{pk}(\bar{a}_i) = 1$ (i.e if the values sent by the prover are indeed valid). Let $\boldsymbol{t}$ be the quadratic polynomial whose property of being a zero polynomial is checked by $\tilde{D}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{a})$ (note that $\tilde{D}$ cannot be called directly since both $\boldsymbol{u}$ and $\boldsymbol{a}$ are encoded). The verifier accepts iff $Test(pk, \boldsymbol{t}, \bar{a}_1, \ldots, \bar{a}_k, \tilde{c}_1, \ldots, \tilde{c}_{m'}) = 1$, what by definition of $Test$ means that $\boldsymbol{t}(\bar{a}_1, \ldots, \bar{a}_k, \tilde{c}_1, \ldots, \tilde{c}_{m'}) = 0$.

5.2 Properties

If the Linear Interactive Proof $((\tilde{P}_1, \tilde{P}_2), (\tilde{Q}, \tilde{D}))$ has knowledge error $\varepsilon(\lambda)$, then $(\hat{G}, \hat{P}, \hat{V})$ is a publicly verifiable preprocessing SNARK with knowledge error $\varepsilon(\lambda) + negl(\lambda)$ with the following overheads:

– $time(\hat{G}) = time(\tilde{Q}) + poly(\lambda) \cdot (m + m')$;
– $time(\hat{P}) = time(\tilde{P}_1) + poly(\lambda) \cdot k \cdot m$;
– $time(\hat{V}) = poly(\lambda) \cdot time(\tilde{D})$;
– $|\sigma| = poly(\lambda) \cdot m$;
– $|\tau| = poly(\lambda) \cdot m'$;
– $|\bar{\boldsymbol{a}}| = poly(\lambda) \cdot k$.

We will prove the properties one by one.

Completeness Follows from the completeness of the corresponding Linear Interactive Proof and the correctness of the encryption scheme.

Knowledge By definition of knowledge, we need to show that

$$\Pr\left[\begin{array}{c} \hat{V}(\boldsymbol{y}, \hat{\boldsymbol{a}}, \tau) = 1 \\ (\boldsymbol{y}, \boldsymbol{w}) \notin R_{\mathsf{P}(A,B,C)} \end{array} \middle| \begin{array}{c} (\sigma, \tau) \leftarrow \hat{G}(1^\lambda) \\ (\boldsymbol{y}, \hat{\boldsymbol{a}}) \leftarrow \hat{P}^*(z, \sigma) \\ \boldsymbol{w} \leftarrow E(z, \sigma) \end{array} \right] \leq negl(\lambda) \ ,$$

Since we are dealing with publicly verifiable computation, we assume τ is public and known also to the prover and the extractor. Since the encoding scheme is linear-only, by definition there exists an extractor E_1 such that

$$\Pr\left[\begin{array}{c} \exists i^* : ImVer(\hat{c}_{i*}) = 1, \\ \text{but } \hat{c}_{i^*} \neq Enc_{pk}(\hat{a}_{i^*}) \end{array} \middle| \begin{array}{c} (\sigma, \tau) \leftarrow \hat{G}(1^\lambda) \\ pk \leftarrow \tau \\ (\hat{c}_1, \ldots, \hat{c}_k) \leftarrow \hat{P}^*(pk, z, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \\ (\Pi, \boldsymbol{b}) \leftarrow E_1(pk, z, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \\ (\hat{a}_1, \ldots, \hat{a}_k)^{\mathrm{T}} = \Pi \cdot \boldsymbol{q} + \boldsymbol{b} \end{array} \right] \leq negl(\lambda) \ ,$$

so running $E_1(pk, z, \boldsymbol{c}, \tilde{\boldsymbol{c}})$ gives an affine transformation $\Pi^* = (\Pi, \boldsymbol{b})$. Additionally, since we have already shown that our Linear Interactive Proof satisfies knowledge property on the assumption that Π^* is an affine transformation, there exists an extractor E_2 such that $E_2^{\Pi^*}(\boldsymbol{x}) = \boldsymbol{w}$ where $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$. We may define E that first calls E_1 to get Π^*, and then calls E_2 to get $\boldsymbol{w}$.

Now let us conclude what happens if the verifier $\hat{V}$ accepts if we define E in this way. Suppose $\hat{V}(\boldsymbol{y}, \hat{\boldsymbol{a}}, \tau) = 1$. This means that the checks $ImVer_{pk}(\hat{c}_i)$ for all i and $Test(pk, \boldsymbol{t}, \hat{\boldsymbol{c}}, \tilde{\boldsymbol{c}})$ both succeed. By definition of $Test$ and $ImVer_{pk}$, it happens iff $\boldsymbol{t}(\Pi^*\boldsymbol{q}||\boldsymbol{u}) = 0$ and $\forall i: \ \hat{c}_i = Enc_{pk}(\hat{a}_i)$. Due to linear-onlyness, for a non-linear transformation Π^* the verification $ImVer_{pk}$ succeeds with probability $negl(\lambda)$, so Π^* is indeed affine. For an affine Π^*, if $\boldsymbol{t}(\Pi^*\boldsymbol{q}(r)||\boldsymbol{u}(r))$ is a zero polynomial over r (where r is the randomness used by the verifier), then Π^* is a valid affine function, and hence $\boldsymbol{w}$ returned by E_2 is a valid witness such that $(\boldsymbol{x}, \boldsymbol{w}) \in R_{\mathsf{P}(A,B,C)}$. It remains to show that if $\boldsymbol{t}(\Pi^*\boldsymbol{q}(r)||\boldsymbol{u}(r))$ is not a zero polynomial, then $\boldsymbol{t}(\Pi^*\boldsymbol{q}||\boldsymbol{u}) = 0$ succeeds with negligible probability. Let $Var(r)$ denote the set of polynomial variables. Recall that the encoding scheme satisfies the multivariate Δ-power one-wayness property:

$$\Pr\left[p^* \neq 0, p*(\boldsymbol{s}) = 0 \middle| \begin{array}{c} pk \leftarrow \hat{G}(1^\lambda) \\ \boldsymbol{s} \leftarrow \mathbb{F}^{|Var(r)|} \\ p^* \leftarrow P^*(pk, \boldsymbol{c}, \tilde{\boldsymbol{c}}) \end{array}\right] \leq negl(\lambda) \ .$$

The polynomial $\boldsymbol{t}(\Pi^*\boldsymbol{q}(r)||\boldsymbol{u}(r)) = 0$ is definitely of degree in $poly(\lambda)$, and the number of its variables is also in $poly(\lambda)$, so the definition is applicable. Hence no malicious Π^* may help the prover to adjust the checked polynomial to the randomness r.

5.3 Complexity Overheads

- $time(\hat{G}) = time(\tilde{Q}) + poly(\lambda) \cdot (m + m')$.
  Each of the $m + m'$ elements of $\boldsymbol{q}$ and $\boldsymbol{u}$ requires one encoding of $poly(\lambda)$.
- $time(\hat{P}) = time(\tilde{P}_1) + poly(\lambda) \cdot k^2 \cdot m$.
  Computing each $\bar{a}_i$ requires running Add that performs $O(m)$ encrypted multiplications, so complexity for one particular i is $poly(\lambda) \cdot m$, and the entire $\bar{\boldsymbol{a}}$ is computed in $poly(\lambda) \cdot k \cdot m$.
- $time(\hat{V}) = poly(\lambda) \cdot time(\tilde{D})$.
  The final decision of the verifier performs exactly the same operations as D does, but each of them is performed homomorphically. Hence the time is $poly(\lambda) \cdot time(\tilde{D})$.
- $|\sigma| = poly(\lambda) \cdot m$.
  The quantity σ contains m encrypted elements (and the public key).
- $|\tau| = poly(\lambda) \cdot m'$.
  The quantity τ contains m' encrypted elements (and the public key).
- $|\bar{\boldsymbol{a}}| = poly(\lambda) \cdot k$.
  The quantity $\bar{\boldsymbol{a}}$ contains k encrypted elements.

Complexity overheads w.r.t QAP Let us now consider the complexities of the prover and the verifier in terms of the initial QAP. Recall that the parameters were the following:

- $m = |\boldsymbol{q}| = 6 \cdot (4 + \mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P})) = O(\mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P}))$;
- $m' = |\boldsymbol{u}| = O(|x|)$;
- $k = 5$.

Now we can do the final estimation:

$$\begin{aligned}
time(\hat{G}) &= time(\tilde{Q}) + poly(\lambda) \cdot (m + m') \\
&= O(\mathsf{sdim}(\mathsf{P}) + \mathsf{supp}(\mathsf{P})) + poly(\lambda) \cdot (O(\mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P})) + O(|x|)) \\
&= O(poly(\lambda)(\mathsf{sdim}(\mathsf{P}) + \mathsf{supp}(\mathsf{P}) + \mathsf{size}(\mathsf{P}) + |x|)) \ ; \\
time(\hat{P}) &= time(\tilde{P}_1) + poly(\lambda) \cdot k \cdot m \\
&= O(\mathsf{sdim}(\mathsf{P}) \log(\mathsf{sdim}(\mathsf{P})) + \mathsf{supp}(\mathsf{P})) + poly(\lambda) \cdot 5 \cdot O(\mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P})) \\
&= O(\mathsf{sdim}(\mathsf{P}) \log(\mathsf{sdim}(\mathsf{P})) + \mathsf{supp}(\mathsf{P}) + poly(\lambda) \cdot O(\mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P}))) \ ; \\
time(\hat{V}) &= poly(\lambda) \cdot time(\tilde{D}) \\
&= poly(\lambda) O(|x|) \ ;
\end{aligned}$$

- $|\sigma| = poly(\lambda) \cdot O(\mathsf{size}(\mathsf{P}) + \mathsf{sdim}(\mathsf{P}))$;
- $|\tau| = poly(\lambda) \cdot O(|x|)$;
- $|\bar{\boldsymbol{a}}| = poly(\lambda)$.

Regarding the initial circuit from which the QAP was constructed, on the assumption that it had an introduced artificial multiplication gate after each addition gate, we get $\mathsf{supp}(\mathsf{P}) = O(\mathsf{sdim}(\mathsf{P})) = |C|$, and also $\mathsf{size}(\mathsf{P}) = |C|$. Hence the values will be the following:

- $time(\hat{G}) = O(poly(\lambda)(|C| + |x|))$;
- $time(\hat{P}) = O(|C| \log |C| + poly(\lambda) \cdot O(|C|))$;
- $time(\hat{V}) = poly(\lambda) O(|x|)$;
- $|\sigma| = poly(\lambda) \cdot O(|C|)$;
- $|\tau| = poly(\lambda) \cdot O(|x|)$;
- $|\bar{\boldsymbol{a}}| = poly(\lambda)$.

5.4 Selection of the Encoding Scheme

In [BCI+13], the proposed encoding scheme is based on the knowledge of exponent assumption in bilinear groups. Let $\{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$ be a bilinear group ensemble where each $(\mathbb{G}, \mathbb{G}_T) \in \mathcal{G}_\lambda$ is a pair of groups of prime order of $p \in (2^{\lambda-1}, 2^\lambda)$ with an efficiently-computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

- The public key is $pk = (g, g^\alpha)$ where $\langle g \rangle = \mathbb{G}$ and $\alpha \leftarrow \mathbb{F}_p$ is random.
- The linear-only encoding is defined as $Enc_{pk}(a) = (g^a, g^{\alpha a})$.
- The standard mode encoding is $SEnc_{pk}(a) = g^a$.

– $Add(pk, Enc_{pk}(a_1), \ldots, Enc_{pk}(a_m), \gamma_1, \ldots, \gamma_m)$ computes $Enc_{pk}(\sum_{i=1}^{m} \gamma_i a_i)$. Since $Enc_{pk}(a_i) = (g^{a_i}, g^{\alpha a_i})$, it may compute

$$((g^{a_1})^{\gamma_1} \cdot \ldots \cdot (g^{a_m})^{\gamma_m}, (g^{\alpha a_1})^{\gamma_1} \cdot \ldots \cdot (g^{\alpha a_m})^{\gamma_m}) = (g^{\sum_{i=1}^{m} \gamma_i a_i}, g^{\alpha \sum_{i=1}^{m} \gamma_i a_i})$$
$$= Enc_{pk}(\sum_{i=1}^{m} \gamma_i a_i) \ .$$

– $ImVer_{pk}(f, f')$ outputs 1 iff $e(f, g^\alpha) = e(g, f')$.
  - If indeed $f' = f^\alpha$, then $e(f, g^\alpha) = e(g, f^\alpha) = e(g, f)^\alpha$.
  - If $f' \neq f^\alpha$, then it would be difficult to obtain $e(f, g^\alpha) = e(g, f')$ due to the knowledge of exponent assumption.
– The polynomial test for $\boldsymbol{t}$ on inputs $(a_1, \ldots, a_k)$ and $(\tilde{a}_1, \ldots, \tilde{a}_k)$ can be done by pairings. For each $i \in k$, send $(g^{a_i}, g^{\alpha a_i}) = Enc_{pk}(a_i)$ and $g^{\tilde{a}_i} = SEnc_{pk}(\tilde{a}_i)$. The $\boldsymbol{t}$ is a quadratic polynomial over the variables a_i and $\tilde{a}_i$. In order to check that it is 0, it suffices to model computations that consist of additions and multiplications of at most 2-field-elements.
  - **Multiplication of two encoded values:** Let $a \cdot b$ be some multiplication that is present in the polynomial. The values a and b may be encoded by both Enc_{pk} and $SEnc_{pk}$. In any case, the encoding of a contains g^a, and encoding of b contains g^b, so we may compute $e(g^a, g^b) = e(g, g)^{ab}$. The values $g^{\alpha a}$ and $g^{\alpha b}$ are discarded.
  - **Multiplication of an encoded value by a non-encoded value:** Let a be encoded and b not. Take g^a and compute $g^b, e(g^a, g^b) = e(g, g)^{ab}$.
  - **Multiplication of non-encoded values:** Take a, b, compute g^a, g^b, $e(g^a, g^b) = e(g, g)^{ab}$.
  - **Addition:** assume a and b are given as $e(g, g)^a$ and $e(g, g)^b$. Compute $e(g, g)^a \cdot e(g, g)^b = e(g, g)^{a+b}$.

Enhancement of the Encoding Scheme Since the transition to SNARK depends on the complexity of the encoding scheme, its choice is also important. For example, [BSCG+13] proposes how to choose it in such a way that it would be more appropriate for their particular implementation. It is important to keep in mind the requirements that the underlying group has to achieve in order that the Linear PCP operations would be still efficient. If the order of the group is r, then $r - 1$ should be divisible by 2 since we want the group to have roots of unity of a "large enough" power of 2.

The encoding scheme that is used in [BSCG+13] is a bit different from the one described in the previous subsection. The pairing is defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for distinct groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

– The public key is $pk = (g, h)$ where $\langle g \rangle = \mathbb{G}_1$ and $\langle h \rangle = \mathbb{G}_2$.
– Both the linear-only and the standard mode encodings are defined as $Enc_{pk}(a) = (g^a, h^a)$.

– $Add(pk, Enc_{pk}(a_1), \ldots, Enc_{pk}(a_m), \gamma_1, \ldots, \gamma_m)$ computes

$$((g^{a_1})^{\gamma_1} \cdot \ldots \cdot (g^{a_m})^{\gamma_m}, (h^{a_1})^{\gamma_1} \cdot \ldots \cdot (h^{a_m})^{\gamma_m}) = (g^{\sum_{i=1}^{m} \gamma_i a_i}, h^{\sum_{i=1}^{m} \gamma_i a_i}) = Enc_{pk}(\sum_{i=1}^{m} \gamma_i a_i) \ .$$

– $ImVer_{pk}(f, f')$ outputs 1 iff $e(f, h) = e(g, f')$.
– The polynomial test for $\boldsymbol{t}$ on inputs $(a_1, \ldots, a_k)$ and $(\tilde{a}_1, \ldots, \tilde{a}_k)$ can be also done by pairings. For each $i \in k$, send $(g^{a_i}, h^{a_i}) = Enc_{pk}(a_i)$ and $(g^{\tilde{a}_i}, h^{\tilde{a}_i}) = Enc_{pk}(\tilde{a}_i)$.
  - **Multiplication of two encoded values:** Let $a \cdot b$ be some multiplication that is present in the polynomial. We may take g^a from the encoding of a, h^b from the encoding of b, and compute $e(g^a, h^b) = e(g, h)^{ab}$. The values h^a and g^b are discarded.
  - **Multiplication of an encoded value by a non-encoded value:** Let a be encoded and b not. Take g^a and compute $h^b, e(g^a, h^b) = e(g, h)^{ab}$.
  - **Multiplication of non-encoded values:** Take a, b, compute $g^a, h^b, e(g^a, h^b) = e(g, h)^{ab}$.
  - **Addition:** assume a and b are given as $e(g, h)^a$ and $e(g, h)^b$. Compute $e(g, h)^a \cdot e(g, h)^b = e(g, h)^{a+b}$.

The efficiency improvements related to encoding scheme are the following:

– **Reducing the number of group operations in** P. The SNARK prover P faces several large instances of a multi-exponentiation problem: given group elements $g_1, \ldots, g_m \in \mathbb{G}$ and some integers $a_1, \ldots, a_m$ compute $\Pi_{i=1}^{m} g^{a_i}$. This is being done by a multi-exponentiation algorithm.
– **Reducing the number of group operations in** G. The SNARK generator G is instead faced with several large instances of the following exponentiation problem: given a group element $g \in \mathbb{G}$ and integers $a_1, \ldots, a_m$, compute the tuple $(g^{a_1}, \ldots, g^{a_m})$. The number of required group operations is reduced by using the standard technique of pre-computing a table of powers of g.
– **Reducing the cost of group operations**. The instantiation of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ should be done in a clever way. The multiplication and the pairing should be efficient, and in at the same time the group has to remain efficient for FFT. The authors of [BSCG+13] have found an appropriate group that satisfies all these requirements. It was found experimentally in 2^{38} trials.
– **Reducing the number of operations in** $\mathbb{G}_2$. It is a known fact that operations in $\mathbb{G}_2$ are more expensive than operations in $\mathbb{G}_1$.

6 Zero-knowledge Program Verification in Secure Multiparty Computation

This section presents one particular solution for using zero-knowledge program verification in secure multiparty computation. Consider the case where several

semihonest parties securely compute some functionality. If the protocol is secure only against passive adversaries, then some party might have cheated in the computation by deviating the computation rules. Since security against malicious adversary may be too expensive, another possible solution is to make cheating detectable after the computation. Any party should be able to prove to the court that it performed its computation correctly, but without providing any extra information about its own shares and the randomness it used.

The general idea is that any protocol can be represented as a circuit, which contains some special wires that represent communication. A straightforward solution for verifying if the party P_i is honest would be the following.

- During the computation, all the parties publish the encryptions of the values that have been sent through the communication wires. The receiver party does not proceed unless the value it received indeed corresponds to the published encryption, and hence the communication variables are committed.
- When P_i wants to prove that it acted honestly, a QAP is generated from a subcircuit that contains all the computation of P_i, and where the communicated values are considered as inputs/outputs.
- The party P_i has to prove that it knows an evaluation of the remaining input variables and the intermediate gates such that the computation indeed corresponds to the committed communicated values.

One problem of this straightforward approach is the number of encryptions. Although their number is linear in the size of the circuit, it is still too inefficient in practice. The solution that is proposed here is designed for the 3-party case, from which at most one is malicious. This additional assumption allows to reduce the number of encryptions and make them linear in the communication size. The solution is proposed without formal security proofs, and it can be considered a draft that can be improved afterwards.

In our settings, we have the parties M_1, M_2, M_3, and the Court. The proposed solution has the following computational overheads.

- During the computation, each value sent from M_i to M_j should be committed as a homomorphically encrypted value. A more efficient solution would be to use some kind of non-homomorphic commitment that is easier to compute, and that can be converted to a homomorphically encrypted value in the end, after all the computation has finished.
- In order to verify M_1's honestness, M_2 and M_3 have to pefrorm an exponentiation of each of the committed homomorphic communication values (since they need to be multiplied by $\boldsymbol{q}$ somehow).
- Each party has to compute k more homomorphic encryptions for a security parameter k. The initial idea was that these k encryptions can be avoided, but it is sufficient only to make it clear that "someone is dishonest", and in the case of a problem the Court should proceed with the k-encryption version anyway (although if all the users behave honestly at this point, then this additional check may be skipped).
- The Court has to do in total a constant number of operations.

In total, we get that the number of required homomorphic encryptions is $O(c+k)$, where c is the number of communications and k is the security parameter that should be around 80 in standard settings. The number of homomorphic exponentiations is $O(c)$.

The circuit in general has the following parameters:

- Input vectors: $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, $\boldsymbol{x_3}$ where $\boldsymbol{x_i}$ is owned by M_i.
- Output vectors: $\boldsymbol{y_1}$, $\boldsymbol{y_2}$, $\boldsymbol{y_3}$ where $\boldsymbol{y_i}$ is given to M_i.
- Intermediate gates: $\boldsymbol{z_1}$, $\boldsymbol{z_2}$, $\boldsymbol{z_3}$ where $\boldsymbol{z_i}$ is computed by M_i.
- Communication values: $\boldsymbol{c_{12}}$, $\boldsymbol{c_{13}}$, $\boldsymbol{c_{21}}$, $\boldsymbol{c_{23}}$, $\boldsymbol{c_{31}}$, $\boldsymbol{c_{32}}$, where $\boldsymbol{c_{ij}}$ is a vector of values that have been sent from M_i to M_j.

The communication values are special since they are committed. Each time when M_i sends some value c to M_j, the following happens:

- M_i computes $Enc_{pk,r}(c)$ in a homomorphic encoding scheme and publishes $Enc_{pk,r}(c)$.
- M_i sends $sign_{pk(i)}(c, r)$ to P_j.
- M_j verifies if the published $Enc_{pk,r}(c)$ indeed corresponds to (c, r), otherwise it does not proceed.

Consider the case where we want to prove that M_1 is honest. The idea is that, given the committed values that were sent to M_1 and from M_1, it has to prove that it knows appropriate $\boldsymbol{x_1}$ and $\boldsymbol{z_1}$ such that the communication commitments are consistent. In this way, the verification has to be modeled up to the last communication which in fact covers almost the entire protocol.

For each of the 6 queries (according to the Linear IP presented in QAP section), we need to compute one scalar product. In general, this can be done in any case using homomorphic encryption, but that would be very slow. Hence the following is being done:

- M_2 and M_3 agree on $\boldsymbol{q}$. Namely, M_2 generates τ and sends it to M_3. Since everything else depend on the circuit, each of them has sufficient information to generate $\boldsymbol{q}$.
- Let $\boldsymbol{q} = (\boldsymbol{q_P}||\boldsymbol{q_S})$ where $\boldsymbol{q_P}$ corresponds to the variables related to the private inner computations of P_1, and $\boldsymbol{q_S}$ corresponds to the communication variables. Let $\boldsymbol{\pi} = (\boldsymbol{\pi_P}||\boldsymbol{\pi_S})$ be the corresponding values that M_1 should know. Since $\boldsymbol{q_S}$ corresponds to the communication variables, the vector $\boldsymbol{\pi_S}$ is actually already committed. Both M_2 and M_3 agree on $\boldsymbol{r}$ and compute and publish $Enc_{pk}(\boldsymbol{\pi_S})^{\boldsymbol{q_S(r)}}$ (here $\boldsymbol{r}$ denotes the randomness used in the exponentiation, and $Enc_{pk}(\boldsymbol{\pi_S})^{\boldsymbol{q_S(r)}}$ is just a short notation for exponentiating i-th element of $\boldsymbol{\pi_S}$ with i-th element of $\boldsymbol{q_S}$). In this way, one of these published values (let they be denoted v_2 and v_3) may be malicious, but not both. However, since honest M_2 and M_3 use the same randomness, we have $v_2 = v_3$, and inconsistency can be easily detected by comparing these two values:
  - If the values are different, then we know that either M_2 or M_3 is malicious, but M_1 is definitely honest since there is at most one malicious party. The protocol returns "M_1 is honest".

- Otherwise we proceed with the assumption that $Enc_{pk}(\langle \boldsymbol{\pi_S}, \boldsymbol{q_S} \rangle)$ is published correctly.

– M_1 owns $\boldsymbol{\pi} = (\boldsymbol{\pi_P} || \boldsymbol{\pi_S})$, and its goal is to compute the scalar multiplication $\langle (\boldsymbol{\pi_P} || \boldsymbol{\pi_S}), (\boldsymbol{q_P} || \boldsymbol{q_S}) \rangle$. The part $\langle \boldsymbol{\pi_S}, \boldsymbol{q_S} \rangle$ has already been computed by M_2 and M_3. Since M_1 is not allowed to generate $\boldsymbol{\pi_P}$ after seeing $\boldsymbol{q_P}$, it must use the help of M_2 and M_3.
  - M_1 generates random vectors $\boldsymbol{r_i}$, $\boldsymbol{a_i}$, $\boldsymbol{b_i}$ for $i \in \{1, \ldots, k\}$ where k is a security parameter. It sends all $\boldsymbol{\pi_P} - \boldsymbol{r_i}$, $\boldsymbol{a_i}$ to M_2 and all $\boldsymbol{r_i}$, $\boldsymbol{b_i}$ to M_3, where all the values are signed with some simple system, without using homomorphic encryption. The idea is that M_2 and M_3 do not know which values are real and which are random.
  - M_1 and M_2 proceed only if the values indeed correspond to the signatures. They compute and publish $Enc_{pk}(\langle \boldsymbol{\pi_P} - \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$, $Enc_{pk}(\langle \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$, $Enc_{pk}(\langle \boldsymbol{a_i}, \boldsymbol{q_P} \rangle)$, $Enc_{pk}(\langle \boldsymbol{b_i}, \boldsymbol{q_P} \rangle)$ for all i.
  - M_1 publishes all $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$. The parties may now verify that they both have computed $Enc_{pk}(\langle \boldsymbol{a_i}, \boldsymbol{q_P} \rangle)$ and $Enc_{pk}(\langle \boldsymbol{b_i}, \boldsymbol{q_P} \rangle)$ correctly. Here M_1 cannot cheat by revealing false $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$ since the receiving parties may present the signatures, and it will be immediately proven that M_1 is dishonest. Otherwise if anything is wrong, then either M_2 claims that M_3 is dishonest, or the other way around, and the Court definitely knows that M_1 is honest since there is at most one malicious party.
  - If all the $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$ are correct, then the probability that at least one of the $Enc_{pk}(\langle \boldsymbol{\pi_P} - \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$, $Enc_{pk}(\langle \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$ is correct is $1 - 1/\binom{2k}{k} > 1 - \frac{1}{2^k}$. The main idea here that it is sufficient to take $k = 80$, and although each of them needs one homomorphic encryption, it is much less than the number of intermediate nodes in a program circuit.
  - After $Enc_{pk}(\langle \boldsymbol{\pi_P} - \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$ and $Enc_{pk}(\langle \boldsymbol{r_i}, \boldsymbol{q_P} \rangle)$ have been computed for all the 6 queries, the vectors $\boldsymbol{q_P}1, \ldots, \boldsymbol{q_P}6$ can be finally sent to M_1. For each i, it computes $Enc_{pk}(\langle \boldsymbol{\pi_P}, \boldsymbol{q_P}i \rangle)$ and finds the values computed by M_2 and M_3 that correspond to it (at least one must correspond with a high probability). Even more, M_1 actually may detect cheating if some of them does not correspond, but unfortunately he cannot prove it. However, this cheating does not harm M_1's proof in any way. The large choice does not allow M_1 to cheat since if M_1 is malicious, then M_2 and M_3 are honest, and hence all of the encryptions encode the same value. M_1 selects the "correct" values computed by M_2 and M_3 and publishes their indices, so that the Court may proceed with them. In this way, since M_1 can just use the values pre-computed by the other parties, seeing $\boldsymbol{q_P}i$ vectors does not help it in any way.

– Now the Court is ready to do its work. For each query $\boldsymbol{q} = (\boldsymbol{q_P} || \boldsymbol{q_S})$, it gets $Enc_{pk,\boldsymbol{r}}(\langle \boldsymbol{\pi_S}, \boldsymbol{q_S} \rangle)$ which is definitely correct, and $Enc_{pk}(\langle \boldsymbol{r}, \boldsymbol{q_P} \rangle)$, $Enc_{pk}(\langle \boldsymbol{\pi_P} - \boldsymbol{r}, \boldsymbol{q_P} \rangle)$ that have been indicated by M_1, and are correct with probability $1 - \frac{1}{2^k}$. If M_1 complains, then with a greater probability it just wants to accuse M_2 and M_3 without reason, and in this case the verification may be repeated again.

– The Court computes $Enc_{pk}(\langle \boldsymbol{\pi_S}, \boldsymbol{q_S} \rangle) \cdot Enc_{pk}(\langle \boldsymbol{\pi_P} - \boldsymbol{r}, \boldsymbol{q_P} \rangle) \cdot Enc_{pk}(\langle \boldsymbol{r}, \boldsymbol{q_P} \rangle)$ for all 6 queries, obtaining the encryptions $Enc_{pk}(a_1), \ldots, Enc_{pk}(a_6)$ where $a_i = \langle \boldsymbol{\pi_i}, \boldsymbol{q_i} \rangle$. Then it performs the short check as described in the previous section. The answer shows if M_1 was honest with the probability of cheating ε that comes from Linear Interactive Proof construction. Since the check contains one pairing operation for each of the 6 queries, we would probably like to improve it. Since M_1 knows the values of a_i (it can freely see all the $\boldsymbol{q_i}$ after its commitments are done), it may compute $Enc_{pk}(a_1 \cdot a_2)$ locally by itself, and afterwards use some proof that it is indeed the value $Enc_{pk}(a_1 \cdot a_2)$.

7 Conclusion

Although the recent developments in program verification have relatively good asymptotic complexities, they may still be too inefficient in practice due to large number of homomorphic encryptions. Unfortunately, although all the encryptions are performed in the preprocessing phase, this preprocessing works for just a single proof instance. If the verification has to be performed with some other input, we cannot in general use the same precomputed challenge anymore since we cannot use the assumption that it is completely random. Hence we either should either generate a new challenge and again perform the entire encryption, or lose some security properties.

For particular computational settings, some intermediate values can be computed more efficiently by using the additional assumptions. The number of encryptions can also be reduced by introducing a constant number of interactions instead of making the proof completely non-interactive.

References

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, pages 90–108, 2013.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. Cryptology ePrint Archive, Report 2013/121, 2013.