

RECAP
oooooooooooo

QUADRATIC ARITHMETIC PROGRAM
oooooooooooooooooooo

PROBABILISTICALLY CHECKABLE PROOFS
oooo

QAP AS A LINEAR PCP
oooo

PROOF OF EXPONENT
oooooo

QAP, PCP, POE

Oct 1, 2024

Distributed Lab

Plan

Recap

Quadratic Arithmetic Program

Probabilistically Checkable Proofs

QAP as a Linear PCP

Proof Of Exponent

Recap

Recap: what is zk-SNARK?

Definition

zk-SNARK

Zero-**K**nowledge **S**uccinct **N**on-interactive **A**Rgument of **K**nowledge.

Recap: what is zk-SNARK?

Definition

zk-SNARK

Zero-**K**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge.

- ✓ **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be “extracted”.

Recap: what is zk-SNARK?

Definition

zk-SNARK

Zero-**K**nowledge **S**uccinct **N**on-interactive **A**Rgument of **K**nowledge.

- ✓ **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be “extracted”.
- ✓ **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.

Recap: what is zk-SNARK?

Definition

zk-SNARK

Zero-**K**nowledge **S**uccinct **N**on-interactive **A**Rgument of **K**nowledge.

- ✓ **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be “extracted”.
- ✓ **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- ✓ **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.

Recap: what is zk-SNARK?

Definition

zk-SNARK

Zero-**K**nowledge **S**uccinct **N**on-interactive **A**RGument of **K**nowledge.

- ✓ **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be “extracted”.
- ✓ **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- ✓ **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.
- ✓ **Zero-Knowledge** — the verifier learns nothing about the data used to produce the proof, despite knowing that this data resolves the given problem and that the prover possesses it.

Recap: Arbitrary Program To Circuits

We can do that in a way like the computer does it — **boolean circuits**.

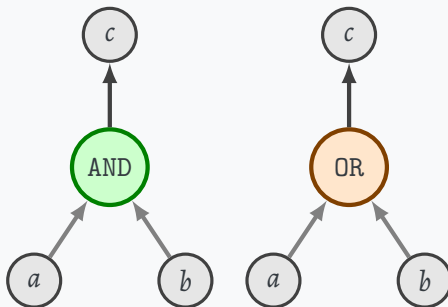


Figure: Boolean AND and OR Gates

But nothing stops us from using something more powerful instead of boolean values...

Recap. Arbitrary Program To Circuits

We can do that in a way like the computer does it — **boolean circuits**.

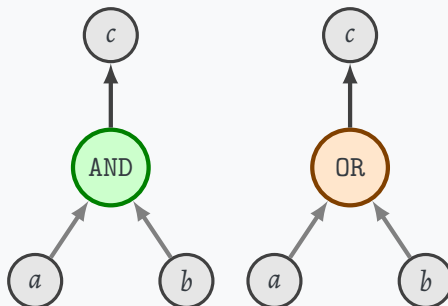


Figure: Boolean AND and OR Gates

> **100000 gates** just for SHA256...

Recap. Arbitrary Program To Circuits

We can do that in a way like the computer does it — **boolean circuits**.

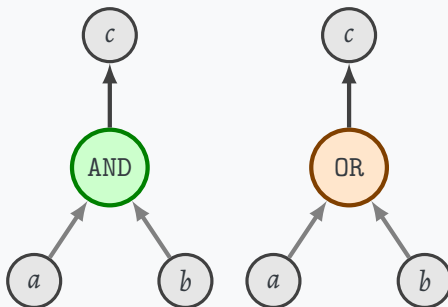


Figure: Boolean AND and OR Gates

> **100000 gates** just for SHA256... But nothing stops us from using something more powerful instead of boolean values, gates.

Recap. Arbitrary Program To Circuits

Similar to Boolean Circuits, the **Arithmetic Circuits** consist of gates and wires.

- **Wires:** elements of some finite field \mathbb{F} .
- **Gates:** field addition (+) and multiplication (\times).

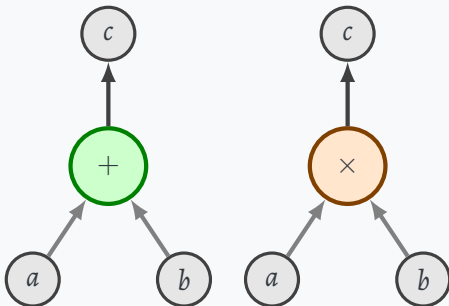


Figure: Addition and Multiplication Gates

Recap. Arbitrary Program To Circuits

Example

How can we translate if statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

Recap. Arbitrary Program To Circuits

Example

How can we translate if statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Recap. Arbitrary Program To Circuits

Example

How can we translate if statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Corresponding equations for the circuit are:

$$r_1 = b \times c, \quad r_3 = 1 - a, \quad r_5 = r_3 \times r_2$$

$$r_2 = b + c, \quad r_4 = a \times r_1, \quad r = r_4 + r_5$$

Recap. Arbitrary Program To Circuits

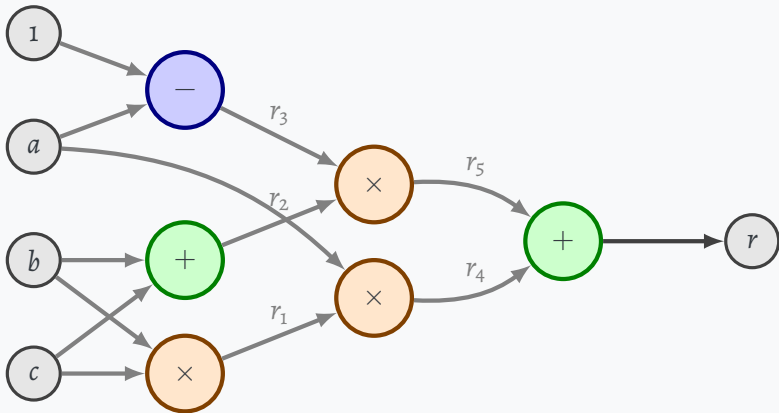


Figure: Example of a circuit evaluating the if statement logic.

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Where $\langle \mathbf{u}, \mathbf{v} \rangle$ is a dot product.

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^n u_i v_i$$

Recap. R1CS

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Where $\langle \mathbf{u}, \mathbf{v} \rangle$ is a dot product.

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^n u_i v_i$$

Thus

$$\left(\sum_{i=1}^n a_i w_i \right) \times \left(\sum_{j=1}^n b_j w_j \right) = \sum_{k=1}^n c_k w_k$$

That is, actually, a quadratic equation with multiple variables.

Recap. R1CS

Example

Consider the most basic circuit with one multiplication gate:

$x_1 \times x_2 = r$. The witness vector $\mathbf{w} = (r, x_1, x_2)$. So

$$w_2 \times w_3 = w_1$$

$$(0 + w_2 + 0) \times (0 + 0 + w_3) = w_1 + 0 + 0$$

$$(0w_1 + 1w_2 + 0w_3) \times (0w_1 + 0w_2 + 1w_3) = 1w_1 + 0w_2 + 0w_3$$

Therefore the coefficients vectors are:

$$\mathbf{a} = (0, 1, 0), \quad \mathbf{b} = (0, 0, 1), \quad \mathbf{c} = (1, 0, 0).$$

The general form of our constraint is:

$$(a_1w_1 + a_2w_2 + a_3w_3)(b_1w_1 + b_2w_2 + b_3w_3) = c_1w_1 + c_2w_2 + c_3w_3$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad (\text{binary check}) \quad (1)$$

$$x_2 \times x_3 = \text{mult} \quad (2)$$

$$x_1 \times \text{mult} = \text{selectMult} \quad (3)$$

$$(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult} \quad (4)$$

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad (\text{binary check}) \quad (1)$$

$$x_2 \times x_3 = \text{mult} \quad (2)$$

$$x_1 \times \text{mult} = \text{selectMult} \quad (3)$$

$$(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult} \quad (4)$$

The witness vector: $\mathbf{w} = (1, r, x_1, x_2, x_3, \text{mult}, \text{selectMult})$.

Recap. R1CS

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad (\text{binary check}) \quad (1)$$

$$x_2 \times x_3 = \text{mult} \quad (2)$$

$$x_1 \times \text{mult} = \text{selectMult} \quad (3)$$

$$(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult} \quad (4)$$

The witness vector: $\mathbf{w} = (1, r, x_1, x_2, x_3, \text{mult}, \text{selectMult})$.

The coefficients vectors:

$$\mathbf{a}_1 = (0, 0, 1, 0, 0, 0, 0), \quad \mathbf{b}_1 = (0, 0, 1, 0, 0, 0, 0), \quad \mathbf{c}_1 = (0, 0, 1, 0, 0, 0, 0)$$

$$\mathbf{a}_2 = (0, 0, 0, 1, 0, 0, 0), \quad \mathbf{b}_2 = (0, 0, 0, 0, 1, 0, 0), \quad \mathbf{c}_2 = (0, 0, 0, 0, 0, 1, 0)$$

$$\mathbf{a}_3 = (0, 0, 1, 0, 0, 0, 0), \quad \mathbf{b}_3 = (0, 0, 0, 0, 0, 1, 0), \quad \mathbf{c}_3 = (0, 0, 0, 0, 0, 0, 1)$$

$$\mathbf{a}_4 = (1, 0, -1, 0, 0, 0, 0), \quad \mathbf{b}_4 = (0, 0, 0, 1, 1, 0, 0), \quad \mathbf{c}_4 = (0, 1, 0, 0, 0, 0, -1)$$

Quadratic Arithmetic Program

Problems we have for now:

Problems we have for now:

- ✓ Although Rank-1 Constraint Systems provide a powerful method for representing computations, they are not succinct.

Problems we have for now:

- ✓ Although Rank-1 Constraint Systems provide a powerful method for representing computations, they are not succinct.
- ✓ We need to transform our computations into a form that is more convenient for proving statements about them.

Notice

A very convenient form for representing computations is **polynomials!**

Idea: Instead of checking polynomial equality $P(x) = Q(x)$ at multiple points $Q(x_1), \dots, Q(x_n)$ (essentially, checking each constraint), we check it only once at $\tau \xleftarrow{R} \mathbb{F}$: $P(\tau) = Q(\tau)$. Soundness is guaranteed by the **Schwartz-Zippel Lemma**.

We finished with:

$$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, \quad \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m, \quad \mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m,$$

We finished with:

$$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, \quad \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m, \quad \mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m,$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

We finished with:

$$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, \quad \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m, \quad \mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m,$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

An example of a single “if” statement:

$$\mathbf{a}_1 = (0, 0, 1, 0, 0, 0, 0)$$

$$\mathbf{a}_2 = (0, 0, 0, 1, 0, 0, 0)$$

$$\mathbf{a}_3 = (0, 0, 1, 0, 0, 0, 0)$$

$$\mathbf{a}_4 = (1, 0, -1, 0, 0, 0, 0)$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We finished with:

$$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m, \quad \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m, \quad \mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m,$$

Of course, they form corresponding matrices:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

An example of a single “if” statement:

$$\mathbf{a}_1 = (0, 0, 1, 0, 0, 0, 0)$$

$$\mathbf{a}_2 = (0, 0, 0, 1, 0, 0, 0)$$

$$\mathbf{a}_3 = (0, 0, 1, 0, 0, 0, 0)$$

$$\mathbf{a}_4 = (1, 0, -1, 0, 0, 0, 0)$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Pleeeeeeenty of zeroes, right? And this is just one out of 3 matrices...

The previous witness vector:

$$\mathbf{w} = (1, r, \overset{3}{\boxed{x_1}}, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{bmatrix} 0 & 0 & \overset{3}{\boxed{1}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The previous witness vector:

$$\mathbf{w} = (1, r, \overset{3}{x_1}, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{bmatrix} 0 & 0 & \overset{3}{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Consider 4th constraint: $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult}$

$$\begin{bmatrix} 0 & 0 & \overset{3}{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

The previous witness vector:

$$\mathbf{w} = (1, r, \overset{3}{\boxed{x_1}}, x_2, x_3, \text{mult}, \text{selectMult})$$

Let's take a closer look at the matrix columns:

$$\begin{bmatrix} \circ & \circ & \overset{3}{\boxed{1}} & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & 1 & \circ & \circ & \circ \\ \circ & \circ & 1 & \circ & \circ & \circ & \circ \\ 1 & \circ & -1 & \circ & \circ & \circ & \circ \end{bmatrix}$$

Consider 4th constraint: $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult}$

$$\begin{bmatrix} \circ & \circ & \overset{3}{\boxed{1}} & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & 1 & \circ & \circ & \circ \\ \circ & \circ & 1 & \circ & \circ & \circ & \circ \\ 4 \quad \boxed{1} & \circ & \boxed{-1} & \circ & \circ & \circ & \circ \end{bmatrix}$$

So, every column is a mapping of constraint number to a coefficient for the witness element.

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^n y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.$$

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^n y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j}, \quad i \in \{1, 2, \dots, m\}, \quad j \in \{1, 2, \dots, n\}$$

As we know, such a mapping can be builds using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^n y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j}, \quad i \in \{1, 2, \dots, m\}, \quad j \in \{1, 2, \dots, n\}$$

The same is true for matrices B and C , with $3n$ polynomials in total, n for each of the coefficients matrices:

$$A_1(x), A_2(x), \dots, A_n(x), B_1(x), B_2(x), \dots, B_n(x), C_1(x), C_2(x), \dots, C_n(x)$$

As we know, such a mapping can be built using Lagrange interpolation polynomial with the following formula:

$$L(x) = \sum_{i=0}^n y_i \ell_i(x), \quad \ell_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.$$

There are n columns and m constraints. So, it results in n polynomials such that:

$$A_j(i) = a_{i,j}, \quad i \in \{1, 2, \dots, m\}, \quad j \in \{1, 2, \dots, n\}$$

The same is true for matrices B and C , with $3n$ polynomials in total, n for each of the coefficient matrices:

$$A_1(x), A_2(x), \dots, A_n(x), B_1(x), B_2(x), \dots, B_n(x), C_1(x), C_2(x), \dots, C_n(x)$$

Note

We could have assigned any *unique* index from \mathbb{F} to each constraint (say, t_i for each $i \in \{1, \dots, m\}$) and interpolate through these points:

$$A_j(t_i) = a_{i,j}, \quad i \in \{1, 2, \dots, m\}, \quad j \in \{1, 2, \dots, n\}$$

Example

Considering the witness vector \mathbf{w} and matrix A from the previous example, for the variable x_1 , the next set of points can be derived:

$$\{(1, 1), (2, 0), (3, 1), (4, -1)\}$$

The Lagrange interpolation polynomial for this set of points:

$$\begin{aligned} \ell_1(x) &= -\frac{(x-2)(x-3)(x-4)}{6}, & \ell_2(x) &= \frac{(x-1)(x-3)(x-4)}{2}, \\ \ell_3(x) &= -\frac{(x-1)(x-2)(x-4)}{2}, & \ell_4(x) &= \frac{(x-1)(x-2)(x-3)}{6}. \end{aligned}$$

Thus, the polynomial is given by:

$$\begin{aligned} A_{x_1}(x) &= 1 \cdot \ell_1(x) + 0 \cdot \ell_2(x) + 1 \cdot \ell_3(x) + (-1) \cdot \ell_4(x) \\ &= -\frac{5}{6}x^3 + 6x^2 - \frac{79}{6}x + 9 \end{aligned}$$

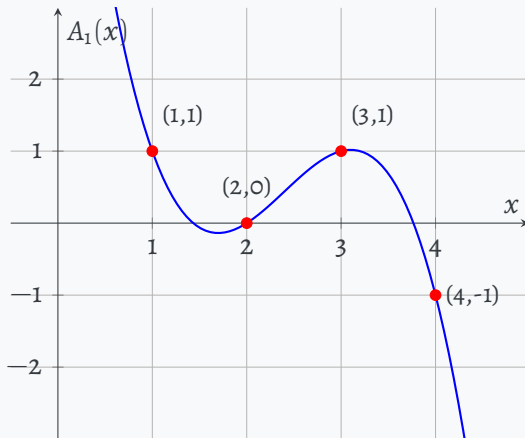


Illustration: The Lagrange interpolation polynomial for points $\{(1, 1), (2, 0), (3, 1), (4, -1)\}$ visualized over \mathbb{R} .

Question

But what does it change? We “exchanged” $3n$ columns for $3n$ polynomials.

Question

But what does it change? We “exchanged” $3n$ columns for $3n$ polynomials.

Consider two polynomials $p(x)$ and $q(x)$:

$$p(x) = -\frac{1}{2}x^2 + \frac{3}{2}x, \quad q(x) = \frac{1}{3}x^3 - 2x^2 + \frac{8}{3}x + 1.$$

With corresponding sets of points:

$$\{(0, 0), (1, 1), (2, 1), (3, 0)\}, \quad \{(0, 1), (1, 2), (2, 1), (3, 0)\}$$

Question

But what does it change? We “exchanged” $3n$ columns for $3n$ polynomials.

Consider two polynomials $p(x)$ and $q(x)$:

$$p(x) = -\frac{1}{2}x^2 + \frac{3}{2}x, \quad q(x) = \frac{1}{3}x^3 - 2x^2 + \frac{8}{3}x + 1.$$

With corresponding sets of points:

$$\{(0, 0), (1, 1), (2, 1), (3, 0)\}, \quad \{(0, 1), (1, 2), (2, 1), (3, 0)\}$$

The sum of these polynomials can be calculated as:

$$r(x) = \frac{1}{3}x^3 - 2\frac{1}{2}x^2 + 4\frac{1}{6}x + 1$$

The resulting polynomial $r(x)$ corresponds to the set of points:

$$\{(0, 1), (1, 3), (2, 2), (3, 0)\}$$

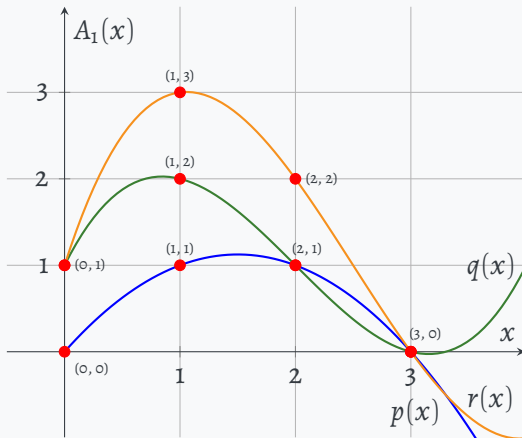


Figure: Addition of two polynomials

Now, using coefficients encoded with polynomials, we can build a constraint number $X \in \{1, \dots, m\}$ in the next way:

$$\begin{aligned} & (w_1 A_1(X) + w_2 A_2(X) + \dots + w_n A_n(X)) \times \\ & \times (w_1 B_1(X) + w_2 B_2(X) + \dots + w_n B_n(X)) = \\ & = (w_1 C_1(X) + w_2 C_2(X) + \dots + w_n C_n(X)) \end{aligned}$$

Now, using coefficients encoded with polynomials, we can build a constraint number $X \in \{1, \dots, m\}$ in the next way:

$$\begin{aligned} & (w_1A_1(X) + w_2A_2(X) + \dots + w_nA_n(X)) \times \\ & \times (w_1B_1(X) + w_2B_2(X) + \dots + w_nB_n(X)) = \\ & = (w_1C_1(X) + w_2C_2(X) + \dots + w_nC_n(X)) \end{aligned}$$

Or written more concisely:

$$\left(\sum_{i=1}^n w_i A_i(X) \right) \times \left(\sum_{i=1}^n w_i B_i(X) \right) = \left(\sum_{i=1}^n w_i C_i(X) \right)$$

Hold on, but why does it hold? Let us substitute any $X = j$ into this equation:

$$\left(\sum_{i=1}^n w_i A_i(j) \right) \times \left(\sum_{i=1}^n w_i B_i(j) \right) = \left(\sum_{i=1}^n w_i C_i(j) \right) \quad \forall j \in \{1, \dots, m\}$$

Hold on, but why does it hold? Let us substitute any $X = j$ into this equation:

$$\left(\sum_{i=1}^n w_i A_i(j) \right) \times \left(\sum_{i=1}^n w_i B_i(j) \right) = \left(\sum_{i=1}^n w_i C_i(j) \right) \quad \forall j \in \{1, \dots, m\}$$

Recall that we interpolated polynomials to have $A_i(j) = a_{j,i}$. Therefore, the equation above can be reduced to:

$$\left(\sum_{i=1}^n w_i a_{j,i} \right) \times \left(\sum_{i=1}^n w_i b_{j,i} \right) = \left(\sum_{i=1}^n w_i c_{j,i} \right) \quad \forall j \in \{1, \dots, m\}$$

Hold on, but why does it hold? Let us substitute any $X = j$ into this equation:

$$\left(\sum_{i=1}^n w_i A_i(j) \right) \times \left(\sum_{i=1}^n w_i B_i(j) \right) = \left(\sum_{i=1}^n w_i C_i(j) \right) \quad \forall j \in \{1, \dots, m\}$$

Recall that we interpolated polynomials to have $A_i(j) = a_{j,i}$. Therefore, the equation above can be reduced to:

$$\left(\sum_{i=1}^n w_i a_{j,i} \right) \times \left(\sum_{i=1}^n w_i b_{j,i} \right) = \left(\sum_{i=1}^n w_i c_{j,i} \right) \quad \forall j \in \{1, \dots, m\}$$

But hold on again! Notice that $\sum_{i=1}^n w_i a_{j,i} = \langle \mathbf{w}, \mathbf{a}_j \rangle$ and therefore we have:

$$\langle \mathbf{w}, \mathbf{a}_j \rangle \times \langle \mathbf{w}, \mathbf{b}_j \rangle = \langle \mathbf{w}, \mathbf{c}_j \rangle \quad \forall j \in \{1, \dots, m\},$$

so we ended up with the initial m constraint equations!

Now let us define polynomials $A(X)$, $B(X)$, $C(X)$ for easier notation:

$$A(X) = \sum_{i=1}^n w_i A_i(X), \quad B(X) = \sum_{i=1}^n w_i B_i(X), \quad C(X) = \sum_{i=1}^n w_i C_i(X)$$

Now let us define polynomials $A(X)$, $B(X)$, $C(X)$ for easier notation:

$$A(X) = \sum_{i=1}^n w_i A_i(X), \quad B(X) = \sum_{i=1}^n w_i B_i(X), \quad C(X) = \sum_{i=1}^n w_i C_i(X)$$

Therefore:

$$A(X) \times B(X) = C(X)$$

Now, we can define a polynomial $M(X)$, that has zeros at all elements from the set $\Omega = \{1, \dots, m\}$

$$M(X) = A(X) \times B(X) - C(X)$$

Now let us define polynomials $A(X)$, $B(X)$, $C(X)$ for easier notation:

$$A(X) = \sum_{i=1}^n w_i A_i(X), \quad B(X) = \sum_{i=1}^n w_i B_i(X), \quad C(X) = \sum_{i=1}^n w_i C_i(X)$$

Therefore:

$$A(X) \times B(X) = C(X)$$

Now, we can define a polynomial $M(X)$, that has zeros at all elements from the set $\Omega = \{1, \dots, m\}$

$$M(X) = A(X) \times B(X) - C(X)$$

It means, that $M(X)$ can be divided by **vanishing polynomial** $Z_\Omega(X)$ without a remainder!

$$Z_\Omega(X) = \prod_{i=1}^m (X - i), \quad H(X) = \frac{M(X)}{Z_\Omega(X)} \text{ is a polynomial}$$

Definition (Quadratic Arithmetic Program)

Suppose that m R1CS constraints with a witness of size n are written in a form

$$A\mathbf{w} \odot B\mathbf{w} = C\mathbf{w}, \quad (A, B, C \in \mathbb{F}^{m \times n})$$

Then, the **Quadratic Arithmetic Program** consists of $3n$ polynomials $A_1, \dots, A_n, B_1, \dots, B_n, C_1, \dots, C_n$ such that:

$$A_j(i) = a_{ij}, \quad B_j(i) = b_{ij}, \quad C_j(i) = c_{ij}, \quad \forall i \in \{1, \dots, m\} \quad \forall j \in \{1, \dots, n\}$$

Then, $\mathbf{w} \in \mathbb{F}^n$ is a valid assignment for the given QAP and **target polynomial** $Z(X) = \prod_{i=1}^m (X - i)$ if and only if there exists such a polynomial $H(X)$ such that

$$\left(\sum_{i=1}^n w_i A_i(X) \right) \left(\sum_{i=1}^n w_i B_i(X) \right) - \left(\sum_{i=1}^n w_i C_i(X) \right) = Z(X) H(X)$$

Probabilistically Checkable Proofs

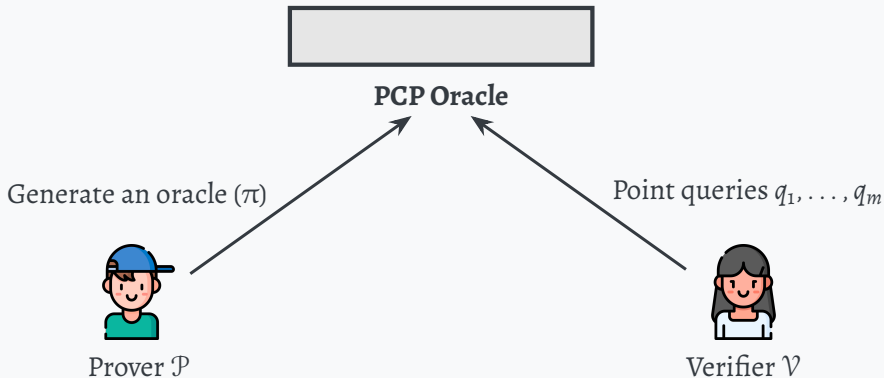


Figure: Illustration of a Probabilistically Checkable Proof (PCP) system. The prover \mathcal{P} generates a PCP oracle π that is queried by the verifier \mathcal{V} at specific points q_1, \dots, q_m .

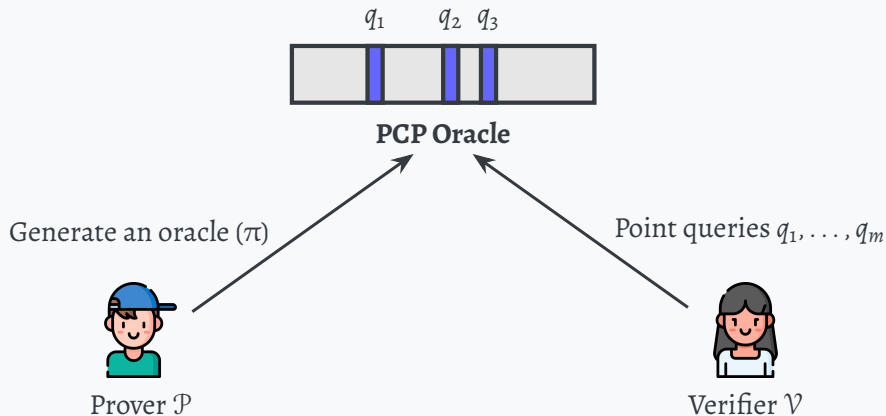


Figure: Illustration of a Probabilistically Checkable Proof (PCP) system. The prover \mathcal{P} generates a PCP oracle π that is queried by the verifier \mathcal{V} at specific points q_1, \dots, q_m .

Three main extensions of PCPs that are frequently used in SNARKs are:

- **IPCP (Interactive PCP)**: The prover commits to the PCP oracle and then, based on the interaction between the prover and verifier, the verifier queries the oracle and decides whether to accept the proof.
- **IOP (Interactive Oracle Proof)**: The prover and verifier interact and on each round, the prover commits to a new oracle. The verifier queries the oracle and decides whether to accept the proof.
- **LPCP (Linear PCP)**: The prover commits to a linear function and the verifier queries the function at specific points.

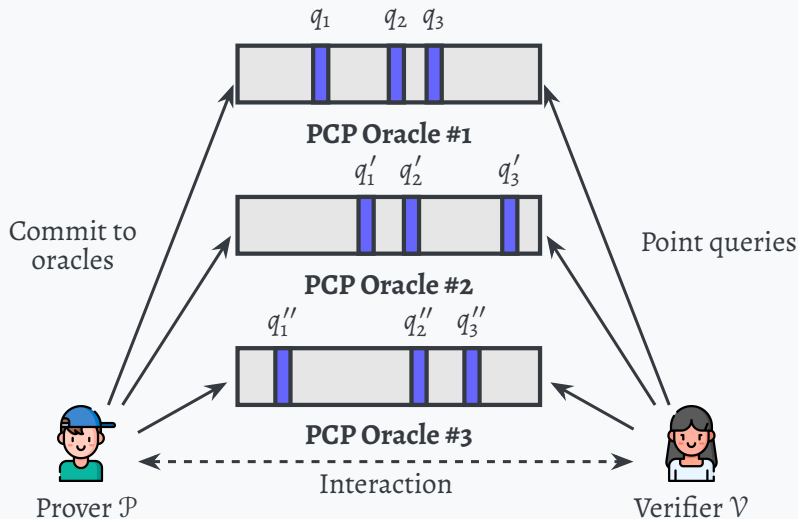


Figure: Illustration of an Interactive Oracle Proof (IOP). On each round i ($1 \leq i \leq r$), \mathcal{V} sends a message m_i , and \mathcal{P} commits to a new oracle π_i , which \mathcal{V} can query at $\mathbf{q}_i = (q_{i,1}, \dots, q_{i,m})$.

QAP as a Linear PCP

Definition (Linear PCP)

A **Linear PCP** is a PCP where the prover commits to a linear function $\pi = (\pi_1, \dots, \pi_k)$ and the verifier queries the function at specific points q_1, \dots, q_r . Then, the prover responds with the values of the function at these points:

$$\langle \pi_1, q_1 \rangle, \langle \pi_2, q_2 \rangle, \dots, \langle \pi_r, q_r \rangle.$$

Example (QAP as a Linear PCP)

Recall that key QAP equation is:

$$L(x) \times R(x) - O(x) = Z(x)H(x).$$

Now, consider the following **linear PCP for QAP**:

1. \mathcal{P} commits to an extended witness \mathbf{w} and coefficients $\mathbf{h} = (h_1, \dots, h_n)$ of $H(x)$.
2. \mathcal{V} samples $\gamma \xleftarrow{R} \mathbb{F}$ and sends query $\gamma = (\gamma, \gamma^2, \dots, \gamma^n)$ to \mathcal{P} .
3. \mathcal{P} reveals the following values:

$$\pi_1 \leftarrow \langle \mathbf{w}, \mathbf{L}(\gamma) \rangle,$$

$$\pi_2 \leftarrow \langle \mathbf{w}, \mathbf{R}(\gamma) \rangle,$$

$$\pi_3 \leftarrow \langle \mathbf{w}, \mathbf{O}(\gamma) \rangle,$$

$$\pi_4 \leftarrow Z(\gamma) \cdot \langle \mathbf{h}, \gamma \rangle.$$

4. \mathcal{V} checks whether $\pi_1\pi_2 - \pi_3 = \pi_4$.

Question

Why is it safe to use such a check?

The polynomials $L(x)$, $R(x)$ and $O(x)$ are interpolated polynomials using $|C|$ (number of gates) points, so:

$$\deg(L) \leq |C|, \quad \deg(R) \leq |C|, \quad \deg(O) \leq |C|$$

Question

Why is it safe to use such a check?

The polynomials $L(x)$, $R(x)$ and $O(x)$ are interpolated polynomials using $|C|$ (number of gates) points, so:

$$\deg(L) \leq |C|, \quad \deg(R) \leq |C|, \quad \deg(O) \leq |C|$$

Thus, we can estimate the degree of polynomial $M(x) = L(x)R(x) - O(x)$.

$$\deg(M) \leq \max\{\deg(L) + \deg(R), \deg(O)\} \leq 2|C|$$

Question

Why is it safe to use such a check?

The polynomials $L(x)$, $R(x)$ and $O(x)$ are interpolated polynomials using $|C|$ (number of gates) points, so:

$$\deg(L) \leq |C|, \quad \deg(R) \leq |C|, \quad \deg(O) \leq |C|$$

Thus, we can estimate the degree of polynomial $M(x) = L(x)R(x) - O(x)$.

$$\deg(M) \leq \max\{\deg(L) + \deg(R), \deg(O)\} \leq 2|C|$$

If an adversary \mathcal{A} does not know a valid witness \mathbf{w} , he can compute a polynomial $(\tilde{M}(x), \tilde{H}(x)) \leftarrow \mathcal{A}(\cdot)$ that satisfies a verifier \mathcal{V} :

$$\Pr_{s \xleftarrow{R} \mathbb{F}} [\tilde{M}(s) = Z(s)\tilde{H}(s)] \leq \frac{2|C|}{|\mathbb{F}|}$$

If $|\mathbb{F}|$ is large enough, $2|C|/|\mathbb{F}|$ is *negligible*.

Proof Of Exponent

Encrypted Verification

Let's try to prove that we know some polynomial $p(x)$ that can be divided to $t(x)$ without a remainder.

Encrypted Verification

Let's try to prove that we know some polynomial $p(x)$ that can be divided to $t(x)$ without a remainder.

Consider polynomial: $p(x) = x^2 - 5x$.

Encrypted Verification

Let's try to prove that we know some polynomial $p(x)$ that can be divided to $t(x)$ without a remainder.

Consider polynomial: $p(x) = x^2 - 5x$. And some homomorphic encryption with a generator g .

To evaluate encrypted polynomial e.q.:

$$g^{p(\tau)}$$

Encrypted Verification

Let's try to prove that we know some polynomial $p(x)$ that can be divided to $t(x)$ without a remainder.

Consider polynomial: $p(x) = x^2 - 5x$. And some homomorphic encryption with a generator g .

To evaluate encrypted polynomial e.q.:

$$g^{p(\tau)} = g^{(\tau^2 - 5\tau)}$$

Encrypted Verification

Let's try to prove that we know some polynomial $p(x)$ that can be divided to $t(x)$ without a remainder.

Consider polynomial: $p(x) = x^2 - 5x$. And some homomorphic encryption with a generator g .

To evaluate encrypted polynomial e.q.:

$$g^{p(\tau)} = g^{(\tau^2 - 5\tau)} = (g^{\tau^2})^1 (g^{\tau^1})^{-5}$$

Prover needs encrypted powers of tau: $\{g^{\tau^i}\}_{i \in [d]}$.

Encrypted Verification

Verifier:

✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}$ and $g^{h(\tau)}$.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}$ and $g^{h(\tau)}$.
- ✓ Provides encrypted polynomials $g^{p(\tau)}$ and $g^{h(\tau)}$ to the verifier.

Encrypted Verification

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}$ and $g^{h(\tau)}$.
- ✓ Provides encrypted polynomials $g^{p(\tau)}$ and $g^{h(\tau)}$ to the verifier.

Verifier:

- ✓ Checks whether $g^{p(\tau)} = (g^{h(\tau)})^{t(\tau)}$.

That doesn't work...

Verifier:

✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Adversary:

- ✓ Picks a random value $r \xleftarrow{R} \mathbb{F}$, calculates g^r .

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Adversary:

- ✓ Picks a random value $r \xleftarrow{R} \mathbb{F}$, calculates g^r .
- ✓ Calculates $g^{t(\tau)}$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Adversary:

- ✓ Picks a random value $r \xleftarrow{R} \mathbb{F}$, calculates g^r .
- ✓ Calculates $g^{t(\tau)}$.
- ✓ Calculates $g^{\tilde{p}(\tau)} = (g^{t(\tau)})^r$.

That doesn't work...

Verifier:

- ✓ Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$.
- ✓ Calculates $t(\tau)$.
- ✓ Outputs prover parameters $\{g^{\tau^i}\}_{i \in [d]}$.

Adversary:

- ✓ Picks a random value $r \xleftarrow{R} \mathbb{F}$, calculates g^r .
- ✓ Calculates $g^{t(\tau)}$.
- ✓ Calculates $g^{\tilde{p}(\tau)} = (g^{t(\tau)})^r$.

Verifier:

- ✓ Checks whether $g^{\tilde{p}(\tau)} = (g^s)^{t(\tau)}$.

Proof Of Exponent

Verifier:

✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}, g^{h(\tau)}$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}, g^{h(\tau)}$.
- ✓ Using $\{g^{a\tau^i}\}_{i \in [d]}$ calculates $g^{p'(\tau)} = g^{ap(\tau)}$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}, g^{h(\tau)}$.
- ✓ Using $\{g^{a\tau^i}\}_{i \in [d]}$ calculates $g^{p'(\tau)} = g^{ap(\tau)}$.
- ✓ Provides encrypted polynomials to the verifier.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}, g^{h(\tau)}$.
- ✓ Using $\{g^{a\tau^i}\}_{i \in [d]}$ calculates $g^{p'(\tau)} = g^{ap(\tau)}$.
- ✓ Provides encrypted polynomials to the verifier.

Verifier:

- ✓ Checks whether $g^{p(\tau)} = (g^{h(\tau)})^{t(\tau)}$.

Proof Of Exponent

Verifier:

- ✓ Picks a random values $\tau \xleftarrow{R} \mathbb{F}, a \xleftarrow{R} \mathbb{F}$.
- ✓ Calculates the public parameters $\{g^{\tau^i}\}_{i \in [d]}$ and $\{g^{a\tau^i}\}_{i \in [d]}$
- ✓ Calculates $t(\tau)$.

Prover:

- ✓ Calculates $h(x) = \frac{p(x)}{t(x)}$.
- ✓ Using $\{g^{\tau^i}\}_{i \in [d]}$ calculates $g^{p(\tau)}, g^{h(\tau)}$.
- ✓ Using $\{g^{a\tau^i}\}_{i \in [d]}$ calculates $g^{p'(\tau)} = g^{ap(\tau)}$.
- ✓ Provides encrypted polynomials to the verifier.

Verifier:

- ✓ Checks whether $g^{p(\tau)} = (g^{h(\tau)})^{t(\tau)}$.
- ✓ Checks whether $g^{p'(\tau)} = (g^{p(\tau)})^a = g^{ap(\tau)}$.

Thanks for your attention!