Pinocchio – Short Signatures for Computation

– A Pen&Paper Example –

Mirco Richter[1]
October 22, 2019

[1]LeastAuthority

Introduction

What we do today

- Major takeaway: Some understanding of zero knowledge, verified computations.
- Use an actual (3-bit security) cryptographic scheme on an oversimplified problem as running example.
- Use the Pinocchio protocol to derive a toy verified computation.

References and Further Readings

- Original Source: Gentry, Howell, Parno and Raykova (2013): "Pinocchio: Nearly Practical Verifiable Computation". In: 2013 IEEE Symposium on Security and Privacy.
- Optimized version: Jens Groth (2016): "On the Size of Pairing-based Non-interactive Arguments". Cryptology ePrint Archive, Report 2016/260.
- Great introduction: Maksym Petkus (2019): "Why and How zk-SNARK Works: Definitive Explanation"
- Companion paper: Mirco Richter (2018): "A (somewhat) easy pen & paper example of the Pinocchio protocol". https://drive.google.com/file/d/0B-WxC9ydKhlRZG92dnJ0RmdWRkZKUXR5Q3FTd0pZMl9Tdnln/view

Introduction

What is verified computing?

- Public key signatures are short proofs of static data.
- But there is static data and dynamic computation.

–

- Can we have signatures (short proofs) for computation?
- Can we keep certain details of the computation private, but still get verifiable signatures?

–

- ZK-SNARK $\Leftrightarrow$ **Z**ero **K**nowledge **S**uccinct **N**on-interactive **Ar**guments (of) **K**nowledge.

Introduction

Why do we need this?

One Example: Zero Knowledge Proof of Knowledge

- Task: Convince everyone, that you know a dataset, which hashes to a publicly known digest string (Knowledge of a preimage).
- Naive Solution: Publish the dataset. If the hashes are equal, everyone is convinced.
- Problem: Publishing the dataset might not be an option.
- Better Solution: Implement the hash function not native (e.g. in x86-assembly), but as a ZK-SNARK.

Roadmap

Steps:

1. The 3-bit cryptographic scheme.
2. The toy example function.
3. The algebraic circuit.
4. The quadratic arithmetic program.
5. The setup phase.
6. The worker phase.
7. The verifier phase.

Cryptographic Scheme

The cryptographic scheme

The Pinocchio protocol requires:

- A finite cyclic group $(\mathbb{G}, \cdot)$
- A generator g of that group.
- A bilinear map $B(\cdot, \cdot) : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, such that:
  - Order: $\mathbb{G}_T$ and $\mathbb{G}$ have same order
  - Biliniearity: $B(g^j, h^k) = B(g, h)^{j \cdot k}$ for all $j, k \in \mathbb{Z}, g, h \in \mathbb{G}$
  - Non-triviality: Es gibt ein $g \in \mathbb{G}$ mit $B(g, g) \neq id_{\mathbb{G}_T}$

$\Rightarrow$ Usually realized by cryptographically strong, pairing friendly elliptic curves.

Cryptographic Scheme

Our 3,5-Bit System

- $\mathbb{G} = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$.
- Multiplication: $x \bullet y := x \cdot y (mod_{23})$, e.g. ordinary integer multiplication modulo 23.
- Generator: 2.
- Non-trivial bilinear map:

$$B(\cdot, \cdot) : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_{23}^*; (g, h) \mapsto 2^{log_2(g) \cdot log_2(h)} (mod_{23})$$

Cryptographic Scheme

Our 3,5-Bit System

To get familiar with the scheme, lets compute something:

- $9 \bullet 13 =$
- $9 \cdot 13(mod_{23}) =$
- $117(mod_{23}) =$
- $(5 \cdot 23 + 2)(mod_{23}) =$
- 2

Cryptographic Scheme

Our 3,5-Bit System

The underlying finite field:

- Prime field is $\mathbb{F}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- Addition: Normal integer addition modulo 11
- Multiplication: Normal integer multiplication modulo 11

Cryptographic Scheme

Our 3,5-Bit System

To get familiar with the scheme, lets solve an equation for x in $\mathbb{F}_{11}$

- $(3 \cdot x + 4) \cdot 5 = 3x$
- $3 \cdot 5 \cdot x + 4 \cdot 5 = 3x$
- $4 \cdot x + 9 = 3x$
- $4 \cdot x - 3x = -9$
- $4 \cdot x + 8x = 2$
- $(4 + 8) \cdot x = 2$
- $1 \cdot x = 2$
- $x = 2$

Cryptographic Scheme

Our 3,5-Bit System

Exponentiation and Logarithms:

- 2 is a generator:
  - $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 9$,
  - $2^6 = 18$, $2^7 = 13$, $2^8 = 3$, $2^9 = 6$, $2^{10} = 12$, $2^{11} = 1$
- Base 2 logarithms:
  - $0 = log_2(1)$, $1 = log_2(2)$, $2 = log_2(4)$, $3 = log_2(8)$
  - $4 = log_2(16)$, $5 = log_2(9)$, $6 = log_2(18)$
  - $7 = log_2(13)$, $8 = log_2(3)$, $9 = log_2(6)$, $10 = log_2(12)$
- Interconnection is

$$2^{\bullet} : \mathbb{F}_{11} \leftrightarrow \mathbb{G} : log_2(\bullet)$$

Cryptographic Scheme

Our 3,5-Bit System

Why not use ordinary numbers?

In groups like $\mathbb{G}$, certain computations are much harder (even for computers), than similar computations for ordinary numbers are. For example it is believed that finding a solution x to the equation

$$a^x = b$$

is infeasible in actual cryptographic schemes (This is known as the discrete logarithm problem).

The Main Example

Task: Implement the following function as a SNARK in our 3,5-bit cryptographic scheme.

Function:

$$f : \mathbb{F}_{11} \times \mathbb{F}_{11} \times \mathbb{F}_{11} \to \mathbb{F}_{11}; (x_1, x_2, x_3) \mapsto (x_1 \cdot x_2) \cdot x_3$$

Setup – Algebraic Circuit Representation

The Algebraic Circuit DAG

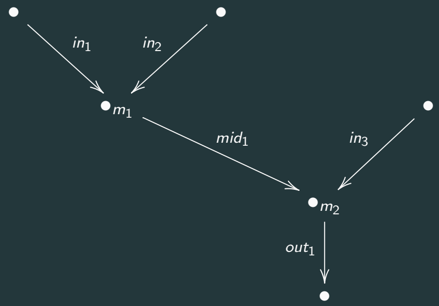
- Algebraic circuits (over field $\mathbb{F}$) are directed acyclic graphs, that represent computation:
  - vertices with only outgoing edges (leafs, sources) represent inputs to the computation.
  - vertices with only ingoing edges (roots, sinks) represent outputs from the computation.
  - internal vertices represent field operations (Either addition or multiplication).
- Circuit execution: Send input values from leafs along edges, through internal vertices to roots.
- Algebraic circuits are usually derived by Compilers, that transform higher languages to circuits.

Note: Different Compiler give very different circuit representations and Compiler optimization is important.

Setup – Algebraic Circuit Representation

Example Circuit

Valid circuit for $f : \mathbb{F}_{11} \times \mathbb{F}_{11} \times \mathbb{F}_{11} \to \mathbb{F}_{11}; (x_1, x_2, x_3) \mapsto (x_1 \cdot x_2) \cdot x_3$ is given by:



- Two multiplication vertices m_1 and m_2
- Index set $I := \{in_1, in_2, in_3, mid_1, out_1\}$

Setup – Algebraic Circuit Representation

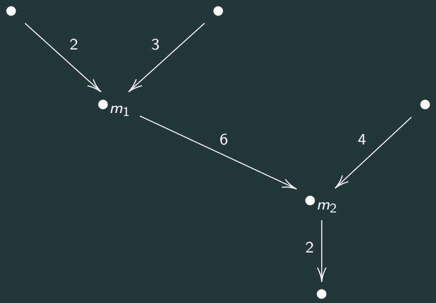
What are Assignments?

- An Assignment associates field elements to all edges (indices) in an algebraic circuit.
- An Assignment is valid, if the field element arise from executing the circuit.
- Every other assignment is invalid.
- Valid assignments are proofs for proper circuit execution.

Setup – Algebraic Circuit Representation

Example Assignments

Valid assignment: $I_{valid} := \{in_1, in_2, in_3, mid_1, out_1\} = \{2, 3, 4, 6, 2\}$

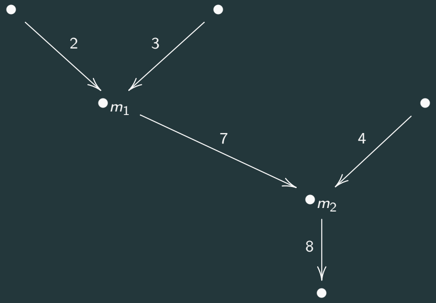


Appears from multiplying the input values at m_1, m_2 in $\mathbb{F}_{11}$

Setup – Algebraic Circuit Representation

Example Assignments

Non valid assignment: $I_{err} := \{in_1, in_2, in_3, mid_1, out_1\} = \{2, 3, 4, 7, 8\}$



Can not appear from multiplying the input values at m_1, m_2 in $\mathbb{F}_{11}$

Setup – Quadratic Arithmetic Programs

The QAP of a Circuit

- QAPs are sets of polynomials.
- QAPs are building blocks to encode circuits into polynomials t and assignments into polynomials p.
- First major point: p is divisible by t, if and only if p is derived from a valid assignment. Then another polynomial h exists with

$$p = h \cdot t$$

- Second major point: With overwhelmingly high probability, the equation can be verified in a single point s. E.g. its enough to check

$$p(s) = h(s) \cdot t(s)$$

- Checking knowledge of p and h in a single point leads towards short proofs.

Setup – Quadratic Arithmetic Programs

- How do we compute QAPs?
- I circuit indices: $QAP := \{t, \{v_k\}_{k\in I}, \{w_k\}_{k\in I}, \{y_k\}_{k\in I}\}$
- Choose random elements $\{m_1, \cdots m_k\}$ from base field for every multiplication vertex in the circuit.
- Target polynomial: $t(x) = (x - m_1) \cdot \ldots \cdot (x - m_k)$
- Polynomial from $\{v_k\}_{k\in I}$ is 1 at m_j, if edge k is left input to multiplication gate $\bullet_{m_j}$ and zero at m_j, otherwise.
- Polynomial from $\{w_k\}_{k\in I}$ is 1 at m_j, if edge k is right input to multiplication gate $\bullet_{m_j}$ and zero at point m_j, otherwise.
- A polynomial from $\{y_k\}_{k\in I}$ is 1 at m_j, if edge k is output of multiplication gate $\bullet_{m_j}$ and zero at point m_j, otherwise.
- Circuit assignment $\{c_k\}_{k\in I}$ defines the polynomial

$$p := (\sum_{k\in I} c_k v_k) \cdot (\sum_{k\in I} c_k w_k) - \sum_{k\in I} c_k y_y$$

Setup – Quadratic Arithmetic Programs

Compute Example QAP

- Two multiplication vertices. Random choice: $m_1 = 5$ and $m_2 = 7$
  - Target polynomial:
  - $t(x) = (x - m_1)(x - m_2) =$
  - $(x - 5)(x - 7) =$
  - $(x + 6)(x + 4) =$
  - $x^2 + 10x + 2$

Setup – Quadratic Arithmetic Programs

Compute Example QAP

- Compute the building blocks of p at $m_1 = 5$ and $m_2 = 7$
  - $\{v_{in_1}, v_{in_2}, v_{in_3}, v_{mid_1}, v_{out}\}$
  - $\{w_{in_1}, w_{in_2}, w_{in_3}, w_{mid_1}, w_{out}\}$
  - $\{y_{in_1}, y_{in_2}, y_{in_3}, y_{mid_1}, y_{out}\}$

Setup – Quadratic Arithmetic Programs

Compute Example QAP

- Apply Pinocchio rules to the "left edge" polynomials $v_{k \in I}$:
    - $v_{in_1}(5) = 1, \; v_{in_1}(7) = 0$
    - $v_{in_2}(5) = 0, \; v_{in_2}(7) = 0$
    - $v_{in_3}(5) = 0, \; v_{in_3}(7) = 0$
    - $v_{mid_1}(5) = 0, \; v_{mid_1}(7) = 1$
    - $v_{out}(5) = 0, \; v_{out}(7) = 0$

Setup – Quadratic Arithmetic Programs

Example QAP

- Apply Pinocchio rules to the "right edge" polynomials $w_{k \in I}$:
  - $w_{in_1}(5) = 0,\ w_{in_1}(7) = 0$
  - $w_{in_2}(5) = 1,\ w_{in_2}(7) = 0$
  - $w_{in_3}(5) = 0,\ w_{in_3}(7) = 1$
  - $w_{mid_1}(5) = 0,\ w_{mid_1}(7) = 0$
  - $w_{out}(5) = 0,\ w_{out}(7) = 0$

Setup – Quadratic Arithmetic Programs

Example QAP

- Apply Pinocchio rules to the "outgoing edge" polynomials $y_{k \in I}$:
  - $y_{in_1}(5) = 0$, $y_{in_1}(7) = 0$
  - $y_{in_2}(5) = 0$, $y_{in_2}(7) = 0$
  - $y_{in_3}(5) = 0$, $y_{in_3}(7) = 0$
  - $y_{mid_1}(5) = 1$, $y_{mid_1}(7) = 0$
  - $y_{out}(5) = 0$, $y_{out}(7) = 1$

Setup – Quadratic Arithmetic Programs

Example QAP

- Derive the actual polynomials from this.
- Our polynomials specified on two values 5 and 7.
- Linear Polynomial $q(x) = m \cdot x + b$ is fully determined by this.
- Example: For $v_{in_1}(x) = m \cdot x + b$ computation looks like this:
  - $v_{in_1}(5) = m \cdot 5 + b$ and $v_{in_1}(7) = m \cdot 7 + b$.
  - $1 = m \cdot 5 + b$ and $0 = m \cdot 7 + b$.
  - Solve this linear equation gives $m = 5$ and $b = 9$.
  - $v_{in_1}(x) = 5x + 9$

Note: Doing this is computationally expensive and a major part in the overhead of the setup phase.

Setup – Quadratic Arithmetic Programs

Example QAP

$v_{in_1}(x) = 5x + 9$	$w_{in_1}(x) = 0$	$y_{in}(x) = 0$
$v_{in_2}(x) = 0$	$w_{in_2}(x) = 5x + 9$	$y_{in}(x) = 0$
$v_{in_3}(x) = 0$	$w_{in_3}(x) = 6x + 3$	$y_{in_3}(x) = 0$
$v_{mid_1}(x) = 6x + 3$	$w_{mid_1}(x) = 0$	$y_{mid_1}(x) = 5x + 9$
$v_{out}(x) = 0$	$w_{out}(x) = 0$	$y_{out}(x) = 6x + 3$

Setup – Quadratic Arithmetic Programs

Example QAP

$$QAP_{\mathbb{F}_{11}}(C_f) = \{x^2 + 10x + 2, \left\{ \begin{array}{c} \{5x+9, 0, 0, 6x+3, 0\} \\ \{0, 5x+9, 6x+3, 0, 0\} \\ \{0, 0, 0, 5x+9, 6x+3\} \end{array} \right\}\}$$

Setup – Quadratic Arithmetic Programs

Example – Circuit Satisfiability and Polynomial Devision

- Remember: $\{c_k\}_{k\in I}$ is valid assignment $\Leftrightarrow$ p is divisible by t.

$$p := (\sum_{k\in I} c_k v_k) \cdot (\sum_{k\in I} c_k w_k) - \sum_{k\in I} c_k y_y$$

- Valid example $I = \{2, 3, 4, 6, 2\}$:
- $(2(5x+9)+6(6x+3)) \cdot (3(5x+9)+4(6x+3)) - (6(5x+9)+2(6x+3)) =$
- $(2\cdot5x+2\cdot9+6\cdot6x+6\cdot3)\cdot(3\cdot5x+3\cdot9+4\cdot6x+4\cdot3)-(6\cdot5x+6\cdot9+2\cdot6x+2\cdot3) =$
- $(10x+7+3x+7)\cdot(4x+5+2x+1)-(8x+10+1x+6) =$
- $(2x+3)\cdot(6x+6)-(9x-5) =$
- $x^2+x+7x+7+2x+6$
- $\Rightarrow p(x) = x^2+10x+2$ – Equal to t hence divisible

Setup – Quadratic Arithmetic Programs

Example – Circuit Satisfiability and Polynomial Devision

- Remember: $\{c_k\}_{k\in I}$ is valid assignment $\Leftrightarrow$ p is divisible by t.

$$p := (\sum_{k\in I} c_k v_k)\cdot(\sum_{k\in I} c_k w_k) - \sum_{k\in I} c_k y_y$$

- Non valid example $I = \{2,3,4,5,9\}$:
- $(2(5x+9)+5(6x+3))\cdot(3(5x+9)+4(6x+3))-(5(5x+9)+9(6x+3)) =$
- $(2\cdot5x+2\cdot9+5\cdot6x+5\cdot3)\cdot(3\cdot5x+3\cdot9+4\cdot6x+4\cdot3)-(5\cdot5x+5\cdot9+9\cdot6x+9\cdot3) =$
- $(10x+7+8x+4)\cdot(4x+5+2x+1)-(3x+1+10x+5) =$
- $7x\cdot(6x+6)-(2x+6) =$
- $(9x^2+9x)+9x+5 =$
- $\Rightarrow p(x) = 9x^2+7x+5$
- Not divisible by t: $(9x^2+7x+5) : (x^2+10x+2) = 9+\frac{5x+4}{x^2+10x+2}$

Setup – Common Reference String

The trusted setup phase

Suppose cryptographic scheme, circuit and QAP is public knowledge now.
Trusted third party then generates the following data:

- random elements $r_v, r_w, s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \in \mathbb{F}$
- Proofer key $PK_{QAP(C_f)}$.
- Verifier key $VK_{QAP(C_f)}$.
- Toxic waste: Must delete random elements after key generation.

Setup – Common Reference String

Proofer Key $PK_{QAP(C_f)}$

- Given generator g and circuit degree d:

$$\left\{ \begin{array}{lll} \{g^{r_v v_k(s)}\}_{k \in I_{mid}} & \{g^{r_w w_k(s)}\}_{k \in I_{mid}} & \{g^{r_v r_w y_k(s)}\}_{k \in I_{mid}} \\ \{g^{r_v \alpha_v v_k(s)}\}_{k \in I_{mid}} & \{g^{r_w \alpha_w w_k(s)}\}_{k \in I_{mid}} & \{g^{r_v r_w \alpha_y y_k(s)}\}_{k \in I_{mid}} \\ \{g^{s^i}\}_{i \in \{1,\ldots,d\}} & & \{g^{\beta(r_v v_k(s) + r_w w_k(s) + r_v r_w y_k(s))}\}_{k \in I_{mid}} \end{array} \right\}$$

- Set of group elements, used to encrypt the non I/O-related part of polynomial p.
- Size depends linear on the number of internal (non I/O) edges in the circuit.
- Not unique.

Setup – Common Reference String

Verifier Key $VK_{QAP(C_f)}$

- Given generator g:

$$\left\{ \begin{array}{llll} g^{1} & g^{\alpha_v} & g^{\alpha_w} & g^{\alpha_\gamma} \\ g^{\gamma} & g^{\beta\gamma} & g^{t(s)} & \\ & & & \{g^{r_v v_k(s)}, g^{r_w w_k(s)}, g^{r_v r_w y_k(s)}\}_{k \in I_{I/O}} \end{array} \right\}$$

- Set of group elements, used to encrypt the I/O part of polynomial p.
- Size depends linear on the number of I/O-edges in the circuit.
- Not unique.

Setup – Common Reference String

Example Random Elements

- $r_v = 9$, $r_w = 8$, $s = 7$, $\alpha_v = 6$, $\alpha_w = 5$
- $\alpha_y = 4$, $\beta = 3$, $\gamma = 2$
- $r_y = r_v \cdot r_w = 9 \cdot 8 = 6$

Setup – Common Reference String

Encrypted Random Elements

- Using our generator 2, we write these elements in the exponent:
- $g_v = g^{r_v} = 2^9 = 6$
- $g_w = g^{r_w} = 2^8 = 3$
- $g_y = g^{r_y} = 2^6 = 18$
- $g^{\alpha_v} = 2^6 = 18$
- $g^{\alpha_w} = 2^5 = 9$
- $g^{\alpha_y} = 2^4 = 16$
- $g^{\gamma} = 2^2 = 4$
- $g^{\beta\gamma} = 2^6 = 18$

Setup – Common Reference String

Example Proofer key

$$\left\{\begin{array}{lll} \{6^{v_{mid_1}(7)}\}, & \{3^{w_{mid_1}(7)}\}, & \{18^{y_{mid_1}(7)}\}, \\ \{6^{6\cdot v_{mid_1}(7)}\}, & \{3^{5\cdot w_{mid_1}(7)}\}, & \{18^{4\cdot y_{mid_1}(7)}\} \\ \{2^7, 2^{7^2}\}, & & \{6^{3\cdot v_{mid_1}(7)} \cdot 3^{3\cdot w_{mid_1}(7)} \cdot 18^{3\cdot y_{mid_1}(7)}\} \end{array}\right\}$$

$$\left\{\begin{array}{lll} \{6^{6\cdot 7+3}\}, & \{3^{0}\}, & \{18^{5\cdot 7+9}\}, \\ \{6^{6\cdot(6\cdot 7+3)}\}, & \{3^{5\cdot 0}\}, & \{18^{4\cdot(5\cdot 7+9)}\} \\ \{2^7, 2^{7^2}\}, & & \{6^{3\cdot(6\cdot 7+3)} \cdot 3^{3\cdot 0} \cdot 18^{3\cdot(5\cdot 7+9)}\} \end{array}\right\}$$

$$\left\{\begin{array}{lll} \{6^{9+3}\}, & \{3^{0}\}, & \{18^{2+9}\}, \\ \{6^{6\cdot(9+3)}\}, & \{3^{5\cdot 0}\}, & \{18^{4\cdot(2+9)}\} \\ \{2^7, 2^{5}\}, & & \{6^{3\cdot(9+3)} \cdot 3^{3\cdot 0} \cdot 18^{3\cdot(2+9)}\} \end{array}\right\}$$

Setup – Common Reference String

Example Proofer key

$$PK_{QAP(C_f)} = \left\{ \begin{array}{lll} \{6\}, & \{1\}, & \{1\}, \\ \{12\}, & \{1\}, & \{1\} \\ \{13, 9\}, & & \{9\} \end{array} \right\}$$

Setup – Common Reference String

Example Verifier key

$$\left\{\begin{array}{llll} g^{1} & g^{\alpha_v} & g^{\alpha_w} & g^{\alpha_y} \\ g^{\gamma} & g^{\beta\gamma} & g^{t(s)} & \\ g_v^{v_0(s)} & g_w^{w_0(s)} & g_y^{y_0(s)} & \\ g_v^{v_{in_1}(s)} & g_w^{w_{in_1}(s)} & g_y^{y_{in_1}(s)} & \\ g_v^{v_{in_2}(s)} & g_w^{w_{in_2}(s)} & g_y^{y_{in_2}(s)} & \\ g_v^{v_{in_3}(s)} & g_w^{w_{in_3}(s)} & g_y^{y_{in_3}(s)} & \\ g_v^{v_{out}(s)} & g_w^{w_{out}(s)} & g_y^{y_{out}(s)} & \end{array}\right\}$$

$$\left\{\begin{array}{llll} 2 & 18 & 9 & 16 \\ 4 & 18 & 2^{t(7)} & \\ 6^{0} & 3^{0} & 18^{0} & \\ 6^{v_{in_1}(7)} & 3^{w_{in_1}(7)} & 18^{y_{in_1}(7)} & \\ 6^{v_{in_2}(7)} & 3^{w_{in_2}(7)} & 18^{y_{in_2}(7)} & \\ 6^{v_{in_3}(7)} & 3^{w_{in_3}(7)} & 18^{y_{in_3}(7)} & \\ 6^{v_{out}(7)} & 3^{w_{out}(7)} & 18^{y_{out}(7)} & \end{array}\right\}$$

Setup – Common Reference String

Example Verifier key

$$VK(PK_{QAP(C_f)}) = \left\{ \begin{array}{llll} 2 & 18 & 9 & 16 \\ 4 & 18 & 1 & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \\ 1 & 3 & 1 & \\ 1 & 1 & 18 & \end{array} \right\}$$

Setup – Common Reference String

Example Common Reference String

$$CRS_{QAP(C_f)} = \left\{ \left\{ \begin{array}{lll} \{6\}, & \{1\}, & \{1\}, \\ \{12\}, & \{1\}, & \{1\} \\ \{13, 9\}, & & \{9\} \end{array} \right\}, \left\{ \begin{array}{llll} 2 & 18 & 9 & 16 \\ 4 & 18 & 1 & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & \\ 1 & 3 & 1 & \\ 1 & 1 & 18 & \end{array} \right\} \right\}$$

Worker phase

Proof generation

- Computation: Given input set I_{in}, execute circuit C_f to compute intermediate values I_{mid} and result I_{out}.
- Proof Generation:
  - Use valid assignment I and QAP to compute polynomial p.
  - Derive quotient polynomial $h = p/t$.
  - Use proofer key $PK_{QAP(C_f)}$ to compute $\pi_{PK_{QAP(C_f)}}(I)$:

$$\left\{ \begin{array}{cccc} g^{r_v v_m(s)}, & g^{r_w w_m(s)}, & g^{r_v r_w y_m(s)}, & g^{h(s)} \\ g^{r_v \alpha_v v_m(s)}, & g^{r_w \alpha_w w_m(s)}, & g^{r_v r_w \alpha_y y_m(s)} & \\ & & & g^{r_v \beta v_m(s)} \cdot g^{r_w \beta w_m(s)} \cdot g^{r_v r_w \beta y_m(s)} \end{array} \right\}$$

Worker phase

Proof generation

- $v_m(x) = \sum_{k \in I_{mid}} c_k v_k(x)$
- $w_m(x) = \sum_{k \in I_{mid}} c_k w_k(x)$
- $y_m(x) = \sum_{k \in I_{mid}} c_k y_k(x)$
- Proof has constant size and consists of exactly 8 group elements, independent from the circuit size.

Worker phase

How is a proof generated from the proofer key?

- All v_k's, w_k's and y_k's are part of the QAP.
- Worker does not know g^{r_v}, g^{r_w}, $g^{r_v r_w}$, α_v, α_w,α_y, s, or β, because deleted after key generation by trusted party.
- Worker uses Proofer-key and exponential laws to generate the proof
  - $g^x \cdot g^y = g^{x+y}$
  - $(g^x)^y = g^{x \cdot y}$
- Since all c_k are known from execution and all $g^{r_v v_k(s)}$, $g^{r_v \alpha_v v_k(s)}$ are provided in the proofer key:

$$g^{r_v v_m(s)} = g^{r_v \sum_{k \in I_{mid}} c_k v_k(s)} = \Pi_{k \in I_{mid}} (g^{r_v v_k(s)})^{c_k}$$

$$g^{r_v \alpha_v v_m(s)} = g^{r_v \sum_{k \in I_{mid}} c_k \alpha_v v_k(s)} = \Pi_{k \in I_{mid}} (g^{r_v \alpha v_k(s)})^{c_k}$$

Worker phase

Example Proof generation

- Since we only have a single middle index $c_{mid} = 6$ we get the proof

$$\left\{ \begin{array}{cccc} 6^6, & 1^6, & 1^6, & 2 \\ 12^6, & 1^6, & 1^6 & \\ & & & 9^6 \end{array} \right\}$$

- $$\pi_{PK_{QAP(C_f)}}(2,3,4;2) = \left\{ \begin{array}{cccc} 12, & 1, & 1, & 2 \\ 9, & 1, & 1 & \\ & & & 3 \end{array} \right\}$$

- Middle values (details of computation) are invisible.

Verifier Phase

Proof verification

- Last Step: Proof Verification.
- Task: Given input set I_{in}, output set I_{out} and proof π, verify proof correctness.
- Verify that worker knows polynomial p, such that
- p is divisible by t
- p is build from alleged input and output values:

$$B(g^{r_v v_{I/O}(s)} g^{r_v v_{mid}(s)}, g^{r_w w_{I/O}(s)} g^{r_w w_{mid}(s)}) = B(g^{r_v r_w t(s)}, g^{h(s)}) B(g^{r_v r_w y_{I/O}(s)} g^{r_v r_w y_{mid}(s)}, g)$$

- $B(g^{r_v \alpha_v v_{mid}(s)}, g) = B(g^{r_v v_{mid}(s)}, g^{\alpha_v})$
- $B(g^{r_w \alpha_w w_{mid}(s)}, g) = B(g^{r_w w_{mid}(s)}, g^{\alpha_w})$
- $B(g^{r_v r_w \alpha_y y_{mid}(s)}, g) = B(g^{r_v r_w \alpha_y y_{mid}(s)}, g^{\alpha_y})$
- $B(g^{Z}, g^{\gamma}) = B(g^{r_v V_{mid}} g^{r_w W_{mid}} g^{r_v r_w Y_{mid}}, g^{\beta\gamma})$

Verifier Phase

Proof verification

- Task: Verify $p = t \cdot h$ for some h
- Succinct version $p(s) = t(s) \cdot h(s)$ is enough with high probability.
- However, we check the encrypted version $k^{p(s)} = k^{t(s) \cdot h(s)}$.
- Point is: Encrypted version is

$$B(g^{r_v v_{I/O}(s)} g^{r_v v_{mid}(s)}, g^{r_w w_{I/O}(s)} g^{r_w w_{mid}(s)}) = B(g^{r_v r_w t(s)}, g^{h(s)}) B(g^{r_v r_w y_{I/O}(s)} g^{r_v r_w y_{mid}(s)}, g)$$

Verifier phase

- To see that, define $k := B(g^{r_v}, g^{r_w})$
- Task: Proof $k^{p(s)} = k^{t(s) \cdot h(s)}$
- $B(g^{r_v}, g^{r_w})^{p(s)} = B(g^{r_v}, g^{r_w})^{t(s) \cdot h(s)}$
- $B(g, g)^{r_v r_w p(s)} = B(g, g)^{r_v r_w t(s) \cdot h(s)}$
- $B(g, g)^{r_v r_w (v_{I/O}(s) + v_{mid}(s)) \cdot (w_{I/O}(s) + w_{mid}(s)) - r_v r_w (y_{I/O}(s) + y_{mid}(s))} = B(g, g)^{r_v r_w t(s) \cdot h(s)}$
- $B(g, g)^{r_v (v_{I/O}(s) + v_{mid}(s)) \cdot r_w (w_{I/O}(s) + w_{mid}(s))} = B(g, g)^{r_v r_w t(s) \cdot h(s)} B(g, g)^{r_v r_w (y_{I/O}(s) + y_{mid}(s))}$
- $B(g^{r_v (v_{I/O}(s) + v_{mid}(s))}, g^{r_w (w_{I/O}(s) + w_{mid}(s))}) = B(g^{r_v r_w t(s)}, g^{h(s)}) B(g^{r_v r_w (y_{I/O}(s) + y_{mid}(s))}, g)$
- $B(g^{r_v v_{I/O}(s)} g^{r_v v_{mid}(s)}, g^{r_w w_{I/O}(s)} g^{r_w w_{mid}(s)}) = (g^{r_v r_w t(s)}, g^{h(s)}) B(g^{r_v r_w y_{I/O}(s)} g^{r_v r_w y_{mid}(s)}, g)$

Verifier phase

Example Proof Verification

- $B(g^{r_v v_{I/O}(s)} g^{r_v v_{mid}(s)}, g^{r_w w_{I/O}(s)} g^{r_w w_{mid}(s)}) = B(g^{r_v r_w t(s)}, g^{h(s)}) e(g^{r_v r_w y_{I/O}(s)} g^{r_v r_w y_{mid}(s)}, g)$
- $B(1 \cdot 12, 1 \cdot 12 \cdot 1) = B(1, 2) B(2 \cdot 1, 2)$
- $B(12, 12) = B(1, 2) B(2, 2)$
- $2^{log_2(12) \cdot log_2(12)} (mod_{23}) = 2^{log_2(1) \cdot log_2(2)} (mod_{23}) \cdot 2^{log_2(2) \cdot log_2(2)} (mod_{23})$
- $2^{10 \cdot 10} (mod_{23}) = 2^{0 \cdot 1} (mod_{23}) \cdot 2^{1 \cdot 1} (mod_{23})$
- $2^{10 \cdot 10} (mod_{23}) = 1 \cdot 2^{1 \cdot 1} (mod_{23})$
- $2 = 2$
- The other checks are analog and left to the reader as an exercise ;-)

Zero Knowledge Protocol Extension

Zero Knowledge and Randomization

- Suppose the worker does not want to publish (some of) the inputs.
- Setup: Extend verifier key in setup phase with $\{g^{r_v \alpha_v t(s)}, g^{r_w \alpha_w t(s)}, g^{r_y \alpha_y t(s)}, g^{r_v \beta t(s)}, g^{r_w \beta t(s)}, g^{r_y \beta t(s)}\}$
- Worker: Generate random elements R_v, R_w, R_y, use $v_R(x) = v(x) + R_v t(x)$, $w_R(x) = w(x) + R_w t(x)$ and $y_R(x) = y(x) + R_y t(x)$ instead.
- $p := (\sum_{k \in I} c_k v_k + R_v t(x)) \cdot (\sum_{k \in I} c_k w_k + R_w t(x)) - (\sum_{k \in I} c_k y_y + R_y t(x))$ has the same divisibility properties w.r.t. t.
- "Spread" the randomness across the I/O and middle parts of v_R, w_R and y_R, to get the required randomness in the proof and the zero knowledge on the I/O.

THAT's ALL FOR TODAY. [...]