A (somewhat) easy pen&paper example of the Pinocchio protocol (Part 1)

Mirco Richter (TOPOS Ledger)

October 12, 2018

Abstract

This paper arose from a workshop, the author gave at the Zero Knowledge Summit - ZK0x02 in Berlin, organized by ledgerZ. It is the first part of a small series and provides a relatively easy example of the Pinocchio protocol [HPR], that is understandable without support from a computer. It uses simple, but still non trivial cryptography to explain all the critical steps in the process of verified computing a la Pinocchio.

1 Preliminaries

In this paper, we apply the Pinocchio protocol [HPR] to a very basic example in "pen&paper" style. It therefore serves as a simple exercise and the goal is to calculate every single step by hand, using non trivial but still somewhat easy cryptography.

We assume that the reader knows how to add, subtract and multiply integers and how to do integer division with remainder. We use the mathematical terms *group* and *field* all the time, but it is not really necessary to fully understand these concepts. A reader without a mathematical background can think of them as types with certain operations (like in functional programming). A group has a single operation and a field has two of them.

What is really necessary though, is time, interest and a motivation for "chess like problems", where a few simple rules can lead to amazing results.

In addition we assume, that a reader is familiar with the basic underlying idea of verified computing. For a basic and simple introduction see Vitaliks post about zk-SNARKS[1], which is also largely based on the Pinocchio protocol (Note however that our approach is a bit closer to the original Pinocchio protocol as we don't look directly into R1CS).

[1] https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6

2 The problem

As the reader might have noticed, most papers about easy zk-SNARKS, Pinocchio and similar examples, actually "snarkify" not a computer program, but an arithmetic function instead. This is due to the nature of those protocols. Although it is possible to apply the process to arbitrary (finite and terminating) programs, things become complex fast. So simple arithmetic functions are indeed the most basic examples to outline the process.

In this paper we choose a simple arithmetic function, which just multiplies three values. To be more precise, we deal with the map

$$f(x_1, x_2, x_3) = (x_1 \cdot x_2) \cdot x_3$$

that takes three input values x_1, x_2 andx_3 and multiplies them together.

Remark 1. If you are not a type theorist, or a functional programmer, than this might look fine. Note however, that we haven't said anything about the types of x_1, x_2 and x_3, yet. Are we dealing with integers, floats, or something more exotic? And what does "multiplication" actually mean in our context? As we will see later, x_1, x_2 and x_3 have to be elements from a so called prime field and 'multiplication' is the product in that prime field. We will explain prime fields in a moment.

Having stated the actual problem, our roadmap for the rest of this exercise is as follows:

1. We develop our cryptographic scheme. This will also clarify what a prime field is and how our multiplication is actually defined.
2. Setup phase 1: We transform our function into a so called "arithmetic circuit".
3. Setup phase 2: We transform the arithmetic circuit into a "quadratic arithmetic program".
4. Key generation phase. We compute a proofer-, a verifier- and a cheating-key.
5. Execution and proofer phase. We choose values for x_1, x_2 as well as x_3 and compute the result of our function. Then we use the proofer key to generate a proof, that our computation is correct.
6. Verifier phase. We get the chosen input values x_1, x_2 and x_3, the output $f(x_1, x_2, x_3)$ and an alleged proof from the proofer and take the appropriate verifier key to check whether the proof is correct or not.

3 Our Cryptographic Scheme

Before we start this section, a little remainder for beginners in finite field arithmetics: If you are serious about understanding this, take your time! Every little

equation in this section might take you an hours or so to fully understand. This is normal for most people and mathematics shouldn't be read easily. Almost any word and symbol has meaning and it just takes time to get it.

In any case, this section is the most complicated one. However it is not important for everyone to understand anything really. What is important though, is the part about how to do 'addition' and 'multiplication' in the prime field $\mathbb{F}_{11}$ and 'multiplication' in the group $\mathbb{G}$. We will explain that in a moment.

That's being said, the protocol requires a special type of public key cryptography, which has a so called *non-trivial bilinear map*. In real world applications, such a scheme is build on the properties of certain elliptic curves[2] . However actual computations on these curves are best managed by computers and are to large and complex for our pen&paper approach. We therefore have to choose a much simpler cryptographic scheme to deal with. Our scheme will be explained later in detail.

In any case, the Pinocchio protocol requires a group $(\mathbb{G}, \cdot)$ with the following properties:

- $(\mathbb{G}, \cdot)$ has to be a commutative group with a finite number of elements, that has a generator (A finite, cyclic group).
- There must be another group $\mathbb{G}_T$ of the same order as $\mathbb{G}$ and a map $B(\cdot, \cdot) : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:

$$\begin{aligned} bilinieearity \quad & B(j \cdot g, k \cdot h) = B(g, h)^{j \cdot k} \quad && \forall j, k \in \mathbb{Z}, g, h \in \mathbb{G} \\ non - trivial \quad & && \exists g \in \mathbb{G} \qquad B(g, g) \neq id_{\mathbb{G}_T} \end{aligned} \tag{1}$$

- All polynomials, required by the protocol are defined over the field of discrete logarithms of a generator of $\mathbb{G}$.

Fortunately it is not important to understand anything we just said in order to follow this paper. I just added it for completeness. In our example we choose the following group with 11 elements:

$$\mathbb{G} := \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\} \tag{2}$$

Multiplication is defined by ordinary integer multiplication "modulo 23". That is if you have two elements, say x and y taken from $\mathbb{G}$, you multiply them as if they where ordinary numbers and then *integer divide* the result by 23 and keep the reminder. We can write this formally as:

$$x \bullet y := x \cdot y (mod_{23})$$

To distinguish the multiplication in $\mathbb{G}$ from ordinary integer multiplication, we picture it with the bullet symbol '$\bullet$'.

[2]According to the paper [LB-SNARK], this can be generalized to lattice based cryptographic schemes, which then might be resistant against an attack of future quantum computers.

Example 2. For a better understanding, lets look at an example: Suppose we choose 9 and 13. Both numbers are in our set $\mathbb{G}$, so this is a valid choice (Note: 7 would not be a valid choice as the number 7 is not an element of $\mathbb{G}$!). Then we compute their product as follows:

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

and as we see, the result is in $\mathbb{G}$ again. This means, that in our chosen group, the result of $9 \bullet 13$ is 2, since the ordinary product of 9 and 13 is 117 and the reminder of the integer division of 117 by 23 is 2 ($117 = 5 \cdot 23 + 2$). Its not that hard, but important for everything that follows. Lets do one more example:

$$\begin{aligned} 3 \bullet 4 = \\ 3 \cdot 4 (mod_{23}) = \\ 12 (mod_{23}) = \\ (0 \cdot 23 + 12)(mod_{23}) = \\ 12 \end{aligned}$$

It all boils down to integer division from school. However if you are more familiar with real number division, this might look a bit alien at first. Just keep in mind, that whenever something has a type of $\mathbb{G}$, multiplication is done in the way we just described.

Using this new multiplication, we can also define exponentiation in $\mathbb{G}$. This is done exactly like with ordinary numbers. For any given positive integer n we have

$$g^n = g \bullet g \bullet \ldots \bullet g$$

so g to the power of n just means multiply g n-times with itself using the multiplication we just described. This is everything we need to know about our group $\mathbb{G}$.

Remark 3. For the experts: Our group $\mathbb{G}$ is in fact the subgroup of $\mathbb{F}_{23}^*$ generated by the element 2, as we have

$$\begin{aligned} 2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 16, 2^5 = 9, 2^6 = 18, \\ 2^7 = 13, 2^8 = 3, 2^9 = 6, 2^{10} = 12, 2^{11} = 1 \end{aligned} \tag{3}$$

This immediately implies that $\mathbb{G}$ is indeed a group and the "field of discrete logarithms" for the generator 2 from $\mathbb{G}$ is the prime field $\mathbb{F}_{11}$.

In addition to our group $\mathbb{G}$, we also need the field of discrete logarithms of the generator 2, which is $\mathbb{F}_{11}$. So what is the field $\mathbb{F}_{11}$? Short answer: It is

the integers 'modulo 11'. However this is not very explanatory and to be more precise, the set $\mathbb{F}_{11}$ can be described as

$$\mathbb{F}_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

So it contains 11 Elements. In $\mathbb{G}$ we just had one way to combine two elements: The multiplication, which we pictured with the symbol "$\bullet$". In $\mathbb{F}_{11}$ we have two ways to combine elements and we call them 'addition' and 'multiplication', because they behave very much similar to ordinary multiplication and addition of numbers. The following two tables describe how these two operations are defined in $\mathbb{F}_{11}$.

+	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

$\bullet$	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

In order to add or multiply two elements you just have to look up the results in the intersection of the appropriate row and column. What's actually happening is, that you do ordinary integer operations, but "modulo 11" at the end.

Example 4. To make things clear lets compute a very basic example:

$$\begin{aligned} (5+8) \cdot 7 &= \\ 2 \cdot 7 &= \\ 3 & \end{aligned}$$

All we had to do is look in the addition table to see that $5+8$ equals 2 and then to see that $2 \cdot 7$ equals 3.

The addition table also helps to determine the negative of a number in $\mathbb{F}_{11}$. The negative of a number is precisely that number, which you have to add to the former to get zero in the end. For example the number -4 in $\mathbb{F}_{11}$ is 7, since $4 + 7 = 0$ as you can see in the lookup table. So, to find the negative of a number, just use the appropriate row in the lookup table, which represents the given element. Then search for the entry 0 and the number that represents the associated column is its negative.

Example 5. Lets put everything together into an example. The challenge is to solve the equation $(3 \cdot x + 4) \cdot 5 = 3x$ for x in $\mathbb{F}_{11}$. This means that all symbols in the equation are of type $\mathbb{F}_{11}$. We can do this exactly as we would with real number, but with the single exception, that we have to use the lookup tables for addition and multiplication:

$$\begin{aligned}
(3 \cdot x + 4) \cdot 5 &= 3x \\
3 \cdot 5 \cdot x + 4 \cdot 5 &= 3x \\
4 \cdot x + 9 &= 3x \\
4 \cdot x - 3x &= -9 \\
4 \cdot x + 8x &= 2 \\
(4 + 8) \cdot x &= 2 \\
1 \cdot x &= 2 \\
x &= 2
\end{aligned}$$

Now that we have some basic understanding of prime field arithmetics, a last question that might remains is: Why can't we just use ordinary numbers? The answer is, that in groups like $\mathbb{G}$, certain computations are much harder (even for computers), than compareable computations for ordinary numbers like integers or fractions. This is a key insight in cryptography. For example it is generally believed that finding a solution x to the equation

$$a^x = b$$

for a, b elements of a group like $\mathbb{G}$ is hard, if $\mathbb{G}$ is very large. (In our example lookup table (3) gives all possible results, but similar tables are not accessible for large groups[3]). This is known as *the discrete logarithm problem* (because formally we could write $x = log_a(b)$). As we will see in the verification step, it is only because we know table (3), that we can compute our example.

The last thing that is required by the Pinocchio protocol, is a non-trivial bilinear map, with the properties described in (1). The following function will work:

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

[3]Real world cryptographic groups similar to $\mathbb{G}$ have a lot of elements. Like in the order of the number of atoms in the observable universe (which is around 10^{80}).

(The product in the exponent is the multiplication in $\mathbb{Z}_{11}$). Of course you could argue, that we are cheating somewhat, since we have to solve the discrete logarithm problem, to compute this function, as we need to know what $log_2(g)$ means. However the point is to compute a simple example, which in turn requires a simple biliniear map like B.

To understand how B is computed, lets see how the discrete logarithm is defined. It is the inverse to exponentiation and $log_g(h)$ is a number x, such that $g^x = h$. In our example we can derived any discrete logarithm from lookup table (3). For example we have

$$log_2(18) = 6, \qquad log_2(9) = 5$$

since we can easily find $2^6 = 18$ and $2^5 = 9$ in (3). This way computing B is manageable. We only need it in the verification step, though.

Remark 6. For the experts: Note that our group is written with a multiplicative group law. Therefore the defining equation $B(j \cdot g, k \cdot h) = B(g, h)^{j \cdot k}$ for $j, k \in \mathbb{Z}$ has to be understood as $B(g^j, h^k) = B(g, h)^{j \cdot k}$ as $j \cdot g$ actually has no meaning for $j \in \mathbb{Z}$ and $g \in \mathbb{G}$ in our case. The bilinearity of function B is then computed as

$$\begin{aligned} B(g^j, h^k) = \\ 2^{log_2(g^j) \cdot log_2(h^k)}(mod_{23}) = \\ 2^{j \cdot log_2(g) \cdot k \cdot log_2(h)}(mod_{23}) = \\ (2^{log_2(g) \cdot log_2(h)})^{j \cdot k}(mod_{23}) = \\ ((2^{log_2(g) \cdot log_2(h)})(mod_{23}))^{j \cdot k} = \\ B(g, h)^{j \cdot k}. \end{aligned}$$

using basic properties of logarithm functions and exponential laws. It is also clear that B is non trivial as, for example $B(2, 2) = 2$ is not the unit in $\mathbb{G}$. Therefore B is indeed a non trivial biliniear map according to the Pinocchio protocol.

4 The Arithmetic Circuit

Now that we have established our basic cryptographic scheme, we are finally able to define the function we want to "snarkify" properly. Considering our group $\mathbb{G}$ as given in (2), we first have to chose a generator of that group[4]. We chose 2 and we can see from the lookup table (3), that 2 indeed generates every other element of $\mathbb{G}$. For example we have

$$9 = 2^5 = 2 \bullet 2 \bullet 2 \bullet 2 \bullet 2$$

[4] A generator is a element g, such that every other element in the group can be constructed using only g and the multiplication law.

Now the Pinocchio protocol requires, that the input values of all programs are elements from the field of discrete logarithms of the chosen generator. In our case this is the field $\mathbb{F}_{11}$. Therefore a type-safe definition of our function is

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

which just means, that the type is a function, which takes three elements from the field $\mathbb{F}_{11}$, multiplies them according to the multiplication rule in $\mathbb{F}_{11}$ and returns an element of $\mathbb{F}_{11}$ again. This is our example computer program and snarkification of this function a la Pinocchio is the actual problem of this paper.

In a next step, any given (finite and terminating) computer program is then transformed into a so called *arithmetic circuit*. Arithmetic circuits are nothing but directed acyclic graphs (DAGs) defined over a field $\mathbb{F}$, where vertices (usually called gates) either represent the addition, or the multiplication in that field. Arithmetic circuits are just a different way to describe a computer program, well suited for verified computation.

As the arithmetic circuit represents a computer program, there must be a way to *execute* the circuit. This is done as follows. Input values of the program are associated to the incoming edges of the DAG and are combined according to the rules of the vertices (If a vertex represents, say, an addition gate, all input values are added to get the output values of that vertex). The output values are then traveled along the outgoing edges of that vertex into the input of the ascendant vertices in the DAG. This is repeated until only outgoing edges are reached. The values at these outgoing edges represent the result of the computation.

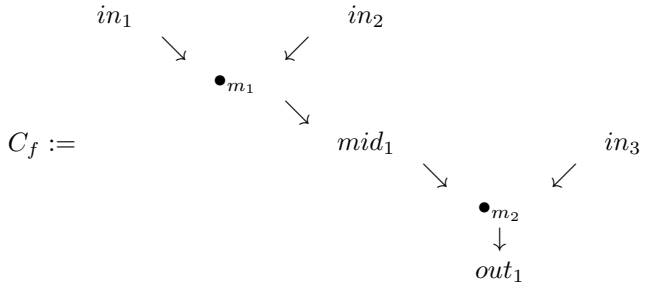
This way, every finite and terminating program can be transformed into an arithmetic circuit over a given field $\mathbb{F}$. Note however, that the transformation is not unique and can be done in different ways. This might lead to largely varying circuit sizes. Just think of it as similar to how a compiler transforms a higher programming language into low level assembly code. The size of the assembly code depends on the compiler, on the level of optimization and so on.

To define our circuit we strictly follow the Pinocchio protocol as follows:

- Chose a field $\mathbb{F}$.
- Vertices are decorated with either the addition or the multiplication in $\mathbb{F}$.
- Incoming edges are decorated with indices in_j, where j counts the position of that edge in some graph order.
- Outgoing edges are decorated with indices out_j, where j counts the position of that edge in some graph order.
- Internal edges are decorated with indices mid_j, if they are the outgoing edges of multiplication gates only. j counts the position of that edge in some graph order.
- The degree of the circuit is the number of vertices, with associated multiplication gate.

This is not very precise (we said nothing about the graph order, nor did we say how to actually derive a graph from a given computer program), but it is enough for our example.

Now considering our problem regarding the function f, we choose the field $\mathbb{F}_{11}$. Then a possible associated circuit looks like this:



So we have two vertices m_1 and m_2, decorated both with the multiplication in $\mathbb{F}_{11}$. In addition all edges are labled, too. Three input indices (in_1, in_2, in_3) on the incoming edges, one output index out_1 at the single outgoing edge and an additional lable for the edge after first multiplication gate mid_1[5]. As all labeled edges of the graph represent an index in the Pinocchio protocol, we get the following index set, which is important in the next step:

$$I := \{in_1, in_2, in_3, mid_1, out_1\} \tag{4}$$

The circuit represents f and we can *execute the circuit* to compute the function. Executing the circuit means associating actual values to *all* indices in I, not just inputs and outputs. For example, if we choose the three input values $in_1 = 2$, $in_2 = 3$ and $in_3 = 4$ from our prime field $\mathbb{F}_{11}$ we get

$$\begin{aligned} mid_1 &= in_1 \cdot in_2 = 2 \cdot 3 = 6 \\ out_1 &= mid_1 \cdot in_3 = 6 \cdot 4 = 2 \end{aligned}$$

Therefore executing our circuit C_f with input values $\{2, 3, 4\}$ gives $\{2, 3, 4, 6, 2\}$ as the result. Such a set is called a *valid assignment*, because all values appear from an actual execution of every gate in the circuit. In contrast, the set $\{2, 3, 4, 7, 8\}$ is not a valid assignment, since the values $mid_1 = 7$ and $out_1 = 8$ can not be derived from our circuit, given the input values $in_1 = 2$, $in_2 = 3$ and $in_3 = 4$.

The point here is, that the only way to get a valid assignment, is to apply all gates in the circuit to the given input values. This is a first key ingredient to verify that we have done the computation, as intended.

[5]Of course in our case the circuit DAG is just a binary tree. Trees are a particular subclass of DAGs.

5 The Quadratic arithmetic program

The next step is to use the circuit and the index set I to derive a so called *quadratic arithmetic program* (QAP). QAPs are nothing but sets of certain polynomials, that encode the circuit in a different way.

To build a quadratic arithmetic program from a circuit, we first have to choose random (invertible?) elements from the underlying field $\mathbb{F}$ of the circuit. One element for each multiplication gate in the circuit.

Since our circuit is defined over the field $\mathbb{F}_{11}$ and since we have two multiplication gates (labeled m_1 and m_2), we choose

$$m_1 = 5, m_2 = 7$$

There is nothing special about these numbers. They are just what we choose for this example. Any other choice (of invertible elements?) from $\mathbb{F}_{11}$ would also be possible.

To proceed, remember that a polynomial $p \in \mathbb{F}[x]$ of degree n, with undetermined x is nothing but an expression of the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

where all the variables $a_1, \ldots a_n$ are from the field $\mathbb{F}$.

This is used in the next step to construct the so called *target polynomial* $t \in \mathbb{F}_{11}[x]$, which is part of the QAP data. According to the Pinocchio protocol, the target polynomial of our QAP is computed as

$$t(x) = (x - m_1)(x - m_2) = (x - 5)(x - 7) = (x + 6)(x + 4) = x^2 + 10x + 2$$

Remember that the computation is done in $\mathbb{F}_{11}$! The purpose of t will be explained later. In addition we have to derive three sets $\{v_k\}_{k\in I}$,$\{w_k\}_{k\in I}$ and $\{y_k\}_{k\in I}$ of polynomials in $\mathbb{F}_{11}[x]$, where I is our index set (4).

These polynomials are determined by their values on the numbers m_1 and m_2, which we associated to the multiplication gates in our circuit. According to the Pinocchio protocol, the rules are as follows:

- A polynomial from $\{v_k\}_{k\in I}$ is equal to 1 at m_j (i.e $v_k(m_j) = 1$), if the edge indexed by k is a left input to the associated multiplication gate $\bullet_{m_j}$ in the circuit and it is zero at m_j, otherwise.
- A polynomial from $\{w_k\}_{k\in I}$ is equal to 1 at m_j (i.e $w_k(m_j) = 1$), if the edge indexed by k is a right input to the associated multiplication gate $\bullet_{m_j}$ in the circuit and it is zero at point m_j, otherwise.
- A polynomial from $\{y_k\}_{k\in I}$ is equal to 1 at m_j (i.e $y_k(m_j) = 1$), if the edge indexed by k is an output to the associated multiplication gate $\bullet_{m_j}$ in the circuit and it is zero at point m_j, otherwise.

Applied to our QAP of circuit C_f, we have to find the following set of polynomials

$$\left\{ \begin{array}{c} \{v_{in_1}, v_{in_2}, v_{in_3}, v_{mid_1}, v_{out}\}, \\ \{w_{in_1}, w_{in_2}, w_{in_3}, w_{mid_1}, w_{out}\}, \\ \{y_{in_1}, y_{in_2}, y_{in_3}, y_{mid_1}, y_{out}\} \end{array} \right\} \tag{5}$$

There are two multiplication gates in our circuit, with associated values $m_1 = 5$ and $m_2 = 7$. Therefore the polynomials are determined by their values at 5 and 7. It is a mathematical fact, that a polynomial of degree n is completely characterized by its values on $n + 1$ points. Since we have two points, each of our polynomials is of degree 1, i.e has the form $p(x) = a_1 x + a_0$.

To see how this works consider the polynomial v_{in_1}: Since the edge in_1 is a left input to the multiplication gate labeled with $m_1 = 5$, but not a left input to the multiplication gate labeled with $m_2 = 7$, the associated polynomial v_{in_1} has the values $v_{in_1}(5) = 1$ and $v_{in_1}(7) = 0$.

A similar reasoning can be applied to all other polynomials from (5) and we get the following values (Exercise):

$$\begin{array}{rr} v_{in_1}(5) = 1, & v_{in_1}(7) = 0 \\ v_{in_2}(5) = 0, & v_{in_2}(7) = 0 \\ v_{in_3}(5) = 0, & v_{in_3}(7) = 0 \\ v_{mid_1}(5) = 0, & v_{mid_1}(7) = 1 \\ v_{out}(5) = 0, & v_{out}(7) = 0 \end{array}$$

$$\begin{array}{rr} w_{in_1}(5) = 0, & w_{in_1}(7) = 0 \\ w_{in_2}(5) = 1, & w_{in_2}(7) = 0 \\ w_{in_3}(5) = 0, & w_{in_3}(7) = 1 \\ w_{mid_1}(5) = 0, & w_{mid_1}(7) = 0 \\ w_{out}(5) = 0, & w_{out}(7) = 0 \end{array}$$

$$\begin{array}{rr} y_{in_1}(5) = 0, & y_{in_1}(7) = 0 \\ y_{in_2}(5) = 0, & y_{in_2}(7) = 0 \\ y_{in_3}(5) = 0, & y_{in_3}(7) = 0 \\ y_{mid_1}(5) = 1, & y_{mid_1}(7) = 0 \\ y_{out}(5) = 0, & y_{out}(7) = 1 \end{array}$$

Note that these tables are not our polynomials, yet! But they are everything we need to actually get them. Remember, that its a mathematical fact, that a set of $n+1$ point in $\mathbb{F} \times \mathbb{F}$, determine a polynomial of degree n uniquely. Computing the actual polynomial from the set of points can be done by various methods

(One such method is Lagrange interpolation). Unfortunately these methods are computationally expensive and in real world applications, they contribute to the observed overhead of the QAP generation phase.

Fortunately we don't need Lagrange interpolation in our example. Each of our polynomials is determined by two points (For example v_{in} is determined by the two points $\{(5,1),(7,0)\}$). Hence they have degree 1, are therefore just affine functions and we can use the ordinary defining equation of such affine functions

$$p(x) = mx + b \tag{6}$$

Remark 7. In our example, all polynomials are defined over the same two values 5 and 7, so we can do a little pre-computation, to get an equation that works for all of them (Following this is not necessary for the rest of the paper). Putting the numbers 5 and 7 into equation (6) we get

$$\begin{aligned} 5m + b &= p(5) & \qquad 7m + b &= p(7) \\ b &= p(5) - 5m & \qquad b &= p(7) - 7m \\ b &= p(5) + 6m & \qquad b &= p(7) + 4m \end{aligned}$$

$$\begin{aligned} p(5) + 6m &= p(7) + 4m \\ 6m - 4m &= p(7) - p(5) \\ 2m &= p(7) + 10p(5) \\ m &= 6(p(7) + 10p(5)) \end{aligned}$$

$$\begin{aligned} b &= p(5) + 6m \\ b &= p(5) + 6 \cdot 6(p(7) + 10p(5)) \\ b &= p(5) + 3(p(7) + 10p(5)) \\ b &= p(5) + 3p(7) + 3 \cdot 10p(5) \\ b &= p(5) + 3p(7) + 8p(5) \\ b &= 9p(5) + 3p(7) \end{aligned}$$

Now substituting b and m in equation (6) by our computed values, we get the following expression for all our polynomials:

$$p(x) = 6(p(7) + 10p(5))x + 9p(5) + 3p(7) \tag{7}$$

In the last step we substitute the values of $p(5) \leftarrow v_k(5)$ and $p(7) \leftarrow v_k(7)$ (and similar for w_k and y_k) into equation (7) to get the actual polynomials. For

v_{in} the computation looks like this:

$$\begin{aligned} v_{in_1}(x) &= 6(v_{in_1}(7) + 10v_{in_1}(5))x + 9v_{in_1}(5) + 3v_{in_1}(7) \\ &= 6(0 + 10 \cdot 1)x + 9 \cdot 1 + 3 \cdot 0 \\ &= 6 \cdot 10x + 9 \\ &= 5x + 9 \end{aligned}$$

(Again, keep in mind that the arithmetics are done in $\mathbb{F}_{11}$). Apply the same computation to all the other polynomials from (5) and you get the following expressions:

$$\begin{array}{rrr} v_{in_1}(x) = 5x + 9, & w_{in_1}(x) = 0, & y_{in}(x) = 0 \\ v_{in_2}(x) = 0, & w_{in_2}(x) = 5x + 9, & y_{in}(x) = 0 \\ v_{in_3}(x) = 0, & w_{in_3}(x) = 6x + 3, & y_{in_3}(x) = 0 \\ v_{mid_1}(x) = 6x + 3, & w_{mid_1}(x) = 0, & y_{mid_1}(x) = 5x + 9 \\ v_{out}(x) = 0, & w_{out}(x) = 0, & y_{out}(x) = 6x + 3 \end{array}$$

This is everything we need to write down our quadratic arithmetic program. According to the Pinocchio protocol the QAP of our circuit C_f looks like this:

$$QAP_{\mathbb{F}_{11}}(C_f) = \{x^2 + 10x + 2, \left\{ \begin{array}{c} \{5x + 9, 0, 0, 6x + 3, 0\}, \\ \{0, 5x + 9, 6x + 3, 0, 0\}, \\ \{0, 0, 0, 5x + 9, 6x + 3\} \end{array} \right\}\} \tag{8}$$

6 Circuit Satisfiability and Polynomial Devision

According to the specifications of the Pinocchio protocol, every association of values $\{c_k\}_{k \in I}$ to the labeled edges of a circuit defines a polynomial, that can be build from the associated QAP in the following way

$$p := (v_0 + \sum_{k \in I} c_k v_k)(w_0 + \sum_{k \in I} c_k w_k) - (y_0 + \sum_{k \in I} c_k y_y) \tag{9}$$

(Note that the polynomials v_0, w_0 and y_0 are actually not needed in our example).

The point is, that the polynomials p and t are designed in such a way, that the set $\{c_k\}_{k \in I}$ is a valid association of values to the circuit (i.e. computed by the circuit), if and only if the polynomial p is divisible by the polynomial t.

This means, that we currently have two ways to check if someone actually computed the circuit:

1. Take the same input values and compute the circuit yourself. If both sets $\{c_k\}_{k \in I}$ are identical, (and you didn't made a mistake) then you know that the other person computed the circuit correctly.

2. Take the set $\{c_k\}_{k \in I}$ and the QAP, compute the polynomial p and divide it by t. If there is no remainder, you know that the other party must have computed the circuit properly, without computing it yourself.

To understand this a bit better, lets apply it to our QAP. Every association of values to the index set $\{c_{in_1}, c_{in_2}, c_{in_3}, c_{mid_1}, c_{out_1}\}$ of our circuit C_f defines a polynomial $p \in \mathbb{F}_{11}[x]$ as follows

$$\begin{aligned} p(x) =& (c_{in_1}(5x+9) + c_{mid_1}(6x+3)) \cdot (c_{in_2}(5x+9) + c_{in_3}(6x+3)) \\ & - (c_{mid_1}(5x+9) + c_{out_1}(6x+3)) \end{aligned}$$

Just use the defining equation (9) of p and the polynomials, given in our QAP (8).

From the computation in section (4), we know that the set $\{2, 3, 4, 6, 2\}$ is a valid association to circuit C_f and if we substitute these values into the definition of p we get

$$\begin{aligned} p(x) = (2(5x+9) + 6(6x+3)) \cdot (3(5x+9) + 4(6x+3)) & \\ -(6(5x+9) + 2(6x+3)) = & \\ (2 \cdot 5x + 2 \cdot 9 + 6 \cdot 6x + 6 \cdot 3) \cdot (3 \cdot 5x + 3 \cdot 9 + 4 \cdot 6x + 4 \cdot 3) & \\ -(6 \cdot 5x + 6 \cdot 9 + 2 \cdot 6x + 2 \cdot 3) = & \\ (10x + 7 + 3x + 7) \cdot (4x + 5 + 2x + 1) - (8x + 10 + 1x + 6) = & \\ (2x+3) \cdot (6x+6) - 9x - 5) = & \\ x^2 + x + 7x + 7 + 2x + 6 = & \\ x^2 + 10x + 2 & \end{aligned}$$

We can easily see that in this case p is indeed divisible by t, since both polynomials are actually the same! [6]

Lets go one step further and also look at a counterexample. Suppose we where given the same input value $\{2, 3, 4\}$ as in the previous example, but we are lazy and just make the other values up. Like in the set

$$\{2, 3, 4, 5, 9\}$$

which is not a valid association to our circuit. Putting these values into the

[6] This is just an artifact of our example. In general p and t are not necessarily the same.

definition of the polynomial p, we get:

$$\begin{aligned}
p(x) = (2(5x+9)+5(6x+3)) \cdot (3(5x+9)+4(6x+3)) \\
-(5(5x+9)+9(6x+3)) = \\
(2 \cdot 5x + 2 \cdot 9 + 5 \cdot 6x + 5 \cdot 3) \cdot (3 \cdot 5x + 3 \cdot 9 + 4 \cdot 6x + 4 \cdot 3) \\
-(5 \cdot 5x + 5 \cdot 9 + 9 \cdot 6x + 9 \cdot 3) = \\
(10x + 7 + 8x + 4) \cdot (4x + 5 + 2x + 1) - (3x + 1 + 10x + 5) = \\
(7x) \cdot (6x + 6) - (2x + 6) = \\
(9x^2 + 9x) + 9x + 5 = \\
9x^2 + 7x + 5
\end{aligned}$$

and we can check by ordinary polynomial division, that p is indeed not divisible by t. To see that compute:

$$\begin{aligned}
&(9x^2 + 7x + 5) : (x^2 + 10x + 2) \qquad = 9 + \frac{5x+4}{x^2+10x+2} \\
&-(9x^2 + 2x + 1) \\
&\quad 5x + 4
\end{aligned}$$

So we have the remainder $\frac{5x+4}{x^2+10x+2}$, which is not a polynomial (its of rational function type). This tells us, that the set $\{2, 3, 4, 5, 9\}$ is not a valid asigment, as expected.

Ok, now we are done, right? We have found a way to proof the proper execution of a given circuit.

Unfortunately not. Given a QAP, it is easy to come up with polynomials, that are divisible by t. Indeed given any polynomial $q \in \mathbb{F}[x]$, the product $t \cdot q$ is divisible by t. Therefore it is not enough for a proofer to just send the verifier some polynomial which is divisible by t. Its true, that this represents a valid execution of the circuit, but with respect to what input values? The approach of the Pinocchio protocol is to split p into an input/output related part, which is known by the verifier and a middle part, which can only be known, by someone, who actually executed the circuit. We will see how this works after the following key setup phase.

7 The key setup phase

Suppose we have a circuit C and a corresponding $QAP(C)$. In the key setup phase a trusted third party produces three keys:

- A proofer key, which is necessary to compute a proof for the correct execution of a circuit.
- A verifier key, which is necessary to verify a given proof.
- A cheating key, which can be used to generate false proofs.

We assume that the groups $\mathbb{G}$, $\mathbb{G}_T$ and the bilinear map $B(\cdot,\cdot)$ (See 1) have been determined it advance and that they don't change. They are part of the protocol. The key generator then chooses the following things:

- A generator $g \in \mathbb{G}$, such that its field of discrete logarithms $\mathbb{F}$ is equal to the field over which the given circuit and its associated QAP is defined.
- A set of random elements $s, r_v, r_w, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \in \mathbb{F}$ taken from the field of discrete logarithms of g.

In our example, we assume, that our cryptographic scheme is based on $\mathbb{G}$, $\mathbb{G}_T$ and $B(\cdot,\cdot)$ as explained in section (3). Moreover our trusted third party choses the generator $g = 2$. In addition they chose the eight random values

$$r_v = 9, r_w = 8, s = 7, \alpha_v = 6, \alpha_w = 5, \alpha_y = 4, \beta = 3, \gamma = 2 \tag{10}$$

from $\mathbb{F}_{11}$. There is nothing special about these values. They just represent our particular choice for this example[7]. In addition the key generator has to compute some derived values in $\mathbb{F}_{11}$. In particular we need

$$\begin{aligned} r_y = r_v \cdot r_w = 9 \cdot 8 = 6 \\ \gamma \cdot \beta = 3 \cdot 2 = 6 \end{aligned}$$

In the next step, all these values must be 'written into the exponent' of our generator 2. In contrast to the previous values this computation has to be done in group $\mathbb{G}$. Using lookup table (3), we get:

$$\begin{aligned} g_v = g^{r_v} = 2^9 = 6 \\ g_w = g^{r_w} = 2^8 = 3 \\ g_y = g^{r_y} = 2^6 = 18 \\ g^{\alpha_v} = 2^6 = 18 \\ g^{\alpha_w} = 2^5 = 9 \\ g^{\alpha_y} = 2^4 = 16 \\ g^{\gamma} = 2^2 = 4 \\ g^{\beta\gamma} = 2^6 = 18 \end{aligned} \tag{11}$$

With these values, the key generator is able to compute a proofer and associated verifier key, as explained in the next sections.

[7]Some of these values can be used to define a "cheating key", also called toxic waste. However I don't know which subset is sufficient. But it is an important question and any suggestion is welcome!

7.1 The proofer key

The proofer key, is nothing but a set of "numbers" from our chosen group $\mathbb{G}$. It is computed from the polynomials in the QAP, which represent the indices, that are not related to inputs or outputs in the circuit. In our case, these are all the polynomial from our QAP, indexed by the set $I_{mid} = \{mid_1\}$. To be more precise and according to the Pinocchio protocol a proofer key, associated to the chosen values from (10) is given by

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

where d is the degree of the target polynomial t. So, what we actually do is, we evaluate the polynomials, not related to I/O values in the circuit at a certain point and write that number into the exponent of our chosen generator.

In our example we have $d = 2$. Substituting the values from (10) and (11) into the definition of the proofer key, we get

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

Then we look at the required polynomials from our QAP and write their values at 7 into the exponent of all the entries in the proofer key. This gives:

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

Now it is "just" a matter of computation. We have to keep in mind that all computation in the exponent are done in $\mathbb{Z}_{11}$, while the actual powering is done in our group $\mathbb{G}$. We get

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

$$\left\{ \begin{array}{lll} \{6^1\}, & \{3^0\}, & \{18^0\}, \\ \{6^6\}, & \{3^0\}, & \{18^0\} \\ \{2^7, 2^5\}, & & \{6^3 \cdot 3^0 \cdot 18^0\} \end{array} \right\}$$

$$\left\{ \begin{array}{lll} \{6\}, & \{1\}, & \{1\}, \\ \{6^6\}, & \{1\}, & \{1\} \\ \{2^7, 2^5\}, & & \{6^3\} \end{array} \right\}$$

That's it! We finally got our proofer key, which is nothing but a set of nine elements from the group $\mathbb{G}$:

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

Note that the proofer key is not unique. It depends on the chosen values (10) and the generator. This is important to know, since it give the possibility to compute different proofer keys for the same circuit.

7.2 The verifier key

The verifier key is associated to a given proofer key and it is nothing but a set of "numbers" from our group $\mathbb{G}$. Those numbers are computed from the polynomials in the QAP, which represent the I/O part. In our case, these are the polynomials indexed by the set $I_{I/O} = \{in_1, in_2, in_3, out_1\}$ [8]. According to the Pinocchio protocol the verifier key is defined as

$$\left\{ \begin{array}{lll} g^1, & g^{\alpha_v}, & g^{\alpha_w}, \\ g^{\gamma}, & g^{\beta\gamma}, & g_y^{t(s)}, \\ & & \end{array} \begin{array}{l} g^{\alpha_\gamma}, \\ \\ \{g_v^{v_k(s)}, g_w^{w_k(s)}, g_y^{y_k(s)}\}_{k \in \{0\} \cup I_{I/O}} \end{array} \right\}$$

and using our index set $I_{I/O} = \{in_1, in_2, in_3, out_1\}$, we can unroll this definition, which gives the set

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

To compute it, we have to look at our list of randomly chosen values (10) as well as (11) and apply the multiplication law in $\mathbb{G}$ again. We get

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

[8] Since we have more I/O related values than inner values, our verifier key is actually larger than the proofer key. This is due to our simple example and usually different in real world applications

Then we look at the required polynomials from our QAP and write their values at 7 into the exponent of all the entries in the verifier key. The result is:

$$\left\{\begin{array}{llll} 2, & 18, & 9, & 16, \\ 4, & 18, & 18^{7^2+10\cdot 7+2}, & \\ 6^0, & 3^0, & 18^0, & \\ 6^{5\cdot 7+9}, & 3^0, & 18^0, & \\ 6^0, & 3^{5\cdot 7+9}, & 18^0, & \\ 6^0, & 3^{6\cdot 7+3}, & 18^0, & \\ 6^0, & 3^0, & 18^{6\cdot 7+3} & \end{array}\right\}$$

Again its now "just" a matter of computation. We have to keep in mind that all computation in the exponent are done in $\mathbb{Z}_{11}$, while the actual powering is done in $\mathbb{G}$. We get

$$\left\{\begin{array}{llll} 2, & 18, & 9, & 16, \\ 4, & 18, & 18^{5+4+2}, & \\ 1, & 1, & 1, & \\ 6^{2+9}, & 1, & 1, & \\ 1, & 3^{2+9}, & 1, & \\ 1, & 3^{9+3}, & 1, & \\ 1, & 1, & 18^{9+3} & \end{array}\right\}$$

That's it! We finally have a verifier key, associated to our proofer key $PK_{QAP(C_f)}$. It is nothing but a set of 22 elements from our group $\mathbb{G}$:

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

Note that this verifier key is associated to a given proofer key. It only works for this particular key.

Depending on the application, the circuit, the QAP, the proofer key and the verifier key are now public data. All other values must to be deleted in the final cleanup phase of the trusted key generator party. In other words, we have to trust the key generator to forget all values from (10).

8 The Execution and Proofer phase

Suppose a program f with associated circuit C_f and $QAP(C_f)$ is given and a trusted third party has computed a proofer key $PK_{QAP(C_f)}$, together with an associated verifier key $VK(PK_{QAP(C_f)})$. Another party, usually called the *proofer*, has access to this data and their task is, to compute the result of function f, given input set In, together with a proof, that the result was obtained by the proper execution of circuit C_f.

8.1 Circuit execution

In a first step they execute the circuit C_f with respect to the given input values In. To do so, they apply the values from In to the input edges of the DAG C_f and compute the circuit all the way down, gate by gate, to generate a valid association to the circuit (See section 4). The values Out, associated to the outgoing edges of the circuit are the results of the computation.

Applied to our problem, we use the input values $c_{in_1} = 2$, $c_{in_2} = 3$ and $c_{in_3} = 4$ again. So we have $In = \{2, 3, 4\}$ and as we know from section 4, executing our circuit with respect to this set of input values, gives the valid set of associated values

$$\{c_{in_1}, c_{in_2}, c_{in_3}, c_{mid_1}, c_{out_1}\} = \{2, 3, 4, 6, 2\}$$

Therefor in our case we have the output value 2 and a single intermediate value 6.

8.2 Proof generation

In a second step, the proofer has to generate a proof for the alleged execution of C_f with respect to the input values In. If we assume that the proofer indeed executed the circuit properly, they have learned a valid association of values to the circuit and they can use it to construct a polynomial p as defined in (9). Since their values are valid, we know that the polynomial p is divisible by t. Hence the proofer can compute the polynomial

$$h = p/t$$

Using h and the proofer key, they are able to generate a proof for the proper execution of the circuit C_f.

In our example the polynomial $p \in \mathbb{F}_{11}[x]$, is given by $p(x) = x^2 + 10x + 2$ (See Section 6) and since the target polynomial t of our QAP is also $x^2+10x+2$, we get the constant (somewhat trivial) polynomial $h(x) = 1$ [9]. With I_{mid}, h and the proofer key at hand, we are able to generate a proof for the proper execution of our circuit. According to the Pinocchio protocol such a proof consist of the following data:

$$\left\{ \begin{array}{ccccc} g_v^{v_m(s)}, & g_w^{w_m(s)}, & g_y^{y_m(s)}, & & g^{h(s)} \\ g_v^{\alpha_v v_m(s)}, & g_w^{\alpha_w w_m(s)}, & g_y^{\alpha_y y_m(s)} & & \\ & & & g_v^{\beta v_m(s)} \cdot g_w^{\beta w_m(s)} \cdot g_y^{\beta y_m(s)} & \end{array} \right\}$$

where the exponents are that part of p, which is neither related to input nor output values:

[9] Note that h is usually not just the constant function 1, but a more complicated polynomial. It's just an artifact of our example, that makes everything a bit more easy.

$$v_m(x) = \sum_{k \in I_{mid}} c_k v_k(x)$$
$$w_m(x) = \sum_{k \in I_{mid}} c_k w_k(x)$$
$$y_m(x) = \sum_{k \in I_{mid}} c_k y_k(x)$$

Remember that all the v_k's, w_k's and y_k's are part of the QAP and hence are publicly available. However the proofer does not know any of the values g_v, g_w, g_y, α_v, α_w,α_y, s, or β, because they were (hopefully) deleted at the end of the key generation phase.

How can they compute the proof then? The answer is based on the two exponential laws

$$g^x \cdot g^y = g^{x+y}, (g^x)^y = g^{x \cdot y} \tag{12}$$

which hold for all group elements $g \in \mathbb{G}$, as well as exponents $x, y \in \mathbb{Z}$ and these laws basically give a recipe for the computation of any polynomial in the exponent.

For example, they can be used to compute $g^{h(s)}$, without actually knowing s. To see that, observe that $h(s)$ can be written as

$$h(s) = h_n s^n + h_{n-1} s^{n-1} + h_1 s + h_0$$

and we might use the previously mentioned exponential laws to rewrite the expression $g^{h(s)}$ according to

$$\begin{aligned} g^{h(s)} = \\ g^{h_n s^n + h_{n-1} s^{n-1} + h_1 s + h_0} = \\ g^{h_n s^n} \cdot g^{h_{n-1} s^{n-1}} \cdot \dots \cdot g^{h_1 s} \cdot g^{h_0} = \\ (g^{s^n})^{h_n} \cdot (g^{s^{n-1}})^{h_{n-1}} \cdot \dots \cdot (g^s)^{h_1} \cdot (g)^{h_0} \end{aligned}$$

This can be exploited by the proofer, as they know all the numbers $h_0, \dots, h_n$ from the polynomial division p/t and the elements $g, g^s, g^{s^2} \dots, g^{s^n}$ are part of the proofer key. So they are indeed able to compute $g^{h(s)}$ without actually knowing s.[10]

In a similar way, the proofer is able compute all other values required by the proof, using the data from the proofer key. For example:

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

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

[10]Note that in our example $2^{h(s)}$ is very simple since $h(s) = 1$ for any $s \in \mathbb{F}_{11}$, but again this is an artifact of our example.

(Here the Π symbol means to multiply all elements indexed by I_{mid} together). The c_k's are known from the execution of the circuit and the $g_v^{v_k(s)}$ as well as $g_v^{\alpha_v v_k(s)}$ are part of the proofer key. The same reasoning applies to all other elements in the proof.

Now lets apply this to our actual problem. Based on the previously mentioned reasoning, we can use the data from the proofer key to get $g_v^{v_m(s)} = (g_v^{v_{mid_1}(s)})^{c_{mid_1}} = 6^6$ and a similar computation can be applied to all other values in the proof (which is a good exercise). The result is

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

To simplify this further, keep in mind that exponentiation is done in group $\mathbb{G}$. After further simplification, we finally get the following proof for the proper execution of C_f with input values 2, 3 and 4, outcome 2 and proofer key $PK_{QAP(C_f)}$:

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

Note that a different proofer key, would imply a different proof.

9 The verifier phase

In the final step, we suppose that a program f with circuit C_f as well as associated quadratic arithmetic program $QAP(C_f)$ is given and a trusted third party has computed a proofer key $PK_{QAP(C_f)}$, together with an appropriate verifier key $VK(PK_{QAP(C_f)})$. In addition a proofer claims, that they have executed the circuit properly with respect to a given set of input values In. Their result is Out and they also computed a proof $\pi_{PK_{QAP(C_f)}}(In; Out)$ to show that their claims are indeed correct.

The task of the verifier is then to check the claims of the proofer. The verifier therefore gets the Input In, the output Out as well as the proof $\pi_{PK_{QAP(C_f)}}(In; Out)$ from the proofer. In addition they have access to the verifier key $VK(PK_{QAP(C_f)})$ of the used proofer key and to the QAP of the circuit.

Loosely speaking, the verifier key enables the verifier to evaluate the Input/Output related parts of the polynomial p 'in the exponent' at the secret point s, without knowing any of the values from (10), using the exponential

laws (12):

$$g_v^{v_{I/O}(s)} = g_v^{\sum_{k\in I_{I/O}} c_k v_k(s)} = \Pi_{k\in I_{I/O}} (g_v^{v_k(s)})^{c_k}$$
$$g_w^{w_{I/O}(s)} = g_v^{\sum_{k\in I_{I/O}} c_k w_k(s)} = \Pi_{k\in I_{I/O}} (g_w^{w_w(s)})^{c_k}$$
$$g_y^{y_{I/O}(s)} = g_v^{\sum_{k\in I_{I/O}} c_k y_k(s)} = \Pi_{k\in I_{I/O}} (g_y^{y_k(s)})^{c_k}$$

All required $g_v^{v_k(s)}$'s, $g_w^{w_k(s)}$'s and $g_y^{yk(s)}$'s are elements of the verification key and the I/O related numbers c_k's are provided by the proofer. Hence the verifier is able to compute these expressions.

If we apply this to our actual problem, using the previously computed verifier key $VK(PK_{QAP(C_f)})$, we get the following results:

$$\begin{aligned} g_v^{v_{I/O}(s)} = (g_v^{v_{in_1}(s)})^{c_{in_1}} \bullet (g_v^{v_{in_2}(s)})^{c_{in_2}} \bullet (g_v^{v_{in_3}(s)})^{c_{in_3}} \bullet (g_v^{v_{out}(s)})^{c_{out}} = \\ (g_v^{v_{in_1}(s)})^2 \bullet (g_v^{v_{in_2}(s)})^3 \bullet (g_v^{v_{in_3}(s)})^4 \bullet (g_v^{v_{out}(s)})^2 = \\ 1^2 \bullet 1^3 \bullet 1^4 \bullet 1^2 = \\ 1 \end{aligned}$$

$$\begin{aligned} g_w^{w_{I/O}(s)} = (g_w^{w_{in_1}(s)})^{c_{in_1}} \bullet (g_w^{w_{in_2}(s)})^{c_{in_2}} \bullet (g_w^{w_{in_3}(s)})^{c_{in_3}} \bullet (g_w^{w_{out}(s)})^{c_{out}} = \\ (g_w^{w_{in_1}(s)})^2 \bullet (g_w^{w_{in_2}(s)})^3 \bullet (g_w^{w_{in_3}(s)})^4 \bullet (g_w^{w_{out}(s)})^2 = \\ 1^2 \bullet 1^3 \bullet 3^4 \bullet 1^2 = \\ 3^4 = \\ 12 \end{aligned}$$

$$\begin{aligned} g_y^{y_{I/O}(s)} = (g_y^{y_{in_1}(s)})^{c_{in_1}} \bullet (g_y^{y_{in_2}(s)})^{c_{in_2}} \bullet (g_y^{y_{in_3}(s)})^{c_{in_3}} \bullet (g_y^{y_{out}(s)})^{c_{out}} = \\ (g_y^{y_{in_1}(s)})^2 \bullet (g_y^{y_{in_2}(s)})^3 \bullet (g_y^{y_{in_3}(s)})^4 \bullet (g_y^{y_{out}(s)})^2 = \\ 1^2 \bullet 1^3 \bullet 1^4 \bullet 18^2 = \\ 18^2 = \\ 2 \end{aligned}$$

This looks complicated, but actually every single step is easily deduced from the previous steps of the paper. To finally verify any alleged proof the verifier has to check whether or not the polynomial p is divisible by t. However the verifier does not know p. Nevertheless they can use the non trivial bilinear map (1) to check the equation

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

without actually knowing either s, p, nor h. This (somewhat) implies the divisibility of p by t.[11]. The computation is done in the exponent of some generator element $k \in \mathbb{G}_T$, so the actual required task is to check

$$k^{p(s)} = k^{h(s) \cdot t(s)} \tag{13}$$

which then also implies $p(s) = h(s) \cdot t(s)$. This can be done using the bilinear map $B(\cdot, \cdot)$ from (1). According to the Pinocchio protocol the verifier has to check the following identity

$$\begin{aligned} B(g_v^{v_0(s)} g_v^{v_{I/O}(s)} g_v^{v_{mid}(s)}, g_w^{w_0(s)} g_w^{w_{I/O}(s)} g_w^{w_{mid}(s)}) = \\ B(g_y^{t(s)}, g^{h(s)}) \bullet B(g_y^{y_0(s)} g_y^{y_{I/O}(s)} g_y^{y_{mid}(s)}, g) \end{aligned} \tag{14}$$

This might look rather complicated, but you can check yourself, that all required values are actually known to the verifier. Just search for each part of the equation, symbol by symbol.

Remark 8. The key inside here is that equation (13) and (14) are actually the same. Seeing this is a bit tricky as we have to play some serious "algebraic chess" to transform one equation into the other. Fortunately the exact details are not really that important. For those who are interested, it goes like this:

To start, apply the first exponential law (12) to equation (14). This gives the following identity

$$\begin{aligned} B(g_v^{v_0(s)+v_{I/O}(s)+v_{mid}(s)}, g_w^{w_0(s)+w_{I/O}(s)+w_{mid}(s)}) = \\ B(g_y^{t(s)}, g^{h(s)}) \bullet B(g_y^{y_0(s)+y_{I/O}(s)+y_{mid}(s)}, g) \end{aligned}$$

Then use the definition of g_v, g_w and g_y to transform the previous equation into the following expression

$$\begin{aligned} B(g^{r_v(v_0(s)+v_{I/O}(s)+v_{mid}(s))}, g^{r_w(w_0(s)+w_{I/O}(s)+w_{mid}(s))}) = \\ B(g^{r_y t(s)}, g^{h(s)}) \bullet B(g^{r_y(y_0(s)+y_{I/O}(s)+y_{mid}(s))}, g) \end{aligned}$$

In the next step, use the bilinear property of the map B as defined in (1). Since our group $\mathbb{G}$ uses multiplicative notation, we get

$$\begin{aligned} B(g,g)^{r_v(v_0(s)+v_{I/O}(s)+v_{mid}(s)) \cdot r_w(w_0(s)+w_{I/O}(s)+w_{mid}(s))} = \\ B(g,g)^{r_y t(s) \cdot h(s)} \bullet B(g,g)^{r_y(y_0(s)+y_{I/O}(s)+y_{mid}(s))} \end{aligned}$$

Because the term $B(g,g)^{r_y(y_0(s)+y_{I/O}(s)+y_{mid}(s))}$ is an element of $\mathbb{G}$, it has an inverse and we can use that to rewrite the previous equation into

$$\begin{aligned} B(g,g)^{r_v r_w(v_0(s)+v_{I/O}(s)+v_{mid}(s)) \cdot (w_0(s)+w_{I/O}(s)+w_{mid}(s)) - r_y(y_0(s)+y_{I/O}(s)+y_{mid}(s))} \\ = B(g,g)^{r_y t(s) \cdot h(s)} \end{aligned}$$

[11] Strictly speaking, this implies, that p is divisible by t at the evaluation point s only. However I can not say much about the implications on the security of the protocol.

Up to the constant factor $r_y = r_v \cdot r_w$, the exponent on the left side is actually nothing but $p(s)$ and based on this observation, we can transform the previous identity into

$$\begin{aligned} B(g,g)^{r_y p(s)} &= B(g,g)^{r_y t(s) \cdot h(s)} \\ B(g,g^{r_y})^{p(s)} &= B(g,g^{r_y})^{t(s) \cdot h(s)} \end{aligned}$$

To see the equivalence of this identity with equation (13), observe the group element $k \in \mathbb{G}_T$ defined $k := B(g, g^{r_y})$ is a generator, since B is not trivial. We therefore get

$$k^{p(s)} = k^{t(s) \cdot h(s)}$$

Ok. To proceed with our actual problem, i.e. to see that the claimed proof of our problem is indeed correct, we have to apply equation (14). Using the definition of our bilinear map, we get

$$\begin{aligned} B(1 \cdot 1 \cdot 12, 1 \cdot 12 \cdot 1) &= B(1,2) \bullet B(1 \cdot 2 \cdot 1, 2) \\ B(12,12) &= B(1,2) \bullet B(2,2) \\ 2^{log_2(12) \cdot log_2(12)}(mod_{23}) &= 2^{log_2(1) \cdot log_2(2)}(mod_{23}) \bullet 2^{log_2(2) \cdot log_2(2)}(mod_{23}) \\ 2^{10 \cdot 10}(mod_{23}) &= 2^{0 \cdot 1}(mod_{23}) \bullet 2^{1 \cdot 1}(mod_{23}) \\ 2^{10 \cdot 10}(mod_{23}) &= 1 \bullet 2^{1 \cdot 1}(mod_{23}) \\ 2 &= 2 \end{aligned}$$

This verifies, that the proofer was indeed able to find a polynomial p which is divisible by t at the secret point s. In addition it also verifies, that the circuit was used to compute p with respect to the input values $In = \{2,3,4\}$ and that $Out = 2$ is the correct outcome of the computation.

In principle this would be enough to verify the proper execution of the circuit. However the Pinocchio protocol requires some additional technical checks, related to Pinocchios particular choice of QAPs. The equations are:

$$\begin{aligned} B(g_v^{\alpha_v v_{mid}(s)}, g) &= B(g_v^{v_{mid}(s)}, g^{\alpha_v}) \\ B(g_w^{\alpha_w w_{mid}(s)}, g) &= B(g_w^{w_{mid}(s)}, g^{\alpha_w}) \\ B(g_y^{\alpha_y y_{mid}(s)}, g) &= B(g_y^{\alpha_y y_{mid}(s)}, g^{\alpha_y}) \\ B(g^Z, g^\gamma) &= B(g_v^{V_{mid}} g_w^{W_{mid}} g_y^{Y_{mid}}, g^{\beta\gamma}) \end{aligned}$$

In a nutshell they show that the $\alpha_v v_{mid}(s)$, $\alpha_w w_{mid}(s)$ and $\alpha_y y_{mid}(s)$ where

indeed computed properly (Exercise). In our problem we get

$$\begin{aligned} B(g_v^{\alpha_v v_{mid}(s)}, g) &= B(g_v^{v_{mid}(s)}, g^{\alpha_v}) \\ B(9,2) &= B(12,18) \\ 2^{log_2(9)\cdot log_2(2)}(mod_{23}) &= 2^{log_2(12)\cdot log_2(18)}(mod_{23}) \\ 2^{5\cdot 1}(mod_{23}) &= 2^{10\cdot 6}(mod_{23}) \\ 9 &= 9 \end{aligned}$$

$$\begin{aligned} B(g_w^{\alpha_w w_{mid}(s)}, g) &= B(g_w^{w_{mid}(s)}, g^{\alpha_w}) \\ B(1,2) &= B(1,9) \\ 2^{log_2(1)\cdot log_2(2)}(mod_{23}) &= 2^{log_2(1)\cdot log_2(9)}(mod_{23}) \\ 2^{0\cdot 1}(mod_{23}) &= 2^{0\cdot 5}(mod_{23}) \\ 1 &= 1 \end{aligned}$$

$$\begin{aligned} B(g_y^{\alpha_y y_{mid}(s)}, g) &= B(g_y^{y_{mid}(s)}, g^{\alpha_y}) \\ B(1,2) &= B(1,g^{16}) \\ 2^{log_2(1)\cdot log_2(2)}(mod_{23}) &= 2^{log_2(1)\cdot log_2(16)}(mod_{23}) \\ 2^{0\cdot 1}(mod_{23}) &= 2^{0\cdot 4}(mod_{23}) \\ 1 &= 1 \end{aligned}$$

$$\begin{aligned} B(g^Z, g^\gamma) &= B(g_v^{V_{mid}} g_w^{W_{mid}} g_y^{Y_{mid}}, g^{\beta\gamma}) \\ B(3,4) &= B(12\cdot 1\cdot 1, 18) \\ B(3,4) &= B(12,18) \\ 2^{log_2(3)\cdot log_2(4)}(mod_{23}) &= 2^{log_2(12)\cdot log_2(18)}(mod_{23}) \\ 2^{8\cdot 2}(mod_{23}) &= 2^{10\cdot 6}(mod_{23}) \\ 9 &= 9 \end{aligned}$$

Since all identities are satisfied the result of the verifier phase is $true$ as expected. If one of these equations would not hold, the result would be $false$. This shows that the claims of the proofer are indeed correct and that the results of the computation can be trusted under the cryptographic assumptions made in the Pinocchio protocol.

10 Conclusion

This was part one, of a small series. As you might have noticed, we haven't said much about the cheating key, nor have we said anything about zero knowledge.

We will look into this in part 2, where we proceed with our example to integrate zero knowledge and where we see how the cheating key can be used, to generate false proofs.

References

[HPR] Gentry, Howell, Parno and Raykova: "Pinocchio: Nearly Practical Verifiable Computation". In: 2013 IEEE Symposium on Security and Privacy.

[LB-SNARK] Rosario Gennaro, Michele Minelli, Anca Nitulescu, Michele Orru "Lattice-Based zk-SNARKs from Square Span Programs". In: City College of New York, USA.