Groth16 Algorithm

March 15, 2022

Contents

1 State of the art

1.1 Groth 16

The aim of the algorithm is to prove to a verifier that we know a secret without revealing it.

1.2 Some good to know

The algorithm will use a QAP to generate and to check the proof, and this QAP need to be generated by yourself or a **trust party**.
If we think about an architecture client-server where the server want to prove to the client that he knows a secret. Then the client (or a trusted party of the client) will have to generate a QAP to check this secret and send some keys to the server. The server will create the proof and send it back to the client. To finish the client will check the proof and accept or reject it.
This algorithm is sound, complet and can be zero-knowledge (ZK).

2 Definitions

2.1 QAP

A QAP is a representation of an arithmetic circuit. If we have Q an arithmetic circuit that compute something with $a_{1...l}$ the outputs and inputs of the circuit, and $a_{l+1...m}$ the witness of the circuit (i.e the intermediate values). A QAP is a triplet set of polynomials $\{A_i[X], B_i[X], C_i[X]\}_{i=0}^{m}$, each A_i, B_i, C_i are of degree n, such that

$$(\sum_{i=0}^{m} a_i A_i)(\sum_{i=0}^{m} a_i B_i) = \sum_{i=0}^{m} a_i C_i$$

Now considering our QAP defined above with an n-degree polynomial Z[X] we say this accept a vector $x \in F^n$ such that Z(x) divide our equation above.

2.1.1 Find Z

If we have our set of polynomials $\{A_i[X], B_i[X], C_i[X]\}_{i=0}^{m}$ we can find Z with this method :
Generate an arbitrary set of n points $S \subseteq F$. Construct A(x), B(x), C(x) in such a way that $A(s_i), B(s_i), C(s_i)$ are the i-th rows of A, B, C respectively. The Z(x) is defined in such a way that $\forall s \in S : Z(s) = 0$.
Then we have :

$$(\sum_{i=0}^{m} a_i A_i(x)) * (\sum_{i=0}^{m} a_i B_i(x)) = \sum_{i=0}^{m} a_i C_i(x) \bmod Z(x)$$

See [Pan] for more explanations. And for a concrete example go to practical section.

2.2 Bilinear map

$(p, G_1, G_2, G_T, e, g, h)$ define a bilinear map such that :
- $e : G_1 * G_2 \to G_T$
- G_1, G_2, G_T are groups of prime order p
- g is a generator of G_1
- h is a generator of G_2
- e(g,h) is a generator of G_T
- $e(g^a, h^b) = e(g, h)^{ab}$

2.3 Zero-knowledge proof

A zero-knowledge proof enable a prover to convince a verifier that a statement is true without revealing anything else. It have 3 core property :

- **Completeness** : Given a statement and a witness, the prover can convince the verifier.
- **Soundness** : A malicious prover cannot convince the verifier of a false statement.
- **Zero-knowledge** : The proof does not reveal anything but the truth of the statement, in particular it doesn't reveal the prover's witness.

3 Some notations

Given a bilinear map $(p, G_1, G_2, G_T, e, g, h)$, we write $[a]_1$ for g^a, $[b]_2$ for h^b, and $[c]_T$ for $e(g,h)^c$. With this notation $g = [1]_1, h = [1]_2$ and $e(g,h) = [1]_T$, while the neutral elements are $[0]_1$, $[0]_2$ and $[0]_T$.

4 Snark protocol in clear

If the client has a QAP with the polynomials $\{A_i[X],\ B_i[X],\ C_i[X]\}_{i=0}^{n}$ where he knows the value $a_{1\ldots l,l+1\ldots m}$ to solve it.
Polynomials A_i, B_i and C_i are of degree n and Z is of degree n.
Let's call the QAP **R** such that $R = \{F, m, l, \{A_i, B_i, C_i\}_{i=0}^{m}, \{Z_i\}_{i=0}^{n}\}$
We define 3 methods Setup, Prove and Verify such that :
$Setup(R) \to (\sigma, \tau)$
$Prove(R, \sigma, a_{i\ldots l}, a_{l+1\ldots m}) \to \pi$
$Verify(R, \sigma, a_{i\ldots l}, \pi) \to 0/1$

4.1 Setup

Setup(R) :

1. $\alpha \xleftarrow{\$} F^*$
2. $\beta \xleftarrow{\$} F^*$
3. $\gamma \xleftarrow{\$} F^*$
4. $\delta \xleftarrow{\$} F^*$
5. $s \xleftarrow{\$} F^*$
6. $\tau = (\alpha, \beta, \gamma, \delta, s)$
7. $\sigma = (\alpha, \beta, \gamma, \delta, \{s^i\}_{i=0}^{n-1})$
8. return (τ, σ)

4.2 Prove

$Prove(R, \sigma, a_{i\ldots l}, a_{l+1\ldots m})$:

1. r$\xleftarrow{\$}$F
2. k$\xleftarrow{\$}$F
3. U $= \alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta$
4. V $= \beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta$
5. We compute H(s) such that
$$(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$$
6. $\mathbf{Aa} = \sum_{i=l+1}^{m} a_i A_i$
7. $\mathbf{Ba} = \sum_{i=l+1}^{m} a_i B_i$
8. $\mathbf{Ca} = \sum_{i=l+1}^{m} a_i C_i$
9. $S = \frac{\beta \mathbf{Aa}(s) + \alpha \mathbf{Ba}(s) + \mathbf{Ca}(s)}{\delta}$
10. W $= S + \frac{H(s)Z(s)}{\delta} + Uk + rV - rk\delta$
11. $\pi = (U, V, W)$
12. return π

4.3 Verify

$\underline{Verify(R, \sigma, a_{i...l}, \pi):}$

1. $\mathbf{Aa} = \sum_{i=0}^{l} a_i A_i$
2. $\mathbf{Ba} = \sum_{i=0}^{l} a_i B_i$
3. $\mathbf{Ca} = \sum_{i=0}^{l} a_i C_i$
4. $Y = \frac{\beta \mathbf{Aa}(s) + \alpha \mathbf{Ba}(s) + \mathbf{Ca}(s)}{\gamma}$
5. if $UV == \alpha\beta + Y\gamma + W\delta$: return 1
6. else : return 0

See [Gro] for more explanations.

4.4 Proof of the equation

What we want is to check the following equality : $(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$
Detailled calculation for the if statement in verify function :
$UV = (\alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta)(\beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta)$

$= \alpha\beta + \alpha(\sum_{i=0}^{m} a_i B_i(s)) + k\alpha\delta + \beta(\sum_{i=0}^{m} a_i A_i(s)) + (\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) + k\delta(\sum_{i=0}^{m} a_i A_i(s)) + r\delta\beta + r\delta(\sum_{i=0}^{m} a_i B_i(s) + rk\delta\delta$

$\alpha\beta + Y\gamma + W\delta = \alpha\beta + (\frac{\beta\sum_{i=0}^{l} a_i A_i(s) + \alpha\sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s)}{\gamma})\gamma + (S + \frac{H(s)Z(s)}{\delta} + Uk + rV - rk\delta)\delta$

$= \alpha\beta + \beta\sum_{i=0}^{l} a_i A_i(s) + \alpha\sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s) + (S + Uk + rV - rk\delta)\delta + H(s)Z(s)$

$= \alpha\beta + \beta\sum_{i=0}^{l} a_i A_i(s) + \alpha\sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s) + (\frac{\beta\sum_{i=l+1}^{m} a_i A_i(s) + \alpha\sum_{i=l+1}^{m} a_i B_i(s) + \sum_{i=l+1}^{m} a_i C_i(s)}{\delta} + (\alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta)k + r(\beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta) - rk\delta)\delta + H(s)Z(s)$

$= \alpha\beta + \beta\sum_{i=0}^{l} a_i A_i(s) + \alpha\sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s) + \beta\sum_{i=l+1}^{m} a_i A_i(s) + \alpha\sum_{i=l+1}^{m} a_i B_i(s) + \sum_{i=l+1}^{m} a_i C_i(s) + ((\alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta)k + r(\beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta) - rk\delta)\delta + H(s)Z(s)$

$= \alpha\beta + \beta\sum_{i=0}^{m} a_i A_i(s) + \alpha\sum_{i=0}^{m} a_i B_i(s) + \sum_{i=0}^{m} a_i C_i(s) + ((\alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta)k + r(\beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta) - rk\delta)\delta + H(s)Z(s)$

$= \alpha\beta + \beta\sum_{i=0}^{m} a_i A_i(s) + \alpha\sum_{i=0}^{m} a_i B_i(s) + \sum_{i=0}^{m} a_i C_i(s) + (k\alpha + k\sum_{i=0}^{m} a_i A_i(s) + kr\delta + r\beta + r\sum_{i=0}^{m} a_i B_i(s) + rk\delta - rk\delta)\delta + H(s)Z(s)$

$= \alpha\beta + \beta\sum_{i=0}^{m} a_i A_i(s) + \alpha\sum_{i=0}^{m} a_i B_i(s) + \sum_{i=0}^{m} a_i C_i(s) + k\delta\alpha + k\delta\sum_{i=0}^{m} a_i A_i(s) + kr\delta\delta + r\delta\beta + r\delta\sum_{i=0}^{m} a_i B_i(s) + H(s)Z(s)$

$\alpha\beta + \alpha(\sum_{i=0}^{m} a_i B_i(s)) + k\alpha\delta + \beta(\sum_{i=0}^{m} a_i A_i(s)) + (\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) + k\delta(\sum_{i=0}^{m} a_i A_i(s)) + r\delta\beta + r\delta(\sum_{i=0}^{m} a_i B_i(s) + rk\delta\delta = \alpha\beta + \beta\sum_{i=0}^{m} a_i A_i(s) + \alpha\sum_{i=0}^{m} a_i B_i(s) + \sum_{i=0}^{m} a_i C_i(s) + k\delta\alpha + k\delta\sum_{i=0}^{m} a_i A_i(s) + kr\delta\delta + r\delta\beta + r\delta\sum_{i=0}^{m} a_i B_i(s) + H(s)Z(s)$

$\Leftrightarrow (\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$

4.5 Back to client-server architecture

So now if we go back to our problem where our client want to know if the server has some knowledge :

Client	Server
Compute R Setup(R) $\rightarrow (\sigma, \tau)$ Send σ, a_i inputs and R to the server	
	The server compute with his knowledge a_i outputs and the witness $Prove(R, \sigma, a_{i...l}, a_{l+1...m}) \rightarrow \pi$ Send π and a_i outputs to the client
Run $Verify(R, \sigma, a_{i...l}, \pi) \rightarrow 0/1$	

If the ouput is 1 the client knows that the server knows the witness and the outputs send by the server are correct. But if it's 0 the client knows that the server doesn't know the secret or has calculated a wrong output value. By the way this protocol is zero-knowledge on the witness, someone who intercept the communication will not learn anything about the witness.

4.6 Problem

But we have a problem with these scheme. Since the server knows α, β, δ, x. He can cheat by this way :
Pick U,V over F at random
Compute W=$\frac{UV-\alpha\beta-\sum_{i=0}^{l}(a_i(\beta A_i(s)+\alpha B_i(s)+C_i(s)))}{\delta}$
Now our equality is :
$UV = \alpha\beta + \frac{\beta\sum_{i=0}^{l} a_iA_i(s)+\alpha\sum_{i=0}^{l} a_iB_i(s)+\sum_{i=0}^{l} a_iC_i(s)}{\gamma}\gamma + \frac{UV-\alpha\beta-\sum_{i=0}^{l}(a_i(\beta A_i(x)+\alpha B_i(x)+C_i(x)))}{\delta}\delta$

$UV = \alpha\beta + \beta\sum_{i=0}^{l} a_iA_i(s) + \alpha\sum_{i=0}^{l} a_iB_i(s) + \sum_{i=0}^{l} a_iC_i(s) + UV - \alpha\beta - \sum_{i=0}^{l}(a_i(\beta A_i(s) + \alpha B_i(s) + C_i(s)))$

$UV = \alpha\beta + \beta\sum_{i=0}^{l} a_iA_i(s) + \alpha\sum_{i=0}^{l} a_iB_i(s) + \sum_{i=0}^{l} a_iC_i(s) + UV - \alpha\beta - \sum_{i=0}^{l}(a_i(\beta A_i(s) + \alpha B_i(s) + C_i(s)))$

0 = 0 So the if statement in the verify function will return true even if the person who compute the proof didn't know the $a_{l+1..m}$ (the witness).

5 Snark protocol in ZK

Now we have our protocol and the idea behind that. But an interesting property would be to have a Zero Knowledge protocol. What we mean is that someone who get the σ above doesn't learn anything about the problem we are trying to give a proof. To perform ZK computation we use some pairing with elliptic curves. $G1 * G2 \rightarrow Gt$
Now let's define R like $R = \{p, G1, G2, Gt, e, g, h, l, A, B, C, Z\}$. With $(p, G1, G2, Gt, e, g, h)$ a bilinear map.
For the same method Setup, Prove and Verify we just change the value of R and their output.

5.1 Setup

Setup(R) :

1. $\alpha \leftarrow^{\$} F^*$
2. $\beta \leftarrow^{\$} F^*$
3. $\gamma \leftarrow^{\$} F^*$
4. $\delta \leftarrow^{\$} F^*$
5. $s \leftarrow^{\$} F^*$
6. $\tau = (\alpha, \beta, \gamma, \delta, s)$
7. $\alpha_1 = [\alpha]_1$
8. $\beta_1 = [\beta]_1$
9. $\gamma_1 = [\gamma]_1$
10. $\delta_1 = [\delta]_1$
11. $\mathbf{S1} = \{[s^i]_1\}_{i=0}^{n-1}$
12. $\mathbf{Sa1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)+C_i(s)}{\delta}]_1\}_{i=l+1}^{m}$
13. $\mathbf{Sb1} = \{[\frac{s^i Z_i(s)}{\delta}]_1\}_{i=0}^{n-1}$
14. $\mathbf{Sc1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)+C_i(s)}{\gamma}]_1\}_{i=0}^{l}$
15. $\sigma1 = (\alpha_1, \beta_1, \gamma_1, \delta_1, \mathbf{S1}, \mathbf{Sa1}, \mathbf{Sb1}, \mathbf{Sc1}, g, G_1, p)$
16. $\beta_2 = [\beta]_2$

```
17. $\gamma_2 = [\gamma]_2$
18. $\delta_2 = [\delta]_2$
19. $\mathbf{S2} = \{[s^i]_2\}_{i=0}^{n-1}$
20. $\sigma 2 = (\beta_2, \gamma_2, \delta_2, \mathbf{S2}, h, G_2, p)$
21. $\sigma = (\sigma 1, \sigma 2, A_i, B_i, C_i)$
22. return $(\sigma, \tau)$
```

5.2 Prove

$\underline{Prove(\sigma, a_{i\ldots l}, a_{l+1\ldots m}):}$

```
1. r$\xleftarrow{\$}$F
2. k$\xleftarrow{\$}$F
3. $\mathbf{Aa} = \sum_{i=0}^{m} a_i A_i$
4. $\mathbf{Ba} = \sum_{i=0}^{m} a_i B_i$
5. $\mathbf{Ca} = \sum_{i=0}^{m} a_i C_i$
6. U = $\alpha_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Aa}_i} (\delta_1)^r$
7. $\mathbf{V1}$= $\beta_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Ba}_i} (\delta_1)^k$
8. $\mathbf{V2}$= $\beta_2 \Pi_{i=0}^{m} \mathbf{S2}_i^{\mathbf{Ba}_i} (\delta_2)^k$
9. We compute H(s) such that
   $$(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$$
10. $S = \Pi_{i=l+1}^{m} (\mathbf{Sa1}_i)^{a_i}$
11. W = $\frac{S(\Pi_{i=0}^{n} (\mathbf{Sb1}_i)^{H_i}) U^k \mathbf{V1}^k}{\delta_1{}^{rk}}$
12. $\pi = (U, \mathbf{V2}, W)$
13. return $\pi$
```

5.3 Verify

$\underline{Verify(\sigma, a_{i\ldots l}, \pi):}$

```
1. $Y = \Pi_{i=0}^{l} (\mathbf{Sc1}_i)^{a_i}$
2. if $e(U, \mathbf{V2}) == e(\alpha_1, \beta_2) e(Y, \gamma_2) e(W, \delta_2)$ : return 1
3. else : return 0
```

So compare to our previous protocol now some poeple who listen our channel will not know anything about our problem.

5.4 Proof of the equality

What we want is to check the following equality :

$$(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$$

Here is the detailled calcul for the if statement in verify :
$e(U, \mathbf{V2}) = e((\alpha_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Aa}_i} (\delta_1)^r), (\beta_2 \Pi_{i=0}^{m} \mathbf{S2}_i^{\mathbf{Ba}_i} (\delta_2)^k))$

$= [(\alpha + \sum_{i=0}^{m} a_i A_i(s) + r\delta)(\beta + \sum_{i=0}^{m} a_i B_i(s) + k\delta)]_T$

$e(\alpha_1, \beta_2)e(Y, \gamma_2)e(W, \delta_2) = e(\alpha_1, \beta_2)e(\Pi_{i=0}^{l} (\mathbf{Sc1}_i)^{a_i}, \gamma_2)e(\frac{S(\Pi_{i=0}^{n} (\mathbf{Sb1}_i)^{H_i})U^k \mathbf{V1}^k}{{\delta_1}^{rk}}, \delta_2)$

$= [\alpha\beta]_T[(\frac{\beta \sum_{i=0}^{l} a_i A_i(s) + \alpha \sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s)}{\gamma})\gamma]_T[(S + \frac{H(s)Z(s)}{\delta} + Uk + rV - rk\delta)\delta]_T$

$= [\alpha\beta + (\frac{\beta \sum_{i=0}^{l} a_i A_i(s) + \alpha \sum_{i=0}^{l} a_i B_i(s) + \sum_{i=0}^{l} a_i C_i(s)}{\gamma})\gamma + (S + \frac{H(s)Z(s)}{\delta} + Uk + rV - rk\delta)\delta]_T$

For our both side we have the same equation as the proof in 4.4 just in the g_T exponent.

5.5 Good point

We have solved our problem above, with the encryption of α, β, γ, s the server can't compute W as before :
W=$\frac{UV - \alpha\beta - \sum_{i=0}^{l} (a_i(\beta A_i(s) + \alpha B_i(s) + C_i(s)))}{\delta}$

or he has break the descrete logarithm problem in order to find s,β, α or γ.

6 More practical

6.1 R1CS from Polynomials

The aim of this section is to give an explanation with an example of how to construct a **rank-1 constraint system** (R1CS) from a polynomial.
Let's say we have the polynomial $x^3 + x + 5$ and we want to construct a corresponding R1CS.

6.1.1 Flattening

The first step is a "flattening" procedure. We transform our original polynomial to a succession of equation of type $x = y(op)z$.
Going back to our example we have :
$sym_1 = x * x$ (1)
$y = sym_1 * x$ (2)
$sym_2 = y + x$ (3)
$out = sym_2 + 5$ (4)

6.1.2 From flattening to R1CS

Now we can convert our succession of operation into a R1CS. An R1CS is a sequence of groups of three vectors (a,b,c) and the solution to an R1CS is a vector s where $sa * sb - sc = 0$
The length of each vector is equal to the total number of variables in the system, including a dummy variable "one" which represent the number 1.
So in our case the vector length is 7 which is :
"one", "x", "out", "sym_1", "y", "sym_2"
And the assignement of each variable will correspond to one operation. So for our first operation we have :
$a = [0, 1, 0, 0, 0, 0]$
$b = [0, 1, 0, 0, 0, 0]$
$c = [0, 0, 0, 1, 0, 0]$
I just skip the second one which is approximately the same as the first one, and for the third to perform the addition we have this :
$a = [0, 1, 0, 0, 1, 0]$
$b = [1, 0, 0, 0, 0, 0]$
$c = [0, 0, 0, 0, 0, 1]$
And the last :

$a = [5, 0, 0, 0, 0, 1]$
$b = [1, 0, 0, 0, 0, 0]$
$c = [0, 0, 1, 0, 0, 0]$
So the complete R1CS is :
A
$[0, 1, 0, 0, 0, 0]$
$[0, 0, 0, 1, 0, 0]$
$[0, 1, 0, 0, 1, 0]$
$[5, 0, 0, 0, 0, 1]$
B
$[0, 1, 0, 0, 0, 0]$
$[0, 1, 0, 0, 0, 0]$
$[1, 0, 0, 0, 0, 0]$
$[1, 0, 0, 0, 0, 0]$
C
$[0, 0, 0, 1, 0, 0]$
$[0, 0, 0, 0, 1, 0]$
$[0, 0, 0, 0, 0, 1]$
$[0, 0, 1, 0, 0, 0]$

6.1.3 From R1CS to QAP

Now we want to use our previous R1CS to convert it into a **quadratic arithmetic program** (QAP) form, which implement the same logic with polynomials. Such that if we evaluate the polynomials at x=1 then we get our first set of vectors, at x=2 the second and so forth.
To perform this transformation we will use the Lagrange Interpolation.
In our case we have 12 vectors of length 6. And we have to transform these into 6 groups of 3 polynomials, each one of degree 3. For example if we take the 1st column of A [0,0,0,5]. We can consider each element as the y-coordinate corresponding to x=1,2,3,4. So we get 4 sets of points (1,0), (2,0), (3,0), (4,5), it's an arbitrary interpretation we use to convert the R1CS into the QAP form. With lagrange interpolation we can find the polynomial passing through these 4 points.
It gives $0.833333333333333 * x^3 - 5.00000000000000 * x^2 + 9.16666666666667 * x - 5.00000000000000$.
We just have to do this for all our points and then we have :
A polynomials
$[-5.0, 9.166, -5.0, 0.833]$
$[8.0, -11.333, 5.0, -0.666]$
$[0.0, 0.0, 0.0, 0.0]$
$[-6.0, 9.5, -4.0, 0.5]$
$[4.0, -7.0, 3.5, -0.5]$
$[-1.0, 1.833, -1.0, 0.166]$
B polynomials
$[3.0, -5.166, 2.5, -0.333]$
$[-2.0, 5.166, -2.5, 0.333]$
$[0.0, 0.0, 0.0, 0.0]$
$[0.0, 0.0, 0.0, 0.0]$
$[0.0, 0.0, 0.0, 0.0]$
$[0.0, 0.0, 0.0, 0.0]$
C polynomials
$[0.0, 0.0, 0.0, 0.0]$
$[0.0, 0.0, 0.0, 0.0]$
$[-1.0, 1.833, -1.0, 0.166]$
$[4.0, -4.333, 1.5, -0.166]$
$[-6.0, 9.5, -4.0, 0.5]$
$[4.0, -7.0, 3.5, -0.5]$
And then to finish we have to defined Z as $(x - 1)(x - 2)(x - 3)$... the polynomials that is equal to zero at all points corresponding to logic gates. So in our case Z=(x-1)(x-2)(x-3)(x-4)
For a more complete explanation [But] and [Anob]

7 Go back to our problem

Now we have this protocol with zkSNARKs we want something quite similar. Back to a connexion client-server the problem is the following. A client want to evaluate a polynomial on a point, but he want that the server achieve this and gave him a proof of the correctness of the result.
We have something like this :

Client	Server
P$\in F[X]$ a polynomial, we want to evaluate it on x $\in F$ Send P and x to the server	
	Compute y = P(x) Compute π a proof that y is correct Send π and y to the client
With π anyone can check that y is correct	

In order to compute the proof of our computation we will use the SNARK protocol with QAP explained above.

7.1 Protocol in clear

Client	Server
With P$\in F[X]$ Compute R corresponding to P Setup(R) $\rightarrow (\sigma, \tau)$ Send σ, R and the a_i input to the server	
	The server compute R with the a_i input to get the : a_i output and the witness Then he generate : $Prove(R, \sigma, a_{1...l}, a_{l+1...m}) \rightarrow \pi$ Send π, a_i ouput, to the client
Someone who want to check the proof run $Verify(R, \sigma, a_{i...l}, \pi) \rightarrow 0/1$	

If the ouput is 1 the client know that the server computation is correct.

7.2 Protocol in ZK

Now we have our protocol let's imagine that the client don't wan't his polynomial P to be known by the server. So what we want is that the server don't learn anything about what he's currently computing.
A possible solution is to cipher the polynomials P before creating the R.
Maybe there is some other solution.... Currently thinking.

	Client	Communications	Server
Setup	With P$\in F[X]$ Compute R corresponding to P $\alpha, \beta, \gamma, \delta, s \leftarrow^{\$} F^*$ $\alpha_1 = [\alpha]_1, \beta_1 = [\beta]_1, \gamma_1 = [\gamma]_1$ $\delta_1 = [\delta]_1, \mathbf{S1} = \{[s^i]_1\}_{i=0}^{n-1},$ $\mathbf{CisDelta} = \{[\frac{C_i(s)}{\delta}]_1\}_{i=l+1}^{m}$ $\mathbf{Sa1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)}{\delta}]_1\}_{i=l+1}^{m},$ $\mathbf{Sb1} = \{[\frac{s^i Z_i(s)}{\delta}]_1\}_{i=0}^{n-1}$ $\beta_2 = [\beta]_2, \gamma_2 = [\gamma]_2, \delta_2 = [\delta]_2, \mathbf{S2} = \{[s^i]_2\}_{i=0}^{n-1}$ crs=$(\alpha_1, \beta_1, \gamma_1, \delta_1, \mathbf{S1},$ $\mathbf{CisDelta}, \mathbf{Sa1}, \mathbf{Sb1}, \beta_2, \gamma_2, \delta_2, \mathbf{S2}, g, h, G_1, G_2)$ $S = \{g^{\frac{\beta x_i(s)+\alpha y_i(s)+z_i(s)}{\gamma}}\}_{i=0}^{n}$ $\mathbf{CisDeltaStart} = \{[\frac{C_i(s)}{\delta}]_1\}_{i=0}^{l}$ $\mathbf{Sc1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)}{\gamma}]_1\}_{i=0}^{l}$ $\mathbf{Sc} = \{[C_i s^i]_1\}_{i=0}^{n-1}$ vk=$(\alpha_1, \beta_2, S, \mathbf{CisDeltaStart}, \gamma_2, \delta_2, \mathbf{Sc1}, \mathbf{Sc})$	$\xrightarrow{crs, a_{input}, R}$	
Eval		$\xleftarrow{\pi, a_{output}}$	Compute $a_{witness}, a_{output}$ from R on a_{input} $r, k \leftarrow^{\$} F*$ $\mathbf{Aa} = \sum_{i=0}^{m} a_i A_i$ $\mathbf{Ba} = \sum_{i=0}^{m} a_i B_i$ $\mathbf{Ca} = \sum_{i=0}^{m} a_i C_i$ U $= \alpha_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Aa}_i} (\delta_1)^r$ $\mathbf{V1}= \beta_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Ba}_i} (\delta_1)^k$ $\mathbf{V2}= \beta_2 \Pi_{i=0}^{m} \mathbf{S2}_i^{\mathbf{Ba}_i} (\delta_2)^k$ We compute H(s) such that $(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s))$ $= \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$ $S = \Pi_{i=l+1}^{m} (\mathbf{Sa1}_i)^{a_i} \mathbf{CisDelta}_i$ W $= \frac{\Pi_{i=0}^{n} S_i (\Pi_{i=0}^{n} (\mathbf{Sb1}_i)^{H_i}) U^k \mathbf{V1}^k}{\delta 1^{rk}}$ $\pi = (U, W, \mathbf{V2})$

Verif	$Y = \Pi_{i=0}^{l} (\mathbf{Sc1}_i)^{a_i} \mathbf{CisDeltaStart}_i$
	if $e(U, \mathbf{V2}) == e(\alpha_1, \beta_2) e(Y, \gamma_2) e(W, \delta_2)$: return 1
	else : return 0		

7.3 Evaluation of the polynomial

7.3.1 Setup

With p prime, g the generator of G1, h the generator of G2 and e such that e(g,h) is the generator of Gt. Let's call the QAP R such that $R = \{F, m, l, \{A_{i,j}, B_{i,j}, C_{i,j}\}_{i=0,j=0}^{i=m,j=n}, \{Z_i\}_{i=0}^{n}\}$

Setup$(1^{\lambda}, R)$:

Let's compute :

1. $\alpha \leftarrow^{\$} F*$
2. $\beta \leftarrow^{\$} F*$
3. $\gamma \leftarrow^{\$} F*$
4. $\delta \leftarrow^{\$} F*$
5. $s \leftarrow^{\$} F*$
6. $\alpha_1 = [\alpha]_1$
7. $\beta_1 = [\beta]_1$
8. $\gamma_1 = [\gamma]_1$
9. $\delta_1 = [\delta]_1$

10. $\mathbf{S1} = \{[s^i]_1\}_{i=0}^{n-1}$

11. $\mathbf{CisDelta} = \{[\frac{C_i(s)}{\delta}]_1\}_{i=l+1}^{m}$

12. $\mathbf{Sa1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)}{\delta}]_1\}_{i=l+1}^{m}$

13. $\mathbf{Sb1} = \{[\frac{s^i Z_i(s)}{\delta}]_1\}_{i=0}^{n-1}$

14. $\beta_2 = [\beta]_2$

15. $\gamma_2 = [\gamma]_2$

16. $\delta_2 = [\delta]_2$

17. $\mathbf{S2} = \{[s^i]_2\}_{i=0}^{n-1}$

18. crs=$(\alpha_1, \beta_1, \gamma_1, \delta_1, \mathbf{S1}, \mathbf{CisDelta}, \mathbf{Sa1}, \mathbf{Sb1}, \beta_2, \gamma_2, \delta_2, \mathbf{S2}, g, h, G_1, G_2)$

19. $S = \{g^{\frac{\beta x_i(s)+\alpha y_i(s)+z_i(s)}{\gamma}}\}_{i=0}^{n}$

20. $\mathbf{CisDeltaStart} = \{[\frac{C_i(s)}{\delta}]_1\}_{i=0}^{l}$

21. $\mathbf{Sc1} = \{[\frac{\beta A_i(s)+\alpha B_i(s)}{\gamma}]_1\}_{i=0}^{l}$

22. $\mathbf{Sc} = \{[C_i s^i]_1\}_{i=0}^{n-1}$

23. vk=$(\alpha_1, \beta_2, S, \mathbf{CisDeltaStart}, \gamma_2, \delta_2, \mathbf{Sc1}, \mathbf{Sc})$

24. return crs, vk

Send **crs** to the prover and **vk** to the verifier.

7.3.2 Prove

We have $a_0 = 1$ it's a constant due to the R1CS constraint. Prove(crs, $a_{1..l}$, $a_{l+1..m}$) :

1. $r \leftarrow^{\$} F*$

2. $k \leftarrow^{\$} F*$

3. $\mathbf{Aa} = \sum_{i=0}^{m} a_i A_i$

4. $\mathbf{Ba} = \sum_{i=0}^{m} a_i B_i$

5. $\mathbf{Ca} = \sum_{i=0}^{m} a_i C_i$

6. U $= \alpha_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Aa}_i} (\delta_1)^r$

7. $\mathbf{V1}$= $\beta_1 \Pi_{i=0}^{m} \mathbf{S1}_i^{\mathbf{Ba}_i} (\delta_1)^k$

8. $\mathbf{V2}$= $\beta_2 \Pi_{i=0}^{m} \mathbf{S2}_i^{\mathbf{Ba}_i} (\delta_2)^k$

9. We compute H(s) such that
$$(\sum_{i=0}^{m} a_i A_i(s))(\sum_{i=0}^{m} a_i B_i(s)) = \sum_{i=0}^{m} a_i C_i(s) + H(s)Z(s)$$

10. $S = \Pi_{i=l+1}^{m} (\mathbf{Sa1}_i)^{a_i} \mathbf{CisDelta}_i$

11. W $= \frac{(\Pi_{i=0}^{n} S_i)(\Pi_{i=0}^{n} (\mathbf{Sb1}_i)^{H_i}) U^k \mathbf{V1}^k}{\delta 1^{rk}}$

12. $\pi = (U, W, \mathbf{V2})$

13. return $\pi = (U, W, \mathbf{V2})$, a_l // the one corresponding to the output.

7.3.3 Verify

Verify(vk, $a_{0..l-1}$, a_l, π) :

1. $Y = \Pi_{i=0}^{l}(\mathbf{Sc1}_i)^{a_i}\mathbf{CisDeltaStart}_i$
2. if $e(U, \mathbf{V2}) == e(\alpha_1, \beta_2)e(Y, \gamma_2)e(W, \delta_2)$: return 1
3. else : return 0

8 Implementation in libsnark

The major difference is in the setup of the verification key and the provable key, where we don't send the A_i, B_i, C_i but the $A_i(s), B_i(s), C_i(s)$ with the R1CS constrainsts. And for the $\alpha, \beta, \sigma, \gamma$ they are already injected in the $A_i(s), B_i(s), C_i(s)$ above.

9 Results with libsnark

I've done some implementation with libsnark and here are my results compare to our protocol with pairing and paillier, for a polynomial in clear with libsnark :

9.1 Time of calculation

	Libsnark			Paillier 1024			Paillier 2048		
Degree	Setup	Client	Server	Setup	Client	Server	Setup	Client	Server
256	0.1678	0.0249	0.1640	0.1824	0.0011	0.1638	0.6749	0.0017	0.2653
512	0.2946	0.0261	0.2843	0.3851	0.0011	0.3165	1.1360	0.0019	0.5072
1024	0.5177	0.0272	0.5551	0.6832	0.0011	0.6485	1.8827	0.0017	0.9836
2048	0.9732	0.0285	0.9687	1.3459	0.0011	1.2551	3.8696	0.0017	1.9426
4096	1.7896	0.0267	1.7556	2.7452	0.0011	2.5792	7.5359	0.0017	3.9480
8192	3.0986	0.026 84	3.1388	5.5579	0.0011	5.3739	14.2259	0.0017	7.4721
16384	5.9548	0.0270	6.0528	10.6597	0.0011	10.4383	29.2171	0.0020	15.3933
32768	10.8519	0.0268	11.3400	21.3708	0.0011	20.3710	56.6373	0.0017	30.4662
65536	20.1288	0.0266	21.5019	41.8292	0.0011	41.0267	113.1845	0.0017	61.1323
131072	37.8404	0.0265	40.9986	83.8971	0.0011	82.3237	225.7390	0.0019	122.0703

9.2 Data to save

Data in bits

	Libsnark				
Degree	Client save after setup	Data send to server setup	Server save after setup	Eval data send	Response of data eval
256	3629	1 482 698	1 482 698	254	2548
512	3629	2 966 474	2 966 474	254	2548
1024	3629	5 934 026	5 934 026	254	2548
2048	3629	11 869 130	11 869 130	254	2548
4096	3629	23 739 338	23 739 338	254	2548
8192	3629	47 479 754	47 479 754	254	2548
16384	3629	94 960 586	94 960 586	254	2548
32768	3629	189 922 250	189 922 250	254	2548
65536	3629	379 845 578	379 845 578	254	2548
131072	3629	759 692 234	759 692 234	254	2548

Data in bytes

	Libsnark				
Degree	Client save after setup	Data send to server setup	Server save after setup	Eval data send	Response of data eval
256	453.625	185 337.25	185 337.25	31.75	318.5
512	453.625	370 809.25	370 809.25	31.75	318.5
1024	453.625	741 753.25	741 753.25	31.75	318.5
2048	453.625	1 483 641.25	1 483 641.25	31.75	318.5
4096	453.625	2 967 417.25	2 967 417.25	31.75	318.5
8192	453.625	5 934 969.25	5 934 969.25	31.75	318.5
16384	453.625	11 870 073.25	11 870 073.25	31.75	318.5
32768	453.625	23 740 281.25	23 740 281.25	31.75	318.5
65536	453.625	47 480 697.25	47 480 697.25	31.75	318.5
131072	453.625	94 961 529.25	94 961 529.25	31.75	318.5

10 More link and lectures

Some others sources that help me to understand how SNARK works :

- This one explain briefly how to create the QAP and how the SNARK protocol works in ZK [ENS]
- This link explain how to create a QAP from a polynomial equality [But]
- This link give an example of how to construct a zkSNARK from an arithmetic circuit [May]
- This link explain how to agregating multiple Groth16 proofs for the same SRS [Anoa]

Articles

[Anoa] Anonymous. “Practical Groth16 Aggregation”. In: (). URL: https://docs.zkproof.org/pages/standards/accepted-workshop4/proposal-aggregation.pdf.

[Anob] Anonymous. “zkSNARKs: R1CS and QAP”. In: (). URL: https://risencrypto.github.io/zkSnarks/.

[But] Vitalik Buterin. “How to construct a QAP”. In: (). URL: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[ENS] ENS. “QAP”. In: (). URL: https://www.di.ens.fr/~nitulesc/files/slides/QAP.pdf.

[Gro] Jens Groth. “On the Size of Pairing-based Non-interactive Arguments”. In: (). URL: https://eprint.iacr.org/2016/260.pdf.

[May] Hartwig Mayer. “zk-SNARK explained: Basic Principles”. In: (). URL: https://blog.coinfabrik.com/wp-content/uploads/2017/03/zkSNARK-explained_basic_principles.pdf.

[Pan] Alisa Pankova. “Succinct Non-Interactive Arguments from Quadratic Arithmetic Programs”. In: (). URL: https://courses.cs.ut.ee/MTAT.07.022/2013_fall/uploads/Main/alisa-report.