

ZK-SNARKs

Suppose we have an equation that looks this way:

$$x^3 + x + 5 = 35$$

We want to use ZK-knowledge technique to prove that we know the solution but without revealing the actual value. Let's see how can we do it.

TL;DR

1. Prover converts the problem statement into Arithmetic Circuit which involves variables and gates (normally thousands and maybe millions).
2. Prover transforms Arithmetic Circuit into set of equations - R1Cs (Rank 1 constraints system).
3. Prover converts R1CS into QAP (Quadratic Arithmetic Problem), turning set of equations into 1 equation in polynomial view that must hold true at selected points. The polynomial representation is useful because a degree d polynomial can have at most d shared points with any other polynomial. So if we, for example, consider an integer range of x from 1 to 10^{77} , the number of points where evaluations are different is $10^{77} - d$. Henceforth the probability that x accidentally "hits" any of the d shared points is equal to $d/10^{77}$, which is considered negligible.
4. Verifier accepts these points and verifies through pairing checks on Elliptic Curve that these points are valid for the provided equation. This step is needed to show that we know QAP equation without revealing polynomials coefficients (so prover and verifier can't cheat) and at the same time verifier would be able to check that the statement is valid.

1. Problem statement to Arithmetic Circuit

We have to transform our equation into Arithmetic Circuit. It's about converting an original equation into a sequence of statements that are of two forms:

$y = x$ (where y can be a variable or a number) and

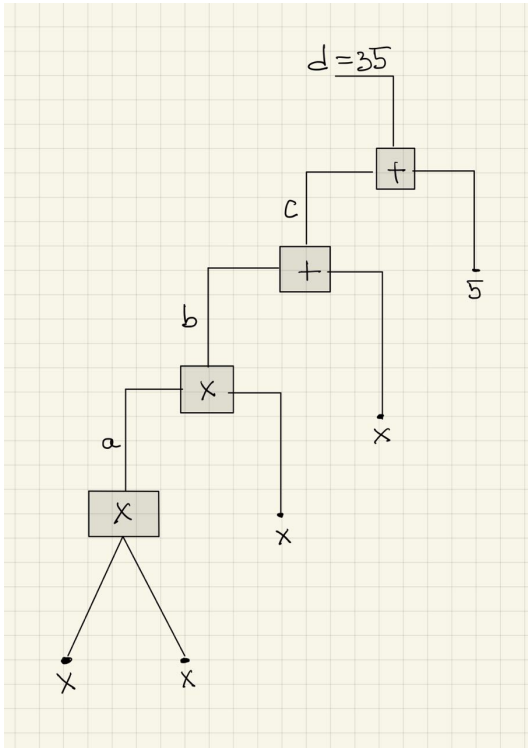
$y = x \text{ (op) } z$ (where op can be $+$, $-$, $*$, $/$ and y and z can be variables, numbers or themselves sub-expressions). Also there is no such operations as power of 2 or whatever so we have to replace all such operations with variables:

```

a = x * x // x2
b = a * x // x3
c = b + x // x3 + x
d = b - x
e = c + 5 // x3 + x + 5

```

Here is our Arithmetic Circuit in graphical format:



As you can see - we have 4 gates and 6 variables (wires) $(1, x, a, b, c, d)$. 1 is used to represent all constants.

2. Arithmetic Circuit to 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 s must satisfy the equation $s.a' * s.b' - s.c' = 0$ ($*$ - is multiplication), or say different: $s.a' * s.b' = s.c'$. You can think as a and b - inputs to the gates and c is output.

Let's try to create equations. Cause we know the solution ($x=3$) we can create R1CS based on the solution.

So if we take the variables, our vector s will look like:

```
s = [1, x, a, b, c, d]
```

Vectors a' , b' , c' will have the same dimension as s . So vectors will have $\text{len} = 6$ (6 variables).

In case $x=3$,

```
s = [1, x, a, b, c, d] = [1, 3, 9, 27, 30, 35].
```

Then:

```
1st gate:  
x * x = a = 9  
a' = [0, 1, 0, 0, 0, 0]  
b' = [0, 1, 0, 0, 0, 0]  
c' = [0, 0, 1, 0, 0, 0]
```

```
2nd gate:  
b = a * x = 9 * x = 27  
a' = [0, 0, 1, 0, 0, 0]  
b' = [0, 1, 0, 0, 0, 0]  
c' = [0, 0, 0, 1, 0, 0]
```

```
3d gate:  
c = b + x = (b + x) * 1  
a' = b + x = [0, 1, 0, 1, 0, 0]  
b' = 1 = [1, 0, 0, 0, 0, 0]  
c' = [0, 0, 0, 0, 1, 0]
```

```
4th gate:  
d = c + 5 = (c + 5) * 1  
a' = c + 5 = [5, 0, 0, 0, 1, 0]  
b' = 1 = [1, 0, 0, 0, 0, 0]  
c' = [0, 0, 0, 0, 0, 1]
```

So next step is to turn this set of constraints into 1 set in polynomial view.

3. R1CS to QAP

Now we will convert our constraints into single polynomial, where evaluating the polynomial at each x coordinate will represent one of our constraints.

So $x = 1$ will give us 1st constraint, $x = 2$ will give us 2nd etc.

And there is an Interpolation Theorem:

There is only 1 polynomial of degree N which goes through $N + 1$ points.

Since we have 4 points, our polynomial will be degree 3 and $x = [1..4]$

Our polynomial will look like:

$$A(x) * B(x) - C(x) = 0$$

Where:

$$\begin{aligned} A &= A_1(x)*1 + A_2(x)*x + A_3(x)*a + A_4(x)*b + A_5(x)*c + A_6(x)*d \\ B &= B_1(x)*1 + B_2(x)*x + B_3(x)*a + B_4(x)*b + B_5(x)*c + B_6(x)*d \\ C &= C_1(x)*1 + C_2(x)*x + C_3(x)*a + C_4(x)*b + C_5(x)*c + C_6(x)*d \end{aligned}$$

$$\begin{aligned} A_i(x) &= a[i] \text{ for the corresponding equation, for example } A_2(3) = 1, \text{ etc.} \\ B_i(x) &= b[i] \text{ for the corresponding equation} \\ C_i(x) &= c[i] \text{ for the corresponding equation} \end{aligned}$$

So we turned our set of 4 constraints into 1 vectorized constraint which is polynomial.

Let's modify our polynomial to such a view that:

$$A(x) * B(x) - C(x) = t * Z(x)$$

Z is defined as $(x - 1) * (x - 2) * (x - 3) \dots$ - the simplest polynomial that is equal to zero at all points that correspond to logic gates. It is a fact of algebra that any polynomial that is equal to zero at all of these points has to be a multiple of this minimal polynomial. Example:

Suppose we have a set of points $(-2, 0), (1, 0), (3, 0)$. The minimal polynomial would be $(x + 2)(x - 1)(x - 3)$, which has degree 3. This polynomial is equal to zero at each of the points $(-2, 0), (1, 0), (3, 0)$. Now, if we have another polynomial $P(x)$ that is equal to zero at each of these points, we know that $P(x)$ must be a multiple of $(x + 2)(x - 1)(x - 3)$.

3). In other words, there exists some constant k such that $P(x) = k * (x + 2)(x - 1)(x - 3)$.

What's important here is that if we want to test correctness of this, we don't have to evaluate the polynomial $A \cdot s * B \cdot s - C \cdot s$ at every point corresponding to a gate; instead, we use t (which is provided by prover) and divide it by another polynomial, z , and check that z evenly divides t - that is, the division t / z leaves no remainder.

If we'll try to falsify any of the variables in the R1CS solution that we are deriving this QAP solution from — say, set the last one to 31 instead of 30, then we get a t polynomial which doesn't satisfy the equation and would not be a multiple of z .

So currently our scheme works kinda this way:

1. Verifier chooses a random value for x and evaluates his QAP locally
2. Verifier gives x to the prover and asks to evaluate the polynomial in question
3. Prover evaluates his polynomial at x and gives the result to the verifier
4. Verifier checks if the local result is equal to the prover's result, and if so then the statement is proven with a high confidence

So as a prover you have to give A, B, C and to prove that you know the solution. But there are problems:

1. A, B, C are big and you don't want to transfer that much data (as we remember there are thousands and maybe millions of variables and gates).
2. We also want to hide A, B, C from the verifier because otherwise he will be able to cheat and verify false proofs. The same is for prover.
3. And we want our prove scheme to be non-interactive since prover and verifier can be both malicious.

So you want such properties from your proof:

1. To be succinct.
2. To be Zero Knowledge, so to hide solution S .
3. Verifying should be able to easily verify the statement without recalculation of all the stuff.

4. QAP to Elliptic Curve

We want to show that we know A, B, C and t without revealing polynomials and verifier can check that the statement is valid. We will use Elliptic Pairing for this.

All math will be done by modulo N , so all numbers will fit in $n=[0..N-1]$.

Let's take a look at some Elliptic Curve:

$$y^2 = ax^3 + bx + c$$

And there is **The Elliptic Curve Discrete Logarithm Problem** - given a point G on an elliptic curve and some multiple of that point $n*G$, it's computationally infeasible to say what is n .

So the idea is to transform such polynomials into the points which will hold the same relationship but because of **The Elliptic Curve Discrete Logarithm Problem** real values of computed polynomials will be impossible to reveal.

We will do this by using reference point G . We can say that if we have some value n , bijection of this value on elliptic curve will be nG .

So in our case n is a secret, nG - public key pair.

Let's then say we have parameters n , p and q and there's a relation:

$$2n + 3p = q$$

Since our relation holds for our parameters, it will also be held for points, so:

$$2N + 3P = Q, \text{ where } N = nG, P = pG, Q = qG.$$

As you see, if we have a linear relationship between values, we can easily transform this into the points and still have the same relationship being verifiable.

But if we have relation:

$$n * p = q$$

it's not translated into: (since multiplication is undefined)

$$N * P = Q$$

So there are Elliptic Curve pairings. A **pairing** is a map that takes two points as its input and outputs another point. We'll call our pairing e . And for some reference points g_1 , g_2 , and g_3 on the curve and integers a and b such properties are true:

$$e : G_1 \times G_1 \rightarrow G_T$$

$$e(a * g_1, g_2) = e(g_1, g_2)^a$$

$$e(g_1, b * g_2) = e(g_1, g_2)^b$$

$$e(g_1 + g_2, g_3) = e(g_1, g_3)e(g_2, g_3)$$

$$e(g_1, g_2 + g_3) = e(g_1, g_2)e(g_1, g_3)$$

Now let's take a look at original equation. Suppose $np = q$ and there are points such as $N = nG$, $P = pG$, $Q = qG$. Then:

$$\begin{aligned} e(N, P) &= e(nG, pG) = e(G, G)^{np} \\ e(Q, G) &= e(G, G)^q \\ e(G, G)^{np} &= e(G, G)^q \Rightarrow np = q. \end{aligned}$$

So we can say if we can check that these two points are equal - $e(N, P)$ and $e(Q, G)$, then our relation holds true: $np = q$. The transformations above include such things as **generator of the group of points**, **bilinear pairing function** and we won't go in details about it considering it's enough for high-level understanding.

Let's go back to our polynomial:

$$A(x) * B(x) - C(x) = t * Z(x)$$

As you can see, since we will move from polynomial to points on Elliptic Curve, we don't need to show that our Elliptic Curves will be the same at all points but rather that at some point on Elliptic Curve our polynomial will match right solution.

Let's choose some point t as well as points A , B , C and calculate our polynomials:

$$\begin{aligned} A(t) * B(t) - C(t) &= t' * Z(t), \\ \text{where } A(t) &= \sum Si(Ai(t)*G), B(t) = \sum Si(Bi(t)*G), C(t) = \sum Si(Ci(t)*G), Z(t) = (t-1)*(t-2)... \\ &\text{and } Si - \text{solution vector element} \end{aligned}$$

Then by using all the above properties, move to Elliptic Curve pairings:

$$\begin{aligned} A(t) * B(t) - C(t) &= t' * Z(t), \\ e(AG, BG)/e(CG, G) &= e(t'G, ZG) \Rightarrow \end{aligned}$$

$$e(G, G)^{AB} * e(G, G)^{(-C)} = e(G, G)^{(t'Z)} \Rightarrow \\ AB - C = t'Z$$

So prover sends in proof not polynomials but points AG , BG , CG , $t'G$. And the real A , B , C and t' are blinded. And verifier checks that such equation holds true on Elliptic Curve:

$$e(AG, BG) * e(CG, -G) = e(t'G, ZG).$$

And the last part is λ . So the idea is that it's not really safe to provide only points AG , BG , CG , $t'G$ directly because prover can cheat and send arbitrary points that may satisfy required solution. So instead we will involve parameter λ which should be unknown for both prover and verifier. And prover sends not only AG , BG but also λAG , λBG which are computed during setup phase.

And then verifier checks such arbitrary pairings, suppose it's $e(AG, \lambda BG)$ and $e(\lambda AG, BG)$. And as you may remember, $Ai(x) = a[i]$ (value in R1CS form) for the corresponding equation. Then:

$$e(AG, \lambda BG) = e(A, B)^{\lambda} \\ e(\lambda AG, BG) = e(A, B)^{\lambda}$$

Since λ is unknown for both prover and sender and it's computationally infeasible to calculate it from λAG or λBG , above relation will hold true then and only then if prover sent values that satisfy the solution.

Now, whoever wants to verify that my secret solution actually satisfies the cubic equation can do so by computing pairings and checking the pairing equation: $e(AG, BG) * e(CG, -G) = e(t'G, ZG)$.

It goes far beyond the document about how exactly pairing check should be performed but let's see how it can be done on a high-level.

As an example:

Consider the two points $P = (1, 2)$ and $Q = (3, 5)$ on the curve $y^2 + xy = x^3 + x^2 + 7$ over $GF(2^3)$ (Galois Field with 2^3 elements). To do the pairing we should:

1. Map the points to elements in the finite field:

The first step is to map the points to elements in the field $GF(2^3)$ which

is Galois Field with 2^3 elements. This can be done using a process called "coordinate compression".

2. Evaluate the pairing function:

Next, Depends on chosen pairing-friendly curve pairing is evaluated using the mapped points. The pairing function itself involves several mathematical operations, including exponentiation and reduction, but the exact details are beyond the scope of this answer.

3. Reduce the result to a representative in the target group:

Finally, the result of the pairing is reduced to a representative in the target group, which is usually the group of non-zero elements of a finite field. This is done using a process called "final exponentiation".

4. Compare the binary representation of the pairing values, which are elements of a finite field.

Sources:


Groth16

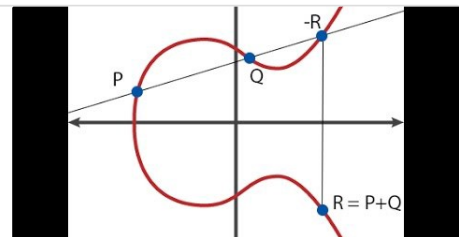
The Zero Knowledge Blog

 <http://www.zeroknowledgeblog.com/index.php/groth16>

Exploring Elliptic Curve Pairings


Trigger warning: math. One of the key cryptographic primitives behind various constructions, including deterministic threshold signatures, zk-SNARKs and other simpler forms of zero-knowledge proofs is the elliptic

 <https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>



Quadratic Arithmetic Programs: from Zero to Hero

There has been a lot of interest lately in the technology behind zk-SNARKs, and people are increasingly trying to demystify something that many have come to call "moon math" due to its perceived sheer indecipherable

 <https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>

• Algebraic Circuit

• R1CS

• QAP

• Linear PCP

• Linear Interactive Proof

