Polynomials and Elliptic Curves

rkm0959

Security Researcher @ KALOS

February 9th

Outline

The Goal

We want succint proofs. ZK-ness is easy to add in PLONKish.

We want our computation to be provable with a smaller one.

Table of Contents

The Beginning

Alice wants to prove that

$$u_i \cdot v_i = w_i$$

for all $i \in [0, n)$. The values are in $\mathbb{F}_p$, but that's not important for now.

Sending every u_i, v_i, w_i is pointless. What to do?

Polynomials Enter the Battle: Interpolation

Lagrange Interpolation

For x_i, y_i with $i \in [0, n)$ and $x_i \neq x_j$ for all $i \neq j$, there is a unique polynomial f of degree less than n such that $f(x_i) = y_i$ for all $i \in [0, n)$.

In general cases, takes $\mathcal{O}(n \log^2 n)$ via Divide & Conquer + FFT.

In a special case $x_i = \omega^i$, this *is* FFT, so $\mathcal{O}(n \log n)$.

FFT is another monster, so let's cut to the chase: it costs $\tilde{\mathcal{O}}(n)$.

Polynomials Enter the Battle: Interpolation

Therefore, we may consider polynomials U, V, W such that

$$U(\omega^i) = u_i, \quad V(\omega^i) = v_i, \quad W(\omega^i) = w_i$$

holds. Here, $\omega^n = 1$. Details will be added later.

We now want to prove, for $i \in [0, n)$

$$U(\omega^i) \cdot V(\omega^i) = W(\omega^i)$$

Polynomials Enter the Battle: Division

This is equivalent to

$$U(x) \cdot V(x) - W(x) \equiv 0 \pmod{\prod_{i=0}^{n-1}(x - \omega^i)}$$

which is, after simplification,

$$U(x) \cdot V(x) - W(x) \equiv 0 \pmod{x^n - 1}$$

or, going further,

$$U(x) \cdot V(x) - W(x) = (x^n - 1)Q(x)$$

Polynomials Enter the Battle: Schwartz-Zippel

Schwartz-Zippel Lemma

$P \in \mathbb{F}[x_1, \cdots, x_n]$ is a non-zero polynomial with total degree d over finite field $\mathbb{F}$. If $r_1, r_2, \cdots, r_n$ are i.i.d. randomly selected from $\mathbb{F}$, then

$$Pr[P(r_1, r_2, \cdots, r_n) = 0] \leq \frac{d}{|\mathbb{F}|}$$

In other words, non-zero polynomials evaluated on a random point is non-zero with an overwhelming probability, if $|\mathbb{F}|$ is large enough.

Polynomials Enter the Battle: PCS

Therefore, it suffices to select random $t \in \mathbb{F}$ and check

$$U(t) \cdot V(t) - W(t) = (t^n - 1)Q(t)$$

Our values u_i, v_i, w_i are *embedded* in U, V, W.
However, U, V, W are big data structures, and evaluating $U(t)$ is not an easy task. If someone could prove that $U(t)$ is evaluated correctly...

Polynomials Enter the Battle: PCS

In general, Polynomial Commitment Schemes allow

- Commit: $f \to \mathcal{C}(f)$. This is binding and hiding.
- Prove: One can generate proof π such that $f(x) = y$
- Verify: One can verify $f(x) = y$ given $x, y, \mathcal{C}(f), \pi$.

Polynomial Commitment Schemes

There are many things to look for in a PCS:

- Requirement of a Trusted Setup
- Proving Time & Verifying Time
- Useful Properties of $\mathcal{C}(f)$ (linearity)
- Additional Techniques (degree bounding, etc)

Polynomial Commitment Schemes

There are many types of PCS:

- Pairing Based 타원곡선 기반
- Discrete Logarithm Based
- Hash / Code Based
- Groups of Unknown Order Based

The first two usually use Elliptic Curves, which is why we need to learn.

The Remaining Pieces

Selecting t: The selection of t should be done after the $\mathcal{C}(U), \mathcal{C}(V), \mathcal{C}(W)$ are committed, and it should be random - but since we want everything *non-interactive*, we apply Fiat-Shamir to select t via a hash function.

Adding ZK: It'll probably depend on what SNARK you are using - but usually, adding random points to interpolate U, V, W on, or adding random multiples of $(x^n - 1)$ works in PLONKish. For more details involving how to add ZK, take a look at the Appendix of 2022/086. (PLONKUP)

Table of Contents

Definition

As we will not use characteristic 2 or 3, we choose

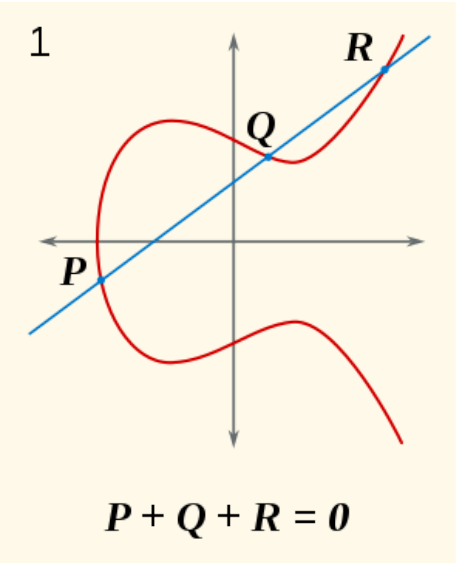
$$y^2 = x^3 + ax + b$$

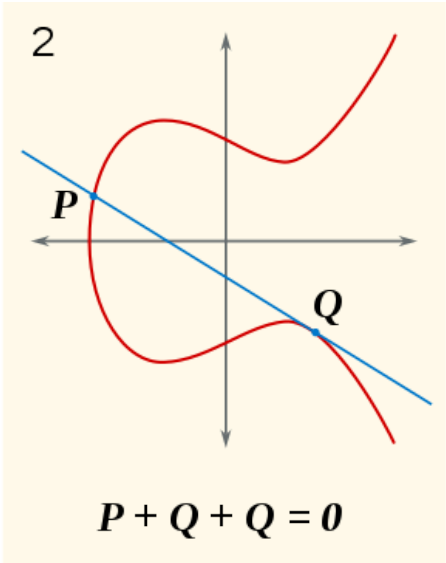
as our Elliptic Curve definition - over $\mathbb{F}_p$ or $\mathbb{F}_{p^k}$.

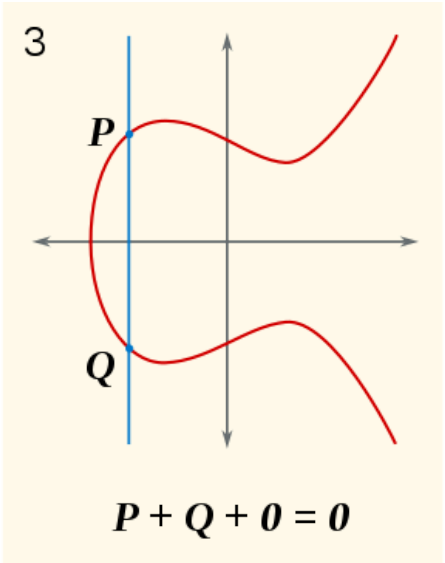
Alongside all (x, y) that satisfy the above, we add a point at infinity ∞.

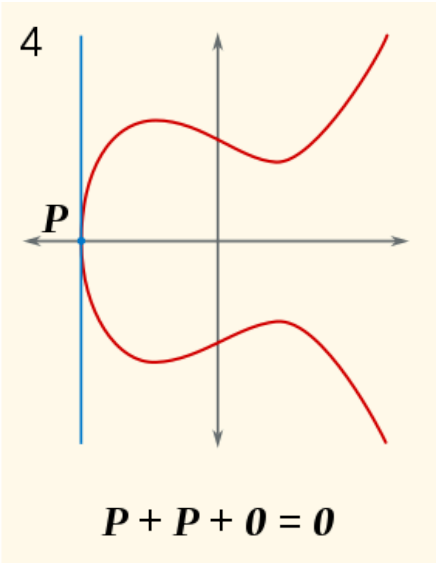
Elliptic Curve as Abelian Group

First, we need to define an addition over E.









Elliptic Curve as Abelian Group

Take $P = (x_p, y_p)$, $Q = (x_q, y_q)$, then

- If $x_p = x_q$ and $y_p + y_q = 0$, then $P + Q = \infty$.
- If $P \neq Q$, then take $\lambda = \frac{y_q - y_p}{x_q - x_p}$. $P + Q = R = (x_r, y_r)$, where $x_r = \lambda^2 - x_p - x_q$ and $y_r = \lambda(x_p - x_r) - y_p$.
- If $P = Q$, then take $\lambda = \frac{3x_p^2 + a}{2y_p}$ then do the same.

There are various ways to show that this is commutative.

Discrete Logarithm

We use ECDSA everywhere, and the fundamental assumption behind it is that the discrete logarithm problem is hard on the elliptic curve.

Consider a private key a and public key aG where G is some point on E. In usual cases, finding a from aG will be difficult. Your private key on ETH is this a, and public key aG is known if you made at least one transaction.

ECDSA is an interesting topic on its own, but let's skip it for now.

Computations over the Elliptic Curve

Computing aG efficiently is a very important problem.

The approaches are vast - some utilize the *structure* of E very nicely, while some put in the honest work into the pieces of the computation.

Idea 1: Efficient $\mathbb{F}_p$ Computation

The simplest idea is to just make $\mathbb{F}_p$ computations faster.

There are three operations: addition, multiplication, division.
Division is the most expensive, but we'll get rid of it. Addition is simple.

How about multiplication?

Idea 1: Efficient $\mathbb{F}_p$ Computation

There are several approaches on $\mathbb{F}_p$ multiplication.

- Barrett Reduction
- Montgomery Reduction
- Hardware Improvement (Ingoyama, etc)
- https://www.ingonyama.com/blogs/fast-modular-multiplication

These algorithms mostly focus on the reduction part post-multiplication.
If you want $ab \pmod p$, you compute ab then reduce it $\pmod p$.

- first part: integer multiplication
- second part: reduction

Idea 2: Removing Division

If you work on Jacobian coordinates, where

$$(X, Y, Z) \rightarrow (X/Z^2, Y/Z^3)$$

You can perform group operations without division.

Idea 3: Standrad Binary Exponentiation

This is the same as computing $a^b \pmod{n}$.

```
let bits = bit_representation(s) # the vector of bits (from LSB to MSB) representing s
let res = 𝒪 # point at infinity
let temp = P # track doubled P val
for bit in bits:
    if bit == 1:
        res = res + temp # point add
    temp = temp + temp # double
return res
```

Idea 4: Binary Exponentiation, but NAF Form

The idea here is that computing $-P$ is very easy. This is comparable to the more familiar $a^b \pmod{n}$ case, where $a^{-1} \pmod{n}$ costs a lot.

One may compute $a = \sum_{i=0}^{t} 2^i b_i$, where $b_i \in \{-1, 0, 1\}$ and no two consecutive b_is are non-zero. This is the **NAF** form.

So for example, for $15P$ we do $16P - P$ instead of $P + 2P + 4P + 8P$.

There are many along this line. Montogery Ladder, w-ary NAF, window...

Hasse's Bound

For security purposes, you want the order of the elliptic curve n to be a large prime. A useful fact on n is the Hasse's Theorem.

Hasse's Theorem

If E is an elliptic curve over finite field $\mathbb{F}_q$, then

$$|q + 1 - \sharp E(\mathbb{F}_q)| \leq 2\sqrt{q}$$

For a proof, consult Washington's book.

n is also efficiently computable in poly-logarithmic time.

Discrete Logarithm as Hidden Addition

Discrete Logarithm Problem (DLP)

For a generic group G with $|G| = p$ a prime, we are given g and $h = g^a$. The DLP is to find a given g, h, and it is a computationally hard problem.

Consider $a, b \in \mathbb{F}_p$ and a generic group G with $|G| = p$ and generator g. We can encode and hide the value a as $E(a) = g^a$ and the value b as $E(b) = g^b$. Then, the value corresponding to $a + b$ is $g^{a+b} = g^a \cdot g^b$. Therefore, we have a method of calculating $E(a + b)$ from $E(a)$ and $E(b)$, while hiding the actual values of a, b. This will be relevant later.

Pairings as Hidden Multiplication

Bilinear Pairings

Let q be a prime and G_1, G_2, G_T be groups of order q.
A pairing is a map $e : G_1 \times G_2 \to G_T$ such that

- $\forall a, b \in \mathbb{F}_q$ and $P \in G_1,\ Q \in G_2,\ e(aP, bQ) = ab \cdot e(P, Q)$
- e is non-trivial, but still efficiently computable

Assume that DLP on each of G_1, G_2, G_T are all hard.
Choose generators $g_1, g_2, g_T \in G_1, G_2, G_T$ such that $e(g_1, g_2) = g_T$.
Denote $E_1(x) = xg_1,\ E_2(x) = xg_2,\ E_T(x) = xg_T$.
Now $e(E_1(a), E_2(b)) = E_T(ab)$, which enables multiplication.

G_1, G_2 will be all elliptic curves, as we'll discuss later.

Table of Contents

Goal

Here, we showcase several interesting research directions that cryptographers have worked on. For more details, consult the relevant paper or work. In this presentation, we will skim over them.

Instantiation: BN254

BN254 is the elliptic curve that is precompiled in ETH. It's

$$y^2 = x^3 + 3$$

over $\mathbb{F}_q$ where $u = 4965661367192848881$ and

$$q = 36u^4 + 36u^3 + 24u^2 + 6u + 1$$
$$r = 36u^4 + 36u^3 + 18u^2 + 6u + 1$$

where r is the group size.

Instantiation: BLS12-381

BLS12-381 is the elliptic curve used in PoS ETH.

$$y^2 = x^3 + 4$$

over $\mathbb{F}_q$ where u = -0xd201000000010000 and

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

where r is the subgroup size.

Frobenius Endomorphism

Let's work on the algebraic closure $\overline{\mathbb{F}_p}$.

Due to the Frobenius Endomorphism, it can be shown that

$$(x, y) \to (x^p, y^p)$$

is a valid endomorphism on E. We denote this π. For example,

$$E(\mathbb{F}_p) = \ker([1] - \pi)$$

With $t = p + 1 - \sharp E(\mathbb{F}_p)$, it can be shown that

$$\pi^2 - [t] \circ \pi + [p] = 0$$

Endomorphisms and GLV

For BLS12-381 and BN254, note that $a = 0$, i.e. j-invariant is 0.
This, alongside with the fact that $q \equiv r \equiv 1 \pmod 6$, makes it possible for an additional efficiently computable homomorphism.

Take λ_1, λ_2 such that $\lambda_1^3 \equiv 1 \pmod r$ and $\lambda_2^3 \equiv 1 \pmod q$. Then,

$$[\lambda_1](x, y) = (\lambda_2 x, y)$$

This makes it very easy to multiply by λ_1.

Endomorphisms and GLV

Therefore, it makes sense to decompose $[m]G$ into

$$[m]G = [c_0]G + [c_1][\lambda_1]G$$

where c_0, c_1 are both small. This is finding

$$m \equiv c_0 + c_1\lambda \pmod{r}$$

with small c_0, c_1 - a lattice reduction can do this.

Weierstrudel: Aztec's BN254 Masterpiece

For BN254, researchers at Aztec has developed an on-chain contract for computing aG efficiently. Their work was so great that it was more gas-efficient than the EIP196 precompile. This was where **Huff** was born.

It uses

- extremely efficient assembly & memory/stack handling
- various forms of coordinates for usefulness
- windowed NAF form with some precomputation
- GLV based multiplication

https://github.com/AztecProtocol/weierstrudel

Multi-Scalar Multiplication

Consider computing the MSM

$$[a_1]G_1 + \cdots + [a_n]G_n$$

If the scalars are λ bits, it would need $\mathcal{O}(\lambda n)$ operations.

Pippenger's algorithm reduces this to

$$\mathcal{O}\left(\frac{\lambda n}{\log n}\right)$$

under usual parameters.

Multi-Scalar Multiplication

Roughly speaking, the algorithm goes along the lines of

- Split into chunks of $\log n$ bits
- Solve each chunks via a nice double counting strategy
- Merge the results of all chunks

https://www.youtube.com/watch?v=Bl5mQA7UL2I

Further Notes on Pairings

Bilinear Pairings

Let r be a prime and G_1, G_2, G_T be groups of order r.
A pairing is a map $e : G_1 \times G_2 \to G_T$ such that

- $\forall a, b \in \mathbb{F}_r$ and $P \in G_1,\ Q \in G_2,\ e(aP, bQ) = ab \cdot e(P, Q)$
- e is non-trivial, but still efficiently computable

Further Notes on Pairings

The questions we can ask are

- What is r? What are G_1, G_2, G_T?
- How is e computed?

The second question is quite a heavy task. We'll focus on the first one.

Keywords for the second question for interested readers:

- Divisors, Principal Divisors, Weil Reciprocity
- Miller Loop, Tate Pairing, Weil Pairing, Optimal Ate Pairing
- https://infossm.github.io/blog/2022/11/22/PairingMaster/
- https://infossm.github.io/blog/2022/12/05/BLSBNMaster/

The Groups

We need a group of order r. To find them, an easy point to start is

$$E[r] = \{P : [r]P = \infty\}$$

the r-torsion. If r is coprime to q, then

$$E[r] \cong \mathbb{Z}_r \times \mathbb{Z}_r$$

In our case E itself will have some points on the r-torsion.

The Groups

To have the entire r-torsion, we need to extend E from $\mathbb{F}_q$ to $\mathbb{F}_{q^k}$.
k is the embedding degree - the minimum k that $q^k \equiv 1 \pmod{r}$.
Here, we would have $E[r] \subset E(\mathbb{F}_{q^k})$ and we're ready for pairings.

In rare case $k = 1$, but this is not the case for us. A low k would be beneficial for fast computation, but would lead to more attacks in the discrete logarithm side. In both BLS12-381 and BN254, $k = 12$.

k가 너무작으면 풀기쉽고 너무크면
오래걸려서 적정량이 12가나왔음

The Groups

Here, G_1 and G_2 are respectively defined as

$$G_1 = E[r] \cap \ker([1] - \pi)$$
$$G_2 = E[r] \cap \ker([q] - \pi)$$

G_1 is in $E(\mathbb{F}_q)$, but G_2 resides in $E(\mathbb{F}_{q^k})$. This means that all G_2 computations will be very costly. To remediate this situation, a *twist* is needed.

Sextic Twist

For BLS12-381, the transformation

$$(x, y) \to (x/v^2, y/v^3)$$

for $v^6 = (1+i)^{-1}$ creates a new elliptic curve

$$E' : \quad y'^2 = x'^3 + 4(1+i)$$

and G_2 is sent to $E'(\mathbb{F}_{p^2})$. Therefore, working over $\mathbb{F}_{p^2}$ suffices.

For BN254, a similar tactic with $v^6 = 9 + i$ works.

Approaches to G_1, G_2

For pairings, there are several helper functions we need, such as

- Hashing to the Elliptic Curve
- Clearing out the Cofactor
- Subgroup Membership Verification

All three are paper worthy challenges that are still being studied.

Hashing to the Curve

Suppose we were to send a $\{0,1\}^\star$ to an elliptic curve point.
We would be able to hash it to a finite field value $\mathbb{F}_p$, but then what?
Since not all x coordinates are valid, some extra care needs to be put in.
This leads to undesirable properties, such as not being constant time.

Hashing to the Curve

- SWU map is a potential solution, but is too slow
- Simplified SWU map is faster, but requires $ab \neq 0$
- WB19 modifies the simplified SWU map so it works on BLS12-381

To modify the SWU map, the key idea is to map to another elliptic curve with $ab \neq 0$ via an isogeny. Velu's formulas can be used here.

Cofactor Clearing

In many cases, the subgroup you want is not the entire $E(\mathbb{F}_q)$. To arrive at the subgroup itself, you need to multiply by the cofactor h. This is yet another case for multiplying a scalar to a point. As with the case of GLV, some endomorphisms and tricks regarding the group structures are used.

Cofactor Clearing

We showcase two cases. First, G_1 for BLS12-381. Recall

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

Defining $n = (1-u)/3$, it can be shown that the group structure is

$$\mathbb{Z}_{3rn} \times \mathbb{Z}_n$$

Therefore, we simply need to multiply $3n = 1 - u$ to clear out everything.

Cofactor Clearing

Second, G_2 for BN254 (no cofactor clearing for G_1).
It can be shown that the cofactor is $q + t - 1$.
With ϕ as the untwist-Frobenius-twist,

$$\phi^2 - [t] \circ \phi + [q] = 0$$

so

$$[q + t - 1]P = [t](\phi(P) + P) - P - \phi^2(P)$$

Subgroup Membership Testing

Refer to https://eprint.iacr.org/2022/348.pdf, the current state of the art.

- derive the equation for membership testing
- compute each side of the equation via endomorphisms

Cycles of Elliptic Curves

In some cases, we want the case where $\sharp E_1(\mathbb{F}_p) = q$ and $\sharp E_2(\mathbb{F}_q) = p$. This is to chain SNARKs recursively - more details will be filled later.

This would mean that $t_1 = p + 1 - q$ and $t_2 = q + 1 - p$. Here,

$$4p - t_1^2 = 4q - t_2^2 = DV^2$$

in the case $D = 3$, the CM method can be used to derive an elliptic curve.

A good example for curve generation: https://github.com/zcash/pasta.

Cycles of Elliptic Curves

- Generate T, V so that $T^2 + 3V^2$ is close to a power of 2.
- If $p = T^2 + 3V^2$ is a prime $1 \pmod 6$, yield it
- Find the possible options for q, then run search for b in $y^2 = x^3 + b$