KZG

1/ Polynomial commitments via Merkle trees

1.1 Recap on polynomials

The polynomial $p(X)$ can be anything. For example: $p(X)$ can be: $6x^5 + 0x^4 + 0x^3 + 0x^2 + 0x^1 - 55x^0$, which can be also expressed as:

$$6x^5 - 55$$

notice that, even in $6x^5 - 55$, there are actually **6 coefficients(系数)** → `6, 0, 0, 0, 0, -55`

> (0's are omitted for brevity, 只是简洁起见, 省略了 0)

There is exactly one and only one polynomial for a specific coefficient set. (对于特定的系数集，唯一对应一个多项式) , 也就是说, 我们的多项式的正式定义为 :

$$p(X) = \sum_{i=0}^{n} c_i x^i \quad = \quad (6 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + \cdots + (-55) \cdot x^0)$$

this basically means:

- multiply the coefficients c_i with x^i
- then add them all together ...

in the end, we will have a polynomial with a degree of n (means, the highest power of x will be n)

1.2 What are polynomial commitments?

To commit to some polynomial $p(X)$, and we will claim that **at the point z, this very polynomial will give the result** y (在 z 这一点上, 这个多项式将给出结果 y) :

$$p(z) = y$$

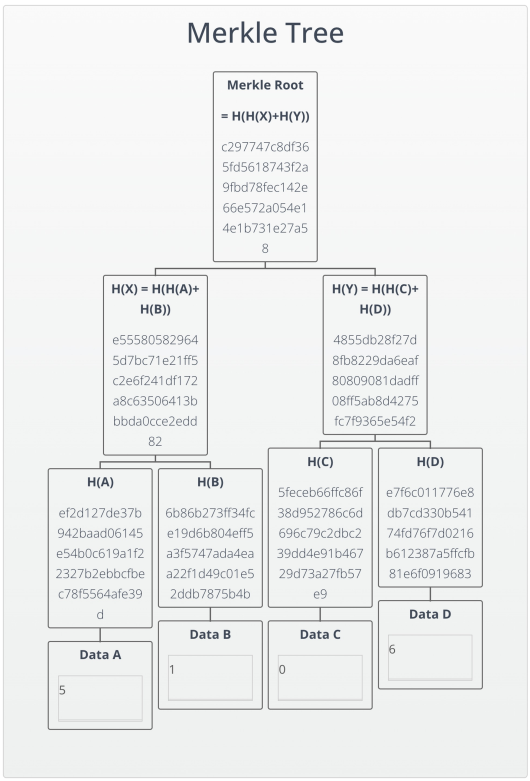
The verifier will request a proof that we did not change the polynomial in order to make $p(z) = y$. In other words, the verifier wants to be sure about the equation $p(z) = y$ will hold only for the very polynomial that is selected in the beginning. (Verifier 会要求 Prover 提供**没有更改多项式的证明**，以便确定是 $p(z) = y$ 而不是 $Cat(dog) = y$。换句话说，验证者想要确定 Prover 确实使用了多项式的系数 和 Verifier 发送的挑战值 z. ($p(z) = y$ 仅适用于开始时选择的多项式。))

To provide a proof to the verifier, we can commit to the polynomial selected in the beginning. How? By calculating the merkle root of the coefficients. (为了向 Verifier 提供这个 Proof，我们可以提交我们的多项式。How? 通过计算系数的默克尔树根。)

1.3 using Merkle Trees

Say our $p(X) = 5x^3 + (1)x^2 + 0x^1 + 6(x^0) = 5x^3 + x^2 + 6$, The coefficients are: `5, 1, 0, 6`

Now, we will build a merkle tree out of these coefficients :



As the first thing, we will send our commitment (the merkle root) to the verifier. Then the verifier will give us a value (say, it will be 2), and ask for the evaluation for this value for our polynomial. (① 我们计算系数的 Merkle root , ② Verifier 发送挑战值 z)

Now, say we are the prover, and our claim is $p(2) = 50$, But there can be multiple polynomial that can hold the above equation, for example: $p'(X) = 25x$. (很多多项式都能满足这个条件. 比如 $p'(X) = 25x$ 、$Cat(dog) == 50$ ...)

We should prove that the equation $p(2) = 50$ holds for the original polynomial, which is $p(X) = 5x^3 + x^2 + 6$.

To prove this, we can send all the coefficients to the verifier `5, 1, 0, 6` and the merkle root: `c297747c8df365fd5618743f2a9fbd78fec142e66e572a054e14e1b731e27a58`

- 这里发送了 2 个东西 : ① 真实的系数(非零知识) ② 系数的 Merkle root

The Verifier will check for **2 things** :

1. given the polynomial coefficients, she will construct the polynomial, and compute the result $p(2)$ herself, and check if Prover's claimed value is matching with what she computed . → $p(2) \overset{?}{=} 50$.
2. if the first check holds, then she will construct the merkle tree for these coefficients herself, and compare our merkle root provided in the proof, with her merkle root (which she computed herself). If they are matching, that means we did not change the polynomial! (如果第一个检查成立，那么她将自己为这些系数构建 merkle 树，并将证明中提供的我们的 merkle 根与她的 merkle 根（她自己计算的）进行比较。如果它们匹配，那就意味着我们没有改变多项式！)

1.4 Summary of it.

Summary of the protocol:

1. prover has a polynomial, he **commits** to it, and sends to the commitment to the verifier
2. verifier has a Challenge value z , sends this to the prover, and asks for an evaluation for this value ($p(z)$)
3. prover evaluates the value, sends the result, and sends the **coefficients** of the polynomial (把系数和盘托出)
4. `Verifier` constructs the polynomial herself:
    1. computes the result for the value, checks if the prover’s evaluation is the same
    2. constructs the merkle tree for the coefficients, checks if the prover’s merkle root is the same

---

Let’s overview the properties of this scheme:

- the commitment was a single element, which is the merkle root (typically 32 bytes 的哈希值)
- the prover had to send all the coefficients (把多项式系数悉数奉上)
    - the whole polynomial is shared with the verifier, **nothing is secret in this scheme** (并不是零知识)

  - the data need to be sent to the verifier is linear: if our polynomial had 1.000.000 coefficients, then we will send 1.000.000 coefficients... In other words, the proof grows as the polynomial grows, this is not very scalable
- the verifier had to reconstruct the polynomial, re-calculate the whole computation (多项式的系数越多越大, 要发送的数据包就越大)
- verifier had to reconstruct the whole merkle tree

It is silly and inefficient. But it will help us understand what polynomial commitments are, and then we will significantly improve it using **Kate Proofs**. (这是愚蠢和低效的。但这将帮助我们理解什么是多项式承诺，然后我们将使用 Kate Proofs 显著改进它。)

What we will achieve with **Kate scheme** :

- commitment size will be 48 bytes (little larger than 32 bytes)
- The proof size, *independent* from the size of the polynomial , is always only one group element. Verification, independent from the size of the polynomial, requires two group multiplications and two pairings, no matter what the degree of the polynomial is. This means, the proof size is constant! UNBELIEVABLE!
  - (`Proof` 和 `Verification` 大小都独立于多项式得, 即和多项式的大小没关系 ; 多项式再大, 它们也是常量)
- The scheme hides the polynomial (practically: yes; theoretically: not 100%, if the polynomial is very simple, it can be guessed. But it shouldn't be simple in the first place hehe)
  - 在实践中可以认为是 100% , 在理论上不是 100%: 如果多项式很简单，可以被猜出来。但它本来就不应该简单, 嘿嘿)

The promises are amazing! Let's try to learn how it works.

What is Commitment

举两个日常生活中的例子。

Lottery (彩票) : 在您能够看到彩票的中奖结果之前，您必须首先确定您选择的号码。这一承诺将使您能够证明在看到结果之前您确实选择了这些数字。这种承诺通常被称为彩票。

Registration and Login : A lot of social applications require you to prove your digital identity to use them. There are two stages;

- **Registration**: This is where you put in your details such as your email address, name, password and phone number. You can **think of this as a commitment** to a **particular identity**.
- **Login**: This is where you use the email address and password from registration to prove that you are the same person. Ideally(理想情况下), only you know these login details.

As you can see, commitment schemes are crucial where one needs to prove something after an event has happened. This analogy also carries over to the cryptographic settings we will consider. (当人们需要在**事件发生后**证明某些东西曾**真实发生过** 时，承诺方案是至关重要的。这个类比也延续到了密码学)

2/ Combining Elliptic Curves and Secure MPC

本部分是对 `椭圆曲线计算` 和 `MPC 多方安全计算` 的说明 ;

2.1 MPC (trusted setup)

In cryptography, we have access to a powerful tool, called secure multi party computation (MPC). It basically allows us to collectively generate a secret (let's denote this with s , and derive some other things from this secret, and in the end (here is the good part), no-one will know the secret!

In order to reveal s, all the participants would have to collude (they cannot reveal the secret even if all of them are compromised, except one **想揭露 s 需要所有人串通, 即使只有一个不参与也不行**) .

> MPC 是指多方计算协议（Multi-Party Computation），它是一种密码学协议，允许多个参与者在不泄露私有数据的情况下进行计算。
>
> 在 MPC 中，每**个参与者都持有一些私有数据，并且希望通过计算得到某些结果**，但是他们不想将自己的私有数据暴露给其他参与者。
>
> MPC 的基本思想是将计算任务分解成多个子任务，并将这些子任务分配给不同的参与者进行计算。每个参与者只能看到自己持有的数据和其他参与者提供的公共信息，而不能看到其他参与者的私有数据。通过使用加密技术和协议来保护隐私和安全性，MPC可以确保所有参与者都能够获得正确的结果，而不会泄露任何私有信息。

2.2 Elliptic Curve intro.

Let $\mathbb{G}$ be the points on our elliptic curve (是椭圆曲线上的点), and G be the generator of $\mathbb{G}$.

A generator is a point on elliptic curve, a base point, which, when added to itself *generates* all other points on the curve. A simple analogy would be 1 for integers: $1+1=2,\ \ 1+1+1=3$, and so on, you can get any integer by adding 1 to itself enough times.

Then, $[x]$ is defined a s → $[x]=xG=\underbrace{G+...+G}_{\text{x times}}$. In other words, $[x]$ converts x to a point in the elliptic curve (it will correspond to one of the points in G).

A good property of elliptic curves is you **cannot derive the value of x from $[x]$**. After multiplying x with G, the value of x is now hidden. You cannot simply divide $[x]$ with G

So, the first thing we will do in our scheme, is to generate a secret number s (from `Setup MPC`), and then compute $[s^0],[s^1],[s^2],...,[s^n]$, (where n is the order(degree) of our `polynomial`), in an MPC setting.

The values $[s^0],[s^1],[s^2],...,[s^n]$ will be **public knowledge** , everyone will have access to these. But remember, it is impossible to deduce s from these , you will **never know the real s .**

What we can do with $[s^0],[s^1],[s^2],...,[s^n]$ → is:

- we can multiply those with a coefficient → $c\left[s^i\right]=cs^iG=\left[cs^i\right]$. Multiplying s with G will allow us to hide s (thanks to elliptic curves) (我们不用再双手奉上多项式的系数了! 现在可以通过和 $[s]$ 的计算将其隐藏起来 !)
- that means, we can actually compute $[p(s)]$.
  - by basically multiplying every coefficient c_i with $\left[s^i\right]$
  - going back to the example, our coefficients were: $6,0,0,0,0,-55$
  - so we should compute $6\left[s^5\right]+0\left[s^4\right]+0\left[s^3\right]+0\left[s^2\right]+0\left[s^1\right]-55\left[s^0\right]$
  - the whole thing can be shown with: $[p(s)]=\left[\sum_{i=0}^n c_is^i\right]=\sum_{i=0}^n c_i\left[s^i\right]$

LOOK :

$$[p(s)]=\left[\sum_{i=0}^n c_is^i\right]=\sum_{i=0}^n c_i\left[s^i\right]$$

This means, we just computed the elliptic curve point equivalent of p(s), (which is $[p(s)]$), without knowing the secret value s ~ _ ~

> supplement ① : we can often see like $g,g^\alpha,g^{(\alpha^2)},...,g^{(\alpha^t)}$,
> just like $\alpha[g],..,\alpha^2[g]==[\alpha],..,[\alpha^2]==[\alpha^i]==[s^i]...$
>
> some other entries use: τ (`tau ceremony trusted setup`), 2^{21} from Zcash's powers of `tau ceremony`

> supplement ② :
> 实际的运算是定义在有限域上的 :

$$\begin{aligned} \text{Commit}\,(p)=p(\alpha) \\ g^{p(\alpha)}= \\ g^{\sum_{i=0}^n \alpha^i c_i}= \\ g^{\alpha^n c_n+...+\alpha^2c_2+\alpha c_1+c_0}= \\ \prod_{i=0}^n(g^{\alpha^i})^{c_i}\end{aligned}$$

2.3 secrets have been revealed

An important vulnerability to be aware of is that if we know α, we can easily break **binding** by finding two polynomials that evaluate to the same point (找到两个计算结果相同的多项式 break `binding`) :

$$\begin{aligned} if\ \ we\ \ know:\ \ \alpha=3 \\ p_1\,(x)=x^3+10\,x^2+8\,x+6 \\ p_2\,(x)=7\,x^2+19\,x+27 \\ g^{p_1(\alpha)}=g^{p_2(\alpha)} \\ g^{\alpha^3+10\,\alpha^2+8\,\alpha+6}=g^{7\,\alpha^2+19\,\alpha+27} \\ g^{147}=g^{147}\end{aligned}$$

Luckily we can rely on the **t-polyDH** assumption (an extension of **q-SDH**) to help us establish **hiding** and **binding** and prevent this vulnerability.

3/ Kate Commitment

> Pronounced /kah-tey/

In KZG polynomial commitment scheme, the prover

1. first needs to commit to the polynomial $p(X)$
   1. (this polynomial will be generated from our data points, for example, our data points can be the coefficients of this polynomial),
2. and then submit a proof π , along with his claim $p(z) = y$.
   1. The point z will be selected by the verifier, and sent to the prover after the prover sends his commitment $C = [p(s)]$, which is a commitment to the polynomial $p(X)$.

Now, the things that prover needs to know/compute:

- $C = [p(s)]$ → The prover knows the polynomial $p(X)$, and we showed above how $[p(s)]$ can be computed via an MPC setting.
- $p(z) = y$ → z is provided to the prover from the verifier. The prover knows the polynomial itself, this is trivial
- π, the proof → we haven't talked about it yet. Let's begin!

3.1 polynomial math (not too much !)

If a polynomial $p(X)$ has a zero at point m ($p(X)$ 在 m 点取值为 0) , that means the polynomial $p(X)$ is divisible by the term $X - m$. Being divisible by the term $X - m$ means we can write the following:

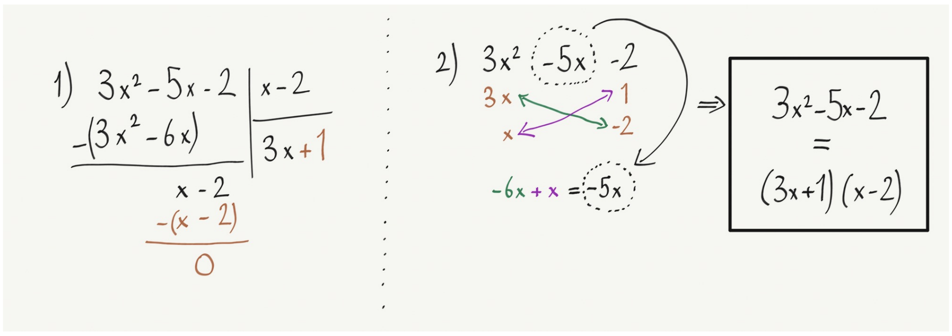
$$p(X) = (X - m) \cdot q(X)$$

for some other polynomial $q(X)$. And this equation clearly evaluates to zero at the point $X = m$

An example:

1. $p(X) = 3x^2 - 5x - 2$ (let this be our `polynomial`)
2. $p(2) = 3(2^2) - 5(2) - 2 = 0$ (and we find a number `"2"` that makes our polynomial evaluate to 0)
3. $p(X) = 3x^2 - 5x - 2 = (X - 2).q(X)$
4. $\frac{3x^2-5x-2}{x-2} = q(X)$
5. $q(X) = 3x + 1$

If you are confused about steps 4 and 5, we can divide polynomials using 2 different methods, one of them is by simple division (1), and the other one is by factorization (因式分解/交叉相乘) (2).



Coming to the **crucial** part (至关重要!) :

1. we want to prove $p(z) = y$
2. we know $p(X) - y$ (对 ↑ 式移项) will be zero at the point $X = z$
3. in other words, we know $(X - z)$ divides $(p(X) - y)$
4. what we have is: $\frac{p(X)-y}{X-z} = q(X)$
5. we can compute the polynomial $q(X)$ now (as shown above).
   1. X can be anything, so let it be s
   2. **we now have the following relationship:** $\frac{p(s)-y}{s-z} = q(s)$
   3. or another way to display this relationship: $p(s) - y = q(s) \cdot (s - z)$

> We cannot work directly with this relationship, because neither the `prover` nor the `verifier` does know the value s. What they know is, $[s^i]$.

3.2 Why convinced?

This relationship itself can be the thing that will convince the verifier.
以上带 $q(s)$ 的步骤已经可以让 `Verifier` 信服 `Prover` 没有改动过原始的 $p(X)$. 想知道 Why 的话, 以下是一些延展阅读 :

we already committed to $p(X)$ via $C = [p(s)]$, this commitment is ensuring we cannot change $p(X)$. Reason (反证法) 如下 :

- Let's assume otherwise. Say for another polynomial $p'(X) \neq p(X)$, we arrive at the same commitment, which means $[p'(s)] = [p(s)]$. And that means $p'(s) - p(s) = 0$. Then, we will be able to cheat by changing $p(X)$ with $p'(X)$. Now, we should prove that **finding $p'(X)$ should be practically impossible**. 这段说的是 : 不可能找到另一个 $[p'(s)] = [p(s)]$
- For two different polynomials, we can write the following: $r(X) = p(X) - p'(X)$.
- We know $r(X)$ is not a `constant polynomial` in this case (a.k.a. it's order is not 0) (aka: Also Known As).
- 任意非常数的 n 次多项式至多可以有 n 个零点，这是因为如果 $r(z) = 0$ ，$r(X)$ 就可以被线性因子 $X - z$ 整除；因为每一个零点都意味着可以被一个线性因子整除，同时每经过一次除法会降低一阶，所以推理可知至多存在 n 个零点[2]。
- 因为 Prover 不知道 s 的真实值 (只知道 s 在椭圆曲线上的运算结果)， 如果想找到 $p'(s) - p(s) = 0$. 只能通过在尽可能多的地方让 $p'(s) - p(s) = 0$ 。如上所证，他们只能在至多 n 个点上使 $p'(s) - p(s) = 0$ ，那么成功的可能性就很小，因为 n 比起椭圆曲线的 degree `p` (大素数) 要小很多，s 恰好被选中成为 $p'(s) = p(s)$ 成立点的概率是微乎其微的。
- 来感受一下这个概率的大小，假设我们采用现有 **largest** trusted setups ，即 $n = 2^{28}$ ，and compare it to the curve order $p \approx 2^{256}$ 对比： 攻击者想设立的多项式 $p'(X)$ 来与 $p(X)$ 最多重合 $n = 228$ 个点，得到相同承诺的概率是 $\frac{2^{28}}{2^{256}} = 2^{28-256} \approx 2 \cdot 10^{-69}$ 。That is an `incredibly low probability` and in practice means the attacker **cannot** pull this off.
- For reference, finding a collision in SHA256 is: $4.3 * 10^{-60}$
- Means, finding $p'(X)$ is around $10^9/2 = 500000000$ times harder than finding a collision in SHA256... Wow!

OK !!!! now that we proved $p(X)$ cannot be changed after being committed on, let's inspect the other parts of this relationship → $\frac{p(X)-y}{X-z} = q(X)$ (上面, 我们用反证法证明了 为什么 $p(X)$ 不能被替换作假)

下面来考虑 $\frac{p(X)-y}{X-z} = q(X)$ 式中的 other parts.

- z cannot be changed, since the verifier herself send it, she would know...
- y cannot be changed. Why?
  - 还是反证法, 假设 y 可以被替换成其他的: The prover wants to cheat with another value $y' \neq y$.
  - recall $p(z) - y = 0$,
  - then, $p(z) - y' \neq 0$ (recall 假设: $y' \neq y$)
  - the prover needs to divide $p(X) - y'$ with $X - z$ in order to find $q(X)$.
  - But the prover cannot divide $p(X) - y'$ with $X - z$, since $p(z) - y' \neq 0$.
  - if the prover can find a fake $q'(X)$, such that $q'(X) = \frac{(p(s)-y')}{s-z}$, then he can have the following relationship:
  - $q'(s)(s-z) = p(s) - y'$,
  - which is equivalent to $(\frac{(p(s)-y')}{s-z})(s-z) = p(s) - y'$,
  - which is equivalent to $p(s) - y' = p(s) - y'$, which will hold :)
  - but this cannot be done, since the attacker has to know about s to compute $q'(X) = \frac{(p(s)-y')}{s-z}$. As long as s is kept secret, we don't need to worry about this.

3.3 Finalizing our proof format for Kate Commitment

Let's recall our relationship:

$$p(s) - y = q(s) \cdot (s - z)$$

And remember, we **cannot work directly** with ↑ relationship, because neither the prover nor the verifier knows s. What they know is, $[s^i]$. And seems like that is enough!

The proof evaluation will work as the following:

$$[q(s)] \cdot [s - z] = \pi \cdot [s - z] = C - [y]$$

$[q(s)]$ is our proof π : 设 $[q(s)]$ 是 Proof π , 记住, 后面要用

What we've done is, we used $[s^i]$ values to turn our equation into points in the elliptic curve.
(我们想使用 $[s^i]$ values 将上式转换为椭圆曲线中的点。

$$\begin{aligned}\pi &= \text{CreateWitness}\,(p, x, y) \\ &= g^{\frac{C-[y]}{[s]-z}}\end{aligned}$$

So if we multiply π with $[s - z]$, and check if that is equal to $C - [y]$, we should be good to go for this scheme! And remember, all these are points on the elliptic curve.

However... there is one issue... we cannot multiply elliptic curve points `: )`

> In KZG commitment scheme, we cannot directly multiply elliptic curve points because the group operation on elliptic curves is not commutative. (不可交换)
>
> This means that the result of multiplying two points on an elliptic curve depends on the order in which the multiplication is performed. (非交换群, 群元素的乘法结果取决于运算顺序)
>
> However, pairings provide a way to compute a bilinear map between two different groups, one of which is typically an elliptic curve group. This bilinear map allows us to perform certain operations that are not possible with elliptic curve point multiplication alone.
> 配对提供了一种计算两个不同组之间双线性映射的方法，其中一个组通常是椭圆曲线组。这个双线性映射允许我们执行某些单独使用椭圆曲线点乘无法实现的操作。
>
> Specifically, in KZG commitment scheme, we use pairings to compute a bilinear map between two different groups: the group of points on an `elliptic curve` and a multiplicative group of finite `field elements`. This allows us to perform polynomial evaluations at specific points by computing pairings between certain elliptic curve points and finite field elements.
> 具体来说，在 KZG 承诺方案中，我们使用配对来计算两个不同组之间的双线性映射：椭圆曲线上的点组和有限域元素的乘法组。这使我们能够通过计算某些椭圆曲线点和有限域元素之间的配对来在特定点执行 polynomial evaluations。
>
> `Chatgpt`：通过以这种方式使用配对，我们可以在不依赖椭圆曲线上的直接点乘的情况下实现高效和安全的承诺。配对通过启用有效的承诺验证和防止某些类型的攻击（例如流氓密钥攻击或颠覆攻击 rogue public key attacks or subversion attacks.）来提供额外的安全保证。

4/ Pairing! (math warning! skip this if you want to treat it as a blackbox)

Pairing is a bilinear mapping.

4.1 What is a bilinear?

Linearity (the property of preserving linear combinations) exists for 1d functions such as $f(r)$, if that function obeys: (Linearity（保留线性组合的特性）存在于某些一元函数，如果该函数服从：）

$$f(ar_1 + br_2) = af(r_1) + bf(r_2)$$

For `2d` functions such as $f(r, s)$, the linearity attribute can exist for one dimension, or the other, or both. (线性性可以存在于一维、另一维或 both) If both, then the function is said to be "bilinear"

$$\begin{gathered} f(ar_1 + br_2, s) = af(r_1, s) + bf(r_2, s) \\ f(r, as_1 + bs_2) = af(r, s_1) + bf(r, s_2) \end{gathered}$$

4.2 What is bilinear mapping?

Elliptic curve pairings, or "pairings" for short (defined by the operator e), are a beautiful yet extremely complicated construction. They enable us to take two points on an elliptic curve (usually in two different groups) and produce a new point in a third and different group
(双线性配对是一个美丽但极其复杂的结构。它们使我们能够在椭圆曲线上取两个 point (通常在两个不同的群中) 并在第三个群中产生一个 new point)

In plain english, we define a function e , that is taking two elements from different space, and producing an element in another third space.
(简单来说，定义一个函数，它从不同的空间中获取两个元素，并在另一个第三空间中生成一个元素。)

双线性可以给我们以下性质(还有很多)：

$$\begin{aligned} e\left(P^a, R\right) &= e(P, R)^a \\ e\left(P, R^b\right) &= e(P, R)^b \\ e\left(P^a, R^b\right) &= e(P, R)^{ab} \\ e\left(P + Q, R\right) &= e\left(P, R\right) e\left(Q, R\right) \\ e\left(P, Q + R\right) &= e\left(P, Q\right) e\left(P, R\right) \end{aligned}$$

Understanding how pairings work is a topic for another day, but here are some resources if you're curious:

1. Exploring Elliptic Curve Pairings
2. BLS12-381 For The Rest Of Us
3. An Introduction to Pairing-Based Cryptography
4. Pairings In Cryptography
5. Bilinear Pairings
6. Pairings for beginners

4.3 About Pairing

This is a pairing: $e(Q, P)$

Pairing is an abstract operation. Abstract, because the definition can vary. There is Tate pairing, Weil pairing, Ate pairing, and so on... Each of them defining the pairing via different operations.
(配对是一种抽象操作。其定义可能会有所不同。 有 Tate 配对、Weil 配对、Ate 配对等等...... 其每一个都通过不同的操作来定义配对。)

However, the format of the input and output, and the properties of the `pairing` is fixed.
(但是， `input` 和 `output` 的格式以及“配对”的属性是固定的。)

- Input: 2 points (P and Q) on 2 subgroups of the same curve ($\mathbb{G}_1$ and $\mathbb{G}_2$). (Input: 同一曲线 2 个不同子群上的 2 个 点 : P and Q)
  - $\mathbb{G}_1$ is a subgroup of points for the elliptic curve, which in the form $y^2 = x^3 + b$, and both x and y are simple integers ($x, y \in F_p$)
  - $\mathbb{G}_2$ is another subgroup of points (points satisfy the same equation as $\mathbb{G}_1$, 点和 $\mathbb{G}_1$ 满足相同的方程), but both x and y are elements of supercharged complex numbers ($x, y \in F_{p^{12}}$). x and y are in the format of $w^{12} - 18 * w^6 + 82 = 0$ (可能涉及 field extension 相关的内容, 使用如此大的有限域提供了额外的安全性，但也增加了计算复杂性。)
- Output: an integer (or a complex number) in the multiplicative group $\mathbb{G}_T$ of order n. (输出：n 阶乘法群 $\mathbb{G}_T$ 中的整数（或复数）)

In a crypto setting pairing should embody the following properties:

- $e(P, Q + R) = e(P, Q) * e(P, R)$
- $e(P + S, Q) = e(P, Q) * e(S, Q)$ (this is actually the same as above, I wanted to include it for verbosity 为了冗余)
- $e(aP, bR) = e(P, R)^{ab} = e(P, bR)^a = e(aP, R)^b = e(bP, aR)$
  - (simply, coefficients are interchangeable, because it is `bilinear`). （简单说，系数是可以互换的，因为它是“双线性的”）。
  - Notice that the coefficients are turning into exponents(指数) here if they move out of the parenthesis (请注意，如果系数移出括号(parenthesis)，则它们会在此处变成指数(exponents)
  - $\mathbb{G}_1$ and $\mathbb{G}_2$ are also can be defined as multiplicative groups.(乘法群). So the property turned into this: $e(P^a, R^b) = e(P, R)^{ab} = e(P, R^b)^a = e(P^b, R)^a = e(P^b, R^a)$. 简而言之，这是一些符号差异，无关紧要。
- $e(P, Q) \neq 1$ (non-degeneracy property , 非退化性)

4.4 Coming back to Kate proofs

Recall: Let $\mathbb{G}$ be the points on our elliptic curve, and G be the generator of $\mathbb{G}$.
Then, $[x]$ is defined as → $[x] = xG$.
In other words, $[x]$ converts x to a point in the elliptic curve (corresponding to one of the points in $\mathbb{G}$).

Now we change it as the following:

- Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two different subgroups of the same elliptic curve
- G be the generator of $\mathbb{G}_1$, and H be the generator of $\mathbb{G}_2$,
  - G 是子群 $\mathbb{G}_1$ 的生成元, H 是子群 $\mathbb{G}_2$ 的生成元
  - These generators are typically chosen randomly during a `Trusted Setup` phase.
  - 这些生成元通常是在 `Trusted Setup` 阶段选择的 ;
- Then, $[x]_1$ is defined, as → $[x]_1 = xG$
- And, $[x]_2$ is defined, as → $[x]_2 = xH$
- Lastly, let’s define our pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is the multiplicative target group

Of course, from the secret value s (we dont know), now **two different sets** will be distributed, one for $[s^i]_1$, and one for $[s^i]_2$

The verifier should now check the equality of:

$$e(\pi,\ [s - z]_2) = e(C - [y]_1,\ H) \qquad ps:\ [q(s)]\ is\ our\ proof\ \pi.$$

which is equivalent to (现在上面的等式两边已被分别映射到 T 空间):

$$[q(s) \cdot (s - z)]_T = [p(s) - y]_T$$

The verifier is able to do the computations :

- $[q(s)]$ is provided to the `Verifier` from the `Prover`
- z is selected by the `Verifier`

- the `Verifier` can compute $[s-z]_2$, since she has access to $[s]_2$, $[s-z]_2 = [s]_2 - [z]_2$
- C 或者说 $P(s)$ is provided to `Verifier` by the `Prover`
- she also knows the value y (sent by the prover)
- H is public (the generator of $\mathbb{G}_2$)
- `pairing function` is public

4.5 Full workflow

The full workflow:

1. using an MPC setup, the secret s is generated, and using this secret value, two sets will be distributed publicly, one for $[s^i]_1$, and one for $[s^i]_2$. The secret s is then discarded forever. (s 被用来生成 $[s^i]_1$ 和 $[s^i]_2$ 后, 会被永久丢弃, 再也不被人所知)
2. the prover selects a polynomial, and commits to it: $C = [p(s)]_1$, using $[s^i]_1$, and sends this to the verifier.
3. verifier asks for the evaluation of the committed polynomial at point z.
4. prover sends the following to the verifier:
    1. π (π is $[q(s)]$)
    2. y
5. the verifier checks the equation: $e(\pi, [s-z]_2) = e(C - [y]_1, H)$
    1. if the equation holds, the verifier accepts the proof
    2. if the equation does not hold, the verifier rejects the proof

We just showed how to prove an evaluation of a polynomial at a single point, using only one element as the proof!

$$\begin{aligned} \text{VerifyEval}(C, \pi, x, y) = \\ e(\pi, g^{[s-z]}) = e(g^{p([s])-y}, g) \\ e(g^{\frac{p([s]-p(z)}{[s]-z}}, g^{[s-z]}) = e(g^{p([s])-y}, g) \\ e(g,g)^{p([s])-p(z)} = e(g,g)^{-y+p([s])} \\ e(g,g)^{-y+p([s])} = e(g,g)^{-y+p([s])} \end{aligned}$$

Simply put, verification boils down (归结) to checking that two target group EC points are equal. By doing this, we can confirm that both sides of this equality are computing the same thing. (简单地说，verification 归结为检查两个目标 group 的椭圆曲线点是否相等。 通过这样做，我们可以确认这个等式的两边都在计算同样的事情。)

5/ Batch-proofs (multi proofs)

We can go even further! Let’s show how to evaluate a polynomial at ANY number of points and prove it, using still ONLY ONE element.

> 到目前为止，我们已经介绍了如何验证一个在单点上求值的多项式。 这本身就很不可思议，但如果我们想证明**一个多项式上的多个点**的评估，我们就必须一次又一次地重复同样的协议 (back and forth)。这显然是没有效率的。为了解决这个问题，我们将研究现有 `KZG10` 技术的扩展，学习如何 "批量 "验证多项式上的点。

Say we want to prove the evaluation of k points: $(z_0, y_0), (z_1, y_1), \ldots, (z_{k-1}, y_{k-1})$.

Using an interpolation polynomial (多项式插值), we can find a polynomial of degree less than k, that goes through all these points. (使用插值多项式，我们可以找到次数小于 k 的一个多项式，它通过了所有的 k points: $(z_0, y_0), (z_1, y_1), \ldots, (z_{k-1}, y_{k-1})$ 。Using Lagrange interpolation (使用拉格朗日插值), we can compute this interpolation polynomial. I know this may sound advanced, and it is. Google and YouTube will be your friends for understanding how Lagrange interpolation works.

这听起来很高级，不要怕，没那么难

The formula(公式) for computing the interpolation polynomial is as follows:

$$I(X) = \sum_{i=0}^{k-1} y^i \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{X - z_j}{z_i - z_j}$$

使用这个公式, 可以构造出一个穿过以上 k points: $(z_0, y_0), (z_1, y_1), \ldots, (z_{k-1}, y_{k-1})$ 所有点的一个多项式, 至于原因的可以自己了解下, 比较简单

Since our original polynomial $p(X)$ is passing through all these points, the polynomial $p(X) - I(X)$ will be zero at each $z_0, z_1, \ldots, z_{k-1}$.
In other words, this polynomial will be divisible by: $(X - z_0), (X - z_1), \ldots, (X - z_{k-1})$. So we define our zero polynomial as:

$$Z(X) = (X - z_0)(X - z_1) \ldots (X - z_{k-1})$$

Using the zero polynomial, we can again establish a similar relationship that we did before:

$$q(X) = \frac{p(X) - I(X)}{Z(X)}$$

We now define the Kate multiproof for the evaluations of the points:

$$(z_0, y_0), (z_1, y_1), \ldots, (z_{k-1}, y_{k-1}) : \pi = [q(s)]_1$$

Notice that our proof is still a single element!

Some differences in this scheme:

- the `Verifier` needs to compute $Z(X)$ and $I(X)$
  - for computing these, she is going to need the k points : $(z_0, y_0), (z_1, y_1), \ldots, (z_{k-1}, y_{k-1})$
- she also will compute $[Z(s)]_2$ and $[I(s)]_1$
  - for computing these, she needs the sets $[s^i]_1$ and $[s^i]_2$

The equation the `Verifier` needs to check is as the following:

$$e(\pi, [Z(s)]_2) = e(C - [I(s)]_1, H)$$

which evaluates into:

$$[q(s) \cdot Z(s)]_T = [p(s) - I(s)]_T$$

And via this, we just proved the evaluation of ANY number of points on our polynomial, again providing a single element as a proof!

6/ Kate as a vector commitment

easy to convert Kate polynomial commitment into a **`vector commitment`** . And via doing this, we will achieve the following :

- instead of sending $\log n$ hashes to prove a single element, we will send only one element!
- our proof will consist of only one element, regardless of the vector size! (我们的证明将只包含一个元素，而不管向量大小！)

Here is how:

Previously, we were creating a polynomial using our data points as the coefficients.
Now, we will do something different:

- say we have the elements: $a_0, a_1, \ldots, a_{n-1}$. And say $p(X)$ is the polynomial that will evaluate to these elements as the following: $p(i) = a_i$.

So, for example, to reach a_2, we will plug in the value 2 to our $p(X)$. We know there always is a polynomial that is passing through the points we like. Again, Lagrange interpolation will help us:

$$\begin{gathered} \frac{p(X) - y}{X - z} = q(X) \qquad (\ \ X\ \ can\ \ be\ \ s \\ p(s) - y = q(s) \cdot (s - z) = \\ [q(s)] \cdot [s - z] = \pi \cdot [s - z] = \quad p(s) - y = C - [y] \\ \pi \cdot [s - z] = C - [y] \\ e(\pi, [s - z]_2) = e(C - [y]_1, H) \\ [q(s) \cdot (s - z)]_T = [p(s) - y]_T \end{gathered}$$

KZG10

FFT ...

Reference:

1. https://blog.subspace.network/kzg-polynomial-commitments-cd64af8ec868
2. https://taoa.io/posts/Understanding-KZG10-Polynomial-Commitments
3. https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html

PLONK : https://vitalik.ca/general/2019/09/22/plonk.html