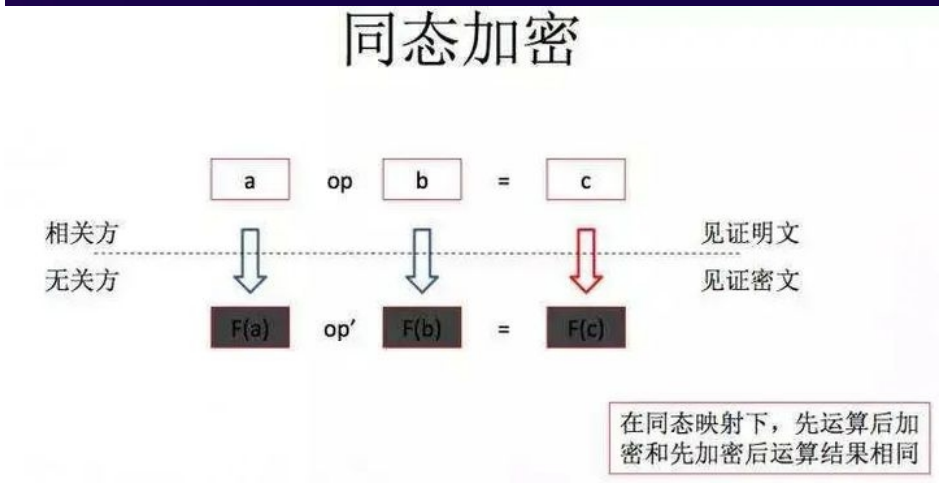
数学之美2022季

区块链数学基础
同态隐藏和双线性配对

深圳市元宇元宙智能科技有限公司

同态加密概述

1. 加法同态，如果存在有效算法 $\oplus$ ，
   $E(x+y)=E(x)\oplus E(y)$或者$x+y=D(E(x)\oplus E(y))$ 成立，并且不泄露x 和y .
2. 乘法同态，如果存在有效算法 ，E(x×y)=E(x) E(y)或者 xy=D(E(x) E(y))成立，
   并且不泄漏 x 和 y。
3. 混合乘法同态，如果存在有效算法 ，E(x×y)=E(x) y 或者 xy=D(E(x) y)成立，
   并且不泄漏 x。
4. 减法同态，如果存在有效算法〇- ，E(x-y)=E(x)〇- E(y)或者 x-y=D(E(x)〇-
   E(y))成立，并且不泄漏 x 和 y，则称 E 为减法同态。
5. 除法同态，如果存在有效算法〇/ ，E(x/y)=E(x)〇/ E(y)或者 x/y=D(E(x)〇/
   E(y))成立，并且不泄漏 x 和 y，则称 E 为减法同态。
6. 代数同态，如果 E 既是加法同态又是乘法同态。
   算术同态，如果 E 同时为加法同态、减法同态、乘法同态和除法同态。



f(x)=a*x，对加法是同态的。很容易验证f(x+y)=f(x)+f(y)
f(x)=x^a，对乘法法是同态的。很容易验证f(x*y)=f(x)*f(y)

部分同态加密算法

指的是该同态加密方案只能做无限次同态加密加法或者只能做无限次同态加密乘法。在对同态加密的研究中，部分同态加密算法是要先于同态加密算法出现的。比较经典的是公钥密码体制中的RSA算法、ElGamal 算法和 Paillier 算法等。

RSA算法对乘法是同态的

用户按照以下步骤生成公私钥：

1. 秘密选取两个大的素数p和q。
2. 计算p和q的乘积：$n = p \times q$。
3. 随机选取一个与$\phi(n)=(p-1)\times(q-1)$[注：欧拉函数性质(3)]互质的数e，应用中通常选取e=65537【注：这是有原因的】。
4. 计算e模$\phi(n)$的模反元素d，也即是计算满足：
   $e\cdot d = 1(mod\phi(n))$ = $e\cdot d$ =1$(mod(p-1)\cdot(q-1))$。【求模逆元可参考历史文章】
5. 将(e,n)作为公开的公钥；d作为私钥，秘密保存。

$$E(x)E(y) \equiv x^e y^e \equiv (xy)^e \equiv E(xy)(mod\, n)$$

（2）A用B的公钥（n,e)进行如下运算得到密文

$C = M^e \ mod \ n$

2. 解密过程

B收到密文C后，做如下运算：

$M = c^d \ mod \ n$ 得到原始消息M。

用文字描述就是：两个加密信息的乘积等于两个信息乘积的加密。

作业：证明RSA算法/写一段RSA实现代码

用户按照以下步骤生成公私钥：

1. 秘密选取两个大的素数p和q。
2. 计算p和q的乘积：$n = p \times q$。
3. 随机选取一个与$\phi(n)=(p-1)\times(q-1)$[注：欧拉函数性质(3)]互质的数e，应用中通常选取e=65537【注：这是有原因的】。
4. 计算e模$\phi(n)$的模反元素d，也即是计算满足：
   $e \cdot d = 1 (mod \phi(n))$ = $e \cdot d$ =1$(mod (p-1)\cdot(q-1))$。【求模逆元可参考历史文章】
5. 将(e,n)作为公开的公钥；d作为私钥，秘密保存。

（2）A用B的公钥（n,e)进行如下运算得到密文

$C = M^e \ mod \ n$

2. 解密过程

B收到密文C后，做如下运算：

$M = c^d \ mod \ n$ 得到原始消息M。

提示：需要用到欧拉定理

ElGamal算法对加法同态

密钥生成

密钥生成的步骤如下：

Alice利用生成元g产生一个q阶循环群G的有效描述。该循环群需要满足一定的安全性质。

Alice从 $\{1,\ldots,q-1\}$ 中随机选择一个 x。

Alice计算 $h := g^x$ 。

Alice公开h以及G,q,g的描述作为其**公钥**，并保留x作为其**私钥**。私钥保密 [1] 。

加密

使用Alice的公钥(G,q,g,h)向她加密一条消息m的加密算法工作方式如下：

Bob从 $\{1,\ldots,q-1\}$ 随机选择一个y，然后计算 $c_1 := g^y$ 。

Bob计算共享秘密 $s := h^y$ 。

Bob把他要发送的秘密消息m映射为G上的一个元素m'。

Bob计算 $c_2 := m' \cdot s$ 。

Bob将密文 $(c_1, c_2) = (g^y, m' \cdot h^y) = (g^y, m' \cdot (g^x)^y)$ 发送给Alice。

解密

利用私钥x对密文 (c_1, c_2) 进行解密的算法工作方式如下 [1] ：

Alice计算共享秘密 $s := {c_1}^x$

然后计算 $m' := c_2 \cdot s^{-1}$ ，并将其映射回明文 m，其中 s^{-1} 是s在群G上的逆元。（例如：如果G是整数模n乘法群的一个子群，那么逆元就是模逆元）。

解密算法是能够正确解密出明文的，因为

$$c_2 \cdot s^{-1} = m' \cdot h^y \cdot (g^{xy})^{-1} = m' \cdot g^{xy} \cdot g^{-xy} = m'.$$

同态隐藏

Here's a toy example of why HH is useful for Zero-Knowledge proofs: Suppose Alice wants to prove to Bob she knows numbers x, y such that $x + y = 7$ (Of course, it's not too exciting knowing such x, y, but this is a good example to explain the concept with).

1. Alice sends $E(x)$ and $E(y)$ to Bob.
2. Bob computes $E(x + y)$ from these values (which he is able to do since E is an HH).
3. Bob also computes $E(7)$, and now checks whether $E(x + y) = E(7)$. He accepts Alice's proof only if equality holds.

$$E(x+y) = g^{x+y \bmod p-1} = g^x \cdot g^y = E(x) \cdot E(y).$$

同态隐藏将在zkSnark中扮演非常重要的角色。

同态隐藏的应用：双盲验证多项式1/4

Alice有一个多项式P

Bob提供一个随机数s

$$P(X) = a_0 + a_1 \cdot X + a_2 \cdot X^2 + \ldots + a_d \cdot X^d$$

1. Bob sends to Alice the hidings $E(1), E(s), \ldots, E(s^d)$.
2. Alice computes $E(P(s))$ from the elements sent in the first step, and sends $E(P(s))$ to Bob. (Alice can do this since E supports linear combinations, and $P(s)$ is a linear combination of $1, s, \ldots, s^d$.)

Bob在不暴露s的前提下，验证了Alice有一个多项式P，但是Bob也没有得到P的任何知识（除了知道是d阶的）

还有一个重要漏洞：Bob得到了E(P(s))，但他无法判断alice是不是骗了他，就是说alice可能发一个假的结果

同态隐藏的应用：双盲验证多项式KCA 2/4

利用ECDLP难题，Bob可以验证Alice具备系数知识

在有限循环群G中，如果$\alpha, \beta \neq 0$且$b = \alpha \cdot a$，那么就称(a, b)为一个α对。

也就是说，b是a的α倍（模p）。有了这个概念，就可以进行一项"系数知识测试"：

- Bob秘密选一个随机的α值，生成一个α对：$(a, b) = (a, \alpha \cdot a)$
- Bob把这个α对(a, b)发送给Alice
- Alice需要回复一个不同的α对：(a', b')
- Bob验证(a', b')是不是一个α对，如果是则接受该回复

同态隐藏的应用：双盲验证多项式d-KCA 3/4

- 假设G是一个有限循环群，g是它的一个生成元
- 选取同态隐藏函数$E(x) = x \cdot g$
- Bob随机选择一个α和一个s，把生成的α对发送给Allice：

  $(a_0, b_0) = (E(1), \alpha \cdot E(1))$

  $(a_1, b_1) = (E(s), \alpha \cdot E(s))$

  …… ……

  $(a_d, b_d) = (E(s^d), \alpha \cdot E(s^d))$

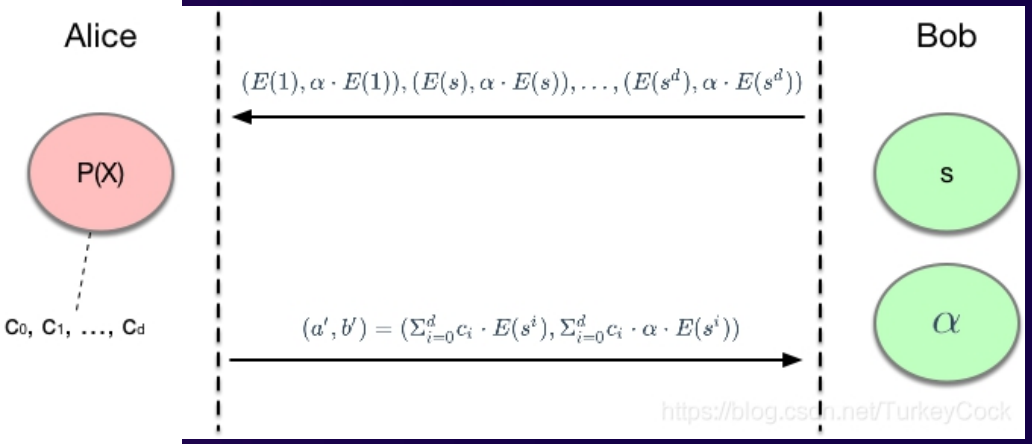
- Alice需要保守的秘密是$P(X)$的系数：$P(X) = c_0 + c_1 \cdot X + \ldots + c_d \cdot X^d$
- Alice计算新的α对：

  $a' = P(s) \cdot g = c_0 \cdot g + c_1 \cdot s \cdot g + \ldots + c_d \cdot s^d \cdot g = c_0 \cdot a_0 + c_1 \cdot a_1 + \ldots + c_d \cdot a_d = \Sigma_{i=0}^{d} c_i \cdot a_i$

  $b' = \alpha \cdot a' = \Sigma_{i=0}^{d} c_i \cdot \alpha \cdot a_i = \Sigma_{i=0}^{d} c_i \cdot b_i$

  然后把(a', b')发送给Bob

- Bob验证(a', b')是否是α对，如果是的话就接受该回复



Prover因为不知道alpha，他只能执行一个d-阶的多项式，才能得到alpha*E(P(s))，因此alpha的作用在于限制了Prover使用的多项式阶，Prover就不可以使用P(x)*任意一个多项式来欺骗V

同态隐藏的应用：双盲验证多项式 4/4
Alice对bob隐藏所有知识

目前为止我们用了：
1、Bob传给alice E(s^i)，而非s，这样保证A无法知道s --- **幂函数的同态隐藏特性**
2、Alice通过回复E(P(s))，E(a*P(s))，证明他确实知道一个多项式，并且满足同态隐藏验证条件，即Bob可以对E(a*P(s)) 验证其是否等于a*E(P(s)) --- **d-KCA**

但是，Bob可以自己构造一个fakeP，通过判断fakeP(s)是否等于P(s)来刺探Alice的部分信息，虽然很有限，但也是不被允许的。

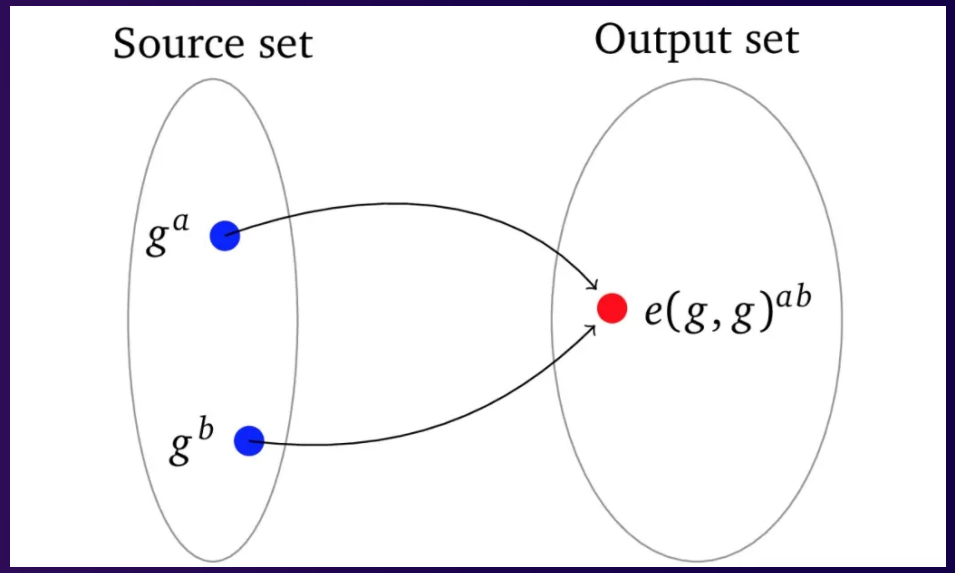
解决方案其实很简单，alice加一个偏置t即可，t是随机数，Alice返回Bob
E(P(s)+t*T(s)), E(a*P(s)+a*t*T(s))，很容易验证E(a*P(s)+a*t*T(s))是E(P(s)+t*T(s))的a 倍数，但bob就无法刺探alice的E(P(s))是否等于他的值，来判断他是否给出一个有效的多项式

最后一个问题，回到了起点

如何判断 E(P(s)) = E(H(s)*T(s) = E(H(s)) * E(T(s)) ?
问题的根源在于 E(x)在我们的定义里，得到的是有限椭圆曲线群的一个点P(x, y)， 直接用两个点相乘，是毫无意义的。也就是说，两个加密结果是无法直接相乘的！！！
由此引出来一个，双线性pairing，来解决这个问题。

双线性配对

配对操作（双线性映射）是一个数学结构，表示为函数 *e(g,g)*，它给定一个数据集中的两个加密的输入（即 g^a, g^b），可以将他们确定性地映射到另一组不同的输出数据集上的它们的乘积，即 $e(g^a, g^b) = e(g, g)^{ab}$



在某种意义上，这个类似于一个哈希函数，他将所有可能的输入值映射到可能的输出值的集合中的一个元素上，通常情况下这个过程是不可逆的。

zkSnark用到的双线性配对的一些良好性质

配对函数 e(g,g) 可以初步（严格来说是不对的）地类比成“交换”每一个输出的基数和指数的操作，使得基数 g 在交换过程中被修改成了指数的方式，即 $g^a \rightarrow a^g$。"被转换"的两个输入一起被修改了，这样原始值 a 和 b 就在同一个指数下相乘了，即：

$$e(g^a, g^b) = a^g \cdot b^g = (ab)^g$$

因而因为基数在“转换”中被修改了，所以在另一个配对中不能再使用这个结果 $(ab)^g$ *（即：$e((ab)^g, g^d*)$）构造出想要的加密乘积 abd 了。配对的核心性质可以表示成下面的等式：

$$e(g^a, g^b) = r(g^b, g^a) = e(g^{ab}, g^1) = e(g^1, g^{ab}) = e(g^1, g^a)^b = e(g^1, g^1)^{ab}$$

严格来讲一个配对的结果是在目标集的一个不同生成元 g 下对原始值乘积的加密，即 $*e(g^a, g^b) = g^{ab}*$。因而它具备同态加密的性质，也就是说我们可以把乘法配对的加密乘积放到一起：

$$e(g^a, g^b) \cdot e(g^c, g^d) = g^{ab} \cdot g^{cd} = g^{ab+cd} = r(g, g)^{ab+cd}$$

> 注意：配对操作是通过改变椭圆曲线来实现这些性质的，现在我们用的符号 g^n 就代表曲线上一个由生成元自相加了 n 次的点，而不是我们前面用到的乘法群生成元。

谢 谢 观 看