

nChain 阅读报告

by Jeffrey shang

声明：github markdown 文件无法支持公式渲染, 安装插件 MathJax Plugin for Github [Click](#) 即可解决.

前置知识：

- 多项式的线性运算
- 拉格朗日插值法
- Shamir门限共享

▼ Shamir 秘密共享 [Click](#)

Shamir 门限共享是理解后面内容的基础, 一定要理解!!!

- 历史

1979年, Adi Shamir 基于多项式插值算法设计了 Shamir(t,n) 门限秘密共享体制;

- 用途

假设我们需要在 n 个用户间共享一个秘密 S . 首先, 我们把秘密 S 进行特定运算, 得到 n 个秘密碎片 S_i ($0 \leq i \leq n$), 交给 n 个人保存, 当至少 t 个人同时拿出自己所拥有的秘密碎片 S_i 时, 即可还原出最初的秘密 S ; 如果只有 $t - 1$ 个秘密碎片 (甚至更少), 则无法得到关于秘密 S 的任何信息.

- 算法

包括两个算法: **秘密份额生成** 和 **秘密重构**

a. 秘密份额生成

首先, 构造一个 $t - 1$ 阶多项式 F , 即: $F = c_{t-1}x^{t-1} + \dots + c_1x + S \pmod p$. 其中, S 为我们想要共享的秘密, p 为素数, 且 $S \in \mathbb{Z}_p$.

取 n 个不相等的 x 值, 如 $x = 1, 2, \dots, n$, 代入到 $F(x)$ 中, 得到 n 组 (x_i, y_i) , $1 \leq i \leq n$, 分配给 n 个人, 同时公开 p , 销毁多项式, 每个人负责保密自己的 (x_i, y_i) .

b. 秘密重构

给定 t 个点, 可以确定一个唯一的 $t - 1$ 阶多项式. 因此, 拥有秘密份额的 n 个人中的任意 t 个人就可以恢复出 $t - 1$ 阶多项式 F ;

当 $x = 0$ 时, 可通过 $F(0) = S$ 恢复出 S .


- 椭圆曲线

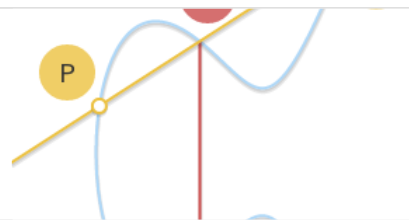
▼ 椭圆曲线资料

椭圆曲线在密码学中的使用是在1985年由 Neal Koblitz 和 Victor Miller 分别独立提出的, 并在2004年至2005年开始广泛应用. 与已有的方案相比, 椭圆曲线密码能够大大缩减所需的密钥长度(实现同等安全强度).

Elliptic Curve Cryptography: a gentle introduction

Those of you who know what public-key cryptography is may have already heard of ECC, ECDH or ECDSA. The first is an acronym for Elliptic Curve Cryptography, the others are names for algorithms based

 <https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/>



ECC椭圆曲线加密算法：介绍


这篇是翻译，因为写的太好了！！已经拿到作者授权，就翻译了一下，供大家参考。如有不同见解，欢迎提出，我们一起讨论，学习就是一个成长的过程啊。现在我们听说过的公钥加密有ECC，ECDH 或者 ECDSA。

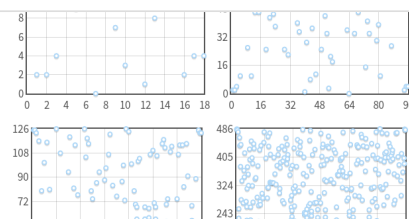
 <https://zhuanlan.zhihu.com/p/36326221>



Elliptic Curve Cryptography: finite fields and discrete logarithms


In the previous post, we have seen how elliptic curves over the real numbers can be used to define a group. Specifically, we have defined a rule for point addition: given three aligned points, their sum is zero ($SP +$

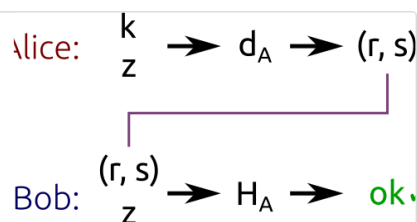
 <https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/>



Elliptic Curve Cryptography: ECDH and ECDSA

In the previous posts, we have seen what an elliptic curve is and we have defined a group law in order to do some math with the points of elliptic curves. Then we have restricted elliptic curves to finite fields of integers

 <https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/>



一. 介绍

本文包含以下内容：

1. 介绍如何在 N 个实体间共享秘密，如何实现两个共享秘密的加法和乘法以及求逆；介绍拥有一般数量的参与者的情况，然后，给出一个只有三个参与者的实例。
2. 研究一个具体的使用案例：门限签名方案；在该方案中，参与者必须合作才能生成消息的签名；同时，介绍拥有一般数量的参与者的情况下的方案，并研究一个实例。
3. 门限签名方案允许在不生成签名私钥的情况下产生合法的数字签名；所谓的“门限非最优”，是指重构秘密所需的秘密份额数量小于生成签名所需的秘密份额数量，因此，当某个秘密份额丢失时，签名的私钥（即共享秘密）仍然可以恢复出来；同时，这种方案计算高效、容易实现；

二. 秘密共享

此章节围绕着以下几点进行介绍：

- 如何创建共享秘密。
- 如何验证多方共享信息的正确性（称为联合可验证随机秘密共享）。
- 如何计算两个秘密的加法，乘积和逆元的秘密份额。

2.1 联合可验证随机秘密共享[1]

这一节里面，介绍“联合可验证随机秘密共享”技术，简称 JVRSS. 其中有以下重点关注的问题：

- 如何创建秘密共享？

理解这一点需要对“Shamir 门限秘密共享”具有深入理解. 前面的“介绍”部分已经强调理解“Shamir 门限共享”的重要性, 从这节开始, 以及后面的每个章节也都躲不开对“Shamir 门限共享”的讨论.

思想解析：

这里的“秘密共享创建”和前置知识中的“Shamir 秘密共享”看似相同, 但其实大不一样. 不同之处就在于: Shamir 门限共享所遵循的逻辑是 先确定共享的秘密 S 的值, 然后根据秘密 S 构建多项式, 将多项式中的 (x, y) 点作为秘密份额 分配给参与者. 而在这里, 创建秘密共享的逻辑则截然相反: 参与者们首先自己选择随机值, 构成多项式, 生成随机的秘密份额, 然后根据生成的秘密份额“表示”共享的秘密 S , 而参与者不知道其确切的值.

换句话说, Shamir 秘密共享是先确定共享秘密, 再对共享秘密进行分割得到秘密份额, 也就是说这个秘密是事先知道的, 并且恢复以后允许各个参与者知道共享秘密的确切值的;

而这里的 创建秘密共享则是 各个参与者先随机生成秘密份额, 在需要使用共享秘密时, 再根据秘密份额构建出“共享秘密的表示形式”, 这里的共享秘密是随机的, 大家都不知道其具体的值, 即使在使用时, 参与者也只知道它的表示形式, 而不知道共享秘密究竟是什么! 即共享秘密是**可用而不可见的**.

下面通过具体例子的数学推导来解释“如何来创建秘密共享”!

首先假设参与者个数 $N = 3$, 门限 $t = 1$, 参与者分别是 p_1, p_2, p_3 , 对于参与者 $p_i, 1 \leq i \leq 3$, 其选择随机数 a_{i0}, a_{i1} , 构造一阶多项式 $f_i(x) = a_{i0} + a_{i1}x$. 那么, 对于三个参与者, 其分别拥有多项式:

$$\begin{aligned}p_1: f_1(x) &= a_{10} + a_{11}x \\p_2: f_2(x) &= a_{20} + a_{21}x \\p_3: f_3(x) &= a_{30} + a_{31}x\end{aligned}$$

然后, 我们把这三个多项式加起来看看会得到些什么?

$$\begin{aligned}f_1(x) + f_2(x) + f_3(x) &= a_{10} + a_{11}x + a_{20} + a_{21}x + a_{30} + a_{31}x \\&= (a_{10} + a_{20} + a_{30}) + (a_{11} + a_{21} + a_{31})x\end{aligned}$$

令 $f(x) = f_1(x) + f_2(x) + f_3(x)$, 则:

$$f(x) = (a_{10} + a_{20} + a_{30}) + (a_{11} + a_{21} + a_{31})x$$

令 $a_0 = a_{10} + a_{20} + a_{30}, a_1 = a_{11} + a_{21} + a_{31}$, 则:

$$f(x) = a_0 + a_1x$$

到现在，我们得到了秘密共享的多项式 $f(x)$ 。而根据 Shamir 秘密共享， a_0 就是我们费尽千辛万苦最终要共享的秘密。

经过上面的推导，我们发现了根据三个参与者所构造的三个多项式构造出最终的秘密的方法。

方法解析：

首先回忆一下“Shamir 门限共享”的知识：

$f(x)$ 是 1 阶多项式，假设我们要恢复秘密 a_0 ，至少需要 2 个份额，也就是多项式上的两个点，比如 $(2, f(2))$ 和 $(3, f(3))$ 。因此，为了不泄露秘密，每个参与者拥有的份额，即 $f(x)$ 的点数，要少于 2 个，即每个参与者有且只有 1 个份额。比如， p_1 拥有份额 $f(1)$ ， p_2 拥有份额 $f(2)$ ， p_3 拥有份额 $f(3)$ 。

那么， p_1 ， p_2 ， p_3 如何得到 $f(1)$ ， $f(2)$ ， $f(3)$ 呢？

根据 $f(x)$ 的表达式，我们反推一下：

$$\begin{aligned} f(x) &= a_0 + a_1x \\ &= (a_{10} + a_{20} + a_{30}) + (a_{11} + a_{21} + a_{31})x \\ &= (a_{10} + a_{11}x) + (a_{20} + a_{21}x) + (a_{30} + a_{31}x) \\ &= f_1(x) + f_2(x) + f_3(x) \end{aligned}$$

所以， $f(x)$ 的值，可以通过计算三个参与者拥有的多项式 $f_1(x)$ ， $f_2(x)$ ， $f_3(x)$ 得到，即：

$$\begin{aligned} f(1) &= f_1(1) + f_2(1) + f_3(1) \\ f(2) &= f_1(2) + f_2(2) + f_3(2) \\ f(3) &= f_1(3) + f_2(3) + f_3(3) \end{aligned}$$

因此，要计算 $f(2)$ 和 $f(3)$ 的值，我们需要计算 $f_1(2)$ ， $f_2(2)$ ， $f_3(2)$ ， $f_1(3)$ ， $f_2(3)$ 和 $f_3(3)$ ，然后求和即可。然而，多项式 $f_1(x)$ ， $f_2(x)$ ， $f_3(x)$ 分别由参与者 p_1 ， p_2 ， p_3 随机生成，任何一个参与者都只拥有其中一个份额，如 $f_1(2)$ 或 $f_2(2)$ 或 $f_3(2)$ 。所以，就需要让各个参与者根据自己的多项式计算，然后再“互通有无”，比如 p_2 和 p_3 分别发送 $f_2(1)$ ， $f_3(1)$ 给 p_1 ，这样 p_1 就拥有了 $f_1(1)$ ， $f_2(1)$ ， $f_3(1)$ ，进而计算出 $f(1) = f_1(1) + f_2(1) + f_3(1)$ 。同理， p_2 ， p_3 也可以算出 $f(2)$ ， $f(3)$ 。

但是为了防止单个参与者恢复出秘密，要遵循的原则是，每个参与者只能获得一个秘密份额，即多项式 $f(x)$ 的一个点。例如， p_1 拥有的点为 $(1, f(1))$ 。因此， p_2 和 p_3 也只能发送 $f_2(1)$ ， $f_3(1)$ 给 p_1 ，不能发送其他任何额外的份额。

现在， p_1 ， p_2 ， p_3 各自拥有多项式 $f(x)$ 的一个份额 $f(1)$ ， $f(2)$ ， $f(3)$ 。其中，任意两个用户即可根据自己的份额联合重构出秘密 a_0 (或者 a)。

• 如何验证多方共享秘密的正确性？

由于每个参与者 p_i 都构造了一个多项式来共享秘密, .

• 如何联合计算 椭圆曲线 (ECC) 的公钥?

假设 ECC 的私钥为 S , 群的生成元为 G , 则公钥为 SG , 那么, 根据 椭圆曲线密码的安全性假设, 知道 G 和 sG , 想要解出 s 在计算上是一件非常困难的事情. 因此, 要想求出公钥, 首先必须计算出私钥, 即共享的秘密 S .

由于私钥 S 由 N 个参与者 p_1, p_2, \dots, p_N 共享, 每个参与者拥有份额 s_i , $\sum_{i=1}^N s_i = S$. 因此, 需要根据拉格朗日插值算法重构出每个参与者 p_i 自己的秘密份额 s_i , 然后计算得到公钥份额 $s_i G$, 最后, 通过 $\sum_{i=1}^{t+1} s_i G$, 就可以得到 SG .

以上创建秘密共享的算法, 记作 $JVRSS(\cdot)$

2.2 共享秘密的加法

假设两个共享秘密多项式为 $f(x) = a_0 + a_1x$ 和 $g(x) = b_0 + b_1x$, 其中共享的秘密分别是 a_0 和 b_0 . 下面来推导多个参与者如何通过联合计算得到两个秘密的和, 即 $a_0 + b_0$.

首先, 令:

$$F(x) = f(x) + g(x) = a_0 + a_1x + b_0 + b_1x = (a_0 + b_0) + (a_1 + b_1)x$$

则对于多项式 $F(x)$, 其共享的秘密恰好为 $a_0 + b_0$.

我们在 2.1 节已经介绍了多项式 $f(x)$ 中的秘密 a_0 是如何重构出来的. 现在, 使用同样的思想: 对于共享秘密多项式 $F(x)$, 要想利用拉格朗日插值法恢复其共享秘密 $a_0 + b_0$, 至少需要两个点, 假设为 $F(2), F(3)$. 根据上式, $F(2) = f(2) + g(2)$, $F(3) = f(3) + g(3)$. 因此, 问题又回到了求 $f(x)$ 和 $g(x)$ 的值上.

每个用户分别计算出:

$$\begin{aligned} p_1 : f(1) &= f_1(1) + f_2(1) + f_3(1) \\ p_1 : g(1) &= g_1(1) + g_2(1) + g_3(1) \\ p_2 : f(2) &= f_1(2) + f_2(2) + f_3(2) \\ p_2 : g(2) &= g_1(2) + g_2(2) + g_3(2) \\ p_3 : f(3) &= f_1(3) + f_2(3) + f_3(3) \\ p_3 : g(3) &= g_1(3) + g_2(3) + g_3(3) \end{aligned}$$

然后, 每个参与者 p_1, p_2, p_3 分别计算出 $F(x)$ 的份额:

$$\begin{aligned} p_1 : F(1) &= f(1) + g(1) \\ p_2 : F(2) &= f(2) + g(2) \\ p_3 : F(3) &= f(3) + g(3) \end{aligned}$$

最终, 任意两个参与者可以联合计算出共享秘密多项式 $F(x)$ 的秘密 $a_0 + b_0$.

以上计算求共享秘密的求和的算法, 记作 $ADDSS(\cdot)$

2.3 共享秘密的乘法

假设两个共享秘密多项式为 $f(x) = a_0 + a_1x$ 和 $g(x) = b_0 + b_1x$, 其中共享的秘密分别是 a_0 和 b_0 . 下面来推导多个参与者如何通过联合计算得到两个秘密的和, 即 $a_0 \cdot b_0$.

首先, 令:

$$G(x) = f(x) \cdot g(x) = (a_0 + a_1x) \cdot (b_0 + b_1x) = (a_0 \cdot b_0) + (a_0 \cdot b_1 + a_1 \cdot b_0)x + a_1 \cdot b_1 \cdot x^2.$$

则对于多项式 $F(x)$, 其共享的秘密恰好为 $a_0 \cdot b_0$.

同理, 对于共享秘密多项式 $G(x)$, 阶数为 $t = 2$, 要想利用拉格朗日插值法恢复其共享秘密 $a_0 \cdot b_0$, 至少需要 $t + 1 = 3$ 个点, 假设为 $G(1), G(2), G(3)$. 根据上式, $G(1) = f(1) \cdot g(1), G(2) = f(2) \cdot g(2), G(3) = f(3) \cdot g(3)$. 因此, 问题又回到了求 $f(x)$ 和 $g(x)$ 的值上.

每个用户分别计算出:

$$\begin{aligned} p_1 : f(1) &= f_1(1) + f_2(1) + f_3(1) \\ p_1 : g(1) &= g_1(1) + g_2(1) + g_3(1) \\ p_2 : f(2) &= f_1(2) + f_2(2) + f_3(2) \\ p_2 : g(2) &= g_1(2) + g_2(2) + g_3(2) \\ p_3 : f(3) &= f_1(3) + f_2(3) + f_3(3) \\ p_3 : g(3) &= g_1(3) + g_2(3) + g_3(3) \end{aligned}$$

然后, 每个参与者 p_1, p_2, p_3 分别计算出 $G(x)$ 的份额:

$$\begin{aligned} p_1 : G(1) &= f(1) \cdot g(1) \\ p_2 : G(2) &= f(2) \cdot g(2) \\ p_3 : G(3) &= f(3) \cdot g(3) \end{aligned}$$

最终, 三个参与者可以联合计算出共享秘密多项式 $G(x)$ 的秘密 $a_0 \cdot b_0$.

以上计算求共享秘密的乘积的算法, 记作 $PROSS(\cdot)$

2.4 求一个共享秘密的逆

求共享秘密的逆时, 所使用的思想与上述“求和”, “求积”, 有所不同. 因为在上述两个运算中, 对两个共享秘密求和可以通过“使用两个共享秘密相对应的秘密份额的和进行拉格朗日插值”来实现; 而对两个共

享秘密求积也可以通过“使用两个共享秘密相对应的秘密份额的积进行拉格朗日插值”来实现;然而,一个共享秘密的逆元却无法通过使用共享秘密的份额的逆元进行拉格朗日插值得到.

求 a^{-1} 的方法如下所示:

2.3 节介绍了如何联合计算两个共享秘密 a, b 的乘积, 这里假设联合计算的秘密 a, b 的乘积为.

$$\mu = a \cdot b$$

则有:

$$\begin{aligned}\mu^{-1} &= (ab)^{-1} \\ \rightarrow a^{-1} &= \mu^{-1} \cdot b\end{aligned}$$

因此, 恢复出秘密 a^{-1} 就等价于恢复秘密 $\mu^{-1} \cdot b$, 其中, μ^{-1} 为各参与方各自提前计算的值, 可以简单地理解为“常数”. 而 b 是一个共享秘密, 各参与方可以根据秘密份额 b_1, b_2, b_3 联合计算得到 (具体方法前面已经讲过). 那么, $\mu^{-1} \cdot b$, 也就是 a^{-1} 就可以通过份额 $\mu^{-1} \cdot b_1, \mu^{-1} \cdot b_2, \mu^{-1} \cdot b_3$ 得到.

以上计算求共享秘密的逆的算法, 记作 $INVSS(\cdot)$

三. 门限签名

首先, 熟悉 ECDSA 的基本构造:

- 签名算法

签名者用于自己的公私钥对 $(a, a \cdot G)$, 给定一个消息 m , 首先计算 m 的 SHA-256 的摘要: $e = SHA - 256(SHA - 256(message))$, 具体签名生成过程如下:

- 选择随机数 $k \in [1, n - 1]$, 其中, n 是 secp256k1 曲线的阶, 也叫做临时私钥.
- 计算 临时私钥 k 对应的临时公钥: $k \cdot G = (x, y)$.
- 计算 $r = x \bmod n$, 如果 $r = 0$, 则回到第一步重新选择 k .
- 计算临时私钥 k 的在 secp256k1 群中的乘法逆元 k^{-1} .
- 计算 $s = k^{-1}(e + ar) \bmod n$, 若 $s = 0$, 就回到第一步重新选择 k .
- 消息 m 对应的签名为 (r, s) .

注意: 签名的临时私钥 k 不能重复使用!!!

- 验证算法

验证者拥有的信息包括: 一个消息 m , 签名私钥所对应的公钥 $(a \cdot G)$, 以及对应的签名 (r, s) , 可以通过以下步骤验证签名:

- 计算消息 m 的摘要 $e = SHA - 256(SHA - 256(message))$.
- 计算 $s^{-1} \bmod n$.

- 计算 $j_1 = es^{-1} \bmod n$ and $j_2 = rs^{-1} \bmod n$.
- 计算 $Q = j_1 \cdot G + j_2(a \cdot G)$
- 如果 Q 为 无穷远点, 则签名无效; 否则, 令 $Q = (x, y)$, 计算 $u = x \bmod n$. 如果 $u = r$, 则签名有效.

以上就是 ECDSA 的签名和验证算法.

3.1 共享密钥生成和验证

为了在 N 个参与者中共享一个私钥 a , 需要 $M = 2t + 1$ 个参与者联合才能生成签名, 需要执行

所有参与者使用 $JVRS$ 算法生成并验证他们共享的签名密钥 a 并计算共享的公钥 $(a \cdot G)$. 在第 2.1 节中描述, 结果是每个参与者都有一个共享私钥的份额 a_i . 以及对应的共享公钥 $(a \cdot G)$.

3.2 临时密钥份额生成

参与者 p_1, p_2, p_3 分别选择临时密钥 k_1, k_2, k_3 , 并分别根据 $INVSS(\cdot)$ 算法计算出 $k_1^{-1}, k_2^{-1}, k_3^{-1}$. 最终根据 $JVRS(\cdot)$ 算法计算出临时密钥 k 和 临时公钥 $(x, y) = k \cdot G$. 同时, 计算 $r = x \bmod n$. 各个参与者各自在本地存储 $(r, k_1^{-1}), (r, k_2^{-1}), (r, k_3^{-1})$.

注: 每次联合签名都要生成新的临时密钥, 否则, 签名私钥 a 可以被攻击者计算出来.

3.3 签名生成

注: 为了协调多个参与方进行联合计算生成给定消息的签名, 在这里引入一个第三方, 称为“协调者”.

通过以下方式, 实现一个 $(3, 3)$ 的门限签名, 即 共有 3 个参与者, 同时至少有 3 个参与者联合计算才能生成签名:

- 首先, 协调者给定一个待签名的消息 m . 当各个参与者 p_1, p_2, p_3 生成新的临时密钥 k_1, k_2, k_3 , 进而生成新的 $(r, k_1^{-1}), (r, k_2^{-1}), (r, k_3^{-1})$.
- 各个参与者计算消息 m 的 $hash$ 值 e , 即, $e = hash(m)$.
- 各个参与者生成签名的份额:

$$\begin{aligned}s_1 &= k_1^{-1}(e + a_1 r) \mod n \\s_2 &= k_2^{-1}(e + a_2 r) \mod n \\s_3 &= k_3^{-1}(e + a_3 r) \mod n\end{aligned}$$

- 各个参与者将 $(r, s_1), (r, s_1), (r, s_1)$ 发送给协调者.
- 协调者根据签名份额 s_1, s_2, s_3 计算出签名 s , 最终得到签名 (r, s) .

四. 参考文献

- [1] Pedersen, T. P. "A threshold cryptosystem without a trusted party." Workshop on the Theory and Application of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1991. [Click](#)
- [2] Gennaro, R, et al. "Robust threshold DSS signatures." International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1996. [Click](#)