多重签名 MultiSig：Schnorr 协议与 ECDSA 协议

2020 年 10 月 21 日

1 摘要

1. 什么是多重签名，多重签名的作用，ECDSA 如何实现多重签名的。
2. 比特币为什么放弃 ECDSA 多签名方案而选择 Schnorr 多签名方案。
3. 比特币的 Schnorr 多重签名的实现与工程应用。

2 什么是多重签名以及其作用

众所周知，一个数字货币所有权的三要素为：地址 (Address)，公钥 (Public Key)，私钥 (Private Key)。地址相当于大家的银行账户；公钥由本人公开，用于加密和验证签名；而私钥用来进行解密和签名，是数字货币所属权的表征，相当于银行卡的密码。

常规的账户或者轻钱包都是一个数字货币地址对应一个私钥，动用这个地址中的资金需要私钥的掌握者发起签名才行。而多重签名（multi-sig）技术，也叫 M-of-N 多重签名，在比特币中，用 N 把钥匙生成一个多重签名的地址，需要其中 M 把钥匙才能花费这个地址上的比特币，$N \geq M$，这就是 M-of-N 的多重签名。比如说，某笔资金对应有 3 个私钥，而必须至少有其中任意 2 个私钥参与签名才能动用，只有 1 个私钥参与签名则是无效的。

在 BitCoin Wiki [2] 中是这样描述的，多重签名是指需要多个密钥来授权比特币交易，而不是一个密钥单个签名。关于多签名的应用介绍，可查看 BitCoin Wiki [2]。

2.1 多重签名的作用

使用多签名有三个作用：**联名账户（多方共同管理资产）、提高安全性、秘钥备份**。

2.1.1 联名账户

通过提供多方的地址，比特币变得“被束缚”，因此需要这些方的合作才能与他们进行任何交易。这些团体可以是人员，机构或程序脚本。

2.1.2 提高安全性

从钱包中花费的私钥可以分布在多台机器上，从而消除了其中任何一台机器的故障，其理由是恶意软件和黑客不太可能感染所有机器。花费资金所需的密钥数量越多（即 M 越接近 N），攻击者成功窃取您的资金就越困难，但是实际使用该钱包的麻烦就越大。

2.1.3 秘钥备份

将多个密钥存储在不同位置的 M-of-N 钱包可以用作备份。例如，在 2-3 的多重签名钱包中，一个密钥的丢失不会导致该钱包的丢失，因为其他两个密钥可用于回收资金。备份的冗余为差值 N 减去 M，因此，例如 3-of-5 多签名钱包（无其他种子备份）的冗余为 2，这意味着仍可从中恢复任何 2 个密钥的丢失。

3 比特币的多重签名

在比特币中，多签名有多种用处：

- 把持有比特币的责任分给多个人，多方共同管理资产。
- 避免单点故障，从而使钱包受到损害的难度大大增加。
- M-of-N 的备份，丢失单个秘钥不会导致钱包丢失。

比特币多签名地址生成方式：前面说过，一般情况下一个地址对应一个私钥，在比特币中哪多个私钥如何对应一个地址呢？Bitcoin Wiki [2] 上介绍了生成一个 2-of-3 的多重签名地址的过程。比特币多重签名地址界面化生成过程可参考 [3]。Go 实现比特币多重签名服务 [4]。

在这里不得不说一下 “P2SH(pay-to-script-hash) 多重签名的脚本 [5]”。这一点暂不详述，网络上有很多参考资料。

4 ECDSA 如何实现数字签名

椭圆曲线数字签名算法（Elliptic Curve Digital Signature Algorithm, ECSDA）生成原理 [6]：

4.1 签名

ECDSA 的签名过程包括基于哈希函数生成消息摘要、椭圆曲线计算和模计算。签名过程的输入包括用比特串表示的任意长度的消息 M、一套有效的椭圆曲线域参数 $T = (p, a, b, G, n, h)$、私钥 d。签名过程的输出是两个整数 (r, s)，其中 $1 \leq r \leq n-1, 1 \leq s \leq n-1$。

1. **密钥生成** 在 $[1, n-1]$ 中选定一个随机的整数 d 作为私钥，计算其对应的公钥 $Q = dG$
2. **消息摘要的生成** 用 SHA-1 哈希函数计算哈希值 $e = H(M)$，得到一个 160 比特的整数
3. **椭圆曲线计算**
   (a) 在区间 $[1, n-1]$ 中选择一个随机整数 k
   (b) 计算椭圆曲线的点 $(x_1, y_1) = kG$
4. **模运算**
   (a) 转换域元素 x_1 到整数 $\overline{x_1}$
   (b) 设置 $r = \overline{x_1} \bmod n$
   (c) 如果 $r = 0$，则转到椭圆曲线计算的步骤 3a 重新选择一个随机整数 k
   (d) 计算 $s = k^{-1}(e + dr) \bmod n$
   (e) 如果 $s = 0$，则转到椭圆曲线计算的步骤 3a 重新选择一个随机整数 k
5. **签名组成** M 的签名就是模运算得到的两个整数：(r, s)

4.2 验签

验签过程由生成消息摘要、模运算、椭圆曲线计算和签名验证组成。验签过程的输入有收到的用比特串表示的消息 M'、收到的该消息的签名 (r', s')、一套有效的椭圆曲线域参数和一个有效的公钥 Q。

1. **消息摘要的生成** 用 SHA-1 哈希函数计算哈希值 $e' = H(M')$，得到一个 160 比特的整数
2. **模运算**
   (a) 如果 r' 或 s' 不是区间 $[1, n-1]$ 内的整数，则拒绝该签名

(b) 计算 $c = (s')^{-1} \bmod n$

(c) 计算 $u_1 = e'c \bmod n$ 和 $u_2 = r'c \bmod n$

3. **椭圆曲线计算** 计算椭圆曲线点 $(x_1, y_1) = u_1G + u_2G$，如果 (x_1, y_1) 是无穷远点，则拒绝改签名

4. **验证**

(a) 转换域元素 x_1 到整数 $\overline{x_1}$

(b) 计算 $v = \overline{x_1} \bmod n$

(c) 如果 $r' = v$，则签名是真实的；否则消息可能已被篡改，或被不正确的签名，或该签名来自攻击者的伪造。该签名应作废

比特币中 ECDSA 使用的椭圆曲线 secp256k1 参数由以下 6 元组 $T = (p, a, b, G, n, h)$ 定义如下：

$$\begin{aligned} p = & \ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF \\ & \ FFFFFFFE\ FFFFFC2F \\ = & \ 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \end{aligned}$$

$\mathbb{F}_p$ 上的曲线 $E : y^2 = x^3 + ax + b$ 定义为

$$\begin{aligned} a &= 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000 \\ b &= 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000007 \end{aligned}$$

压缩格式的基点 G 为

$$G = 02\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798$$

非压缩格式的基点 G 为

$$G = 04\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798\ 483ADA77\ 26A3C465\ 5DA4FBFC\ 0E1108A8\ FD17B448\ A6855419\ 9C47D08F\ FB10D4B8$$

G 的度 n 和 cofactor 为

$$\begin{aligned} n &= FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ BAAEDCE6\ AF48A03B\ BFD25E8C\ D0364141 \\ h &= 01 \end{aligned}$$

在使用 ECDSA 进行多签的情况下，如果共有 N 个私钥进行了签名，则验证时需要对 N 个签名各自进行验证。

5 比特币为什么放弃 ECDSA 多签名而选择 Schnorr 多签名

我们都知道以前比特币协议使用的是 ECDSA 签名方案，最后又提出 Schnorr 多签名方案，这是为什么呢?

ECDSA 签名占据空间大，而 Schnorr 签名占据空间小，且支持多重签名聚合（多个参与方可以基于一个聚合公钥共同产生出一个聚合签名），防篡改。BCH 开发人员 Mark Lundeberg 对聚合签名的看法：聚合签名的优异性是在一笔交易中所有涉及的输入只需要一个合并签名就可以

完成，而用一个签名代替多个签名的好处是显而易见。这导致数据量大幅减少，预计通过 Schnorr 签名将使区块链存储和带宽减少至少 25%，使 BCH 网络更快，更加高效，并且也可以有效的防止垃圾交易攻击（攻击者希望通过尽可能多的占用比特币的交易空间来使比特币拥堵，其手段之一就是通过频繁地从多个来源发送交易使这个交易中包括数十个签名，如果使用 ECDSA ，这些签名就会占据很多的空间。）。Lundeberg 还指出，Schnorr 机制甚至可以避免第二方的篡改行为：即使在聚合的情况下，Schnorr 签名也无法被篡改，除非所有签名者重新创建签名。

与 ECDSA 相比，Schnorr 签名有以下优势：证明安全性、线性、不可延展性、效率高，占用可减少。

5.1 线性

Schnorr 签名是线性的，这一点非常重要。使用 Schnorr 签名的各方可以生成对其各自密钥的签名聚合。以这一特性作为基础，可以构建更高效和隐私性更强的区块链系统。

5.2 不可延展性

ECDSA 签名具有可塑性，不知晓私钥的第三方可以将给定公钥和消息的现有有效签名更改为对同一私钥和消息的另一个有效签名，这一问题直到 SegWit 激活后才得以修复。BIP62 [9] 和 BIP66 [10] 中讨论了此问题。而如果使用 Schnorr 签名则可以避免类似的情况出现。

5.3 安全性证明

在随机预言机模型中，Schnorr 的验证非常简单，ECDSA 不存在这种证明 [7]。

5.4 效率高、占用空间少

链上交易所占空间更少，可以腾出更多存储空间，而且交易的确认时间也大大缩短。

6 Schnorr 多重签名的实现

6.1 符号定义

- x_i 是私钥，与之对应的公钥 X_i（$X_i = x_i \cdot G$，G 是椭圆曲线的基点）
- 待签名的消息为 m
- $H(\cdot)$ 是哈希函数，一般为 SHA256

6.2 Schnorr 签名

参考 这篇文章。

6.3 Naive Schnorr multi-signatures

1. 令 $X = \sum_{i=1}^{n} X_i$
2. 每个签名者选择一个随机数 r_i，且共享给其他所有签名者 $R_i = r_i \cdot G$，令 $R = \sum_{i=1}^{n} R_i$
3. 每位签名者计算 $s_i = r_i + H(X, R, m) \cdot x_i$
4. 最终的签名就是 (R, s)，其中 $s = \sum_{i=1}^{n} s_i$
5. 验证者构造验证等式 $s \cdot G = R + H(X, R, m) \cdot X$ 是否成立

可以发现，上述算法能够满足私钥聚合策略，多个签名联合在一起生成了一个签名，而隐藏了单个签名。但是，该方案并不安全。请考虑一下情形：Alice 和 Bob 想要一起产生多重签名。Alice 有 (x_A, X_A) 密钥对，Bob 有 (x_B, X_B) 密钥对。然而，没有防止 Bob 生成他的公钥为 $X'_B = X_B - X_A$。如果他这样做，其他人将假定 $X'_B + X_A$ 是 Alice 和 Bob 需要合作才能签名的聚合密钥。不幸的是，该总和等于 X_B，而 Bob 可以自己签名。这被称为流氓密钥攻击 (Rogue-key Attack)，一种防御方法是要求 Alice 和 Bob 首先证明他们实际上拥有其声称的公钥相对应的私钥。理想情况下，我们需要构造一种方案，其安全性不依赖于密钥的额外验证。

6.4 Bellare-Neven(BN) Multi-Signature

与 Naive Schnorr multi-signatures 相比，MS-BN [8] 算法是一种安全的多签名算法。仍然以上述方式来解释这个算法：

1. 令 $L = H(X_1, X_2, \dots)$
2. 每个签名者选择一个随机数 r_i，且共享给其他所有签名者 $R_i = r_i \cdot G$，令 $R = \sum_{i=1}^{n} R_i$
3. 每位签名者计算 $s_i = r_i + H(L, X_i, R, m) \cdot x_i$
4. 最终的签名就是 (R, s)，其中 $s = \sum_{i=1}^{n} s_i$
5. 验证者构造验证等式 $s \cdot G = R + \sum_{i=1}^{n} H(L, X_i, R, m) \cdot X_i$ 是否成立

从第 5 步可看出，MS-BN 算法不再具有秘钥聚合性，因为在验证的时候需要整个公钥列表。换句话说，此方案以放弃密钥聚合的属性来获得安全性。

原始的 MS-BN 算法第一步在共享 R_i 之前，要向其他签名者共享 $H(R_i)$。这一步在离散对数 (DL) 假设下安全性证明是必须的步骤。DL 问题 [11] 定义如下：

定义 6.1 (DL 问题)**.** *令 $(\mathbb{G}, p, g)$ 为群参数。如果对于输入的随机群元素 X，一个算法 $\mathcal{A}$ 至多执行 t 次后以至少概率 ϵ 找到使得 $X = g^x$ 的 $x \in \{0, \dots, p-1\}$，则称这个算法能够 (t, ϵ) 解决关于 $\mathbb{G}, p, g$ 的 DL 问题，其中概率的取值方式为随机抽取 X 和 $\mathcal{A}$ 的随机投币实验。*

6.5 Simple Schnorr Multi-Signatures

基于 Schnorr 的聚合签名方案目前有多种实现，Blockstream 给出的方案是 MuSig，各个实现方式的区别以及 MuSig 的具体原理可以参照 [11,12]。Simple Schnorr Multi-Signatures (SSMS) 算法简要过程如下：

1. 令 $L = H(X_1, X_2, \dots)$，$X = \sum_{i=1}^{n} H(L, X_i) X_i$
2. 每个签名者选择一个随机数 r_i，且共享给其他所有签名者 $R_i = r_i \cdot G$，令 $R = \sum_{i=1}^{n} R_i$
3. 每位签名者计算 $s_i = r_i + H(X, R, m) \cdot H(L, X_i) \cdot x_i$
4. 最终的签名就是 (R, s)，其中 $s = \sum_{i=1}^{n} s_i$
5. 验证者构造验证等式 $s \cdot G = R + H(X, R, m) \cdot X_i$ 是否成立

不同之处，在于将 X 定义为不是各个公钥 X_i 的简单和，而是定义成为这些公钥的倍数之和，其中乘数取决于所有参与密钥的哈希。SSMS 在普通的公钥模型中可使用密钥聚合，并具备应有的安全性，其与标准的 Schnorr 签名具有相同的密钥和签名大小。

6.5.1 SSMS 原始算法

以下部分用 `MuSig` 表示 SSMS，这个算法的参数包括一个群 $(\mathbb{G}, p, g)$，其中

- p 是一个 k 比特的整数
- $\mathbb{G}$ 是阶为 p 的循环群
- g 是 $\mathbb{G}$ 生成元

外加均将 $\{0,1\}^*$ 映射到 $\{0,1\}^\ell$ 的三个哈希函数 H_{com}、H_{agg} 和 H_{sig}。值得注意的是对于密码学的群（素数域的乘法子群和椭圆曲线群），群参数生成的正确性可以用论文第 2.2 节的方法高效地验证。

为了后续部分描述的简洁性，我们抛弃引言部分使用的 $\langle L \rangle$ 符号：当一个公钥集合（我们当前场景下是群元素）$L = \{pk_1 = X_1, \ldots, pk_n = X_n\}$ 输入到哈希函数，我们假设它会首先被用唯一的方式编码，例如，按字典序排列。

密钥生成 每个签名方生成一个随机私钥 $x \leftarrow_\$ \mathbb{Z}_p$，并计算器对应的公钥 $X = g^x$

签名 令 X_1 和 x_1 表示特定签名方的公钥和私钥，m 为待签名消息，$X_2, \ldots, X_n$ 是其他协同的签名方，$L = \{X_1, \ldots, X_n\}$ 是签名过程设计的公钥集合。对于 $i \in \{1, \ldots, n\}$，签名方计算

$$a_i = H_{agg}(L, X_i)$$

然后是“聚合”的公钥 $\tilde{X} = \prod_{i=1}^n X_i^{a_i}$。接下来，签名方随机生成 $r_1 \leftarrow_\$ \mathbb{Z}_p$，计算 $R_1 = g^{r_1}$，$t_1 = H_{com}(R_1)$，并把 t_1 发送给其他所有签名方。一旦收到其他签名方的承诺 $t_2, \ldots, t_n$，他/她发送 R_1。一旦收到其他签名方的 $R_2, \ldots, R_n$，他/她会检查 $t_i = H_{com}(R_i)$，其中 $i \in \{2, \ldots, n\}$，如果发现其中某一等式不成立，则终止执行协议；否则继续往下，计算

$$R = \prod_{i=1}^n R_i$$
$$c = H_{sig}(\tilde{X}, R, m)$$
$$s_1 = r_1 + c \cdot a_1 \cdot x_1 \bmod p$$

把 s_1 发送给所有其他签名方。最后，收到其他签名方的 $s_2, \ldots, s_n$ 后，签名方计算 $s = \sum_{i=1}^n s_i \bmod p$。最终签名为 $\sigma = (R, s)$。

验签 给定公钥集合 $L = \{X_1, \ldots, X_n\}$，消息 m 和签名 $\sigma = (R, s)$，验证者为 $i \in \{1, \ldots, n\}$ 计算

$$a_i = H_{agg}(L, X_i) \quad \Rightarrow \tilde{X} = \prod_{i=1}^n X_i^{a_i} \quad \Rightarrow c = H_{sig}(\tilde{X}, R, m)$$

如果 $g^s = R \prod_{i=1}^n X_i^{a_i \cdot c} = R \cdot \tilde{X}^c$，则签名合法

算法正确性的验证比较明显。可以看到验证过程和标准的 Schnorr 签名（在把公钥包含到哈希函数调用方面）类似，这里使用的是“聚合”的公钥 $\tilde{X} = \prod_{i=1}^n X_i^{a_i}$。论文的第 4.3 节介绍了这个方案的（安全）变种。

参考文献

[1] 多重签名 MultiSig：Schnorr 协议与 ECDSA 协议

[2] BitCoin Wiki

[3] 比特币多重签名机制使用篇

[4] Go 实现比特币多重签名服务

[5] P2SH(pay-to-script-hash) 多重签名的脚本

[6] 应用密码学（第四版），胡向东、魏琴芳，胡蓉编著。

[7] Seurin Y. On the exact security of schnorr-type signatures in the random oracle model[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2012: 554-571.

[8] Bellare M , Neven G . Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma[J]. 2006:390.

[9] https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki

[10] https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki

[11] Simple Schnorr Multi-Signatures with Applications to Bitcoin

[12] The MuSig Schnorr Signature Scheme