

# 精通比特币

- 第四章：密钥、地址、钱包
- 第五章：交易

# 密钥、地址、钱包

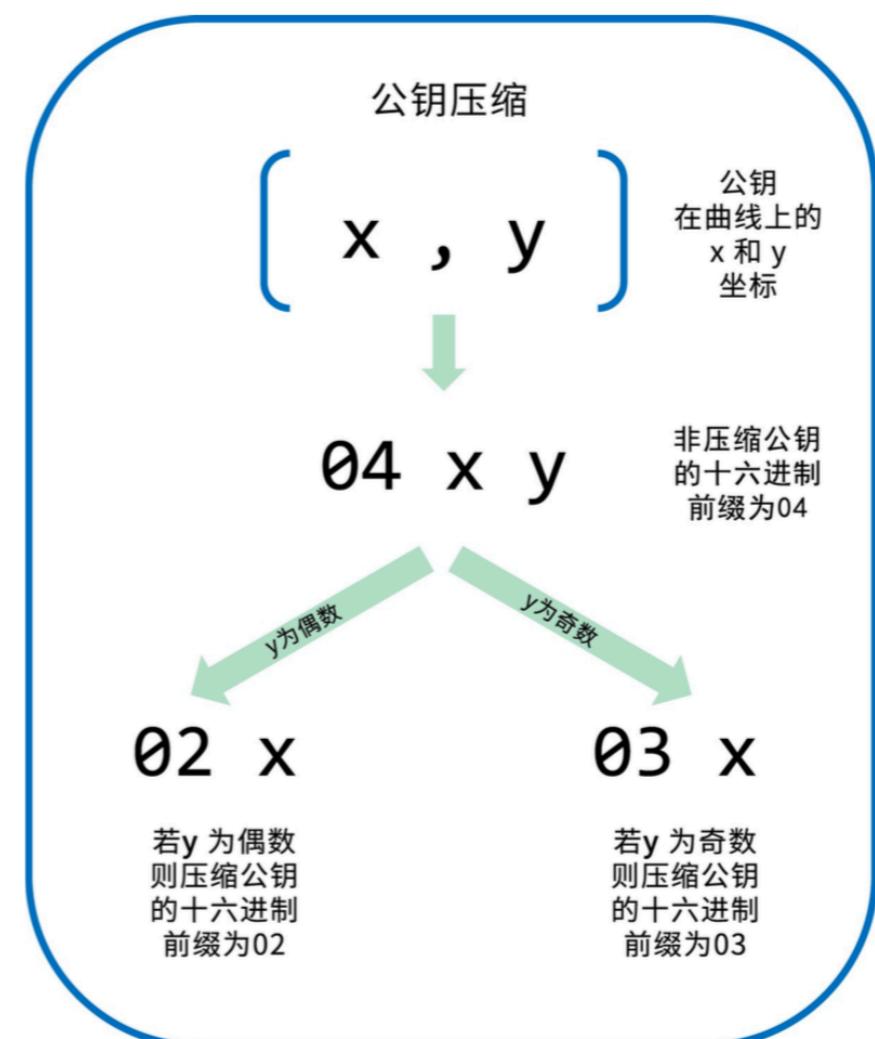
# 私钥

- 值域：256 位， $1 \sim 1.158 * 10^{77}$  之间的任意数字
- 随机性：使用操作系统底层随机数生成器、掷骰子
- 表示方法：原始十六进制、钱包导入格式、压缩钱包导入格式
- BIP-38 加密：密码 + 私钥 WIF-compressed 格式，使用 AES 加密以及 Base58Check 编码，以 6P 开头

| 种类                          | 版本    | 描述  |
|-----------------------------|-------|---|
| 16 进制                       | 无     | 64 个 16 进制数                                     |
| WIF<br>钱包导入格式               | 5     | Base58Check 编码：带有 128 前缀与<br>32 位校验和的 Base58 编码 |
| WIF-compressed<br>压缩式钱包导入格式 | K 或 L | 同上，并且在私钥编码前添加后缀 0x01                            |

# 公钥

- 非压缩式公钥：520 位；压缩式公钥：264 位
- $K(\text{公钥}) = k(\text{私钥}) * G(\text{椭圆曲线生成点})$
- 问题：同一私钥可以对应两种类型的公钥，而公钥与比特币地址一一对应
- 解决方案：使用不同的 WIF 私钥



# secp256k1 椭圆曲线

- 方程:  $y^2 = x^3 + 7$  over  $\mathbb{F}_p$
- $\mathbb{F}_p$  表示素数阶  $p$  的有限域,  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 = 115792089237316195423570985008687907853269984665640564039457584007908834671663$
- 生成点 G: 压缩格式 – 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
- 椭圆曲线的阶:  
115792089237316195423570985008687907852837564279074904382 605163141518161494337

# 有限域

- 有限域的个数只能是质数或者质数的幂次方
- 满足数学算术定理：交换律、结合律
- $a + b = (a + b) \bmod p$ 、 $a - b = (a - b) \bmod p$ 、 $a * b = (a * b) \bmod p$

$$a / b = (a * b ^ {(p - 2)}) \bmod p, a ^ {(p - 1)} = 1$$

$$2 + 3 = 5 \% 7 = 5$$

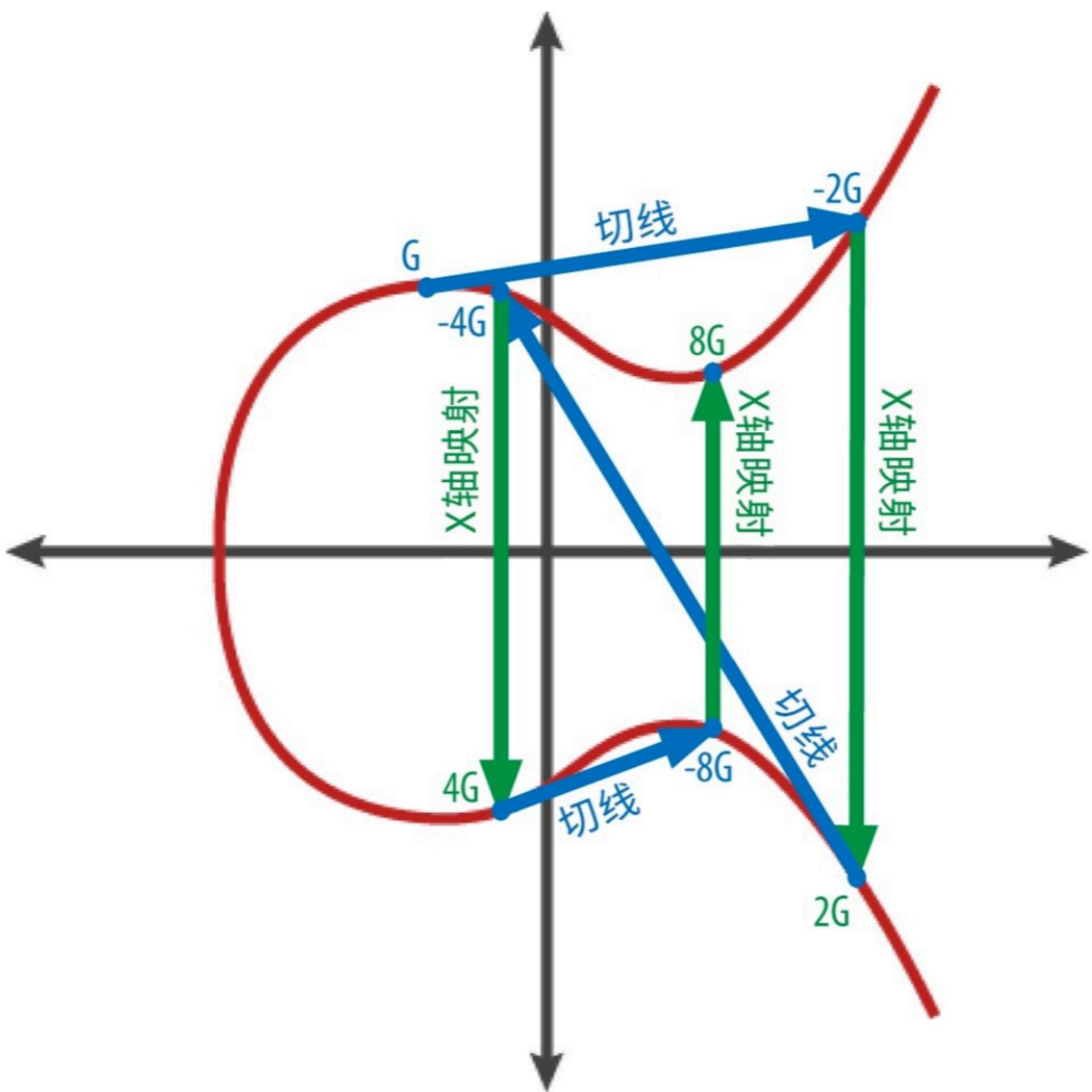
$$2 - 5 = -3 \% 7 = 4$$

$$6 * 3 = 18 \% 7 = 4$$

$$3 / 2 = 3 * 2^5 = 96 \% 7 = 5$$

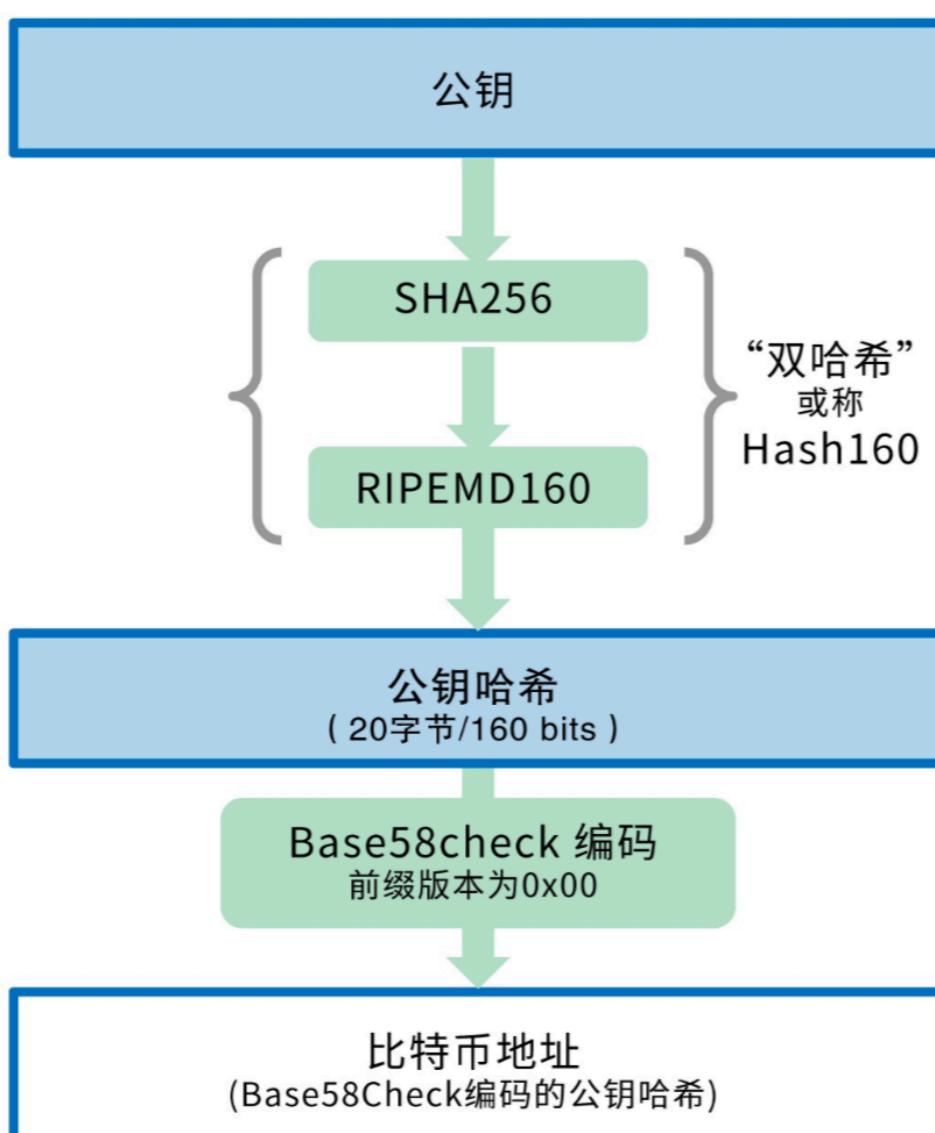
$$5 * 2 = 10 \% 7 = 3$$

$$5 ^ 6 = 15625 \% 7 = 1$$

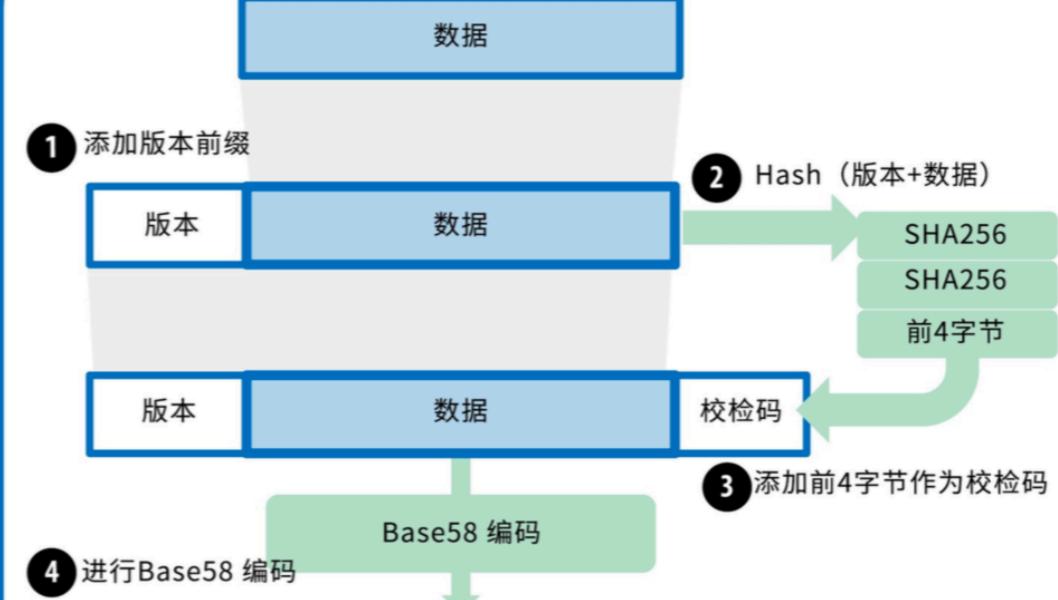


# 地址

从公钥到比特币地址



从公钥到比特币地址



| 种类     | 版本前缀 | Base58 格式 |
|--------|------|-----------|
| 比特币地址  | 0x00 | 1         |
| 脚本哈希地址 | 0x05 | 3         |

# 靓号地址

- 以特定字符开头的比特币地址，识别度高
- 既能加强安全性，又能削弱安全性

| Length | Pattern      | Frequency            | Average search time |
|--------|--------------|----------------------|---------------------|
| 1      | 1K           | 1 in 58 keys         | < 1 milliseconds    |
| 2      | 1Ki          | 1 in 3,364           | 50 milliseconds     |
| 3      | 1Kid         | 1 in 195,000         | < 2 seconds         |
| 4      | 1Kids        | 1 in 11 million      | 1 minute            |
| 5      | 1KidsC       | 1 in 656 million     | 1 hour              |
| 6      | 1KidsCh      | 1 in 38 billion      | 2 days              |
| 7      | 1KidsCha     | 1 in 2.2 trillion    | 3–4 months          |
| 8      | 1KidsChar    | 1 in 128 trillion    | 13–18 years         |
| 9      | 1KidsChari   | 1 in 7 quadrillion   | 800 years           |
| 10     | 1KidsCharit  | 1 in 400 quadrillion | 46,000 years        |
| 11     | 1KidsCharity | 1 in 23 quintillion  | 2.5 million years   |

# 纸钱包

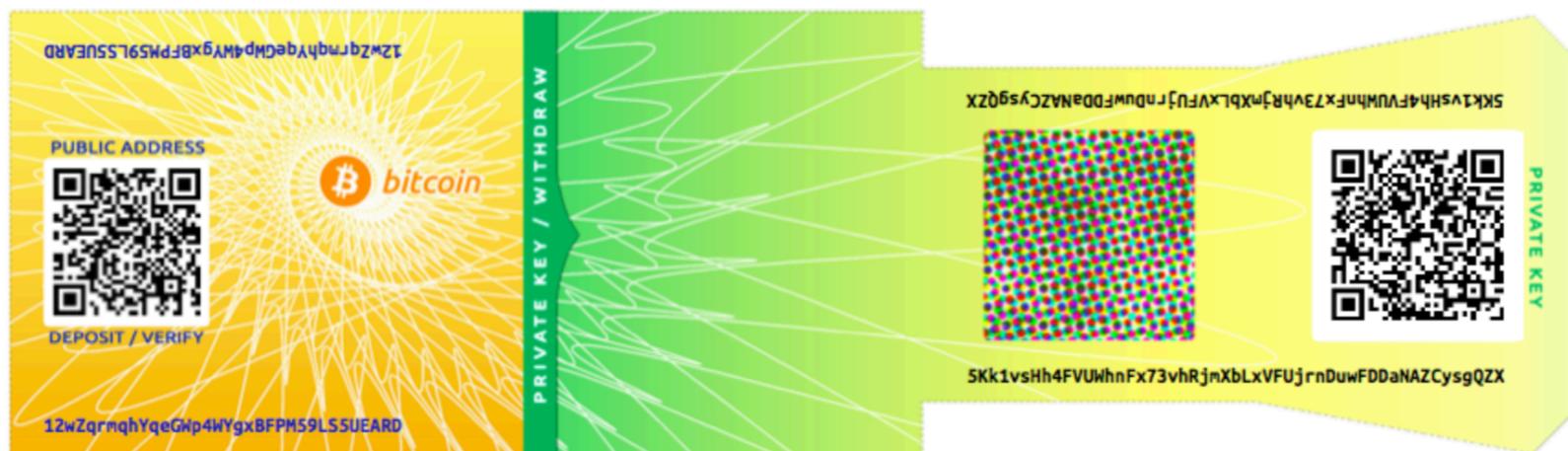
- 将私钥打印在纸上，冷存储/离线存储



明文存储



加密存储



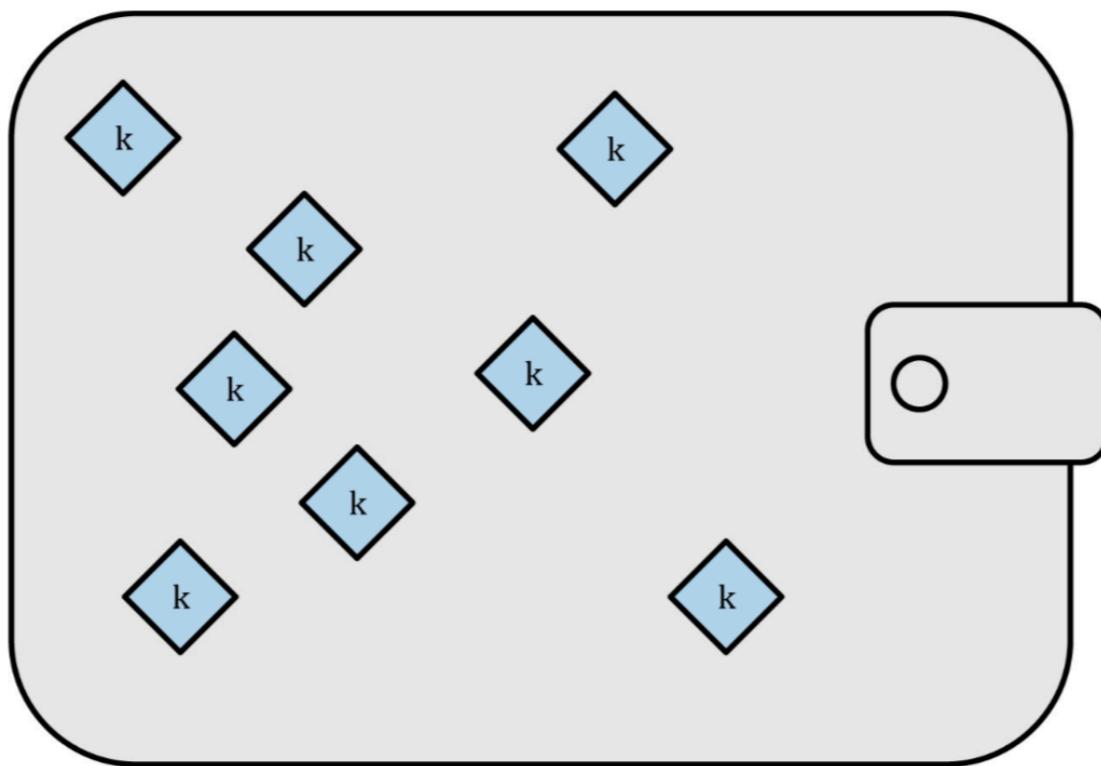
私钥位于折叠处

# 比特币钱包

- 非确定性（随机）钱包
- 确定性（种子）钱包

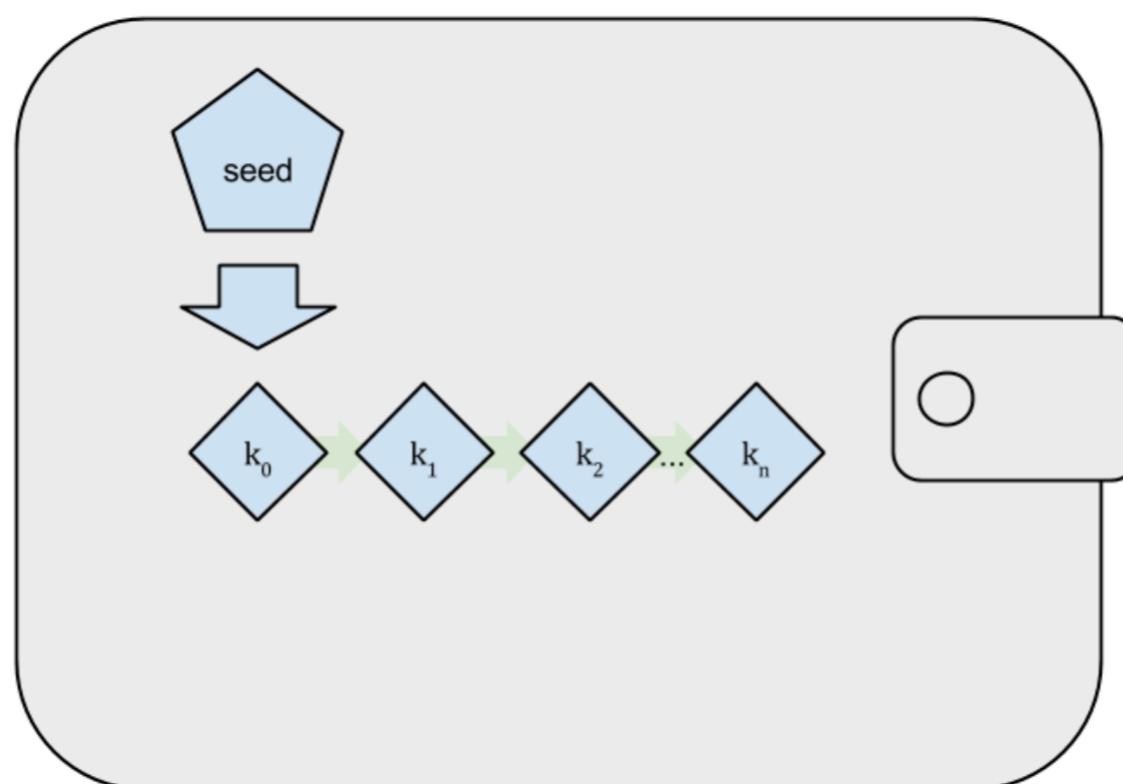
# 非确定性钱包

- 随机生成 100 个私钥，每个私钥只使用一次
- 优点：足够安全，每个比特币地址只使用一次
- 缺点：难以管理、备份、导入



# 确定性（种子）钱包

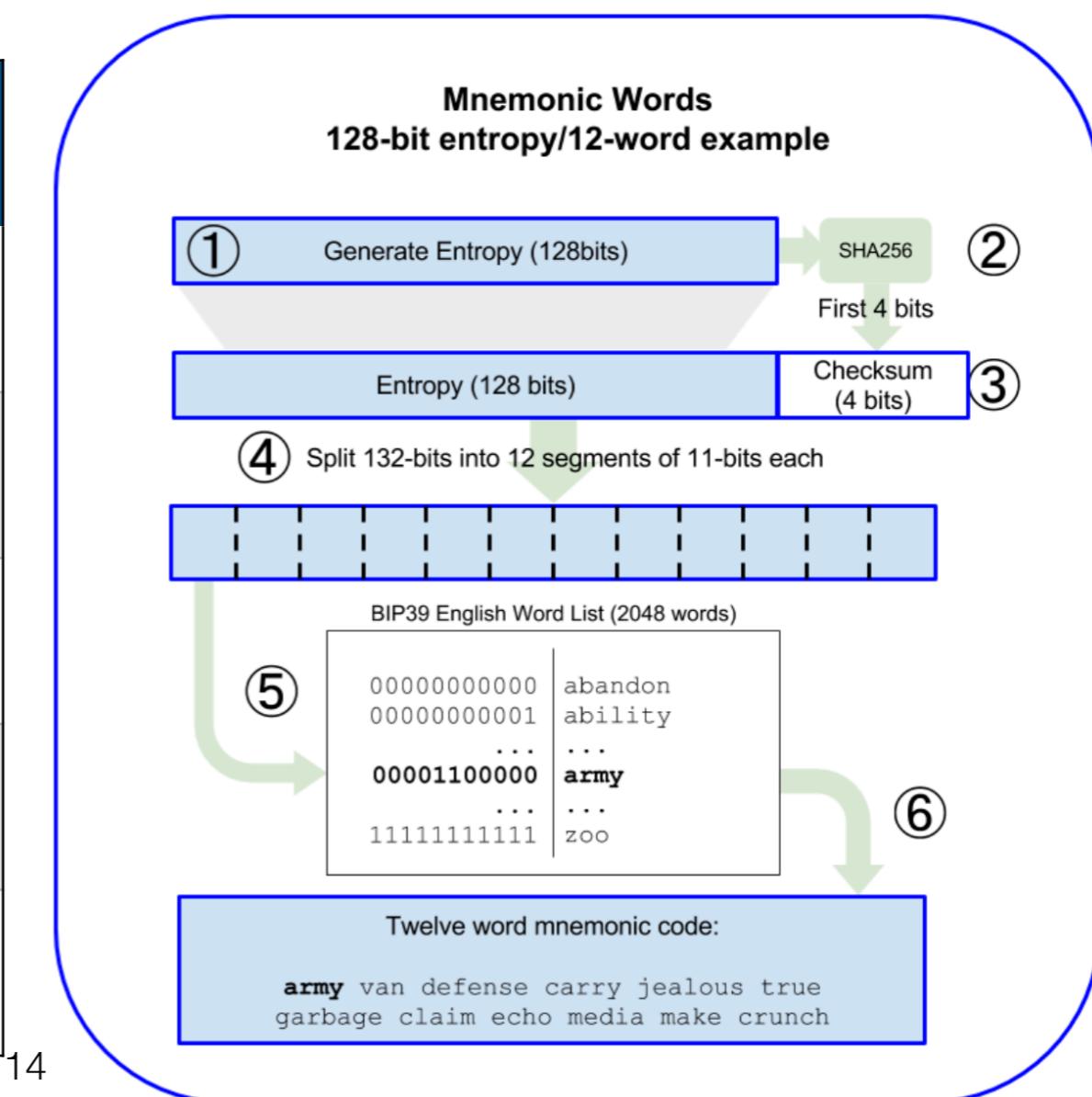
- 所有私钥从同一种子衍生而来 (BIP-32/BIP-44 HD 钱包)
- 种子表示方式：助记词、随机序列
- 优点：只需备份种子，可以恢复出所有私钥



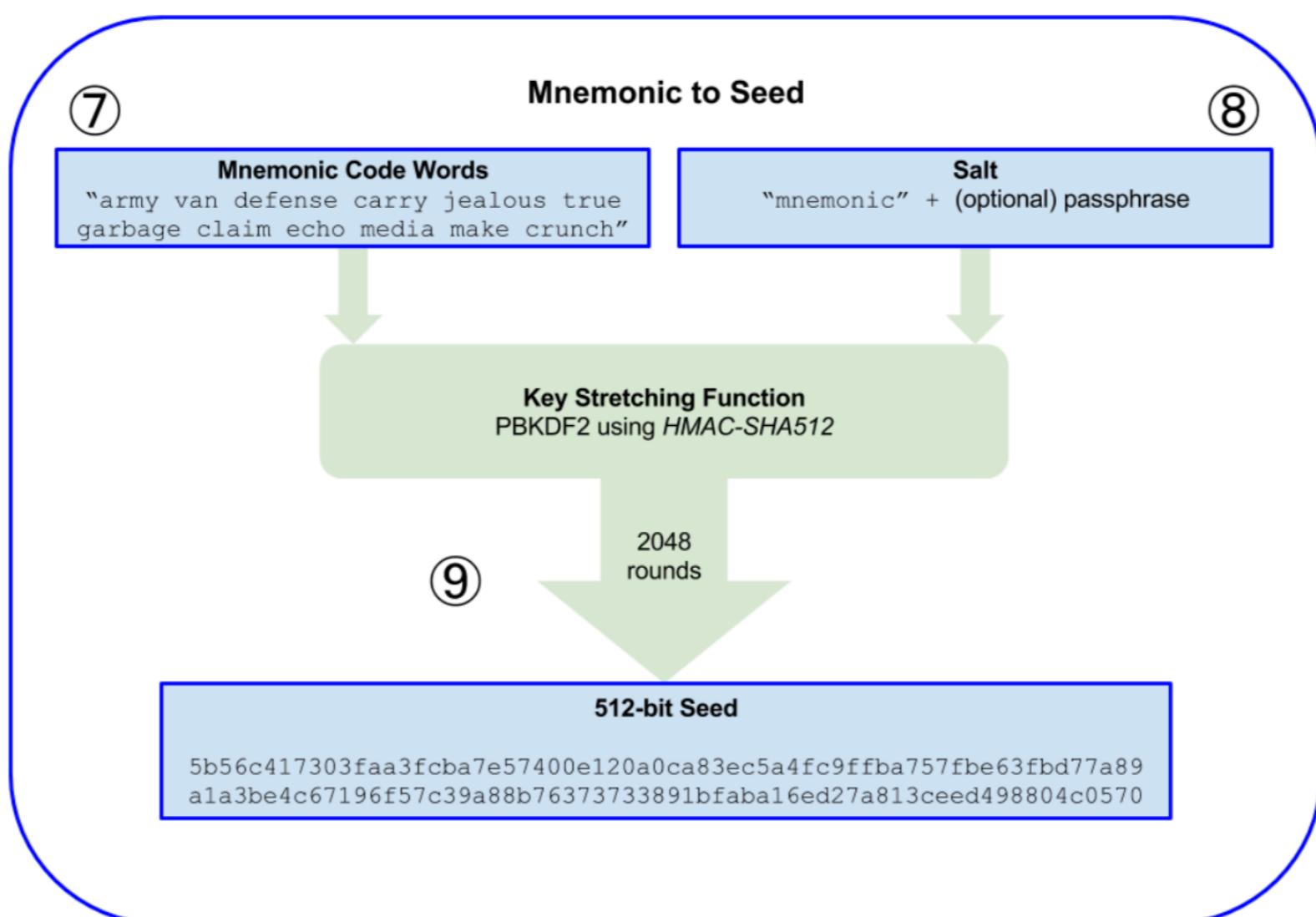
# 助记词

- BIP-39: 使用英文单词序列代表随机选取的熵，然后对熵使用密钥伸展函数 PBKDF2 得到确定性钱包的种子（512 位）

| 熵(bits) | 校验符(bits) | 熵+校验符 | 助记码长 |
|---------|-----------|-------|------|
| 128     | 4         | 132   | 12   |
| 160     | 5         | 165   | 15   |
| 192     | 6         | 198   | 18   |
| 224     | 7         | 231   | 21   |
| 256     | 8         | 264   | 24   |

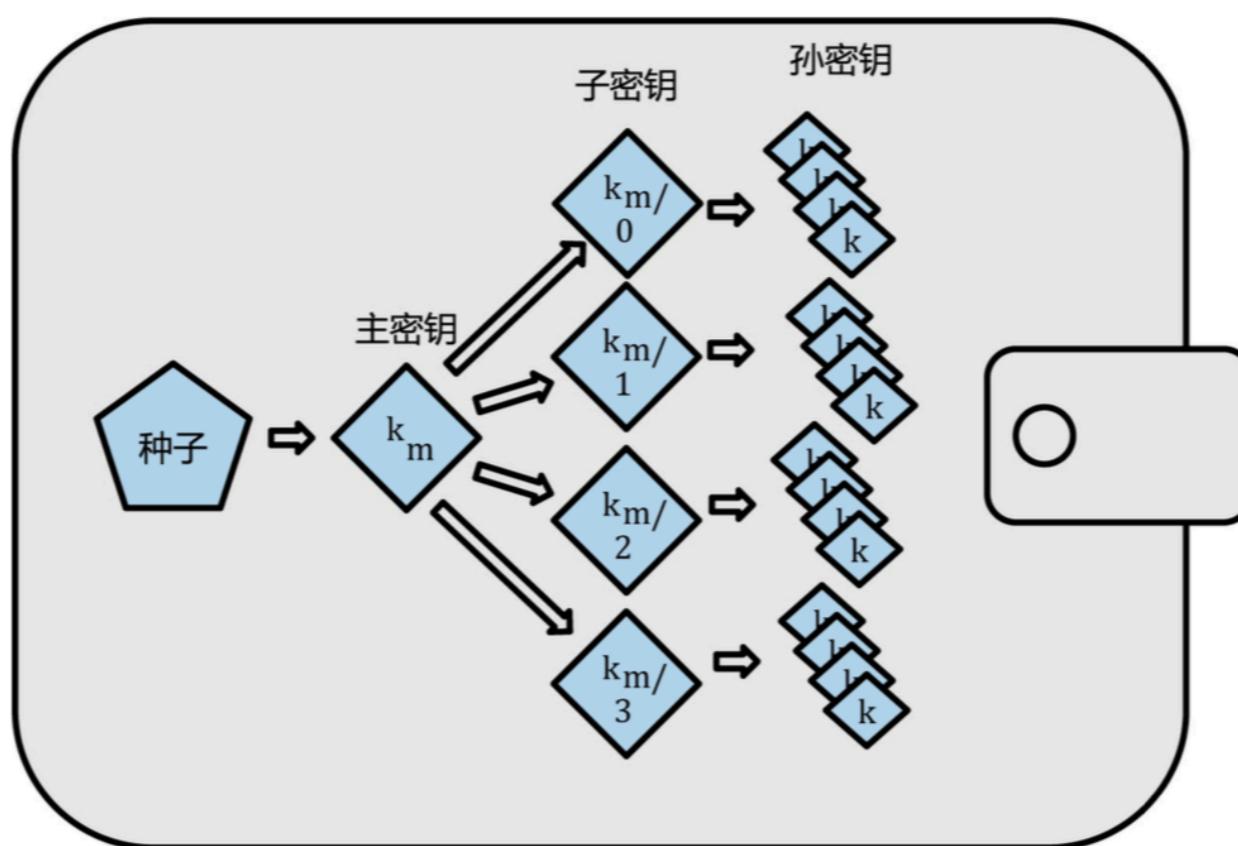


- 密钥伸展函数需要两个参数：助记词、盐值
- 盐值目的：a. 使建立一个查找表变得非常困难，防止暴力破解；b. 允许使用一个密码来对种子进行二次保护



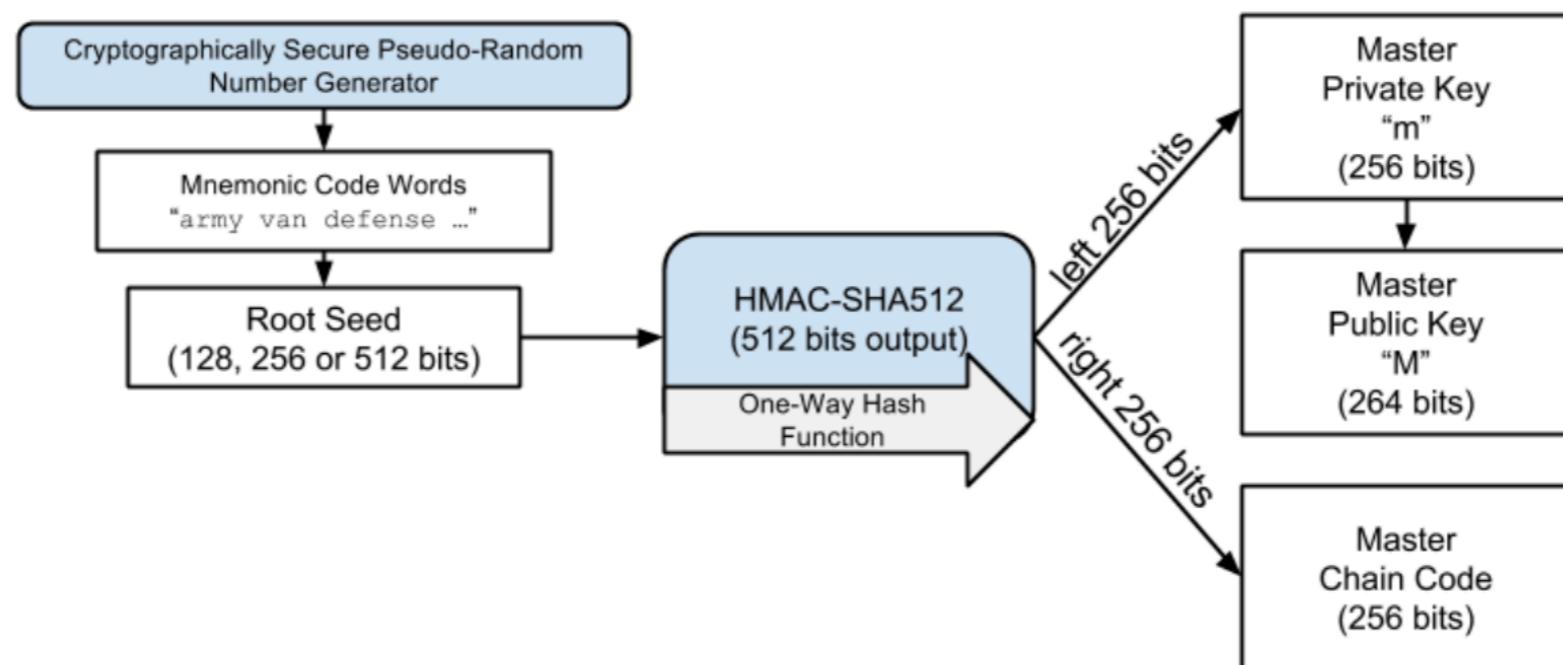
# 分层确定性钱包

- hierarchical deterministic wallets (HD 钱包): BIP-32/BIP-44
- 优势: a. 树形结构赋予密钥更多组织上的意义; b. 用户无需接触相应私钥即可产生一系列公钥



# 创建 HD 钱包

- 对根种子使用 HMAC-SHA512 哈希函数，将得到的哈希值左 256 位作为主私钥，右 256 位作为主链码，主公钥由主私钥与 secp256k1 生成点 G 相乘得到
- 主链码：用于创建子密钥时引入的熵



# 扩展密钥

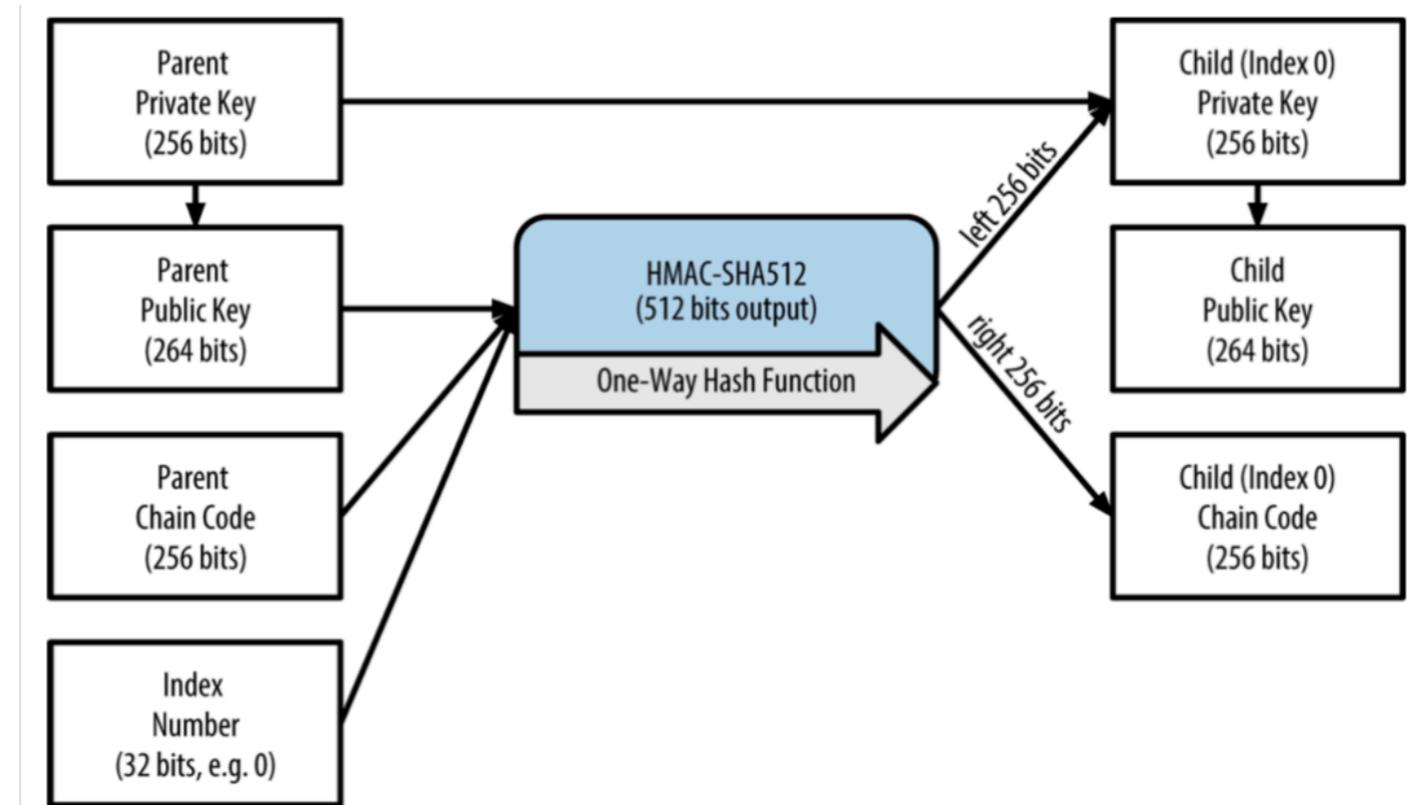
- 两种类型：
  - 扩展私钥：私钥 + 链码，可以被用来创建所有的子私钥，当然由子私钥也可以得到子公钥
  - 扩展公钥：公钥 + 链码，可以被用来创建所有的子公钥，仅仅能产生子公钥
- 扩展密钥的格式：使用 base58Check 编码方式，以“xprv”或者“xpub”开头

# 子私钥派生

- 子密钥派生函数 (CKD)：从父密钥处派生得到子密钥

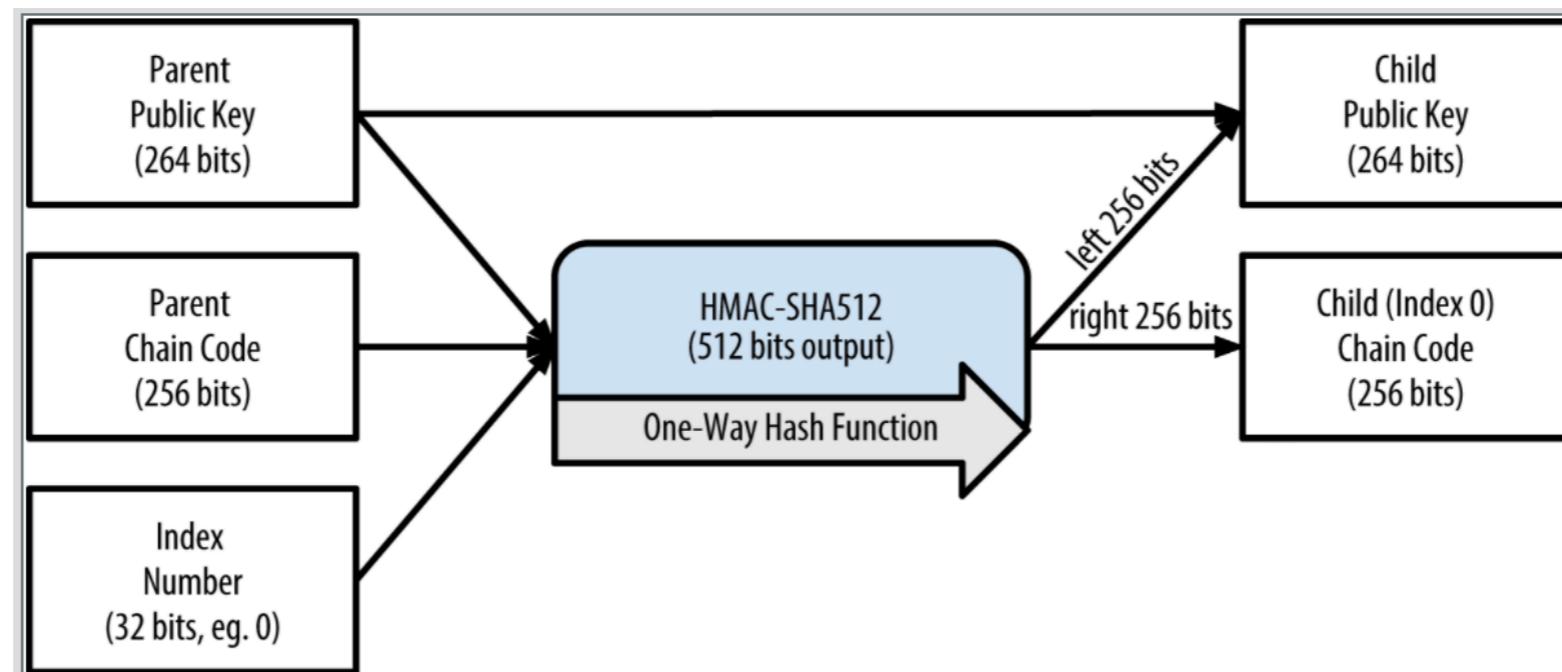
- 父私钥或者公钥
- 链码（256 位），作为种子使用
- 索引号（32 位）：0x00000000-0x7fffffff

- 首先通过父公钥、父链码、索引号进行 HMAC-SHA512 哈希
- 将结果的左 256 位与父私钥相加（对 n 取模）来产生子私钥
- 右 256 位作为子密钥的链码使用



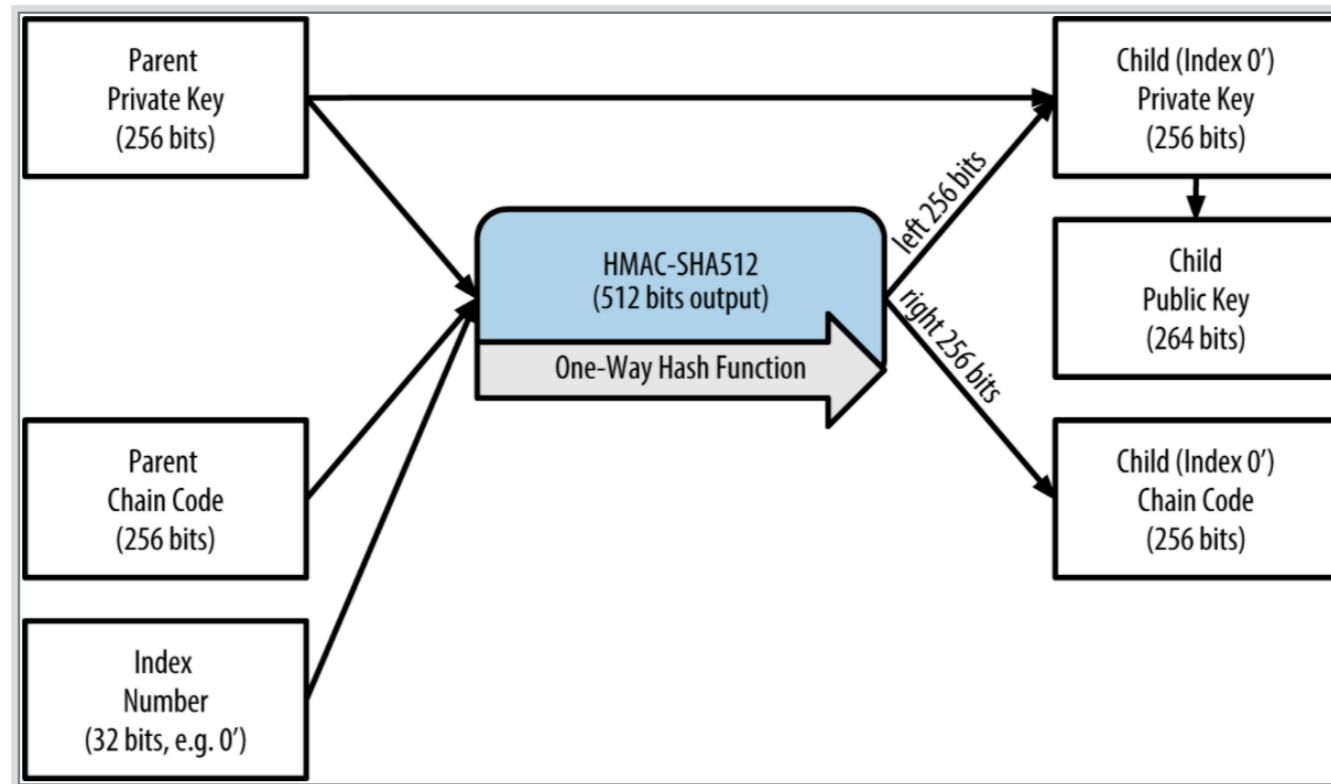
# 子公钥派生

- 两种方式获取子公钥：一是从父扩展私钥，一是从父扩展公钥
- $\text{child\_pub} = \text{parent\_pub} + \text{left\_256} * G$  等价于
- $\text{child\_priv} = \text{parent\_priv} + \text{left\_256} \pmod n$     $\text{child\_pub} = \text{child\_priv} * G$
- 缺点：泄露了链码



# 强化子密钥派生

- 打破了父公钥与子链码之间的联系
- 使用父私钥来派生子链码
- 混合架构
- 索引号：0x80000000 (0' 表示) - 0xffffffff



# HD 钱包密钥识别符

- 路径命名式

| HD 钱包路径 | 描述                     |
|---------|------------------------|
| m/0     | 主私钥 (m) 的第一个 (0) 常规子私钥 |
| m/0/0   | 第一个子私钥 (m/0) 的第一个常规子私钥 |
| m/0'/0  | 第一个强化子私钥的第一个常规子私钥      |
| M/0     | 主公钥 (M) 的第一个常规子公钥      |

- BIP44: m / purpose' / coin\_type' / account' / change / address\_index

| HD 钱包路径             | 描述                     |
|---------------------|------------------------|
| M/44' /0' /0' /0/2  | 比特币主账户的第 3 个接收付款公钥     |
| M/44' /0' /3' /1/14 | 第 4 个比特币账户的第 15 个 找零公钥 |
| m/44' /2' /0' /0/1  | 莱特币主账户的第 2 个私钥，签名交易使用  |

# 交易

# 基本概念

- 交易输出：不可分、记录在链上 
- UTXO 集合：全节点追踪所有可用的输出，数百万之多
- 交易：消耗 UTXO 集合中的元素，并增添新的元素
- 地址余额：该地址对应的所有 UTXO 之和
- 交易类型：coinbase（创币交易）、正常交易

# 交易结构

| 字节数 | 名称          | 数据类型             | 描述                             |
|-----|-------------|------------------|--------------------------------|
| 4   | version 版本号 | uint32_t         | 当前版本号为 1，新的共识规则可能会使用更高的数字      |
| 可变  | tx_in 数量    | compactSize uint | 交易中 input 的个数                  |
| 可变  | tx_in       | txIn             | 交易输入，见后面的描述                    |
| 可变  | tx_out 数量   | compactSize uint | 交易中 output 的个数                 |
| 可变  | tx_out      | txOut            | 交易输出，见后面的描述                    |
| 4   | lock_time   | uint32_t         | 时间戳或者区块号，小于 5 亿为区块号，大于 5 亿为时间戳 |

# 交易输入结构 (非 coinbase)

| 字节数 | 名称               | 数据类型             | 描述                          |
|-----|------------------|------------------|-----------------------------|
| 36  | previous_output  | outpoint         | 该 input 引用的输出               |
| 可变  | script bytes     | compactSize uint | 签名脚本的字节数，最大值是 10000         |
| 可变  | signature script | char[]           | 满足引用的输出中的公钥脚本。只包含数据 push 操作 |
| 4   | sequence         | uint32_t         | 序列号，默认是 0xffffffff          |

## outpoint 结构

| 字节数 | 名称    | 数据类型     | 描述                          |
|-----|-------|----------|-----------------------------|
| 32  | hash  | char[32] | 拥有所花费的输出的交易 ID              |
| 4   | index | uint32_t | 引用的交易的第几个输出，第一个为 0x00000000 |

# coinbase 交易输入结构

| 字节数    | 名称  | 数据类型             | 描述   |
|--------|---|------------------|--|
| 32     | hash  | char[32]         |  32 字节的 null, 因为 coinbase 交易无引用输出 |
| 4      | index   | uint32_t         | 0xffffffff   |
| 可变     | script bytes  | compactSize uint | coinbase script 的字节数, 最大为 100 字节   |
| 可变 (4) | height             | script           | BIP34 增添的区块高度。使用脚本语言来表示, 主网是 0x03, 表示紧随其后的 3 个字节来表示区块高度, 使用小端模式  |
| 可变     | coinbase script  | None             | 任意数据, 但是不能超过 100 - 可变(4), 矿工通常在该字段中放置 extranonce 来更新区块的 merkle 根哈希值  |
| 4      | sequence  | uin32_t          | 序列号  |

# 交易输出结构

| 字节数 | 名称              | 数据类型             | 描述   |
|-----|-----------------|------------------|--|
| 8   | value           | int64_t          | 支付的 satoshis (聪) 数量, 1 btc = 10 ^8 satoshi |
| 可变  | pk_script bytes | compactSize uint | pubkey script 的字节数量, 最大值是 10000 字节         |
| 可变  | pk_script       | char[]           | 定义了花费这笔输出的需满足的条件                           |

# 交易原始数据（字节流）



- 对交易对象进行序列化，便于网络中传输，文件中存储

```
01000000135c8fef9b4d780a71c8e40f361f5d45054  
f0ef5c7404149fe3c046b4d519333e01000006a473  
04402201e1d109e50b17dbf371ed8681061bb46d53  
efb65b3b8f6d33d69fd9a775ec068022034066fd9a3  
cac459b2e13badaa6ca374f973a597a886848b80d9  
d6da3879443901210271fa5d0090754cb23472d02e  
4537349dba49c7f638d33551d0f06a8f8eb15b73feff  
ffff023e6d2a0000000001976a91462f6d23f96442f5  
4e87a4983bfccbf84af83442788ac94527600000000  
001976a914f1ff425d107f61e54d1aa977ebe466d328  
52f14288ac00000000
```

```
{  
    "lock_time":0,  
    "size":225,  
    "inputs": [  
        {  
            "prev_out": {  
                "index":1,  
                "hash": "3e3319d5b446c0e39f1404745ceff05450d4f561f3408e1ca780d7b4f  
9fec835"  
            },  
            "script": "47304402201e1d109e50b17dbf371ed8681061bb46d53efb65b3b8f6d33  
d69fd9a775ec068022034066fd9a3cac459b2e13badaa6ca374f973a597a886848b80d9d6da387944  
3901210271fa5d0090754cb23472d02e4537349dba49c7f638d33551d0f06a8f8eb15b73"  
        },  
        {  
            "version":1,  
            "vin_sz":1,  
            "hash": "0d505effa50386d8e97497a112ad8d4012a6c22a8f5cdaf653cbea2b94073a0f",  
            "vout_sz":2,  
            "out": [  
                {  
                    "script_string": "OP_DUP OP_HASH160  
62f6d23f96442f54e87a4983bfccbf84af834427 OP_EQUALVERIFY OP_CHECKSIG",  
                    "address": "1A2GuF5drR2pHp3qjDDAivkEqsJ1XHayzr",  
                    "value": 2780478,  
                    "script": "76a91462f6d23f96442f54e87a4983bfccbf84af83442788ac"  
                },  
                {  
                    "script_string": "OP_DUP OP_HASH160  
f1ff425d107f61e54d1aa977ebe466d32852f142 OP_EQUALVERIFY OP_CHECKSIG",  
                    "address": "1P4ZdxJhV1wBpqoCiuvwDVqR9f7kkbCJMp",  
                    "value": 7754388,  
                    "script": "76a914f1ff425d107f61e54d1aa977ebe466d32852f14288ac"  
                }  
            ]  
        }  
    ]  
}
```



放于 <https://blockchain.info/zh-cn/decode-tx> 处进行解析

01000000 – version 1

01 – 交易输入的数量 1

35c8fef9b4d780a71c8e40f361f5d45054f0ef5c7404149fe3c046b4d519333e – 引用交易的 ID

01000000 – 输出的索引 1



6a – 解锁脚本的大小 106 字节

47 – 代表 push 操作 71 字节



304402201e1d109e50b17dbf371ed8681061bb46d53efb65b3b8f6d33d69fd9a775ec068022034066fd9a3ca  
c459b2e13badaa6ca374f973a597a886848b80d9d6da3879443901 – secp256k1 签名 71 字节，最后的 01  
代表 SIGHASH\_ALL



21 – 代表 push 操作 33 字节

0271fa5d0090754cb23472d02e4537349dba49c7f638d33551d0f06a8f8eb15b73 – 压缩过后的公钥 33 字节

02 代表偶数

feffff – 代表序列号 0xffffffff 代表该交易不可被替代，但是 locktime 是生效的



02 – 交易输出的数量 2

3e6d2a0000000000 – 金额 0x2a6d3e 2780478 satoshi

19 – pubkey script 的大小 25 个字节

76 – OP\_DUP

a9 – OP\_HASH160

14 – push 20 个字节



62f6d23f96442f54e87a4983bfccbf84af834427 – 目标地址的公钥

88 – OP\_EQUALVERIFY

ac – OP\_CHECKSIG

9452760000000000 – 第二个输出的金额 0x765294 7754388 satoshi

19 76 a9 14

f1ff425d107f61e54d1aa977ebe466d32852f142

88 ac

00000000 – 锁定时间



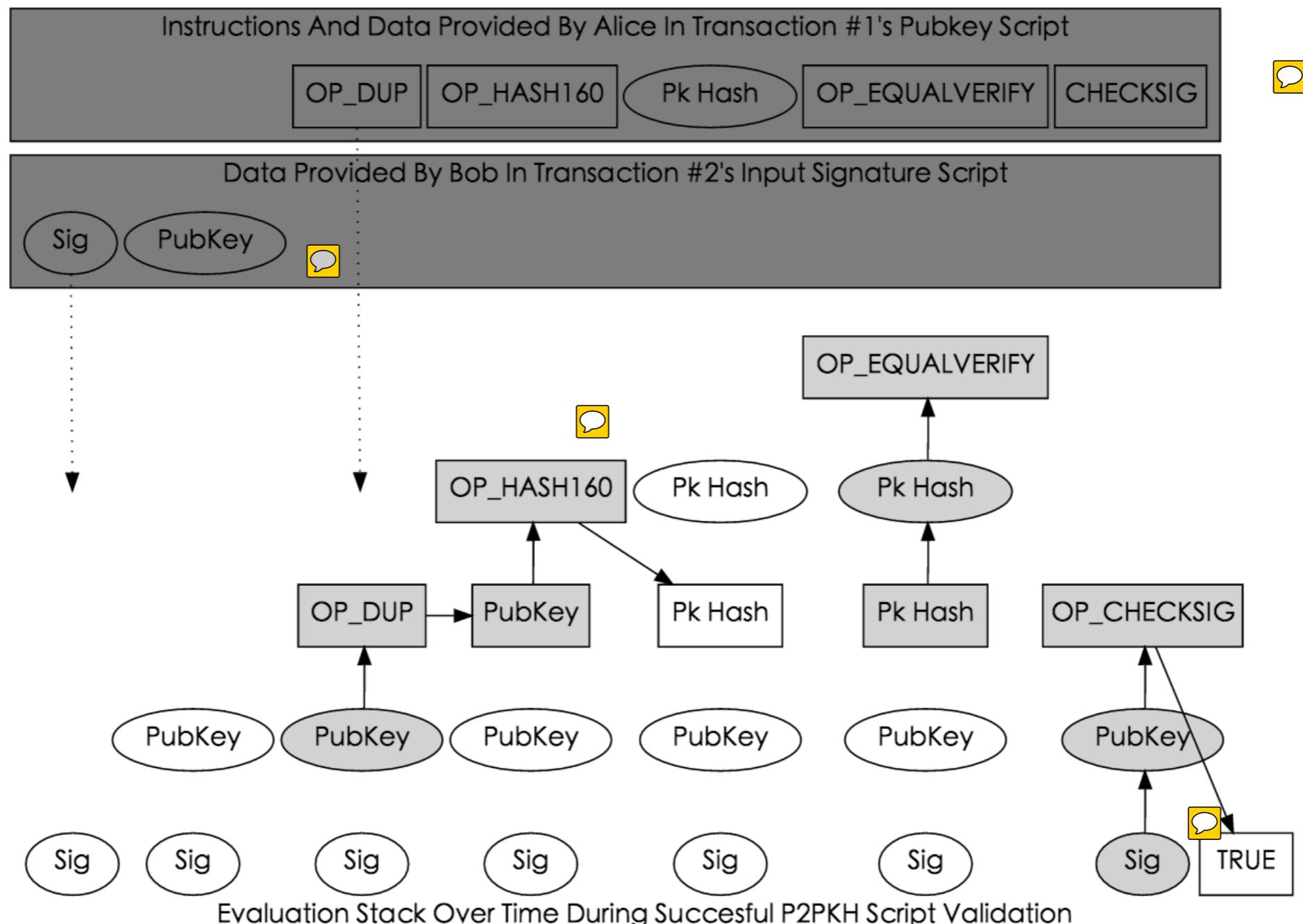
# 交易手续费

- 产生：交易输入金额之和 - 交易输出金额之和
- 计价规则：交易体积大小（KB）、市场定价，与交易金额无关
- 作用：1. 激励矿工进行挖矿，保证网络的安全；2. 作为防范攻击者使用交易对比特币网络进行洪泛攻击的安全机制
- `minrelaytxfee` 字段定义手续费，目前为 0.00001 btc；矿工可以覆盖该值，小于该值的交易几乎不会被打包
- 动态费用调整：钱包、交易所会根据过去几个块的平均费用来计算一个合理的费用

# 交易脚本与脚本语言

- 类 forth 语言，基于栈的语言，逆波兰式
- 图灵不完备<sup>对话框</sup>，安全
- Stateless Verification 无状态验证 <sup>对话框</sup>
- 两种类型：锁定（交易输出）、解锁脚本（交易输入）
- Pay-to-Public-Key-Hash (P2PKH)、Pay-to-Script-Hash (P2SH)

# P2PKH



# 比特币签名

- Elliptic Curve Digital Signature Algorithm ECDSA 椭圆曲线数字签名算法
- 验签操作码：OP\_CHECKSIG、OP\_CHECKSIGVERIFY、 OP\_CHECKMULTISIG、OP\_CHECKMULTISIGVERIFY
- 三个目的：a. 证明拥有相应私钥；b. 授权证据不可否认；c. 签名防止交易被修改
- 交易的每一个 input 是单独签名，因此一笔交易可以属于多个拥有者

# 数字签名工作原理

- 两部分：1. 使用私钥、消息来生成签名的算法；2. 允许任何人使用消息、公钥来验证签名的算法
- 比特币生成签名算法： $\text{Sig} = \mathcal{F}_{\text{sig}}(\mathcal{F}_{\text{hash}}(m), dA)$ ,  $dA$  为签名私钥,  $m$  是交易或者交易部分,  $\mathcal{F}_{\text{hash}}$  是哈希函数,  $\mathcal{F}_{\text{sig}}$  是签名算法,  $\text{Sig}$  为结果签名
- $m$  消息种类：由签名哈希类型 (SIGHASH) 定义
- $\text{Sig} = (R, S)$ ,  $R$ 、 $S$  使用 DER 编码
- 比特币验证签名算法：给定消息、签名、公钥，有效返回 TRUE



# 签名序列化(DER)

304402201e1d109e50b17dbf371ed8681061bb46d53efb65b3b8f6d33d69fd9a775ec068022034066fd9a3  
cac459b2e13badaa6ca374f973a597a886848b80d9d6da3879443901

30 – 代表 DER 序列的开始

44 – 代表序列的长度 (68 字节)

02 – 代表下面跟着一个整型数值

20 – 整型数值的长度 (32 字节)

R – 1e1d109e50b17dbf371ed8681061bb46d53efb65b3b8f6d33d69fd9a775ec068

02 – 另一个跟随的整型值

20 – 整形数值的长度 (32 字节)

S – 34066fd9a3cac459b2e13badaa6ca374f973a597a886848b80d9d6da38794439

01 – 代表使用的消息哈希种类 (SIGHASH\_ALL)

# ECDSA 数学原理

- 生成一对临时的公私钥  $k$ 、 $P$ ,  $P = k * G$ , 数字签名中的  $R$  为  $P$  的  $x$  坐标
- 生成公式:  $S = k^{-1}(Hash(m) + dA * R) \bmod p$ 
  - 其中  $dA$  为签名私钥;  $m$  是交易数据;  $p$  为椭圆曲线的素数阶
- 验证公式:  $P = S^{-1} * Hash(m) * G + S^{-1} * R * Qa$ 
  - 其中  $Qa$  为  $dA$  对应公钥
  - 如果求解出的  $P$  的  $x$  坐标与  $R$  相等, 那么认为该签名有效
- 签名中随机数的重要性 (RFC 6979) : 使用同一  $k$  对不同的交易进行签名, 会导致签名私钥泄露

# SIGHASH 类別

- 三种基本类型外加一个修饰符 ANYONECANPAY
- 上文提到的  $m = \text{交易特定部分} + \text{SIGHASH 标志}$

| SIGHASH 标志            | 值    | 描述                                |
|-----------------------|------|-----------------------------------|
| ALL                   | 0x01 | 对所有输入与输出签名                        |
| NONE                  | 0x02 | 只对所有输入签名                          |
| SINGLE                | 0x03 | 对所有输入签名，并且只对与签名所在的输入索引号所对应的输出进行签名 |
| ALL   ANYONECANPAY    | 0x81 | 对一个输入与所有输出签名                      |
| NONE   ANYONECANPAY   | 0x82 | 只对一个输入签名                          |
| SINGLE   ANYONECANPAY | 0x83 | 只对一个输入以及该输入对应的输出进行签名              |

# SIGHASH 举例

| SIGHASH 标志 | 举例描述 |
|------------|------|
|------------|------|

ALL | ANYONECANPAY

可以被用来构建众筹式交易，想筹集资金的人构建一笔只有单个输出的交易，该输出将目标金额发送给资金筹集者，这样一笔交易现在当然是无效的，因为它没有输入。然而，其它人可以在该交易中增添属于自己的输入作为捐款，他们使用 ALL | ANYONECANPAY 来签署自己的输入，除非收集到足够的输入，否则该交易就是无效的

NONE

可以被用来创建具有特定金额的不记名支票或者是空白支票，它允许输出被改变，任何人都可以将他们自己的地址写入到输出的锁定脚本中，然后使用该笔交易。然而，该笔交易在比特币区块链网络中只会生效一次，其它交易会因为 UTXO 已被引用而失败

NONE | ANYONECANPAY

可以被用来构建微尘收集者，当人们钱包中存在极小金额的 UTXO 时，花费它显得不值当，因为手续费太高，因此他们可以将该 UTXO 捐赠给想要聚集进行花费的人

# 高级交易及脚本

# 多方签名

- M of N: 目前的实现限制在 1-of-1 到 15-of-15
- 锁定脚本: M <Public Key 1> ... <Public Key N> N CHECKMULTISIG
- 解锁脚本: <Signature 1> ... <Signature M>
- But A bug in CHECKMULTISIG: 消耗  $M+N+3$  个栈的元素, 因此实际中用到的解锁脚本是这样的 – 0 <Signature 1> ... <Signature M>

# Pay-to-Script-Hash(P2SH)



- 传统多方签名脚本的缺点： a. 发送方需要使用特定的钱包来定制交易脚本，学习成本高；  
b. 发送方构造的交易体积是普通交易的数倍，导致手续费高，即成本转移到了发送方； c. 发送方构造的交易的输出在花费之前将存在于每个全节点的 RAM 中，内存压力变大
- 复杂的锁定脚本由其数字指纹代替，简化了复杂交易脚本的使用
- Redeem Script    2 PubKey1 PubKey2 PubKey3 3 CHECKMULTISIG
- Locking Script    HASH160 <20-byte hash of redeem script> EQUAL
- Unlocking Script    Sig1 Sig2 <redeem script>
- 两阶段执行，首先验证赎回脚本（前期作为栈中的一个元素，而不进行展开）的哈希值与锁定脚本中是否相同；其次将赎回脚本展开，再次进行验证
- P2SH 编码为以 3 开头的比特币地址

# 数据记录 (RETURN)



- 比特币作为分布式和时间戳化的账本，能做的不仅仅是支付！！！
- 数字公证服务：将文件的哈希值作为交易的输出记录在区块链上，proof-of-existence
- 由使用地址作为数字指纹 => RETURN (于 Bitcoin Core 0.9 中引进)
- RETURN <data> : 可证明的不可花费的输出，限制在最多 80 字节
- isStandard() : 最对允许有一个 RETURN 输出，但是 RETURN 能与其它输出相结合
- command-line : datacarrier 控制是否转发或打包带有 RETURN 的交易；  
datacarriersize 定义该节点最多能接受的字节数，默认是 83: RETURN(1) + pushdata(2) + 80

# coinbase 交易 of Genesis Block 0

## input

04ffff001d0104455468652054696d  
65732030332f4a616e2f323030392  
04368616e63656c6c6f72206f6e20  
6272696e6b206f66207365636f6e6  
4206261696c6f757420666f722062  
616e6b73

## The Times 03/Jan/2009 Chancellor on brink of second bailout for banks





# 绝对时间锁



- 交易级别的时间锁 nLocktime, 局限性：在未来花费某些输出变得可能，但是没有让输出在未来某时刻之前被花费变得不可能
- **例：**假设 Alice 签署了一笔交易，该交易的输出时 Bob 的地址，并且设置 nLocktime 为 3 个月后，Alice 将该笔交易发送给 Bob。这时，Bob 不能将这笔交易提交至比特币网络，因为此时 3 个月期限还未到，然而，Alice 完全可以创建另外一笔交易花费同一输出，并且让该笔交易立即生效，这样 Bob 3 个月后也不能花费该笔交易。
- UTXO 级别的时间锁：CHECKLOCKTIMEVERIFY (CLTV BIP-65)



# CLTV



- <now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG

- CLTV 失败的条件：



- 栈是空的；或
- 栈顶元素小于 0；或
- 栈顶元素的时间锁类型 (区块高度或unix时间戳) 与花费交易的 nLocktime 字段的时间锁类型不同；或
- 栈顶元素大于交易的 nLocktime 字段
- 输入的 nSequence 值为 0xffffffff

# 相对时间锁

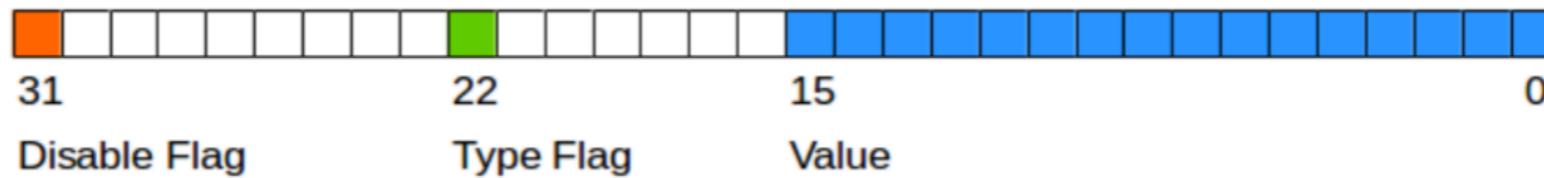
- nLocktime 和 CLTV 都是绝对时间锁，定义了一个特定的时间点
- 用于双向状态通道和闪电网络中 
- 交易级别的相对时间锁：通过输入中的 nSequence 实现 (BIP-68)
- UTXO (脚本) 级别：CHECKSEQUENCEVERIFY (CSV BIP-112)
- BIP-68、BIP-112 均在 2016 年 5 月的对共识规则的软分叉升级中激活

# 交易级别相对时间锁

- 自 BIP-68 激活后,  $nSequence < 2^{31}$  时, 解释为相对时间锁;  
 $nSequence \geq 2^{31}$  保留为未来使用;  $nSequence$  为 `0xffffffff` 代表交易中的 `nLocktime` 字段不生效,  $nSequence$  为 `0xfffffff` 代表 `nLocktime` 生效并且不可替代



- `nSequence` 字段可解释为区块高度, 也可解释为 unix 时间戳, 不过与 `nLocktime` 字段解释的模型不太一样: 第 23 个 bit 代表锁类型, 如果设置为 1, 那么 `nSequence` 解释为 512 秒的整数倍; 如果为 0, 则解释为高度。另外只取最低 16 位来解释



# CSV



- 与 CLTV 用法相似
- 当交易输入中的 nSequence 大于等于输出脚本中 CSV 前面的时间锁时，该笔交易才是有效的，也就意味着 UTXO 被确认后的一段时间内，才能生效
- 必须匹配交易的 nSequence 字段类型，即区块对区块，时间戳对时间戳
- 在链式交易（均离线，未上传至比特币网络）中尤其实用，子交易只有在父交易上传至比特币网络并且被打包确认一段时间后才能生效

# MTP

- 区块头内的时间戳由矿工设置，因此矿工为了奖励可以将并不成熟的时间锁交易打包在内。
- Median-Time-Past (BIP-113, activated): 定义为过去 11 个区块的时间戳的中位数，被用到所有时间锁的计算中
- 包括 nLocktime、CLTV、nSequence、CSV



# 时间锁—抵抗手续费阻击

手续费阻击（Fee-Sniping）是一种理论上的攻击场景，在该场景下矿工尝试使用来自未来区块的更高手续费的交易来重写过去的区块，使他们能获得最大的利润。

例如，假设目前最高的区块号是 #100000，如果一些矿工尝试重新挖 #100000 号区块，而不是挖 #100001 号区块来延长区块链，那么这些矿工就可以包含任何有效的交易到他们的候选区块 #100000 中。实际上他们有动机去选择利润最大的交易打包进区块，这时他们选择交易的来源就有两种：一是原先包含在区块 #100000 中的交易；二是来自于当前矿工的交易池中的交易。本质上来说，他们有选择的权利，将来自于目前的交易打包进已经产生的区块 #100000 中。当然，目前来讲，这种做法并不是利润最大化的，因为区块的奖励远远大于手续费，但是在未来，矿工的整个激励都会来源于手续费，那个时候，这种攻击场景就会普遍存在了

解决方案：比特币核心客户端创建交易的时候，将该笔交易的 nLocktime 限制在下一个区块，也就是，在上面的场景中，nLocktime 将是 #100001。为了实现这点，比特币核心客户端将所有交易的所有输入的 nSequence 设置为 0xFFFFFFFF 来使 nLocktime 生效

# 控制流脚本

- 操作码： IF、 ELSE、 ENDIF、 NOTIF
- 控制流可以相互嵌套，但是共识规则对脚本的大小有一个限制



```
IF                               script A : <Alice's Pubkey> CHECKSIG  
  script A  
ELSE                            unlocking script A : <Alice's Sig> 1  
  IF                               script B : <Bob's Pubkey> CHECKSIG  
    script B  
  ELSE                            unlocking script B : <Bob's Sig> 1 1  
    script C : <Carl's Pubkey> CHECKSIG  
  ENDIF                           unlocking script C : <Carl's Sig> 0 1  
ENDIF
```

谢谢聆听