《椭圆曲线密码算法在比特币中的应用》

简介

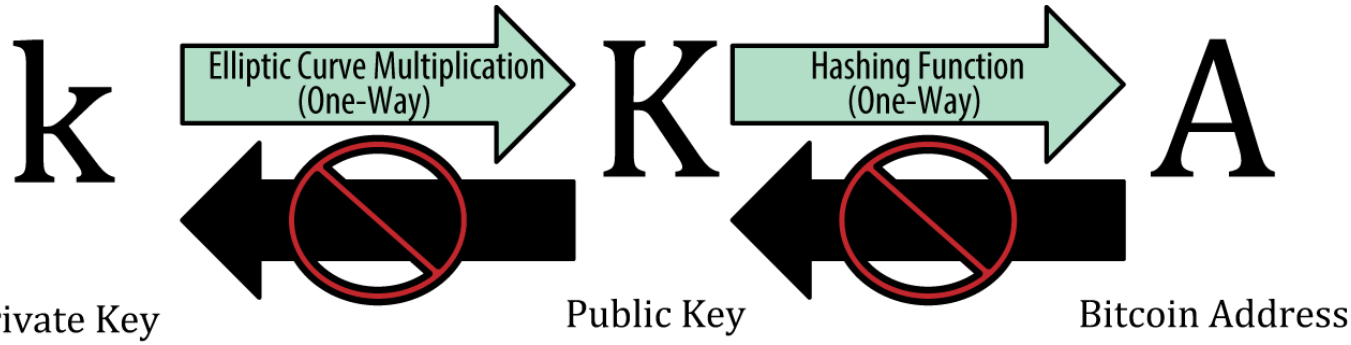
- 不同于以太坊的账户模型，比特币采用的是UTXO模型，比特币的所有权是通过*数字密钥*、*比特币地址和数字签名*来确定的。数字密钥实际上并不存储在网络中，而是由用户生成之后，存储在一个叫做*钱包*的文件或简单的数据库中。用户钱包中的数字密钥完全独立于比特币协议，可由用户的钱包软件生成并管理，而无需参照区块链或访问互联网。
- 大多数比特币交易都需要在区块链中存储一个有效的数字签名。该数字签名只能由密钥产生，因此拥有密钥副本就等于拥有了该帐户中比特币的控制权。用于支出资金的数字签名也称为*见证（witness）*，这是密码学中的术语。比特币交易中的见证数据证明了资金的真正所有权。
- 密钥是成对出现的，由私钥和公钥所组成。公钥就像银行的帐号，而私钥就像银行卡的密码或支票的签名。比特币的用户很少会直接看到数字密钥。一般情况下，它们存储在钱包文件内，由比特币钱包软件进行管理。
- 在比特币交易的支付环节，收款人的公钥由数字指纹表示，称为*比特币地址*，就像支票上收款人名称 （即“付给谁的账户”）。一般情况下，比特币地址由公钥生成并与之对应。然而，并非所有比特币地址都代表公钥；也可以代表其他支付对象，譬如脚本。这样一来，比特币地址就可以抽象成资金接收者，使得交易更灵活，就像纸质支票：可以支付到个人账户或公司账户，也可以支付账单和现金。比特币地址是密钥被用户能够看到的唯一形式，因为这个地址就是需要告诉别人的。
- 以上是对比特币和密码学应用的简单概述，而本篇调研报告将重点关注椭圆曲线密码算法在比特币中的应用，将从与之密切相关的密钥与地址、钱包、交易三部分介绍相关内容。

密钥与地址

- 在比特币系统中，我们用公钥密码学技术创建一个密钥对，用于控制对比特币的访问。密钥对包括一个私钥，和由其衍生出的唯一的公钥。公钥用于接收比特币，而私钥用于支付时进行交易签名。
- 比特币正是使用椭圆曲线密码算法作为其公钥密码学的基础。

私钥与公钥与地址

一个比特币钱包中包含一系列的密钥对，每个密钥对包括一个私钥和一个公钥。私钥（*k*）是一个数字，通常是随机选出的。基于私钥，我们就可以使用椭圆曲线乘法这个单向密码函数产生一个公钥（*K*）。基于公钥（*K*），我们就可以使用一个单向密码哈希函数生成比特币地址（*A*）。私钥、公钥和比特币地址之间的关系如下图所示。



私钥

- 私钥就是一个随机选出的数字而已。拥有和控制了私钥，就相当于控制了该私钥对应的比特币地址中的所有资金。通过证明比特币交易中资金的所有权，私钥可以生成花费该笔资金的签名。
- ***从一个随机数生成私钥*** 是生成密钥的第一步也是最重要的一步，是要找到足够安全的熵源，即随机性来源。生成一个比特币私钥在本质上与“在1到2^{256}之间选一个数字”无异。只要选取的结果是不可预测或不可重复的，那么选取数字的具体方法并不重要。比特币软件使用操作系统底层的随机数生成器来产生256位的熵（随机性）。
- 更准确地说，私钥可以是1和n-1之间的任何数字，其中n是一个常数（n=1.158 * 10^{77}，略小于2^{256}），并被定义为由比特币所使用的椭圆曲线的阶（比特币使用的是 **secp256k1** 标准所定义的一种特殊的椭圆曲线和一系列数学常数。该标准由美国国家标准与技术研究院 （NIST）建立）。要生成这样的一个私钥，我们随机选择一个256位的数字，并检查它是否小于n-1。从编程的角度来看，一般是通过在一个密码学安全的随机源中取出一长串随机字节，对其使用SHA256哈希算法进行运算，这样就可以方便地产生一个256位的数字。如果运算结果小于n，我们就有了一个合适的私钥。否则，我们就用另一个随机数再重复一次。

公钥

以一个随机生成的私钥*k*为起点，将其乘以曲线上一个预定的点，叫做*生成点G*得到曲线上的另一点，这就是相应的公钥 *K*。生成点是secp256k1标准的一部分，比特币密钥的生成点都是相同的：

$$K = k * G$$

其中*k*是私钥，*G*是生成点，在该曲线上所得的点*K*是公钥。因为所有比特币用户的生成点是相同的，一个私钥*k*乘以*G*将得到相同的公钥*K*。*k*和*K*之间的关系是固定的，但只能单向运算，即从*k*得到*K*。这就是可以把比特币地址（*K*的衍生） 与任何人共享而不会泄露私钥（*k*）的原因。

实现了椭圆曲线乘法，我们就可以用随机产生的私钥*k*和与生成点*G*相乘得到公钥*K*。下面给出一个简单例子。

```
k = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
K = k * G

公钥K被定义为一个点 K = (x, y) ，其中：
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

比特币地址

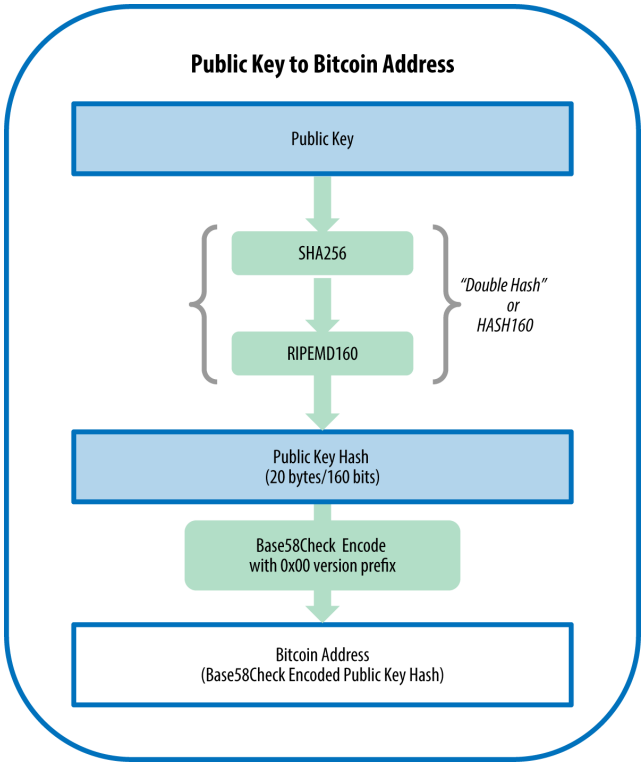
比特币地址可由公钥经过单向哈希算法得到。密码学哈希算法是一种单向函数，接收任意长度的输入产生指纹或哈希。哈希函数在比特币中被广泛使用： 比特币地址、脚本地址以及在挖矿中的工作量证明算法。由公钥生成比特币地址时使用的算法是Secure Hash Algorithm (SHA)和the RACE Integ rity Primitives Evaluation Message Digest (RIPEMD)，具体来说是SHA256和RIPEMD160。

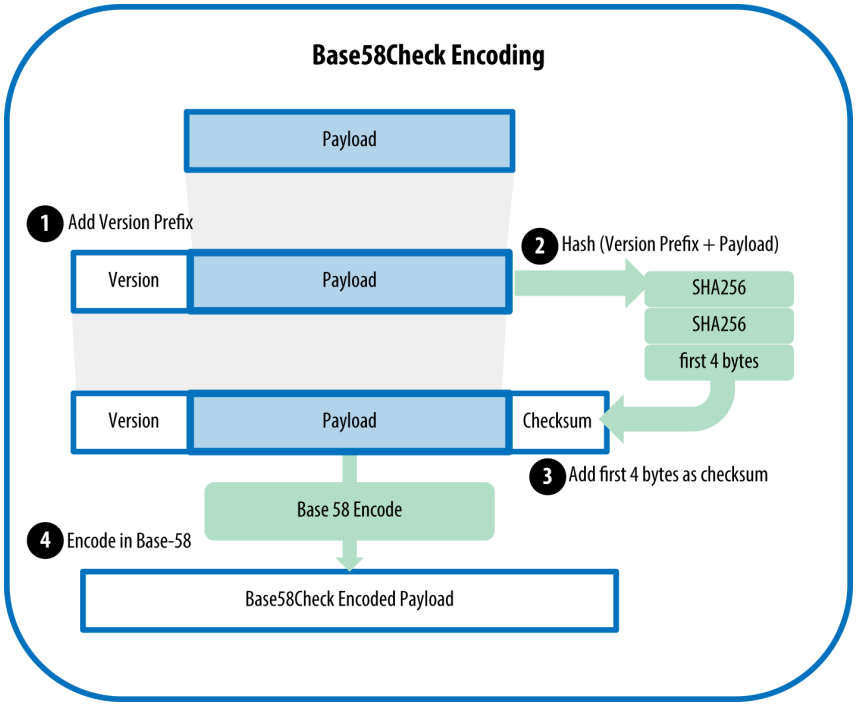
以公钥 *K* 为输入，计算其SHA256哈希值，并以此结果计算RIPEMD160 哈希值，得到一个长度为160位（20字节）的数字：

$$A = RIPEMD160(SHA256(K))$$

公式中，*K*是公钥，*A*是生成的比特币地址。

通常用户见到的比特币地址是经过“**Base58Check**”编码的（参见下面的“Base58Check Encoding”一图），这种编码使用了58个字符（Base58数字系统）和校验码，提高了可读性、避免歧义并有效防止了在地址转录和输入中产生的错误。Base58Check编码也被用于比特币的其它地方，例如比特币地址、私钥、加密的密钥和脚本哈希中，用来提高可读性和录入的正确性。





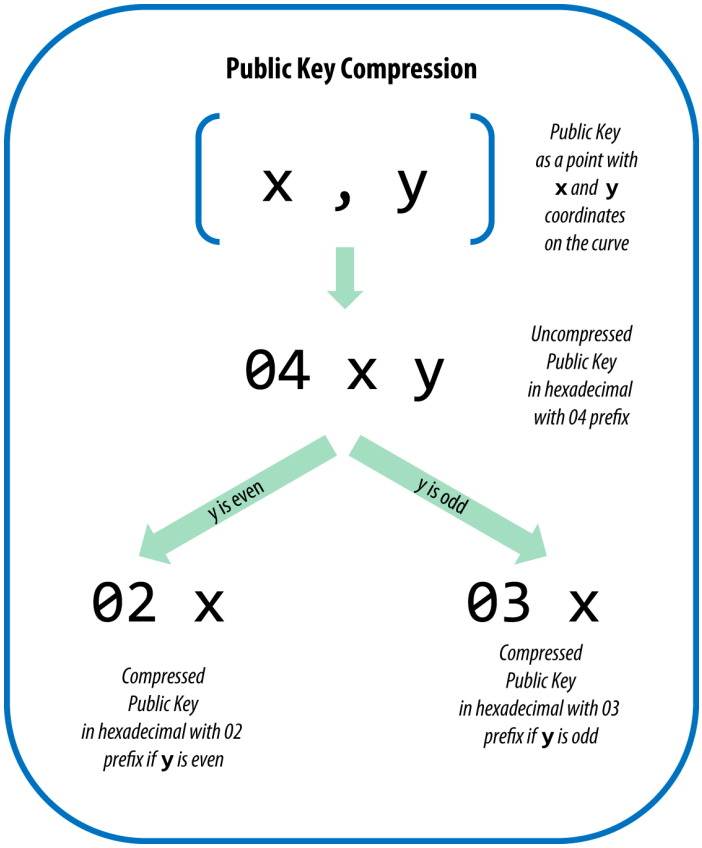
- **WIF(Wallet import format)** 钱包导入格式，(也被称为电子钱包的导出格式)是一种私有的 ESCDSA (椭圆曲线签名算法) 秘钥，意在使私钥更容易复制，私钥的各种形式如下表所示。

Type	Prefix	Description
Raw	None	32字节
Hex	None	64位Hex编码
WIF	5	使用Base58Check进行编码
WIF-compressed	K or L	使用Base58Check进行编码，但在编码前增加后缀0x01

Format	Private key
Hex	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

- 公钥也可以用多种不同格式来表示，通常分为非压缩格式或压缩格式公钥这两种形式。
- 我们从前文可知，公钥是在椭圆曲线上的一个点，由一对坐标（x，y）组成。公钥通常表示为前缀04紧接着两个256位的数字。其中一个256位数字是公钥的x坐标，另一个256位数字是y坐标。前缀04是是非压缩格式公钥， 压缩格式公钥是以02或者03开头。
- 正如前面所说，一个公钥是一个椭圆曲线上的点(x, y)。而椭圆曲线实际是一个数学方程，曲线上的点实际是该方程的一个解。因此，如果我们知道了公钥的x坐标，就可以通过解方程来得到y坐标。这可以让我们只存储公钥的x坐标，略去y坐标，从而将公钥的大小和存储空间减少了256位。这样每笔交易需要的字节数就会减少许多，随着时间推移，就能保存更多的交易数据。
- 未压缩格式公钥使用04作为前缀，而压缩格式公钥是以02或03作为前缀。为什么会有两个前缀：**当我们在素数p阶的有限域上使用二进制算术计算椭圆曲线的时候，y坐标可能是偶数或者奇数**。因此，为了区分y坐标的两种可能值，在生成压缩格式公钥时，如果y是偶数，则使用02作为前缀；如果y是奇数，则使用03作为前

缀。这样就可以让软件能够根据x坐标，正确推导出对应的y坐标，从而将公钥解压缩为在椭圆曲线上点的完整坐标。下图阐释了公钥压缩的过程：



> 注意：实际上“压缩格式私钥”是一种名称上的误导，因为当私钥使用WIF压缩格式导出时，不但没有压缩，反而比“非压缩格式”私钥长出一个字节。这个多出来的一个字节是私钥被加了后缀01，用以表明该私钥是来自于一个较新的钱包，只能被用来生成压缩公钥。私钥并没有压缩的，也不能被压缩。“压缩私钥”实际上表示“只能生成压缩公钥的私钥”，而“非压缩私钥”用来表明“只能生成非压缩公钥的私钥”。为避免更多误解，应该只可以说导出格式 是“WIF压缩格式”或者“WIF”，而不能说这个私钥是“压缩”的。

钱包

- 广义上，钱包是一个应用程序，为用户提供交互界面。钱包控制用户资金访问权限，管理密钥和地址，跟踪余额以及创建和签名交易。
- 狭义上，“钱包”是指用于存储和管理用户密钥的数据结构。
- 在这一小节中，我们将更关注钱包是密钥容器的这一角度，探索椭圆曲线算法在其中的应用。

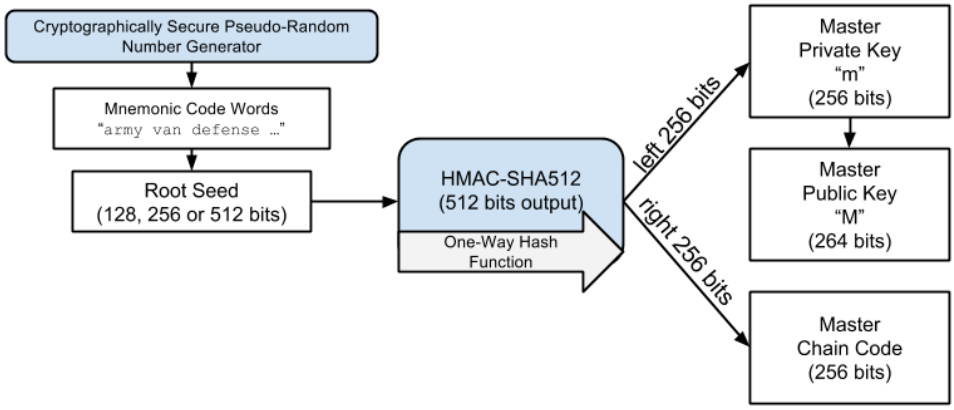
HD分层钱包

根据钱包包含的多个密钥之间是否有关系，主要分为两种类型：

- 第一种类型是*非确定性钱包（nondeterministic wallet）*，其中每个密钥都是从随机数独立生成的。密钥彼此无关。

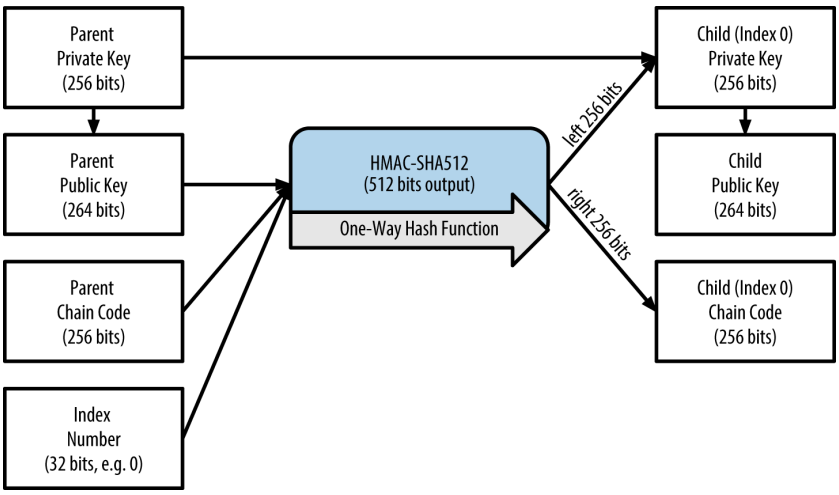
- 第二种类型是确定性钱包（deterministic wallet），其中所有的密钥都是从一个主密钥派生出来，这个主密钥即为种子（seed）。该类型钱包中所有密钥都相互关联，如果有原始种子，则可以再次生成全部密钥。确定性钱包中使用了许多不同的密钥推导方法。最常用的推导方法是使用树状结构，称为分层确定性（hierarchical deterministic）钱包或HD钱包。
    - 相比随机（不确定性）密钥，HD钱包有两个主要的优势。第一，树状结构可以被用来表达附加的组织含义，比如子密钥的特定分支用来接收交易收入款项，另一个分支用来负责接收对外付款的找零。密钥的分支也可以用于公司设置，将不同的分支分配给部门、子公司、特定功能或会计类别。
    - HD钱包的第二个好处是，用户可以创建一系列公钥，而不需要访问对应的私钥。这样，HD钱包就能用在不安全的服务器上，或者仅作为接收用途，它为每个交易发布不同的公钥。公钥不需要被预先加载或者提前衍生，服务器也不需要有用来支付的私钥。

HD钱包密钥的生成过程如下图所示：通过记助词和 Salt 生成Root Seed种子，再通过HMAC-SHA512（单向哈希），输出512位bits，其中左256位作为主私钥（通过椭圆曲线密码算法可生成主公钥，采用的是压缩公钥格式264bits），右256位作为主链码。



> HD钱包的种子生成过程，并没有涉及椭圆曲线密码算法，因此这里不再展开介绍，具体过程可参考：[HD分层钱包 种子生成过程]

衍生子秘钥



- 子秘钥的衍生推导如上图所示：父公钥 + 父链码 + 索引作为输入， 输出512bits，右256bits作为子链码；左256bits 与 父秘钥结合得到子秘钥
- 衍生得到的子私钥，推断不出兄弟姐妹的私钥；如果没有子链码，也无法推断出孙密钥
- 可以得到如下的推导公式：

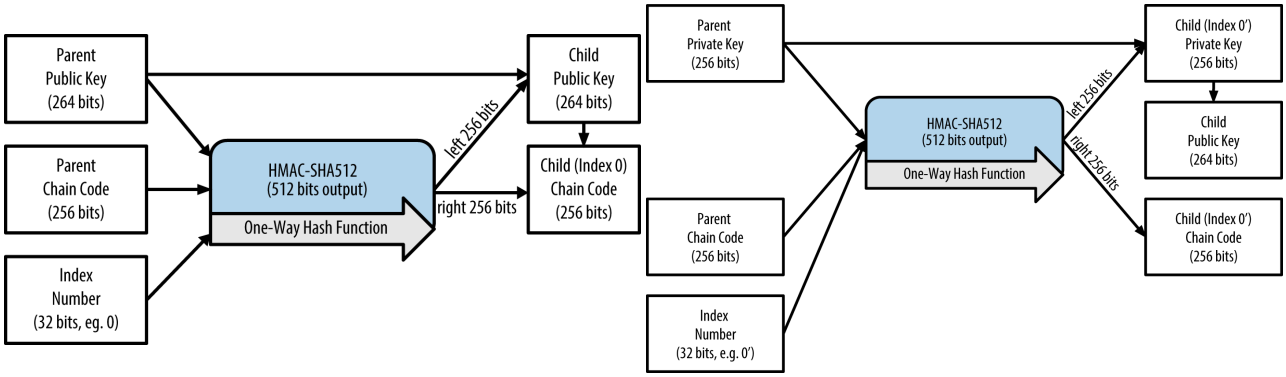
```
child_private_key == (parent_private_key + lefthand_hash_output) % G
child_public_key  == point( (parent_private_key + lefthand_hash_output) % G )
child_public_key  == point(child_private_key) == parent_public_key +
point(lefthand_hash_output)
```

- 可以看到，子公钥的推导有两种方式，一个是通过私钥推导，一个是直接通过公钥推导
- 正如我们之前看到的，密钥衍生函数可以被用来创造密钥树上任何层级的子密钥，基于以下三个输入量：密钥，链码以及想要的子密钥的索引。

  密钥以及链码这两个重要的部分被结合之后，就叫做

  扩展密钥（extended key）

  。术语“扩展密钥”也被认为是“可扩展的密钥”，因为这种密钥可以用来衍生子密钥。
  - 扩展密钥编码用的 Base58Check使用特殊的版本号，Base58编码字符前缀分别为“xprv”和“xpub”



- 上面左图为常规衍生，直接使用父公钥衍生子公钥，这样推导的好处就是不用访问私钥，就能衍生扩展出子公钥，用于收款，可以用在安全性较弱的web服务器上
  - 但是这种常规的扩展方式存在的问题是，当一个攻击者得到父公钥和对应的链码时，他可以衍生扩展出之后所有的子公钥和对应链码，如果此时再获得子私钥，那么就可以推导出这个子私钥之后所有的私钥（孙私钥等），甚至可能会推导出父私钥

```
child_private_key == (parent_private_key + lefthand_hash_output) % G
```

- 右图为强化衍生，不使用父公钥，直接用父私钥 + 父链码 + 索引 作为输入，得到输出（这样就算子公钥和子链码暴露了，也不能推导出父私钥）
  - 简单来说，如果想利用xpub的便捷来衍生公钥的分支，又不想冒泄露链码的风险， 就该从强化父密钥，而不是常规父密钥衍生。最好的方式是，为了避免主密钥泄露，主密钥所衍生的第一层级的子密钥总是通过强化衍生得来。

交易

- 比特币交易是比特币系统中最重要的部分。比特币中的其他一切都是为了确保交易可以被创建、在网络上传播、验证，并最终添加到全局交易分类账本（区块链）中。比特币交易的本质是数据结构，这些数据结构是对比特币交易参与者价值传递的编码。比特币区块链是一本全局复式记账总账簿，每个比特币交易都是在比特币

区块链上的一个公开记录。

- 交易主要的功能就是实现比特币的转移，而其实现过程就用到了密码学技术。支付比特币时，比特币的当前所有者需要在交易中提交其公钥和签名（每次交易的签名都不同，但都由同一个私钥生成）。针对展示的公钥和签名，比特币网络中的所有人都可以验证该交易有效并予以接受，从而确认支付者对该交易中的比特币的所有权。
- 这一小节中，将介绍交易的基本数据结构，关注比特币交易过程的实现，并聚焦密码学技术的使用。

交易的输出

每一笔比特币交易都会创造输出，并被比特币正本记录下来。几乎所有的输出，除了一个例外（见“数据输出操作符”（OP_RETURN）），都能创造称为UTXO的比特币块，然后被整个网络识别，供所有者在未来交易中使用。

交易输出包含两部分：

- 一定量的比特币，面值为“聪”（satoshis），是最小的比特币单位；
- 确定花费输出所需条件的加密难题（cryptographic puzzle），这个加密难题也被称为锁定脚本(locking script), 见证脚本(witness script), 或脚本公钥 (scriptPubKey)。
    - 只有出示对应的解锁脚本，成功执行锁定脚本（相当于进行签名验证的过程），才能花费这笔输出

```
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
OP_EQUALVERIFY OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
OP_EQUALVERIFY OP_CHECKSIG",
  }
]
```

交易的输入

交易的输入包含四个元素：

- 一个交易ID，引用包含将要消费的UTXO的交易
- 一个输出索引（vout），用于标识来自该交易的哪个UTXO被引用（第一个为零）
- 一个 scriptSig（解锁脚本），满足UTXO的消费条件，解锁用于支出
- 一个序列号，用于指定这笔输入的生效时间

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039f
f08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787ec
3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
]
```

数字签名

在上述交易的输入中提到了一个关键元素解锁脚本，实际上这就是应用了椭圆曲线密码算法的数字签名。在比特币的ECDSA算法的实现中，被签名的"消息"是交易，或更确切地说是交易中特定数据子集的哈希值。签名密钥是用户的私钥，结果就是签名：

$$Sig = F_{sig}(F_{hash}(m), d_A)$$

- *dA* 是签名私钥
- *m* 是交易（或其部分数据）
- *Fhash* 是散列函数
- *Fsig* 是签名算法
- *Sig* 是结果签名

函数*Fsig* 产生由两个值组成的签名Sig，通常称为R和S：

$$Sig = （R，S）$$

现在已经计算了两个值R和S，它们就使用一种称为 *可分辨编码规则Distinguished Encoding Rules*或*DER*的国际标准编码方案，序列化为字节流，正如我们在交易输入的JSON字符串中看到的那样。

签名序列化（DER）

我们再来之前在交易输入中展示的一个解锁脚本，其中以下DER编码签名：

```
3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff
08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
```

该签名是用户钱包生成的R和S值的序列化字节流，证明对应用户拥有授权花费该输出的私钥。 序列化格式包含以下9个元素：

- *0x30*表示DER序列的开始
- *0x45* - 序列的长度（69字节）
- *0x02* - 一个整数值
- *0x21* - 整数的长度（33字节）
- *00884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb* - R值

- *0x02* - 接下来是一个整数
- *0x20* - 整数的长度（32字节）
- *4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813* - S值
- 后缀（*0x01*）指示使用的哈希的类型（SIGHASH_ALL）

验证签名

- 要验证签名，必须有签名（*R*和*S*）、序列化交易和公钥（对应于用于创建签名的私钥）。本质上，签名的验证意味着“只有生成此公钥的私钥的所有者，才能在此交易上产生此签名。”
- 签名验证算法采用消息（交易或其部分的哈希值）、签名者的公钥和签名（R和S值），如果签名对该消息和公钥有效，则返回 TRUE 值。

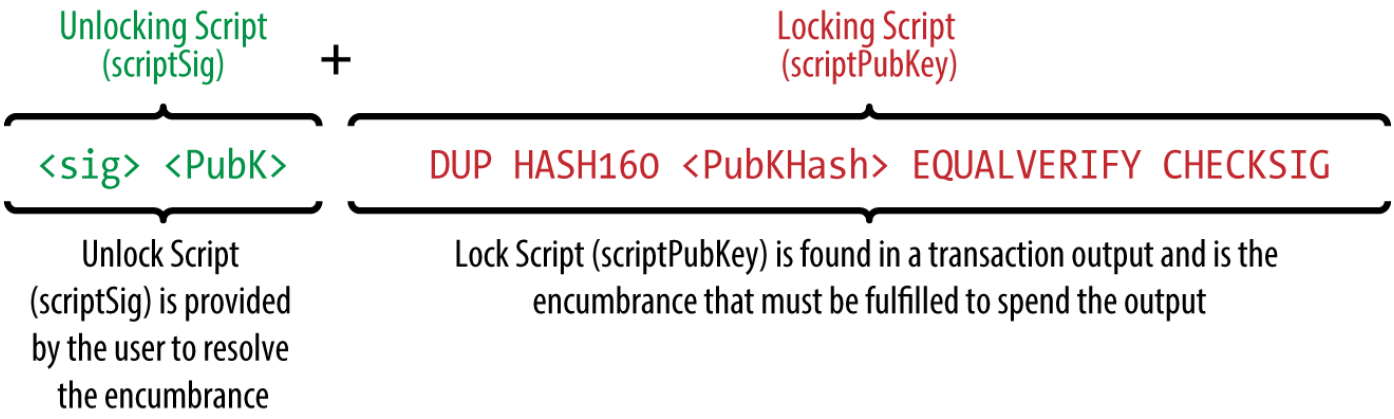
交易脚本

正如之前内容提到的，锁定脚本和解锁脚本正是组成数字签名和签名验证过程的关键数据。

- 锁定脚本：锁定脚本是一个放置在交易输出上面的花费条件——它指定了今后花费这笔输出必须要满足的条件。 由于锁定脚本往往含有一个公钥或比特币地址（公钥哈希值），在历史上它曾被称为*脚本公钥 scriptPubKey*。
- 解锁脚本：解锁脚本是这样一个脚本，它“解决”或满足由锁定脚本放置在输出上的条件，并允许使用输出。解锁脚本是每一笔比特币交易输入的一部分，而且往往含有一个由用户的比特币钱包（通过用户的私钥）生成的数字签名。由于解锁脚本常常包含一个数字签名，因此它曾被称作*脚本签名ScriptSig*。
- 当结合锁定脚本和解锁脚本，在堆栈中执行脚本，如果堆栈中最后剩余的结果为“TRUE”，表明解锁脚本中签名跟锁定脚本中的公钥哈希是相匹配的，即用于签名的私钥跟公钥是匹配的。其中，签名验证的这个验证操作正是锁定脚本中 “**OP_CHECKSIG**” 所执行的，即比对消息（交易或其部分的哈希值）、签名者的公钥和签名（R和S值）。

脚本执行

- 在最初版本的比特币客户端中，解锁脚本和锁定脚本按顺序连起来执行。出于安全因素考虑，在2010年发生了改变，因为存在一个漏洞，允许格式错误的解锁脚本将数据推送到堆栈并损坏锁定脚本。而在当前的方案中，脚本是单独执行的，在两次执行之间传输堆栈，如下所述。
    - 首先，使用堆栈执行引擎执行解锁脚本。如果解锁脚本在执行过程中未报错（例如：没有留下“dangling”操作码），则复制主堆栈，并执行锁定脚本。如果从解锁脚本中复制而来的堆栈数据执行锁定脚本的结果为“TRUE"，那么解锁脚本就成功地满足了锁定脚本所设置的条件，因此，该输入是一个能使用该UTXO的有效授权。如果合并脚本执行后的结果是”TRUE“以外的任何结果，输入都是无效的，因为它不能满足UTXO中所设置的使用该笔资金的条件。



> 具体脚本在堆栈中的执行过程，可参考：P2PKH 脚本执行流程

总结

比特币系统中大量使用了密码学技术，例如使用椭圆曲线密码算法生成密钥和公钥，以及交易中的数字签名和验证，确保比特币的所有者在进行转账操作。密码学实现了比特币的许多有趣特性，包括去中心化信任和控制、所有权认证和基于密码学证明的安全模型。可以说，密码学是比特币实现的一大基石。

参考资料附录

《精通比特币》中文电子书

比特币（地址、私钥）压缩与非压缩的区别

HD分层钱包 种子生成过程

P2PKH 脚本执行流程

Signature verification in python using compressed public key

Exploring Bitcoin: signing the P2PKH input

比特币私钥，公钥和地址的关系

在线搜索交易记录、区块信息（各种数字货币）