

## 第15讲 | 深入区块链技术（七）：哈希与加密算法

2018-04-27 陈浩

深入浅出区块链

[进入课程 >](#)



讲述：黄洲君

时长 09:27 大小 5.44M



区块链最核心的两个技术点是共识机制和密码学，由于共识机制是公链的基础，所以这部分内容我已经在前面的内容中优先讲解了。

接下来，我来讲一讲区块链的密码学基础，有关区块链密码学你只需要了解它的基本原理和优劣即可。

另外，区块链的密码学中文资料也十分丰富，如果你感兴趣的话，可以在学有余力的基础上酌情深入。

区块链中主要应用了两类密码学算法，第一类是哈希算法，第二类是非对称加密算法。

我们先来看看哈希算法。

# 1. 哈希算法

哈希算法是一类数学函数算法，又称散列算法，它是一种数据映射关系。

为了方便举例，我们假设  $h = \text{HASH}(X | z)$ ，你输入一个任意长的数据  $z$ ，经过哈希运算后，返回给你固定长度的数据  $h$ ， $z$  叫做原像， $h$  是哈希结果，又称作“数据指纹”， $z$  可选的数据集合构成了  $X$ 。

哈希算法具有下面的 4 种特性。

1. **原像不可逆**。原像不可逆是指对于任意给定的  $h$ ，都无法依据  $h$  自身的信息推导出  $z$ 。
2. **难题友好性**。难题友好性通俗的理解就是如果要得到难题答案，你只能暴力枚举，没有比这更好的方法。在  $h = \text{HASH}(X | z)$  中，从  $h$  无法推导出  $z$ ，只能不断地计算尝试，那么  $z$  所在的数值集合构成了  $X$ ， $X$  的大小是哈希算法的安全因子之一。
3. **发散性**。发散性是指对于任意的  $z$ ，即使我们只改动非常少的信息量，例如改动 1 个比特位生成  $z'$ ，那么  $\text{HASH}(z)$  与  $\text{HASH}(z')$  就是两个大相径庭的结果，完全不相似。
4. **抗碰撞性**。抗碰撞性是指对于任意两个不相同的  $z$ ，那么他们对应的  $h$  值也不同。如果对于任意的  $y$  不等于  $z$ ，则  $\text{HASH}(y)$  不等于  $\text{HASH}(z)$ ；满足上述定义哈希特性的算法，我们也称作具有严格抗碰撞性。如果我们任意给定一个  $z$ ，你都无法找到另外一个  $z'$ ，使得其值也等于  $h$ ，满足这样的哈希特性的算法就有弱抗碰撞性。

目前流行的 Hash 算法包括了 MD5、SHA-1 和 SHA-2，其中 MD5 被证明不具有强抗碰撞性。SHA（Secure Hash Algorithm）是一个 Hash 函数族，分为 SHA-1、SHA-2、SHA-3，代表了三代哈希标准，目前使用比较多的是 SHA-2 系列。

第一代的 SHA-1 基于 MD4 设计，并且模仿了该算法，SHA-1 已被证明了不具备“强抗碰撞性”，所以安全性不够高。

为了提高安全性，第二代 SHA-2 一共包含了 SHA-224、SHA-256、SHA-384，和 SHA-512 算法（统称为 SHA-2），它们跟 SHA-1 算法原理类似。SHA-3 相关算法也已被提出，它的出现并不是要取代 SHA-2，因为 SHA-2 目前并没有出现明显的弱点。

由于对 MD5、和 SHA-1 出现成功的破解，我们需要一个不同与之前算法，可替换的加密散列算法，也就是现在的 SHA-3。

## 1.1 区块链上的哈希算法

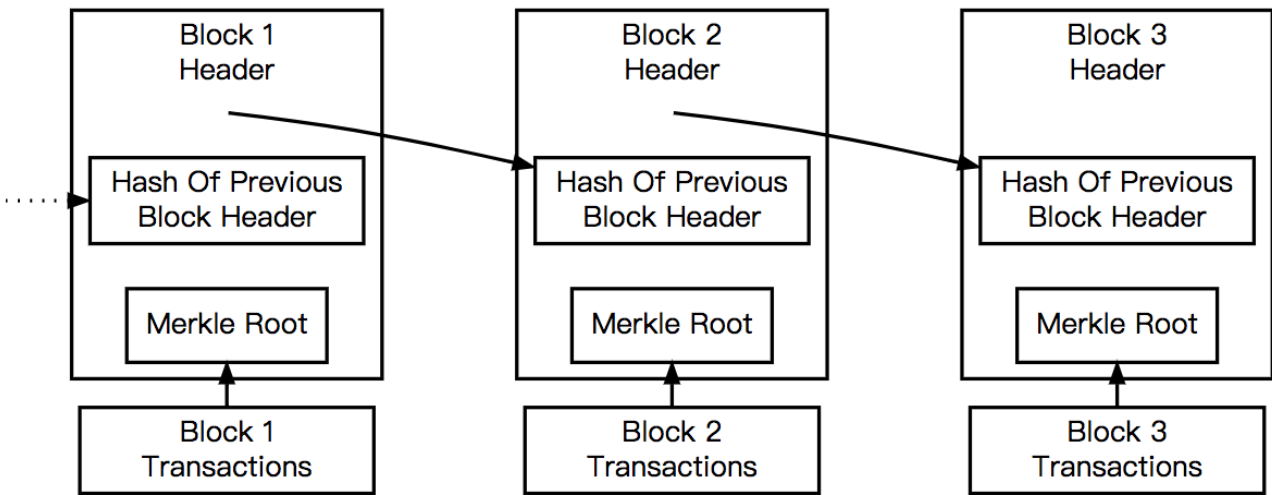
哈希算法被广泛地使用在构造和验证区块、交易的完整性上，由于哈希算法的四个特性，使得我们可以把任意的交易数据做成数据摘要，然后再一个一个链接起来，形成数据块的链式结构。

这样我们可以通过验证每个区块间接地验证交易，然后每个交易原数据也可以做成哈希数据摘要，用于验证交易数据的完整性。

我们借用比特币开发者文档中的图，这个图表达了区块链的基本数据结构，

## Block Chain Overview

[Edit](#) | [History](#) | [Report Issue](#) | [Discuss](#)



Simplified Bitcoin Block Chain

在图中可以看出，当前区块里面包含上一个区块的哈希，形成一个哈希指针链表，由于哈希的发散性，所以这个链表也有极大的发散性。

我们可以用代码模拟一遍，我们先列表构造 5 个简化的区块，其中第一个块是创世区块，我们规定它指向的前向区块的哈希全为零；

后面第 2 个块，第 3 个块中 content 分别记录了两笔交易，这里为了方便理解，我使用了文字表述交易的内容，实际上，区块链上的交易是二进制格式化的数据，而不是文本数据。代码中并没有填充哈希，是在运行时填充的。

复制代码


```
1 #!/usr/bin/env python
2 import hashlib
```

```

3
4 def main():
5     # example:
6     block_headers = [
7         {"prev_block_hash": "0000000000000000000000000000000000000000000000000000000000000000",
8          "prev_block_hash": "to_be_hashed", "content": "2nd block:C pay B 2.0 BTC"},
9         {"prev_block_hash": "to_be_hashed", "content": "3th block:transactions..."},
10        {"prev_block_hash": "to_be_hashed", "content": "4th block:transactions...j"},
11        {"prev_block_hash": "to_be_hashed", "content": "5th block:transactions..."}
12    ]
13
14    # hash prev block header
15    index = 0
16    for header in block_headers:
17        # genesis block, ignore
18        if index == 0:
19            print header
20            index = index + 1
21            continue
22
23        # generate hash chain
24        prev_block_header = block_headers[index - 1]
25        target_buffer = prev_block_header["content"] + prev_block_header["prev_block_hash"]
26        header["prev_block_hash"] = hashlib.sha256(target_buffer).hexdigest()
27        print header
28        index = index + 1
29    if __name__ == '__main__':
30        main()

```

我们可以直接得到结果，这是一个典型的哈希指针链表，每一个区块的 prev\_block\_hash 域指向上一个区块哈希。


 复制代码

```

1 {'content': 'genesis block:A pay C 12.3 BTC', 'prev_block_hash': '00000000000000000000000000000000'}
2
3 {'content': '2nd block:C pay B 2.1 BTC', 'prev_block_hash': '01279c1208a8eca3d4a47a1231f'}
4
5 {'content': '3th block:transactions...', 'prev_block_hash': '6d96c220b22371dc1d2b3549da'}
6
7 {'content': '4th block:transactions...j', 'prev_block_hash': '9e41c61fa151320145a56a38e'}
8
9 {'content': '5th block:transactions...', 'prev_block_hash': '34f002b445a38fa7402e590629'}

```

如果我们将第二块中的 content 从 "C pay B 2.1 BTC" 修改为 "C pay B 2.0 BTC", 那么我们将得到如下结果, 我们可以发现从第三个块往后所有的块指向的前一个区块的哈希都不再与上面的一致。

 复制代码

```
1 { 'content': 'genesis block:A pay C 12.3 BTC', 'prev_block_hash': '00000000000000000000000000000000'
2
3 { 'content': '2nd block:C pay B 2.0 BTC', 'prev_block_hash': '01279c1208a8eca3d4a47a1231:
4
5 { 'content': '3th block:transactions...', 'prev_block_hash': 'f91faad6b874fb97a20ad9cbc5:
6
7 { 'content': '4th block:transactions...j', 'prev_block_hash': '99d17dfe9a9fab68cffd6a82b:
8
9 { 'content': '5th block:transactions...', 'prev_block_hash': 'd2f42291ef0811e5babc1d38ca:
```

以上我们构造了一个极简的区块链的基本结构, 区块头描述了一个区块的基本信息, 在实际应用中, 里面通常包含了下面的几个内容。

区块序号	1087636
哈希	f629387fe4c024cc9174349f694d3decc40861a71c303965476089d44575e7fe
交易号	1
Nonce	3297454200890777420
Mix Hash	60203025907409127602595815945046232587277058052394994649451141494658600339652
版本	1
难度	6381894594765
默克尔树哈希	aeaa2d28785d4d27062bbcffdceaf810a2c75e39ad0d9384de4d3861b9ccb885
前一区块哈希	28083d90834d03249ecc36773f68ca53dafe47cf02815bb099393bd9876c9bff
出块时间	2018-04-07 15:47:17 +0800

图中有当前区块高度、本区块的哈希、前一区块哈希、nonce 值等等。

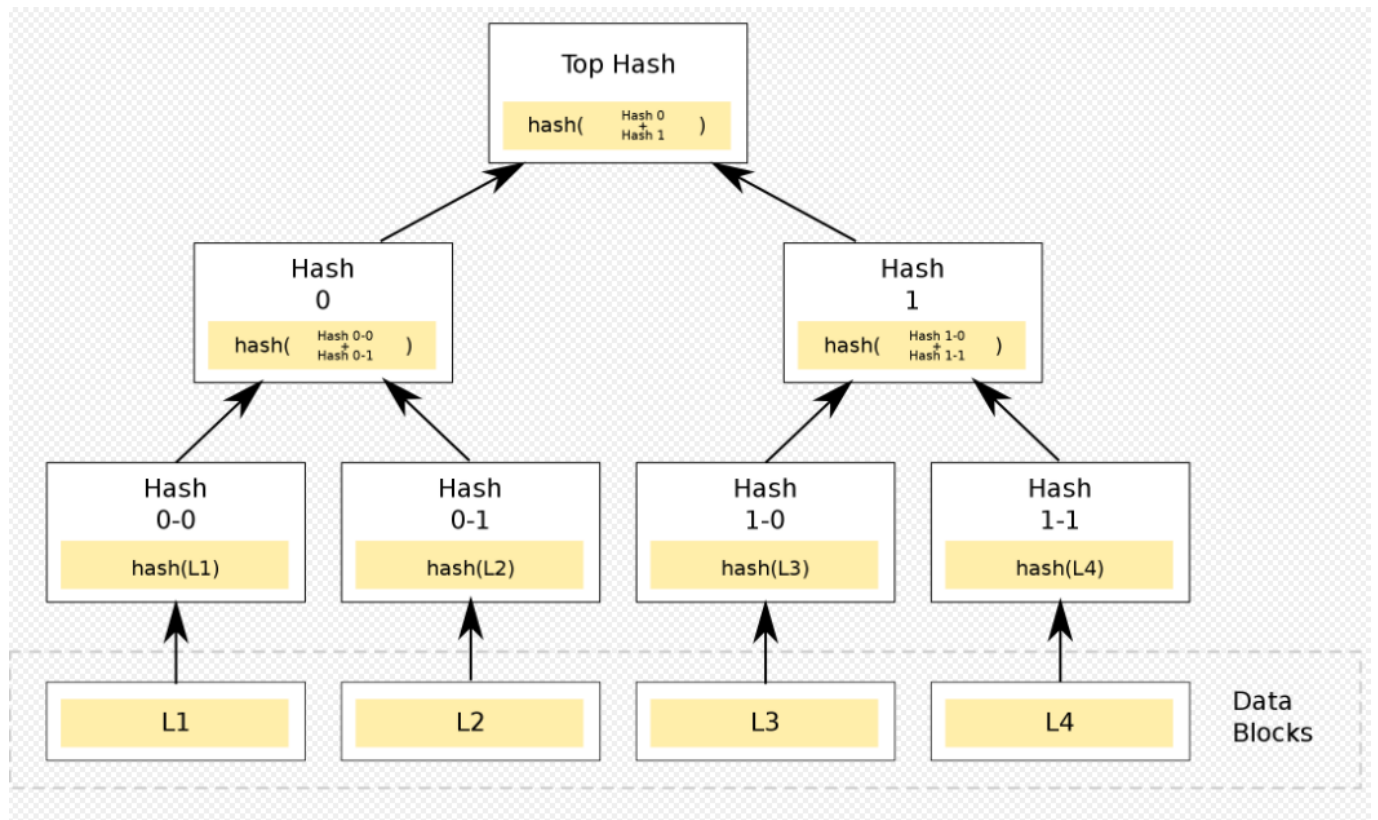
所以前一区块哈希是一个区块头必备的数据域, 这种链式结构具备发散传导性, 越往历史以前的篡改, 越容易导致大面积的影响, 这也叫做历史逆向修改困难。

在 PoW 共识机制的情况下, 这种逆向修改的难度会随着当前全网算力线性增长。

## 1.2 默克尔树 (Merkle tree)

哈希算法的一个重要应用是默克尔树（Merkle tree），默克尔树是一种数据结构，通常是一个二叉树，也有可能是多叉树，它以特定的方式逐层向上计算，直到顶部，最顶层叫做默克尔根，默克尔树最为常见和最简单的是二叉默克尔树。

默克尔树的基本结构如下图。

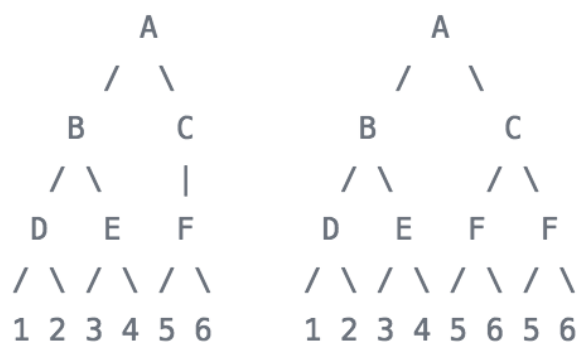


(图片来自维基百科)

比特币和以太坊都使用了默克尔树这种数据结构，只不过里面存放的数据都是哈希。我们在比特币的核心版本源码中可以发现注释中有介绍。

WARNING! If you're reading this because you're learning about crypto and/or designing a new system that will use merkle trees, keep in mind that the following merkle tree algorithm has a serious flaw related to duplicate txids, resulting in a vulnerability (CVE-2012-2459).

The reason is that if the number of hashes in the list at a given time is odd, the last one is duplicated before computing the next level (which is unusual in Merkle trees). This results in certain sequences of transactions leading to the same merkle root. For example, these two trees:



for transaction lists [1,2,3,4,5,6] and [1,2,3,4,5,6,5,6] (where 5 and 6 are repeated) result in the same root hash A (because the hash of both of (F) and (F,F) is C).

The vulnerability results from being able to send a block with such a transaction list, with the same merkle root, and the same block hash as the original without duplication, resulting in failed validation. If the receiving node proceeds to mark that block as permanently invalid however, it will fail to accept further unmodified (and thus potentially valid) versions of the same block. We defend against this by detecting the case where we would hash two identical hashes at the end of the list together, and treating that identically to the block having an invalid merkle root. Assuming no double-SHA256 collisions, this will detect all known ways of changing the transactions without affecting the merkle root.

(图片来自比特币 Core 源码)

以太坊中针对比特币的设计做了改进，叫做默克尔帕特里夏树 (Merkle Patricia tree)，相对于比特币在块头中只有一棵树，以太坊有三棵树。

区块链的挖矿算法也应用了哈希算法，挖矿算法利用的是其难题友好性，我们在 PoW 共识机制中讲解过，这里不再赘述。

## 2. 非对称加密算法

非对称加密算法是相对于对称算法而言的，这两者组成了密码学的核心内容。

这两者的使用区别体现在密钥是否可以公开，对称密钥要求加解密过程均使用相同的密钥，而非对称加密可以提供一对钥匙，私钥自己保管，公钥可以公开。

常见的对称加密算法有 DES、3DES、AES、IDEA，常见的非对称加密算法有 RSA、ECC 等。

在比特币等众多数字货币的项目中，在账户层面主要使用的是非对称加密算法。

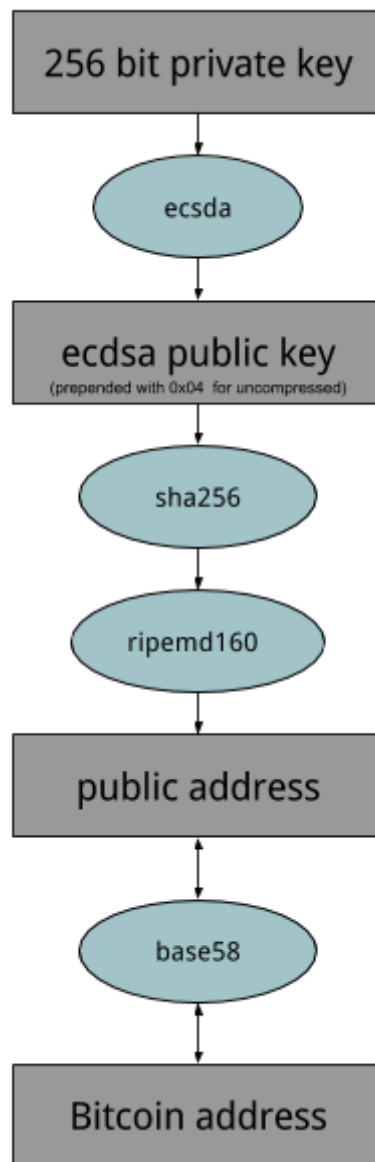
在对称加密算法里，由于双方需要提前共享密钥，在使用过程中有诸多不便，非对称算法的出现解决了这一难题。

在非对称算法中，私钥一般是通过一个随机数产生的，这个随机数我们也叫做种子，从这个角度来说，知道了这个随机数也就等于知道了私钥，不过私钥的产生范围非常大，在比特币中是 2 的 256 次方，差不多在 10 的 70 方数量级上。

如果你产生随机数的算法足够均匀分布，私钥碰撞的可能性比中了 1 亿大奖同时被雷劈中的概率还要小数亿倍。所以区块链对产生随机数的算法要求比较高，它要求真实的均匀随机分布，而不是计算机伪随机数。

如果我们有了私钥，接下来就如图所示：





我们从私钥到公钥，是由非对称加密算法保证的，这种算法在比特币中选择的是 ECDSA 算法，ECDSA 算法中选择的椭圆曲线名为 secp256k1。

而从公钥到地址，是由哈希算法保证的，在这一步使用了 SHA256 和 RIPEMD160。椭圆曲线加密算法 ECC 利用了“寻找离散对数”的难解性提供了单向不可逆性，具体原理你可以找资料了解一下。

在区块链上，一个比特币交易的产生由两部分组成，第一部分是签名加锁，对应到的是交易的输出、第二部分是解锁花费，对应到的是交易的输入，当我们构造一笔交易的时候必然会用到私钥，这是所有数字货币资产控制权由私钥保证的根本原因。具体逻辑我们留到下篇讲解 UTXO 的时候讨论。

最后来谈谈量子威胁的内容。我在讨论比特币等众多数字货币项目的时候，很多人会问我如何看待量子计算的威胁问题，大家认为量子计算的强大计算力威胁到了哈希的抗碰撞性。其

实这不是一个设计缺陷，而是一个发展问题，是可以在区块链的发展过程中解决的。

我对于量子计算的威胁论有以下的看法。

1. 即使出现了量子计算攻破非对称加密算法的问题，那么首先要看是什么算法，例如是 RSA，还是 ECC。
2. 其次要看攻击成本是否足够低，因为理论上的可行性并不代表工程可行性，这是两码事。例如持续攻击比特币要花费 1 亿美金，持续攻击时间超过 20 年才能生效，那么这笔买卖很明显不划算。
3. 量子计算威胁的对象不止加密货币，而是整个密码学体系，如果发生破解事件，很可能是银行、互联网后端系统，目前整个互联网应用都基于 HTTPS，如果 HTTPS 被破解，在量子计算面前传统的账号密码几乎不可用。
4. 量子计算目前发展虽然看起来喜人，但是离实际应用还很远，很多计算其实并非是通用计算，而是专用计算，也就是说一个量子计算机写入的算法只能解决一个特定问题，而且还是概率解，可用性易用性还需要较长时间转化。

## 总结

今天我向你介绍了哈希算法，讲解了区块链哪些地方使用了哈希算法，并介绍了非对称加密算法，最后还谈了一下我对量子计算威胁论的看法。

密码学是所有区块链的基础，可以说如果没有密码学的支撑，区块链将会退化成普通的分布式日志系统。

那么亲爱的读者，比特币地址有哪些类型呢？以太坊的地址又是如何生成的呢？你可以给我留言，我们一起讨论。感谢你的收听，我们下次再见。

参考阅读：

1. 《区块链技术指南》
  2. 《精通比特币》
  3. [https://yeasy.gitbooks.io/blockchain\\_guide/content/crypto/](https://yeasy.gitbooks.io/blockchain_guide/content/crypto/)
-

# 深入浅出区块链

你的区块链入门第一课

陈浩 元界 CTO



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第14讲 | 深入区块链技术（六）：DPoS共识机制

下一篇 第16讲 | 深入区块链技术（八）：UTXO与普通账户模型

## 精选留言 (8)

写留言



我才是二亮

2018-04-28

10

比特币有三种地址类型：

- 1、以数字1开头的P2PKH类型
- 2、以数字3开头的P2SH类型
- 3、以bc1开头的Bech32类型

...

展开

作者回复：赞，打call



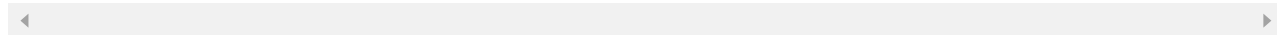
阿痕

2018-05-01

👍 6

文中没有提到，哈希算法和非对称加密相结合可以作为数字签名，这在区块链交易中应用的非常广泛

作者回复: 谢谢补充



寄意兰舟

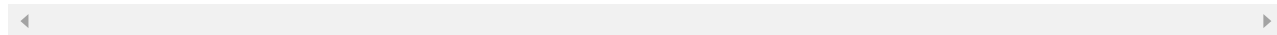
2018-05-13

👍 2

比特币核心版本源码在哪里可以看到啊？能给个链接？

展开 ▾

作者回复: github上搜bitcoin



teletime

2018-05-08

👍 1

公钥转为地址不可逆，那节点如何得到公钥？来进行交易验证

展开 ▾



四正

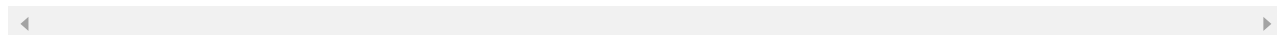
2018-04-28

👍 1

其实，哈希函数本质上就是把无限的信息映射到有限的空间中。无论摘要用多少个比特存储，终究是有限的。那么就必然存在碰撞的情况。当然，实际应用中这个概率可以忽略不计

展开 ▾

作者回复: 是的



朱月俊

2019-03-27

👍

请教一个问题：公钥和地址是客户互相转换的？

展开 ∨



良辰美景

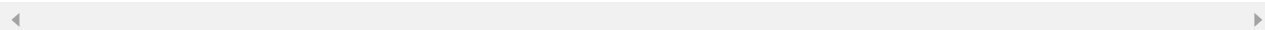
2018-04-28



第二节你有个链接文章说的。每个用户都有个保密印章和印章扫描器。保密印章就是私钥么，印章扫描器又是怎么实现的

作者回复: 为了方便解释，举的例子。

扫描器即是交易验证，又是交易解锁逻辑的实现。



吹牛老爹

2018-04-27



老师给的课后读物很赞

展开 ∨