区块和交易结构

路远

中国科学院软件研究所

前章要点回顾

- **计算安全性** (computational security)
  - 计算资源受限的多项式时间攻击者
  - 攻击者成功的概率(优势)是安全参数的可忽略函数
- **密码学哈希函数**
  - 抗碰撞性 (collision resistance)
- **数字签名**
  - 选择明文攻击下的存在不可伪造性 (EUF-CMA)
  - 基于Shamir秘密分享的门限签名 (threshold signature)
- **数字钱包**
  - 椭圆曲线场景下的私钥和公钥
  - 热钱包 (助记词、口令加密)
  - 冷钱包
- 其他的密码学工具
  - 承诺、向量承诺、零知识证明
  - 可验证随机函数、可验证延迟函数

本讲内容

- 认证数据结构
- 区块链和区块结构
- 创世区块解读
- 交易和生成交易
- 以太坊中的数据结构

认证数据结构：默克尔树和哈希链

一个现实问题：如何审计云存储篡改了数据?

Alice

F = 爱丽丝梦游仙境 **2160p BluRay** x264.mp4

Cloud

云存储为了节约存储空间，替换了文件：

F’ = 爱丽丝梦游仙境 **1080p HDTV** x264.mp4

思考：怎样利用上节课学习的密码学知识，帮助Alice识别自己在云存储上的文件被替换了呢?

一个现实问题：如何审计云存储篡改了数据?

Alice

Cloud

F = 爱丽丝梦游仙境 **2160p BluRay** x264.mp4

digest = hash(F)

云存储为了节约存储空间，替换了文件：

F’ = 爱丽丝梦游仙境 **1080p HDTV** x264.mp4

F’

验证： digest =?= hash(F’)

如果不相等 => 文件被替换

Alice用**杂凑函数**计算并且记录文件 F 的**摘要值**，如果提取回来的文件 F‘ 和摘要不同，说明云存储替换了文件

一个现实问题：如何审计云存储篡改了数据?

Alice

Cloud

F = 爱丽丝梦游仙境 **2160p BluRay** x264.mp4

F_1	F_2	F_3	.....	F_N

digest = hash(F)

云存储为了节约存储空间，替换了文件：

F’ = 爱丽丝梦游仙境 **1080p HDTV** x264.mp4

F’_1	F’_2	F’_3	.....	F’_N

只想看片头和片尾

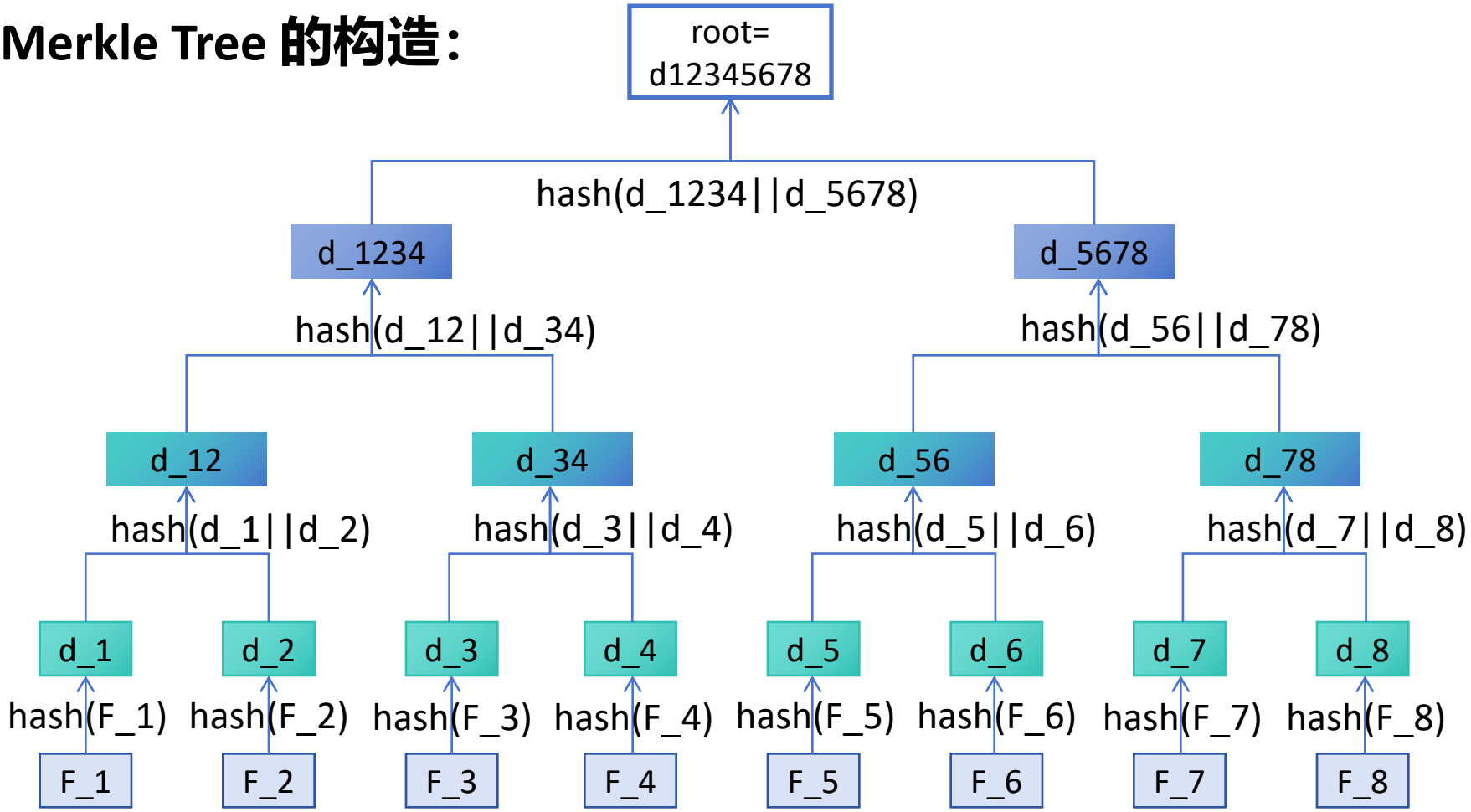
F’_1	F’_N

问题：需要Alice下载整个文件才能完成审计！

思考：比如电影被分为N个片段，怎么帮助Alice验证自己想看的片段（片头、片尾）是正确的?

认证数据结构——默克尔树(Merkle Tree)

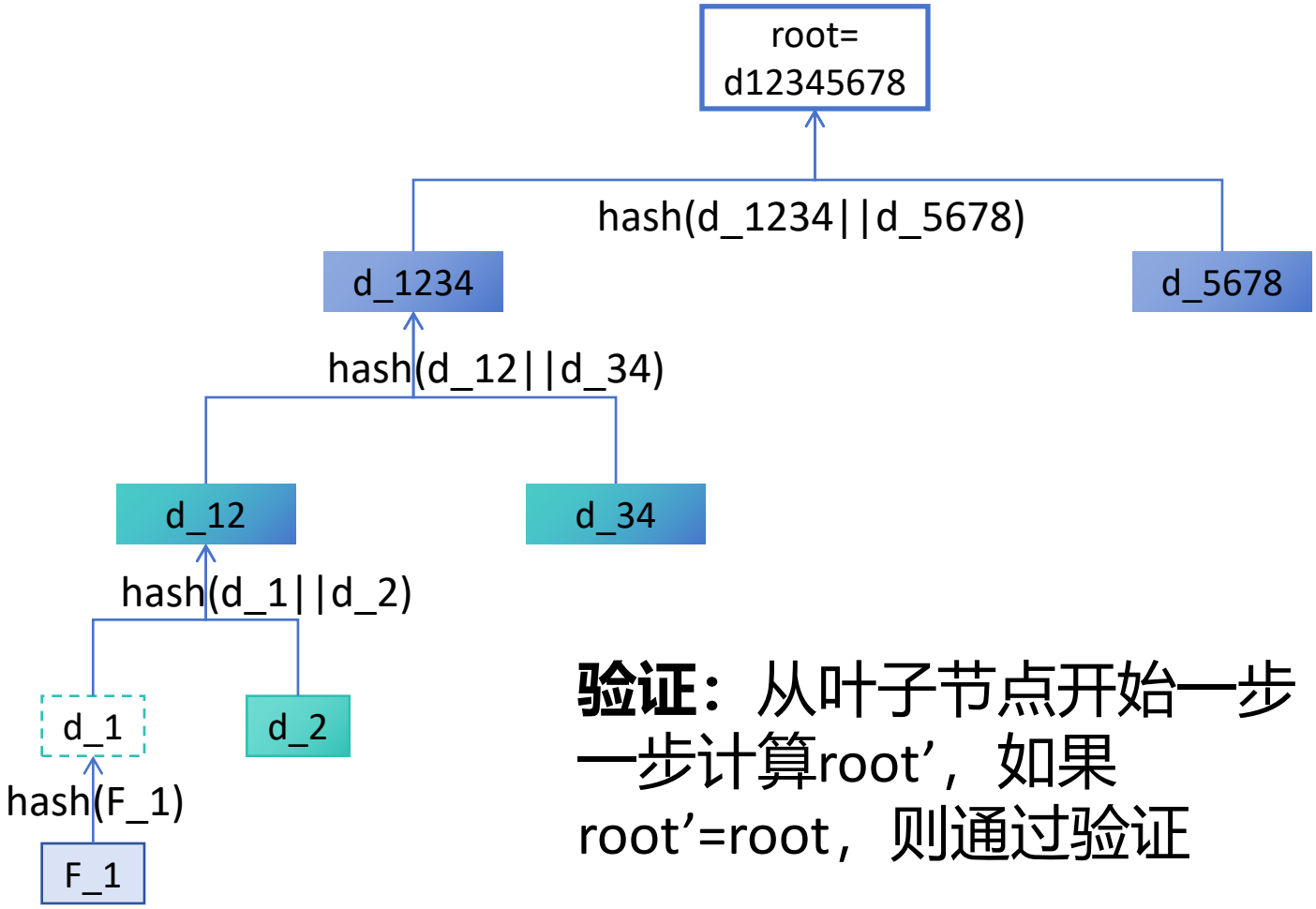
Merkle Tree 的构造：



认证数据结构——默克尔树(Merkle Tree)

叶子节点内容的证明：

如果是第一个位置

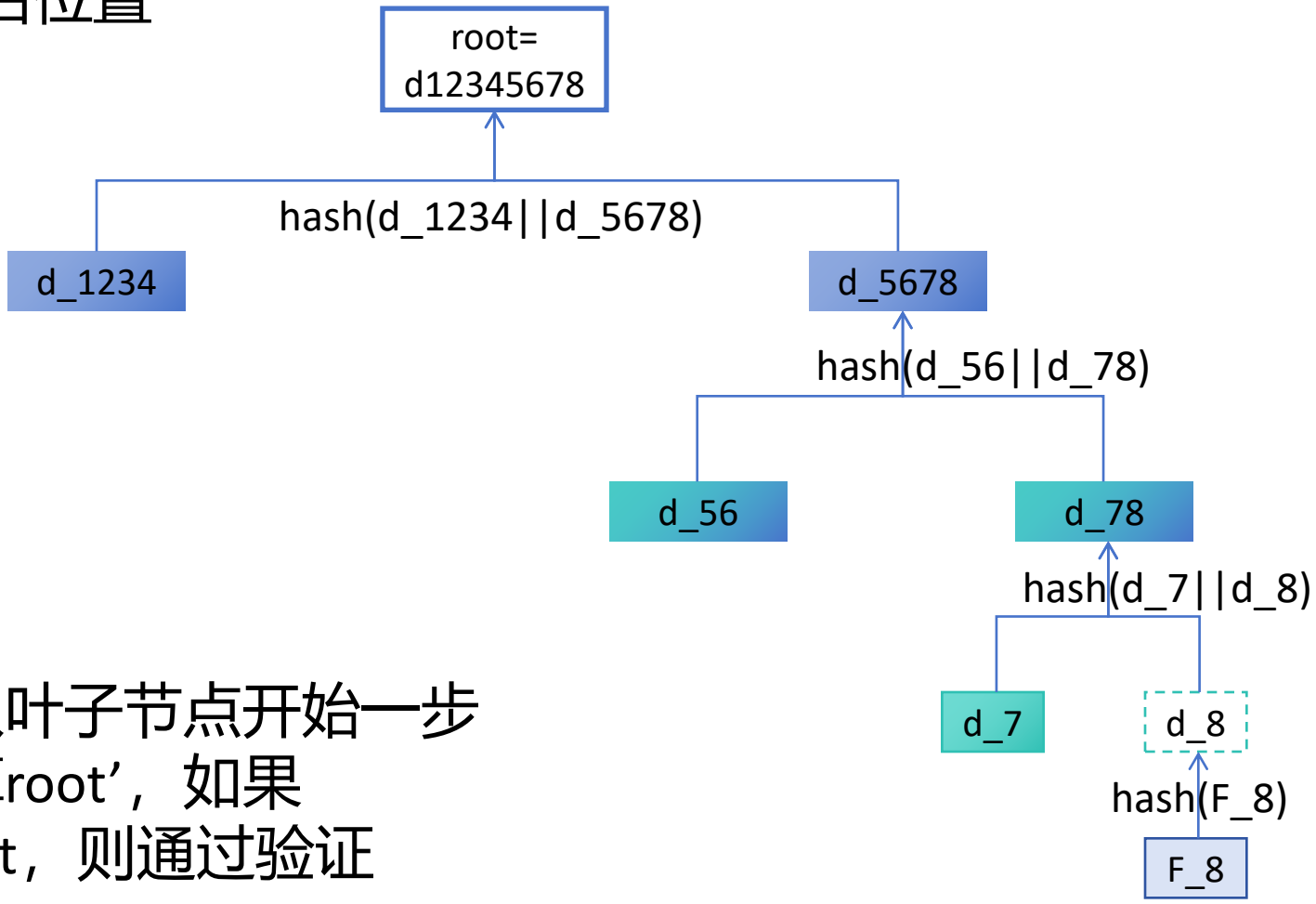


验证：从叶子节点开始一步一步计算root’，如果root’=root，则通过验证

认证数据结构——默克尔树(Merkle Tree)

叶子节点内容的证明：

如果是最后位置

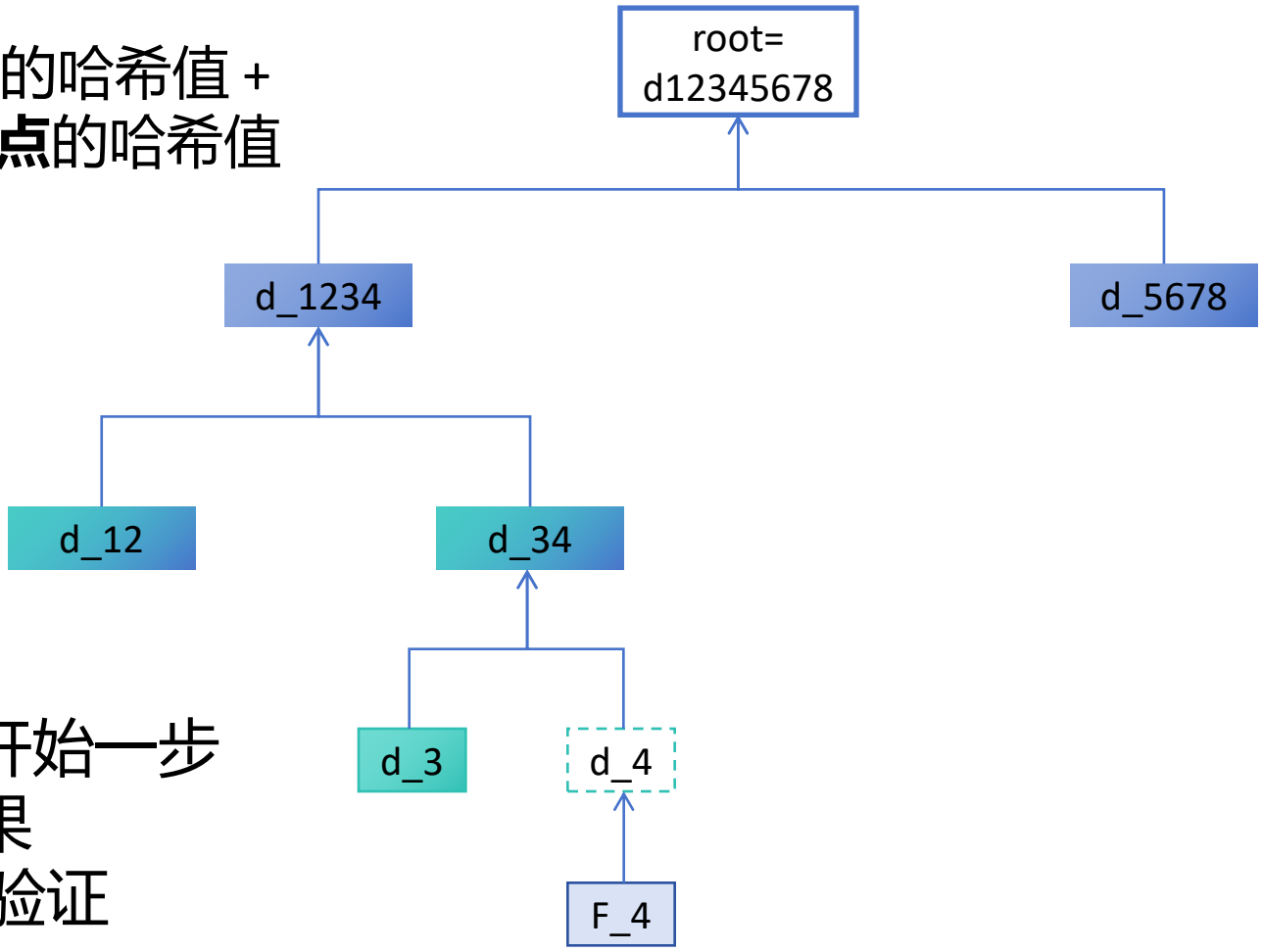


验证：从叶子节点开始一步一步计算root’，如果root’=root，则通过验证

认证数据结构——默克尔树(Merkle Tree)

可以扩展到任意第 i 个叶子节点的内容证明

- 先找到从叶子到root的路径
- **路径上所有节点**的哈希值 + **路径上所有子节点**的哈希值

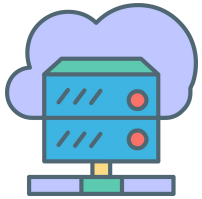


验证：从叶子节点开始一步一步计算root’，如果root’=root，则通过验证

认证数据结构——默克尔树(Merkle Tree)

Alice

Cloud



F = 爱丽丝梦游仙境 **2160p BluRay** x264.mp4

F_1	F_2	F_3	.....	F_N

root = Merkle-Tree(F)

云存储为了节约存储空间，替换了文件：

F’ = 爱丽丝梦游仙境 **1080p HDTV** x264.mp4

F’_1	F’_2	F’_3	.....	F’_N

给我片头和它完整性的证明

F’_1

Merkle proof = <到root路径上所有的哈希值 + 路径上所有子节点的哈希值>

验证： root’=?=root

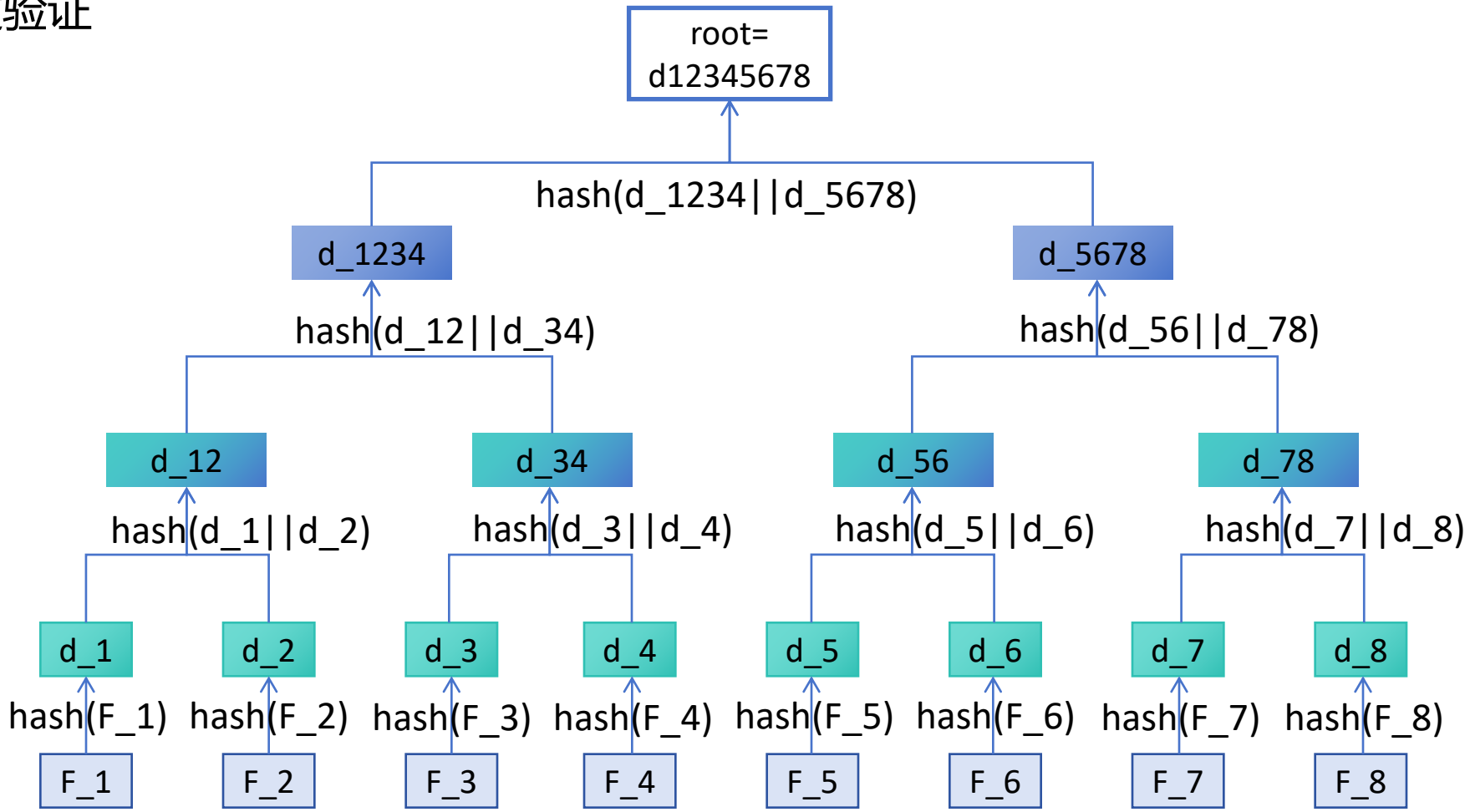
root不相等 => 片头被替换

利用Merkle tree将数据存储
在不可信的存储环境，
防止数据的篡改。

认证数据结构——默克尔树(Merkle Tree)

Merkle Tree 的安全性：

位置绑定性（向量承诺）—— 对一个承诺了N个位置的Merkle tree root，对每个位置 i，攻击者难以可行地找到不同的 F_i 和 F_i'，使得 F_i 和 F_i' 都通过验证



认证数据结构——默克尔树(Merkle Tree)

Merkle Tree 的安全性：

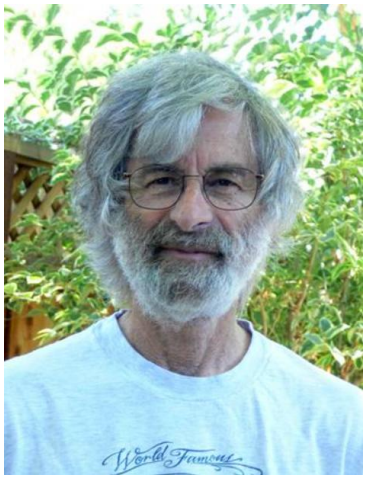
位置绑定性（向量承诺）—— 对一个承诺了N个位置的Merkle tree root，对每个位置 i，攻击者难以可行地找到不同的 F_i 和 F_i’，使得 F_i 和 F_i’ 都通过验证

思考：为什么？能够给出证明么？

提示：如果位置绑定性被敌手攻破，哈希函数的抗碰撞性会怎样？

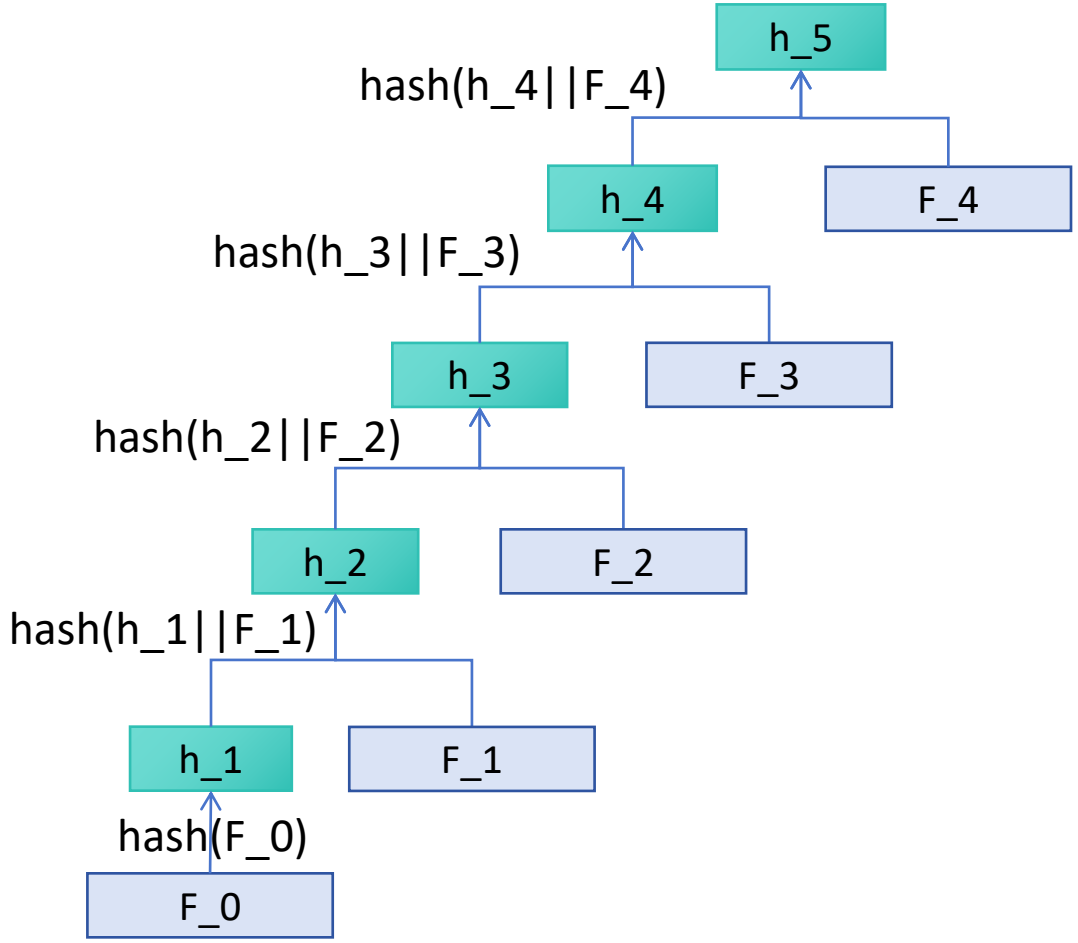
“An algorithm without a proof is a conjecture, it's not a theorem”

—— Leslie Lamport



认证数据结构——哈希链(Hash Chain)

Merkle Tree 的一个特例 —— 哈希链(Hash Chain):



如果 h_5 固定，是否有多项式时间的攻击者可以替换 F_i 呢?

- 没有!

为什么?

- 如果存在多项式算法B，将 F_i 替换成了 F_i'，并且 h_5 不改变。那么一定存在多项式算法A，成功破坏哈希函数的抗碰撞性！ **<= 安全性归约**

用处：

- 链表头不断添加新数据，当表头难以被替换时，形成的日志也是防篡改

认证数据结构：小结

默克尔树(Merkle tree)

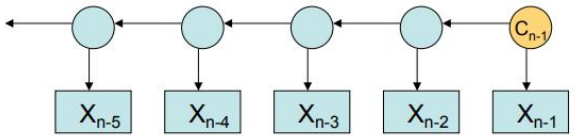
- 类似二叉树结构，每个节点的值都是两个子节点的值经过哈希运算后的结果
- 树根是对所有N个叶子节点的向量承诺
- 证明第 i 个叶子节点需要传输 O(logN) 个哈希值

哈希链(Hash chain)

- 类似链表的结构，可以在当前表头后不断添加新节点
- 链头的哈希值是对链中所有位置内容的承诺

Hash chain log

- Existing approach [Kelsey,Schneier]
  - $C_n=H(C_{n-1} \| X_n)$
  - Logger signs C_n

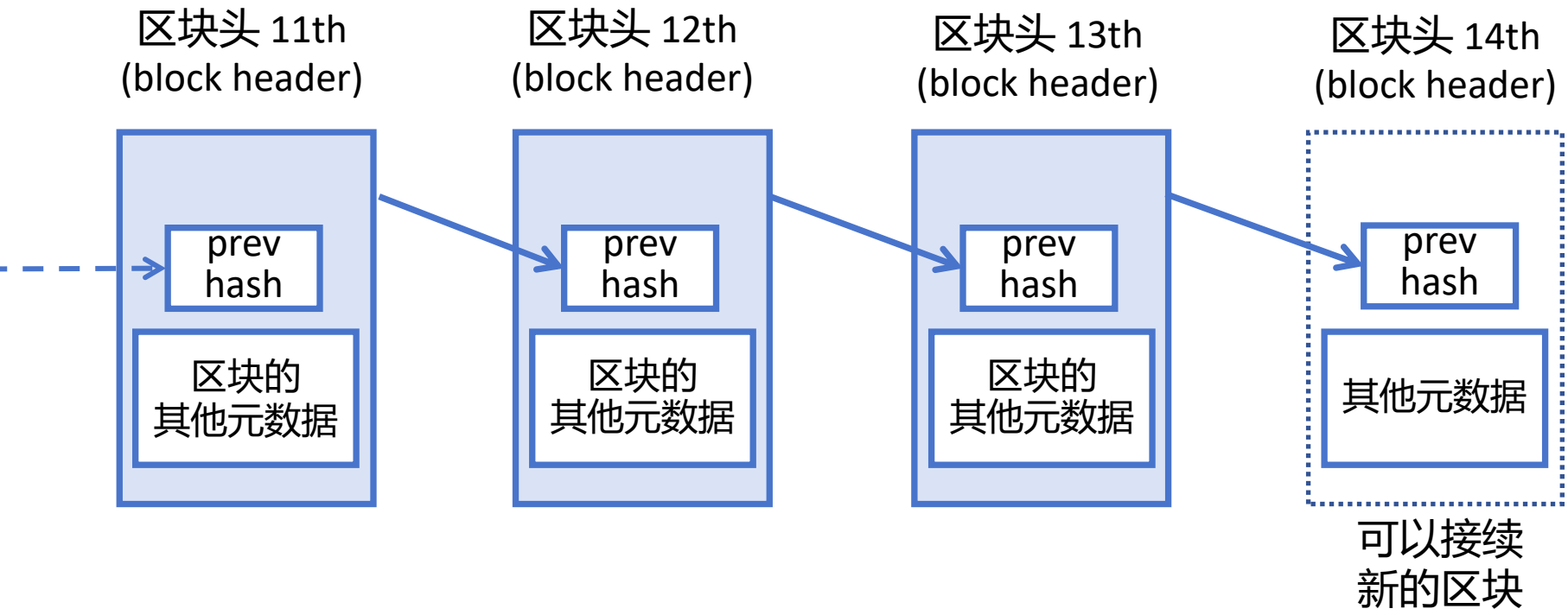


区块链和区块结构

区块链

区块链

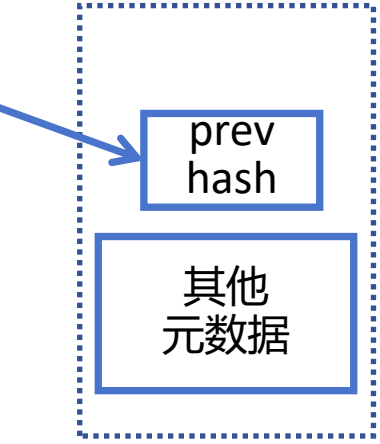
- 区块头 (block header) 的哈希链
- 使用**哈希链**技术实现的 **仅允许续接的线性日志** (append-only linear log)



区块链

区块头结构——以比特币为例

- 每个Bitcoin区块头中有6个字段的元数据
- 一个pre hash、五个字段的其他元数据



Version	区块的版本号，和当前使用的软件版本有关	4字节
Pre_hash	前一个区块头的256比特hash值	32字节
Merkle_root	本区块中所有交易构造的默克尔树的树根hash值	32字节
Timestamp	UNIX格式时间戳	4字节
Difficulty	压缩形式的PoW目标难度	4字节
Nonce	32比特的数字，计算PoW时使用	4字节

区块链

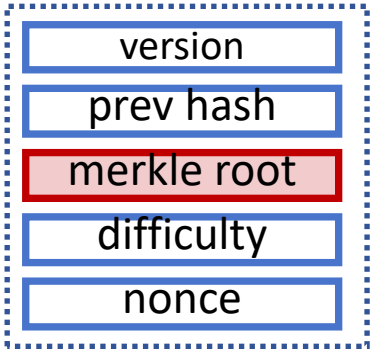
区块头结构——以比特币为例

- 区块头哈希值的计算
  - SHA256(SHA256(Block_Header))，需要特别注意：
    - Block_Header是Version、Pre_hash、Merkle_root、Timestamp、Difficulty、Nonce按照**小端尾序编码成hex后串联在一起**的
    - Block_Header在进行**SHA256计算前要解码为二进制字符串**
    - **哈希值要补0**，补满 256 bits
    - 以比特币的第125552个区块为例

```
>>> import hashlib
>>> from binascii import unhexlify, hexlify
>>> header_hex = ("01000000" +
 "81cd02ab7e569e8bcd9317e2fe99f2de44d49ab2b8851ba4a308000000000000" +
 "e320b6c2fffc8d750423db8b1eb942ae710e951ed797f7affc8892b0f1fc122b" +
 "c7f5d74d" +
 "f2b9441a" +
 "42a14695")
>>> header_bin = unhexlify(header_hex)
>>> hash = hashlib.sha256(hashlib.sha256(header_bin).digest()).digest()
>>> hexlify(hash).decode("utf-8")
'1dbd981fe6985776b644b173a4d0385ddc1aa2a829688d1e0000000000000000'
>>> hexlify(hash[::-1]).decode("utf-8")
'00000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d'
```

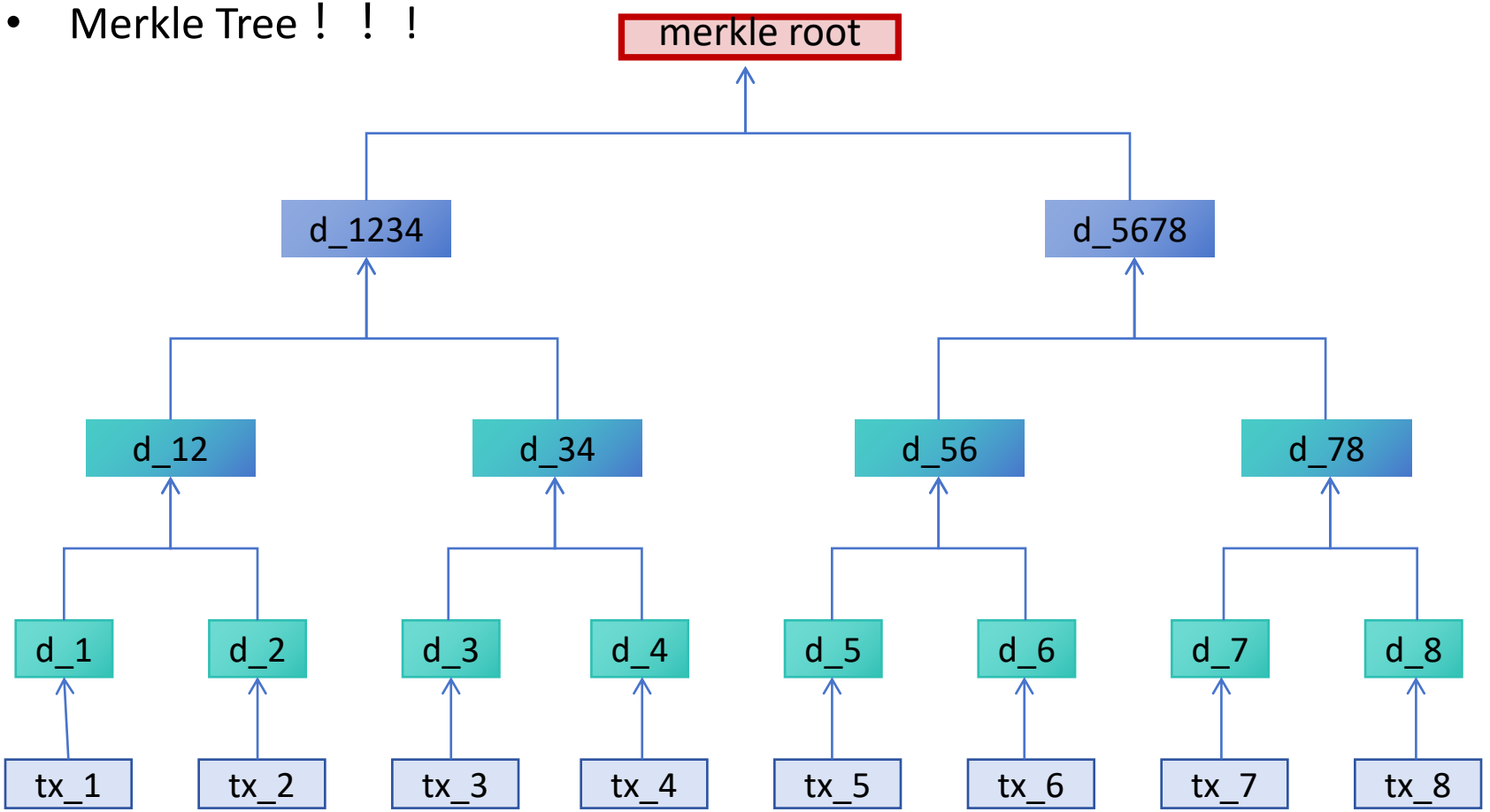
00000000000008a3a41b85b8b29ad444def299fee21793cd8b9e567eab02cd81

区块链



区块

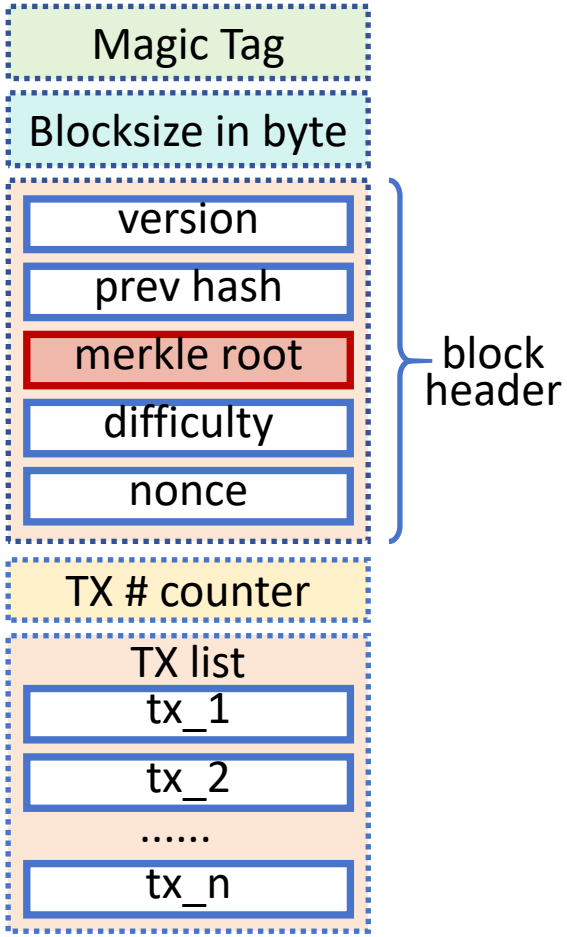
- 区块头 (block header) 只包含了一些元数据，交易等数据怎么办?
  - Merkle Tree！！！



区块链

区块

- 最终的区块结构

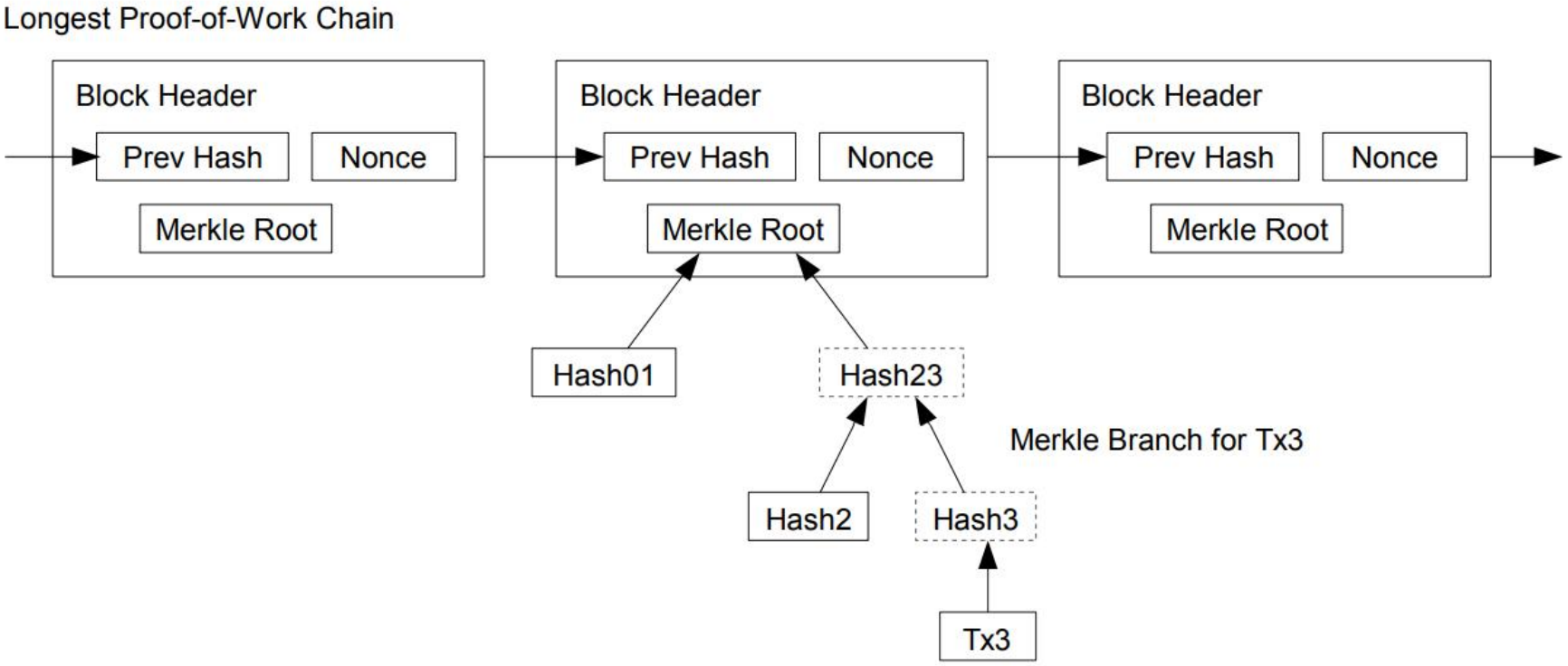


Magic Tag	在比特币的主网中，永远设置为0xD9B4BEF9，标记为区块链主网的区块数据	4字节
Block size	本区块还剩余的字节数量	4字节
Block header	Version、Pre_hash、Merkle_root、Timestamp、Difficulty、Nonce等 6个项目	80字节
TX # counter	可变长度的正整数，表示区块中交易的数量	1-9字节
TX list	本区块中交易的列表	-

简化支付验证 (SPV)

无法下载和存储所有区块的轻量节点，如何验证某笔交易存在于区块链：

- **正确的区块头 + 有效的 Merkle tree proof**

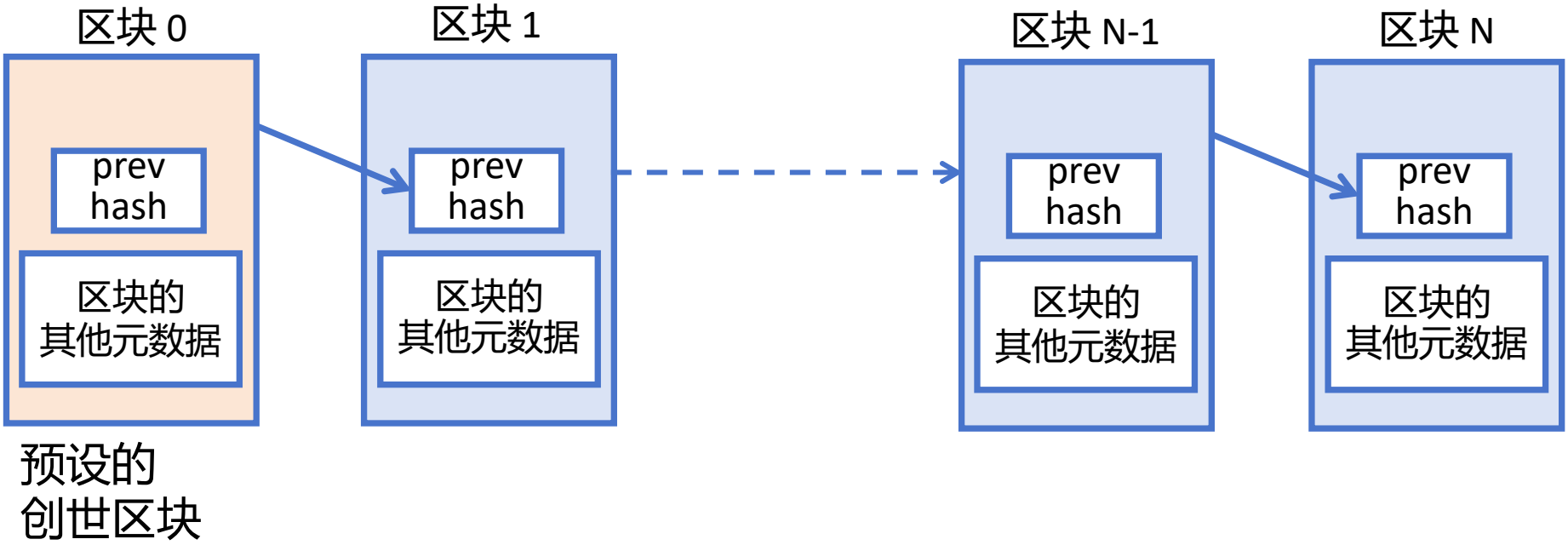


创世区块解读

创世区块

区块链总有第一个区块，它从哪里来?

- 创世区块 (**Genesis**):
  - Satoshi Nakamoto “硬编码” (hardcoded) 在比特币主网的协议中，在比特币所有的发行软件中都是固定的
  - 所有比特币主网的参与节点，通过创世区块对初始状态达成一致



创世区块

Nakamoto 作为设计者在 Genesis 里做了什么?

- 有给自己写一笔巨额财富么?

```
00000000   01 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00000010   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00000020   00 00 00 00 3B A3 ED FD   7A 7B 12 B2 7A C7 2C 3E   ....;£íýz{.²zÇ,>
00000030   67 76 8F 61 7F C8 1B C3   88 8A 51 32 3A 9F B8 AA   gv.a.È.Ã^ŠQ2:Ÿ¸ª
00000040   4B 1E 5E 4A 29 AB 5F 49   FF FF 00 1D 1D AC 2B 7C   K.^J)«_Iÿÿ...¬+|
00000050   01 01 00 00 00 01 00 00   00 00 00 00 00 00 00 00   ................
00000060   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00000070   00 00 00 00 00 00 FF FF   FF FF 4D 04 FF FF 00 1D   ......ÿÿÿÿM.ÿÿ..
00000080   01 04 45 54 68 65 20 54   69 6D 65 73 20 30 33 2F   ..EThe Times 03/
00000090   4A 61 6E 2F 32 30 30 39   20 43 68 61 6E 63 65 6C   Jan/2009 Chancel
000000A0   6C 6F 72 20 6F 6E 20 62   72 69 6E 6B 20 6F 66 20   lor on brink of
000000B0   73 65 63 6F 6E 64 20 62   61 69 6C 6F 75 74 20 66   second bailout f
000000C0   6F 72 20 62 61 6E 6B 73   FF FF FF FF 01 00 F2 05   or banksÿÿÿÿ..ò.
000000D0   2A 01 00 00 00 43 41 04   67 8A FD B0 FE 55 48 27   *....CA.gŠý°þUH'
000000E0   19 67 F1 A6 71 30 B7 10   5C D6 A8 28 E0 39 09 A6   .gñ¦q0·.\Ö¨(à9.¦
000000F0   79 62 E0 EA 1F 61 DE B6   49 F6 BC 3F 4C EF 38 C4   ybàê.aÞ¶Iö¼?Lï8Ä
00000100   F3 55 04 E5 1E C1 12 DE   5C 38 4D F7 BA 0B 8D 57   óU.å.Á.Þ\8M÷º..W
00000110   8A 4C 70 2B 6B F1 1D 5F   AC 00 00 00 00            ŠLp+kñ._¬....
```

创世区块

Nakamoto 作为设计者在 Genesis 里做了什么?

01000000 - **version**
0000000000000000000000000000000000000000000000000000000000000000 - **prev hash**
3BA3EDFD7A7B12B27AC72C3E67768F617FC81BC3888A51323A9FB8AA4B1E5E4A - **merkle root**
29AB5F49 - **timestamp**
FFFF001D - **difficuty (0x00ffff0000000000000000000000000000000000000000000000000000)**
1DAC2B7C - **nonce**
01 - **number of transactions**

- 只写了一笔交易!
- 在一般的区块中，这笔交易奖励给矿工，但是根据Nakamoto设计的共识规则，创世区块的这笔交易不作为可转账的奖励交易
  - => Nakamoto 在 Genesis 区块里给自己奖励了 0 BTC!

创世区块

Nakamoto 作为设计者在 Genesis 里做了什么?

01000000 - **version**
0000000000000000000000000000000000000000000000000000000000000000 - **prev hash**
3BA3EDFD7A7B12B27AC72C3E67768F617FC81BC3888A51323A9FB8AA4B1E5E4A - **merkle root**
29AB5F49 - **timestamp**
FFFF001D - **difficuty (0x00ffff0000000000000000000000000000000000000000000000000000)**
1DAC2B7C - **nonce**
01 - **number of transactions**

- 只写了一笔交易!
- 除了给自己 0 BTC，还给大家留了一段话:
  “The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”

  2009年1月3日，财政大臣正处于实施第二轮银行紧急援助的边缘

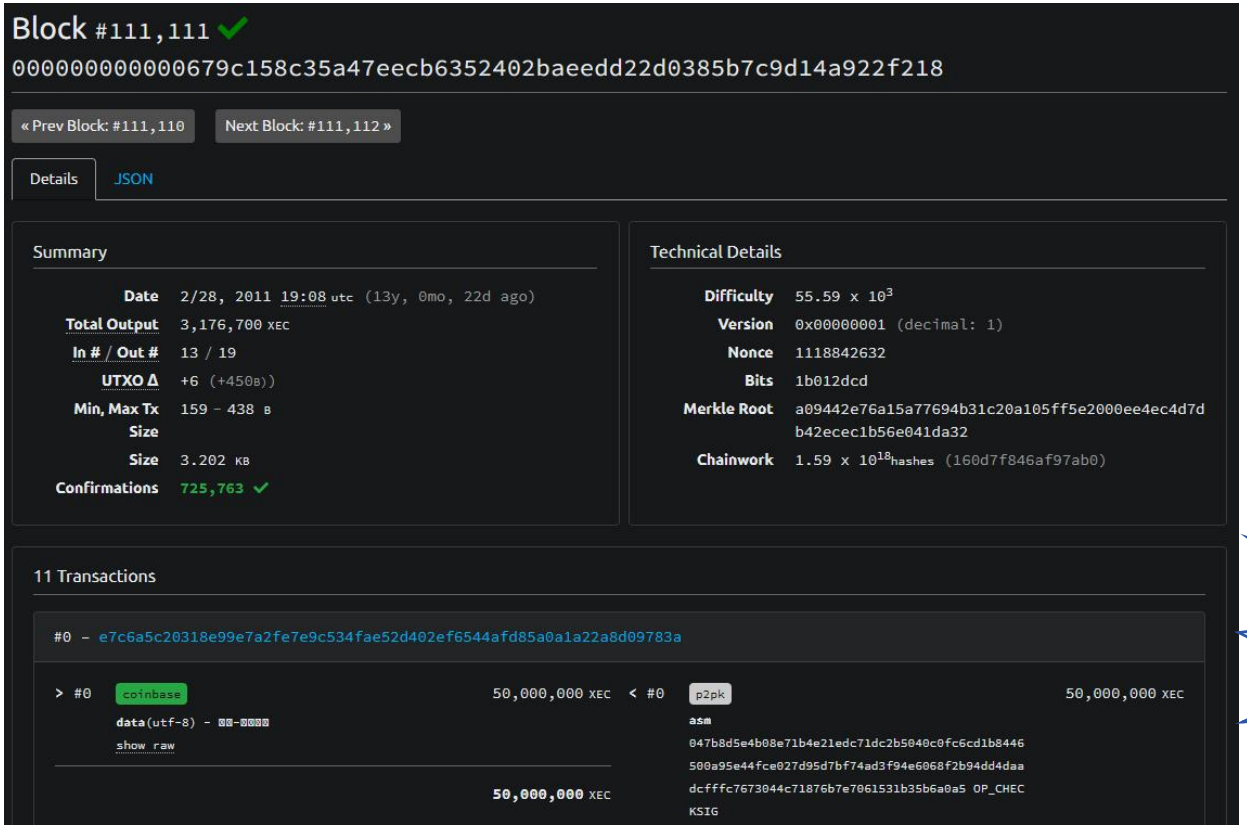
  (《泰晤士报》当天的头条新闻标题)

交易和生成交易

生成交易（coinbase transaction）

创世区块里包含了0个BTC，那比特币系统中的 BTC 从哪里来?

每个区块的第一笔交易都是特殊的，称为**生成交易（coinbase transaction）**，奖励给该区块的生成者指定的比特币地址(或公钥)！



区块111111奖励50个BTC给公钥:

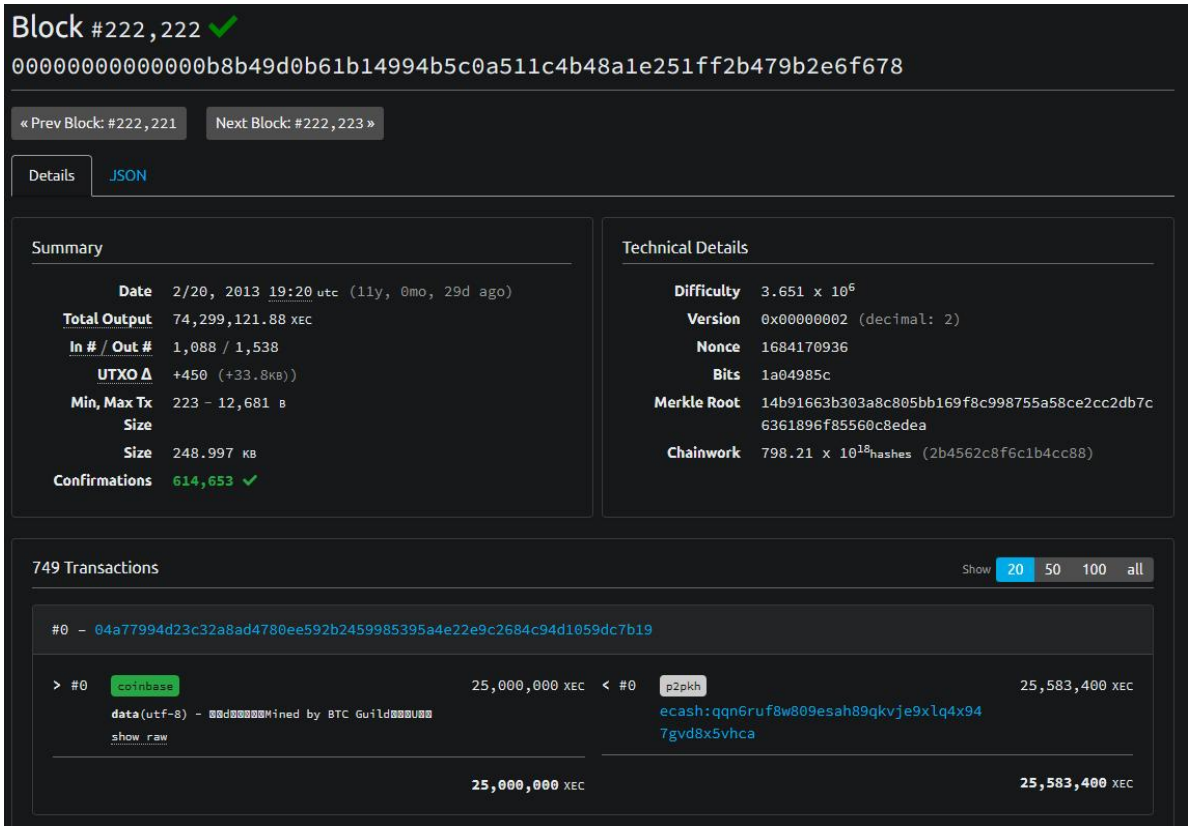
047b8d5e4b08e71b4e21e
dc71dc2b5040c0fc6cd1b84
46500a95e44fce027d95d7
bf74ad3f94e6068f2b94dd4
daadcfffc7673044c71876b
7e7061531b35b6a0a5

可以仅从公钥推导出地址么?

生成交易（coinbase transaction）

创世区块里包含了0个BTC，那比特币系统中的 BTC 从哪里来?

每个区块的第一笔交易都是特殊的，称为**生成交易（coinbase transaction）**，奖励给该区块的生成者指定的比特币地址(或公钥)！



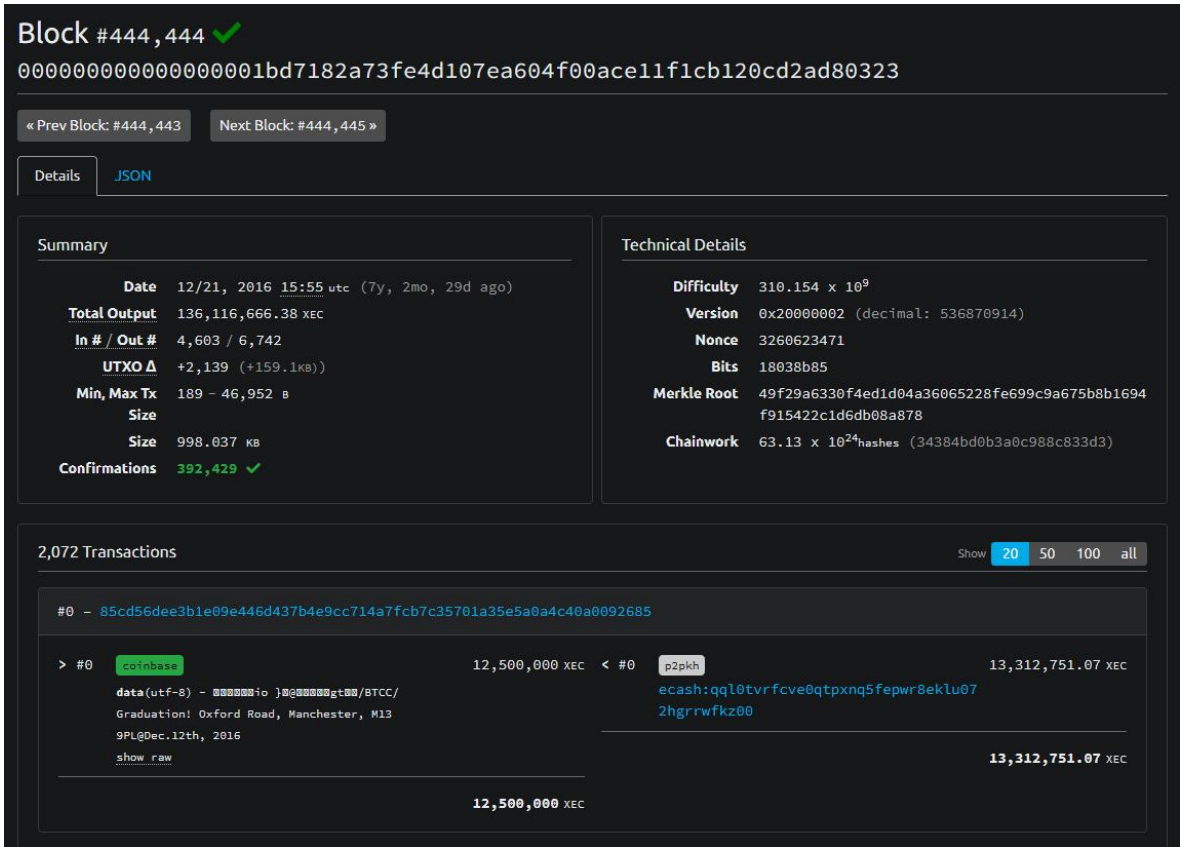
区块222222奖励25个BTC给地址:
ecash:qqn6ruf8w809esah89qkvje9xlq4x947gvd8x5vhca

(或者Base58Check编码
14cZMQk89mRYQkDEj8Rn25AnGoBi5H6uer)

生成交易（coinbase transaction）

创世区块里包含了0个BTC，那比特币系统中的 BTC 从哪里来?

每个区块的第一笔交易都是特殊的，称为**生成交易（coinbase transaction）**，奖励给该区块的生成者指定的比特币地址(或公钥)！



区块444444奖励12.5个BTC给地址:
ecash:qql0tvrfcve0qtpxnq5fepwr8eklu072hgrrwfkz00

(或者Base58Check编码
16juDfzuERhCPk9csU4tevBDwnkLUqP3CF)

可以仅从地址推导出公钥么?

生成交易（coinbase transaction）

创世区块里包含了0个BTC，那比特币系统中的 BTC 从哪里来?

每个区块的第一笔交易都是特殊的，称为**生成交易（coinbase transaction）**，奖励给该区块的生成者指定的比特币地址(或公钥)！

区块111111：
50个BTC

区块222222：
25个BTC

区块444444：
12.5个BTC

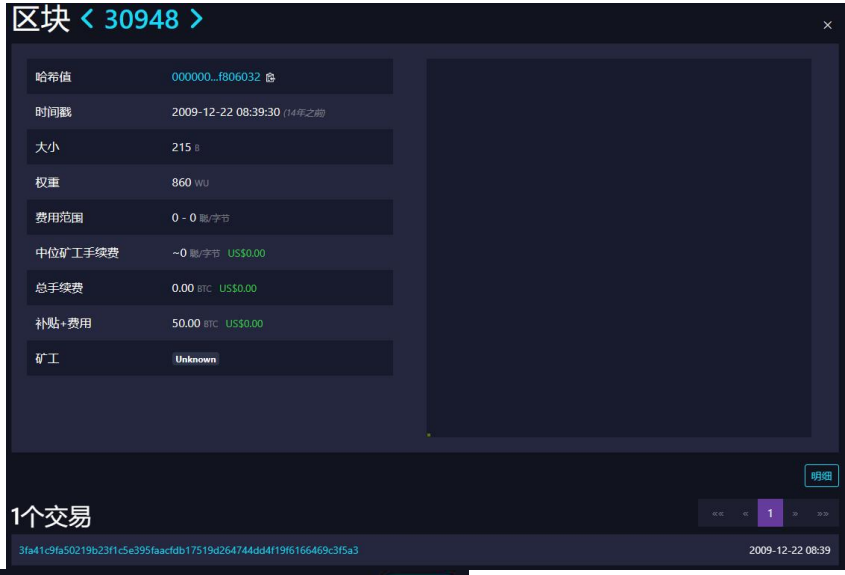
区块666666：
6.25个BTC

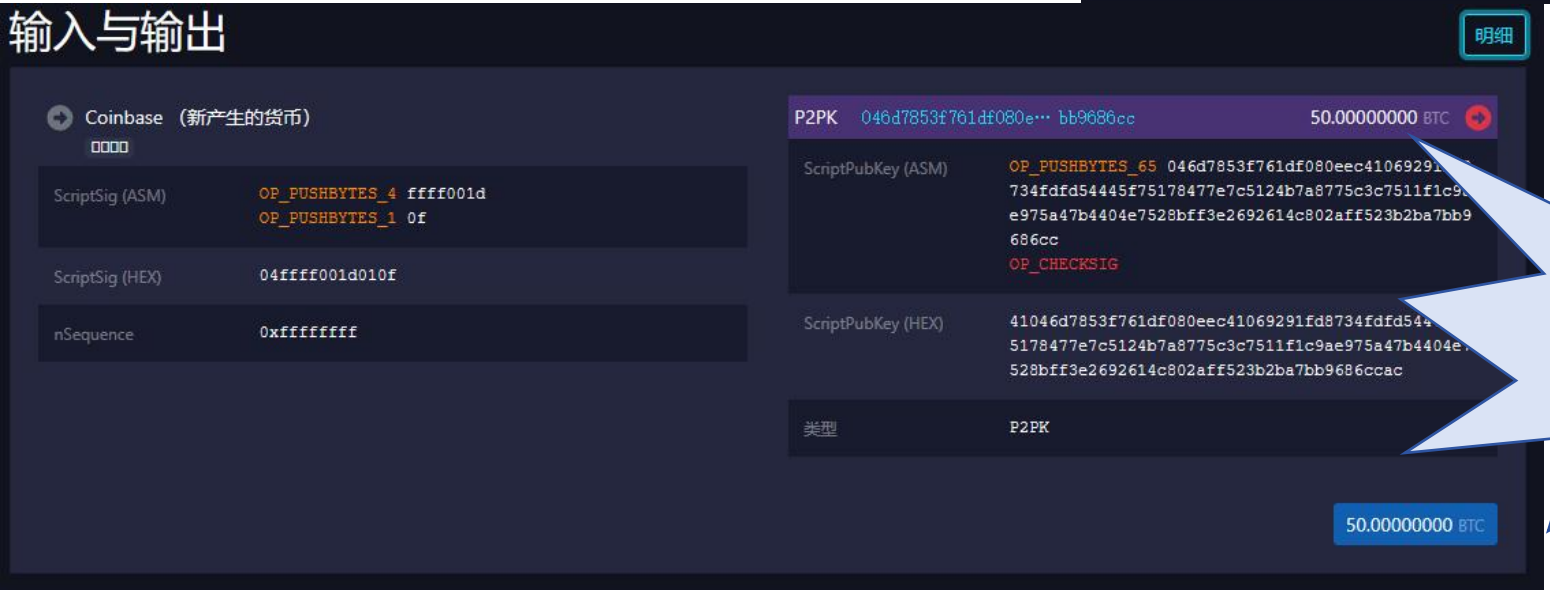
Nakamoto的规定:
每当比特币区块链产生210,000个区块时，区块奖励将减半。
总量最多2100个BTC。

未花费输出(UTXO)和交易

如何继续使用 coinbase 奖励的 BTC 呢?

比如区块30948，通过哈希值为3fa41c9fa50219b23f1c5e395faacfdb17519d264744dd4f19f6166469c3f5a3 的 coinbase交易奖励给了公钥046d7853f761df080eec41069291fd8734fdfd54445f75178477e7c5124b7a8775c3c7511f1c9ae975a47b4404e7528bff3e2692614c802aff523b2ba7bb9686cc 总计50个BTC



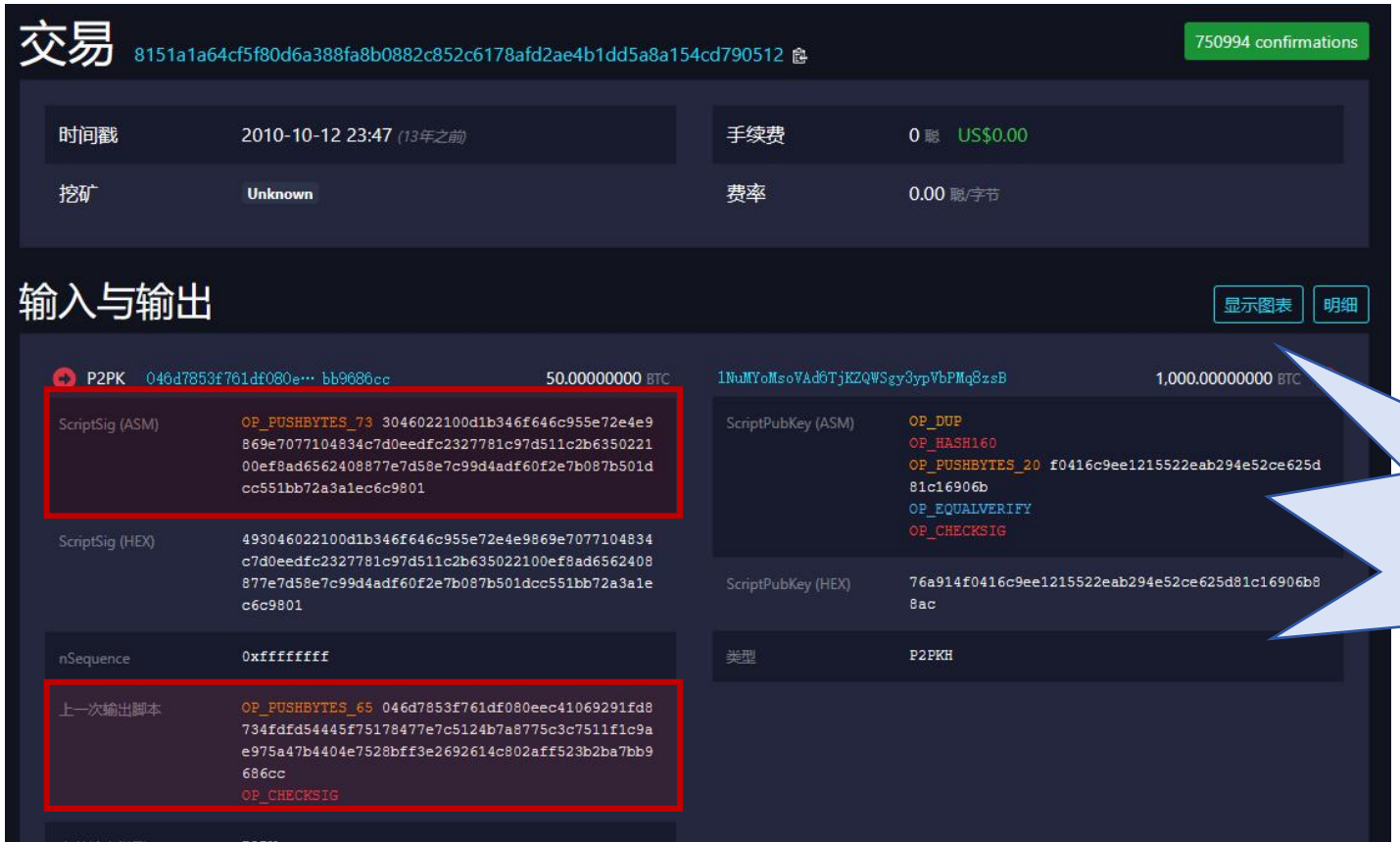


交易3fa41c...
有一笔 50BTC的
未花费输出
(**UTXO**)

未花费输出(UTXO)和交易

如何继续使用 coinbase 奖励的 BTC 呢?

在53787个区块之后，交易3fa41c...的 UTXO 被使用，
和总计20个coinbase交易 的未花费输出一起，通过哈希值为
8151a1a64cf5f80d6a388fa8b0882c852c6178afd2ae4b1dd5a8a154cd790512
的交易转账给了地址 1NuMYoMsoVAd6TjKZQWSgy3ypVbPMq8zsB



签名

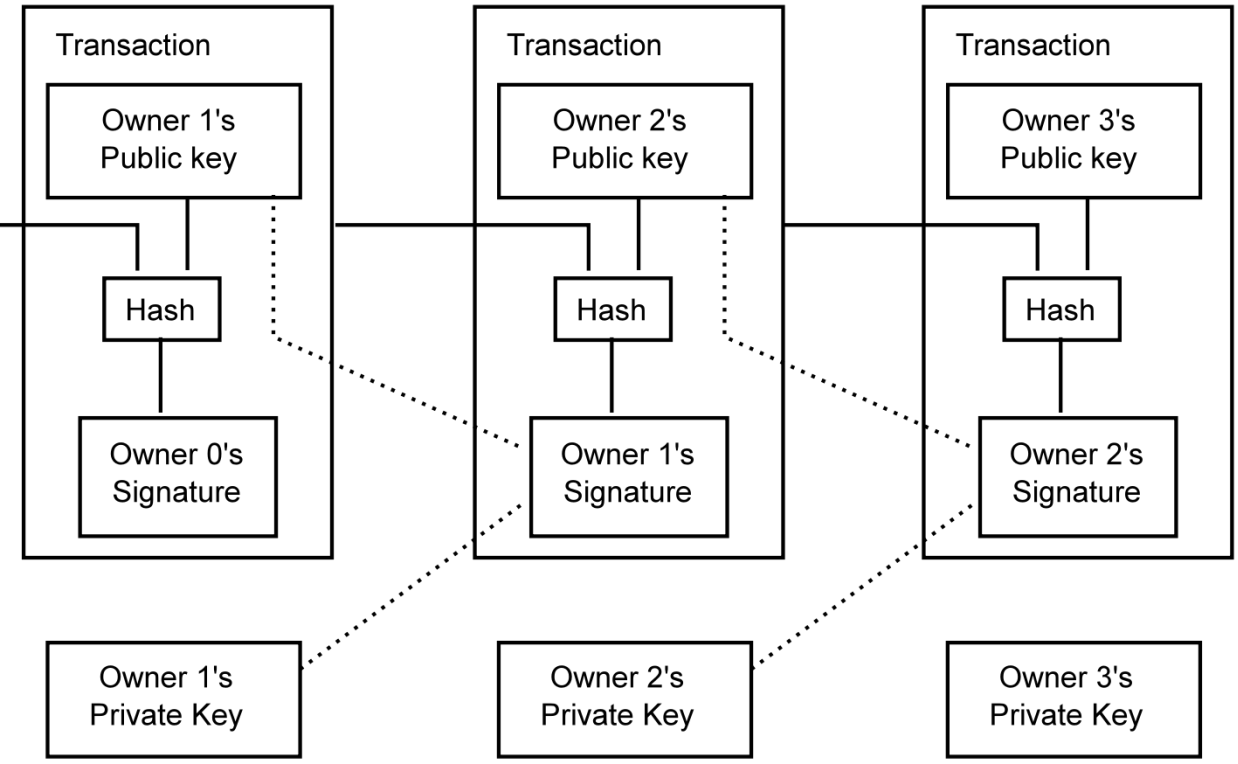
公钥

使用者提供了针对公钥 046d78...有效的数字签名

未花费输出(UTXO)和交易

如何继续使用 coinbase 奖励的 BTC 呢?

未花费输出的公钥 + 签名 → 如果验证成功，转入新的地址！之前的**未花费输出(UTXO)**标记为**已花费**，产生全新的未花费输出(UTXO)



被签名的消息是所有交易数据的hash值；因此签名无法被重用！

多输入多输出交易的签名

当一笔交易有多个输入、多个输出，该怎么签名？对哪一部分签名？

- **输入的控制情况：**
  - 所有输入的私钥由一个实体控制
  - 有些输入的私钥由不同实体控制
- **输出的固定与否：**
  - 固定的收款地址
  - 不固定收款地址

Name
SIGHASH_ALL
SIGHASH_NONE
SIGHASH_SINGLE
SIGHASH_ANYONECANPAY

SIGHASH_ALL: 对所有的输出都签名 => 所有的的收款地址都必须是固定的，本交易的所有收款地址都必须是固定的

SIGHASH_NONE: 不对任何输出签名，输出可以任意指定 => 完全不关心收款地址，本交易的收款地址可以是任意的

SIGHASH_SINGLE: 只对一个特定的输出签名，其他的输出可以任意指定 => 只关心一个特定的收款地址，这个特定地址必须是本交易的收款地址之一

SIGHASH_ANYONECANPAY: 只需对自己的输入签名 => 不关心其他的输入来源，可以和SIGHASH_ALL、SIGHASH_NONE、SIGHASH_SINGLE之一共同使用

多输入多输出交易的签名

当一笔交易有多个输入、多个输出，该怎么签名？对哪一部分签名？

- **SIGHASH_ALL、SIGHASH_NONE、SIGHASH_SINGLE、SIGHASH_ANYONECANPAY 有六种组合方式**

Flag	Functional Meaning
SIGHASH_ALL	Sign all inputs and outputs
SIGHASH_NONE	Sign all inputs and no output
SIGHASH_SINGLE	Sign all inputs and the output with the same index
SIGHASH_ALL | ANYONECANPAY	Sign its own input and all outputs
SIGHASH_NONE | ANYONECANPAY	Sign its own input and no output
SIGHASH_SINGLE | ANYONECANPAY	Sign its own input and the output with the same index

多输入多输出交易的签名

当一笔交易有多个输入、多个输出，该怎么签名？对哪一部分签名？

- **SIGHASH_ALL、SIGHASH_NONE、SIGHASH_SINGLE、SIGHASH_ANYONECANPAY有六种组合方式**

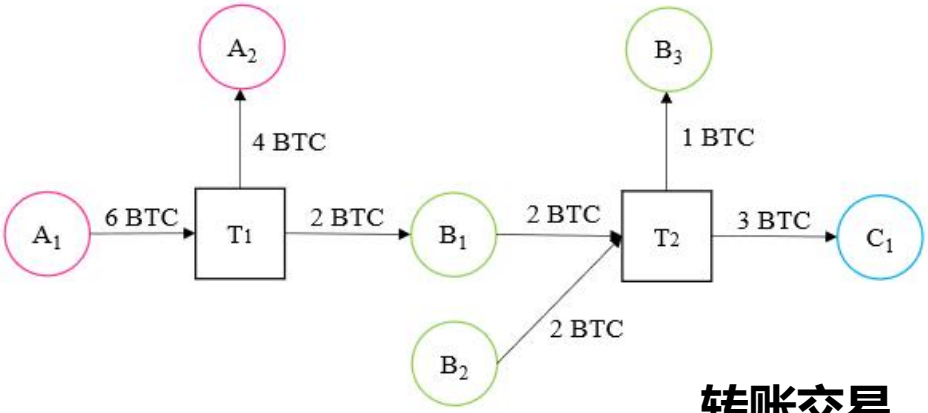
Flag	Functional Meaning
SIGHASH_ALL	Sign all inputs and outputs
SIGHASH_NONE	Sign all inputs and no output
SIGHASH_SINGLE	Sign all inputs and the output with the same index
SIGHASH_ALL | ANYONECANPAY	Sign its own input and all outputs
SIGHASH_NONE | ANYONECANPAY	Sign its own input and no output
SIGHASH_SINGLE | ANYONECANPAY	Sign its own input and the output with the same index

问题：众筹场景应该 使用哪种签名模式?

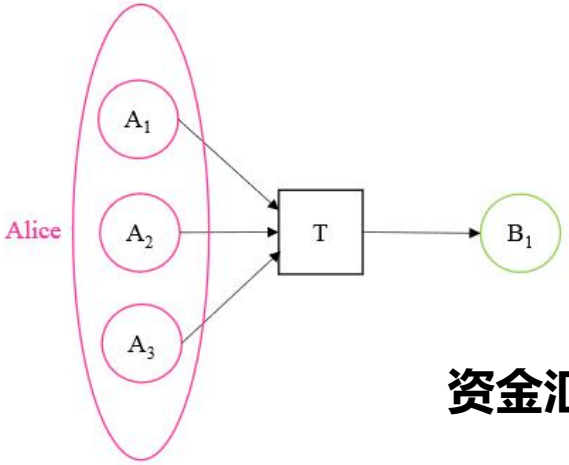
交易的可链接性和去匿名化

单输出交易：资金汇集交易，输入和输出很大可能同一个实体控制

双输出交易：很可能是普通的转账交易，两个输出交易中的某一个可能和输入由于同一个实体控制



转账交易



资金汇集交易

1. **34xp4vRoCGJym3xR7yCVPFHoCNxv4Twseo**
Balance – 248,597 BTC
A wallet address linked to Binance holding about $16.9bn worth of BTC (as of March 5, 2024) amounts to about 1.27% of the entire coins in circulation.
2. **bc1qgdjqv0av3q56jvd82tkdjpy7gdp9ut8tlqmgrpmv24sq90ecnvqqjwvw97**
Balance – 204,010 BTC
A wallet linked to Bitfinex holding about $13.9bn worth of BTC, amounting to 1.1% of the entire coins in circulation.
3. **bc1ql49ydapnjafl5t2cp9zqpjwe6pdgmxy98859v2**
Balance – 131,945 BTC
An anonymous wallet holding over $9.0bn worth of BTC tokens (0.6% of entire coins in circulation).
4. **39884E3j6KZj82FK4vcCrkUvWYL5MQaS3v**
Balance – 115,177 BTC
A Binance-linked wallet also holds $7.8bn worth of BTC.
5. **bc1qazcm763858nkj2dj986etajv6wquslv8uxwczt** Balance – 94,643 BTC
An anonymous wallet holding $6.45n worth of BTC.
6. **37XuVSEpWW4trkfmvWzegTHQt7BdktSKUs**
Balance – 94,505 BTC
An anonymous wallet also holds $6.4bn worth of BTC.
7. **1FeexV6bAHb8ybZjqQMjJrcCrHGW9sb6uF**
Balance – 79,957 BTC
An anonymous wallet holding $5.4bn worth of BTC.
8. **bc1qa5wkgaew2dkv56kfvj49j0av5nml45x9ek9hz6**
Balance – 63,370 BTC
An anonymous wallet holding $4.3bn worth of BTC.
9. **3LYJfcfHPXYJreMsASk2jkn69LWEYKzexb**
Balance – 68,200 BTC
A wallet linked to a Binance BTC reserve holding about $5.6bn worth of BTC tokens.
10. **bc1qjasf9z3h7w3jspkhtgatgpyvvzgpa2wwd2lr0eh5tx44reyn2k7sfc27a4**
Balance – 66,465 BTC
A wallet linked to stablecoin provider Tether held about $4.5bn worth of BTC.

数据来自 https://bitinfocharts.com/top-100-richest-bitcoin-addresses.html

以太坊数据结构

以太坊的账户和交易

比特币使用 **UTXO模型** =>

给某个UTXO提供有效的签名，从而将其转入其他账户或公钥 (产生新的UTXO)

以太坊使用 **账户模型** =>

维护地址和对应余额的”表格”，转账时提供对应账户的签名，在自己的余额里减掉自己的转账金额，在收款地址的余额里加入转账金额

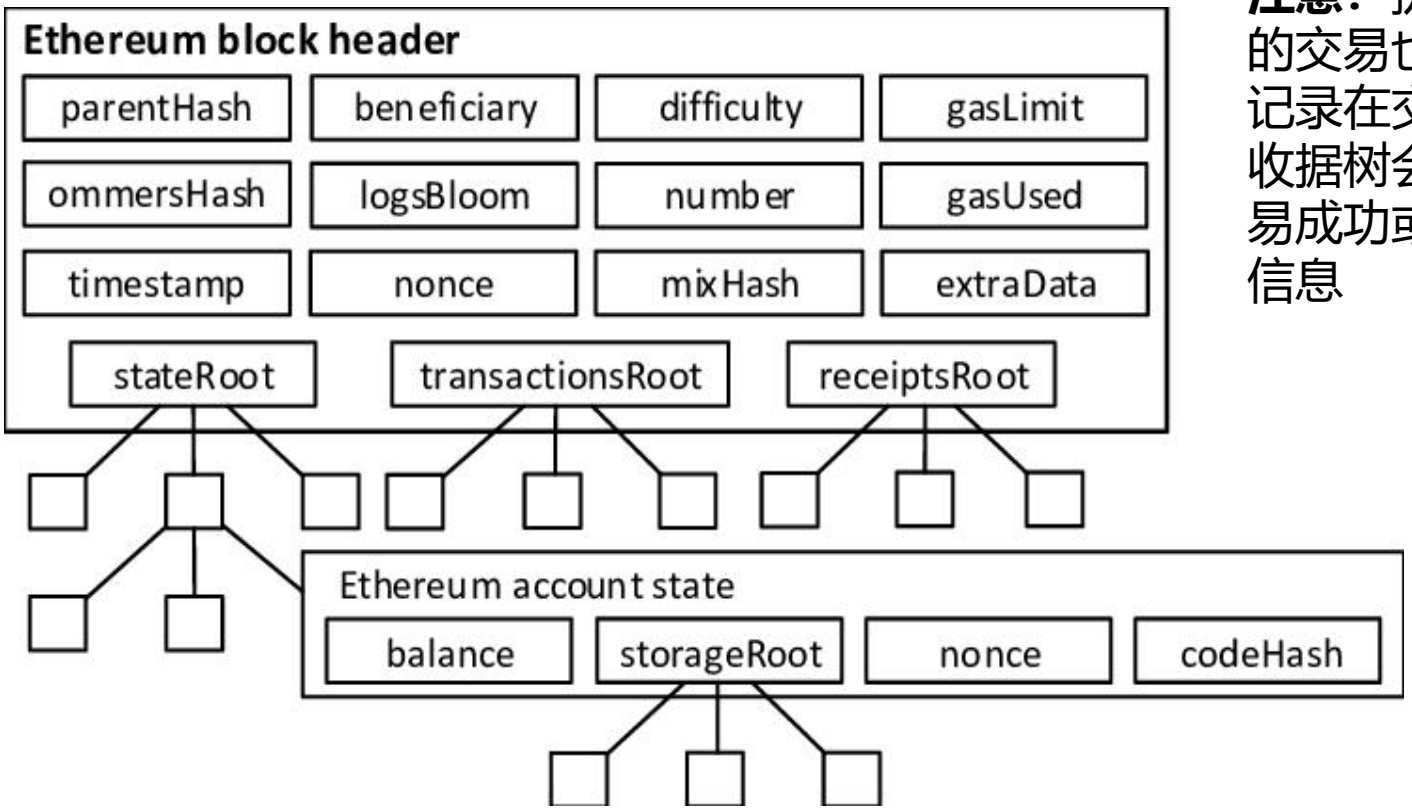
账户 (公钥的哈希值)	余额
0xF96153036f5C7ddd740DD99427BF524Aa4721C8D	100
0x2611a532A8850Dd622522735a04876DAAe70e43D	200 **- 50**
0xEBef7cEf76A38e66fa16BA09CEeCc0D21E51068a	150 **+ 50**
…	…
0x3d1909805175e705914855C14b9FC726484Ba0F0	180

转账 50 ETH

以太坊的区块结构(23年前的PoW版)

PoW时代的以太坊区块结构，和比特币的几大不同：

- 三个哈希树：分别承诺全局的**账户状态**、本区块的**交易**、本区块的交易**收据**(日志)
- 布隆过滤器：提供区块中日志事件的快速搜索方法
- 叔块指针：包含某些分叉的哈希值

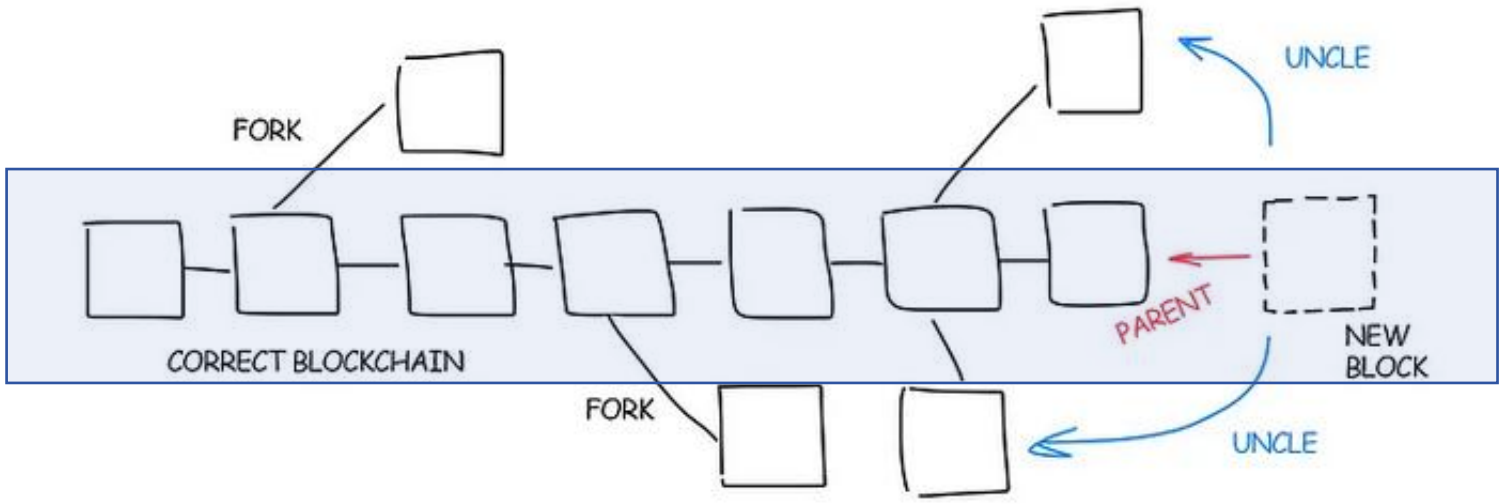


注意：执行错误的交易也可能会记录在交易树，收据树会记录交易成功或失败的信息

叔块指针

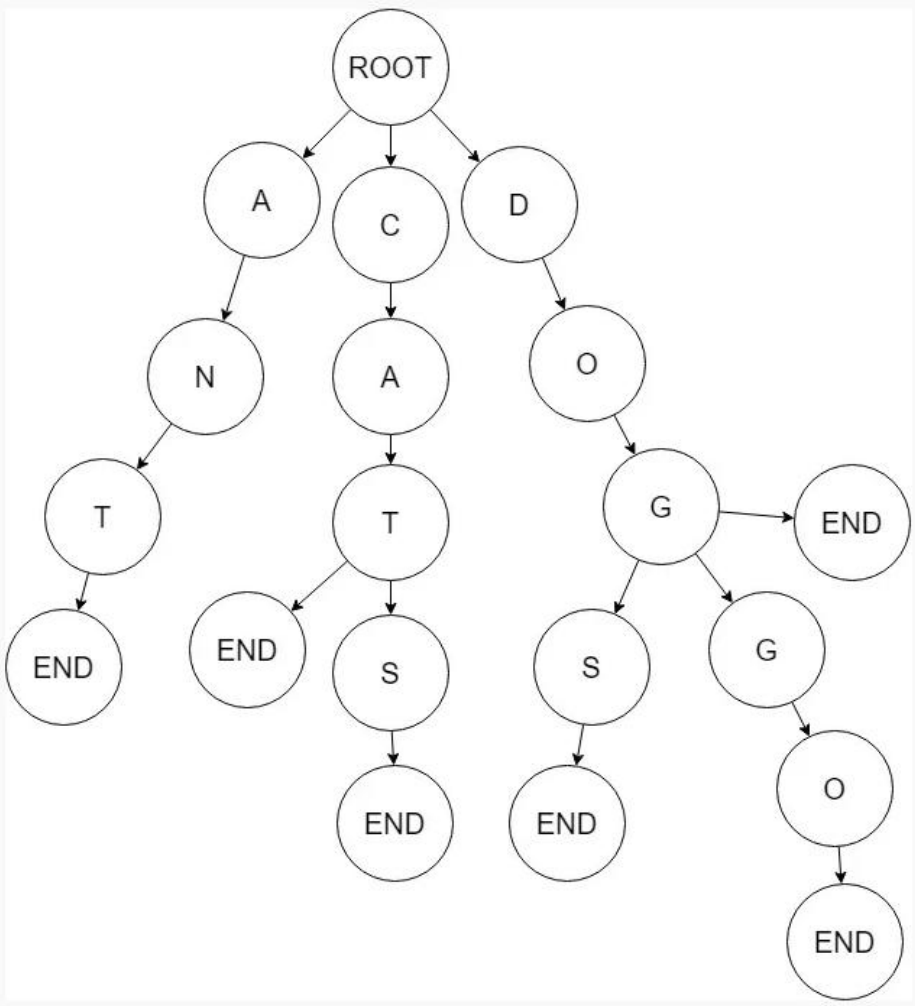
叔块指针：

- 仅仅用来生成区块的奖励
- 不考虑叔块中的交易信息
- 每个新区块最多指向两个叔块
- 不能指向太久远的孤块 (和父块不”同辈”)



默克尔基数树 (Merkle Patricia Trie)

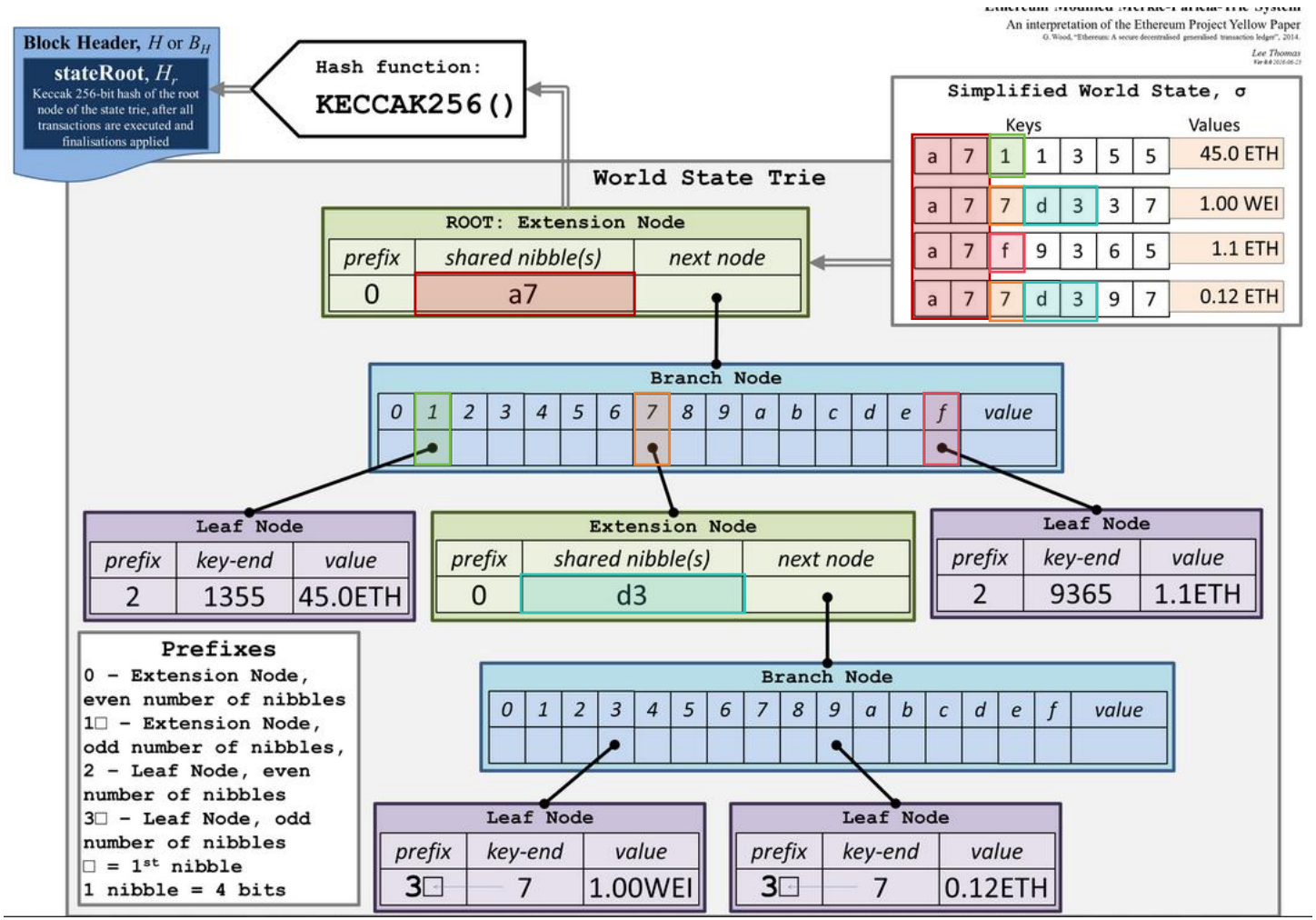
Patricia Trie (Radix Tree/Trie)： key-value数据结构



查询 key=“ant” 对应的 value：
从root一步一步找到ant对应的叶子节点

默克尔基数树 (Merkle Patricia Trie)

Merkle Patricia Trie (具有Patricia Trie结构的哈希树)



假设有4个账号:

a711355
a77d337
a7f9365
a77d397

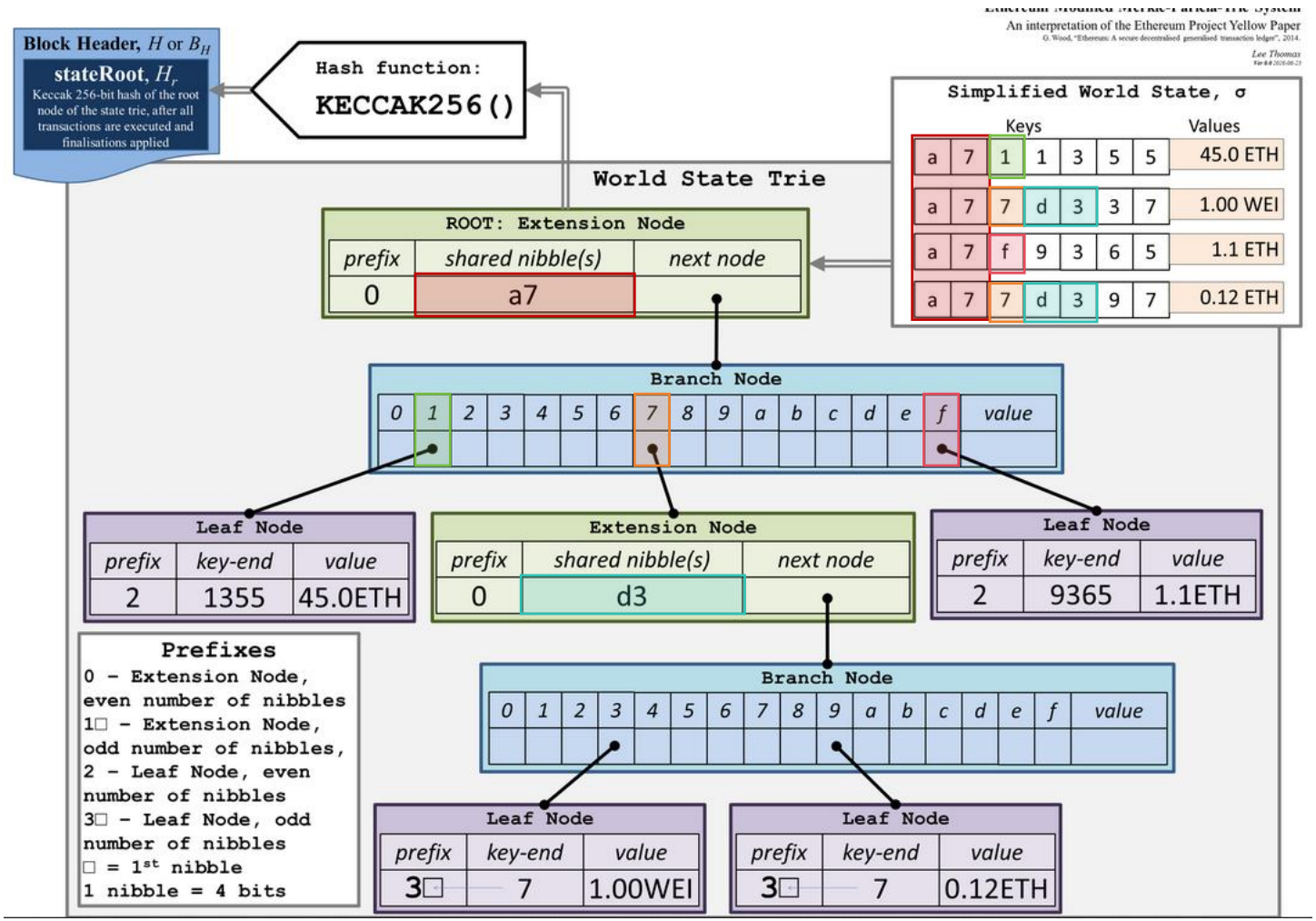
根扩展节点
使用a7作为共享前缀

仍然没有添加完所有账号，因此根扩展节点引入一个**分支节点**

分支节点有16种可能的扩展方式，即包含一个半字节所有的可能性，位置1和位置f包含**叶子节点**，位置7继续添加扩展节点，其余位置为**空节点**

默克尔基数树 (Merkle Patricia Trie)

Merkle Patricia Trie (具有Patricia Trie结构的哈希树)



假设有4个账号:

a711355
a77d337
a7f9365
a77d397

如何给出a711355在MPT的证明?

a7-1是叶子节点

如何给出a722222不在MPT的证明?

a7-2是空节点

布隆过滤器 (Bloom Filter)

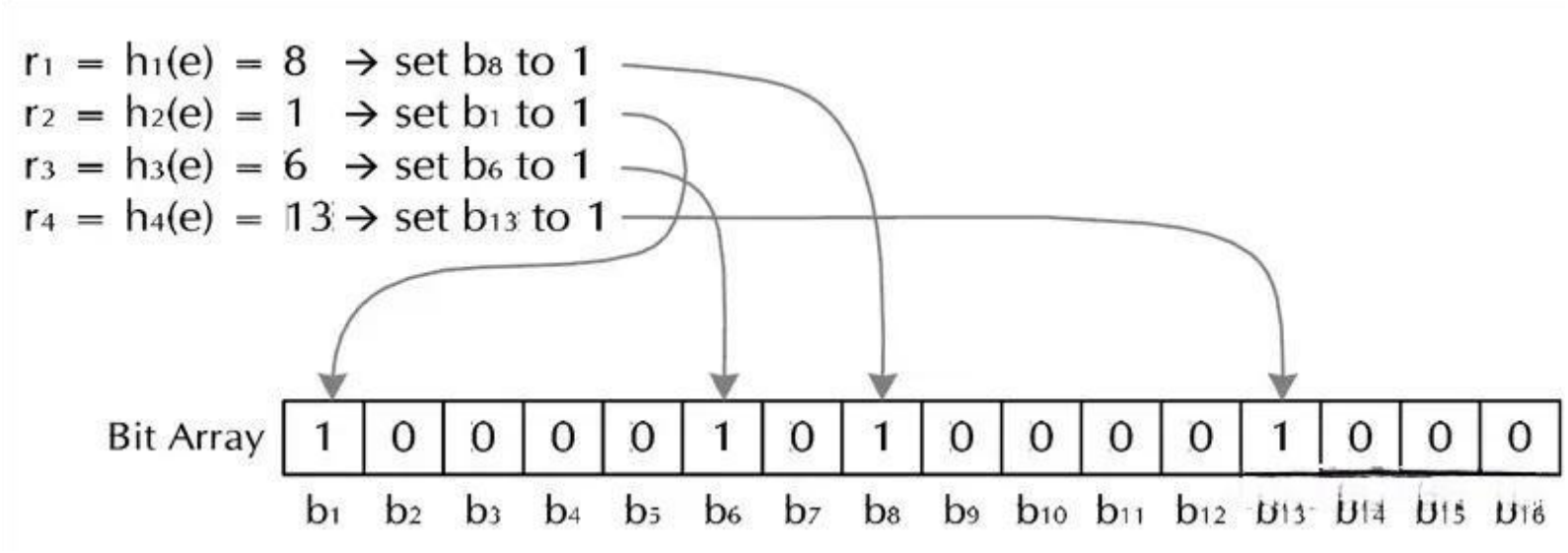
快速检测 x 是否在集合 R：用于查询区块中是否存储了特定的文件

添加元素 x 到集合里时：

- 计算 x 的多个哈希值
- 在一个比特数组中，将哈希值对应的位置设为1
- 比如四个哈希值分别为 8、1、6、13，在比特数组的第1、8、6、13个位置将0设为1

判断元素x是否在集合中时：

- 计算 x 的多个哈希值，得到 8、1、6、13
- 读取比特数组的第1、8、6、13个位置，如果都是1，返回接受；否则返回拒绝



布隆过滤器 (Bloom Filter)

以太坊的 Bloom filter：

- 2048个比特的数组 (注意 2048 = 2^11)
- 哈希函数需要 11 比特，不是直接使用 Keccack256，而是使用了：
  - H_1 = Keccack256 的最低双字节的最低11位
    => 得到第一个 0-2047 之间的index
  - H_2 = Keccack256 的次最低双字节的最低11位
    => 得到第一个 0-2047 之间的index
  - H_3 = Keccack256 的次次最低双字节的最低11位
    => 得到第一个 0-2047 之间的index
- 错误率：
  - 只有false positive(错误接受一个不在集合中的元素)，不存在 false negative
  - m=2048，k=3：
    - n = 100 -> ~0,25%
    - n = 200 -> ~1,64%

$$\varepsilon = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k.$$

练习题

- 如果实现Merkle Tree的哈希函数满足抗碰撞性，试证明：对一个承诺了N个位置的Merkle tree root，对每个位置 $i \in [n]$，攻击者难以可行地找到不同的 F_i 和 F_i' 和对应的Merkle tree证明 π_i 和 π_i'，使得 (F_i, π_i) 和 (F_i', π_i') 都通过验证

- 创世区块中coinbase交易是否包含了可以转出的有效UTXO？

- 在比特币中，第888888个区块的coinbase交易奖励多少 BTC？

- 类比于比特币的SPV技术，是否可以在以太坊中实现类型的轻节点功能，需要几个Merkle Patricia Trie证明？
  注意：以太坊中的交易列表可能包含转账失败的交易。