Blockchain Cryptography

LING Zong, Ph. D.
Senior Software Engineer / Scientist
IBM Almaden Research Center
San Jose, California, U.S.A.

Lecture Outline

Hashing Review
Elliptic Curves
Digital Signature Schemes
Bitcoin Scripting
Proof of Burn
Merkle Tree
Simplified Payment Verification

Hashing Review

Hashing Review

我们定义了一个加密哈希函数:

We define a cryptographic hash function: $H : \{0,1\}^* \longmapsto \{0,1\}^k$

Maps some arbitrarily-sized bit string to some fixed-size bit string.

将任意大小的位串映射为固定大小的位串。
函数是确定性的;相同的输入总是产生相同的输出。
现代密码学的主力——Bruce Schneier

The function is deterministic; **the same input always yields the same output**.

“The workhorses of modern cryptography” - Bruce Schneier

Example: SHA256 maps to a 256-bit string

> echo “Hello, world!” | sha256sum

0xd9014c4624844aa5bac314773d6b689ad467fa4e1d1a50a1b8a99d5a95f72ff5

Properties of a Cryptographic Hash Function

Notation: is hidden, is public

原像计算困难性

Preimage Resistance:

Let $x = \{0,1\}^* =$ message

$y = H(x)$

$x = H^{-1}(y)$ → should be computationally difficult

要找到散列输出的原图像(原始值)应该非常困难

\+ It should be extremely difficult to find the preimage (original value) of a hash output.

Second Preimage Resistance:

Given message x

Finding some x' s.t. $H(x') = H(x)$ should be computationally difficult.

Properties of a Cryptographic Hash Function

抗碰撞

Collision Resistance:

Suppose we have two messages x_1, x_2

The probability that their hashes are equal, P(H(x_1) = H(x_2)), should be “very small”

Equivalently, finding x_1, x_2 such that H(x_1) = H(x_2) should be computationally difficult.

找到碰撞的上限是

The upper bound to find a collision is at most $O(N^{1/2})$ (called a Birthday Attack)

Why are cryptographic hash functions useful?

在比特币和其他加密货币的背景下:

In the context of Bitcoin and other Cryptocurrencies:

- Merkle Trees
- Proof-of-Work (Bitcoin, Litecoin, Dogecoin, etc…)
- Transactions, Blocks, Addresses all referenced by hash value

Also

- Fundamental operation in many cryptographic protocols
- Hash-based Message Authentication Codes (HMACs)
- Password Verification
- Commitment Schemes
- Pseudo-Random Number Generators (PRNGs)

伪随机数产生器

Simple Hash Commitment Scheme

Why are these hash properties useful?

Consider a simple example: Alice and Bob bet $100 on a coin flip

1) Alice calls the outcome of the coin flip
2) Bob flips the coin
3) Alice wins the $100 if her guess was correct

Now, what if Alice and Bob are separated and don't trust one another?

爱丽丝想让鲍勃相信她的猜测，不要在鲍勃抛硬币之前透露她的猜测，否则鲍勃可以作弊

- Alice wants to give Bob a *commitment* to her guess, without revealing her guess before Bob flips the coin, otherwise Bob can cheat!

Simple Hash Commitment Scheme

Instead, we can modify our “protocol” to bind Alice’s guess with a commitment 用承诺约束爱丽丝的猜测

1) Alice chooses a large random number, R.
2) Alice guesses the outcome of the coin flip, B.
3) Alice generates a *commitment* to the coin flip, C = H(B || R)
4) Alice sends this commitment to Bob.
5) Bob flips the coin and sends the value to Alice.
6) Alice sends Bob the random number and her guess: (R’, B’)
7) Bob then checks that C’ = H(R’ || B’) = C = H(B || R), to ensure Alice did not change her guess mid commitment.
8) Both can now agree on who won the $100.

Simple Hash Commitment Scheme - Cheating

How could Bob cheat Alice?

1) When Bob receives C = H(B || R), if he can compute $H^{-1}(C) = B \,||\, R$, Bob can recover Alice's guess and send her the opposite outcome!

   If our hash function, H, is **preimage resistant**, this shouldn't be possible.

How could Alice cheat Bob?

1) Alice sends Bob her commitment C = H(B || R), but reveals the opposite guess, (!B, R'). Alice wins if she can pick R' s.t. C' = H(!B || R') = C.

   This fails if our hash function, H, is **second preimage resistant**!

Elliptic Curves

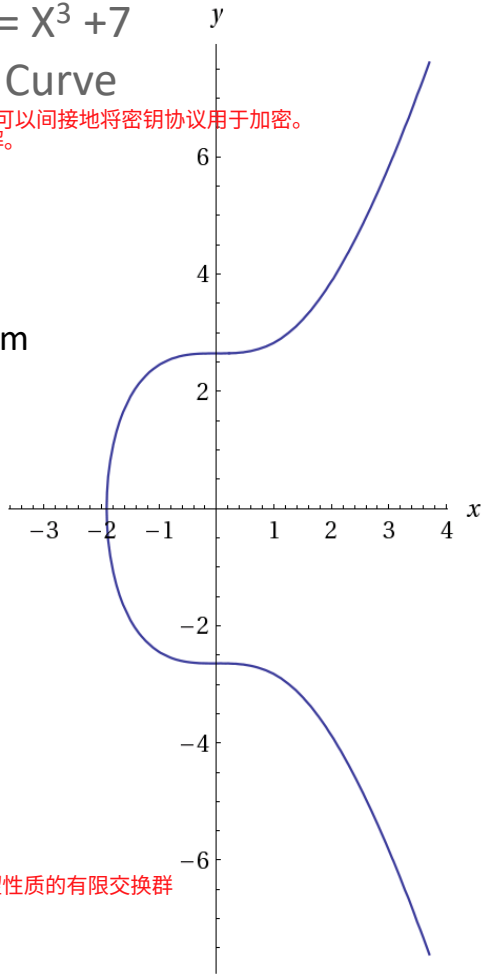
Elliptic Curves

secp256k1 : $Y^2 = X^3 + 7$

Bitcoin's Elliptic Curve

椭圆曲线密码体制是一种基于有限域上椭圆曲线的代数结构的公钥密码体制。
· 与非ec加密(基于普通的Galois字段)相比，ECC需要更小的密钥来提供同等的安全性。
· 椭圆曲线适用于密钥协议、数字签名、伪随机生成器等任务。通过将密钥协议与对称加密方案相结合，可以间接地将密钥协议用于加密。
· 它们也被用于一些在密码学中有应用的基于椭圆曲线的整数因子分解算法，如Lenstra椭圆曲线因子分解。

- **Elliptic-curve cryptography** (**ECC**) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields.
  - ECC requires smaller keys compared to non-EC cryptography (based on plain Galois fields) to provide equivalent security.
  - Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme.
  - They are also used in several integer factorization algorithms based on elliptic curves that have applications in cryptography, such as Lenstra elliptic-curve factorization.



An elliptic curve is defined by the following affine, long Weierstrass form:

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

We usually consider the short Weierstrass form:

$$E : Y^2 = X^3 + aX + b$$

For the most part, all you need to know about elliptic curves is that they provide another finite **abelian group** with certain desired properties.在大多数情况下，你所需要知道的椭圆曲线是它们提供了另一个具有某些期望性质的有限交换群

https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

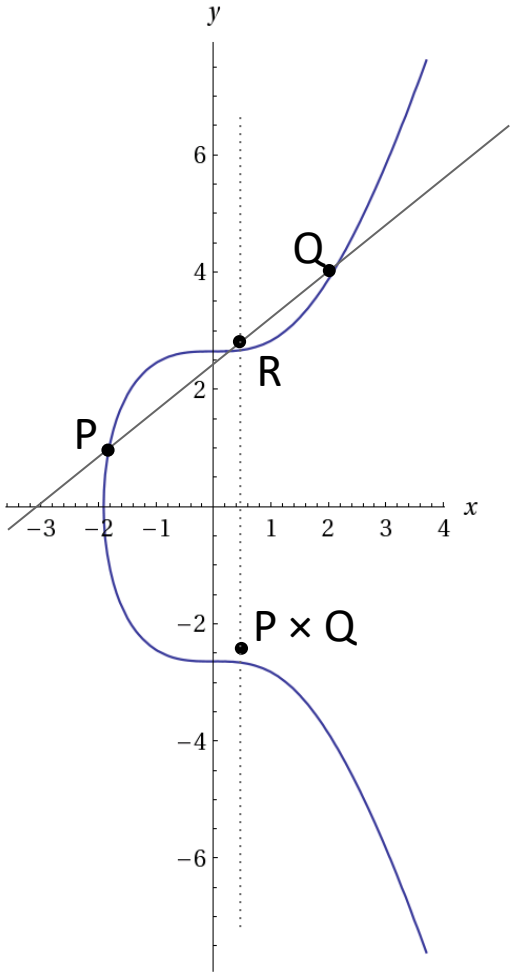
Elliptic Curves - Group Law

One can define a group law on an elliptic curve using the chord-tangent process.

Given two elliptic curve points, P and Q, we define P × Q as the following:

> We find the line intersecting P and Q, which must intersect with one final point, R. If we then reflect R across the x-axis, we obtain another point which we define as P × Q.

我们可以用弦-切过程在椭圆曲线上定义群律。
给定两个椭圆曲线点P和Q，我们定义P × Q为：
我们找到与P和Q相交的直线，它必须与最后一个点R相交，如果我们让R穿过x轴，我们得到另一个点，我们定义为P乘以Q



Elliptic Curves - Group Law

More formally,

$$\text{Let } P = (x_1, y_1),\ Q = (x_2, y_2),\ P \times Q = (x_3, y_3)$$

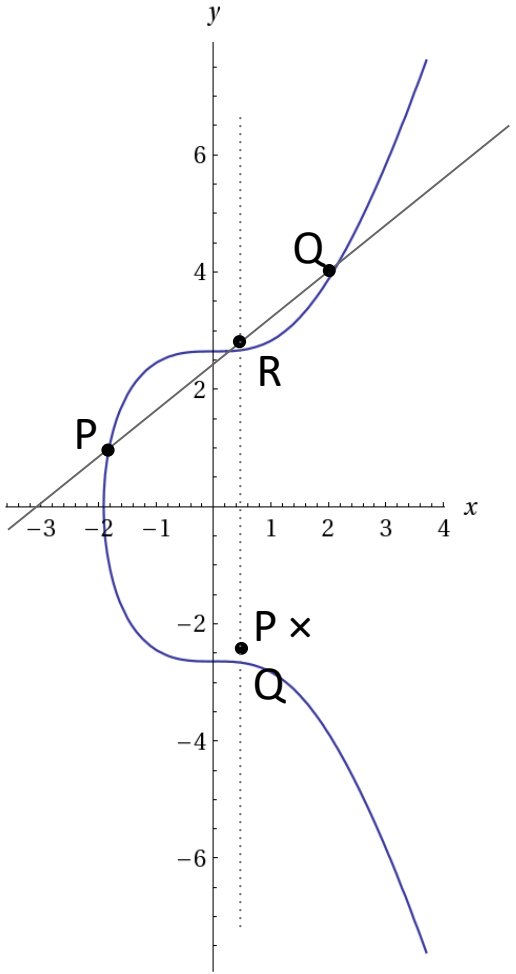
$$s = \frac{y_2 - y_1}{x_2 - x_1}$$

$$x_3 = s^2 - x_1 - x_2$$

$$y_3 = s(x_1 - x_2) - y_1$$

在某些曲线和有限域上，形成一个循环(或近似循环)有限交换群

Over certain curves and finite fields, this forms a cyclic (or nearly cyclic) finite abelian group.



Elliptic Curves - Group Law

If P = Q, we find the tangent（切点） at P, extend it to the point, R, then reflect across the x-axis to P^2.

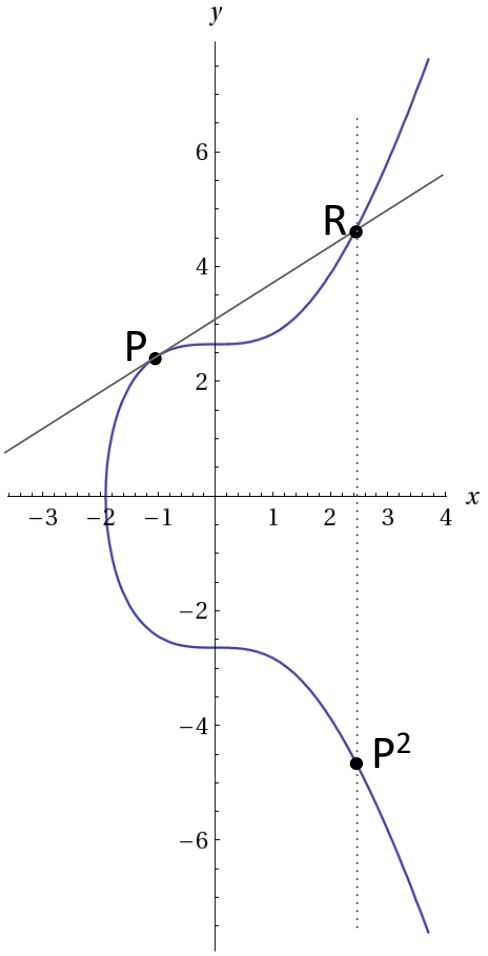
$$P = (x, y)$$

$$s = \frac{3x^2 - a}{2y}$$

$$x' = s^2 - 2x$$

$$y' = s(x - x') - y$$

$$P^2 = (x', y')$$



y^2 = x^3 + 7 | Computed by Wolfram|Alpha

Elliptic Curve Discrete Logarithm Problem (ECDLP)

For some positive integer, m, we can define:

$$P^m = P \times P \times \cdots \times P = \prod_{i=1}^{m} P$$

It is believed that finding m given P and P^m

$$m = \log_P P^m$$

is computationally difficult over certain finite fields and certain curves. 在特定的有限域和特定的曲线上是很难计算的。

因此，ECDLP是椭圆曲线密码学的基础

As such the ECDLP forms the basis of elliptic curve cryptography.

ECDLP

与普通有限域上的离散对数相比，椭圆曲线离散对数问题没有已知的次指数算法(假设曲线不是超奇异的或异常的)。

Compared with discrete logarithms over vanilla finite fields, **the elliptic curve discrete logarithm problem has no known, sub-exponential algorithms** (assuming the curve is not supersingular or otherwise anomalous).

求解椭圆曲线离散对数的最快速、最实用的算法是并行波拉德算法

The fastest, practical algorithm for elliptic curve discrete logarithms is **parallel Pollard's Rho**, which runs in $O(N^{1/2})$.

为了实现相当于128位分组密码的安全性，需要选择一条群序的曲线

To achieve security equivalent to a 128-bit block cipher, we need to choose a curve of group order $\approx 2^{256}$.

将其与RSA或其他基于因数的算法进行比较，后者需要2048位密钥才能实现同等的安全性。

Compare this with RSA or other factoring-based algorithms, which requires ≈ 2048 bit keys for equivalent security.

Digital Signature Schemes

Digital Signature Schemes (DSS)

类似于手写签名。其他人可以验证带有您签名的消息实际上是您写的。同样，没有你也很难伪造签名

Similar to a handwritten signature.

Other people can verify that a message with your signature was, in fact, written by you.

Likewise, it should be difficult to forge a signature without you.

Example: Signing a text file with my gpg key

> echo “Hello, world!” > a.txt && gpg --sign a.txt && base64 a.txt.pgp

owEBTQKy/ZANAwACAXUyGnahI6...QrdcG/B0kkRUmnObdDF7hT/0b9wTg=

GNU隐私保护。GnuPG是RFC4880(也称为PGP)定义的OpenPGP标准的一个完整而自由的实现。GnuPG允许您加密和签署您的数据和通信;它具有一个通用的密钥管理系统，以及用于各种公钥目录的访问模块。

The GNU Privacy Guard. GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories.

DSS Security Definitions

收件人(消息，团体)希望下列属性：

The (message, sig) recipients desire the following properties:

讯息完整性

- **Message integrity** - the message hasn't been modified between sending and receiving.
- **Message origin** - the message was indeed sent by the original sender.
- **Non-repudiation** - the original sender cannot backtrack and claim they did not send the message.

不可否认性

原始发送者不能回溯并声称他们没有发送该消息

Digital Signature Schemes

More formally, a digital signature scheme consists of two algorithms:

+ A signing algorithm, Sign, which uses a secret key, sk.

  s = Sign(m, sk), s is the signature for some message, m.

+ A verification algorithm, Verify, which uses a public key, pk.

  valid = Verify(Sign(m, sk), pk), invalid = Verify(s', pk) for all s' != s.

ECDSA : Elliptic Curve Digital Signature Algorithm

ECDSA is defined by:

E: an elliptic curve.

g: a generator point of the elliptic curve with large prime order, p.

大素数阶椭圆曲线的一个生成点

p: a large, prime integer where $g^p = O$.

H: a cryptographic hash function.

ECDSA - Setup

The signer creates:

The secret key, sk, chosen randomly from [0, …, p-1].

The public key, $pk = g^{sk}$, which should be distributed publicly.

ECDSA - Sign

Sign(m, sk):

$h = H(m)$
Hash the message

$z = h[0 : \log_2 p]$ — Take the $\log_2 p$ left-most bits of h

k = randomly chosen from $[1, \ldots, p-1]$ — k must be kept secret!

$r = x\text{-coord}(g^k) \pmod p$ — $x\text{-coord}(P = (x, y)) = x$

$s = (z + sk \cdot r) \cdot k^{-1} \pmod p$

return (r, s) — **The signature for our message**

ECDSA - Verify

Verify(r, s, m, pk):

$z = H(m)[0, \ldots, \log_2 p]$

$a = z \cdot s^{-1} \pmod{p}$

$b = r \cdot s^{-1} \pmod{p}$

$v = \text{x-coord}(g^a \cdot pk^b) \pmod{p}$

return valid if $v \equiv r \pmod{p}$, otherwise invalid

ECDSA - Quick Proof Sketch

Let $w = s^{-1}$

$$g^a \cdot pk^b = g^{z \cdot w} \cdot g^{sk \cdot r \cdot w}$$

$$= g^{z \cdot w + sk \cdot r \cdot w}$$

$$= g^{(z + sk \cdot r) \cdot w}$$

$$= g^{(z + sk \cdot r) \cdot (z + sk \cdot r)^{(-1)} \cdot (k^{(-1)})^{(-1)}}$$

$$= g^k$$

$$v = \text{x-coord}(g^a \cdot pk^b) \pmod p$$

$$= \text{x-coord}(g^k) \pmod p$$

$$= r \pmod p$$

在比特币中使用ECDSA签名来证明交易输出的所有权

ECDSA signatures are used in Bitcoin to show proof of ownership of the outputs of a transaction!

Account-based vs. Transaction-based ledger

Account-based 基于帐户
●必须跟踪影响Alice的每个事务
●需要额外维护，容易出错

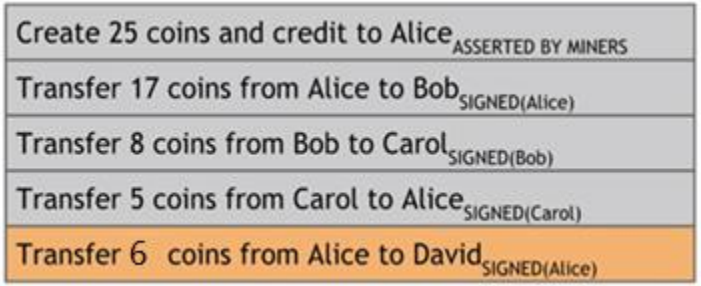
- must track every transaction affecting Alice
- Requires additional maintenance, error-prone

Bitcoin is a transaction-based ledger (triple-entry accounting). 比特币是一种基于交易的分类账(三重分录会计)

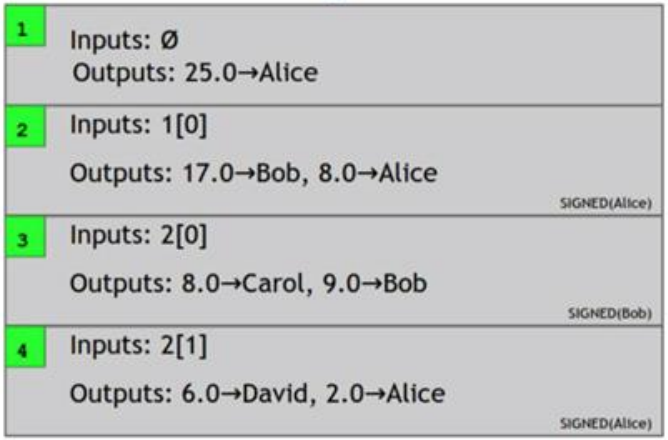
Features: 特点:
●更改地址——必须更改，因为tx输出只使用一次
●有效的验证——只读取最近的历史
●联合支付- Alice + Bob form 1 tx

- Change addresses - Required since tx outputs only spent once
- Efficient verification - only read recent history
- Joint payments - Alice + Bob form 1 tx

(Credit for content organization and figures goes to Princeton textbook)

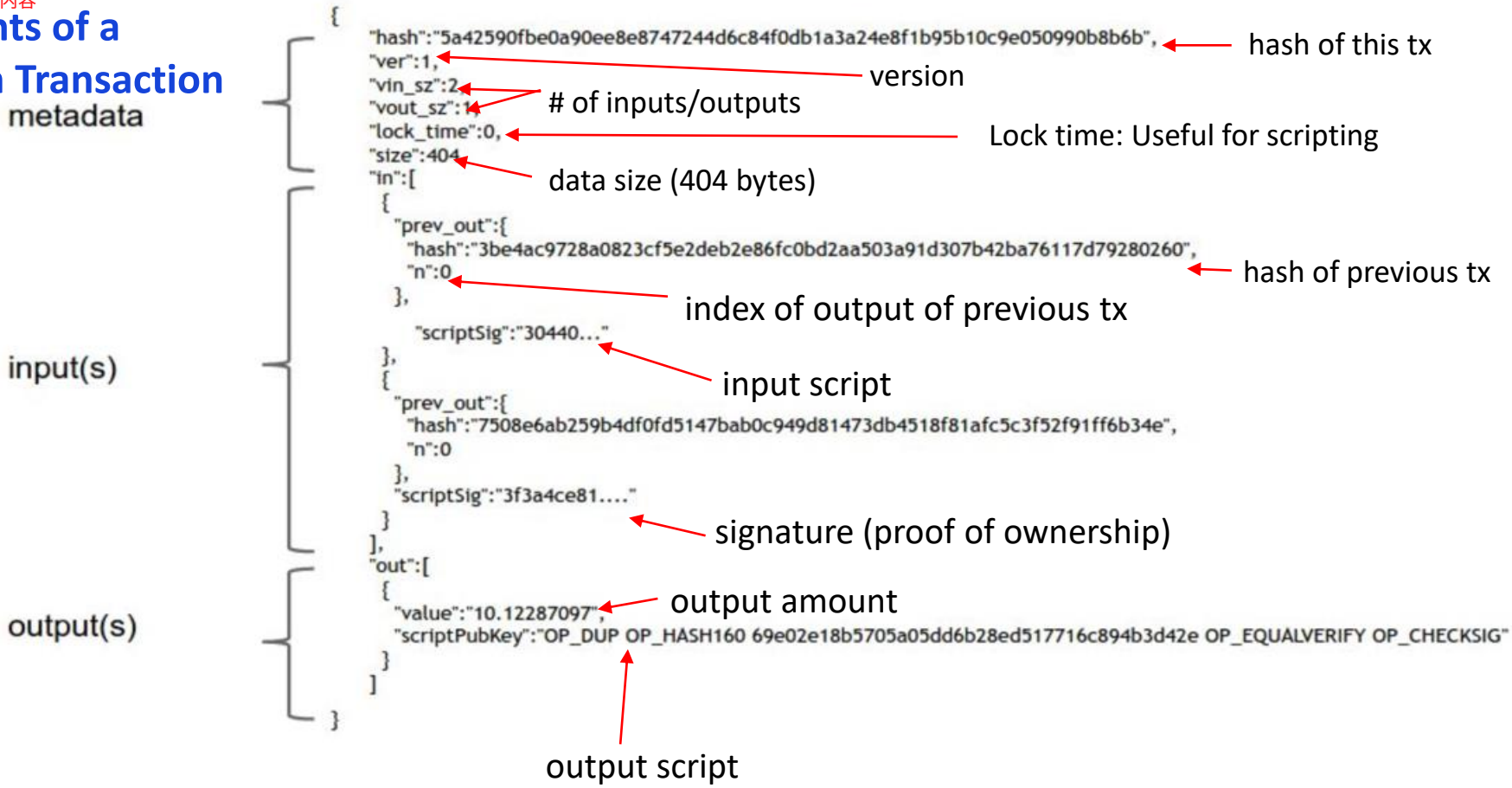


an account-based ledger



a transaction-based ledger, which is very close to Bitcoin

Contents of a Bitcoin Transaction

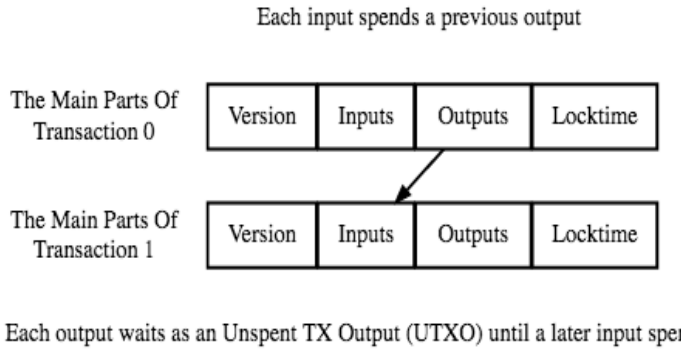


An actual Bitcoin transaction.

Bitcoin Scripting

(Bitcoin.org developer guide)

Bitcoin Scripting



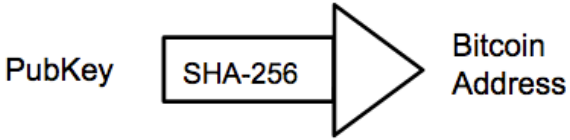
输出“地址”实际上是脚本

Output "addresses" are really scripts.

通过脚本的输入和输出允许比特币的未来可扩展性

Inputs and outputs through scripting allows for future extensibility of Bitcoin.

- Remember:
  - 签名证明公钥的所有权
  - **Signatures** prove ownership of a public key
  - **Bitcoin Address == Hash(PubKey)**
  - Transaction composed of input and output **addresses**
  - Spending Bitcoin is **redeeming** previous transaction outputs

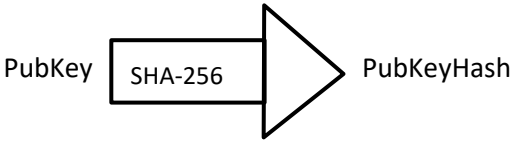


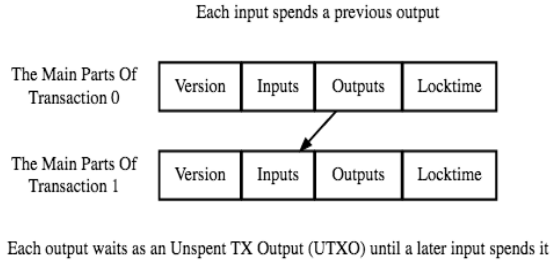
- 此金额可由地址X的所有者签名兑现
  "This amount can be redeemed by a **signature** from owner of address X"
  - No way to get the public key from the address!
- "This amount can be redeemed by the **public key** that hashes to address X, plus a **signature** from the owner of that public key"

这个数目可以由散列到地址X的公钥加上该公钥所有者的签名来赎回

(Quotes from Princeton textbook)

Bitcoin Scripting

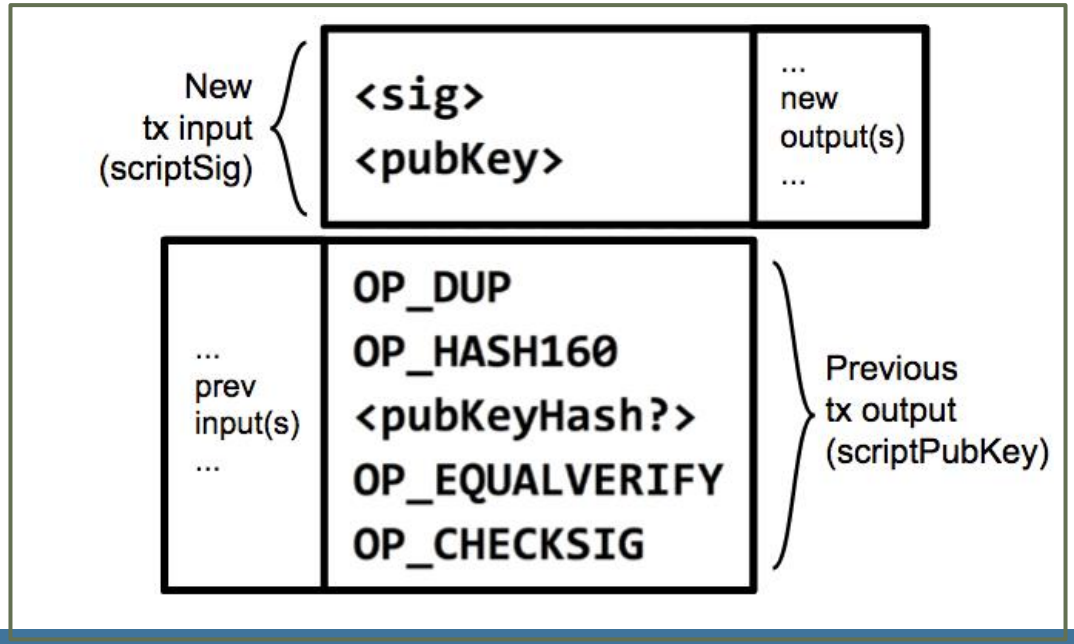




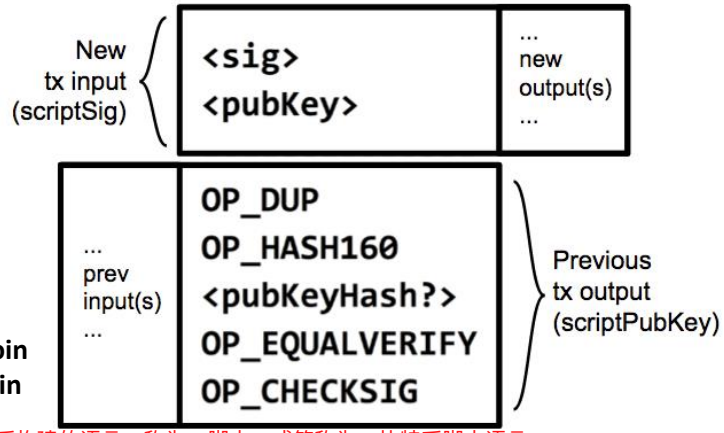
Remember: Hash(PubKey) == Address == "PubKeyHash"

Figure:
Two transactions along with their input and output scripts

两个交易及其输入和输出脚本



Bitcoin Scripting

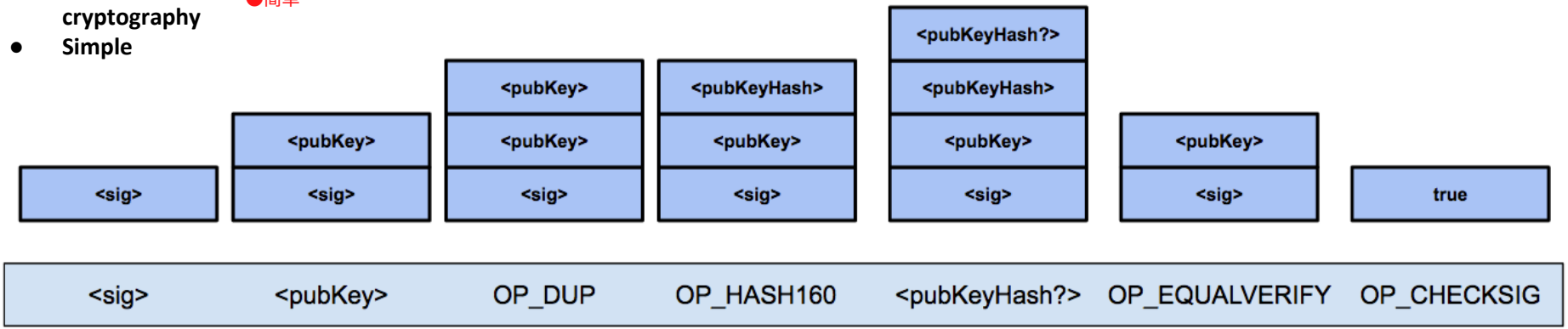


Output says: "This amount can be redeemed by
1) the **<pubKey>** that hashes to address **<pubKeyHash?>**
2) plus a **<sig>** from the owner of that **<pubKey>**
...that will make this script evaluate to **true**."

Language built specifically for Bitcoin called "Script" or simply "the Bitcoin scripting language"

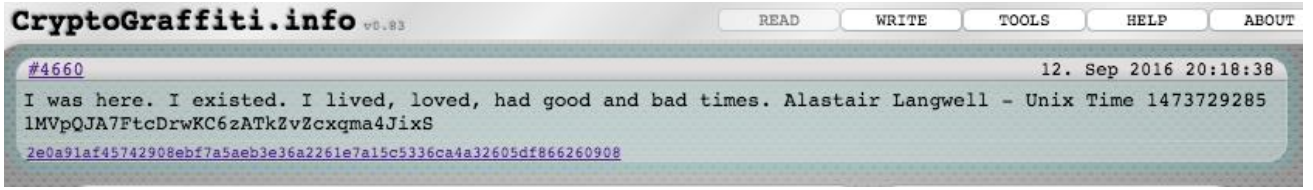
- **Stack based**
- **Native support for cryptography**
- **Simple**

专门为比特币构建的语言，称为“ 脚本 ”或简称为“ 比特币脚本语言 ”
●基于堆栈
●对加密的本机支持
●简单



Proof of Burn

Proof of Burn



如何将任意数据写入比特币区块链?

How to write arbitrary data into the Bitcoin blockchain?

Output script:

OP_RETURN

<arbitrary data>

<harambe>

Proof of Burn

如果到达，则OP_RETURN抛出一个错误

- OP_RETURN throws an error if reached
- Output script can't be spent, therefore you prove that you destroyed some currency 输出脚本无法使用，因此需要证明您销毁了一些货币
- Anything after OP_RETURN is not processed OP_RETURN之后的任何内容都不会被处理

Use cases

证明某物在特定时间点的存在性

- Prove existence of something at a particular point in time
  - Ex. A word you coined, hash of a document/music/creative works
- Bootstrap CalCoin by requiring that you destroy some Bitcoin to get CalCoin

通过要求你摧毁一些比特币来获得CalCoin来引导CalCoin

Pay-to-PubkeyHash vs. Pay-to-Script-Hash

在比特币中，发送方指定脚本。
In Bitcoin, senders specify the script.

卖主说：“ 把你的硬币送到这个公共密钥的散列处。 ”
P2PKH: Vendor says "Send your coins to the hash of this **PubKey**."

- Simplest case
- Is by far the most common case

目前为止最常见的情况

But for complicated scripts (such as multisig), this can be a problem.

但是对于复杂的脚本(如multisig)，这可能是一个问题。
供应商说：“ 要付钱给我，写一个复杂的输出脚本，允许我使用多个签名。 ”
Ex. Vendor says "To pay me, write a complicated output script that will allow me to spend using multiple signatures."

How would the sender know?

Pay-to-PubkeyHash vs. Pay-to-Script-Hash

Solution? **Pay-to-Script-Hash (P2SH)**

“把你的硬币送到公钥的hash去。”

P2PKH: "Send your coins to the hash of this **PubKey**."

“把你的硬币扔到脚本的hash里去。为了赎回这些硬币，你必须公开拥有给定哈希值的脚本，并提供数据让脚本的值为真。

P2SH: "Send your coins to the hash of this **script**. To redeem those coins, you must reveal the script that has the given hash and provide **data** that will make the script evaluate to true."

把复杂的脚本写给收件人

- Offloads complicated script writing to recipients
- **P2SH is the most important improvement to Bitcoin since inception**

P2SH是比特币自诞生以来最重要的改进

Merkle Tree

Merkle Tree

哈希指针的二叉树

A binary tree of hash pointers

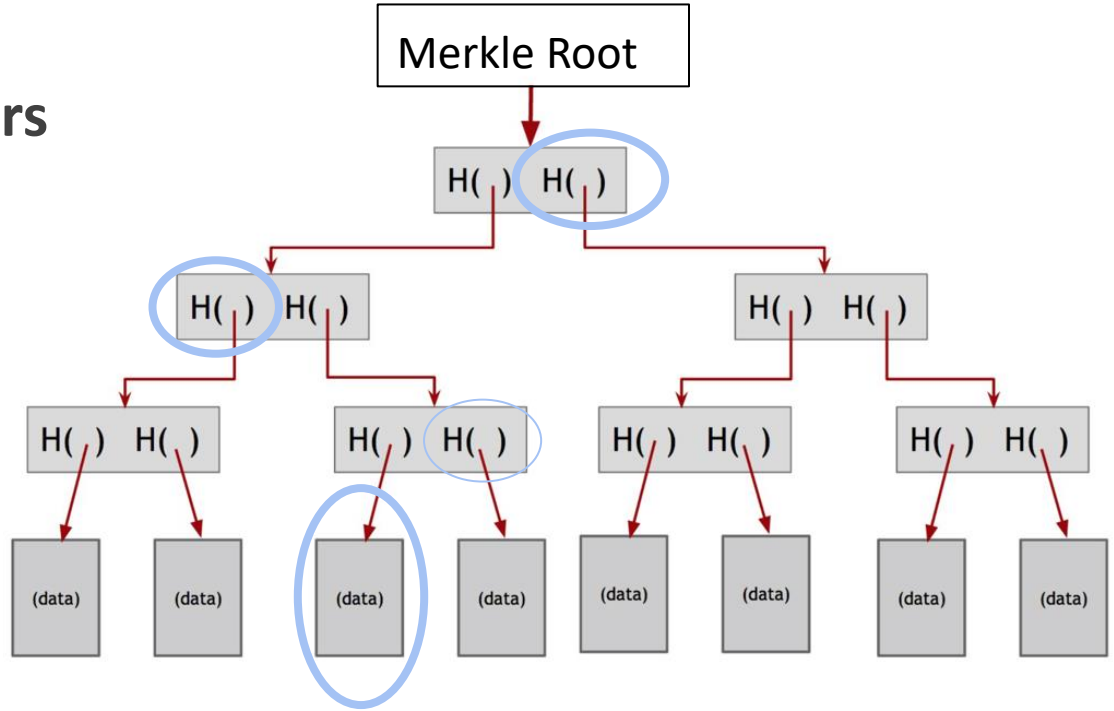
哈希被散列在一起

- Hashes are hashed together

这允许证明数据存在于与Merkle根相对应的树中，而无需实际保存树中的所有数据。

This allows you to prove that your data existed in the tree corresponding to the Merkle Root without actually saving all of the data in the tree.

- For example, if you had a message in this 3rd node at the bottom, to prove inclusion in the tree you simply have to save these three intermediate hashes in the tree and show that you are able to produce the Merkle Root with this data.



例如，如果在底部的第三个节点中有一条消息，为了证明包含在树中，您只需将这三个中间散列保存在树中，并显示您能够使用该数据生成Merkle根

Princeton Textbook Figure 1.7

Merkle Tree - Bitcoin construction

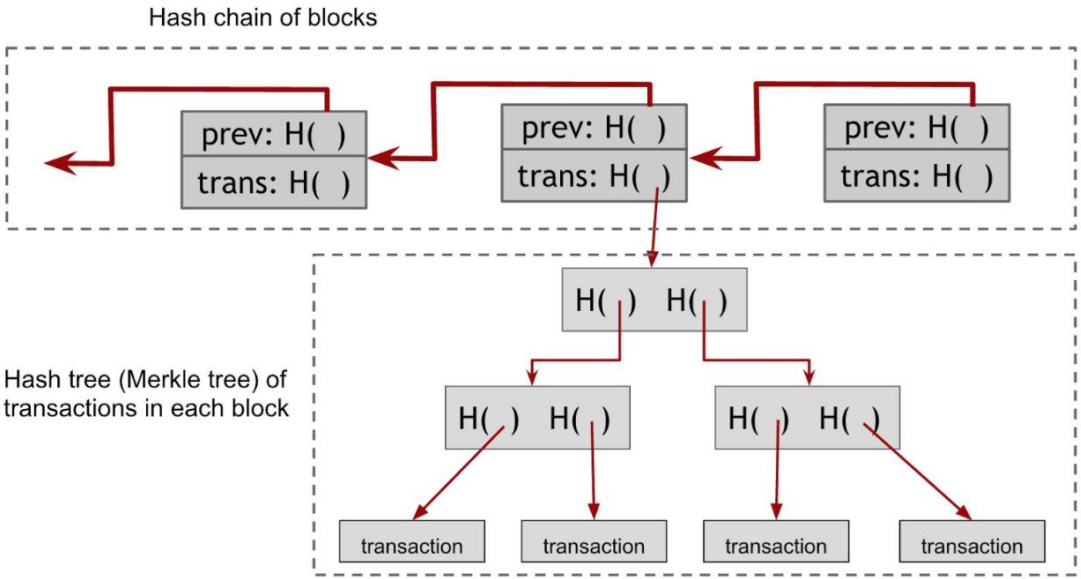
交易是Merkle树中的叶子，包括一个coinbase事务

Transactions are leaves in the Merkle tree, includes a coinbase transaction

Two hash structures

1. Hash chain of blocks
   - These blocks are linked together and based off of each other.
2. A Merkle tree of txs, internal to each block

这些块连接在一起，并以彼此为基础。



Princeton Textbook Figure 3.7/3.8

Merkle Tree - Mining, in more detail

Previously, hash of:

- Merkle Root
- PrevBlockHash
- Nonce (varied value)

below some target value.

Actually two nonces:

1. In the block header
2. In the coinbase tx

实际上两个nonce:
1. 在块头中
2. 在coinbase tx中

Hash of

- PrevBlockHash
- Coinbase nonce (varied value)
  - Affects the Merkle Root
- Block header nonce (varied value)

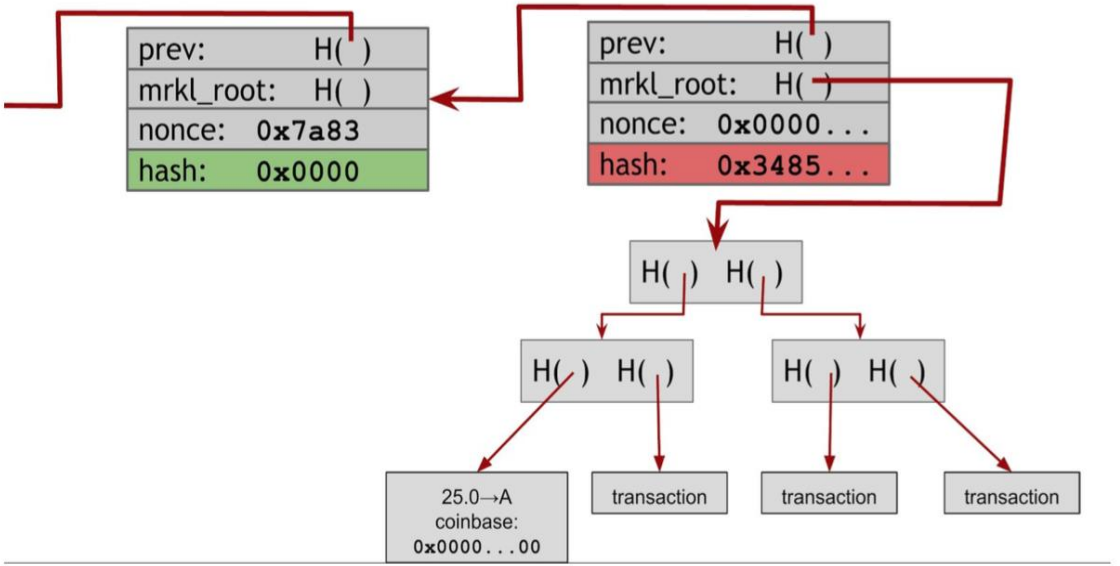


Figure 5.1: Finding a valid block. In this example, the miner tries a nonce of all 0s. It does not produce a valid hash output, so the miner would then proceed to try a different nonce.

Princeton Textbook

Merkle Tree - Bitcoin construction

What if there is no solution?

- Block header nonce is 32 bits
  - Antminer S9 hashes 14 TH/s
  - How long does it take to try all combinations?
  - 2^32 / 14,000,000,000,000 = 0.00031 seconds
  - Exhausted 3260 times per second
- Therefore, must change Merkle root
  - Increment coinbase nonce, then run through block header nonce again
    增加coinbase nonce，然后再次运行块头nonce
  - Incrementing coinbase nonce less efficient because it must propagate up the tree
    增大coinbase的效率会降低，因为它必须沿着树向上传播

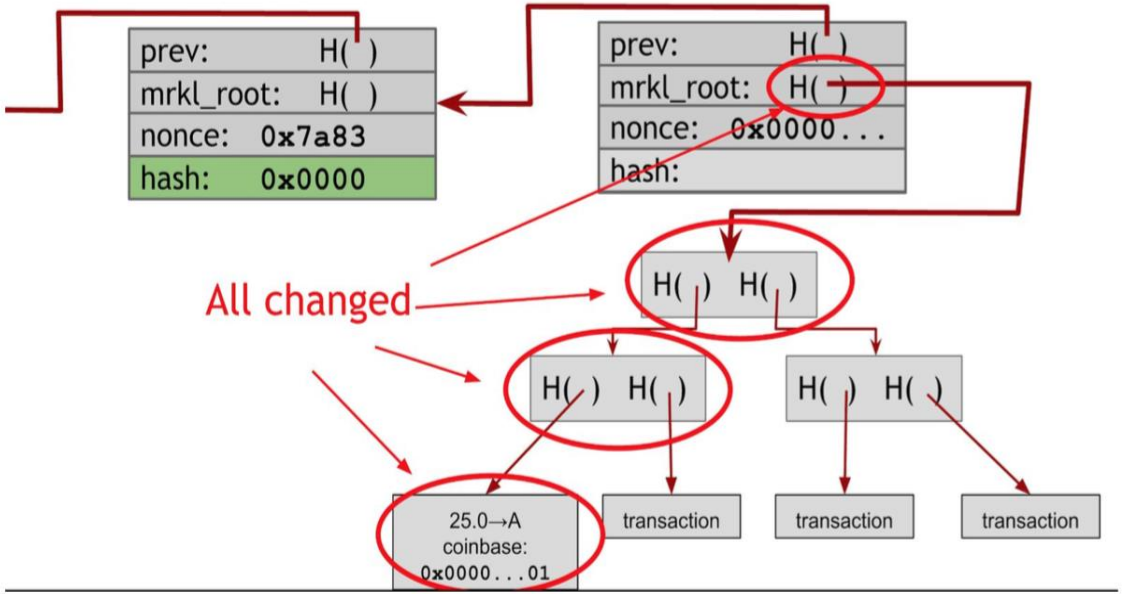


Figure 5.2: Changing a nonce in the coinbase transaction propagates all the way up the Merkle tree.

Princeton Textbook

Simplified Payment Verification

Simplified Payment Verification (SPV)

Current size of Bitcoin blockchain: 83.3 Gigabytes and growing

- 300% of what it was 2 years ago
- Storing the full Bitcoin blockchain will likely not be feasible for the average user, so how do we address this?

存储完整的比特币区块链对普通用户来说可能不可行，那么我们如何解决这个问题呢?

SPV - Simplified Payment Verification

- "SPV nodes" or "thin" client, as opposed to "full nodes"
  - Don't store the full blockchain
  - Only store the pieces of data needed to verify transactions that concern them

只存储验证与它们有关的事务所需的数据片段

- Nearly all nodes on network are SPV nodes
- If you use wallet software where you're most likely running an SPV node

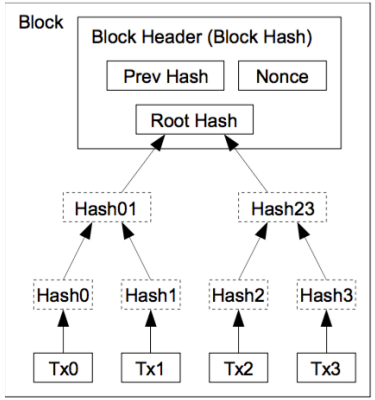
如果你使用钱包软件，你最有可能运行一个SPV节点

SPV - Structure

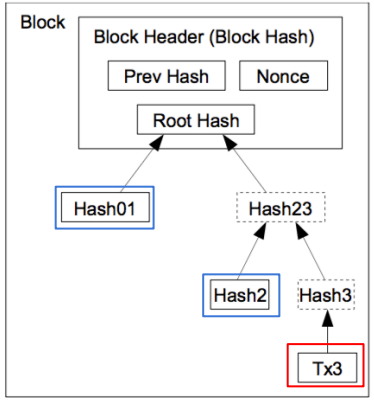
SPV nodes only keep *block headers* of the blockchain

SPV节点只保留区块链的块头
●查询不同的满节点，直到该节点确信其链最长

- Obtained by querying different full nodes until the node is convinced it has the longest chain



Transactions Hashed in a Merkle Tree



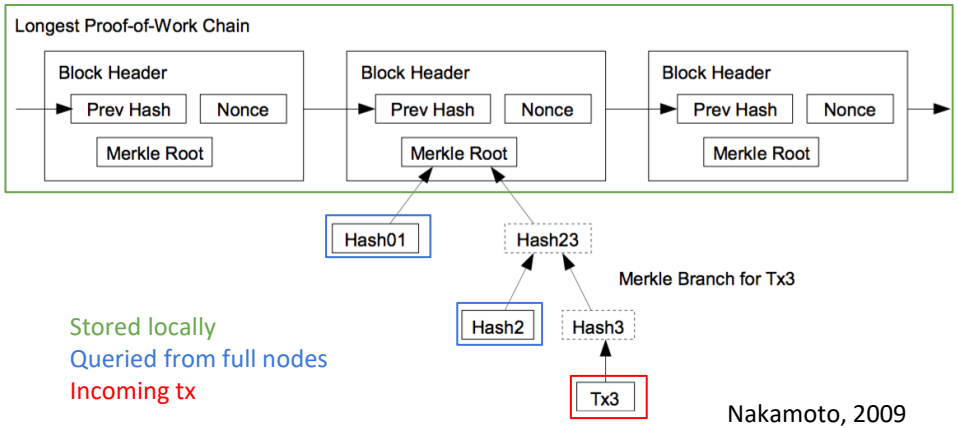
After Pruning Tx0-2 from the Block

Nakamoto, 2009

How to validate an incoming tx?

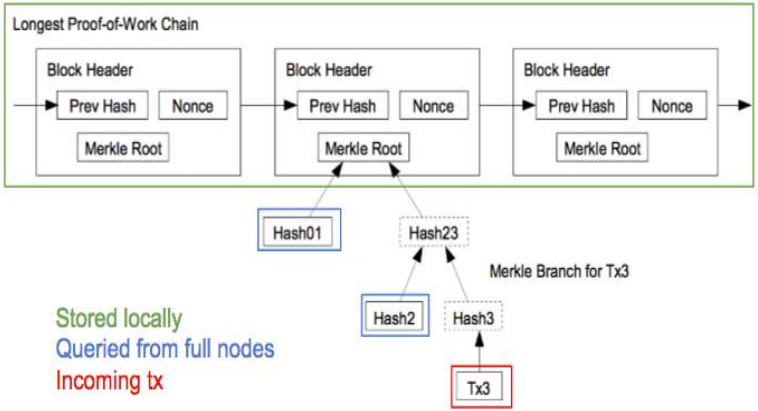
如何验证传入的tx?
· 查询完整节点以获得该事务的Merkle分支
· 哈希tx和中间哈希来获得一个Merkle根，检查它是否与本地保存的根匹配。
· 等待这个tx有足够的确认后再发货

- Query full nodes to get the Merkle branch for that transaction
- Hash tx together with intermediate hashes to obtain a Merkle root, check that it matches the one saved locally.
- Wait for this tx to have enough confirmations before delivering good



Nakamoto, 2009

SPV - Security Analysis



SPV节点:
· 没有完整的tx历史，不知道UTXO集
· 没有相同级别的完整节点安全性
· 不能检查是否每个包含在一个块的tx实际上是有效的
· 只能验证实际影响它们的事务，而且必须惰性地进行验证。

SPV nodes:

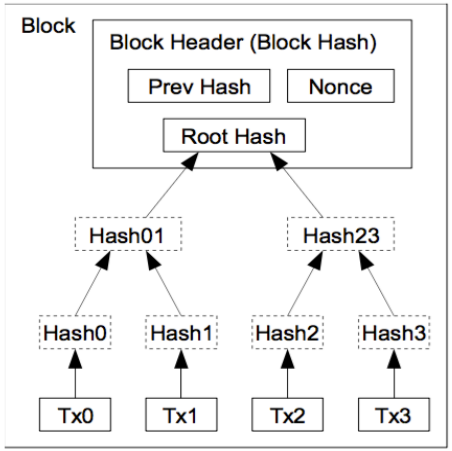
- Don't have full tx history, don't know UTXO set
- Don't have same level of security of full nodes
- Can't check if every tx included in a block is actually valid
- Can only verify transactions that actually affect them, and must do so lazily. Nakamoto, 2009

SPV nodes assume:

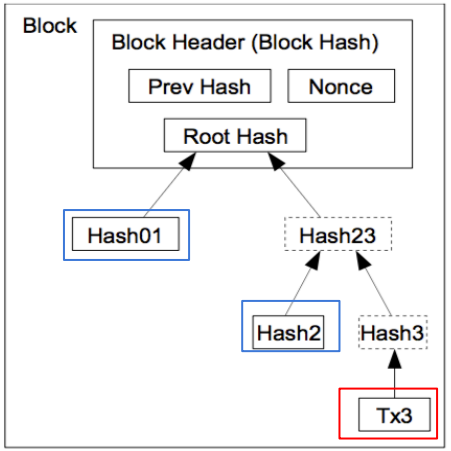
传入的块标头不是一个假链

- ...that incoming block headers aren't a false chain
  - Very expensive for attacks (or anyone) to create blocks 对于攻击者(或任何人)来说，创建块的成本非常高 长期不可持续
  - Not sustainable over the long term
- ...that there ARE full nodes out there validating all transactions 那里有验证所有事务的完整节点 这样做有效率的好处和动机
  - There are efficiency benefits and incentives to doing so
- ...that miners ensure that the transactions they include in their blocks are valid 挖掘者确保其块中包含的事务是有效的 否则他们的块将被全部节点拒绝(非常昂贵的错误!)
  - Otherwise their blocks would be rejected by full nodes (very expensive mistake!)

SPV - Cost savings



Transactions Hashed in a Merkle Tree



After Pruning Tx0-2 from the Block

Nakamoto, 2009

Huge cost savings

- Block headers are only ~1/1000 the size of the full blockchain
  - 83 MB vs. 83 Gigabytes

SPV nodes capitalize on obtaining data for verifying transactions lazily

SPV节点利用获取数据来惰性地验证事务
对那些必须验证许多txs的大商家来说，这可能是个问题

- Could be an issue for big merchants who must verify many txs

对于大多数比特币的消费者和用户来说，SPV是一个不错的折衷

For most consumers and users of Bitcoin, SPV is a decent tradeoff.

Lecture Outline

- ✓ Hashing Review
- ✓ Elliptic Curves
- ✓ Digital Signature Schemes
- ✓ Bitcoin Scripting
- ✓ Proof of Burn
- ✓ Merkle Tree
- ✓ Simplified Payment Verification

完

धन्यवाद

Hindi

多謝

Traditional Chinese

ขอบคุณ

Thai

Спасибо

Russian

Gracias

Spanish

Thank You

English

شكرا

Arabic

Obrigado

Brazilian Portuguese

Grazie

Italian

多谢

Simplified Chinese

Danke

German

Merci

French

நன்றி

Tamil

Tamil

ありがとうございました

Japanese

감사합니다

Korean