Blockchain Notebook – Bitcoin

Initial Edition

Xuan Fu

xuanf@usc.edu

Contents

Introduction

The intention of making this notebook is because I have taken many detours on the journey of learning blockchain. The scattered blockchain knowledge across the Internet prevents general people from understanding it systematically. I hope that through reading and studying this notebook, everyone can quickly understand Blockchain and its tremendous potentials.

My personal understanding of blockchain is as follows:

Blockchain is a linked-list data structure linked by hash "pointers", it has strong traceability and strong tamper-resistant. The success of Bitcoin was not only because of the innovation of computer technology, but also related to the needs of reforming social ideology and financial system. The emergence of the blockchain has shaken the current top-down centralized organizational structure and provided the possibility of a decentralized consensus structure. Blockchain is a fusion discipline that spans computer science, cryptography, economics, finance, and politics. Although blockchain has great potential, but it is not a "panacea". Don't try to solve all problems with blockchain.

In this notebook, I will try to use plain language to explain the blockchain technology. When it comes to professional terms, a certain visual description (bold and italicized text) will be provided to help readers who do not have computer science background.

For blockchain technology discussions, project cooperation or blockchain related work, please contact: xuanf@usc.edu

Bitcoin Cryptography

Cryptocurrencies are not encrypted. They are unencrypted ledgers that are completely public such as account addresses and transaction information. The Bitcoin system mainly uses two concepts from cryptography. Hashing and digital signature.

1. Hashing

The "hashing" refers to the cryptographic hash function, do not confused with the general hash function used in hash table.

Hash table is a data structure in computer science that can improve the efficiency of finding data. You can understand it as a way to place things, which allows you to quickly find what you need later on. For example, if you have a lot of things to be stored in a sequence of numbered drawers.

Hash function in general can be viewed as a black box processor, you put the item into it, the black box will send out a drawer number (this number is the hash code), and then you can find the corresponding drawer according to this number.

The hash function has three important features. The first is called collision resistance. This is not to say that there will be no hash code collisions. The collision resistance means that we cannot create this kind of collision artificially and efficiently. For example, in the case of $x \neq y$, you can only find a pair of x and y through brute force to let Hash(x)=Hash(y). Brute force solution is unrealistic in reality.

Hash code collision means that you put two different things into the black box processor, and then the black box processor sends out the same drawer number (hash code) for these two different things. If you put these two things in the same drawer, the two things collide to each other, so called a hash collision. A well-designed hash function should minimize the possibility of collision.

Brute force is the term for the method of exhaustion in computer science. It is to try all the possibilities one by one to see which is the answer.

So, what is the significance of this collision resistance in reality? Now imagine that we have a message, we put the message into the hash function to find the hash code (Hash (message) = 00000123), now if someone modifies the message to make it a message', we put this message' into the hash function (Hash (message') = 00000527), we would see the hash code changed accordingly so that we can detect whether the original message has been tampered with or not.

Let's take a look at another scenario, if your bank account file is uploaded to a cloud storage server and you download it back in the future, how do you ensure that the content of the file you downloaded is exactly the same as the file you uploaded before and has not been

tampered with? It's very simple. You can check the hash codes of the file before uploading and after downloading.

Note that we have not proven the collision resistance feature at the mathematical level. This feature is derived from practice. Cryptography experts have not yet found a way to create hash collisions artificially and efficiently. However, if the hash function you implemented by yourself is very simple, it is of course easy to crack. For example, people have cracked a hash function called MD5. There are many different implementations of hash functions. Some methods have withstood the test of time, and people have not succeeded in cracking them.

The second feature of the hash function is called hiding, which means that the calculation process of the hash function is irreversible. You can put an object in hash() to get the hash code, but you cannot get the original object through the code. We can theoretically find the original object through brute force, but it is not practical in most cases.

This hiding feature can be understood as we drove from the starting point to the destination through a one-way lane. After we arrived at the destination, we could not go back through the one-way lane to find out our starting point.

By combining collision resistance and hiding, we can achieve digital commitment (digital equivalent of a sealed envelope). For example, we have a stock master who wants to predict the rise and fall of a certain stock tomorrow, if the master publicly stated on TV this behavior will affect the market and hence the results is not objective. Therefore, we can ask the master to put the prediction result in a sealed envelope in advance, and then open it after the stock market closes the next day to see if the prediction is accurate or not. Now, let's think how we can achieve the prediction on the digital level?

We can input the prediction result into the hash function to get the hash code and due to the hiding feature of the hash function, we can publish the hash code with confidence, because people cannot figure out the prediction result through the hash code. Then we announced the prediction result the next day and due to the collision resistance of the hash function, we can assured that the published prediction result is the one that was put into the hash function the day before.

But please note that the condition for the hiding feature is that our input space is large enough and evenly distributed. For example, if we predict that a certain stock will have a 10% raise tomorrow. We cannot simply put the name of the stock in the hash function and get a hash code. Others can easily figure out which stock name the hash code was generated from by using a brute force solution, because there are only a few thousand stocks listed. Putting the name of each stock into the hash function can easily get the answer. Therefore, we usually add a random number after the original object to increase the difficulty of being cracked. Hash (object + a random number).

The third feature is called puzzle friendly, which means that if you don't put the object in the hash function, you cannot predict the hash value of the object. If you want to get a specific hash code value, you can only put objects in the hash function one by one and try it out. This clearly requires a lot of work. So, what is this third feature used for? It is needed for mining. There are many fields in the block header, one of which is called nonce field. The essence of mining is to replace the nonce field one at a time until a hash code value belonging to the target space (Fig 1) is calculated.

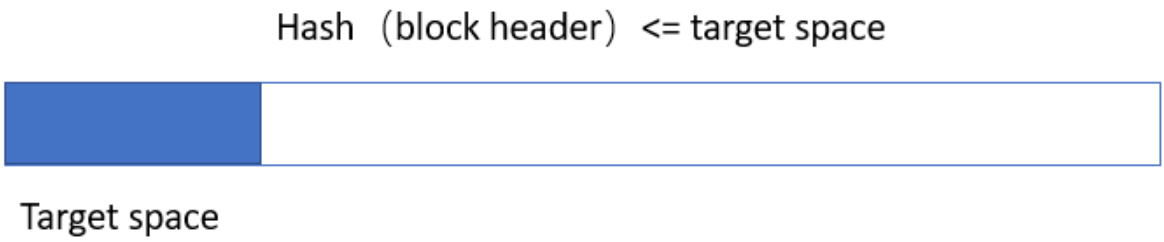


Fig 1

This puzzle friendly feature makes this trial process inevitably require a lot of work, so we call it proof of work. During the mining process, if a node says that it has found a nonce that meets the requirements, it will pack a block with the nonce and send it to the Bitcoin network. After other nodes received this block and they would take out the nonce and use the hash function to verify whether the nonce is correct. The verification process is very simple. Therefore, it can be said that mining is designed to be difficult to solve but easy to verify. The hash function used in Bitcoin is called SHA-256 (secure hash algorithm). This SHA-256 satisfies the all three characteristics, collision resistance, hiding and puzzle friendly.

2. Digital Signature

We have finished the hashing part, now it is time to move on to digital signature, but let's talk about the Bitcoin account first. In our daily life, if we want to open an account, we have to go to the bank with ID documents. This is a centralized process. However, in the Bitcoin system anyone can locally generate a pair of public key and private key. This pair of keys is your Bitcoin account. The concept of public key and private key comes from asymmetric encryption algorithm.

The concepts of public key, private key and (a)symmetric encryption here are all from cryptography, which is a set of methods used to encrypt information to protect communication security.

The characteristics of dual-key encryption are as follows:

a) There is a one-to-one correspondence between public keys and private keys. If there is a public key, there must be a unique private key corresponding to it, and vice versa.
b) All public key and private key pairs are different.
c) The public key can unlock the information encrypted by the private key, and vice versa.
d) It is easy to generate the public key and the private key simultaneously, but it is all most impossible to derive the private key from the public key.

In fact, we used symmetric encryption algorithm at the beginning, which means that the sender and the receiver have the same encryption key. After the sender uses the encryption key to encrypt, the receiver uses the encryption key to decrypt. The premise of this system is that there needs to be a secure channel that can send the encryption key to the two parties who want to communicate with each other. This encryption key distribution is obviously inconvenient. How can we ensure that it will not be stolen by others during the distribution process?

In order to solve this problem, we evolved into asymmetric encryption algorithm. We use public key for encryption and private key for decryption. If A wants to send a message to B, A needs to encrypt it with B's public key, and B can decrypt it with his private key after receiving it. Because the public key does not need to be kept secret, it can be shared with everyone. The person who sends you the message knows your public key. Use your public key to encrypt the message to you. After you receive it, use your own local private key to decrypt it. When you want to reply to this person, you encrypt it with his public key, and after he receives it, he decrypts it with his local private key. We don't need to know the other party's private key, so that we can solve the key distribution concern occurred in a symmetric system.

Here everyone can imagine that you and your girlfriend are writing love letters to each other, and you don't want the love letters to be secretly read by the postman. Therefore, you guys came up with an idea that each of you bought a bunch of locks (equivalent to your respective public keys) and put them in each other's home. Then you each hold a key that can open your own lock (that is, your own private key). Now that you have finished writing your love letter and want to mail it to your girlfriend, you use the lock (the other party's public key) that your girlfriend had put in your house, lock the letter in a box, and then give it to the postman. After the postman gives the locked box to your girlfriend, she can use her key (the other party's private key) to unlock the box and read the content of the love letter. Then your girlfriend wrote you a reply, and she locked the box with the lock you had put in her house (your public key), and then gave the box to you through the postman. You can use your own key (your private key) to unlock the box and read the reply. In this process, both of you do not need to show your own keys (the private keys of both parties), but your locks (the public keys of both parties) can be distributed at will. It doesn't even matter if you throw the locks on the street and let people take it cause public keys are open to everyone.

In the Bitcoin system, we can create a bitcoin account locally by generating a pair of public key and private key. The public key is similar to the account number, and the private key is similar

to the account password. In order to prevent impersonation, we have to send out TX information as follows:

1. I want to send 10 bitcoins to Bob, so I generated "TX information".
2. This "TX information" must first be hashed to generate a "digest", and then I will use my private key to encrypt the “digest” to get the "signature".
3. I posted the "TX information" and "signature" to the Bitcoin network together.
4. After others receive my "TX information" and "signature", they will use my public key to try to unlock the "signature" to get the "digest". If it is successfully unlocked it proves that this "signature" is indeed generated by me. Then others need to hash the "TX information" to get the "digest" again. If the “two digests” are consistent, it proves that the "TX information" has not been tampered with.

For more detailed information on asymmetric encryption please visit：

What is a Digital Signature? (youdzone.com)

Bitcoin Data Structure

1. Hash Pointer

Pointer stores the starting address of an object in memory. In addition to storing the starting address, a hash pointer must also store the hash value of the structure. We use H() to represent Hash pointer. The hash pointer not only allows us to find the memory location of the object, but can also show whether the structure has been tampered with or not.

Data structure is a computer science term. Basic data structures include Array, Linked list, Heap, Stack, Queue, Hash table, Tree, etc. You don't need to be scared by these terms. These data structures are actually digital abstractions of things we encounter in real life, such as the hash table discussed in the previous chapter. In fact, date structure's essence is to tell the computer the way to store data information. For example, a queue structure is just like the queue in real life or after dish washing, you place plates on top of each other this can be viewed as a stack. Each data structure has its best utilization scenario.

Pointer is a special variable in computer science programming language. Its essence is to save the memory address. The computer's memory is piece by piece, and each piece has a corresponding address. You can imagine that the computer memory is like a hotel, each hotel room has a room number, and this room number is the memory address. For example, we have a total of 10 rooms, numbered from 1 to 10. Now we assume that the variable A(Imagine A is a person), and we let A = 10. Then, let's say that A is assigned to room 03, so A runs to room 03 and puts a card with the number 10 on it in the room. Then we define a pointer variable A' (Imagine this is another person), and this A' will go to find A and see which room number A had put the card in. After knowing it, A' will write room 03 on a paper, and then find a free room, such as room 09, and put the piece of paper in room 09. So, we say that the pointer saves the memory address (room number).

The most basic data structure in Bitcoin is the blockchain, which is actually a linked list. Ordinary linked lists are connected by regular pointers, while blockchains are connected by hash pointers. Blockchain is a linked list using hash pointers. Hash pointers can actually replace ordinary pointers as long as the structure is "acyclic". Singly linked lists can use hash pointers, but circular linked lists cannot use hash pointers.

A linked list is a basic data structure in computer science. It is composed of one node connected to the next node. Data can be stored on each node. You can imagine that there is a group of people(nodes) in a line, and everyone puts their hands on the shoulders of the next person. This action is to "link" up. Every person is also carrying a backpack (Store data on nodes) which can hold all kinds of things.

The first block is called a genesis block, and the block just appended to the blockchain is called the most recent block. Each block has a hash pointer that points to the previous block (Fig 2).

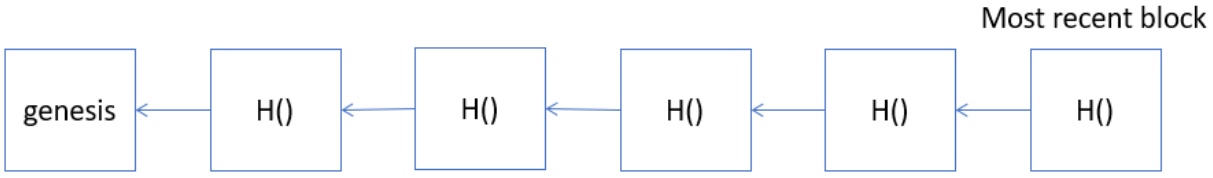


Fig 2

Each hash pointer is calculated by taking the previous block as a whole, meaning that it is calculated together with the hash pointer in the previous block. Therefore, this data structure allows us to easily check whether a block has been tampered (tamper-evident log), because as long as a certain part of the blockchain is changed, this change will propagate via the chain all the way to the most recent block (Fig 3).

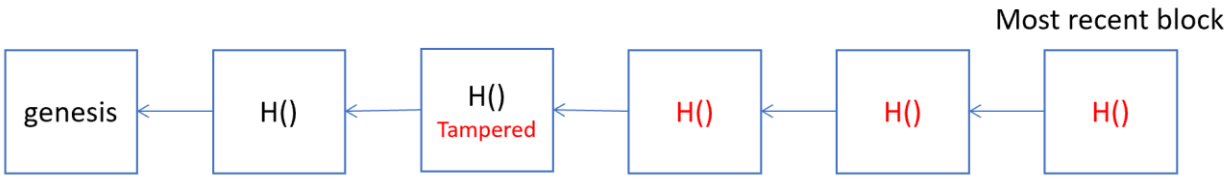


Fig 3

Therefore, we only need to observe the H() of our most recent block to check whether any block has been tampered with. Because of this feature, some nodes in the Bitcoin system do not need to store the content of the entire chain, they can only store the last few thousand blocks. They can go ask for help from other nodes in the system when needed. Even if there is a malicious node, provided you a tampered blockchain. You can easily tell by checking hash codes.

2. Merkle Tree

Another data structure in Bitcoin is the Merkle tree (Fig 4). The difference between the Merkle tree and the binary tree is the use of hash pointers.

The tree structure is a basic data structure in computer science. You can think of your own family tree. It like a tree growing upside down, isn't it? Starting from your ancestors, keep growing branches. The requirement of binary tree is that each generation can only have 2 children, and each child can only have 2 children. And so on.

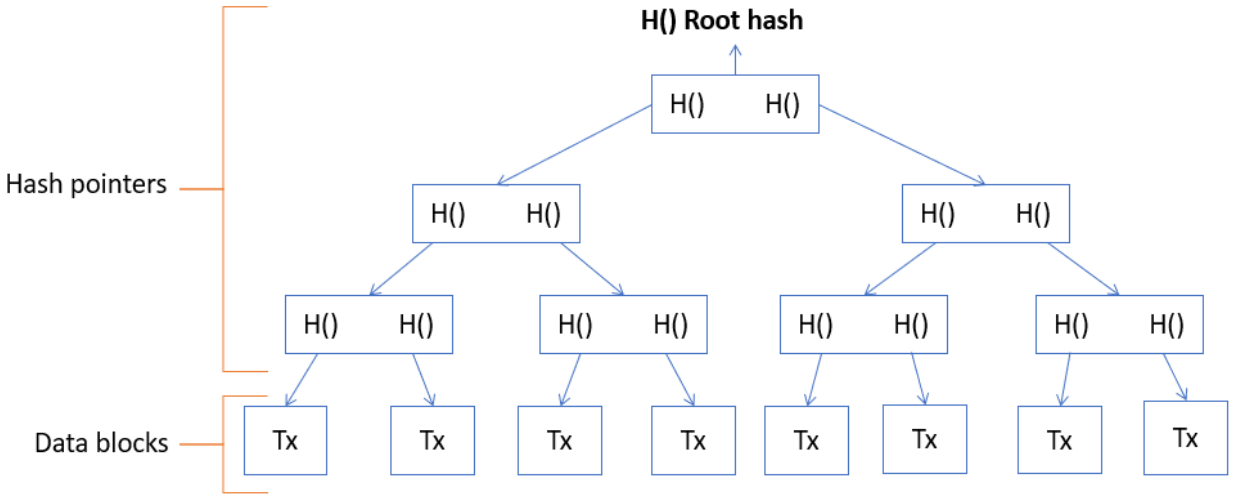


Fig 4

The benefits of Merkle tree data structure are the same as hash pointer linked list. As long as we remember the true root hash value, we can detect whether the information in the tree has been tampered with.

Blocks in Bitcoin are connected by hash pointers, and all transaction information contained in each block is stored in the block in the form of a Merkle tree. Each block is divided into block header and block body. The root hash value is stored in the Block header, but there is no specific information about the transaction. The specific transaction information is stored in the block body.

Bitcoin nodes are divided into full nodes and light nodes. Full nodes store the entire blockchain, and light nodes (such as the Bitcoin wallet app on a mobile phone) only store block headers.

Then the question is, how does a light node can tell a certain transaction does exist on the blockchain? This requires another use of Merkle proof. The light node only knows the root hash value, and now the light node has to prove whether the transaction marked in yellow in the figure 5 exists. The light node will send a request to a full node, requesting the full node to provide him with a Merkel proof. The full node will send the 3 hash values marked in red to the light node. Then the light node can calculate the green hash codes bottom up starting by hashing the yellow Tx, and finally obtain a root hash (Marked in green) code value, and then compare it with the root hash in the block header saved by itself (Marked in black).

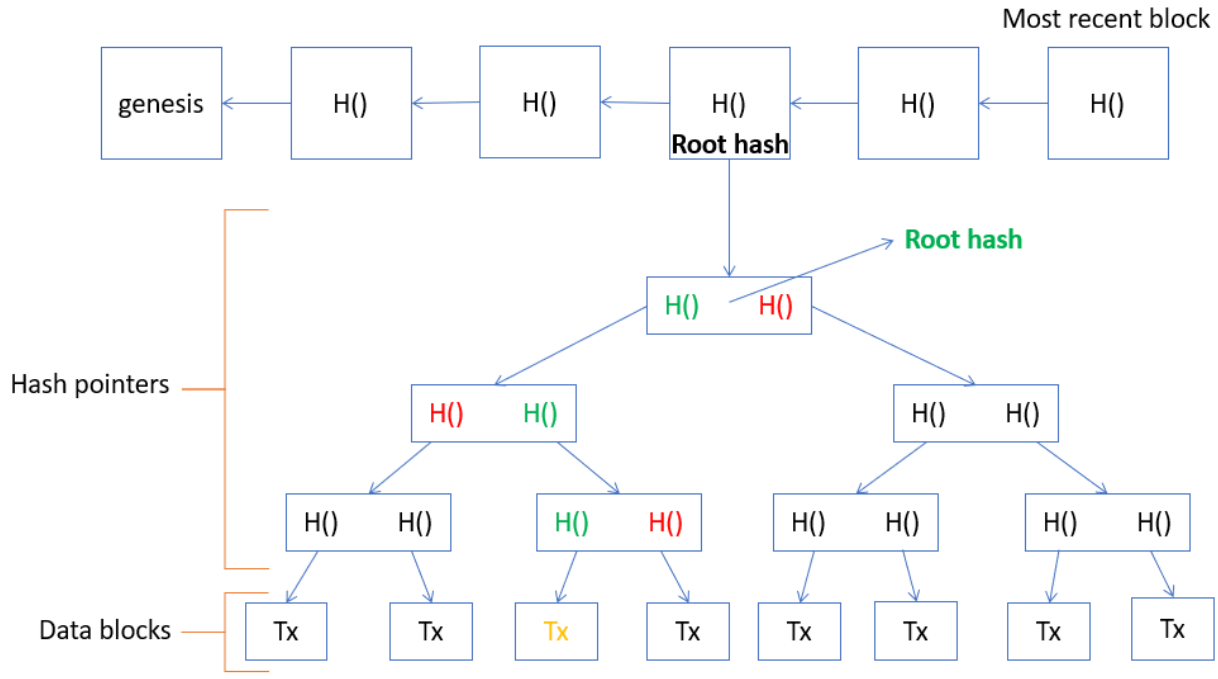


Fig 5

Some people may wonder, can we tamper with the hash code values provided by the full node and try to match the calculated root hash and stored root hash? It is feasible in theory, but in practice it is difficult because of the collision resistance feature, it is very difficult to artificially create hash collisions. Therefore, Merkel proof is reliable to prove whether a transaction actually exists in the blockchain. This verification process is also called proof of membership/inclusion, and the time complexity O(log n).

Can it prove that a certain transaction does not exist? Yes, but it is troublesome and inefficient. The worst way is to pass the entire tree to the light node, but this time complexity is O(n). Is there an efficient way? The answer is no, unless the we had sorted leaf nodes of Merkle tree. Then, we can reduce time complexity to O(log n). Bitcoin did not implement a sorted Merkle tree.

Bitcoin Protocol

1. Digital Currency System Design

Let's put centralization concerns aside for now, and think about how to design and issue digital currency from a central bank's perspective. The current central bank is a centralized institution. It can use the concept of public / private keys pair to issue digital currency. Suppose that the central bank issues a $100 digital currency file (Fig 6) and digitally signs it with the central bank's private key. Now this document is sent out to the people. If A buys something worth $100, then A will transfer this $100 digital file to the seller. After receiving the file this seller can use the central bank's public key to verify that this file was indeed sent out by central bank. Now, the question is does this design work?

Fig 6

The answer is no. Although people can't tamper the content of the file.

Why can't we change the content of the file? Such as turning 100 bucks into 1000 bucks? This is because the $100 digital currency file needs to go through the hash function to generate the digest of the file first, and then the central bank uses its private key to sign the digest, not the original file. The reason that this design does not work is because people can copy and paste the file. The copied file also has a certifiable central bank's signature and this creates the concern of double spending attack.

Therefore, a successful digital currency system must be able to solve the double spending attack concern. Now let's try to improve our design. This time we add a serial number to each $100 file, just like the serial number on real paper bills, and then the central bank builds a database to record the ownership of each $100 file (Fig 7).

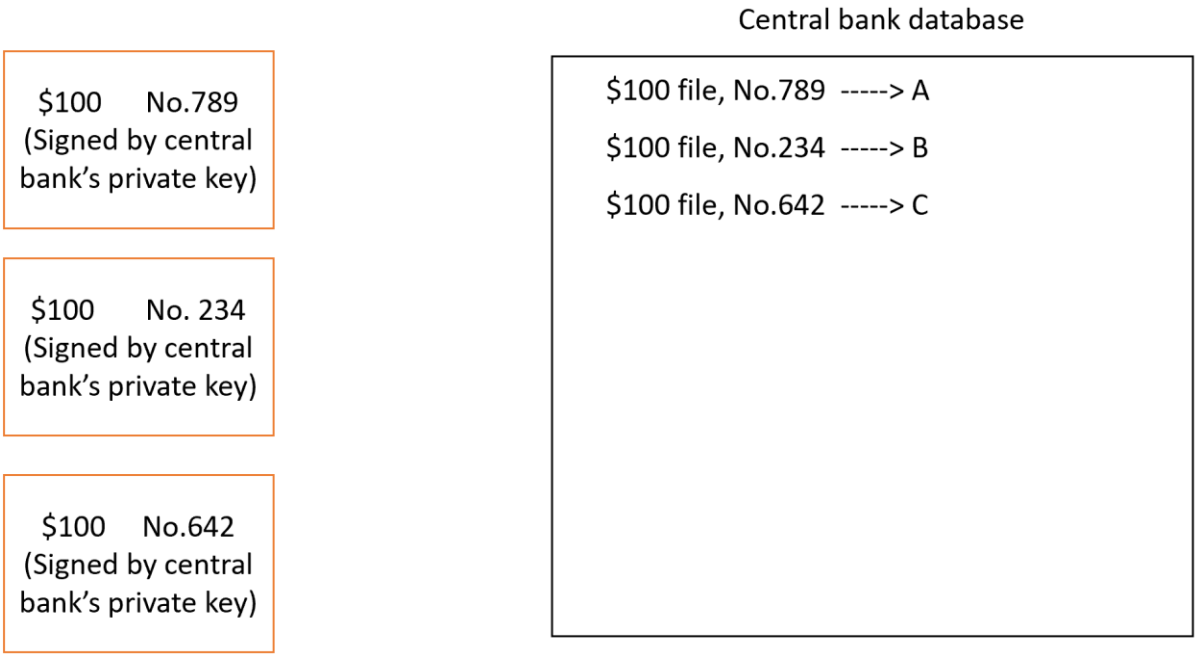


Fig 7

When spending money, not only must the signature be verified, but also the database must be checked to see if the $100 digital currency being spent is actually in the hands of that person. After confirming transaction, the database must be updated. If the person wants to double spend the previous $100 digital currency, by checking with the central bank database we can easily tell that the $100 file has already been spent and is no longer in this person's hands. This updated design prevents people from double spending. Is there any concern with this design? Not really, this design is feasible, but it is an absolutely centralized design. Now let's think about how Bitcoin turned this design into a decentralized manner.

The two major problems that a decentralized digital currency must resolve are as follows:

1. Who has the power to issue digital currency? How much and how often should the currency be issued?
2. How to prevent double spending attack?

Bitcoin solves the first problem by mining. It will be discussed in later chapters. Let's talk about the second question first, how Bitcoin solves the double spending attack. Bitcoin's solution is similar to the centralized design we mentioned above. It also requires a data structure to keep a record of who spent the Bitcoin. The difference is this time the data structure is not maintained by a centralized organization, instead by all users. This data structure is called blockchain.

Suppose that user A obtains the right to issue the next block, the bitcoin system will mint 10 bitcoins and give it to A as reward, this minting process is called coinbase TX (Fig 8). We will

write this transaction information into the blockchain. Then, A transfers these 10 bitcoins to B and C, each with 5 Bitcoins. This transaction requires A's digital signature (A needs to sign the TX with his private key, and other nodes need to verify the sig with A's public key). This transaction also needs to point out the source of the 10 bitcoins being spent here by using a hash pointer. In our case, it came from the coinbase transaction.

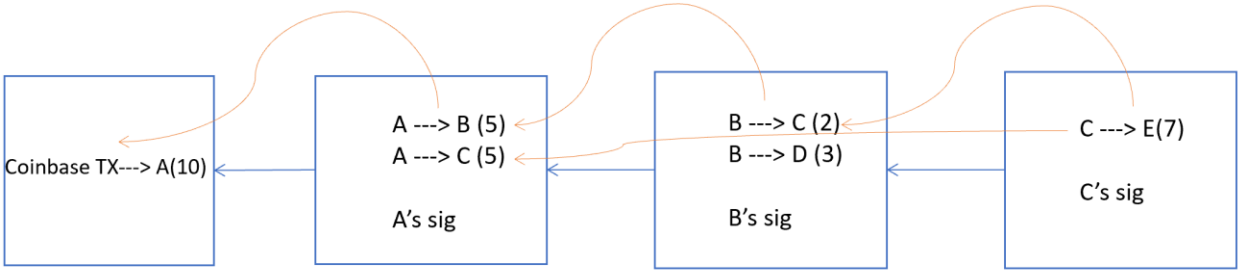


Fig 8

Note that in the Bitcoin system, each transaction contains two parts, inputs and outputs. The input part must indicate the source of the Bitcoin and show the public key of the sender. The output part must have the hash code of the receiver's public key.

Going back with our example above, B transfers the 5 Bitcoins he received from A, to C and D. Of course, B needs to sign with his own private key and explain the source of the 5 Bitcoins. The source of B's 5 Bitcoins is the from the transaction from A to B. Now C wants to transfer his total of 7 bitcoins to E, this time it is a bit complicated to specify the source, because 2 Bitcoins are from B and the other 5 Bitcoins are from A.

Up to this point, we have formed a small blockchain. In this small blockchain, we use two types of hash pointers. One type is to link the blocks, and the other is to indicate the Bitcoin sources, which is to point to a previous transaction.

What is the purpose of these hash pointers that indicate the source? They are used to prevent double spending. For example, if B wants to double spend the 5 bitcoins obtained from A, B would put the TX information in the fifth block and signs the transaction (Fig 9). The hash pointer would point to the transaction from A to B in the second block.

After receiving this fifth block, the other nodes begin to verify it, starting from the fifth block and check backwards, when the third block is checked, B's double spending attempt would fail, because the hash pointers of the transactions in the third block also point to the transaction that A transfers Bitcoins to B in the second block, which means that the 5 Bitcoins that B wants to double spend in the fifth block have already been spent by B himself in the third block.

Therefore, this double spending attack cannot work.

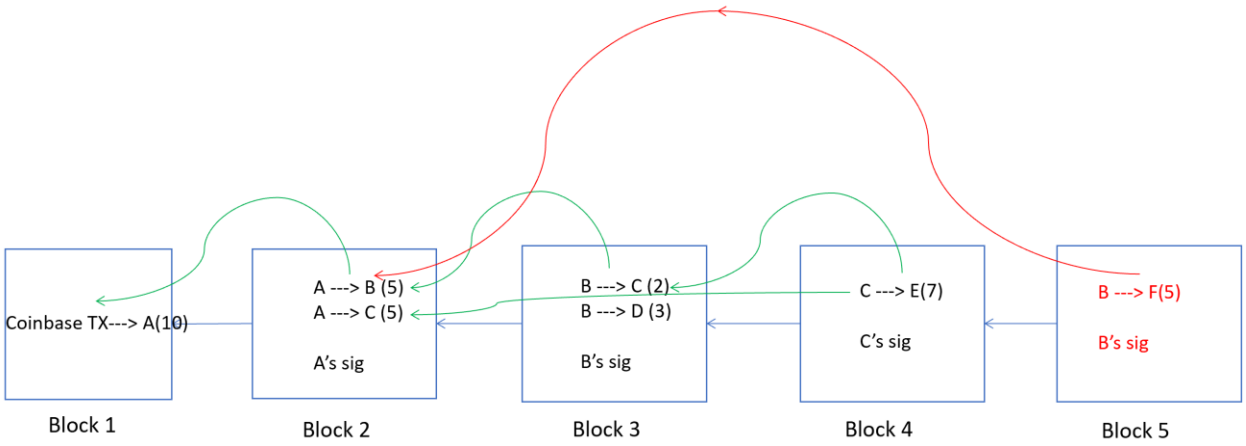


Fig 9

Let's think about the transaction that A transfers to B again. This transaction requires two pieces of information, A's signature and B's address (B's address is B's hashed public key and this address is similar to B's bank account). How did A learn B's address in advance? Just like in real life, the banking system will not help you figuring out someone's account number, and the Bitcoin system will not help you find someone's address. These addresses are either you ask the other party for it, or are known through a third party, such as when selling things, the seller would put their addresses on the trading platform.

What information does B need to know about A? B needs to know A's public key, because A's public key represents A's identity, but more important reason is that all other nodes actually need to know A's public key. Why? Because all other nodes need to use A's public key to verify A's signature. (Private key signs, public key verifies). Each node must independently verify A's signature, because there is a possibility of malicious nodes, other nodes cannot be trusted. Now the question is, how do you know the public key of A? It's actually very simple. Do you still remember the information that the input and output of each transaction should contain? The input part should explain the source of the bitcoin, the sender's public key and the output part should give the hash of the receiver's public key.

Is this safe? How can you trust the input public key actually belongs to A? What if a malicious node B' replaces A's public key with its own public key and pretends to be A? In the initial coinbase TX the output of this transaction contained the hash of A's public key. Therefore, when A initiates the transfer to B and attaches its own public key, every node will trace the source to the initial coinbase TX and they can find the hash of A's address at the time to see if it is the same as the input public key. If B' pretends to be A and put B's public key in the input field, the hash codes would differ.

To emphasize, the encryption of the message sent to the receiver is encrypted with receiver's public key, and the receiver uses their own private key to decrypt the message after receiving

it. When the receiver replies to the sender, the receiver will encrypt the message with sender's public key, and then the sender will decrypt the message with sender's own private key after receiving the message.

The verification process above is implemented through scripts in the Bitcoin system. The input and output of each transaction is a script. The verification is to combine the input script of the current transaction with the output script that provides the source of the Bitcoin to see if it can be executed without error (The input part should explain the source of the coin, the public key of the sender, and the output part should give the hash of the receiver's public key). Later, we will talk about the bitcoin script in detail.

The figure 9 above is simplified because there is only one transaction in each block, in reality there are many transactions in each block, and these transactions together form a Merkle tree. As mentioned earlier, each block is composed of block header and block body (Fig 10). The Block header will include which version of the Bitcoin protocol is currently being used, the hash pointer to the previous block, the root hash value of the Merkle tree, and two fields related to mining, one is mining difficulty target, the other is the random number field nonce. nBits is the target domain encoding value.

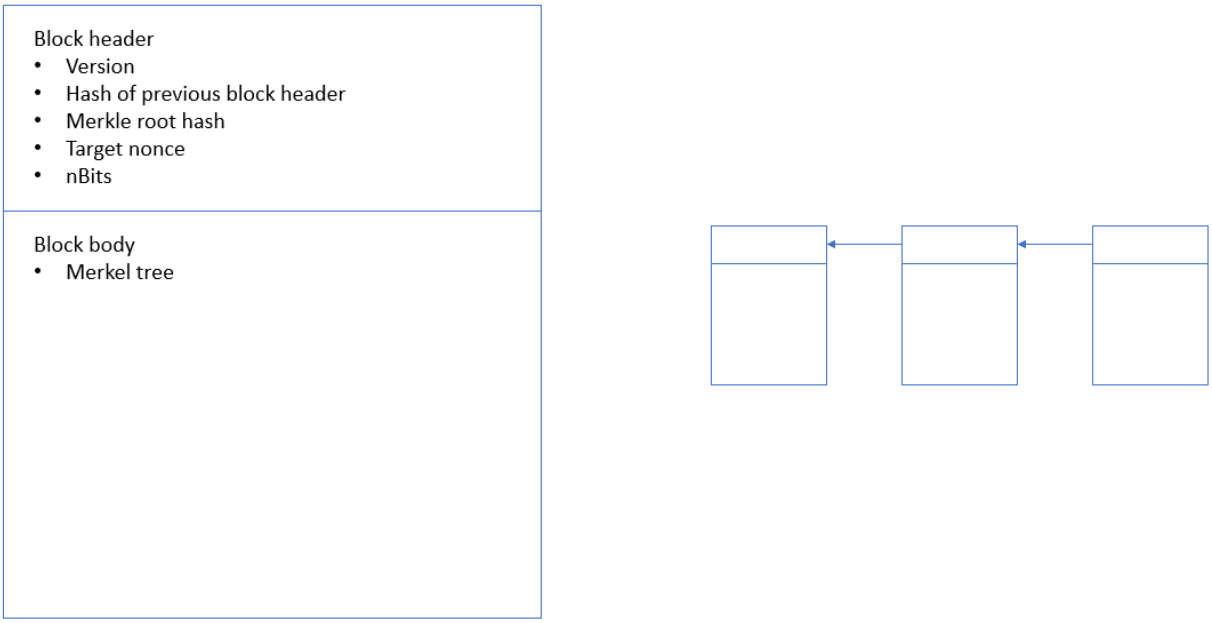


Fig 10

Note that our hash pointer to the previous block is actually obtained based on the block header of the previous block, not the entire block, so the blockchain is actually connected by a serial of block headers. The block body is used to store TXs because the root hash value of the Merkle tree contained in the block header can ensure the information in the block body cannot be tampered with.

As mentioned earlier, there two types of nodes, full nodes and light nodes. Full node will save the entire blockchain and verify transaction information, also known as fully validating node. The light node only saves the block header information. Generally speaking, the light node cannot independently verify the legality of the transaction. For example, the light node does not know whether a transaction is double spending. In fact, most of the nodes in the system are light nodes and these nodes do not really participate in the construction and maintenance of the blockchain, they only pull information out of the blockchain for the verification purpose.

How exactly is the blockchain built and how to decide what information to be written into the next block? Bitcoin is actually a decentralized ledger. Each node cannot just book keeping their own ledgers. We have to achieve a distributed consensus between all nodes. A simple example of distributed consensus is a distributed hash table. What does it mean? There are multiple computers in the system that jointly maintain a global hash table. The content that these computers need to reach a consensus for the key-value pair of the hash table, which means that if computer A reads a key "001" and gets "xuan" as value, when computer B reads key "001" it should also get "xuan" as value.

The concepts of distributed systems are involved here, and there are many FLP impossibility results in distributed systems. For example, in an asynchronous system, even if only one member is faulted, then consensus cannot be reached. Another famous conclusion is CAP theorem, which means that only two of the following three characteristics can be satisfied at the same time.

- Consistency
- Availability
- Partition tolerance

A well-known agreement for distributed consensus is Paxos. This agreement can achieve consistency, but it may not be able to achieve consistency in some cases. The above theories have little to do with the actual application of Bitcoin hence we stop the discussion here.

2. Bitcoin consensus

One of the concerns that the Bitcoin network must deal with is that there must be malicious nodes in the network. How should we design a consensus protocol to ensure safety? Assuming that most nodes are good and malicious nodes are a minority, can we reach consensus through voting by number of accounts? Let's assume a certain node proposes a candidate block, and everyone just votes on it. Any voting system needs to confirm membership first, and only approved people can participate in voting. However, creating an account is really easy in the Bitcoin system. People can generate thousands of public key and private key pairs quickly. After we generated more than half of the total number of accounts, we will naturally be able to control the Bitcoin system. This type of attack is called a sybil attack. Therefore, this number of accounts-based voting design will not work.

How does the Bitcoin system solve this problem? Also, via voting, but instead of account-based voting, Bitcoin vote by computing power. Each node can form a candidate block locally and put transactions that it considers legal in the block. Then, the node will start to try various nonce values to see which nonce can find a hash value that falls within the specified target range (Fig 11). If a node finds a matching nonce, this node will obtain the right to append its candidate block to the blockchain. And in this block, the node can include a coinbase TX to itself for receiving its block reward.

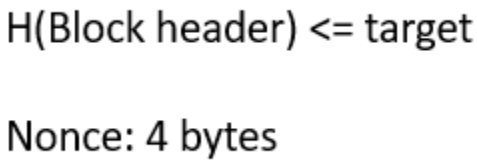


Fig 11

After other nodes received this block, they will start to verify it. There is a field in the block header called nBits (the target domain encoding value) Verification process includes checking if nBits field meets the difficulty requirements in the Bitcoin protocol, checking if the hashed nonce in the block is within the target space, and checking transactions in the block body to see if each transaction is legal (check whether the signatures and the source of the Bitcoin are all legal). If anything is illegal, the block will not be accepted by other nodes. After a node receives a block, how does it know where to append the block? Very simple, look at the hash pointer that the block points to the previous block.

If a new block has gone through all the checks, will it be accepted and written into the blockchain for sure? Suppose we have a node that has obtained the right to append and publishes a block. This block needs to be appended at the middle of the blockchain not the end and the block header and block body are legal, should we accept it（Fig 12）?

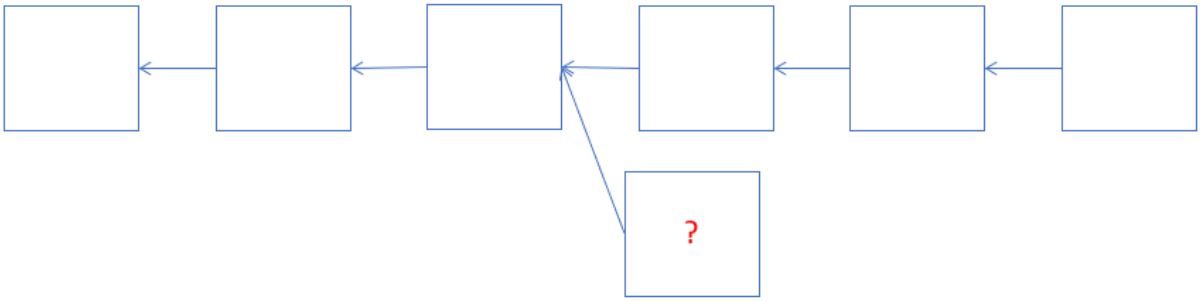


Fig 12

To answer the questions above let's first take a look at the following example (Fig 13). There are three transactions on the blockchain. C transfer to A; A transfers to B and A transfers money

to another account A' that belongs to A as well, but this TX happens on a side chain. Remember, when we judge whether double spending has occurred, it depends on if the two TXs happened on the same chain or not. In our case, A-A’ is not double spending because the A to B and A to A’ happen on different chains. For example, to check the transfer from A to A', we need to trace back from this side chain all the way to the source of C to A, and along this tracing back, A hadn’t transferred Bitcoin to others, so the transfer A to A’ is actually legal. However, please note that although A to A' is legal, this transaction is not on the "longest valid chain". The Bitcoin system stipulates that the accepted block should be extending the longest legal chain. Hence, the TX A to A’ is legal but useless.

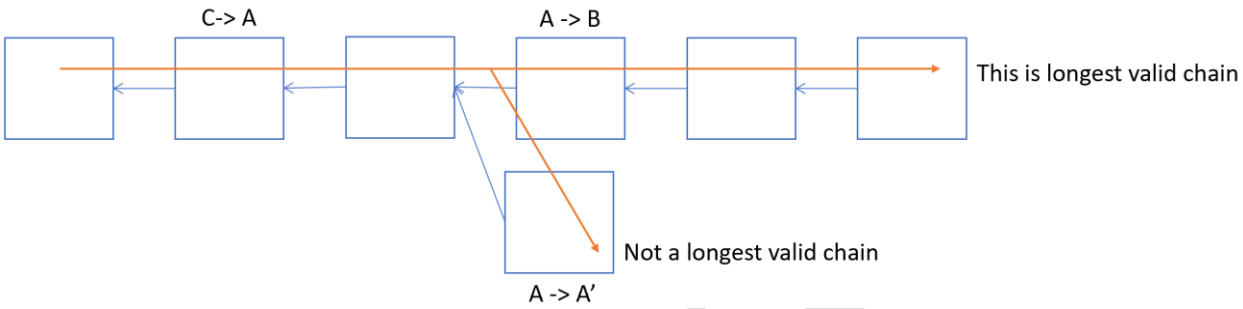


Fig 13

The case above is actually called forking attack, which attempts to roll back a transaction that has already occurred on the longest legal chain by inserting a legal block in the middle of the blockchain. However, the blockchain may also fork during normal operation. For example, if two nodes have obtained the right to append next block at the same time (Fig 14), and both have sent out their own blocks. In this case there will be two chains of equal length. The two chains are both longest legal chain, so which one should we accept? Other nodes in the Bitcoin network, due to their different network locations, some nodes may receive the upper block first, and some may receive the lower block first. Therefore, the situation of two legal chains of equal length may last for a period of time, but in the end, one will win (Fig 15).

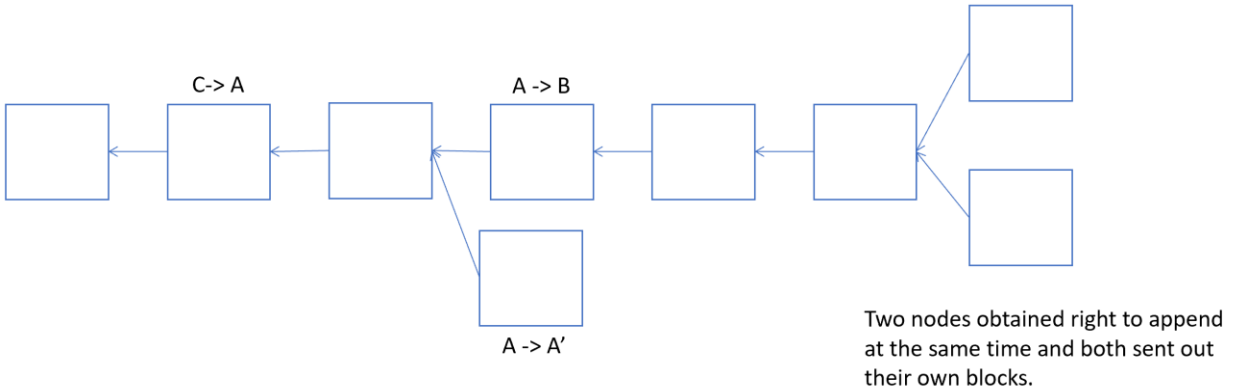


Fig 14

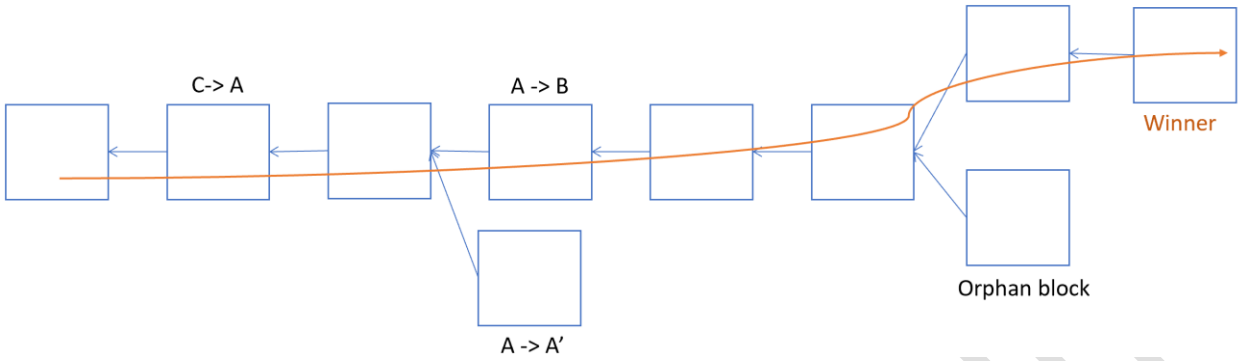


Fig 15

Note that the Bitcoin protocol uses implicit consent. This means that if a node continues to append after the block, you gave out, it means the node accepted your block. The winning chain becomes the longest legal chain, and the losing block is discarded as an orphan block.

Why do these nodes take time and efforts to obtain the right to append the next block? As mentioned before the node that has the right to append has the power to decide which transactions will be written into the block, but this is not the main reason we hope all legal transactions should be written into the block. Bitcoin actually rewards the nodes who obtained the right to append through block reward which is a special coinbase transaction that can be written in the block to award the node. This coinbase transaction is the only way to issue new bitcoins in the bitcoin system. In the early days of Bitcoin, each node that obtained the right to append could be rewarded 50 Bitcoins, but the Bitcoin protocol stipulates that after every 210,000 blocks, this reward should be halved (Fig 16).

50 BTC -> 25 BTC -> 12.5 BTC

Fig 16

Note that in the case above, the node who obtained the right to append but lost the longest valid chain champion, its published block would become an orphan block, which makes the block reward in this block useless because other nodes would not approve it. Other nodes only approve the longest valid chain.

What exactly is the consensus to be achieved in Bitcoin system? In fact, it is the longest valid chain of Bitcoin. This is the “ledger”. Obtaining the right to append requires solving the puzzle, and solving the puzzle requires computing power. The greater the computing power, the greater the possibility of obtaining the right to append. The so-called computing power is actually the number of possible nonces that can be tried per second, also known as hash rate.

So how does this design prevent sybil attacks? The right to append depends on the amount of computing power, not the number of accounts, so even if a person creates hundreds of

accounts, it will not increase the person's hash rate. The essence of so called mining is just solving puzzle trying different nonce values.

Bitcoin System Implementation

1. UTXO

The Bitcoin system is a transaction-based ledger. It does not specifically record how many Bitcoins each account has. If we want to know how many Bitcoins account A has, we have to use A's past transaction information to Extrapolate. The full nodes in the Bitcoin system need to maintain a data structure called UTXO (Unspent transaction output), which is the transaction outputs that have not yet been spent. There are many transactions on the Bitcoin blockchain, some of the outputs of the transactions are spent, and some are not. The set of the outputs of all those transactions that have not been spent is UTXO. A transaction may have multiple outputs. For example, A transfers 5 BTC to B, and transfers 3 BTC to C. After B got the BTC, he spent it. After C got the BTC and did not spend it, only the transaction output from A to C is in UTXO (Fig 17). Each element that exists in UTXO must give the hash value of the output of the transaction it belongs to, and the number of output that the element is in the transaction.

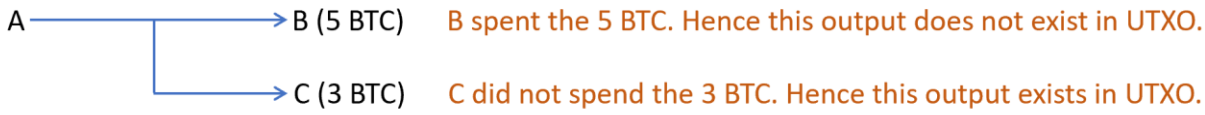


Fig 17

So, what is this UTXO used for? It is used to check double spending. For any new transactions, you need to check UTXO first, because the bitcoin you want to spend must be in the UTXO. All full nodes need to maintain the UTXO structure in memory to quickly detect double spending. Each transaction can have multiple inputs or multiple outputs, and the number of bitcoins for all inputs must be equal to the number of bitcoins for all outputs. The example we gave in the above figure is an example of multiple outputs and a single input. Note these multiple inputs do not necessarily come from the same address, but can come from different addresses. Therefore, a transaction may require multiple signatures, because each input address must provide a corresponding signature.

The total inputs of each transaction will be slightly larger than the total outputs, and the difference is the transaction fee, which will be paid to the node that has the right to append the next block. As we said before, nodes compete for the right to append in order to obtain coinbase tx rewards. The tip obtained by packaging other people's transactions is also a reward to encourage nodes that have obtained the right to append to record this transaction because if there was no transaction fee reward, the node with the bookkeeping right would have no incentive to help others write the TXs into the block.

2. Block Example

On average, the Bitcoin system generates a new block every 10 minutes. After every 210,000 blocks, the block reward is halved (210,000 X 10 minutes = halved every 4 years). Bitcoin is a

transaction-based ledger and there is another design called account-based ledger which is being used in Ethereum. The account-based ledger records how many coins each account has. The transaction-based ledger of Bitcoin requires that every transaction can be traced back. We have to be able to know the source of the Bitcoin, which transaction it came from and which number of output it is. Since Ethereum is an account-based ledger, there is no need to display the source of the coin. Let's take a look at a real block (Fig 18).

Fig 18

“Height” is the serial number of the block.

“Difficultly” is the difficulty of mining, and the difficulty is adjusted every 2016 blocks to keep the block time around 10 minutes.

“Hash” is the hash value of the block header.

“Previous block” is the hash value of the previous block header.

“Merkle root” is the root hash value of the Merkle tree formed by the transactions contained in the block.

The following is the structure of the block header(Fig 19).

Block header

```
13  /** Nodes collect new transactions into a block, hash them into a hash tree,
14   * and scan through nonce values to make the block's hash satisfy proof-of-work
15   * requirements.  When they solve the proof-of-work, they broadcast the block
16   * to everyone and the block is added to the block chain.  The first transaction
17   * in the block is a special one that creates a new coin owned by the creator
18   * of the block.
19   */
20  class CBlockHeader
21  {
22  public:
23      // header
24      int32_t nVersion;
25      uint256 hashPrevBlock;
26      uint256 hashMerkleRoot;
27      uint32_t nTime;
28      uint32_t nBits;
        uint32_t nNonce;
```

source: bitcoin/src/primitives/block.h

Fig 19

It can be seen here that the type of the nonce field is uint32_t (4 bytes). This means at most there are only 2^{32} possible values. As the difficulty of mining increases, adjusting the nonce field alone may not find us a hash value that meets the requirements. So, which of these fields can be modified (Fig 20)?

Block headers are serialized in the 80-byte format described below and then hashed as part of Bitcoin's proof-of-work algorithm, making the serialized header format part of the consensus rules.

Bytes	Name	Data Type	Description
4	version	int32_t	The **block version** number indicates which set of block validation rules to follow. See the list of block versions below.
32	**previous block header hash**	char[32]	A SHA256(SHA256()) hash in internal byte order of the previous block's header. This ensures no previous block can be changed without also changing this block's header.
32	merkle root hash	char[32]	A SHA256(SHA256()) hash in internal byte order. The merkle root is derived from the hashes of all transactions included in this block, ensuring that none of those transactions can be modified without modifying the header. See the merkle trees section below.
4	time	uint32_t	The block time is a Unix epoch time when the miner started hashing the header (according to the miner). Must be strictly greater than the median time of the previous 11 blocks. Full nodes will not accept blocks with headers more than two hours in the future according to their clock.
4	nBits	uint32_t	An encoded version of the target threshold this block's header hash must be less than or equal to. See the nBits format described below.
4	nonce	uint32_t	An arbitrary number miners change to modify the header hash in order to produce a hash less than or equal to the target threshold. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the merkle root updated.

The hashes are in internal byte order; the other values are all in little-endian order.

source: bitcoin.org

Fig 20

“Version” is the version number of the Bitcoin protocol currently in use. This field cannot be modified.

“Previous block header hash” is the hash value of the previous block header and it cannot be modified.

“Merkle root hash” is the root hash value of the Merkle tree, which can be changed.

“Time” is the time when this block was generated. There is some room for adjustment, because the Bitcoin system does not require precise time. As long as the adjustment is minor.

“nBits” is the target domain value for mining. This can only be modified periodically in accordance with the requirements of the agreement, and cannot be modified casually.

“nonce” is the random number field that we need to try.

When adjusting nonce filed is not enough, we can actually modify the Merkle root hash field. Why the Merkle root hash can be modified? As we mentioned before, there is a coinbase minting transaction reward in each released block, which is the only way to generate new bitcoins in the bitcoin system (Fig 21). The bitcoins obtained through this transaction have no inputs, because these bitcoins are generated by the bitcoin system itself. This transaction has a field called coinbase field. You can write any content in it, no one cares, no one checks. If we change the content of the coinbase field, it will affect the root hash value in the block header.

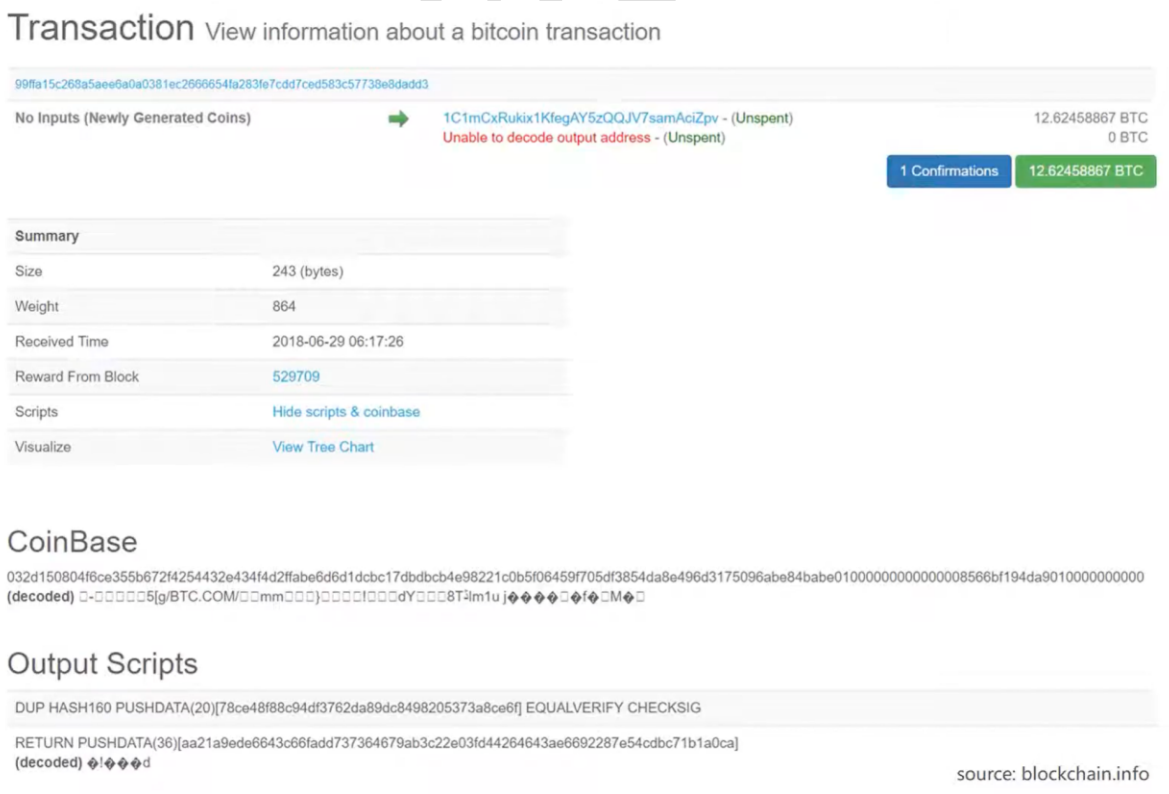


Fig 21

After we changed the coinbase field, the hash value of the transaction is also changed, and then this change would be passed up along the structure of the Merkle tree, eventually leading to a change in the root hash value (Fig 22). So, we can use this coinbase field as extra nonce field. The original nonce field only has 4 bytes (2^32 possible) in size. If it was not enough, we can borrow some additional bytes provided in the coinbase field. For example, we can borrow the first 8 bytes in the coinbase field (2^64). In this way, we can have a total search space of 2^96. Therefore, when mining, there are actually two layers of loops. The outer loop adjusts the borrowed nonce from the coinbase field to calculate the root hash value in the block header, and the inner loop adjusts the original nonce field.

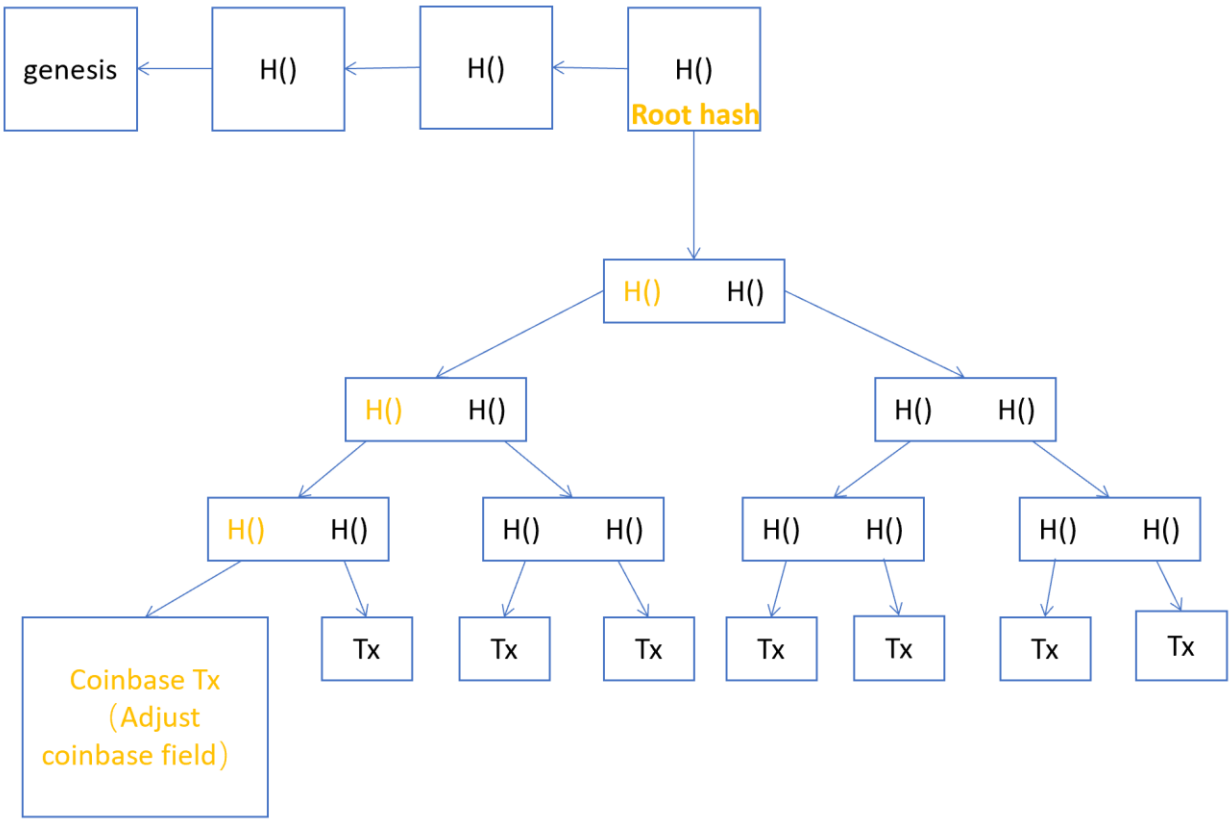


Fig 22

Let's look at a TX case (Fig 23). This TX has 2 inputs and 2 outputs. Although the inputs（The inputs in red box） are marked as output here, don't be fooled. This means that the inputs here are the outputs of other transactions before. In the outputs, you can see that they are all unspent, so they need to be recorded in UTXO. You can also see that there is a difference between the input and output amounts. This difference is the transaction fee. In the bottom of the figure, you can see that the input and output are specified by scripts. Bitcoin verifies the legality of the transaction by pairing the input script with the output script and then executing it. Note that it does not mean that these two scripts are paired, because these two scripts are scripts in the same transaction. Instead, it matches the input script of the current transaction

with the output script of the previous transaction that provided the source of the coin. If the two scripts can be executed together without errors, it means the transaction is legal.

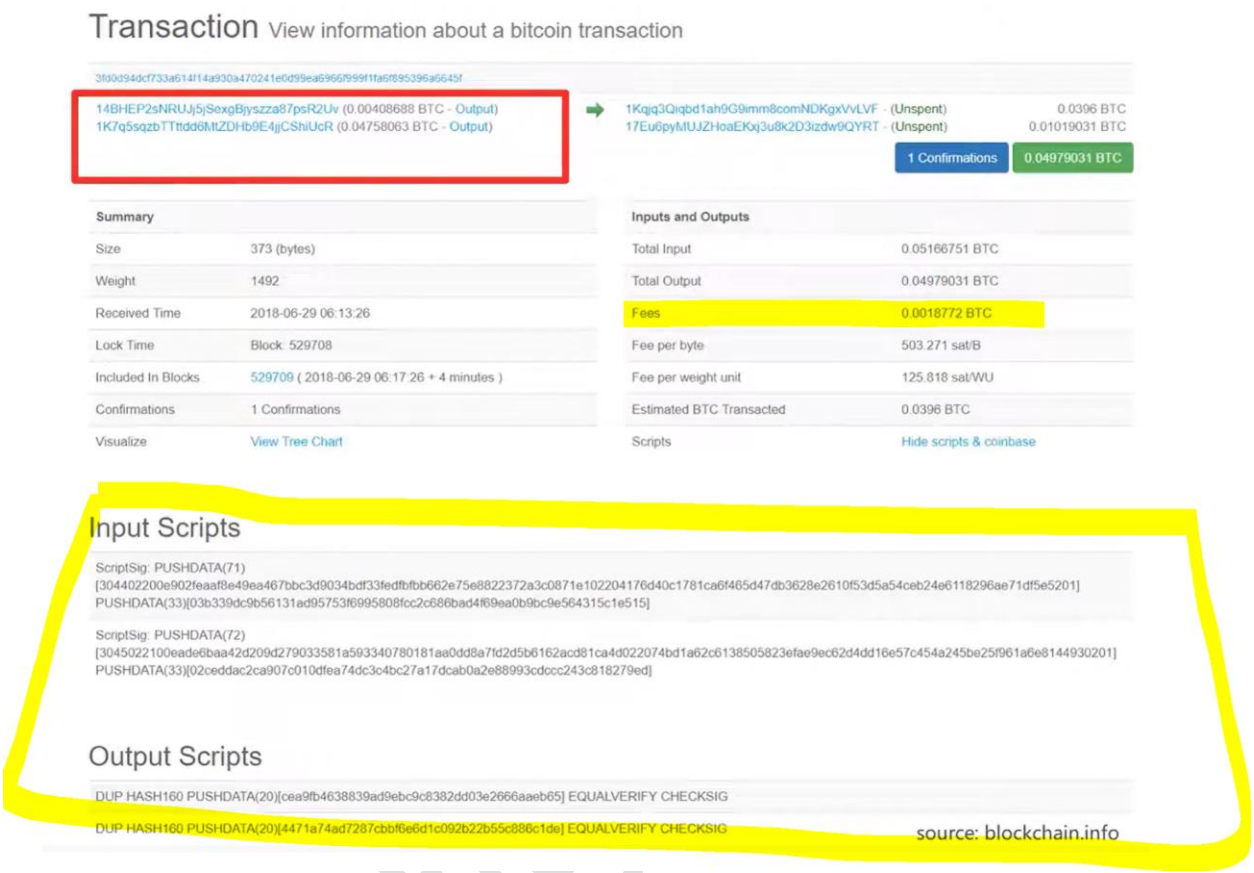


Fig 23

Bitcoin Mining

1. Memoryless

Let's talk about the probability analysis of the mining process. Mining is to constantly try various nonces, and each attempt of a nonce can be regarded as a Bernoulli trail.

The Bernoulli trail is a random experiment with only 2 results. For example, toss a coin. There will only be two outcomes. If the coin is uneven the probability of heads and tails facing up is also different. The probability of one side up is P, and the probability of the other side up is 1-P.

When we mine BTC, the probability of success and fail is very different. Every time we try a nonce, the probability of failing is greater. If we do many Bernoulli trials, these trials will constitute the Bernoulli process.

The Bernoulli process is a series of independent Bernoulli experiments. The feature of this Bernoulli process is memoryless, which means that the results of your past trails will not affect the results of your future trails. For example, if we toss coin 100 times, the result of one toss will not affect the result of next toss.

When there are many ***trails*** but only a few successes, we can use the Poisson process to approximate. When we mine BTC it can be seen that the block time is in accordance with the exponential distribution (Fig 24).

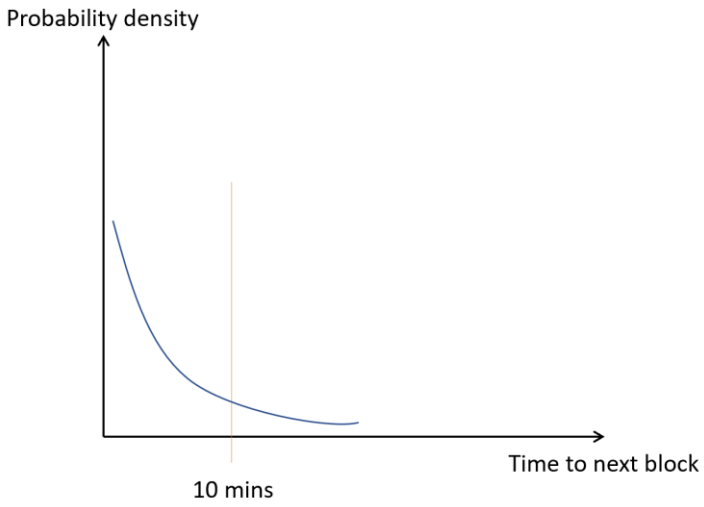


Fig 24

The figure above is the block time of the entire system, not the block time of a specific miner. The average block time of the entire system is around 10 minutes. This average time is designed by the Bitcoin system protocol. By regularly adjusting the mining difficulty, the average block time is maintained at 10 minutes. For each specific miner, the probability that the miner can

mine a block depends on the percentage of the miner's computing power compare to total computing power of the system. For example, if your computing power is 1% of the total computing power, it means that on average for every 100 blocks generated, one of them is mined by you. In other words, it takes you 1,000 minutes to mine a block.

The characteristic of the probability density curve is that if you cut it at any position, the shape of the rest of the curve is the same as the original one. This means that this graph is memoryless. So, what does it mean to us?

For example, we have been mining BTC for 10 minutes already. Since the average block time is also 10 minutes, does this suggest we are going to mine a block soon? The answer is no. We have to wait 10 minutes on average at any point during the mining process, because the memoryless feature.

What is the significance of this feature? The feature guarantees the time spent on mining in the past does not have any to do with the mining time needed in the future. Hence this feature is also called progress free (the past progress will not affect the future progress).

Now imagine that if the mining puzzle of a certain cryptocurrency is not designed as memoryless. Then, the more time we had mined, the greater the chance to mine a block in the future. This brings an issue that miners with stronger computing power will gain a disproportionate advantage because during the same period of time, miners with stronger computing power can always do more work than those miners who do not have much computing power. For example, in an ideal state, miner A's computing power is 10 times more than miner B's computing power, so the probability of A finding a mine should also be 10 times more than B to be fair. If the mining puzzle is not memoryless, the probability of miner A finding a block will exceed 10 times more than B. Therefore, the characteristics of memoryless/progress free ensure the fairness of mining.

2. Total Supply

Let's talk about the total supply amount of Bitcoin. As we said, coinbase TX is the only way to generate new bitcoins, and block rewards are halved every 4 years. Therefore, the number of bitcoins generated actually constitutes a geometric series. There are 21,000,000 Bitcoins in total.

$$21k \times 50 + 21k \times 25 + 21k \times 12.5 + \cdots$$

$$21k \times 50 \times \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) = 21{,}000{,}000\ BTC$$

Solving Bitcoin mining puzzles has no practical significance. Bitcoin is becoming more and more difficult to mine because the system is designed to halve the block reward every 210,000 blocks. This scarcity is artificially designed. Although solving mining puzzle has no practical significance, the design of this mining process is the cornerstone of maintaining the security of

the Bitcoin system. Bitcoin is secured by mining. Mining is an effective design that relies on computing power to vote. As long as most of the computing power is in the hands of honest nodes, the security of the system can be guaranteed. We know that mining rewards are halved every 4 years, but does it decrease people's motivation for mining? It does not because the price of Bitcoin continues to rise. When the block rewards get close to 0, do people still have the motivation to mine? Yes, people can get transaction fees for packaging transactions in the mined block.

3. System Safety

Let's enter the discussion of Bitcoin system safety. Assuming that most of the computing power is in the hands of honest miners, can we guarantee that the transactions written to the blockchain are all legal? If honest miners have 90% of the total computing power, bad miners have the rest 10% of the computing power. Then the bad miners can mine 10 blocks out of 100 blocks on average. What would happen when the block is mined by these bad miners? Can they create a false TX and steal other's coins? The answer is no. These bad miners cannot legally sign the false TX.

What if M still put this false TX in the block (Fig 25)? Other honest nodes will not approve this block due this illegal TX. Thus, other honest nodes will not mine after this block. Instead, they will be keeping mining the last legal block. Hence, M cannot steal A's coins this way.

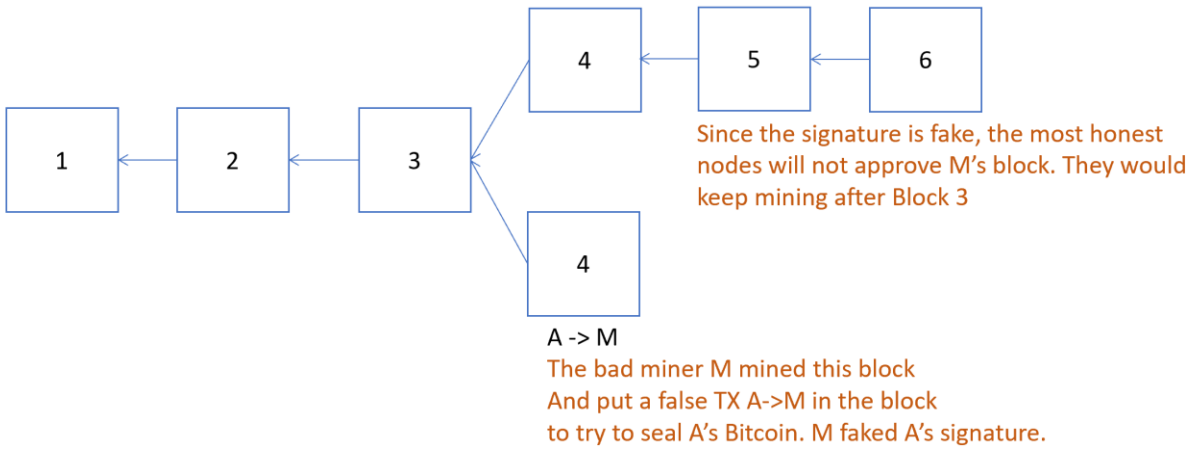


Fig 25

Now, M tries a difference approach. This time M wants to double spend coins (Fig 26). In Block 3, M sent coins to A, then M mined Block 4 and he put M->M' in it to try to double spend. Does it work? Of course not, each TX has a hash pointer pointing to its source. Other nodes can easily tell that M had spent the coins already in Block 3.

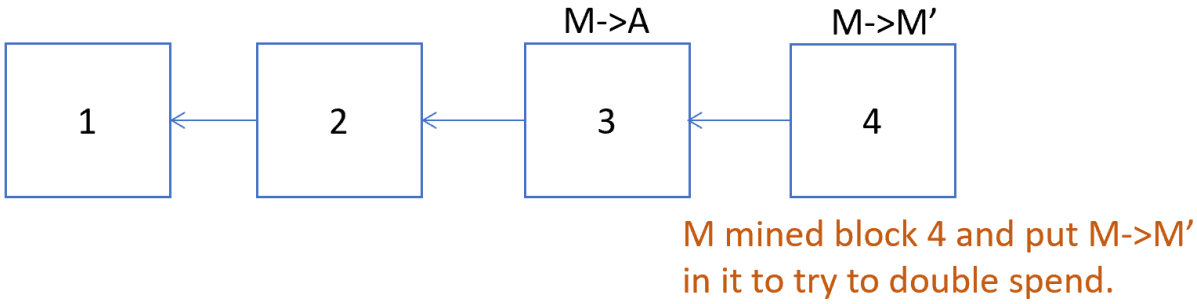


Fig 26

M did not give up. He realized that he should append the block mined after Block 2, not Block 3 (Fig 27).

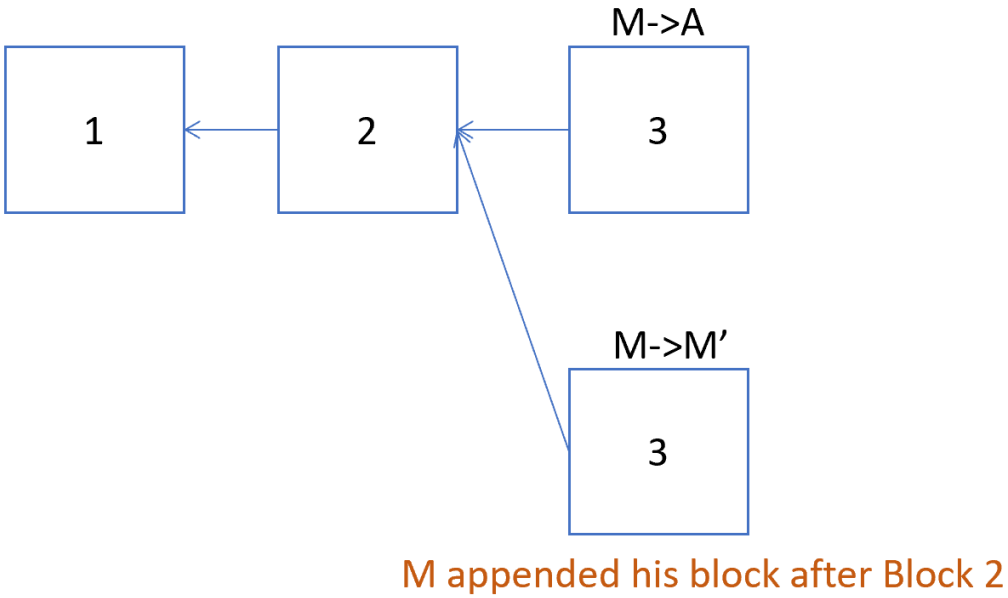


Fig 27

Note that where to append the block is determined at the beginning of mining not after the block is mined. Mining is based on previous block. Therefore, if M wants to append after Block 2, he cannot mine after block 3. In this case, M actually forked the blockchain and if he managed to make his fork the longest valid chain, he will be able to double spend (Fig 28). This is actually the forking attack we talked about before.

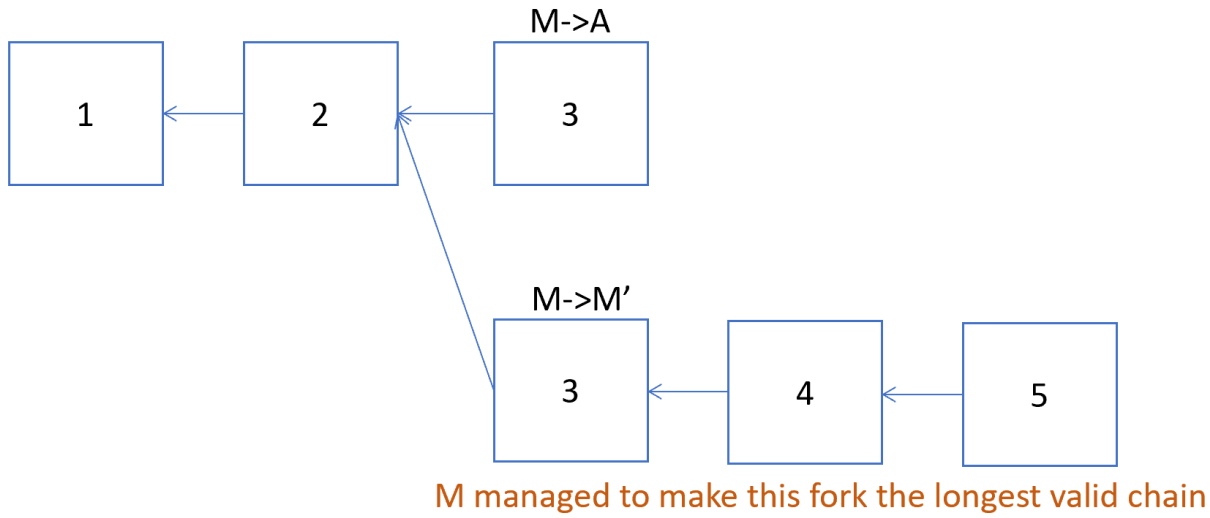


Fig 28

So how can this attack be profitable? For example, M likes to shopping online. He ordered goods from A, and sent bitcoins to A. After receiving the goods. M started forking attack and rolled back the TX that M -> A. If M succeeds in turning his forked chain into the longest valid chain, then M can get the goods without spending bitcoins.

The good news, the forking attack can be prevented. After M->A has been written on the blockchain, A can wait for a few more confirmations to occur (Fig 29). In other words, A will wait for more blocks to be appended before confirming M's order. At this time, it is very difficult for M to develop his forked chain into the longest valid chain because M has to append after Block 2 and at the moment, the longest valid chain's length is much greater than M's forked chain. Hence, M has very little chance to out-length the "longest valid chain" unless M is extremely luck and can mine the next block over and over.

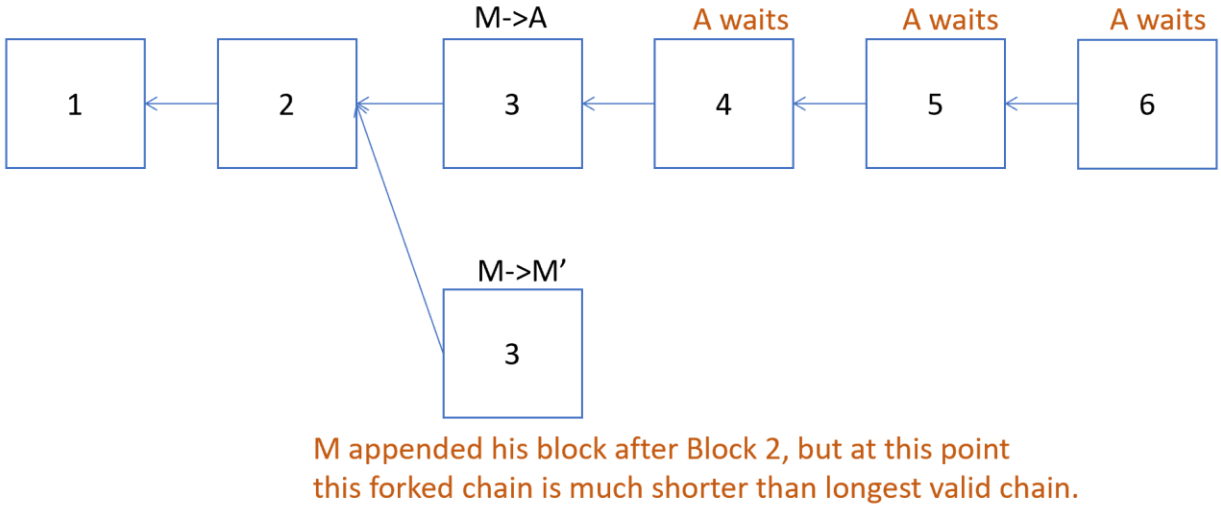


Fig 29

In the Bitcoin protocol, we have to wait for 6 confirmations, which is about 60 minutes (Fig 30). When the M->A transaction was just written to the blockchain, we said that there was one confirmation. After that, every time a block is appended, there would be one more confirmation. After 6 confirmations, we can say that the M->A transaction is “irrevocable”.

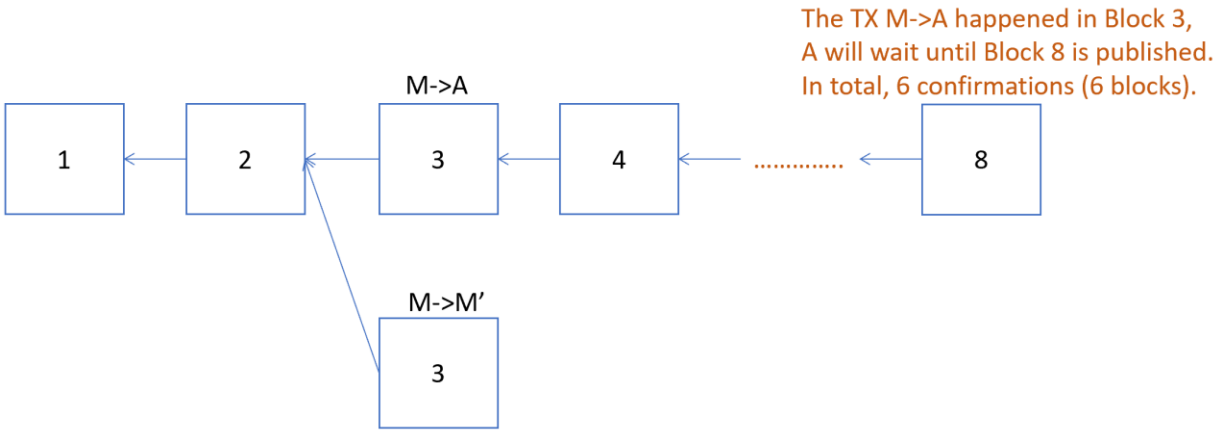


Fig 30

Please note that this so-called irrevocable ledger is only a probability guarantee. If the malicious node is really lucky and continuously obtains the right to append, then theoretically it is indeed possible to roll back the blockchain maliciously. In other words, the content that has just been written to the blockchain is relatively easy to be tampered with, but after several confirmations, the possibility of tampering will decrease exponentially.

In fact, there is another confirmation model called zero confirmation. Take online shopping for example, when we buy a laptop, we pay with Bitcoin, send a transaction to the seller, and then the seller will find a full node to confirm the transaction, and see if the signature and the source of the Bitcoin are legal. If everything is correct, the seller will approve the transaction without waiting for 6 confirmations. It sounds dangerous, isn’t it? Actually, it is not. First, it is set in the Bitcoin protocol that nodes will only recognize the transaction them received first. The transaction of M->A is sent out first, and the transaction of M->M’ is sent out later. Therefore, honest nodes will not approve the transaction of M->M’. Second, because the online shopping platform requires a certain amount of logistics processing time. If the seller finds out that the transaction is not written into the longest valid chain before actual shipping time. The seller can cancel the shipment. Therefore, problems in the Bitcoin system can sometimes be solved from the outside of the system.

Now back to the bad miner M. After the malicious node M gets the right to append, what will happen if some legal transactions are deliberately not packaged into the block? The Bitcoin protocol does not stipulate which transactions must be placed in the block by the node that obtains the right to append. In fact, this is not a big problem, because these legal transactions can be written into the blockchain in future blocks. There are always honest nodes willing to

publish these transactions. Moreover, under normal operation of the blockchain, there may be legal transactions that are not written in the next released block, possibly because there are too many transactions in a short period of time. The Bitcoin protocol stipulates the size of each block cannot exceed 1 MB at most. Therefore, if there are too many transactions, some transactions can only wait for the future blocks to be released.

In fact, there is another attack method called selfish mining. Under normal circumstances, the block that obtains the right to append will immediately publish the block, because if you do not publish it, other nodes might also mined their blocks and just publish, then your block would be useless. However, selfish mining means that after obtaining the right to append, the node will keep the block to itself and continue to mine after the block. Basically, the node tries to grow the length of this chain secretly. When the length is long enough, the node will publish the whole hiding chain and try to beat the main chain.

However, this is also very difficult to succeed, because after hiding the first block, other nodes are not aware that you have gotten a block and they can mine after it. If the computing power of the malicious node is not greater than or equal to 51% of the total computing power, the length of the hidden chain will grow more slowly than the longest legal chain.

What if we are not maliciously using selfish mining, but for normal expansion. what are the benefits? In fact, competition can be reduced to a certain extent (Fig 31). Suppose we find the next block 4, but we do not release it, then others will just keeping mining after block 3. If we have a stronger computing power, because only we know the existence of the block 4, then only we can mine after this block so that we can rule out competition.

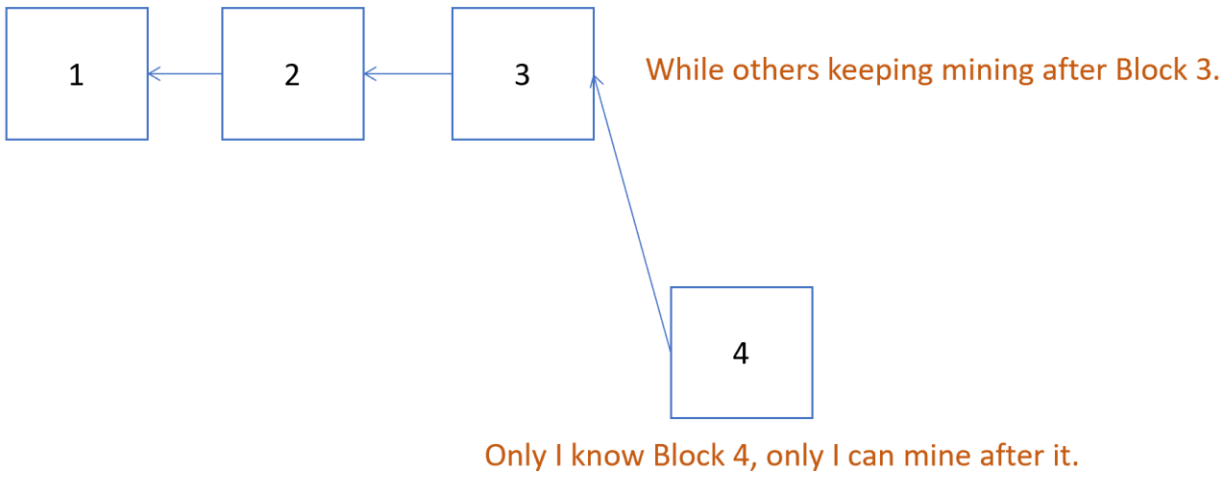


Fig 31

Of course, you need to have strong computing power to achieve this. When others just got their block4, you must already gotten the your block 5 on your chain so that your length can exceed the main chain (Fig 32).

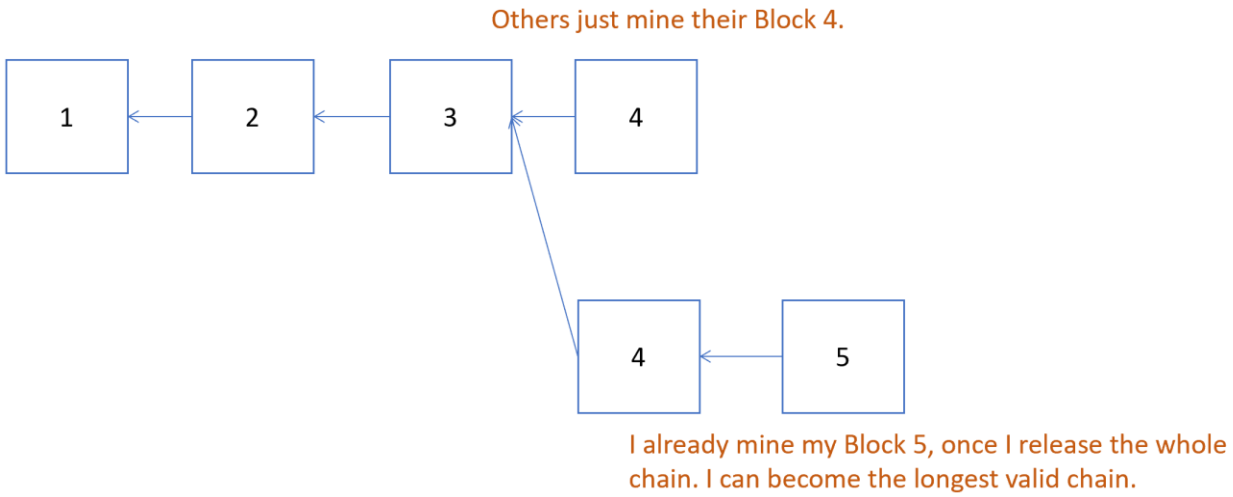


Fig 32

4. Mining Difficulty

As mentioned earlier, mining is to keep trying the nonce value until it finds a hash value within the target domain. Obviously, the smaller the target, the greater the difficulty of mining. Changing the mining difficulty is just to adjust the target space in the entire output space.

$$H(block\ header) \leq target$$

$$difficulty = \frac{difficulty\_1\_target}{target}$$

Bitcoin uses the SHA256 hash algorithm, which can generate a 256-bit hash value, so the entire output space has 2^256 possible values. The difficulty of mining is inversely proportional to the target. The minimum mining difficulty is 1, and the corresponding target is a very large number.

So why do we need to adjust the mining difficulty? That is because as the system's computing power becomes stronger, if the mining difficulty remains the same, the block generation time will become shorter and shorter. It may change from 10 minutes to less than 1 second. Although a shorter block time allows transactions to be written to the blockchain faster, it also brings some concerns.

For example, block time is less than 1 second, but it may take tens of seconds for this block to spread through the Bitcoin network. The nodes who are “far away” might not receive the block soon. Hence, they continue to mine after the last block on chain. Then it is possible for these nodes to find their blocks and release it to the network. When this happens, the blockchain would have lots of forks (Fig 33). If the block time becomes shorter and shorter, number of forks would get larger and larger. Too many forks will make it difficult for the system to reach a consensus, and it is also unsafe.

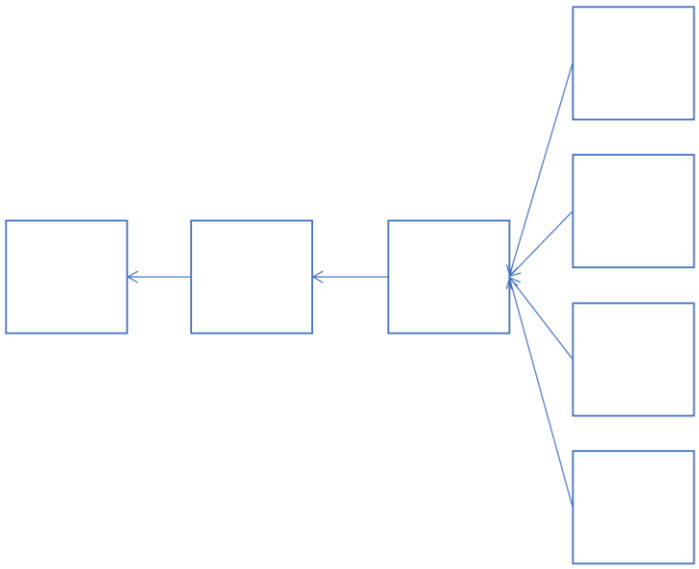


Fig 33

Why would this cause safety concern? That is because the Bitcoin system assumes that most nodes are honest, so the stronger the total computing power, the higher the security and the higher the cost to initiate a 51% attack. What is a 51% attack? It means that malicious nodes account for more than 51% of the total computing power. Suppose that A->B transfers 1000 bitcoins to B, and B waits for 6 confirmations (Fig 34). Now, B thinks that the transfer must be successfully written to the blockchain. At this time, the bad guy A can insert a fork chain before the block that has A->B to roll back the transaction, it is difficult for A to make his fork longer than the longest valid chain when honest nodes are in the majority, but if many forks occur on the longest valid chain, the computing power will eventually be scattered, different nodes may follow different forks to expand. When this happens, the bad guy A can concentrate his computing power to expand his malicious fork chain. A may not need 51% of the computing power to launch an attack, and it may be successful with only 10% of the computing power.

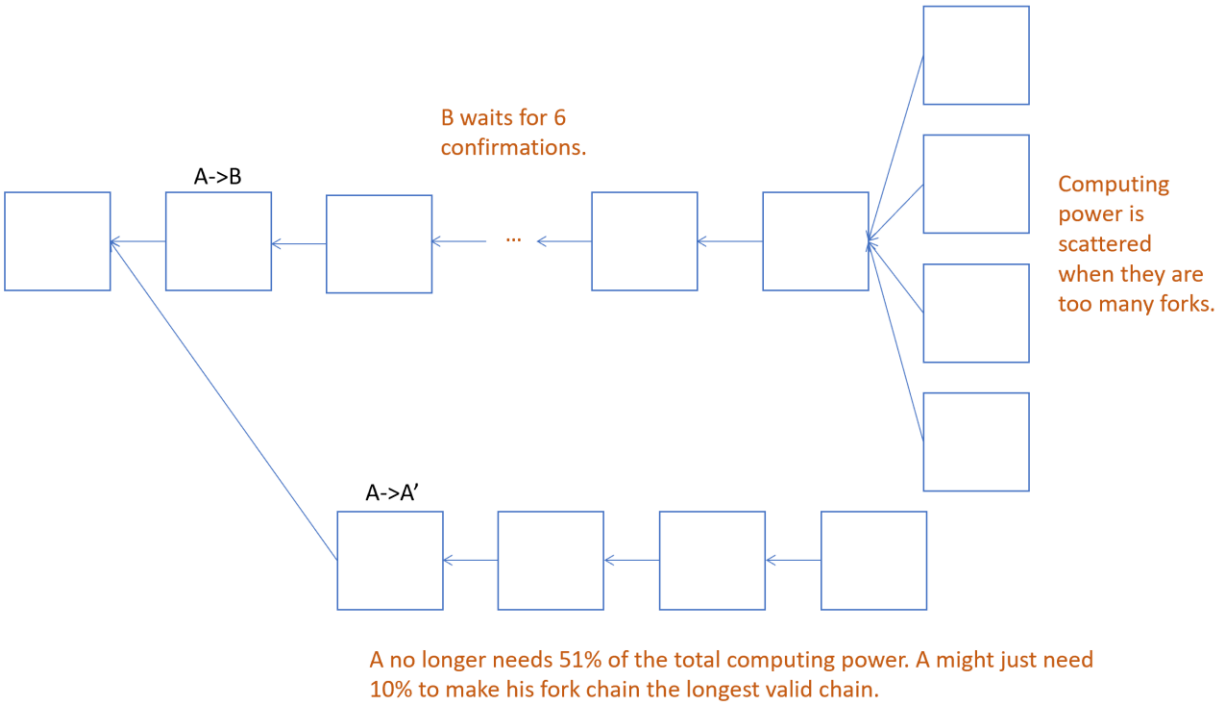


Fig 34

Therefore, the block time is not the shorter the better. The key is to maintain stability (having a stable block time). Does it mean Bitcoin's current 10min block time is optimal? No, not really. If you change it to 8 minutes, it should work too. In fact, many people think that Bitcoin's 10-minute block time is too long to be an efficient trading system, because no one wants to wait at least 10 minutes after paying money. Not to mention people have to wait for another 6 confirmations time. The block time of Ethereum, which will be discussed later, is designed to be 15 seconds, and Ethereum therefore designed a new consensus protocol called Ghost. The orphan blocks generated in ETH network cannot simply be discarded, a certain amount of uncle reward must be given. Ethereum also needs to adjust the mining difficulty to keep the block generation time stays around 15 seconds.

There is no strict rule on how fast the average block time should be, but it must remain stable and cannot be reduced indefinitely. So, how to adjust the mining difficulty exactly? The Bitcoin protocol stipulates that the mining difficulty (that is to adjustment target domain) should be adjusted every 2016 blocks, roughly every 2 weeks.

$$\frac{2016 \times 10\ mins}{60\ mins \times 24\ hrs} = 14\ days$$

$$target = target\ \times \frac{actual\ time\ (time\ spent\ mining\ the\ last\ 2016\ blocks)}{expected\ time\ (2016 \times 10min)}$$

For example, if our actual block time is 3 weeks, then 3 weeks divided by 2 weeks equals 1.5 weeks, and 1.5 times the old target value, the new target value will be larger. As the Target

value increases, the difficulty of mining decreases. Note that there is a limit to the increase and decrease. The limit is 4 times, which means that the target can be increased and decreased at most 4 times than the original value.

Although adjusting the mining difficulty is written in the code, but Bitcoin is open source. What if a malicious node deliberately does not adjust the difficulty? Don't worry too much, because honest nodes will not accept blocks mined by a malicious node at different levels of difficulty. Recall that there is a field in the block header called nBits. The size of this field is 4 bytes. This nBits field is a compressed version of "target", which means that the "target" field is not directly stored in the block header because it is too large. The target is 256 bits and requires 32 bytes. Therefore, the nBits field will reveal the difficulty used when mining the block. Other nodes can tell that the malicious node has not adjusted the target and disapprove the block.

The Ethereum mining to be discussed later is that after each new block, the system may adjust the difficulty, and the adjustment method is much more complicated than Bitcoin. Before the Bitcoin system appeared, there were actually many cryptocurrency designs, but they all failed. Compared with the previous designs, the Bitcoin system may be more "inconvenient" in design, and really conservative in regard to block time, block size. Current Bitcoin design may not necessarily be the optimal.

Figure 35 is the adjustment curve of Bitcoin's mining difficulty, which rises in a stepwise manner and it corresponds to the adjustment period of every two weeks. It can also be seen from the difficulty chart that more and more people are joining mining; the popularity is increasing. If the difficulty of a certain coin continues to decrease, it generally means that the popularity of the coin is gone, and the coin might dead soon.

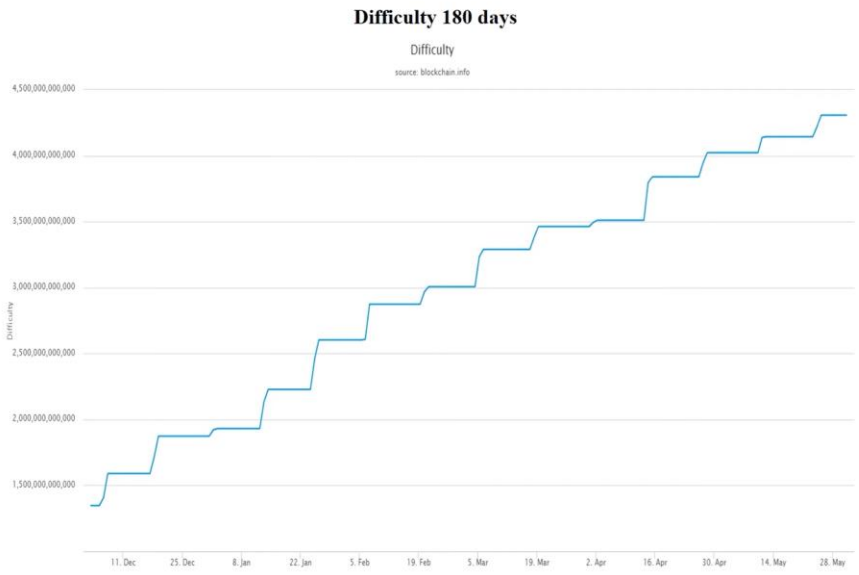


Fig 35

Figure 36 shows the block time, it can be seen that it basically fluctuates up and down around 10min.

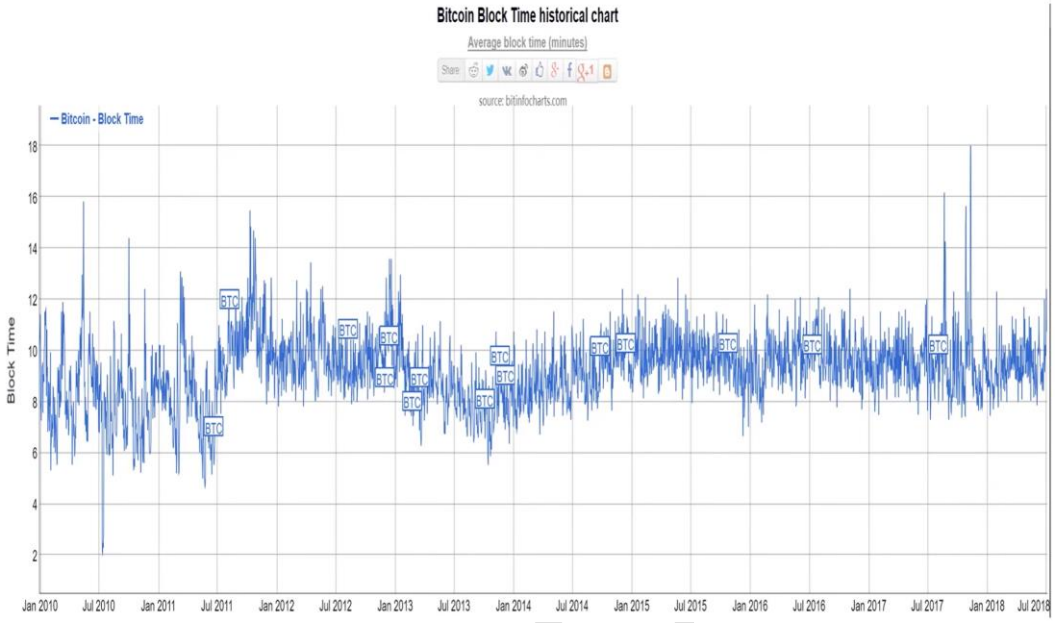


Fig 36

5. Miner & Mining pool

Mining hardware is becoming more and more professional. In the earliest days, ordinary PC CPUs could be used for mining. However, it is very wasteful to do that (buy a computer for mining only). This is because only a small part of the memory is used during mining, most of the CPU components are idle. The operation of calculating the hash value requires only a small amount of CPU Command, the hard disk is also idle. Buying a whole computer for mining is not cost-effective. Therefore, mining equipment began to evolve from CPU to GPU. GPU is mainly used for large-scale parallel computing. For example, deep learning requires a powerful GPU and a large number of matrix multiplications. However, GPU mining is just as wasteful as CPU mining, because there are still many parts of the GPU that are idle during mining, such as those used for floating-point calculations. These floating-point calculations are very useful for deep learning, but Bitcoin mining only uses integers. GPUs are becoming more expensive, partly due to Bitcoin mining, but note that as the difficulty of Bitcoin mining increases, GPU mining is not cost-effective as before, mining process is exceeding the GPU's computing power range and GPU mining is very noisy.

Nowadays, ASIC (application specific integrated circuit) chips are generally used for mining. This chip is born for mining, removed all the functions that are not used for mining. It is very cost effective. Note that if an ASIC was designed for a specific cryptocurrency, then this chip can only mine this specific cryptocurrency, unless other cryptocurrencies use the same mining puzzle. Some newly issued cryptocurrencies will deliberately use some already popular mining

puzzles as their own puzzles, such as using the same mining puzzles same as Bitcoin. In this way, more people can easily mine these new coins. This situation is called merged mining. Except in the case of merged mining, each ASIC can only serve a specific cryptocurrency. The development cycle of ASIC chips is long. For example, Bitcoin ASIC may take at least one year to be developed, although this development speed is already very fast compared to other general-purpose chips.

The sharp drop in the price of Bitcoin will cause miners to lose money. However, when Bitcoin price is skyrocketing, mining competition is greater. You may have to update ASIC chips every few months. Research data shows that most of the profits of the new models of ASIC mining machines were obtained within 2 months after the listing, because the hash rate of the newly listed ASIC mining machines was the strongest at that time. Hence, the timing of purchasing ASIC mining machines is very important. Generally speaking, ASIC mining machines are pre-ordered. If the manufacturer cannot deliver the machines in time, it is very harmful to the miners. Some illegal mining machine manufacturers will mine for 2 months after producing a new generation of mining machines, or take the initiative to help you test the mine and then delivery the machines to miners. This bad behavior can be observed by watching the change of Bitcoin's computing power, if you find that the computing power suddenly increases, it means that some new generation of ASIC miners are out. In fact, these mining machine manufactures are more profitable than ordinary miners.

Some people feel that this kind of arms race in ASIC chips leads to centralization and goes against the original intention of Bitcoin. Therefore, some cryptocurrencies designed alternative mining puzzles (with ASIC resistance) in order to counter the trend of ASICs. The purpose of this design is to allow ordinary computers to participate in mining.

Another trend in mining is the emergence of mining pools. A single miner, even if he uses the most updated ASIC mining machine there is no guarantee in his returns. It is possible to mine a block once a year. As a result, the mining of single miners is a bit similar to buying a lottery ticket, and their income is unstable. And as a full node, these miners are responsible for various responsibilities.

Therefore, mining pools have emerged. The idea is simple, a mining pool organizes the individual miners together. The structure of a mining pool is generally to have a full node to drive many mining machines. Each miner is supervised under the mining pool manager. Miners only responsible for calculating the hash value, other responsibilities of the full node are the responsibility of the manager, such as monitoring transactions, packaging and organizing candidate blocks, monitoring the block release of the Bitcoin network, and so on. The ASIC chip can only be used to calculate the hash value and cannot be used for other functions responsible for the full node.

The emergence of mining pools can help individual miners to get stable income, but how should these miners allocate the block reward? If these miners belong to the same company or

individual, then the company or the person take all the profits, but what if miners are all over the world in a mining pool? In this case it is necessary for the miners and the manager to allocate the task of calculating the hash through the communication protocol of the mining pool. After a miner completes the calculation, the result is returned to the manager. When the manager gets the reward, everyone gets part of it.

How to allocate it exactly? We need a way to reflect the work of each miner and allocate their block reward according to the amount of the work done. The income of individual miners is unstable because mining is too difficult, so if we reduce the difficulty within the mining pool, for example, the hash value required to be calculated beginning with 70 zeros, let's reduce the to 60 zeros This is called a share. This share is also called almost valid block. Now miners will provide this almost valid block to the miner after they get this almost valid block. Note that this almost valid block is only used by the manager to calculate the work done of each miner, it has no other use. In other words, after the mining pool mined a block, the block reward will distribute proportionally according to the number of almost valid blocks submitted by each miner.

Now imagine, is it possible for a miner who got the next block and intentionally do not submit the block, but instead publishing the block himself to win the block reward? The answer is NO, because the miner's task is given by the manager who is responsible for assembling the block, and then handing it over to the miners to try a variety of nonces. We have mentioned that adjusting the nonce field may not be enough, the coinbase field also needs to be adjusted. Hence the manager might give miners different coinbase values to try and there is a recipient address in Coinbase transaction in the candidate block given by the manager to the miner, this address belongs to the manager. The miner has no way to steal the next block because the recipient address is manager's address.

Imagine again, what if the miner ignores the candidate block given by the miner at the beginning, and assembles the block privately, and changes the recipient address to his own? This will not work either. When the miner submits the share to the manager, the manager will not approve it because the domain value of coinbase transaction field has changed.

Can miners intentionally make trouble to manager? For example, a miner got the next block but the miner throws it away. This is possible. Although it hurts everyone. However, there is competition between mining pools, so it is possible that spy miners use this method to attack rival mining pools. These spy miners usually hand in shares only, but throw away blocks. It is like getting paid without working.

Fig 37 shows the mining power metrics in 2018. In 2021, China gave up the throne of the right to append. The United States now ranks first in the world.

Fig 37

In 2014, a mining pool called Ghash had a computing power of more than 51%, which led to the mining pool being able to launch 51% attacks (Fig 38). This mining pool takes the initiative to split itself for the long-term development of Bitcoin.

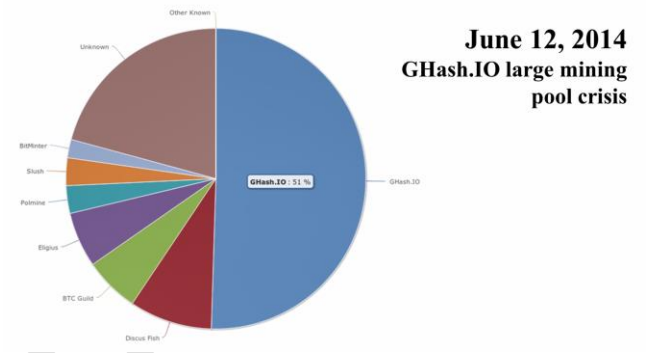


Fig 38

Figure 39 shows the distribution of mining pools in 2018. You can see that each mining pool accounts for a small proportion, but it may be superficial. Assuming that an organization has more than half of the computing power, this organization do not have to put all the computing power in one mining pool, but in several mining pools, so that it can conceal the fact that it has 51% of the computing power. When needed, the organization can combine its distributed computing powers at different pools and launch an attack. It's easy for miners to join a mining pool by complying the mining pool protocol. The pool manager sends the assembled blocks to the miners, and the miners try various nonce values. It is also easy for miners to switch mining pools.

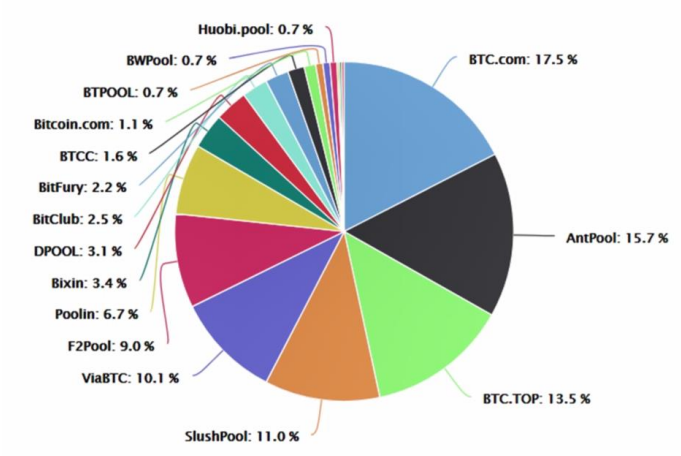


Fig 39

Without mining pools, if an entity wants to launch a 51% attack, it needs to pay a huge hardware cost to buy a large number of mining machines in order to exceed half of the computing power. However, with the emergence of mining pools, this entity does not need to pay for so much hardware cost. It only needs to attract all kinds of miners to join their own mining pools to attack the blockchain. The owner of the mining pool generally draws a management fee from the miner, and the malicious pool may not charge the management fee, or even pay the miners. Therefore, the emergence of large mining pools has caused a certain safety concern to the blockchain.

So, if a mining pool attracts more than 51% of the computing power, what kinds of attacks can be launched? A fork attack can be launched. Forking attack, for example, A transfers 10,000 bitcoins to B, and B waits for 6 confirmations, and B feels that it is definitely safe. At this time, A launches a fork attack and transfers Bitcoin to himself (Fig 40). Since A controls more than 51% of the computer power, the growth rate of A's chain will exceed the original chain and A can make his chain the longest valid chain to roll back the A->B transaction.

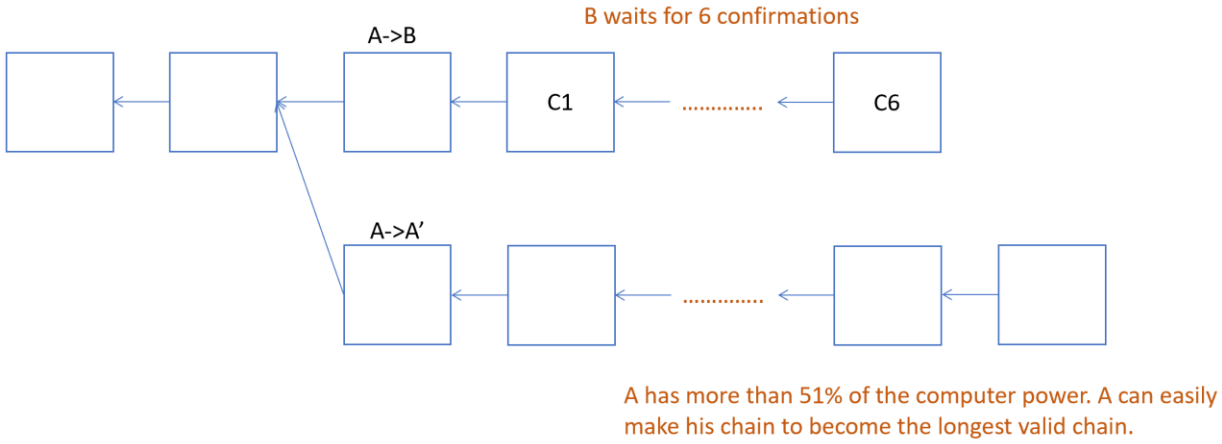


Fig 40

Note that 51% is not a threshold. It does not mean that an attack can only be launched after reaching 51% of computing power. It just means after reaching 51%, the chance of a successful attack is much greater.

In addition, this mining pool can do another bad thing called boycott. For example, this mining pool wants to maliciously boycott A. Now someone has released a block contains the A->B transaction to the blockchain, and this mining pool can prepare a block that does not contain transaction A->B and start boycott forking attack.

As we said before, when there are a large number of good nodes and a small number of bad nodes, there is a probability that the bad nodes will get the right to append. Bad nodes can deliberately exclude certain transactions but other good nodes can put the transaction in future blocks. However, if the bad node has more than 51% of the computing power, then bad nodes have greater chance of getting the next block and every time a transaction of A appears, bad nodes will fork immediately. In this way, other good miners will dare to include A's TXs, because if they include A's TXs their hard-earned blocks will be fork attacked.

So, can a mining pool with more than 51% computing power steal someone else's Bitcoin? This cannot be done because they do not know other's private key. What if this mining pool forcibly put transactions without legal signatures on the blockchain? It will cause a fork, and honest nodes will recognize the original chain and continue mining. Therefore, it is impossible to steal money.

Bitcoin Network

Let's first talk about some basic knowledge in the network.

The so-called network is a structure in which two or more devices (such as computers, printers, routers, etc.) can communicate with each other through communication media (such as network cables, optical fibers, waves, etc.).

The Internet is the network of networks. Different networks are connected by routers. The round disk in the figure 41 below represents routers.

The Internet is generally divided into two modes, Client/Server mode and Peer to Peer (P2P) mode.

Let's look at the Client-Server mode first. Client is the party requesting the service. For example, if you use your computer to search on the Internet, your computer is the client because it requests services. When you use your browser to search information. This browser is the client because it requests services. So, the client can be your computer or a program in your computer. Server corresponds to the server that provides services to the requests sent by your computer. For example, if you want to open the Google website, the Google server will respond to your request. This Client/Server model is a typical centralized structure because Google can unilaterally decide to shut down the server to prevent users from getting the service.

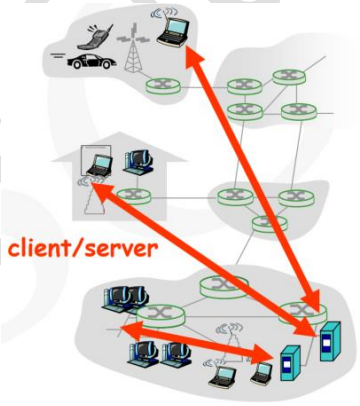


Fig 41

In the P2P model, all computers are equal, which means that each computer is both a client and a server (Fig 42). Everyone provides services to each other. BitTorrent downloading is an example of P2P network. This structure is a decentralized structure because shutting down one computer won't cause a problem there are other computers to provide the service.

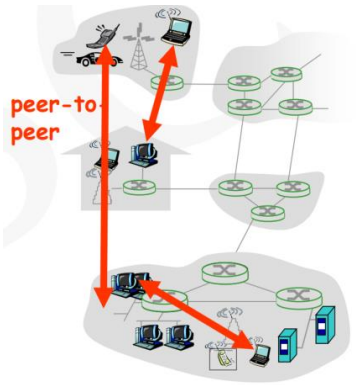


Fig 42

The Bitcoin system uses the P2P model.

1. **P2P Network**

Bitcoin program works in the application layer, and its bottom layer is the p2p overlay network (Fig 43). Users send transactions to the Bitcoin network. After receiving the transactions, nodes pack the transactions into blocks, and then publish the blocks to the Bitcoin network. So how do these newly released transactions and blocks spread on the Bitcoin network?

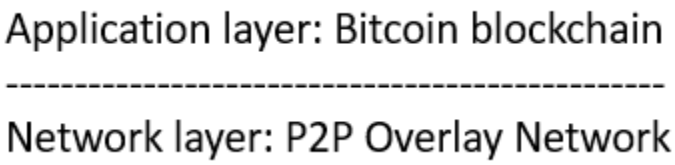


Fig 43

Network Layering. You may have heard many times that the network is layered. This is not to say that the network is layered physically. The communication between computers is not as simple as people talking to the each other. Communication between computers need to comply with a certain protocol. In order to divide the functions of different protocols and keep scalability. We divide the network into 7 layers (application layer, presentation layer, session layer, transport layer, network layer, data link layer, physical layer) and each layer has corresponding protocols (Fig 44). Therefore, we say that Bitcoin is an application that runs on the application layer, and its underlying Network layer uses the P2P mode.

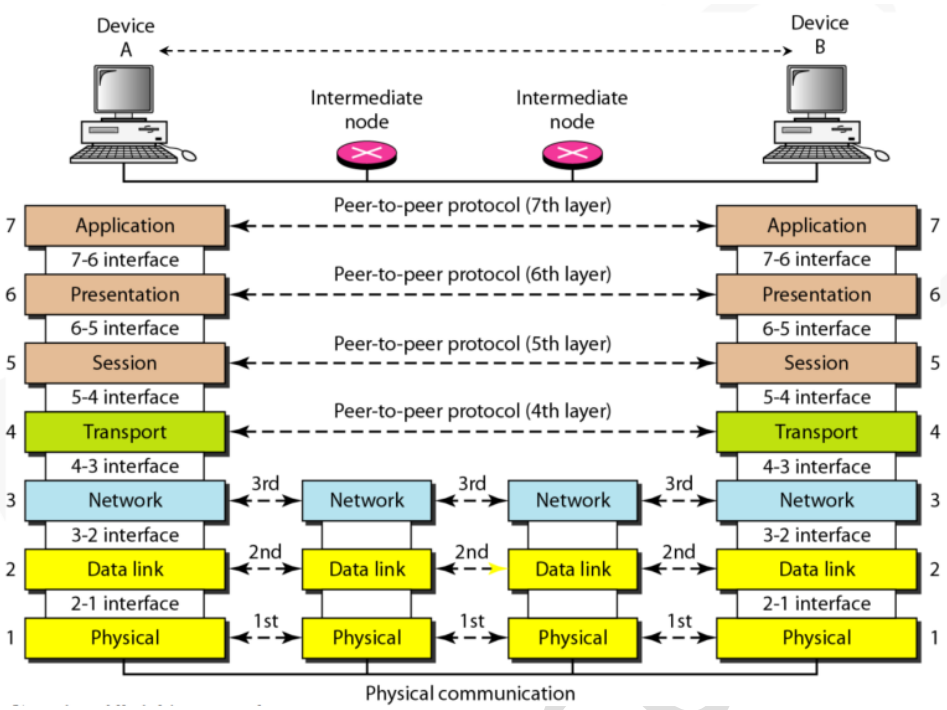


Fig 44

What exactly is a protocol? It's very simple. When humans are writing letters to each other we have to go through the following set of procedures to deliver the letter (Fig 45). This set of procedures is the protocol.

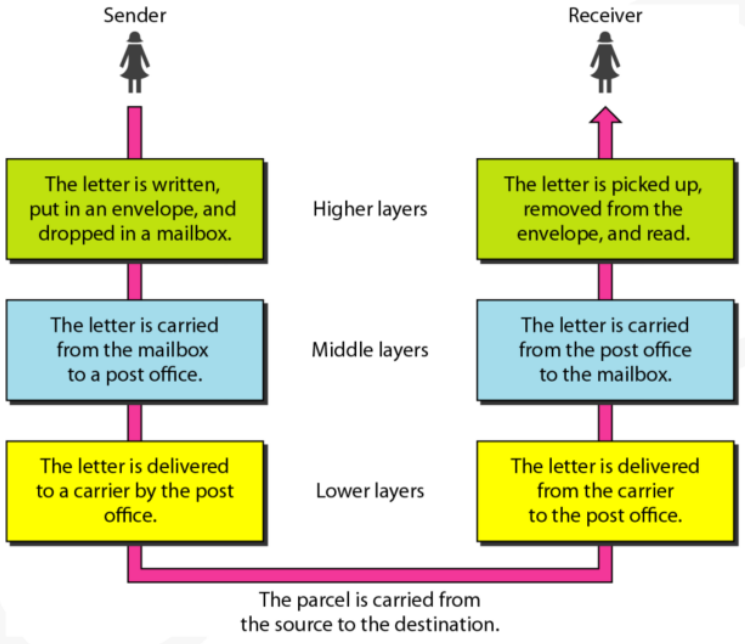


Fig 45

Here is another example that people asking each other about what time it is. The steps followed by both parties is the protocol (Fig 46).

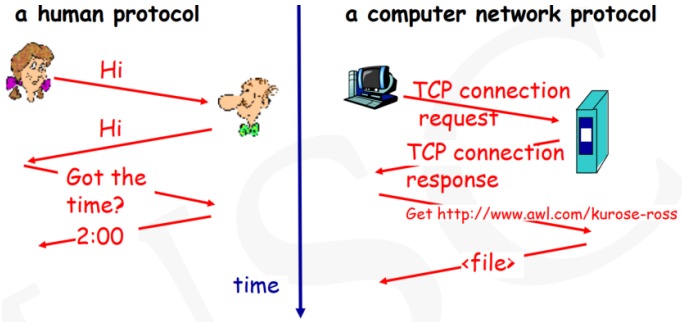


Fig 46

Bitcoin's P2P network is very simple, all nodes are equal. Unlike some other P2P networks, there are super nodes/master nodes. If you want to join the Bitcoin network, you just need to know a seed node. This seed node can tell you the other nodes in the network that it knows. The TCP communication between the nodes is beneficial to penetrate the firewall. You do not need any special operation when leaving the bitcoin network. You can just simply exit the application, and other nodes will delete your information if they do not receive your message within a certain period of time.

TCP/IP, I believe you have seen the term TCP/IP countless times. TCP/IP are protocols that we mentioned above. We need TCP/IP because We have connected various small "networks" into a whole Internet and these small networks may have different technical specifications. If we do not specify a common protocol, various small networks cannot easily communicate to each other. You can think of the small network as different countries in the world and they all speak different languages. Clearly, we need a common language between them for communication. The common language used is TCP/IP. All small networks in the Internet agree to comply by the same communication rules TCP/IP.

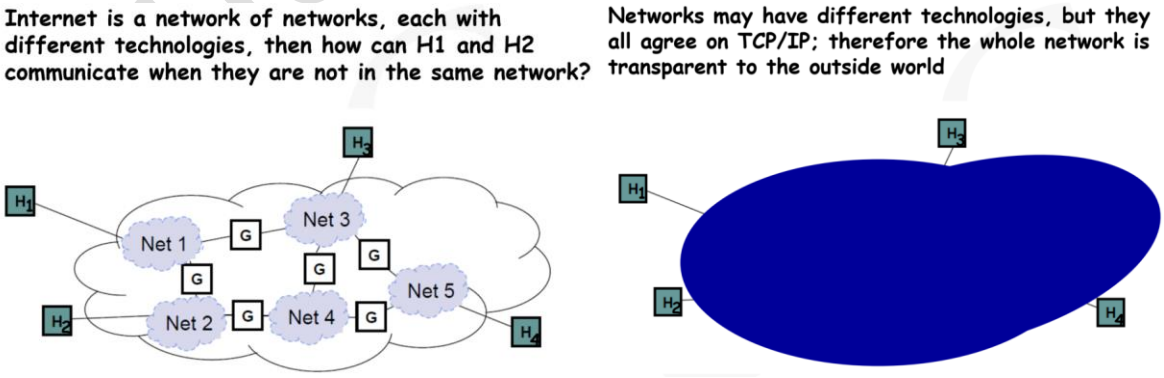


Fig 47

Under the TCP/IP, we actaully changed the 7-layer network layer mentioned above into 5 layers (application layer, transport layer, network layer, data link layer, physical layer). The reason that this 5-layer model is called TCP/IP is because the TCP protocol is mainly used in the transport layer, and the IP protocol is mainly used in the network layer (the TCP is actually the name of the protocol, and IP is also the name of the protocol). However, this is not to say that only TCP and IP are available for the transport layer and network layer, other protocols are also available.

The design principle of the Bitcoin network is to be simple and robust, but not efficient. Each node maintains a set of neighboring nodes. Message dissemination uses flooding. When a node hears a message for the first time, it floods the message to all neighboring nodes. At the same time, the node records that it has heard this message. If the node receives this message again, it will not flood to neighboring nodes. The selection of neighboring nodes of a node is random, the underlying topology does not matter. If you are a node in Beijing, your neighbor node may be in New York. So, the speed of transferring bitcoins to someone downstairs to you is about the same as to someone in Australia. This can enhance robustness, but at the expense of efficiency.

Flooding is a term for computer networks. It means that when a node receives a message, the node will forward the message to all the nodes it knows about. The message rushes to all the neighboring nodes like a flood, so called flooding.

Topology, a term for computer networks, refers to the physical layout of a network. Examples are as follows (Fig 48).

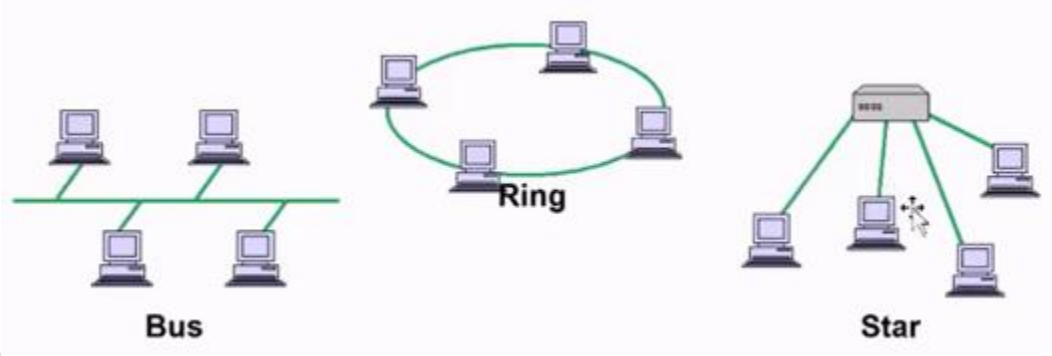


Fig 48

Each node also maintains a collection of transactions pending to be written to the blockchain. When a node hears a transaction information for the first time, it will add the transaction information to its pending TX collection, and forward the message to its neighbor nodes. The premise of forwarding the message is that the transaction must be legal, legally signed and legal source.

Note that we need to consider the race condition, for example, two conflicting transactions (both transactions are trying to spend the Bitcoins having same source) are broadcasted to the network at the same time. Depending on their location in the network, some nodes may receive A->B first, and some may receive A->C first.

Race condition is a scenario that may occur during program execution. It specifically refers to multiple programs that want to operate on the same information, causing the final result of the program to be uncertain. You can think this scenario as when the bus is arrived, people are rushing to get on the bus without queuing up first. Thus, who can get on the bus is not certain.

For example, if a node receives A->B first, it will add A->B to its pending TX collection, and then if the node receives A->C, it will consider A->C as illegal and it will ignore A->C, but if A->C is received first, A->B will be ignored. Note that if the transactions in the pending TX collection have written to the blockchain by other nodes, they must be deleted from the collection. For example, if a node X hears that the newly released block contains A->B, it will delete A->B from its pending TX collection.

So, what if the nodes who have A->B in their pending TX collections find out that the newly released block contains A->C? Then these nodes would have to delete A->B, because since A->C is recognized in the newly released block, it is illegal for these nodes to keep A->B in their pending TX collections. This might happen if another node heard A->C first and mined the next block and published it.

The transmission of new released blocks on the network is similar to the transmission of new transaction information. After each node receives the newly released block, it must check whether the content of the block is legal and whether it is appending the longest valid chain. The larger the block, the slower the propagation speed. The Bitcoin protocol stipulates that the maximum block size is 1 MB. This seems very small, but for the Bitcoin network, it is already quite large, because the Bitcoin network design consumes bandwidth a lot. Bandwidth is the bottleneck. A 1MB block may take tens of seconds to propagate through entire network.

Bit is binary digit. It is the smallest unit of information storage in the computer, bit represents the two logical results 1 or 0, true or false, on or off. It represents two states. Bitcoin's naming should be based on this. In computer circuits, we can distinguish between 1 or 0, true or false, and on or off by controlling the voltage level.

1 Byte = 8 bits

1 KB = 1024 Byte

1 MB = 1024 KB

Note that the propagation of the Bitcoin network is best effort. In other words, when a transaction is posted on the network, not all nodes may receive it. It is also possible that some nodes forward illegal transactions.

2. Full Node & Light Node

Full node	**Light node**
Always online	Not always online
Maintain entire blockchain on local disk	Maintain block headers only
Maintain UTXO in memory for quick validation	Maintain related TXs only
Listen TXs on the Bitcoin network， valid each TX	Can only valid related TXs
Decide which TXs will be packed into the block	Cannot valid new block, but can valid block's mining difficulty
Listen new released block, valid the block	Can only tell the longest chain, cannot tell the longest valid chain
Mine next block	

A full node's verification process is from three aspects. It verifies the legitimacy of each transaction, including coinbase transactions; It verifies that if the block header meets the difficulty requirements after taking the hash value (make sure there are enough zeros in the front), and check whether the difficulty target domain value is correct, and adjust the mining difficulty according to the protocol requirements every two weeks; It verifies that if the longest valid chain is being extended.

Light nodes can verify the difficulty of mining because light nodes maintain block header information. The light node will assume that the full node that issued the block will not publish illegal transactions, because there is usually no benefit in publishing illegal transactions. The light node knows which is the longest chain, but does not know which is the longest valid chain. Most of the nodes in the Bitcoin network are light nodes. If you only need to transfer bitcoin and do not need to mine, then running a light node is enough.

When a node detects that the "new block" has been mined by someone else while mining, what should the node do at this time? The node should stop mining and reassemble the candidate block locally because if you want to mine after the "new block", the transaction content must change. Some transactions in the candidate block you have assembled may have been included in the "new block" already, and the block header will also change, because it contains the root hash value of the Merkle tree which is composed of transactions and a pointer to the previous block, and then restart mining after the "new block".

It seems the node did a lot of work but got nothing, but recall that Bitcoin mining is said to be memoryless/progress free. Therefore, whether you continue to mine the original block or mine after someone else's new block, the probability of you mining out the next block is the same.

How long you have mined in the past will not affect the probability of mining out a block in the future.

When a node mined out a block and sends out a new block assembled by itself, it does not necessarily guarantee that the block sent by the node will win. Sometimes a race condition occurs, or you might have included illegal transaction and causing others to not recognize your block.

Bitcoin's system security is secured from two aspects: cryptography + consensus mechanism. Cryptography means that others do not know your private key, therefore cannot fake your signature, and maliciously steal your Bitcoin (note that the premise of this establishment is that most of the nodes in the system are good and will not accept illegal Signed transaction).

Bitcoin Script

1. Script Language

Bitcoin's script language is very simple, it can only access a stack memory space. It is unlike Java and C which have comprehensive functions. Therefore, Bitcoin's scripting language is called stack-based language.

Programming language refers to languages that programmers use when writing computer programs that can be understood and executed by the machine. These languages have their own grammatical logics such as C, C++, Java, Python, etc. These languages are just like human languages in real life, such as English. English follows certain grammatical rules. You have to speak English, so English speakers can understand it. Programming languages also have certain grammatical rules, they are the languages between humans and machines. The so-called scripting language is just a very simplified programming language.

The script is just a piece of code. It is Turing incomplete and has no loop syntax structure. It can only be executed in sequence, but branches can occur. It is based on the stack.

Stack is a basic data structure in computer science (Fig 49). Its characteristic is that things put in first can only be taken out at the last. You can imagine that you have a deep box. You put plate number 1 in the box, plate number 2 in the box, plate number 3 in the box, and so on. Now if you want to get the plate number 1, you must get the plate number 5 first, then the plate number 4, and so on.

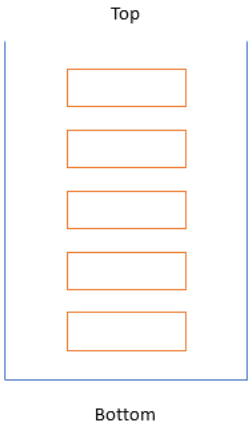


Fig49

Below are script commands (Fig 50 & 51).

Flow control

Word	Opcode	Hex	Input	Output	Description
OP_NOP	97	0x61	Nothing	Nothing	Does nothing.
OP_IF	99	0x63	<expression> if [statements] [else [statements]]* endif		If the top stack value is not False, the statements are executed. The top stack value is removed.
OP_NOTIF	100	0x64	<expression> notif [statements] [else [statements]]* endif		If the top stack value is False, the statements are executed. The top stack value is removed.
OP_ELSE	103	0x67	<expression> if [statements] [else [statements]]* endif		If the preceding OP_IF or OP_NOTIF or OP_ELSE was not executed then these statements are and if the preceding OP_IF or OP_NOTIF or OP_ELSE was executed then these statements are not.
OP_ENDIF	104	0x68	<expression> if [statements] [else [statements]]* endif		Ends an if/else block. All blocks must end, or the transaction is **invalid**. An OP_ENDIF without OP_IF earlier is also **invalid**.
OP_VERIFY	105	0x69	True / false	Nothing / *fail*	**Marks transaction as invalid** if top stack value is not true. The top stack value is removed.
OP_RETURN	106	0x6a	Nothing	*fail*	**Marks transaction as invalid**. Since bitcoin 0.9, a standard way of attaching extra data to transactions is to add a zero-value output with a scriptPubKey consisting of OP_RETURN followed by data. Such outputs are provably unspendable and specially discarded from storage in the UTXO set, reducing their cost to the network. Since 0.12, standard relay rules allow a single output with OP_RETURN, that contains any sequence of push statements (or OP_RESERVED[1]) after the OP_RETURN provided the total scriptPubKey length is at most 83 bytes.

Fig 50

Stack

Word	Opcode	Hex	Input	Output	Description
OP_TOALTSTACK	107	0x6b	x1	(alt)x1	Puts the input onto the top of the alt stack. Removes it from the main stack.
OP_FROMALTSTACK	108	0x6c	(alt)x1	x1	Puts the input onto the top of the main stack. Removes it from the alt stack.
OP_IFDUP	115	0x73	x	x / x x	If the top stack value is not 0, duplicate it.
OP_DEPTH	116	0x74	Nothing	<Stack size>	Puts the number of stack items onto the stack.
OP_DROP	117	0x75	x	Nothing	Removes the top stack item.
OP_DUP	118	0x76	x	x x	Duplicates the top stack item.
OP_NIP	119	0x77	x1 x2	x2	Removes the second-to-top stack item.
OP_OVER	120	0x78	x1 x2	x1 x2 x1	Copies the second-to-top stack item to the top.
OP_PICK	121	0x79	xn ... x2 x1 x0 <n>	xn ... x2 x1 x0 xn	The item *n* back in the stack is copied to the top.
OP_ROLL	122	0x7a	xn ... x2 x1 x0 <n>	... x2 x1 x0 xn	The item *n* back in the stack is moved to the top.
OP_ROT	123	0x7b	x1 x2 x3	x2 x3 x1	The 3rd item down the stack is moved to the top.
OP_SWAP	124	0x7c	x1 x2	x2 x1	The top two items on the stack are swapped.
OP_TUCK	125	0x7d	x1 x2	x2 x1 x2	The item at the top of the stack is copied and inserted before the second-to-top item.
OP_2DROP	109	0x6d	x1 x2	Nothing	Removes the top two stack items.
OP_2DUP	110	0x6e	x1 x2	x1 x2 x1 x2	Duplicates the top two stack items.
OP_3DUP	111	0x6f	x1 x2 x3	x1 x2 x3 x1 x2 x3	Duplicates the top three stack items.
OP_2OVER	112	0x70	x1 x2 x3 x4	x1 x2 x3 x4 x1 x2	Copies the pair of items two spaces back in the stack to the front.
OP_2ROT	113	0x71	x1 x2 x3 x4 x5 x6	x3 x4 x5 x6 x1 x2	The fifth and sixth items back are moved to the top of the stack.
OP_2SWAP	114	0x72	x1 x2 x3 x4	x3 x4 x1 x2	Swaps the top two pairs of items.

Fig 51

Let's look at a simple script example (Fig 52). We have a stack. We first push 10 into the stack, then push 3 into the stack, and then execute the SUB (subtraction instruction, this instruction will remove the top 2 elements of the stack, perform the subtraction, and then push the result back onto the stack), and then push another 7 into the stack. Then execute EQUAL (this will remove the top two numbers and compare them, our case is 7=7, so it is equal, the return value is 1, 1 will be pushed into the stack), and finally execute VERIFY (this will look at the stack top value, if it is 1 then it passes).

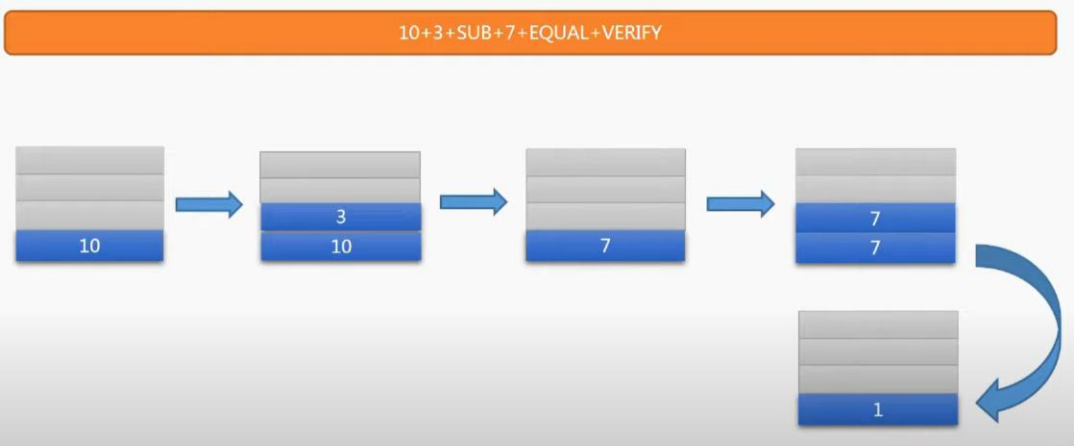


Fig 52

Let's look at a real Bitcoin script case (Fig 53). Just like the simple example above to verify whether 7 is equal to 7, we need to use this script to verify the legitimacyof the public key and signature.

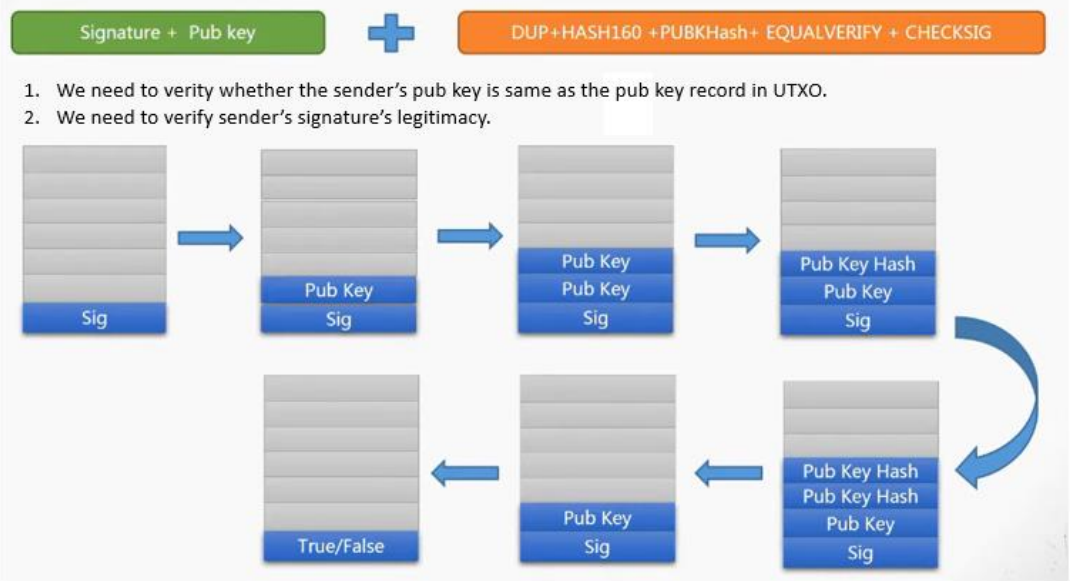


Fig 53

Now, let's take a look at a transaction example (Fig 54).

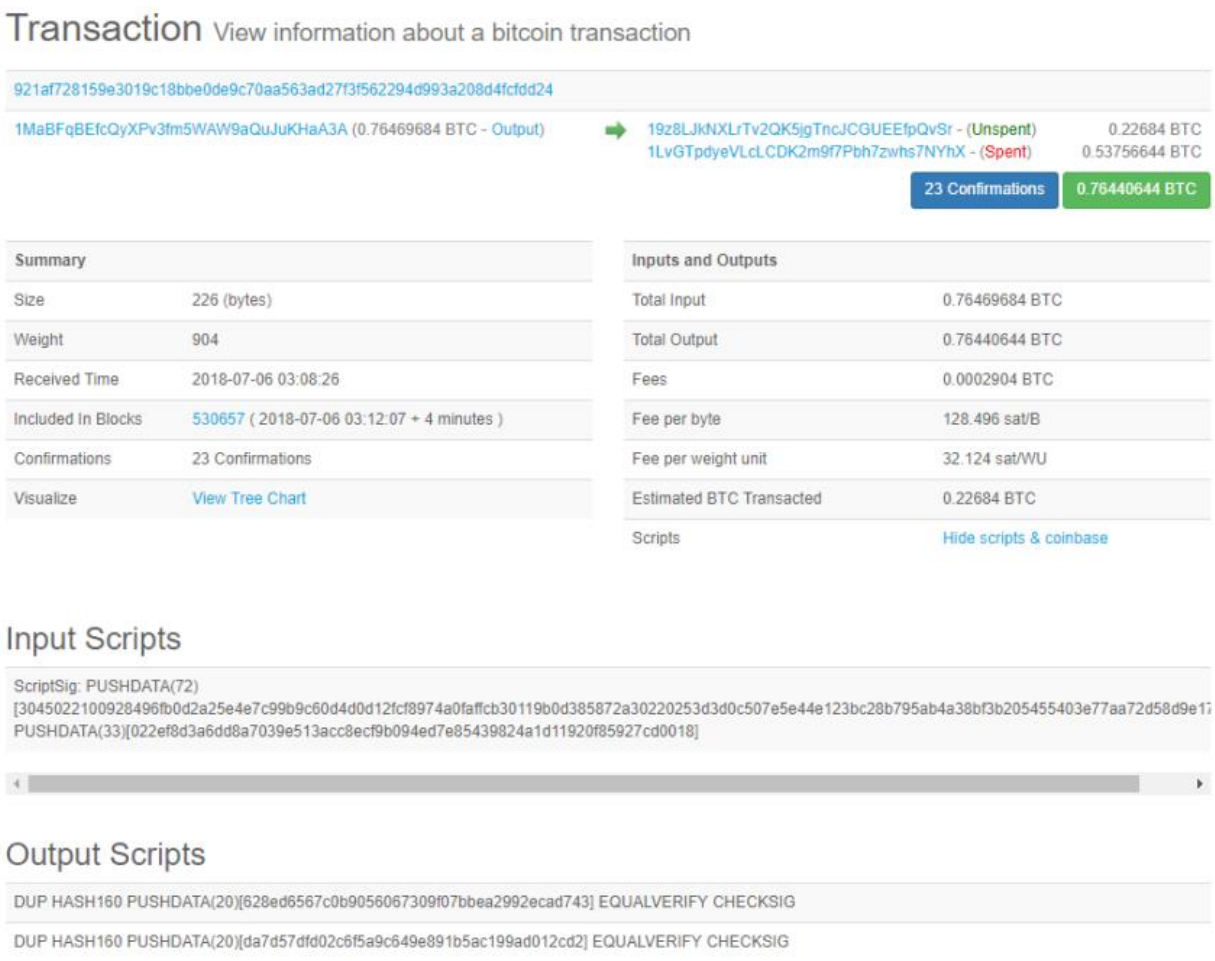


Fig 54

The transaction has 1 input and 2 outputs. Although output is written on the left, it is actually the input of this transaction. The reason why output is written here is because of that the Bitcoin used in this transaction is the output of a previous transaction. The two outputs on the right, the upper one is unspent and the lower one is spent. The transaction has 23 confirmations, so the possibility of being rolled back is very small. The input script consists of two operations, each of which pushes two very long numbers onto the stack. The output here has 2 lines, respectively, to the above 2 outputs, because each output has its own separate section of script.

Let's take a look at the structure of this transaction (Fig 55).


```
"result": {
    "txid": "921a…dd24",
    "hash": "921a…dd24",
    "version": 1,
    "size": 226,
    "locktime": 0,
    "vin": […],
    "vout": […],
    "blockhash": "000000000000000002c510d…5c0b",
    "confirmations": 23,
    "time": 1530846727,
    "blocktime": 1530846727
}
```


Fig 55

“txid” is transaction id

“hash” is the hash value of this transaction

“version” is the Bitcoin protocol version number

“size” is the size of the transaction

“locktime” refers to the effective time of the transaction, 0 means effective immediately.

“Vin and Vout” are the input and output

“blockhash” is the hash value of block where this TX exists.

“confirmation” is how many confirmations have been made for this transaction time is the generation time of this transaction, in seconds

“blocktime” is the generation time of the block where this TX exists, in seconds

The following is the input of the transaction (Fig 56)


```
"vin": [{
    "txid": "c0cb…c57b",
    "vout": 0,
    "scriptSig": {
        "asm": "3045...0018",
        "hex": "4830...0018"
    },
}],
```


Fig 56

A transaction can have multiple inputs, but our case has only one input. For each input, it is necessary to indicate its source and provide a signature. the txid in the figure above represents the hash value of the "previous transaction" Vout means the number of outputs in the "previous transaction"

Scriptsig is the input script, because the simplest form of input script is just a signature to prove that you have the right to spend the Bitcoin. This means that a transaction in Bitcoin may require multiple signatures.

The following is the output of the transaction (Fig 57).


```
"vout": [{
    "value": 0.22684000,
    "n": 0,
    "scriptPubKey": {
        "asm": "DUP HASH160 628e…d743 EQUALVERIFY CHECKSIG",
        "hex": "76a9…88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [ "19z8LJkNXLrTv2QK5jgTncJCGUEEfpQvSr"]
    }
  },{
    "value": 0.53756644,
    "n": 1,
    "scriptPubKey": {
        "asm": "DUP HASH160 da7d…2cd2 EQUALVERIFY CHECKSIG",
        "hex": "76a9…88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": ["1LvGTpdyeVLcLCDK2m9f7Pbh7zwhs7NYhX"]
    }
}],
```


Fig 57

There are 2 outputs.

value is the amount of output, which is how much bitcoin is transferred to the other party

n is the serial number

Scriptpubkey is the output script, because the simplest form of the output script is to give a public key

asm is the content of the output script

reqSigs means how many signatures are needed for this output

type is the output type,

oubkeyhash is the hash of the public key

addresses is the output address.

Let's look at how the input and output scripts are executed (Fig 58). In the following blockchain, A sends Bitcoin to B, and then B sends to C. That is to say, the source of Bitcoin in TX B to C is from the TX A to B. Therefore, the txid and vout in the input of the transaction B to C point to the output of the A to B. Then, to verify the legitimacy of B to C, the input script of B to C and the output script of A to B need to be spliced together.

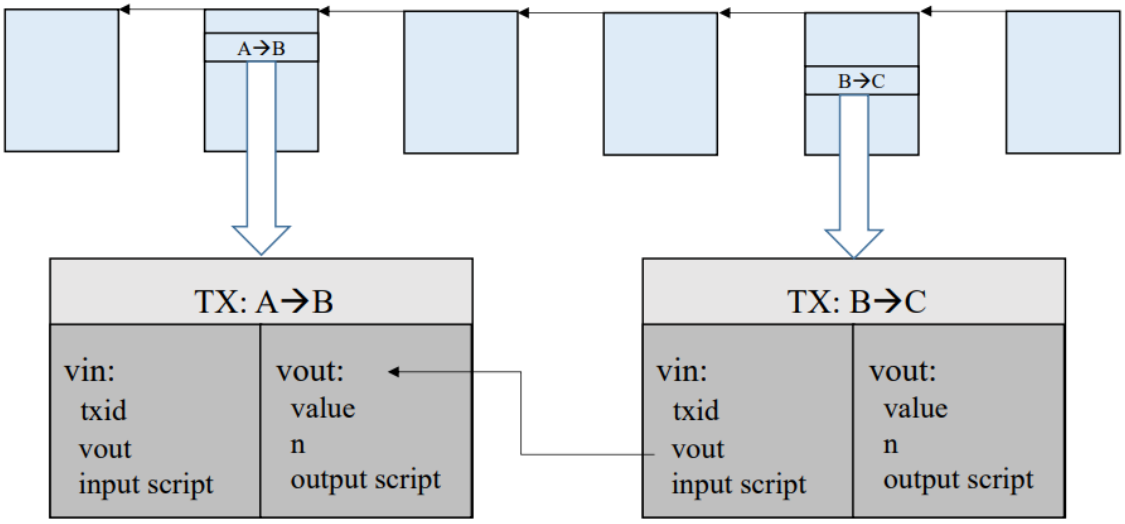


Fig 58

In the early version of Bitcoin, these two scripts will be spliced together and executed from beginning to end (Fig 59). Later, due to some safety considerations, they are executed separately. The input script is executed first, and if there is no error, the output script is executed. If everything is executed smoothly, the final result of the stack top is non-zero, which means true. Then the verification is passed and the transaction is legal. If there is any error, it means illegal. If the transaction has multiple inputs, then each input script must be verified with its own corresponding output script, and it is legal if all the verifications pass.

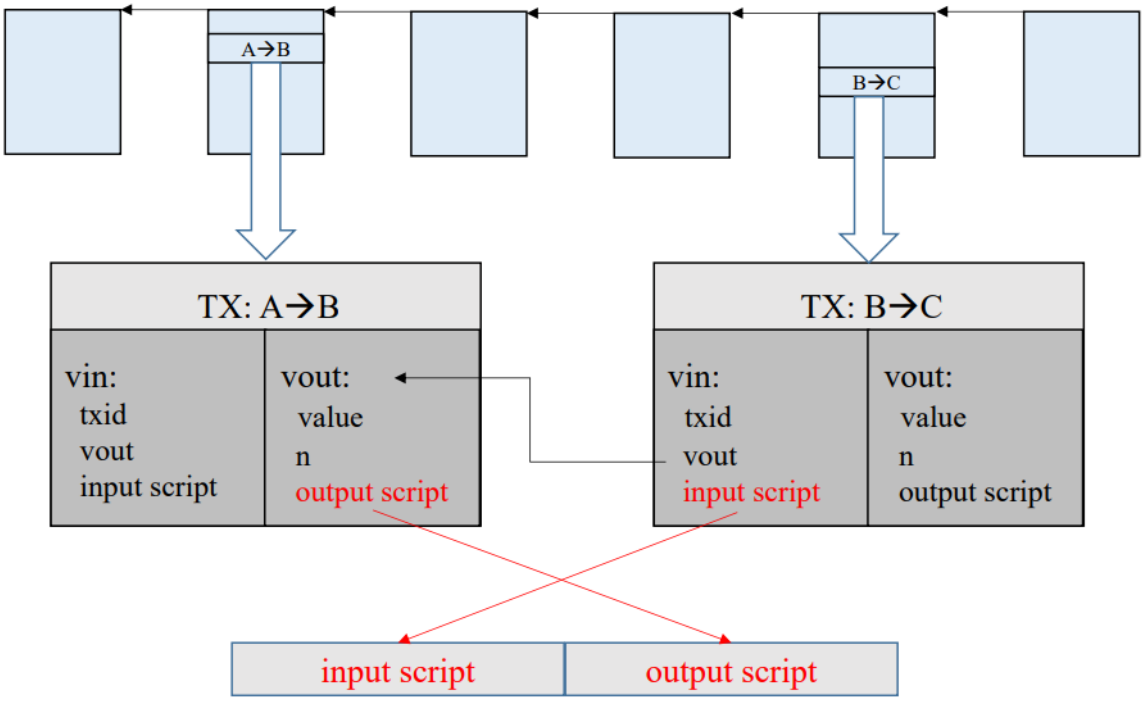


Fig 59

2. Script Type

The simplest form is called pay to public key (Fig 60)

P2PK (Pay to Public Key)

input script:

```
PUSHDATA(Sig)
```

output script:

```
PUSHDATA(PubKey)
CHECKSIG
```

Fig 60

The public key of the receiver will be given directly in the output script. Checksig is the operation of checking the signature. The signature is to use the private key to sign the entire transaction. This form is the simplest, because the public key is directly given in the output script. After concatenating the input and output scripts the combined script looks like below (fig 61). The first line comes from the input script, and the last two lines come from the output script. Note that the actual execution is separate, we put it together here for convenience.

```
PUSHDATA(Sig)
PUSHDATA(PubKey)
CHECKSIG
```

Fig 61

The first line is to push the input signature into the stack, the second line is to push the output public key into the stack, and the third line is to pop up two elements, and use the public key to check whether the signature is correct. If it is correct, the transaction is legal.

Another type is called pay to public key hash (Fig 62).

P2PKH (Pay to Public Key Hash)

input script:

```
PUSHDATA(Sig)
PUSHDATA(PubKey)
```

output script:

```
DUP
HASH160
PUSHDATA(PubKeyHash)
EQUALVERIFY
CHECKSIG
```

Fig 62

The public key of the receiver is not directly given in this output script, only the hash of the public key is given. The public key is given in the input script (signature and public key are given by the input script). DUP hash160 in the output script is to verify the correctness of the signature. This P2PKH is the most commonly used script type.

Again, the combined input and output scripts look like below (Fig 63). The first two lines are from the input script, and the latter are from the output script. The first line pushes the signature onto the stack, the second line pushes the public key onto the stack, and the third line copies the top element of the stack (so there will be an extra public key at the top of the stack). Hash160 pops the top element of the stack, takes the hash and then pushes the obtained hash value onto the stack, so the top of the stack becomes the hash value of the public key.

```
PUSHDATA(Sig)
PUSHDATA(PubKey)
DUP
HASH160
PUSHDATA(PubKeyHash)
EQUALVERIFY
CHECKSIG
```

Fig 63

Pushdata pushes the hash value of the public key provided in the output script into the stack. At this time, there are 2 hash value elements in the stack. Equalverify pops up 2 elements and compares whether they are equal. The purpose of this step is to prevent someone from using their own public key to impersonate the receiver's public key.

Checksig pops up two elements and uses the public key to check whether the signature is correct. If it is correct, the transaction is legal. Any errors in the above process will label the transaction as illegal.

The most complex script type is called Pay to Script Hash (Fig 64).

P2SH (Pay to Script Hash)

BIP 16

input script:

```
...
PUSHDATA(Sig)
...
PUSHDATA(serialized redeemScript)
```

output script:

```
HASH160
PUSHDATA(redeemScriptHash)
EQUAL
```

Fig 64

This output script does not provide the hash of the receiver's public key, but the hash of a script provided by the receiver. This script is called redeem script. When the Bitcoin is spent in the future , The input script should show the specific content of the redeem script, and also need to

show the signature that can make the redeem script run correctly. The verification is divided into two steps. First, it is necessary to verify whether the redemption script given in the input script matches the hash value given in the output script. If it does not match, the redemption script provided is incorrect. If it matches, you need to execute the content in the redemption script again to see if it can be executed correctly. If both steps are passed, the transaction is legal.

Now let's take a look how to use P2SH to implement P2PK (Fig 65)

```
redeemScript:
    PUSHDATA(PubKey)
    CHECKSIG
input script:
    PUSHDATA(Sig)
    PUSHDATA(serialized redeemScript)
output script:
    HASH160
    PUSHDATA(redeemScriptHash)
    EQUAL
```

Fig 65

The input script is to provide the signature and the serialized redemption script. The content of the redemption script itself is to provide the public key and then use checksig to check the signature. The output script is used to verify whether the redemption script given in the input script is correct.

Figure 66 shows the verification step one.

```
PUSHDATA(Sig)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL
```

Fig 66

Let's splice the input and output scripts together first. The first two lines are from the input script, and the latter are from the output script. The first line puts the signature of the input

script on the stack, the second line puts the redemption script on the stack, the third line is to get the hash value (RSH) of the redemption script, and the fourth line is to put the hash given in the output script into the stack, and then there will be the two RSHs and finally use EQUAL to compare to see if they are equal. If they are not equal, game over. If they are equal, the two hash values would disappear from the stack. So far, the first step is over. Then we need to start the second step (Figure 67).

```
PUSHDATA(PubKey)
CHECKSIG
```

Fig 67

We must deserialize the serialized redemption script provided in the input script, then execute the redemption script, put PubKey on the stack, and then use checksig to check the correctness of the signature given in the input script, if the verification passes. Then the entire P2SH verification process is completed.

3. Multi-Signature

So why do we use such a complicated script type like P2SH? For simple transactions, it is indeed too complicated, and P2SH was not available in the original Bitcoin version. This feature was later added by soft fork. The main application scenario of this mode is the support for multi-signature. An output in the Bitcoin system may require multiple signatures to withdraw Bitcoin. For example, the Bitcoin account of a large company may require at least 3 signatures from 5 high-level people's signatures to withdraw Bitcoin. This can prevent someone's private key from leaking or losing, and provides some protection. This function is implemented by checkmultisig.

The implementation method is as follows. In the output script, n public keys are given, and the domain value m is specified at the same time. As long as the input script provides any m signatures corresponding to the n public keys, it can be passed. For example, in the above example, m=3, n=5.

Figure 68 shows the input and output scripts of multi-signature.

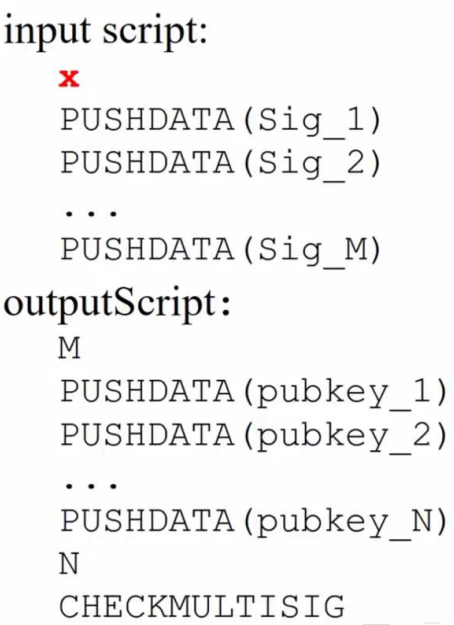


fig 68

There is an × in the first line of the input script because the implementation of checkmultisig in the Bitcoin code has a bug that cannot be modified that one extra element will pop up from the stack, because this is a decentralized system, it is very costly to fix bugs through software upgrades. Hard forks are required if they must be changed. So, the solution is to push a useless element on the stack in the input script. This x represents the useless element. Also note that the relative order of n and m must correspond to each other.

Figure 69 is an execution example of checkmultisig, where m=2 and n=3. The first line's false is the useless element that needs to be push into stack. The second line and the third line push the signatures onto the stack. Then input script ends here. Start to execute output script, and push the domain value m=2 onto the stack. Then push the 3 public keys onto the stack. Then push n=3 onto the stack. Finally execute checkmultisig. See if the stack contains 2 of the 3 signatures. If it is, it is passed.

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

Fig 69

Note that the process above does not use P2SH, but uses Bitcoin's native CheckMultiSig function. However, this native function is actually a bit troublesome. For example, when shopping online, an e-commerce company may say that we use multiple signatures, and we require 3 out of 5 signatures. Another e-commerce company may require for 4 out of 6. Therefore, users must meet the signature requirements of various e-commerce companies when generating transfer information, and also provide the values of m and n. In other words, all these complicated information must be given in the user's output script, which is actually very troublesome because the complexity is exposed to the user.

So, we actually need to use P2SH to implement multi-signature (Fig 70). This basically transfers the complexity from the output script to the input script. The output script is now simple, with only three lines. The complexity has been transferred to redeemScript. The output script only needs to provide the hash value of this redeemScript. And the redeemScript must provide these n public keys, as well as the values of n and m. This redeemScript is provided by the receiver.

Recalling the e-commerce example, the receiver is the e-commerce store. Therefore, e-commerce store only needs to publish the hash value of the redemption script on the website, and then put this hash value into the output script when the user generates the transfer transaction. As for the signature rules like choosing 3 from 5 or 4 from 6, users do not need to know the details. Therefore, it is convenient for users.

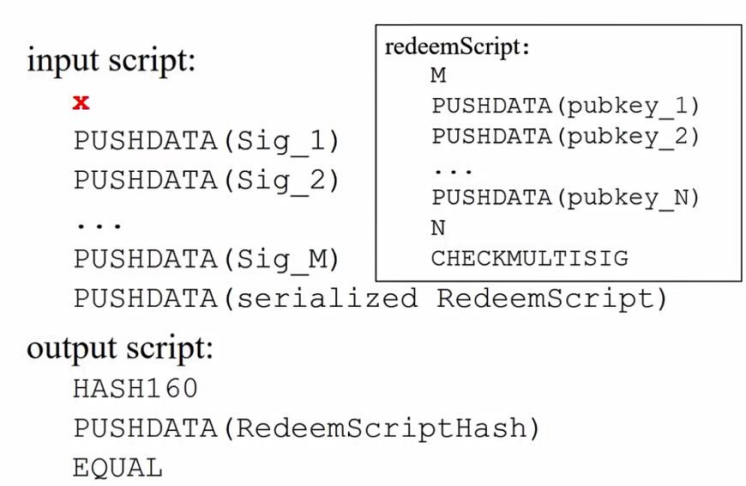


Fig 70

Figure 71 shows the specific script execution process. The input script and output script are spliced together. False is the useless element. Then push the two signatures into the stack. Then put the serialized redemption script into the stack. The execution of the input script is now complete. Start to execute the output script. Take the hash first, and then push the hash value provided in the output script onto the stack. Finally, check whether the two hash values are equal. Then the first phase of verification is complete.

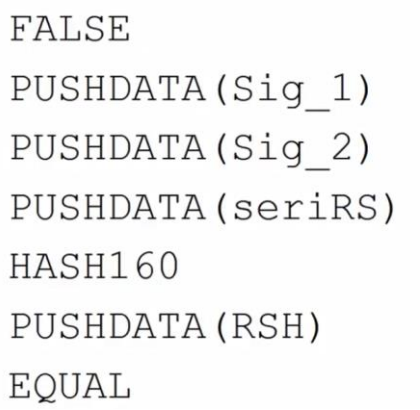


Fig 71

Next, start the second phase of verification, expand the redemption script and execute it (Figure 72). Put m=2 onto the stack. Then push the three public keys onto the stack. Then push n=3 onto the stack. Finally, check the correctness of the multi-signature.

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

Fig 72

4. Proof of Burn

Lastly, we are going to talk about a special script type the return script. This output script starts with return and can be followed by any content. The feature of “return” is to return false unconditionally, so the script containing this operation will never pass the verification. After the execution of the return, it will terminate, and the following content will not have a chance to execute. Why should we use such an output script? In fact, this script is a way to prove the burning of Bitcoin. Burning Bitcoins is mainly used in two situations. For example, you might need to burn some Bitcoin to obtain some altcoins. Another application is that someone uses the permanent preservation feature of the Bitcoin blockchain to save information, such as the content of a certain intellectual property. After hashing it, people can put its hash value after the return statement. This hash value does not occupy much space and will not reveal the specific content of your intellectual property. In this way, if someone claims that he is the author of this intellectual property right in the future, and a dispute occurs, you can pass this proof that you already know this knowledge at a certain point in time. This is similar to the coinbase domain in coinbase transaction, and the content of this domain is up to you. So why don't we just use the coinbase domain to implement this feature, so that we don't need to burn Bitcoin? This is because the coinbase domain method can only be used by nodes that have obtained the right to append, but any user can burn a small amount of Bitcoin in exchange for the opportunity to write content to the blockchain to be permanently stored.

Figure 73 shows a coinbase transaction, and there are 2 outputs. The first output is the normal pay to public key hash, and the output amount is block reward + transaction fee. The second output amount is 0, and the output script is the format of “return”. The purpose of this second output is to write something into the blockchain.

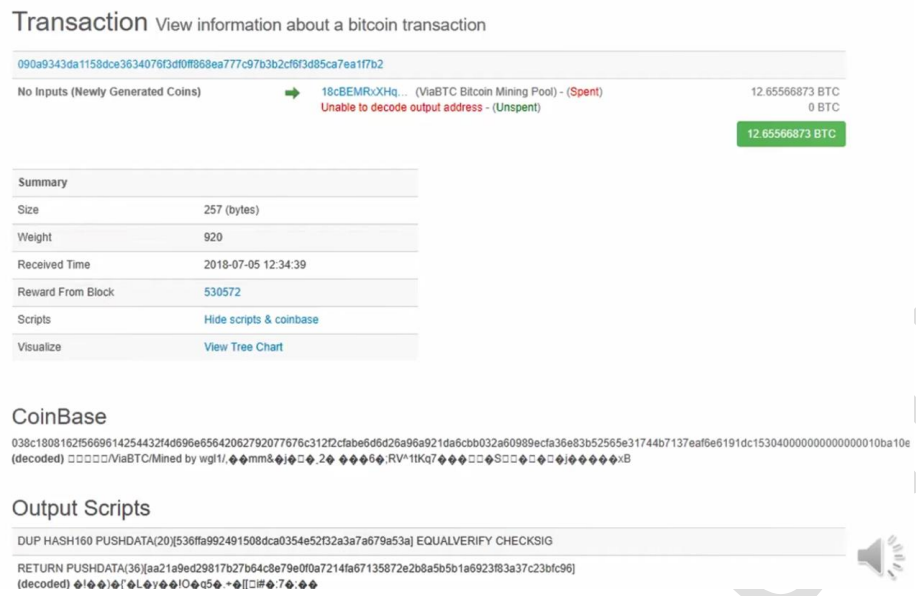


Fig 73

Figure 74 shows a general transaction. You can see that the output script starts with “return”. The input of this transaction is 0.05 bitcoins. The output amount is 0. This transaction does not actually destroy any bitcoins, but uses the input bitcoins as a transaction fee and gives it to the miners who got the block. When a miner sees this type of transaction, he knows that the output will never be cashed out, so there is no need to save it in UTXO.

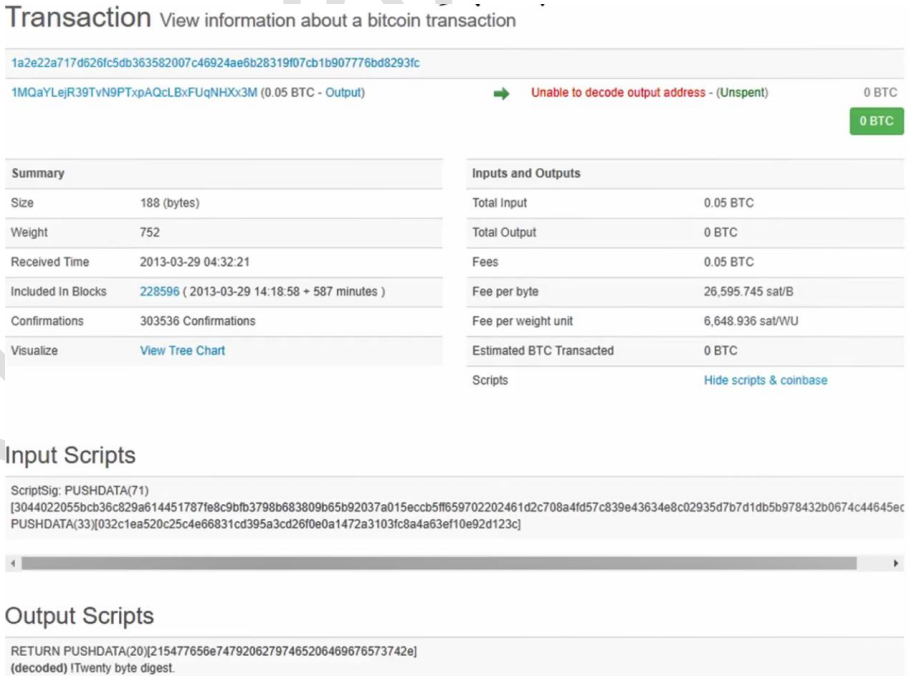


Fig 74

Bitcoin's script language is very basic. Ethereum's smart contract language is much more complicated. Bitcoin’s script language does not support loops, therefore its function is limited, but this limitation is actually intended. Since loops are not supported, there will be no dead loops, and there will be no halting problem concerns. Ethereum's contract language is Turing complete, so a gas fee mechanism is needed to prevent the program from falling into an dead loop. Note that although the Bitcoin script language has limited functions, it is very powerful in cryptographic applications. For example, the implementation of checkmultisig, which checks multiple signatures, can be implemented in one sentence, which is much more convenient than many general high-level languages. Therefore, although Bitcoin's script language is simple, it is however very well optimized.

Bitcoin Forking

Fork, as the name suggests, it means to split a blockchain. A fork may happen due to two different nodes mined a block at the same time, and both publish their block, this will cause a temporary fork called a state fork (a fork caused by a divergence of the blockchain state). The forking attack mentioned before is also a state fork, but the divergence is caused intentionally by malicious miners. So, it is sometimes called deliberated fork.

Another situation where a fork occurs is when the Bitcoin protocol changes. This change requires an upgrade of Bitcoin software, and due to the decentralized nature of Bitcoin, we cannot guarantee that all nodes will follow up with this upgrade. For example, some nodes have not had time to upgrade, or some nodes do not recognize the new protocol and are unwilling to upgrade. The fork generated in this case is called protocol fork. According to the content of the protocol modification, we can further subdivide it into hard fork and soft fork hard fork and soft fork.

1. Hard Fork

If the new protocol will add new features and functions of Bitcoin nodes that have not been upgraded will consider these new features and functions to be illegal. For example, Bitcoin block size is currently 1 MB, but some people think it is too small, which limits the throughput of Bitcoin and increases transaction delays.

1MB is 1,000,000 Bytes, and each transaction is about 250 Bytes. Therefore, each block can only contain about 4000 transactions to the maximum, and the average block time is ten minutes. So approximately Bitcoin can do 7 transactions per second. This speed is indeed too slow, credit cards transaction processing speed is much faster than this. Therefore, some people think that such a small block size limits Bitcoin's transaction throughput and increases latency.

Now suppose that the block size of Bitcoin software updates from 1 MB to 4 MB. And most nodes follow up this update and we call these nodes as “new nodes”. To be precise, this means that the nodes with most computing power in the system updated the Bitcoin software. Then the nodes that have a minority of computing power that have not been updated are called “old nodes” (Figure 75). Now when the “new nodes” mine they would mine big block(4MB). “Old nodes” would not recognize big blocks, they would keep mining small blocks (1MB).

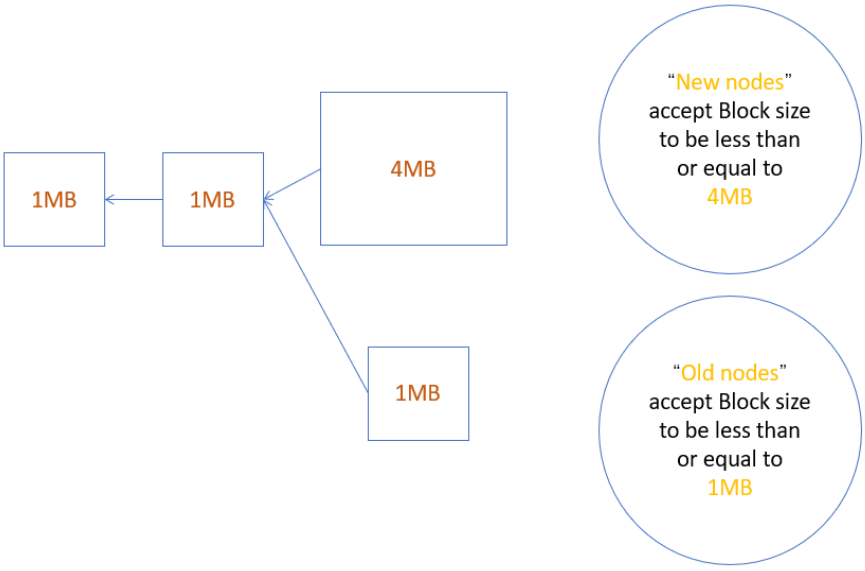


Fig 75

Note that the "new nodes" would approve the small blocks mined by the "old nodes", because the "new nodes" agreement only stipulates that the block size cannot exceed 4M (Fig 76).

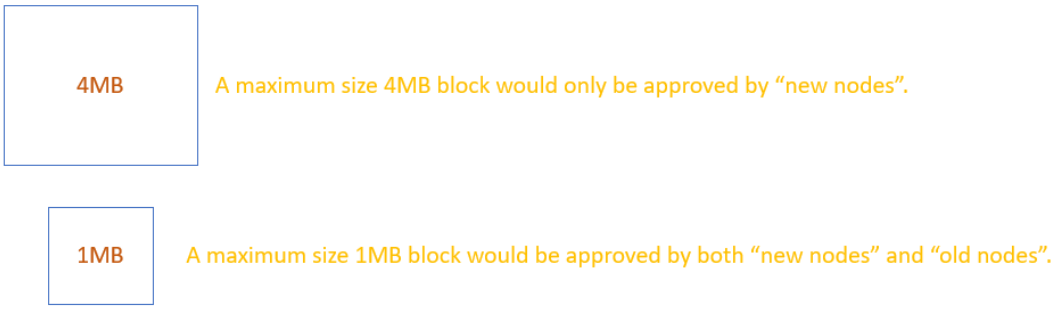


Fig 76

The situation now is that "new nodes" with a majority of computing power will mine along large blocks, and "old nodes" with a minority of computing power will mine along small blocks (Fig 77). Although the "new nodes" approve that both the upper and lower chains are legal, the upper chain is however the longest legal chain due to the strong computing power of the "new nodes", the upper chain length increases faster than the lower chain. Note that the "new nodes" may also publish some blocks less than 1MB in size. For example, in Figure 77, the "new nodes" released a 0.8MB block, which can be approved by both the old and new nodes. However, the "old nodes" will not recognize the chain where this block exists because there are illegal blocks to them in the upper chain (that is, blocks with a size exceeding 1MB). Therefore, such a fork is permanent. As long as these old nodes do not update the software, the fork will not disappear. Therefore, it is called a hard fork.

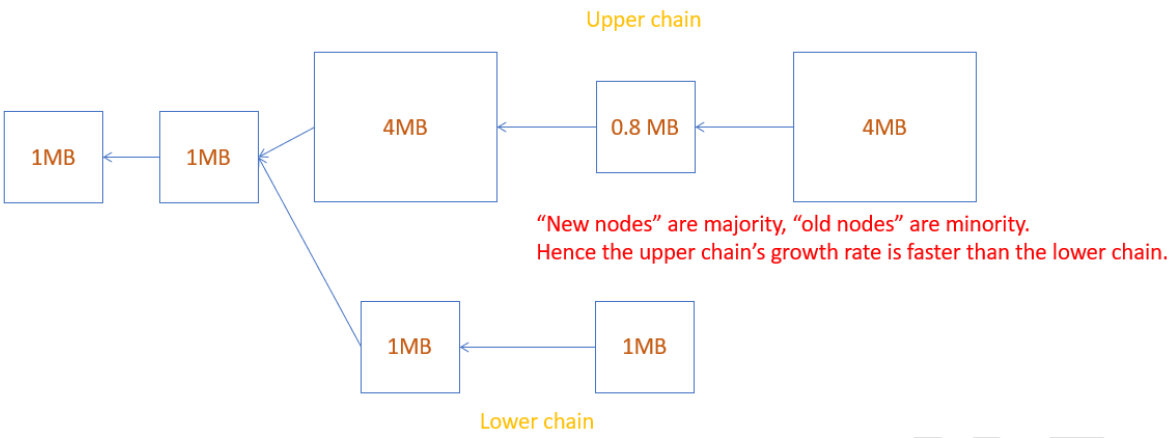


Fig 77

The block size is not the bigger the better, because the Bitcoin underlying network uses the p2p flooding mode, which consumes a lot of bandwidth. Bandwidth is the bottleneck. After a hard fork occurs, it becomes two chains running in parallel, and the two chains would have their own cryptocurrency. It will also lead to a split in the community, some people will recognize the upper chain, and some will recognize the lower chain. So, what happens to the coins before the hard fork after the fork? These coins will be approved by both the upper and lower chains. One original coin will split into two coins. For example, the current ETH is the product of a hard fork. The original Ethereum is actually ETC. Due to hacker attack, the community decided to hard fork into ETH.

Let's take ETH and ETC as example. After the hard fork, the two chains should actually record their own ledges, but because two chains were the same chain before the fork, the users' private keys and public keys are the same. If you don't take some measures, replay attach could happen, for example B to C is on the ETH chain, and then someone replays this transaction on ETC chain, causing B not only transfer 10 ETH to C, also transfers 10 ETC to C (Fig 78).

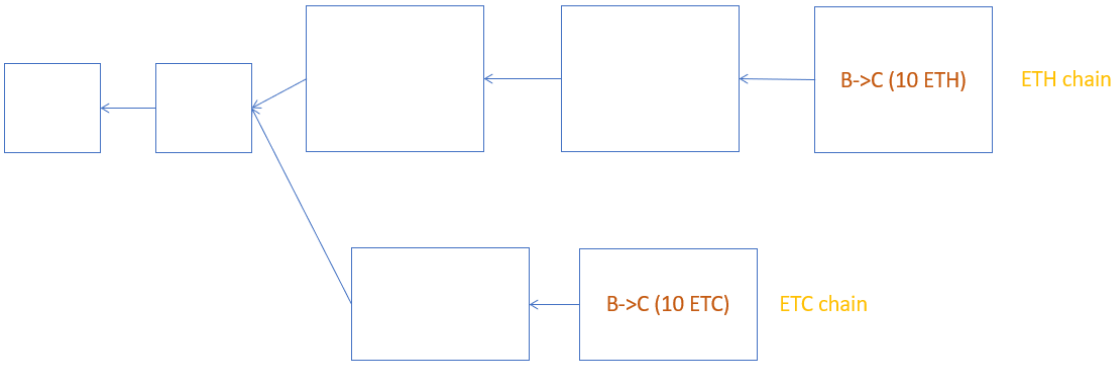


Fig 78

In order to prevent this kind of reply attack, the two chains must have a chain ID to distinguish the two chains after the hard fork.

2. Soft Fork

Next let's talk about soft fork. That is to add some restrictions to the Bitcoin protocol, the originally legal blocks and transactions may become illegal under new restriction, which will cause a soft fork. Now suppose our software update is to change the block size limit from 1M to 0.5M. Recall that decentralized software upgrades are not that easy like changing a couple of parameters. The upgrading method may lead to a hard fork or a soft fork (our example is for better understanding only, no one wants to change 1MB to 0.5 MB, because 1MB is indeed small). We still assume that the "new nodes" account for the majority and approve the 0.5MB software update (Fig 79).

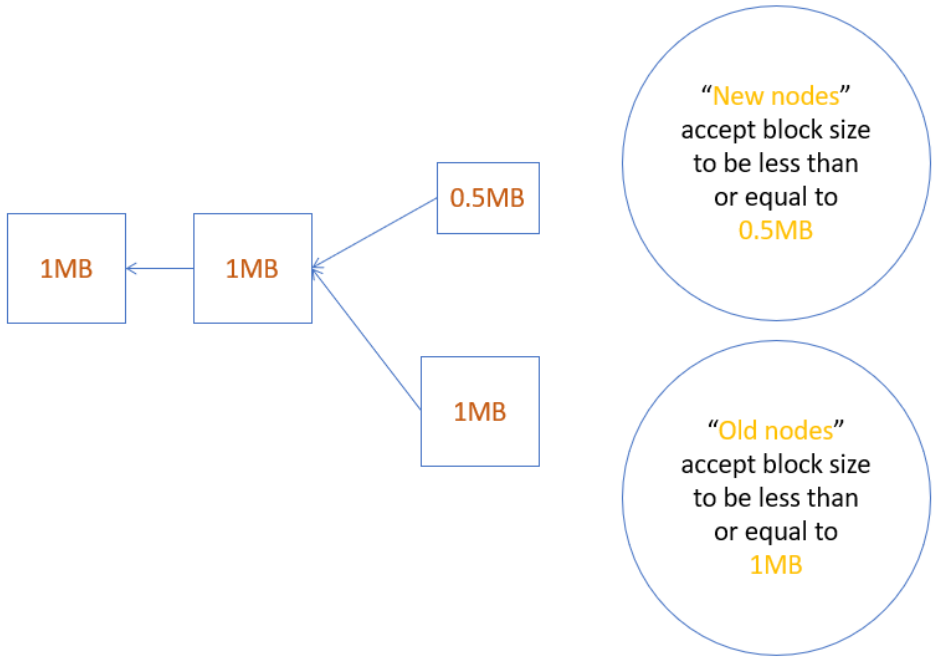


Fig 79

After the upgrade, the "new nodes" start to mine small blocks (0.5MB), which is also approved by the "old nodes". The "old nodes" will mine the original 1MB block, which is not approved by the "new nodes" (Figure 80).

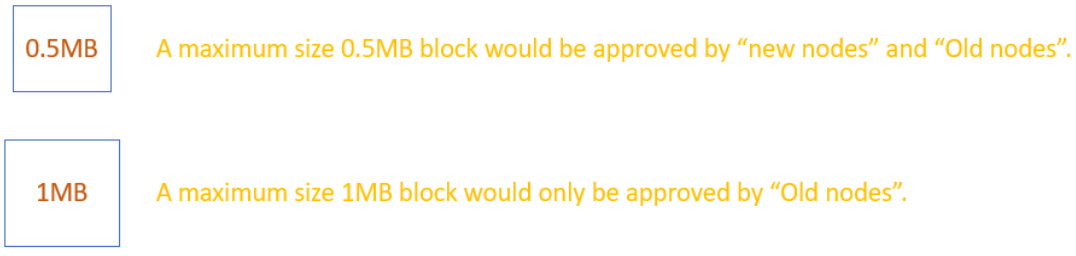


Fig 80

Although the “old nodes” approve both the upper and lower chains, according to the Bitcoin protocol rule that to extend the longest valid chain (the “new nodes” have strong computing power, the upper chain length grows faster), it will cause the “old nodes” to abandon their current fork chain and switch to the longest valid chain and continue mining after it (Figure 81). Then, the “old nodes” would mine the original 1MB block again. The “new nodes” do not approve this new 1MB block, so it forks once again and repeats. Therefore, if the “old nodes” do not update the software, it will cause them to mining for nothing but repeating this fork loop. This situation is temporary and will not cause a parallel running chain like a hard fork would. Therefore, this kind of fork is called a soft fork.

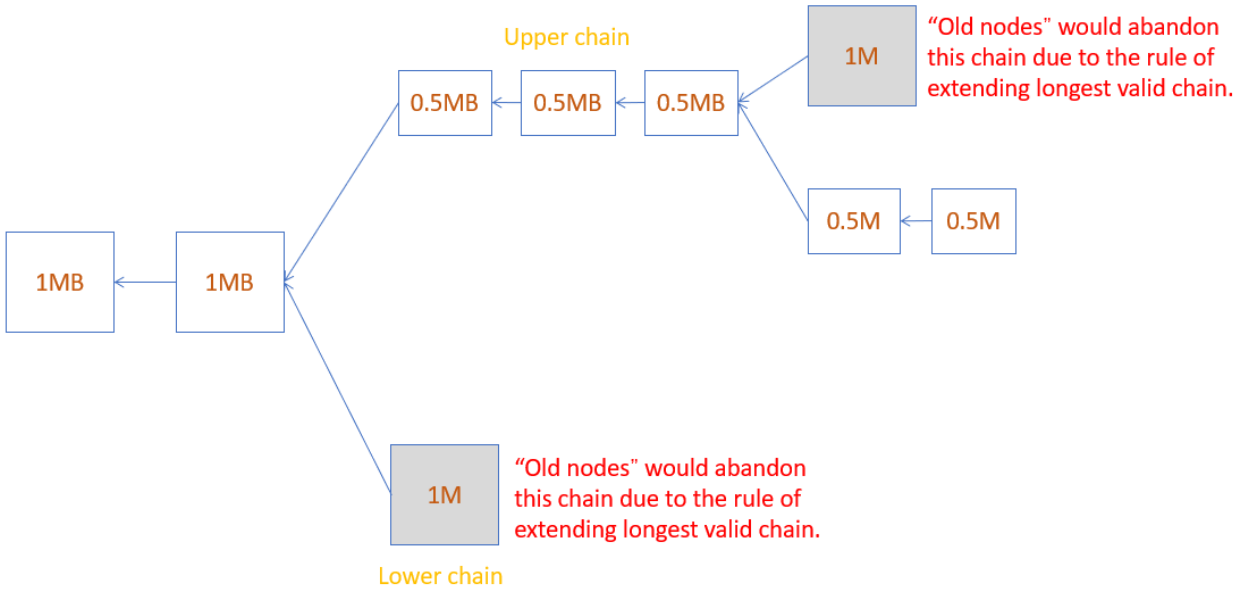


Fig 81

A possible soft fork case would be something like adding some new meanings to some unspecified domains in the current agreement and giving some new rules, such as to the coinbase field. There is currently no rules for this field, no one even checks it. We talked about this field can be used as an extra nonce filed for mining. Without this extra nonce, the search space is 4Bytes, only 2 to the power of 32, this search space is not large enough, so we borrow the first 8 bytes of the coinbase field as extra nonce to let us increase the search space to the 2 to the power of 96. However, the coinbase field is more than 8 bytes, and there is more space in it. Therefore, it was proposed to use the extra space for the root hash value of UTXO. At present, the UTXO is maintained by each full node in memory, so that the full node can quickly find and check whether the transaction is double spending. The content of this collection is not actually written into the blockchain.

Why did people make such a propose? We said before that a light node does not need to save the entire blockchain, only the block headers are enough. The light node verifies whether a transaction is in a certain block by sending a Merkel proof request to a full node. However, if we

want to figure out how many bitcoins a certain account has, the current bitcoin system can't actually prove it. If you are a full node, you can calculate how many bitcoins there are in the account through the account related transaction information in UTXO, but many blockchain wallets are just light nodes. Therefore, if a light node wants to know the balance of an account, it can only ask a full node and let the full node help with the calculation.

However, how can this light node prove that the answer given by the full node is true? The answer is it cannot be approved. If you don't maintain UTXO, you can't verify it. Therefore, some people propose to form a Merkel tree with the contents of UTXO, and then write the root hash of this tree in the coinbase field's extra space, because the stuff in the coinase field will affect the root hash value of the block header. In this way, Merkel proof can be used to prove how much balance an account has.

If this propose is made, it will cause a soft fork, because the blocks released by the "new nodes" are approved by the "old nodes" ("old nodes" do not care about the contents of the coinbase field). However, the "new nodes" do not approve the blocks published by the "old node" ("Old nodes" will not put UTXO Merkle tree's root has in the coinbase field).

A well-known example of soft fork in Bitcoin history is P2SH. This feature was not initially available, but was added through a soft fork.

In summary, after the hard fork, the "new nodes" will approve the blocks mined by the "old nodes". However, the "old nodes" will not approve the blocks mined by the "new nodes", which will result in permanent parallel chains; After the soft fork, the "new nodes" will not approve the blocks mined by the "old nodes", but the "old nodes" will approve the blocks mined by the "new nodes", which will result in temporary parallel chains.

Bitcoin Anonymity

Although you do not need your real name when generating the public key private key pair of the Bitcoin account, but in fact it is difficult for Bitcoin to achieve absolute anonymity. Cash transactions are truly anonymous. If a Bank did not need to open an account with a real name. Whoever has the account password can go withdraw money. Then, bank service's anonymity is actually better, because the bank ledger is private, on the contrary, Bitcoin ledger is completely public.

1. Anonymity Sabotage

Some people suggest that generate a new public key and private key pair every time you receive bitcoins, so that others cannot know which addresses belong to you, but can this really protect your privacy? Not really, because in some cases it is possible to link the various accounts you have created. For example, we said before that each transaction can have multiple inputs and outputs. Suppose we have a transaction like the following figure 82 published on the blockchain. It is very likely that addr1 and addr2 belong to the same person, because this person must control the private keys of these two accounts at the same time. Why there are multiple inputs? Because the things you want to buy with Bitcoin may exceed the balance in just one account you might need more accounts to pay for it. And, one of the output addresses below is likely to be an address for change. Such transactions are generally generated by Bitcoin wallet software. Many wallet software will generate a new change address every time for privacy. Is it still possible to associate input and output addresses? There may be n input addresses and can we figure out if these n addresses all belong to the same person? Can we also find out the address for change?

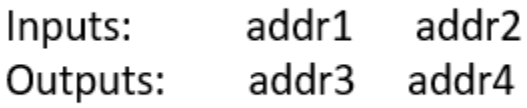


Fig 82

The answer to these questions is YES. For example, in the case of Figure 83, it is clear that addr4 is the address for change. Therefore, if we want stronger privacy protection, we can artificially generate unnecessary outputs. There are only a few commonly used Bitcoin wallets, so if you understand the details of these wallets, you can analyze all the Bitcoin blockchain transactions, because these wallets rarely have the function of deliberately generating unnecessary addresses.

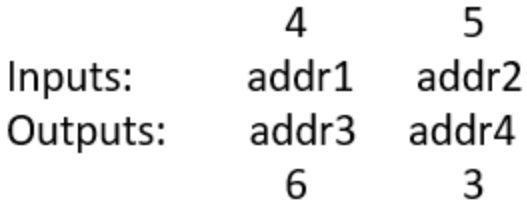


Fig 83

The case we talked about above is a person wants to increase his anonymity by generating multiple address accounts, but after our analysis and discussion, we found that it is possible to link all the accounts of this person.

Next, let's talk about the connection between your Bitcoin account address and your real identity in society. When can others figure out that a certain Bitcoin account corresponds to a certain person in reality? Whenever there is a connection between Bitcoin and the physical world. If you just play inside the blockchain system, it is difficult to determine your identity in real life, but once the real world is involved, such as the transfer of legal tender into and out of the blockchain system, it is likely to reveal your identity. For example, when we buy Bitcoin with dollars, we go to an exchange to buy it, and the exchange will first verify your true social identity. Even if you use OTC transactions (private transactions between individuals), or even gather people in a random park for trading, your identity may be revealed. In order to prevent money laundering with Bitcoin, many countries will focus on the transfer of funds into and out of the Bitcoin world. For example, if you want to turn millions of dollars into bitcoins, or a lot of bitcoins into dollars. This TX is difficult to not be noticed by the judicial department.

Now some merchants directly accept Bitcoin payments, but can this guarantee anonymity and privacy? There has been a study in the past. In order to study the habits of different groups of people using credit cards, an organization collected their TX information, then hashed the card number, and erased the identity of the card owners. Posted these TX information to the public. It did take a long time before people analyzed the hashed transfer records and revealed the identities of these card owners. How did they do it? For example, A knows the person B, and A happens to see that B bought something in a small shop at 2 o'clock this afternoon. Of course, A does not know B's card number, but when these TX records are public (after hashing) , A can filter out a lot of TXs based on the information that B shopping at 2 o'clock this afternoon. Then next time A sees that B buys something in a certain store at a certain moment, A can filter again to further pin point B. After filtering several times, A will soon be able to connect the hashed card number to B. This case tells us that even if the credit card number is hashed, people can still easily analyze who is who. And Bitcoin transaction information is public. If you use your Bitcoin account to pay for your Starbucks several times, people may analyze that this Bitcoin account represents you.

In summary, we can link your different accounts through analysis within the Bitcoin system. And when Bitcoin meets reality, we can figure out your identity. Therefore, the anonymity of Bitcoin is not so good. The person with the best protection of Bitcoin anonymity in the world is Satoshi Nakamoto. He has participated in Bitcoin for so long, but the Bitcoin he got has never been spent. If he/she/they spend the Bitcoins, it has to interact with the real world, which may expose the true identity of Satoshi Nakamoto. There used to be a website called silk road. It is a website for selling and buying illegal things. Basically, the eBay for darknet. In order to evade judicial sanctions, this website uses Bitcoin as the payment method, and the underlying network uses the Onion network. An anonymous mailing service is also used for shipping. The website was blocked by the government after it was running for a few years, and the owner was arrested. And this person can't spend his bitcoins at all, because this reveals his identity.

2. Anonymity Reinforcement

What methods should an ordinary Bitcoin user take to improve anonymity? We said before that Bitcoin runs on the application layer, and the bottom network layer is a p2p network. Therefore, we can improve anonymity from these two aspects. For example, if we go to an Internet cafe and use a screen name, is there any way for others to find out your identity? Of course, you have to register with your ID when you go to an Internet cafe, and then you can be located by checking your IP address. IP address is usually very closely related to the real identity. So first of all, we must find a way to achieve anonymity at the network layer. The good news is that we have a very good solution for the anonymity at the network layer. We can use multi-path forwarding methods, such as onion routing TOR. Each node that forwards this message only knows its previous node, but does not know who sent the message first.

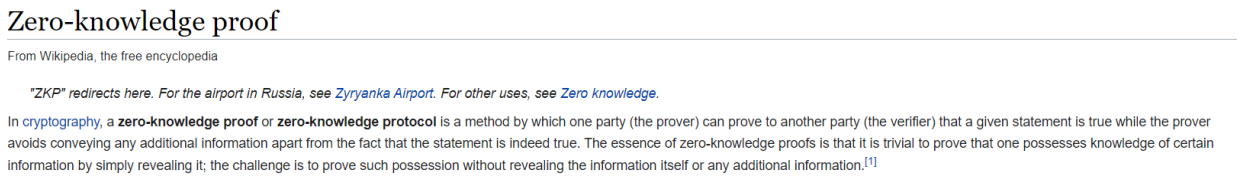
Let's talk about how to improve anonymity on the application layer. As we said before, because Bitcoin is an open ledger, every transaction can be traced back. Even if someone creates a bunch of Bitcoin accounts, we can still link these accounts of this person. However, we can use coin mixing method to get around it that is to mix the coins of different people together. Some websites provide this kind of coin mixing service. Everyone sends coins to this website, and then takes coins back. Generally speaking, the coins you get back would not be the coins you sent originally. However, the current coin mixing services are not very creditworthy, and they may run away with you coins.

Another similar method is that we don't need to do coin mixing intentionally, because some applications have the nature of coin mixing, such as online wallets. Everyone deposits the coins, and when you retrieve your coins later, it may not be the coins you deposited at the beginning. Online wallets have the effect of coin mixing, but there is no guarantee that they will be mixed for sure. Or you can use cryptocurrency exchanges, because most exchanges have the nature of coin mixing. For example, if you have 10 bitcoins, you send them to the exchange, and turn the 10 bitcoins to some stablecoins, then use stablecoins to buy Litecoins, etc. Finally you buy back bitcoins. Then withdraw these bitcoins to your local wallet. Generally speaking, these bitcoins

are not the bitcoins you deposited in the first place. Of course, this premise is that the exchange will not disclose your transaction records to the government. In fact, it is quite difficult for the Bitcoin blockchain to maintain privacy, because Bitcoin is an open ledger, and the information on blockchain is very hard to change. The immutability and publicity are disastrous for privacy protection.

3. Zero-knowledge proof & Blind Signature

Let's talk about Zero-knowledge proof (Fig 84)



Zero-knowledge proof

From Wikipedia, the free encyclopedia

"ZKP" redirects here. For the airport in Russia, see Zyryanka Airport. For other uses, see Zero knowledge.

In cryptography, a **zero-knowledge proof** or **zero-knowledge protocol** is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true while the prover avoids conveying any additional information apart from the fact that the statement is indeed true. The essence of zero-knowledge proofs is that it is trivial to prove that one possesses knowledge of certain information by simply revealing it; the challenge is to prove such possession without revealing the information itself or any additional information.[1]

Fig 84

For example, I (the certifier) wants to prove to you (the verifier) that a certain bitcoin account is mine, but I definitely cannot tell you my private key. How do I prove it? I can generate a signature and provide it to you. If you know the public key of this account, you can use my signature to verify it. This example is actually controversial whether it is a zero-knowledge proof, because I disclosed the signature generated by the private key.

Zero-knowledge proof is based on homomorphic hiding (Fig 85)

Homomorphic hiding

- **If x != y, then E(x) != E(y).**
- **Given the value of E(x), it is almost impossible to get x.**
- **Given the values of E(x) and E(y), we can easily calculate the E() value of x + y, x * y, etc.**

Fig 85

The first property indicates that there will be no collisions. The hash function we talked about at the very beginning may collide. In homomorphic hiding, if the input x is not equal to y, then the values of E(x) and E(y) must be different. This also shows that if E(x) and E(y) are equal, then x must be equal to y.

The second property shows that this encryption function is not reversible. It is difficult to infer the value of x by knowing the value of E(x). This is similar to the hiding of the hash function.

The third property is homomorphic operations. It means that performing algebraic operations on the encrypted values is equivalent to performing algebraic operations first and then do encrypting operations.

Figure 86 shows a zero-knowledge proof example.

Example

- **Alice wants to prove to Bob that she knows a set of x and y so that x + y = 7. Alice can not tell Bob the value of x and y.**

Fig 86

Figure 87 shows the simple solution.

Simple solution

- **Alice provide E(x) and E(y) to Bob.**
- **Bob can calculate E(x + y) by E(x) + E(y).**
- **Bob then calculates E(7), if E(7) = E(x + y) verification is true.**

Fig 87

However, this method is flawed, because Bod can find the values of x and y using brute force. Therefore, Alice should do some randomization before giving E(x) and E(y) to Bod but Alice must ensure that the value of x+y remains unchanged.

Recall that we talked about a centralized digital currency design that let central back to maintain a database to keep track on each signed/serialized digital currency file. This model is centralized, because the central bank has to participate in verifying every transaction, so the central bank knows all your transaction information. Is there a way for the central bank to do this kind of centralized ledgering (check double spending) without letting the central bank know the specific parties of transactions?

Yes, it is possible but the central bank cannot generate the serial number of each digital currency file, and users have to generate the serial numbers of these digital currency file locally, and do not reveal these serial numbers to the central bank.

For example, A wants to transfer money to B from his deposit in the central bank, but he does not want the bank to know that B's money is given by A. Blind signing method is used here:

- The first step is that A generates the serial number locally, and then the bank signs it without knowing the specific serial number. Then reduces A's balance.
- In the second step, A tells B the actual serial number, and gives B the signature that the bank signed in the first step.

- In the third step, B tells the bank the actual serial number and gives the signature obtained from A to the bank. Let the bank do the verification (that is, check whether there is a double spend), and after the verification is successful, increase the balance of B.

4. Zerocoin & Zerocash

Bitcoin provides a certain degree of anonymity, but it cannot completely eliminate connections. Can we redesign a cryptocurrency and use cryptography principles to ensure anonymity from the beginning? Yes, Zerocoin and Zerocash are specifically designed for anonymity (Fig 88).

Zerocoin & Zerocash

- **Zerocoin and Zerocoin integrate anonymization at the protocol layer, and their anonymity comes from cryptographic guarantees.**
- **There is basecoin and a zerocoin in the Zerocoin system. Through the conversion between the basecoin and zerocoin, the correlation between the old address and the new address is eliminated.**
- **The Zerocash system uses the zk-SNARKs protocol and does not rely on basecoin. The blockchain only records the existence of transactions and the proof of the key attributes required by the miners for the normal operation of the system. Neither the transaction address nor the transaction amount is displayed on the blockchain. All transactions are carried out by zero-knowledge proof verification.**

Fig 88

There is a so-called base currency in Zerocoin, for example, it can be Bitcoin. We turn Bitcoin into Zerocoin. Zerocoin uses zero-knowledge proof to prove that the Zerocoin you spend is legal. In other words, bitcoins are all traceable, and you know which coin in the system you are spending. However, Zerocoin only proves that the Zerocoin you are spending was legally existing on a blockchain, but you don't know the source of the coin. Therefore, relevance and traceability are blocked. **Zerocash** does not have base currency. The specific implementation of these two currencies requires knowledge of cryptography. At present, these Zerocoin and **Zerocash** are not mainstream, because the encryption currency causes performance loss for anonymity, and the random source used may have loopholes. In addition, there are not many users who need stronger anonymity. And they are not 100% anonymous, because when they interact with the physical world, there is still a risk of exposure your true identity.

Reference

Blockchain and Application, Beijing University, Professor Zhen Xiao, http://zhenxiao.com

Computer Network, University of Southern California, Professor Shahin Nazarian, Shahin Nazarian, PhD (usc.edu)

Blockchain from Beginning to Master, BlockChainSaw, (1711) BlockChainSaw - YouTube

What is a Digital Signature? (youdzone.com)