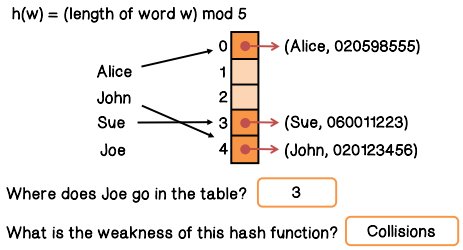
Acknowledgements

First, I would like to acknowledge that the slides that I am going to present are from Joseph Bonneau, Ed Felten, Arvind Narayanan, and Andrew Miller.

Where Does Joe Go Quiz



First, let us do a quiz on hash table. Suppose we want to store pointers to student records in a hash table, the hash function is the length of the name mod 5. Therefore, for Alice, since the length is 5, her record is going to be stored in slot 0, John will be in slot 4, and Sue will be in slot 3, so where will Joe go?

Since the length is three, you should go to slot three, but Sue is already in slot three. Therefore, there is a collision, in fact, that is the weakness of this hash function.

Hash Function Quiz

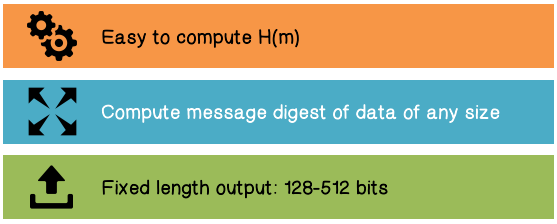
With regards to hash functions select all the true statements:

- [x] Hash functions do not have a key
- [x] Hash functions are also called one-way encryption
- [ ] A major drawback of hash functions is the possibility of two messages having the same hash value
- [x] Hash functions are primarily used for message integrity

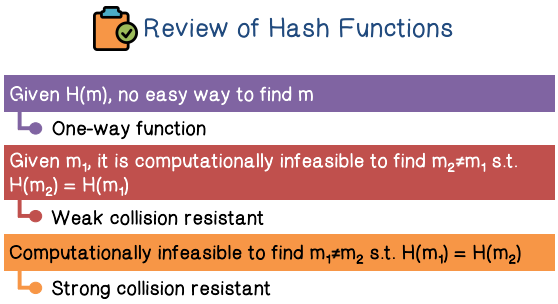
Now, let us do a quiz on cryptographic hash functions. Which statements are true?

Cryptographic hash functions do not have a key. A major drawback of cryptographic functions is that it is easy to find two messages that have the same hash value. This is a false statement. Cryptographic hash functions are primarily used for message integrity. That is true.

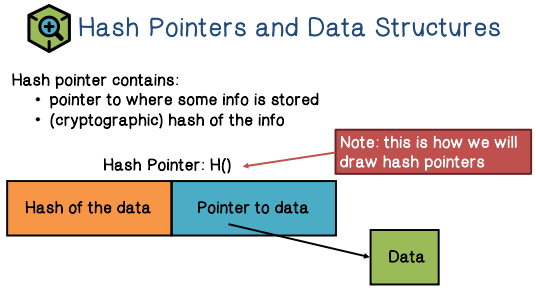
Review of Hash Functions



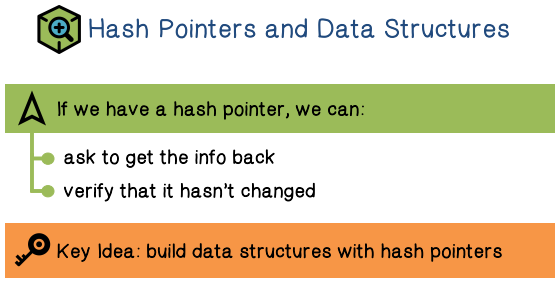
Now, let us have a quick review of hash functions. There are several important properties of cryptographic hash functions. First, it is easy to compute. You can compute the hash of a message of any size. For a given hash function, it produces fixed-length output.



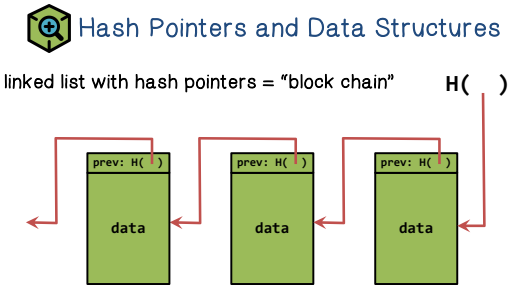
Hash function is a one-way function, that is, given the hash of the message, it is not easy to find the original message. Hash functions are also designed to be collision resistant. That is, given a particular message, m_1, it is computationally infeasible to find a different message, m_2, such that m_2 and m_1 have the same hash value, and that is the weak collision-resistant property. A strong collision resistant property states that, computationally, it is infeasible to find two different messages such that they have the same hash value.



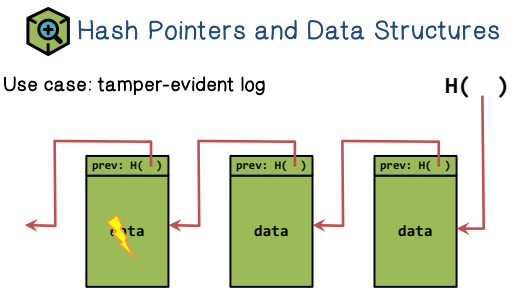
Now, let's introduce a new data structure called hash pointer. We know that a typical data pointer is the address to where the data is stored. A hash pointer contains a pointer to where the data is stored but also the hash of that data. That is, we expand the data pointer to include the hash of the data. We are going to use this notation to represent hash pointers.



If we have a hash pointer, obviously, we can use the data pointer to get the data back. But since we have the hash, we can also verify that the data has not been changed. We can use hash pointers to build very useful data structures.

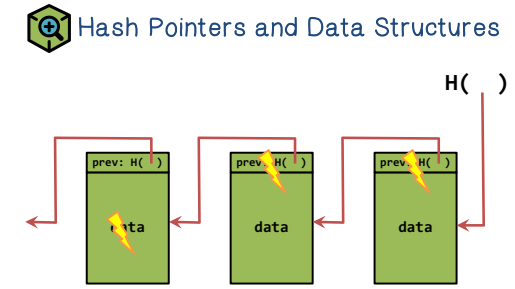


In particular, we can build blockchain, which is a linked list of hash pointers. Suppose we have a hash pointer pointing to a block. This block contains not only data value but also the hash pointer of the previous block, and likewise, the previous block contains not only the data values but also a hash pointer to the previous block. And this is called a blockchain. To construct, or, expand, a blockchain, we append a block at the end and include a hash pointer in the block. We

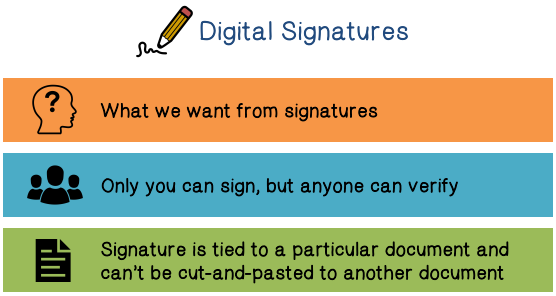


compute a hash of this last block and store the hash pointer. This hash pointer is the root of this blockchain.

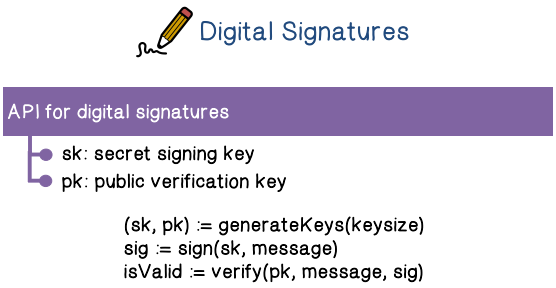
A use case for blockchain is a tamper-evident log. That is, we want it to be a log data structure that stores a bunch of data and allows us to append data onto the end of the log. But if somebody changes the data in the log, it is easy for us to detect that. We can see that if an attacker changes data in this block, the hash will not match the hash pointer stored in the next data block.



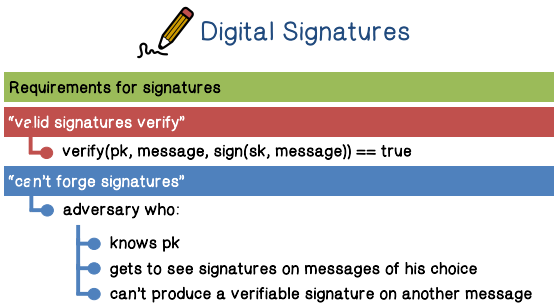
Of course, the attacker can then change the hash value stored in this hash pointer. But the result is that the hash value of this block will not match the hash value stored in the hash pointer of the next block. The attacker can again change this hash value. But then, it will not match the hash value stored in the root. Therefore, tampering a data item will cause the difference of the computed hash of the whole chain to be different than the hash value stored in the hash pointer to the whole chain. That is, by just remembering this single hash pointer, we can easily detect data tampering.



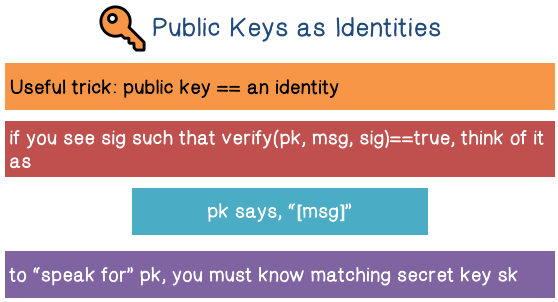
Now, let us have a quick review of digital signatures. What we want from signatures is that only the owner of the private key can sign, but anyone can verify because they can retrieve the owners public key. Further, a signature is tied to a particular document.



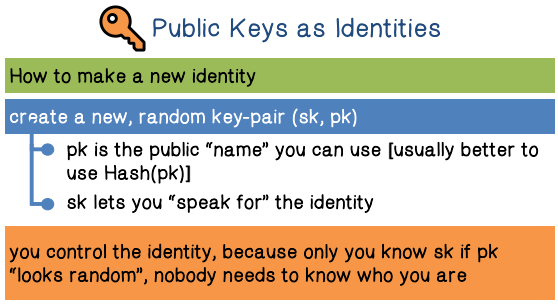
Let us review how digital signatures are created and verified. We have a private signing key and a public verification key. The key pair can be created by specifying the key size. Then, for particular message, we can create a signature by essentially encrypting the message using the private signing key. Given the message and the signature, any person who has obtained the public key can verify the signature. The verification is performed by using the public key to decrypt the signature and matching the result with the message.



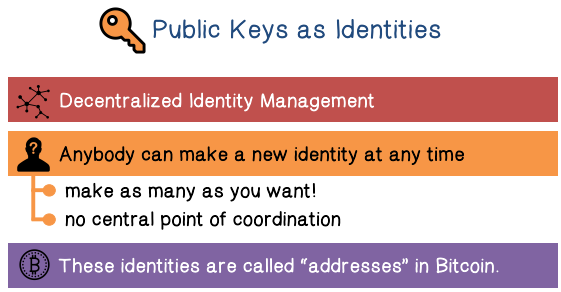
What are the requirements for signatures? Obviously, a valid signatures can be verified, and no one can forge a signature. Public key cryptography guarantees that although anyone can know the public key, but without the private key, which is only known to the owner, no one can forge a signature.



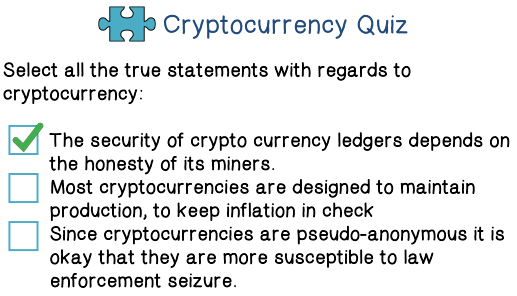
Public key can be used as an identity. For example, if you can verify a signature using a person's public key, you can think of it as a person who owns the public key has stated the message. Of course, you can create a valid signature only because you have the private key. Therefore, by creating a valid signature or speak of a message, it means that you can prove to anybody that you are already the owner of the public key.



Here is how we can use public key to create a new identity. First, we will create a new random pair of public key and private key. The public key or the hash of the public key becomes your name or your ID. But the private key is really what you can use to prove your identity. The advantage of this approach is that since only you have the private key, nobody can forge your identity; and if the public key looks very random, nobody really knows who you are.



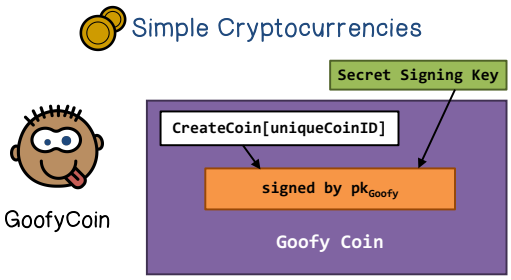
Another advantage is decentralized identity management. That is, anybody can create a new identity at any time. In fact, you can make as many identities as you want. There is no central point of control or coordination. These public key-based identities are called addresses in Bitcoin.



Before we discuss cryptocurrency, let us have a quiz. Select or choose true statements with regards to cryptocurrency.

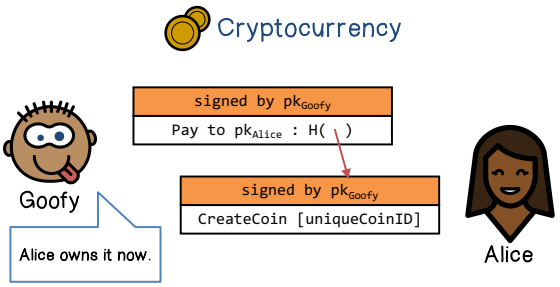
First, the security of cryptocurrency ledgers depends on the honesty of its miners. This is true, and we will discuss some of the details later. Second, most cryptocurrencies are designed to maintain production to keep inflation in check. This is false. Most cryptocurrencies are designed to reduce production mimicking the markets of precious metals. Third, since cryptocurrencies are pseudo-anonymous, it is okay that they are more susceptible to law enforcement seizure. This is false. Cryptocurrencies are pseudo-anonymous and they are less susceptible to law enforcement seizure.

Now let us start with very simple cryptocurrencies. GoofyCoin is about the simplest cryptocurrency we can imagine. There are just two rules of GoofyCoin. The first rule is that Goofy can create new coins anytime he wants and the second rule is that the newly created coins belong to Goofy.



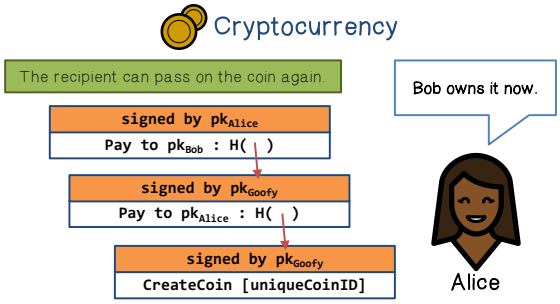
To create a coin, Goofy generates a uniqueCoinID, that is, ID that he has never used before. Then he construct a string, CreateCoin[uniqueCoinID]. He then computes the digital signature of the string using his private signing key. He then computes the signature on this string using his private signing key. The string, together with his signature is a coin. Anyone can verify that the coin contains Goofy's valid signature of a CreateCoin statement and is therefore a valid coin.

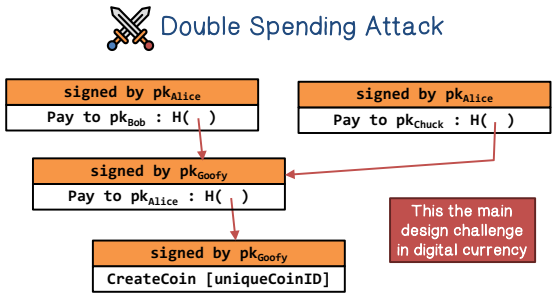
Another rule of GoofyCoin is that whoever owns a coin can transfer the coin to someone else. Transferring a coin is not simply a matter of sending a coin data structure to the recipient, it is done using cryptographic operations.



Let us say Goofy wants to transfer a coin that he created to Alice. To do this, he creates a new statement that says "Pay this to Alice" where this is a hash pointer that references the coin that he created and that he wants to transfer to Alice. As we discussed earlier, identities are just public keys and therefore "Alice" here is really the public key of Alice. Goofy also signs the statement because he owns the coin and therefore he has to sign any transaction that spends the coin. Once this data structure in particular with Goofy signature is created, Alice now owns the coin. She can prove to anyone that she owns the coin because she can present the data structure with Goofy's valid signature. In particular, this data structure contains not only the statement signed by Goofy, it also points to the very coin that was owned by Goofy. Therefore, anyone can verify that this transfer is valid and that Alice is now the new owner of the coin.

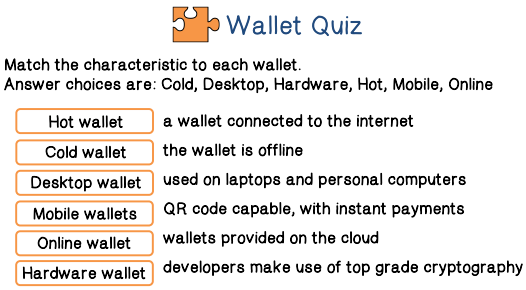
Once Alice owns the coin, she can spend it in turn. To do this, she creates a statement that says, "Pay this to Bob's public key" where this is a hash pointer to the coin that was owned by her and of course, she signs the statement. Now Bob owns the coin and anyone can verify that Bob is really the new owner. That is, anyone can follow the chain of hash pointers to the coin's creation and verify that at each step, the rightful owner signed a statement that says, "Pay this coin to a new owner".





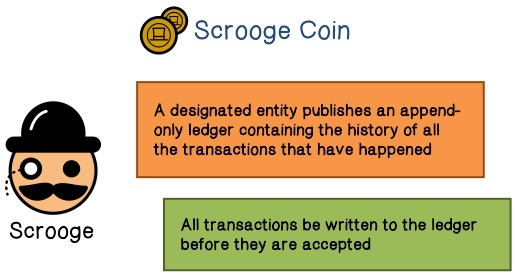
Let us say Alice pass a coin to Bob by sending her a signed statement to Bob, but did not tell anyone else. She could create another signed statement that pays the very same coin to Chuck. To Chuck, this would appear to be a perfectly valid transaction, and now he thinks he is the owner of the coin. Bob and Chuck would both have valid-looking claims to be the owner of this coin. This is called a double spending attack. That is, Alice is spending the same coin twice. Intuitively, we know coins are not supposed to work this way. In fact, double spending attacks are one of the key problems that any cryptocurrency has to solve. GoofyCoin does not solve the double spending attack, and therefore, it is not secure. GoofyCoin is simple, and its mechanism for transferring coins is very similar to Bitcoin. But since it is insecure, it is not a cryptocurrency.

Now let us do a quiz on digital wallets. Match the characteristics of each wallet, with a name.

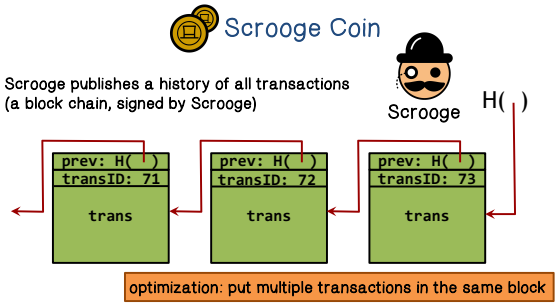


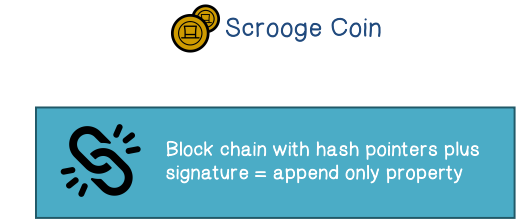
First, a wallet is connected to the internet. This is called a hot wallet. Second, the wallet is offline. This is called the cold wallet. Third, used on laptops and personal computers, these are the desktop wallets. Fourth, QR code capable with instant payments, these are the mobile wallets. Fifth, wallets provided on the cloud, these are the online wallets. Developers make use of top grade cryptography. These are the hardware wallets.

To solve the double spending problem, we will design another cryptocurrency, which we will call Scrooge Coin. Scrooge Coin is built on GoofyCoin, but it is more complicated in terms of data structures. The first key idea is that a designated entity called Scrooge publishes an append-only ledger that contains the history of all the transactions that have happened. The append-only property ensures that any data written to this ledger will remain forever. If the ledger is truly append-only, we can use it to defend against double-spending by requiring that all transactions to be written to the larger before they accepted. That way, it will be visible to anyone if coins were previously sent to a different owner.

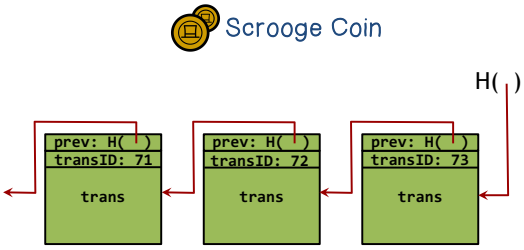


To implement this append-only feature, Scrooge can create a blockchain that he will sign. This blockchain is a series of data blocks. Each with one transaction or multiple transactions. Each block contains the IDs of the transactions, the contents of the transactions, and a hash pointer to the previous block. Scrooge signs a final hash pointer, which binds all the data in the entire blockchain and then publishes the signature along with the blockchain. In Scrooge Coin, a transaction is only valid if it is in the blockchain signed by Scrooge. Anybody can verify that a transaction was endorsed by Scrooge, by checking Scrooge's signature on the block that it appears in. Scrooge makes sure that it does not endorse a new transaction that attempts to spend an already spent coin.

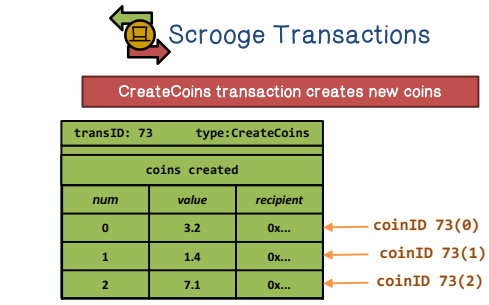




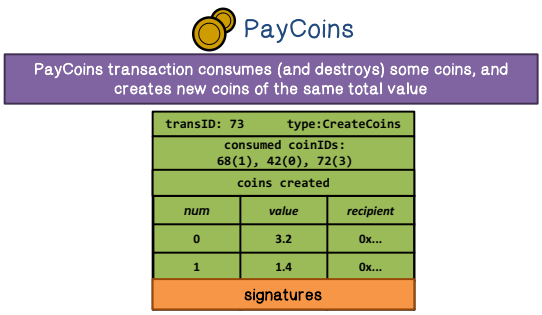
Why do we need a blockchain with hash pointers in addition to having Scrooge signs each block? This ensure the append-only property. If Scrooge tries to add or remove a transaction to the history or change an existing transaction, it will affect all of the following blocks because of the hash pointers.



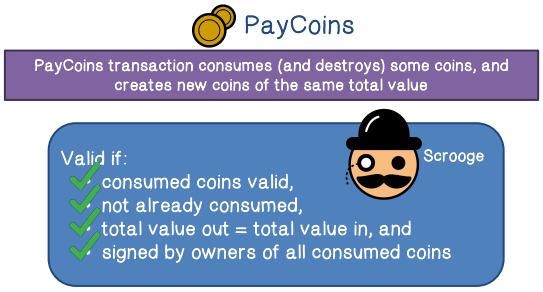
That is, as soon as someone is monitoring the latest hash pointer published by Scrooge, the change will be obvious and easy to catch. In a system where Scrooge signs blocks individually, you will have to keep track of every single signature Scrooge ever issued. A broad chain makes it easy for any two individuals to verify that they have observed the exactly same history of transactions signed by Scrooge.



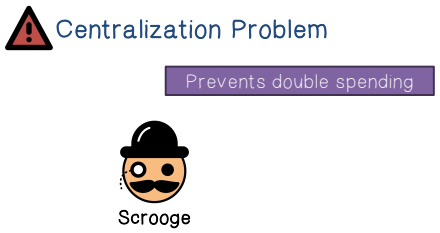
In Scrooge coin, there are two kinds of transactions: the first kind is CreateCoins. In this example, a CreateCoins transaction creates multiple coins. Each coin has a serial number within a transaction number. Each transaction has an ID, and each coin has a serial number within a transaction. Each coin has a value, meaning it is worth a certain number of Scrooge coins and also a recipient, which is the recipient's public key. A Scrooge coin transaction is always valid by definition if it is signed by Scrooge.



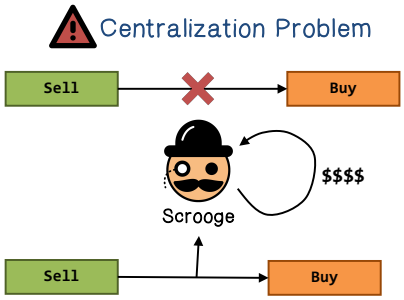
The second kind of transaction is called PayCoins. It consumes some coins, that is, destroys them and creates new coins of the same total value. The new coins might belong to different people, that is, different public keys. This transaction has to be signed by everyone who is paying in a coin, that is, if you are the owner of one of the coins that is going to be consumed in this transaction, then you have to sign this transaction to say that you are really okay with spending this coin.



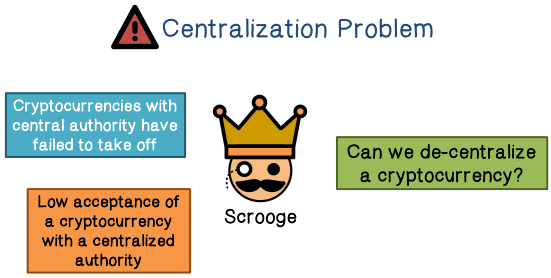
The rules of Scrooge coin say that a PayCoins transaction is valid if four things are true. First, the consumed coins are valid. They really were created in previous transactions. Second, the consumed coins were not already consumed in some previous transaction, that is, this is not a double-spending. Third, the total value of the coins that come out of this transaction is equal to the total value of the coins that went in, that is, only Scrooge can create new value. Forth, the transaction is correctly signed by owners of all the consumed coins. If all of these conditions are met, then this PayCoins transaction is valid and Scrooge will accept it. He will write it into the history by appending it to the blockchain, after which everyone can see that this transaction has happened. It is only at this point that the participants can accept that this transaction has actually occurred.



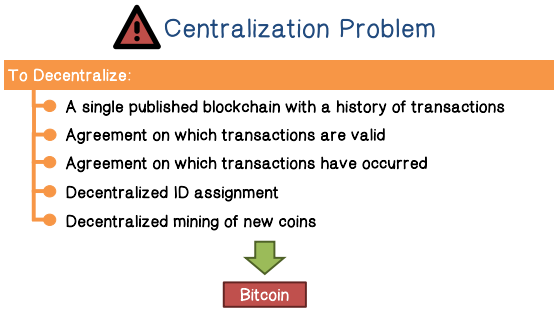
Now we come to the core problem with ScroogeCoin. ScroogeCoin will work in a sense that people can see which coins are valid. It prevents double spending because everyone can look into the blockchain and see that all the transactions are valid and that every coin is consumed only once.



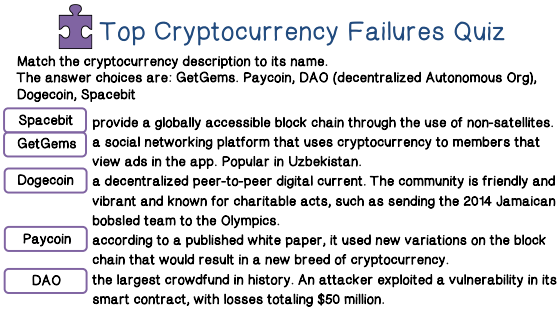
But the problem is that Scrooge has too much influence. He cannot create fake transactions because he cannot forge other people's signatures. But Scrooge could stop endorsing transactions for some users, denying their service and making their coins unspendable, or Scrooge can be greedy and he could refuse to publish transactions unless he gets a fee. Scrooge can of course create as many new coins for himself as he wants.



The problem here is centralization. Although Scrooge is happy with the system, the users might not be. Cryptocurrencies with a central authority have largely failed to take off and the main reason is that it is difficult for people to accept a cryptocurrency with a centralized authority. Therefore, a key technical challenge is how we can create a cryptocurrency without a central authority.

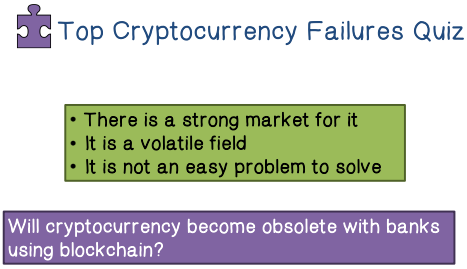


To do this, we must figure out how we can agree upon a single published blockchain with history of all the transactions. The users must all agree which transactions are valid and which transactions have actually occurred. They also need to be able to assign IDs to things in a decentralized way. Finally, the mining of new coins needs to be controlled in a decentralized way. If we can solve all of these problems, then we can build a currency that will be like ScroogeCoin but without a centralized authority. In fact, this will be a system very much like Bitcoin.

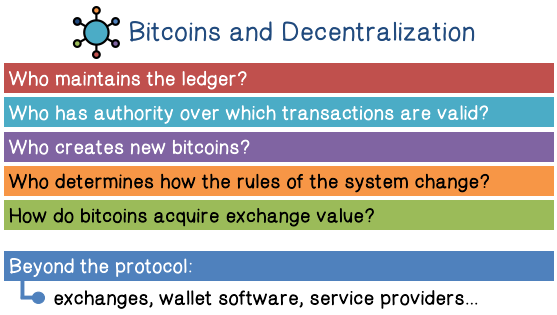


Now, let us do a quiz on cryptocurrency failures, match the description to its name.

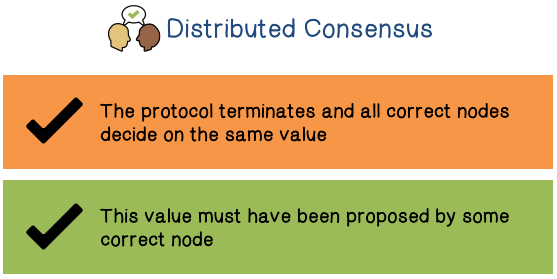
First, provide a globally accessible blockchain through the use of nanosatellites. This is Spacebit. Second, a social networking platform that uses cryptocurrency to members that view ads in the app. Popular in Uzbekistan. This is GetGems. Third, a decentralized peer-to-peer digital currency. The community is friendly and vibrant, and known for charitable acts such as sending the 2014 Jamaican bobsled team to the Olympics. This is Dogecoin. Fourth, according to a published white paper, it used new variations on the blockchain that would result in new breed of cryptocurrency. That is Paycoin. Fifth, the largest crowdfund in history. An attacker exploited a vulnerability in its smart contract with losses totaling $50 million. This is DAO.



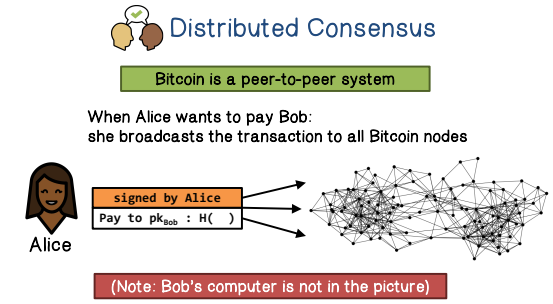
This quiz highlights some important points about cryptocurrency. There is a strong market for it, but it is also a volatile field, and it is not an easy problem to solve. To further muddy the water, at the end of 2016, there was talk of cryptocurrencies disappearing altogether.



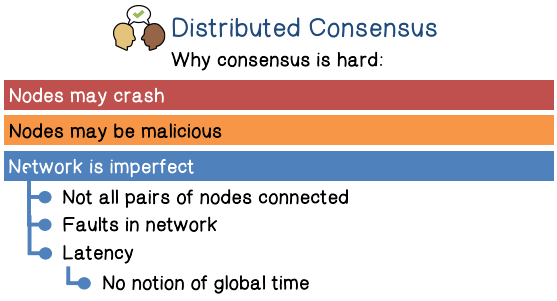
The key technical problems we need to solve in bitcoins are, who maintains the ledger or the block chain, who can decide which transactions are valid, who creates new bitcoins, who determines how the rules of the system can change, and how do bitcoins acquire exchange values? In addition, we need infrastructure support for bitcoins. That includes exchange, wallet software, and service providers, and so on.



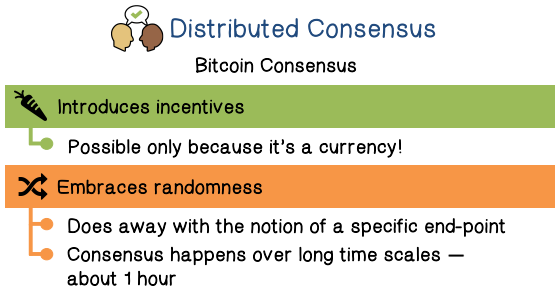
Since there is no centralized authority in Bitcoin, it is very important for us to understand how we can achieve distributed consensus. In Bitcoin, there is no centralized authority and therefore we need to understand how we can achieve distributed consensus. There are two basic requirements for distributed consensus. First, the consensus protocol terminates when all correct nodes decide on the same value, and the value must have been proposed by some correct node. There has been a lot of research on distributed consensus.



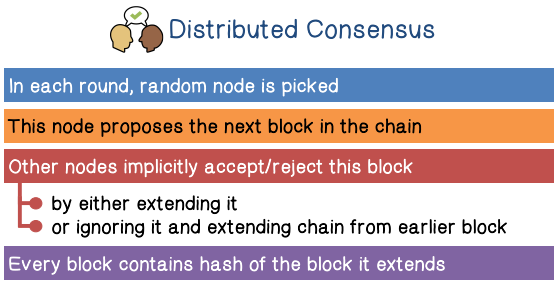
Now let us look at a unique setting of Bitcoin. First, Bitcoin is a peer-to-peer system. When Alice wants to pay Bob, she simply broadcasts a transaction to all Bitcoin nodes. In fact, Alice does not know where Bob's computer really is and the computer may not be online.



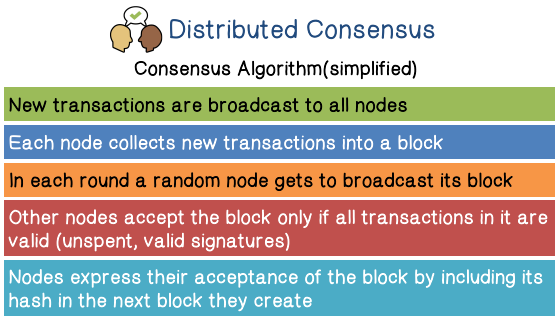
Achieving consensus in a peer-to-peer system is very hard because some nodes may crash, some nodes may be malicious, the network is not perfect because not all pairs of nodes are connected and there are network faults and latency.



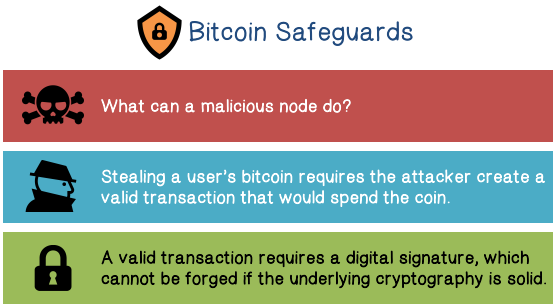
Now let us discuss how we can achieve consensus in Bitcoin. The trick is to use incentive and this is possible only because Bitcoin is a currency. And we embrace randomness or the imperfect nature of the peer-to-peer network by doing away with the exact termination point of the consensus protocol. The new protocol achieves consensus over a long timescale, for example, in about an hour.



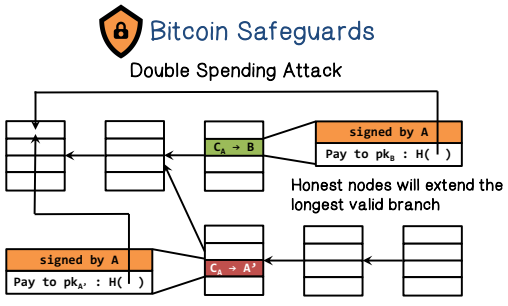
The key idea here is implicit consensus. At each round a random node is picked, this node then proposes the next block in the block chain. Other nodes implicitly accept or reject this block. If they accept they will extend the block chain from this block. If they reject they will ignore this block and instead extend the block chain from an earlier block. Every block here contains not only the transactions, but also a hash pointer to the block that it extends from.



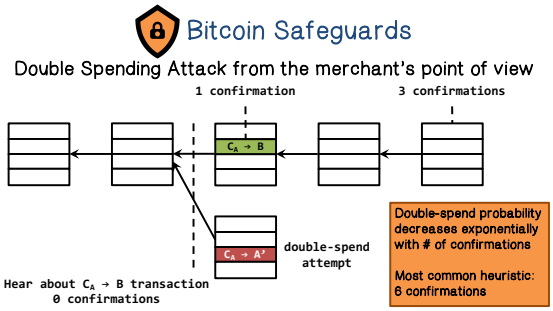
Here is a high level discussion of the consensus algorithm. First, new transactions are broadcast to all nodes, each node collects new transactions into a block, at each round a random node gets a chance to broadcast its block. Other nodes would accept this block if all of the transactions in it are valid, the signatures are valid and it is not a double spending. If the nodes accept this block, they will include a hash pointer to this block in the next block that they create.



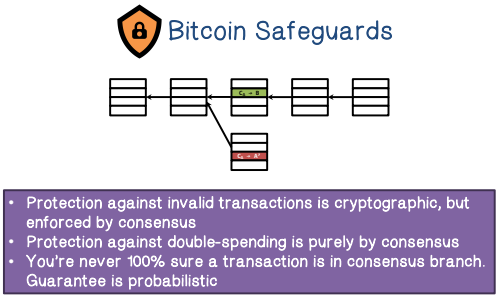
Now let us discuss the security of bitcoins. First, can Alice steal bitcoins that belonged to another user? This is not possible because there is no way Alice can create a valid signature that spends that coin. That is, as long as the underlying cryptography is solid, Alice cannot steal bitcoins.



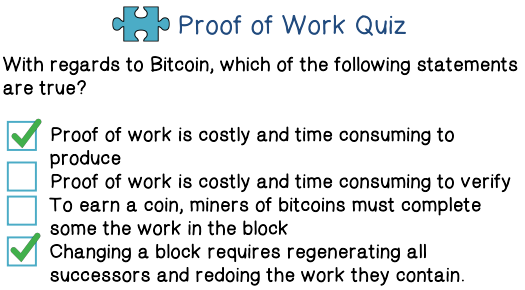
But can Alice launch a double spending attack? Suppose Alice creates two transactions. Here we use C subscript A to represent a coin owned by Alice. Suppose in the first transaction she sends Bob a bitcoin. In the second transaction, Alice double spends by sending the same bitcoin to another node that she controls. Of course, only one of these transactions can be included in the blockchain. If the blockchain get extended from the second transaction, then Bob will be denied this bitcoin.



Suppose Bob is the merchant. In order to protect himself from this attack, Bob should wait until the transaction has multiple confirmations. That is, there are multiple blocks extended from this block in the blockchain. The most common heuristics is to wait until you hear six confirmations.

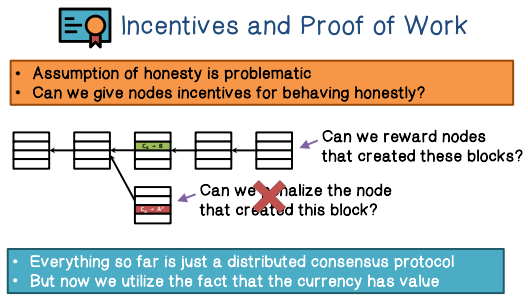


Let us summarize the safeguards in Bitcoin. The protection against invalid transactions is done through cryptography, but also enforced by consensus. Protection against double-spending is purely by consensus. There is no 100% guarantee that a transaction is in the consensus branch. That is, the guarantee is probabilistic. Again, the more confirmation you see, the higher probability that a transaction is valid.

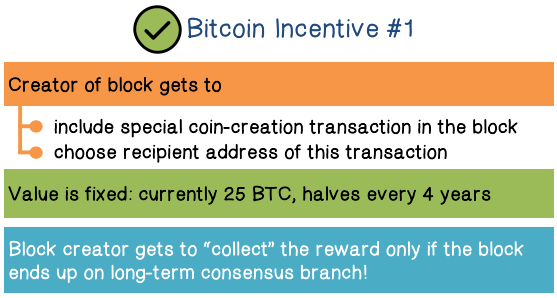


Now, let us do a quiz on proof of work. With regards to Bitcoin, which of the following statements are true?

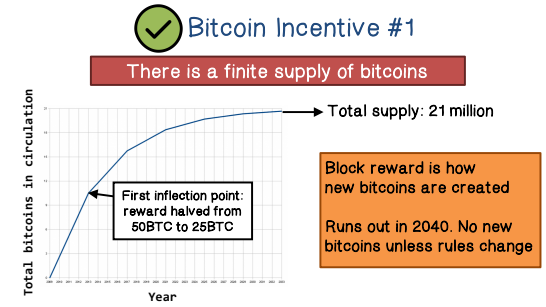
First, proof of work is costly and time consuming to produce, this is true. Second, proof of work is costly and time consuming to verify, this is false. Third, to earn a coin, miners of bitcoins must complete some of the work in the block, this is false because minor must complete all the work in a block. Fourth, changing a block requires degenerating all the successors and re-doing the work that they contain, this is true.



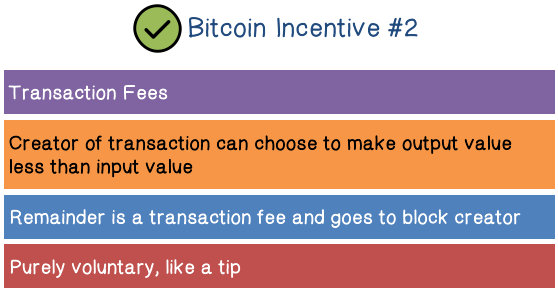
Recall in a double spending example, this is a valid transaction, and we want more confirmation of this block. The question is, can we reward nodes that extend from this transaction block? We cannot simply rely on honesty. In fact, we should use incentive to promote honesty. Since we are dealing with bitcoin which has value, we should be able to provide incentive to nodes that behave honestly.



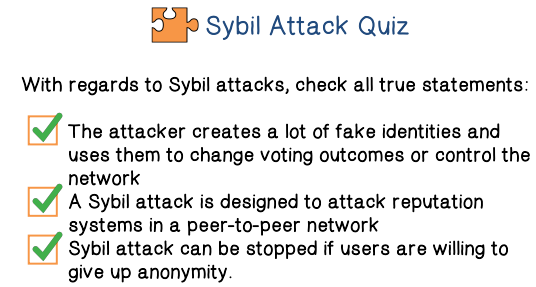
The first incentive is that, for a node to create a block, the node can create a special transaction and can choose the recipient of this transaction. Typically, that is itself. In particular, for this coin creation transaction, the value is fixed as, for example, 25 bitcoins. On the other hand, the block creator can collect this reward only if the block is accepted by the nodes. That means, this block ends up in the long-term consensus branch.



Again, block reward is how bitcoins are created, but there are a finite supply of bitcoins. Unless we change the rules, bitcoins can run out in the year 2040.

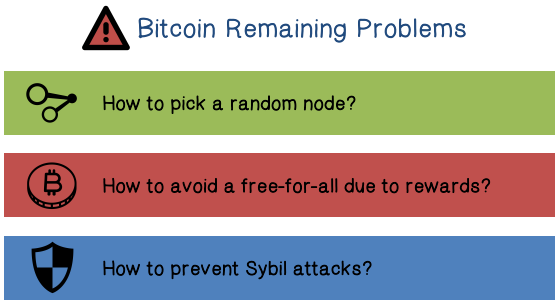


Another incentive that Bitcoin can provide is the so-called transaction fees. Suppose we have a lot of new Bitcoins, then we will now create a new block. How do we reward this node? Remember, a block contains a transaction(s) and therefore this block creator can collect transaction fees. That is, the creator of transactions can choose to make output value less than input value. The difference is the fee paid to the block creator. Currently, this is like a tip. But in the future, when Bitcoins run out, this may become monetary.

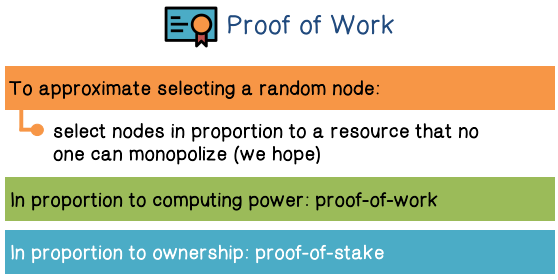


Before we discuss the remaining problems with Bitcoin, let us do a quick review of Sybil attack. Which statements are true?

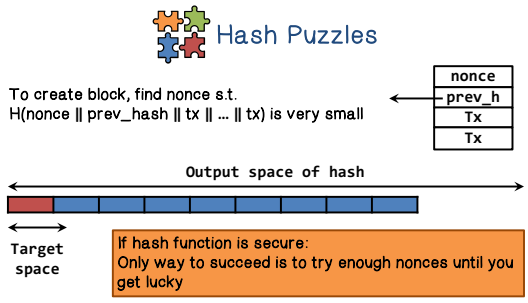
First, the attacker creates a lot of fake identities and use them to change voting outcomes and control the network. This is true. Second, a Sybil attack is designed to attack reputation system in a peer-to-peer network. This is also true. Third, Sybil attack can be stopped if users are willing to give up anonymity. This is also true.



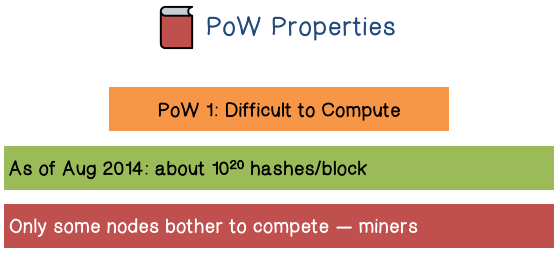
There are several remaining problems with Bitcoin. The first one is, how do we pick a random node? Second, the system can become unstable if everybody wants to run a Bitcoin code in order to capture some of these rewards. Third, how can we prevent an adversary from creating a large number of Sybil nodes to try to subvert the consensus protocol?



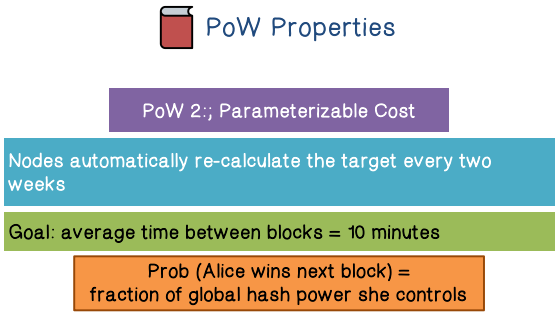
To select a random node, we can select nodes in proportion to resource that no one can monopolize. In particular, we can use proof-of-work to select a node that is in proportion to computing power. Alternatively, we can select nodes in proportion to ownership.



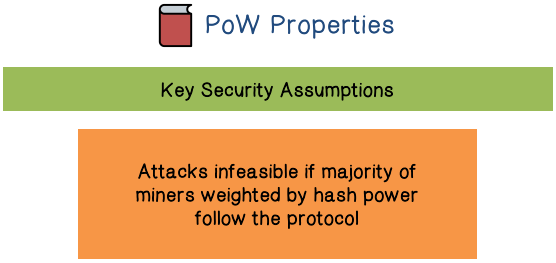
In Bitcoin, we use proof-of-work. To create a block, a no finds a nonce such that the hash of the nonce with a hash pointer to the previous block and all the transactions in this block is very small. Specifically, this hash value has to be smaller than a target value. With this proof-of-work, if Alice has 100 times more computing power than Bob, it doesn't mean that she always wins the race of finding this nonce. It means that she has about a 99 percent chance of winning. In the long run, Bob can still create one percent of the blocks.



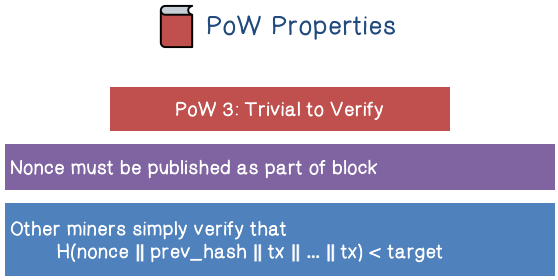
Because the target hash value is very small, it is very hard to find a correct nonce. In fact, only the nodes with a lot of computing power, and these are miners who would do this proof of work.



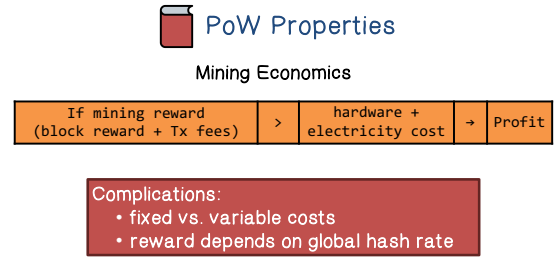
Another nice property is that we can adjust the cost of proof-of-work. For example, if the goal is to be able to create one block every 10 minutes, then the nodes can automatically recalculate the target space (and hence the difficulty of the puzzle) every two weeks. It is obvious that probability that Alice will win the next block depends on the fraction of global computing hash power she can choose.



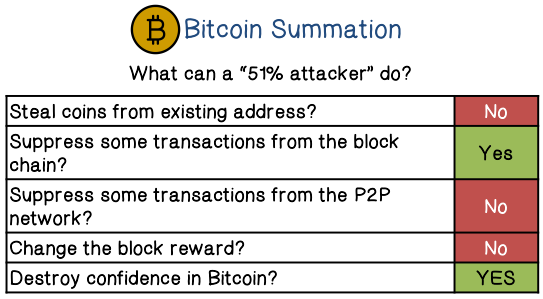
Therefore, the key security assumption is that attacks are infeasible if the majority of the miners weighted by hash power follow the protocol. In other words, the majority of the miners weighted by hash power are honest.



It is obvious that proof-of-work is very easy to verify. In particular, the nonce value has to be published as part of the block. Therefore, other miners can simply verify that the hash value is smaller than target hash value.



Obviously, proof-of-work can be used as incentive when the mining reward is greater than the mining cost.



To summarize this lesson, let us think about what a 51 percent attacker can do, that is, an attacker that controls 51 percent of the computing power. Can the attacker steal Bitcoins? No, because digital signatures can prevent this. Can the attacker suppress some transactions from the blockchain? Yes, because this attacker can refuse to extend from that block. Can the attacker suppress transactions from peer-to-peer network? No, because the transitions are broadcasted to peer-to-peer network. Can they change the block reward? No. That is determined by the whole system. Can the attacker destroy confidence in Bitcoin? Yes. For example, by refusing to extend from the valid block, then Bitcoin may become useless for valid transactions.