Blockchain

VICTOR BRUNELL

*CONTENT FROM **BITCOIN AND CRYPTOCURRENCY TECHNOLOGIES** (NARAYANAN, BONNEAU, FELTEN, MILLER, GOLDFEDER)*

Understanding Blockchains

A *Blockchain* is a decentralized, indelible, and append-only ledger for recording transactions.

To understand how the ledger is constructed and secured, it's important to understand *Cryptographic Hash Functions*.

To understand cryptographic hash functions, it's important to understand:

a) What a hash function is.
b) What properties of a hash function are required to make the function cryptographically secure.

What is a hash function?

A ***hash function*** is a mathematical function with the following three properties:

1. Its input can be any string of any size.
2. It produces a fixed size output. For example, an n-bit input and a 256-bit output.
3. It is efficiently computable.

Efficiently computable means that for a given input string, you can figure out what the output of the hash function is in a reasonable amount of time.

More technically, computing the hash of an *n*-bit string should have a running time that is O(*n*).

Basically, that means the computational complexity increases linearly with the number of bits.

Example hash function:

$$H(x) = x + 7$$

Cryptographically Secure

For a hash function to be cryptographically secure, it should satisfy the following three properties:

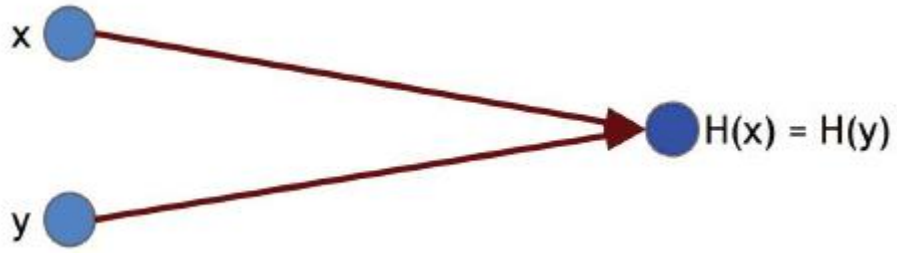
1. Collision-resistance
2. Hiding
3. Puzzle-friendliness

Collision-resistance

A collision occurs when two distinct inputs produce the same output. A hash function *H(.)* is collision-resistant if nobody can find a collision.

Collision-resistance. A hash function *H* is said to be collision resistant if it is infeasible to find two values, *x* and *y* , such that $x \neq y$, yet $H(x) = H(y)$.

A hash collision:



Basically, no two unique inputs to the hash function should produce the same output.

Hash Function Example

Here's a hash function that doesn't satisfy the collision-resistance property:

$$H(x) = x \% 2^{256}$$

This is a pretty simple and bad hash function. We can easily find two values that collide:

$$H(3) \text{ and } H(3 + 2^{256})$$

Both are hashed to the value 3. That is, there's a collision.

This is bad news if you want to ensure that your hash is unique and that nobody can generate it but you, for use as a publicly available address to a Bitcoin wallet for example.

Hiding

Hiding. A hash function H is hiding if when a secret value *r* is chosen from a probability distribution that has *high min-entropy*, then given *H(r || x)* it is infeasible to find *x* .

- || is the *concatenation* operation.
- *Min-entropy* refers to how predictable an outcome is. High min-entropy means the distribution is very spread out.

The hiding property asserts that:

- If we're given the output of the hash function *y = H(x)* , there's no feasible way to figure out what the input, *x*, was.

This is pretty handy if you want to broadcast information publicly and verifiably, but you want to control who can access the information.

Puzzle Friendliness

Puzzle friendliness. A hash function *H* is said to be puzzle-friendly if for every possible n-bit output value *y* , if k is chosen from a distribution with high min-entropy, then it is infeasible to find *x* such that H(k || x) = y in time significantly less than 2^n.

Basically, if someone wants to target the hash function to come out to some particular output value *y*, it's very difficult to do this if *k* is chosen in a random way.

You might have guessed that *k* represents the key that coin miners are trying to find to add a block to the chain and generate coins.

If k happens to be a 256-bit number, the number of keys to be searched is in a space with a size of 2^{256}. For comparison, the number of atoms in the known universe is 10^{78} to 10^{82}. So 2^{256} is a pretty big space to search.

Luckily, computers are great at just randomly guessing answers insanely fast and Bitcoin only requires finding a key that belongs to a subset of possible keys, not just one particular key.

What hash function does Bitcoin use?

SHA-256

SHA-256 uses a compression function that takes 768-bit input and produces 256-bit outputs. The block size is 512 bits.

The *compression function* is a fixed-length *collision-resistant* hash function.

Put simply, it's a compression function that takes an m-bit input and generates an n-bit output.

This is useful for transforming data into a fixed-length format, which can reduce the size/complexity of data structures, like a blockchain.

Applications of Crypto Secure Hash Functions

***Commitments*.** A commitment is the digital analog of taking a value, sealing it in an envelope, and putting that envelope out on the table where everyone can see it. You've committed to what's in the envelope, but the value remains a secret to everyone else. Only you can open the envelope.

Commitment scheme. A commitment scheme consists of two algorithms:

- **com := commit(*msg, nonce*)** The commit function takes a message and secret random value, called a nonce, as input and returns a commitment.
- **verify(*com, msg, nonce*)** The verify function takes a commitment, nonce, and message as input. It returns true if *com* == commit(*msg* , *nonce*) and false otherwise.

We require that the following two security properties hold:

- ***Hiding*** : Given *com*, it is infeasible to find *msg.*
- ***Binding*** : It is infeasible to find two pairs *(msg, nonce)* and *(msg', nonce')* such that *msg ≠ msg'* and commit(*msg, nonce*) == commit(*msg', nonce'*)

In cryptography, the term *nonce* is used to refer to a value that can only be used once.

Applications of Crypto Secure Hash Functions

Search puzzle. A mathematical problem which requires searching a very large space in order to find the solution. In particular, a search puzzle has no shortcuts. That is, there's no way to find a valid solution other than searching that large space.

A search puzzle consists of

- a hash function, *H*().
- a value, *id* (i.e., the ***puzzle-ID***), chosen from a high min-entropy distribution.
- and a target set *Y.*

A solution to this puzzle is a value, *x* , such that

- H(*id* || *x*) ∈ ***Y*** .

The intuition is this: if H has an n-bit output, then it can take any of 2^n values. Solving the puzzle requires finding an input so that the output falls within the set Y, which is typically much smaller than the set of all outputs. The size of Y determines how hard the puzzle is.

If a search puzzle is puzzle-friendly, this implies that ***there's no solving strategy for this puzzle which is much better than just trying random values of x***.

What do we have so far?

1. We have a method for generating fixed-length, unique outputs for a given set of inputs called a hash function.
2. We have a system for committing to information publicly in such a way that only we can access the information using hash functions and the cryptographic properties of *hiding* and *binding*.
3. We have a system for generating puzzles with adjustable difficulty where the only valid solution is discovered by testing random values from a set of values.

What are we missing?

Digital Signatures

A digital signature is supposed to be the digital analog to a handwritten signature on paper. We desire two properties from digital signatures that correspond well to the handwritten signature analogy.

Firstly, only you can make your signature, but anyone who sees it can verify that it’s valid.

Secondly, we want the signature to be tied to a particular document so that the signature cannot be used to indicate your agreement or endorsement of a different document.

We’ll use cryptographic hash functions to accomplish this.

Digital signature scheme

A digital signature scheme consists of the following three algorithms:

- **(sk, pk) := generateKeys(*keysize*)** The generateKeys method takes a key size and generates a key pair. The secret key *sk* is kept privately and used to sign messages. *pk* is the public verification key that you give to everybody. Anyone with this key can verify your signature.
- **sig := sign(*sk* , *message*)** The sign method takes a message and a secret key, *sk* , as input and outputs a signature for *message* under *sk.*
- **isValid := verify(*pk* , *message* , *sig*)** The verify method takes a message, a signature, and a public key as input. It returns a boolean value, *isValid* , that will be ***true*** if *sig* is a valid signature for *message* under public key *pk* , and ***false*** otherwise.

We require that the following two properties hold:

- *Valid signatures must verify*

  verify (*pk* , *message* , **sign** (*sk* , *message*)) == **true**

- Signatures are ***existentially unforgeable***

Public Keys as Identities

If you see a message with a signature that verifies correctly under a public key, *pk*, then you can think of this as *pk* is saying the message.

From this viewpoint, the public key is an identity. In order for someone to speak for the identity *pk*, they must know the corresponding secret key, *sk*.

A consequence of treating public keys as identities is that you can make a new identity whenever you want — you simply create a new fresh key pair, *sk* and *pk* , via the **generateKeys** operation in our digital signature scheme.

Public Keys as Identities

In practice, you may use the hash of *pk* as your identity since public keys are large.

Then in order to verify that a message comes from your identity, we have to check:

1. That *pk* indeed hashes to your identity, and
2. The message verifies under public key *pk*.

Moreover, by default, your public key *pk* will basically look random, and nobody will be able to uncover your real world identity by examining *pk*.

Of course, the NSA probably still has ways of identifying you, by network traffic analysis for example.

Decentralization and Bitcoin Addresses

The idea of generating new identities by creating public key / secure key pairs leads to the idea of decentralized identity management.

Rather than having a central authority that you have to go to in order to register as a user in a system, you can register as a user all by yourself.

You don’t need to be issued a username nor do you need to inform someone that you’re going to be using a particular name.

If you want to be somewhat anonymous for awhile, you can make a new identity, use it just for a little while, and then throw it away.

These identities are called ***addresses***, in Bitcoin jargon.

Constructing the Blockchain

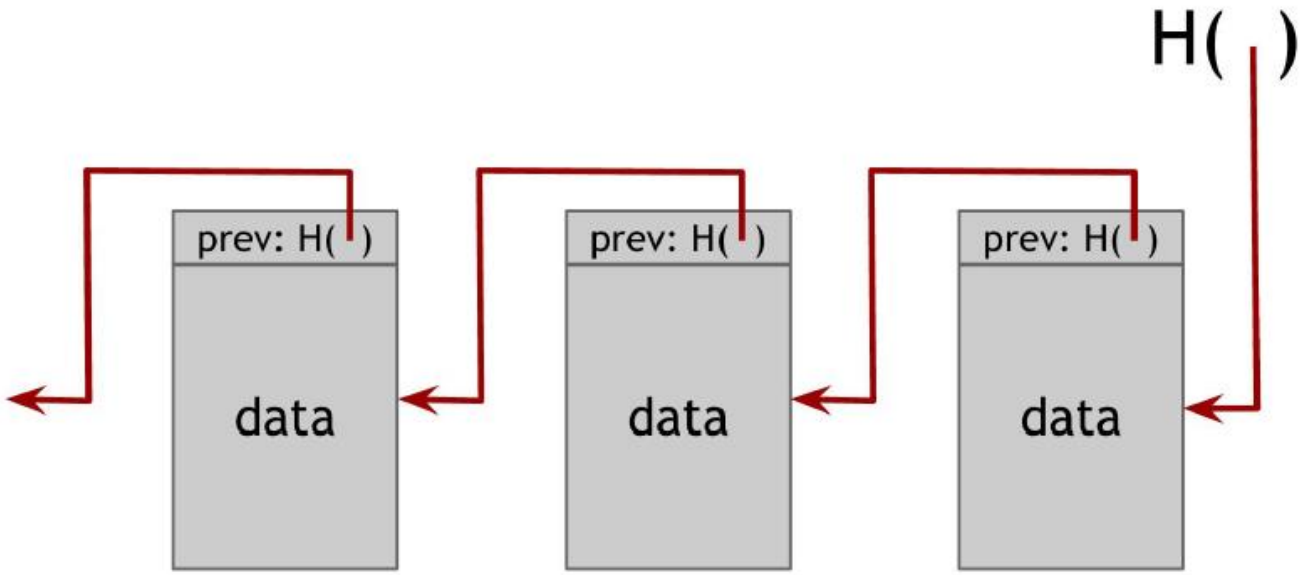
To build a block chain, we need some data structures. Like a linked list for example.

Recall that a linked list is just a chain of data structures. The data structures contain a pointer to the previous data structure and the next data structure. Hence, they are linked together in a list via pointers.

We'll also need to make use of pointers. In particular, we'll need something called a **hash pointer**.

Hash pointer. A hash pointer is a pointer to where data is stored together with a cryptographic hash of the value of that data at some fixed point in time.

Hash Pointers and Blockchains



A block chain is a linked list that is built with hash pointers instead of pointers.

Putting it together

The key idea is this:

If we have a data structure that is linked together by pointers, and those pointers also contain a hash of the data that is being pointed to (i.e., linked together) then we can verify that the data that has been committed to the chain is valid by checking the hash values.

The hash of the data in any data structure in the chain will be a unique value produced by our cryptographic hash function. If the data that was hashed is altered, then we will get a new unique hash value, because our hash function is collision-resistant.

Hence, we will know if anyone has tampered with our data chain. This makes the data *tamper-evident.* That is, if someone tries to change history, it'll be obvious and easily verifiable.

This is pretty important if we want to distribute said data to a bunch of users, but want to ensure that everyone is working with the same history of data that has been added to the chain.

Identifying a Blockchain

So how does Bitcoin create a blockchain and differentiate it from the Litecoin blockchain?

1. Bitcoin generates the initial block, called the genesis block.
2. Bitcoin signs the genesis block.
3. Bitcoin publishes the public signature.

Now anyone can verify that they are on the Bitcoin blockchain by looking at the first block.

Furthermore, since the signature is part of the data that is hashed for the hash pointer in subsequent blocks, ***the values of all other hashes in the chain will depend on the hash in the genesis block***!

Hence, any tampering in the chain will be easily identified.

Blockchains in Practice

In practice, linked lists are a viable but inefficient means of constructing a blockchain.

This is because searching the blockchain to verify transactions would happen in O(n) time because we have to search through the list one node at a time.

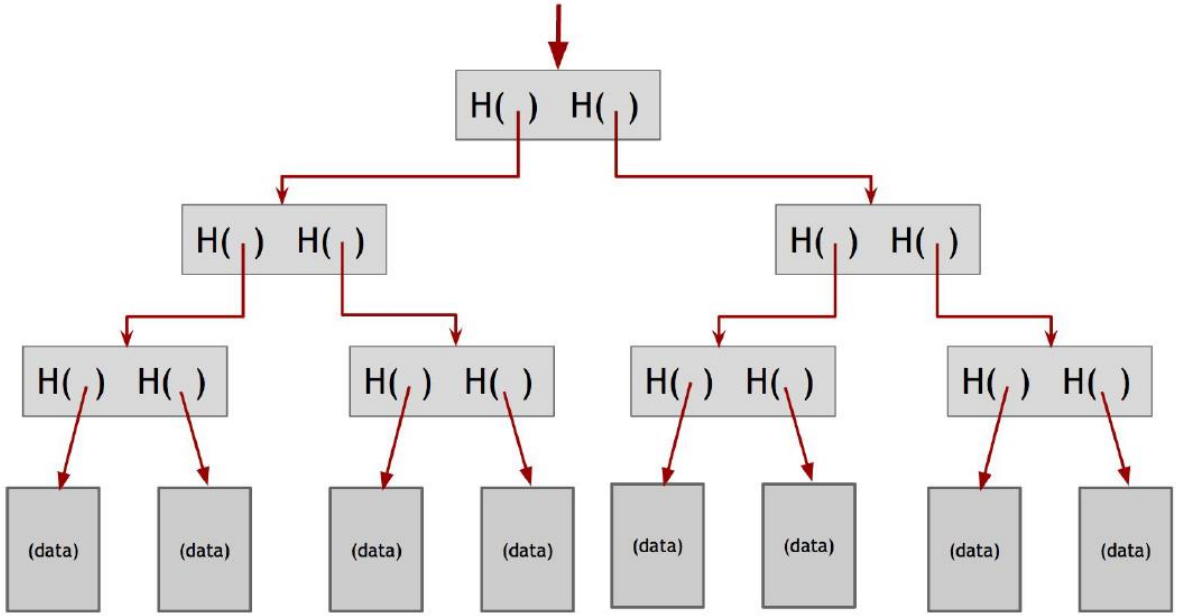
A better data structure for performing searches is a binary tree, because a binary tree can be searched in log(n) time.

This is pretty important, because we're going to have a lot of users verifying a lot of transactions. It should be as efficient as possible.

The type of binary tree Bitcoin employs on its blockchain is called a **Merkle Tree**.

Merkle Trees (i.e., fancy binary trees)

- In a Merkle tree, data blocks are grouped in pairs and the hash of each of these blocks is stored in a parent node.
- The parent nodes are in turn grouped in pairs and their hashes stored one level up the tree.
- This continues all the way up the tree until we reach the root node.



Merkle Trees

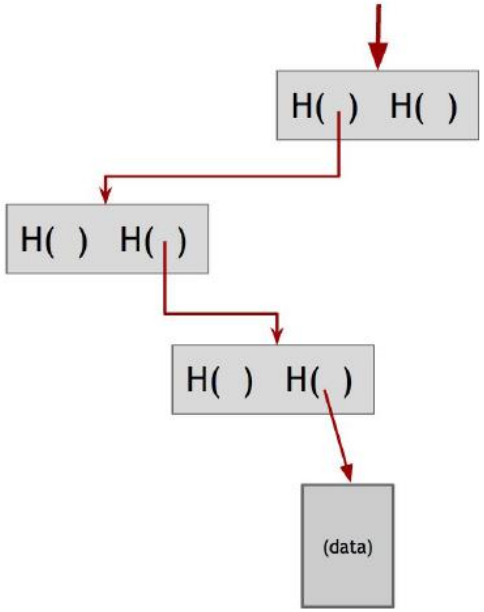
By sorting a Merkle Tree, we arrange the data nodes at the leaves of the tree so that we can quickly identify the position of data in the tree.

This allows us to quickly verify the data as well, because we don't have to traverse an entire linked list to the root of the tree. Instead, we can just follow the path on the tree to the root.

We have to verify far fewer nodes using the Merkle Tree.

Just follow the path from the data to the root of the sorted tree.

By doing this we avoid a lot of node verification.



Bitcoin Transactions

What is a Bitcoin transaction?

A transaction is a data structure that contains:

1. A digital signature.
2. An instruction to pay to a public key.
3. A hash.

The hash represents a pointer to a previous transaction output that a user received and is now spending.

That pointer must reference a transaction that was included in some previous block in the blockchain.

Note that there are two different hash pointers being used:

- Blocks include a hash pointer to the previous block that they're extending.
- Transactions include one or more hash pointers to previous transactions outputs that are being redeemed.

Who verifies transactions?

We have all these transactions being generated, signed, and submitted to be included in the blockchain.

To be included in the blockchain, they need to be verified against the current state of the blockchain. This takes time and, more importantly, computing power.

Bitcoin Miners are the ones performing the verification, in addition to extending the blockchain by proposing new blocks.

These new blocks contain not one, but many verified transactions.

The number of transactions that can be included in a block is limited by the block size, which is 1MB for Bitcoin.

Bitcoin Miners

In addition to verifying transactions, miners also perform a much more computationally intensive task: hash puzzle solving.

Recall that a search puzzle that is puzzle-friendly can only be solved by randomly testing solutions. In a very large test space, this can take a lot of time and energy. The puzzle the miners are working on takes the form:

$$H(nonce \,||\, prev\_hash \,||\, tx \,||\, tx \,||\, \ldots \,||\, tx) < target$$

This is employed by the blockchain in order to determine who gets to propose the next block in the chain to the peer-to-peer network.

Of course, this block must also be verified, to ensure there is no tampering with the blockchain history going on.

This can easily be done by checking the hashes in the tree.

The Mining Process

Put simply, the mining process is as follows:

- A mining node receives a number of transactions broadcast to the network.
- The mining node verifies the transactions against the current blockchain (i.e., the longest verifiable chain in use by the network).
- The node groups the verified transactions into a block, and includes its digital signature, and the hash of the data in the block it is extending.
- The node then has to solve the current network search puzzle to find a nonce (some value) that hashes to a target range defined by the puzzle in order to propose its block.

This equates to searching a very large space for some random value that can fall within the target range.

Since all miners are working on solving the puzzle, and any miner might find the answer, this is in effect a way to randomly select which node proposes the next block, so no node is guaranteed to extend the chain.

This is important, because the longest verified chain is the one other nodes will start to build on. By ensuring a random node proposes the next block, no one node has power over the chain. It is decentralized.

Why mine?

A mining node that generates a block is rewarded in Bitcoin.

The amount of Bitcoin mined is predetermined based on an algorithm built in to the blockchain.

Basically, there is a transaction that a new block can include. The transaction generates coins and sends them to an address, usually the miner's address.

The value of the block reward is determined by the total number of mined blocks and it is halved every 210,000 blocks. This is about every four years.

For any given miner, the mean time to find the next block is given by:

$$mean\ time\ to\ next\ block\ = \frac{10\ minutes}{fraction\ of\ hash\ power}$$

This is important, because it means that no matter how little power a miner is using, they still have a non-zero chance of finding a block and getting a reward. This prevents fully centralized control of the mining process.

Why mine?

Furthermore, a miner can collect transaction fees for each transaction it verifies and includes.

Of course, the person generating the transaction fee doesn't have to include a fee for the miner, but on the flip side, a miner doesn't have to include a transaction that doesn't offer a transaction fee.

This doesn't mean the transaction won't make it onto the chain, because other nodes might not check for this, but it means the transaction might enter the chain slower than it would have otherwise.

For this reason, most exchanges and wallet software auto-include a fee.

One problem that arises from this is that as the network grows and transactions compete for inclusion into mined blocks, the transaction fees tend to increase, because large mining operations will include transactions that pay higher fees.

Summary

Cryptographic hash functions allow us to generate unique values.

These values can be used in search puzzles, key generation, digital signatures, and verification.

The blockchain is a big data structure that makes use of hash pointers to create an append-only and indelible ledger.

New blocks holding transactions can be added to the blockchain (i.e., ledger) by verifying the transactions and solving a search puzzle.

The miners are the ones solving these puzzles and performing the verifications.

They are rewarded by the blockchain with the ability to include a transaction of their own that generates an algorithmically determined number of Bitcoins and collect transaction fees.

You've probably already thought of a bunch of ways you might try to subvert the system. There is only one attack that we really have to worry about though, for reasons I can go into, if you want to get into the weeds on this stuff…

More about Bitcoin

Consensus Protocols

- How does the network come to a consensus on which chain to use?

Security

- How do we prevent double-spends and forgeries?
- What is a 51% attack?

Scripts

- How do Bitcoin scripts work?

Altcoins

- What is the difference between Bitcoin, Litecoin, and Ethereum?

Incentives

- How will the incentive to verify blocks change when no more Bitcoins can be mined?

Maybe in the next presentation. ;)