C2. BLOCKCHAIN AND CRYPTOGRAPHY BASICS

Today everyone has heard of Blockchain but not everybody understands how it works. This section will shows you that Blockchain certainly isn't magic and in fact it is only mathematics.

To briefly define what the ***blockchain*** is we can say that "*a blockchain is a ledger that uses cryptography and incentives to record transactions in a tamper-evident way.* This allows for trust-minimized transactions between pseudonymous parties without requiring a trusted intermediary. It is worth mentioning here as the first blockchain was conceptualized by an individual (or group of individuals) known as Satoshi Nakamoto in 2008.

At its most basic level, blockchain is literally just a chain of *blocks*, but not in the traditional sense of those words: *blocks* on the blockchain are made up of digital pieces of information and specifically, they have three parts:

- Blocks store information about transactions (date, time, sum, etc.);
- Blocks store information about who is participating in transactions but purchase is recorded without any identifying information using a unique *digital signature* (like a username).
- Blocks store information that distinguishes them from other blocks. Each block stores a unique code called a *hash* that allows us to tell it apart from every other block. Hashes are cryptographic codes created by special algorithms.

So, because blockchain technologies make heavily use of cryptography to achieve a consensus, in order to understand the key principals behind the protocols, we need to explore first some concepts such as: *asymmetric encryption*, *hash functions* and *digital signatures*.

Until the 1970s, all publicly known encryption schemes were symmetric: the recipient of an encrypted message would use the same secret key to unscramble the message that the sender had used to scramble it. But that all changed with the invention of asymmetric encryption schemes in which the key to decrypt a message (the *private key*) was different from the key needed to encrypt it (known as the *public key*) and there was no practical way for someone who only had the public key to figure out the private key.

This meant you could publish your public key widely, allowing anyone to use it to encrypt a message that only you could decrypt. This breakthrough transformed the field of cryptography because it became possible for any two people to communicate securely over an unsecured channel without establishing a shared secret first.

Asymmetric encryption also had another groundbreaking application: *digital signatures*. In normal public-key cryptography, a sender encrypts a message with the recipient's public key and then the recipient decrypts it with her private key. But you can also flip this around: have the sender encrypt a message with his own private key and the recipient decrypt it with the sender's public key. That doesn't protect the secrecy of the message (since anyone can get the public key). Instead, it provides cryptographic proof that the message was created by the owner of the private key. Anyone who has the public key can verify the proof without knowing the private key.

Since the 1980s many people dreamed of using digital signatures to build an electronic cash system that's fully decentralized. But there were two big issues: how to introduce new coins into the system and how to solve the double-spending problem. In 2008, Satoshi Nakamoto invented a shared ledger called **the blockchain** that is maintained by computers, called nodes, operating on a peer-to-peer network. Thousands of computers around the world keep separate copies of the entire blockchain, storing every transaction that has happened since the network was launched in 2009.

Some of the nodes are miners that participate in the process of actually updating the blockchain. A miner makes a list of all the transactions it has heard about that aren't already in the blockchain; it checks to make sure that each transaction follows all of the rules of bitcoin. The resulting list of new, valid transactions is called a block. The miner also adds a special transaction granting itself a fixed reward—currently 12.5 bitcoins—for creating the block. To win the right to add the next block, bitcoin miners compete against each other by performing a highly repetitive computation. They add a random value called a *nonce* to the candidate block they have assembled. Then they apply the *SHA-256 hash function*, which produces a short, seemingly random string of 1s and 0s that serves as a cryptographic fingerprint for the block.

Hash Functions

A hash function is a mathematical function that lets you encrypt data. In simple terms, hashing means taking an input string of any length and giving out an output of a fixed length. In the context of cryptocurrencies (like Bitcoin), the transactions are taken as input and run through a hashing algorithm (Bitcoin uses SHA-256) which gives an output of a fixed length.

A *cryptographic hash function* is a special class of hash functions that has various properties making it ideal for cryptography. There are certain properties that a cryptographic hash function needs to have in order to be considered secure:

- ✓ **Deterministic:** no matter how many times you parse through a particular input through a hash function you will always get the same result. This is critical because if you get different hashes every single time it will be impossible to keep track of the input);
- ✓ **Quick Computation:** the hash function should be capable of returning the hash of input quickly to be efficient;
- ✓ **Pre-Image Resistance:** given H(A) it is *infeasible* to determine A, where A is the input and H(A) is the output hash. We already know that it is not impossible to determine the original input from its hash value since the hash functions are deterministic and the hash of a particular input will always be the same, so you can simply compare the hashes and find out the original input but this only works when the given amount of data is very less.
- ✓ **Small Changes In The Input Changes the Hash;**
- ✓ **Collision Resistance:** when two different inputs produce the same output. A hash function is considered to be *collision resistant* if nobody can find a collision. In theory, we know they exist because the number of inputs is infinite, but the number of outputs is finite (there are only so many ways to arrange 64 characters);
- ✓ **Puzzle Friendly:** if someone wanted to generate a hash that came up with the same output it's extremely difficult to find another value that exactly hits this target.

Examples of cryptographic hash functions:

- MD 5: It produces a 128-bit hash. Collision resistance was broken after ~2^21 hashes.
- SHA 1: Produces a 160-bit hash. Collision resistance broke after ~2^61 hashes.
- SHA 256: Produces a 256-bit hash. This is currently being used by Bitcoin.
- Keccak-256: Produces a 256-bit hash and is currently used by Ethereum.

Web Resources for hash functions:

http://www.unit-conversion.info/texttools/category/Hash_and_Encryption#data

Elliptic Curve Digital Signature Algorithm

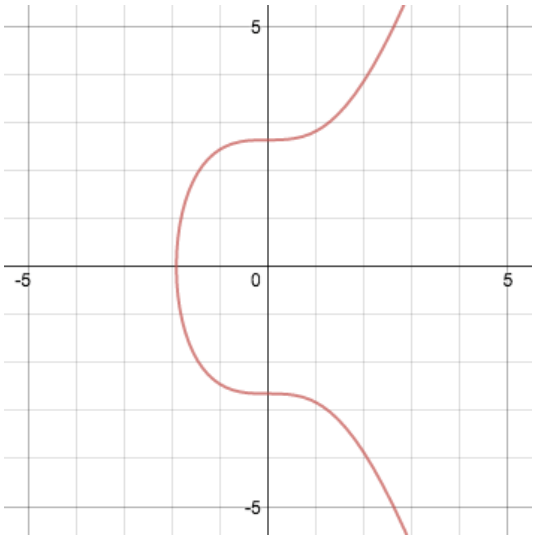
We saw earlier that bitcoin themselves are not stored either centrally or locally but they exist as records on the blockchain, copies of which are shared by a volunteer network of connected computers. To "own" a bitcoin simply means having the ability to transfer control of it to someone else by creating a record of the transfer in the blockchain. But what grants this ability? Access to an ECDSA (short for Elliptic Curve Digital Signature Algorithm) private and public key pair. It's a process that uses an *elliptic curve* and a *finite field* to "sign" data in such a way that third parties can verify the authenticity of the signature while the signer retains the exclusive ability to create the signature.

ECDSA has separate procedures for signing and verification. Each procedure is an algorithm composed of a few arithmetic operations. The signing algorithm makes use of the private key, and the verification process makes use of the public key. But first let's define the elliptic curves and the notions of finite fields.

An **elliptic curve** is represented algebraically as an equation of the form:

$$y^2 = x^3 + ax + b$$

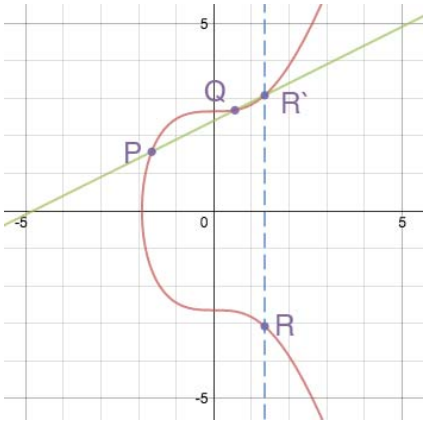
Bitcoin use a = 0 and b = 7 and the graphic looks like this:



Source: https://www.coindesk.com/wp-content/uploads/2014/10/elliptic-curves.png

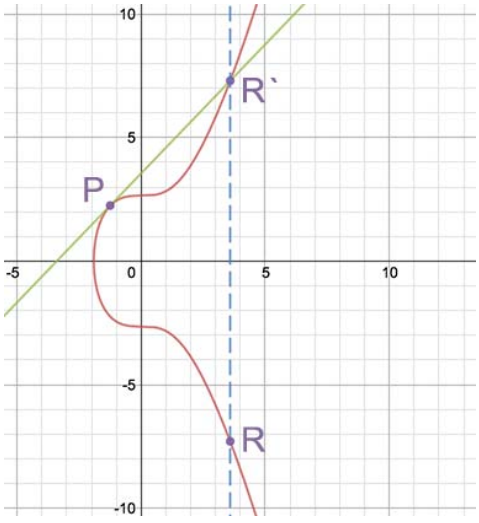
Elliptic curves have useful properties. For example, a non-vertical line intersecting two non-tangent points on the curve will always intersect a third point on the curve. A further property is that a non-vertical line tangent to the curve at one point will intersect precisely one other point on the curve. These properties can be used to define two operations: *point addition* and *point doubling*.

Point addition, $P + Q = R$, is defined as the reflection through the x-axis of the third intersecting point $R`$ on a line that includes P and Q.



Source: https://www.coindesk.com/wp-content/uploads/2014/10/point-addition.png

Similarly, **point doubling**, P + P = R is defined by finding the line tangent to the point to be doubled, P, and taking reflection through the x-axis of the intersecting point R` on the curve to get R. Here's an example of what that would look like:

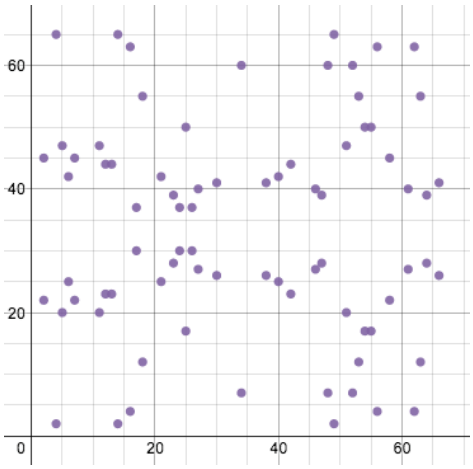


Source: https://www.coindesk.com/wp-content/uploads/2014/10/point-doubling.png

Together, these two operations are used for scalar multiplication, R = a P, defined by adding the point P to itself a times. For example: R = 7P; R = P + (P + (P + (P + (P + (P + P))))).
The process of scalar multiplication is normally simplified by using a combination of point addition and point doubling operations. For example, 7P can be broken down into two point doubling steps and two point addition steps:

R = 7P; R = P + 6P; R = P + 2 (3P); R = P + 2 (P + 2P)

A **finite field**, in the context of ECDSA, can be thought of as a predefined range of positive numbers within which every calculation must fall. Any number outside this range "wraps around" so as to fall within the range. The simplest way to think about this is calculating remainders, as represented by the modulus (mod) operator. For example, 9/7 gives 1 with a remainder of 2. **ECDSA uses elliptic curves in the context of a finite field**, which greatly changes their appearance but not their underlying equations or special properties. The same equation plotted above, in a finite field of modulo 67, looks like this:



Source: https://www.coindesk.com/wp-content/uploads/2014/10/putting-it-together.png

ECDSA and Bitcoin

Bitcoin protocol selects a set of parameters for the elliptic curve and its finite field representation that is fixed for all users of the protocol. The parameters include the *equation* used, the *prime modulo* of the field, and a *base point* that falls on the curve. The *order* of the base point, which is not independently selected but is a function of the other parameters, can be thought of graphically as the number of times the point can be added to itself until its slope is infinite, or a vertical line. The base point is selected such that the order is a large prime number.

Bitcoin uses very large numbers for its base point, prime modulo, and order. In fact, all practical applications of ECDSA use enormous values. The security of the algorithm relies on these values being large, and therefore impractical to brute force or reverse engineer.

In the case of bitcoin:

- Elliptic curve equation: $y^2 = x^3 + 7$
- Prime modulo = $2^{256}-2^{32} - 2^9-2^8-2^7 - 2^6 - 2^4 - 1$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
- Base point = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
- Order = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

How private and public keys are related with ECDSA?

In ECDSA, the private key is an unpredictably chosen number between 1 and the order. The public key is derived from the private key by scalar multiplication of the base point a number of times equal to the value of the private key. Expressed as an equation:

*public key = private key * base point*

This shows that the maximum possible number of private keys (and thus bitcoin addresses) is equal to the order.

References:

1. Katz, J., Lindell, Y., *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2008
2. https://blockgeeks.com/guides/what-is-hashing/
3. https://www.coindesk.com/math-behind-bitcoin
4. https://www.investopedia.com/terms/b/blockchain.asp
5. http://www.michaelnielsen.org/ddi/how-the-bitcoin-protocol-actually-works/
6. https://crushcrypto.com/cryptography-in-blockchain/
7. https://arstechnica.com/tech-policy/2017/12/how-bitcoin-works/