Blockchain and Bitcoin for Mathematicians

Simon Xiang

February 25, 2021

blockchain... a mathy introduction to cryptocurrency to help me understand, because reading is hard for a mathematician

reference used: https://nato.li/blog/bitcoin-for-mathematicians-distilling-the-problem

Why?

The reason we want decentralized currency is because the government has and will continue to spy on us. When you make a transaction, the arbiter decides whether or not it looks right, to avoid double spending. Decentralizing currency means that the people decide whether a transaction is correct rather than big bro, and toppling this requires 51% of the world's processing power. You also no longer have to rely on the bank not to loan out all your money or lose it (see, 2008).

We have some goals for a good new currency:

1. needs to be anonymous
2. avoids double spending
3. has non reversible payments

Preliminary definitions

Definition 1.1. Informally, a **computable** function is a map whose outputs can be obtained by applying an algorithm. That too, is hard to define.

Definition 1.2. A **one-way function** is a map $f : X \to Y$ for X, Y sets that is computable in polynomial time, but for every probabalistic polynomial time map $F : Y \to X$, the probability that F is a right inverse for f is negligible[1].

[1] This isn't too important, but a **negligible function** $\mu : \mathbb{N} \to \mathbb{R}$ satisfies the property that for every positive integer c, there exists an integer N_c such that for all $x > N_c$, $|\mu(x)| < 1/x^c$.

There are some things that are hand wavy here, but you should be able to get the general idea, which is that one-way functions can be computed but are hard to find inverses for (eg integer factorization, finding discrete logarithms). However, it isn't proven that these functions are one-way, or any exist at all: that would imply $P \neq NP$.

Definition 1.3. A (cryptographic) **hash function** is a one-way function $f : X \to Y$ where all the outputs are of fixed size, that is, Y is finite. We call a member of the image of f a **hash**.

For example, the outputs of $f : \mathbb{N} \to \{0, \cdots, 2^{256} - 1\}$ are 256 bits long. sha_{256} is an example of such a hash function.

Definition 1.4. A **trapdoor one-way function** is a one-way function $f : X \times K \to Y$ such that for any $k \in K$, there exists a corresponding "trapdoor" $t_k \in K$ and probabalistic polynomial time map $F : K \times Y \times \{t_k\} \to Y$ such that for all $x \in X$, we have $F(k, f(x,k), t_k) = x'$, $f(x', k) = f(x, k)$ (or $x' = x$).

You can see what this function does by the name and definition. Let X, K be the set of prime numbers, $Y = \mathbb{N}$, and $f(x,k) = xk$. We have f one-way since presumably you cannot factor xk for x, k large primes in a sufficient amount of time. In this case, $t_k = k$ is the trapdoor, and $F(k, xk, t_k) = (xk)/k = x$.

Example 1.1 (Public-key cryptography)**.** The most common use of trapdoor one-way functions is in public-key cryptography. If Alice wants to send Bob a secret message m for his eyes only, Bob generates the keypair (k, t_k), and sends over the *public key* k and the function $f(\cdot, k)$. Then Alice sends $f(m, k)$ over (assuming this can be read publicly), and only Bob can decrypt the message since he has the private key t_k.

Note that we assume the private key is not polynomial-time computable from k: the previous example doesn't work here. To make RSA work with integer factorization, see the details here: https://simonxiang.xyz/blog/proving-rsa-encryption-an-application-of-group-theory-part-1-asymmetric-encryption.

Definition 1.5. A **digital signature scheme** is a collection of pairs of functions $(s_k, v_{k'})$ indexed by keypairs (k, k'), together with the **signing function** $s_k : X \to Y$ and **verification function** $v_{k'} : X \times Y \to \{T, F\}$[2]. The signing and verification functions satisfy the following conditions:

1. s_k and $v_{k'}$ are polynomial-time computable,
2. $\Pr(v_{k'}(x, s_k(x)) = T) = 1$,
3. For every probabalistic polynomial-time algorithm with input k' and an oracle[3] to access s_k, the probability of outputting some pair (x, y) such that $v_{k'}(x, y) = T$ is negligible.

Again, see here for more detail on signing and decrypting signatures. Suppose Alice has a keypair (k, k') where k' is the public key. If Bob wants to check that a message x from Alice is legitimate, Alice attaches the signature $s_k(x)$ to the message. Once Bob confirms that $v_{k'}(x, s_k(x)) = T$, then he is pretty much sure the message is from Alice, and if it equals F, then it is most likely a forgery.

We will not talk about ecdsa and elliptic curve cryptography here. If you want to talk about ECC, take a course in number theory and read the literature.

Simple transactions

A transaction has at least one input and one output. Say Bob has received b bitcoins from Alice and wants to send b bitcoins to Carol. To accomplish this, Bob constructs a transaction that relates his money received from Alice (input) to Carol (output). Bob needs to know where to send the payment, so Carol generates a bitcoin **address** a by selection a private key $k \in \{0, \cdots, 2^{256} - 1\}$ and computing

$$a = (e \circ \text{sha}_{256} \circ f)(k),$$

where f is a one-way function and e adds extra information. The address somewhat functions like the public key $f(k)$ corresponding to k. After Carol sends her address to Bob, he can construct the output of his transaction,

[2] The set $\{T, F\}$ represents the conditions True, False.

[3] Here, oracle just means we have some abstract machine that solves a problem. See here for more info: https://en.wikipedia.org/wiki/Oracle_machine

namely the pair (b, a). Since only Carol has the private key to a, she can spend the b bitcoin later on. More precisely, the output component b is sent with instructions that only whoever has the private key associated with $\text{sha}_{256}(f(k))$ (which Bob gets by inverting e) can spend money from this transaction.

The input of Bob's transaction is the **transaction ID** t of the transaction corresponding to Alice's payment, that is, t represents the sha_{256} hash of the transaction. Then Bob signs the transaction data, or the tuple $(t, (b, a))$ with the private key from his transaction with Alice[4]. The signature has two purposes: to ensure the payment is not a scam, and to ensure that Bob is indeed the owner of the money sent by Alice.

If Carol wants to send b bitcoins to David, she requests an address a' from David and constructs the transaction $(\text{sha}_{256}(t, (b, a)), (b, a'))$, where the first component of the pair is the transaction ID of her transaction with Bob. Then she signs it with her private key k associated with the address a of Bob's payment, "unlocking" the money she received from Bob.

Transactions with multiple inputs and outputs

[4]Note that we never explicitly defined what function we're signing with and all of that. In reality, for Bitcoin elliptic curves are working behind the scene, but if you understand the essence of encryption and signatures and stuff, we can set f to be an arbitrary one-way function and things will work out.