Web3 Cohort by 100xDevs

Goal

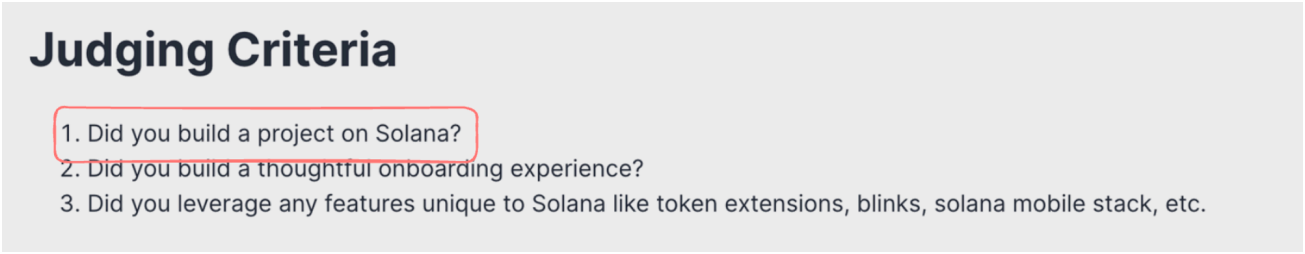
To create a Cohort of people who are great at Blockchains, Web3.

My background in Web3

Cohort 3.0 Exclusive Hackathon

Link - https://earn.superteam.fun/listings/project/100xdevs-solana-mini-hackathon-1/

We're doing an exclusive hackathon with $100 prize for the top 50 submissions



Focus on UX. Have a live link deployed.

Project ideas

1. Web3 Zapier - https://build.superteam.fun/ideas/automated-workflows
2. Whale Alert - https://x.com/cyversalerts/status/1813834131165286464
3. Liquidating an NFT to a token - https://build.superteam.fun/ideas/liquidating-an-nft-to-any-token
4. NFT viewing gallery (maybe in 3D) - ▸
5. Decentralized Fiver - End to End job portal to hire solana devs, pay them, escrow money from the job provider.
6. RPC aggregator - Let a user put in a bunch of RPCs from various providers (Helius, Alchemy, QuickNode) and you should figure out which one to forward requests to (Similar to https://www.ironforge.cloud/)
7. Wallet adapter for a web based wallet - ▸
8. Youtube channel opinions market - Let people trade on a coin associated to a Youtube channel. Creator can come and collect royalties by connecting their YT account. https://www.youtube.com/watch?v=PZNgcH2Jtac
9. Tiplink (even tho we're building it saparately, if you want to build a better version with a twist, you should do it)
10. Github Bounty Dispenser. Make users give their Adhar/Pan. Make users link their github with their wallet address. Allow maintainers to approve bounties. Create a dashboard where

you can track profiles of users/companies and a leaderboard of contributors based on bounty earned

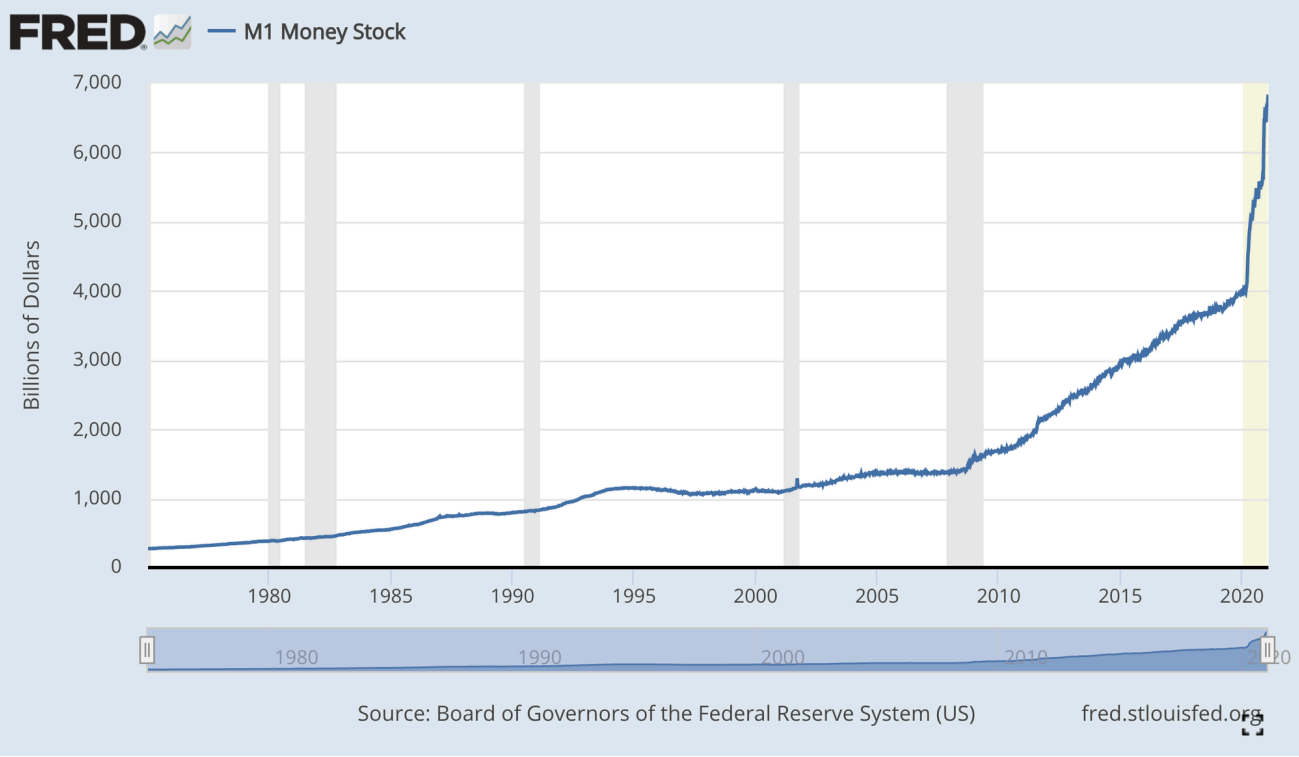
11. UI Library for Solana - NFTCard, TokenCard, SwapCard

Why blockchains?

Inflating currencies

Government has been printing currencies left right and center. This leads to increasing inflation, price of everything goes up.

Holding on to cash is a losers bet in the long run. Holding on to any asset (Gold, Stock, real estate) is better compared to currencies like USD, INR.



Fractional reserve Banking

Banks dont have your money. They lend out most of it.

If there is a bank run (everyone goes to the bank to withdraw their money), banks wont be able to pay everyone

Silicon valley collapsed in 2022. I was in the US when it happened. Most YC companies had their funds in SVB. They were bailed out, but if not, you would've seen a lot of startups die.

Bailouts

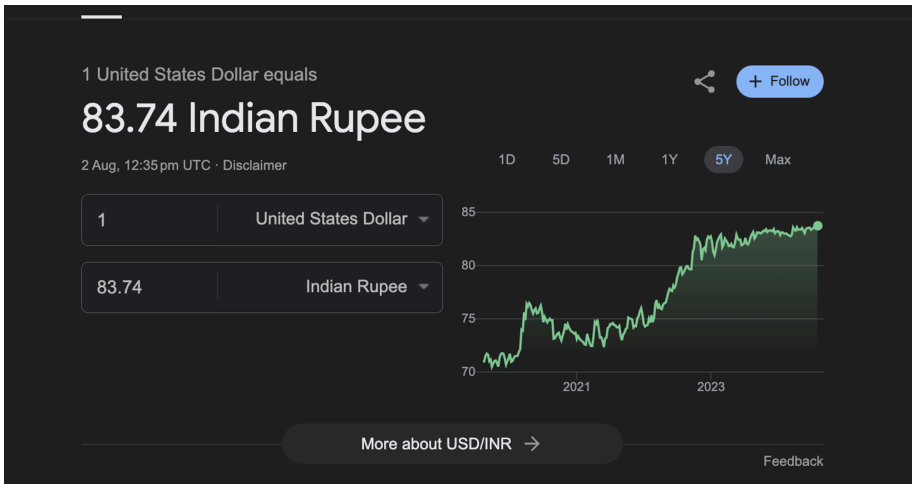
The 2008 Financial crisis was triggered by a financial instrument called mortgage-backed securities.

Even though the banks at Wall Street were at fault, the government ended up bailing them out using Taxpayer money.

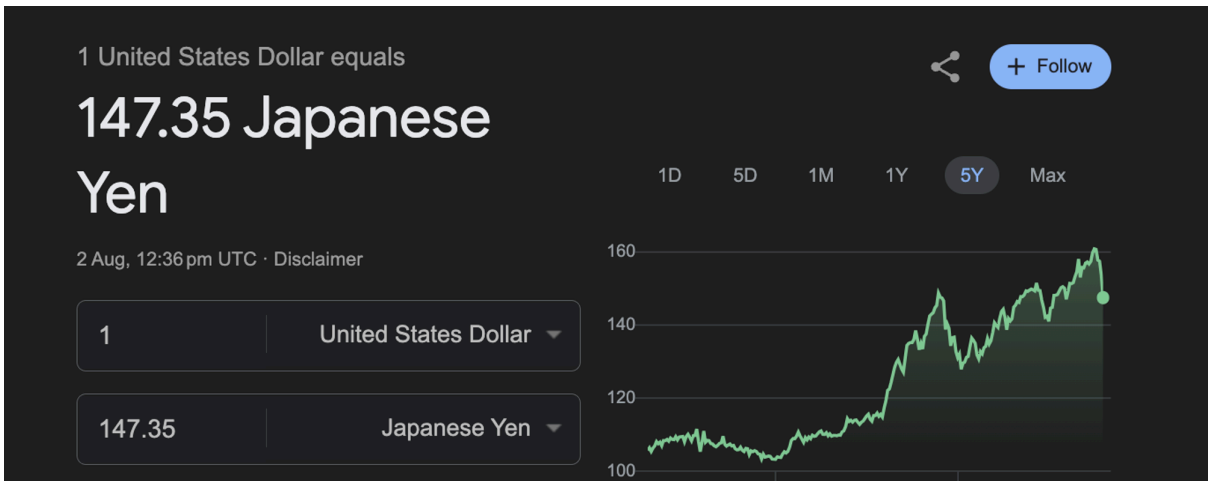
💡 A great movie to look at is **Big Short.** I've watched it ~5 times

INR Depreciation (even worse in countries like Japan)

1. USD



1. JPY

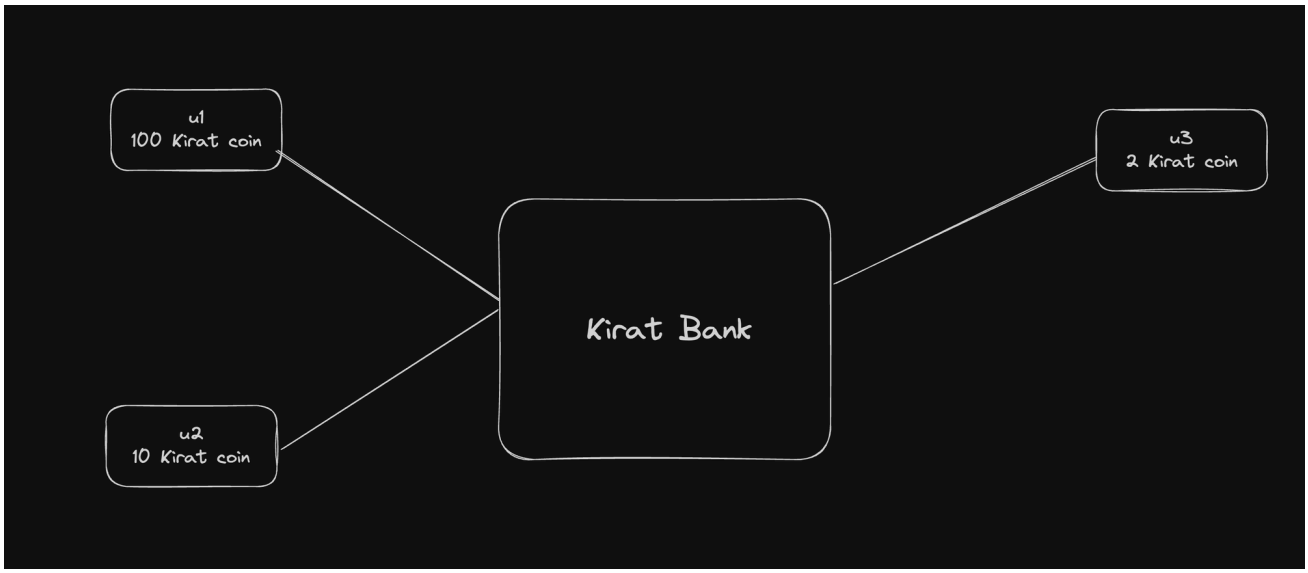


How to create a new currency?

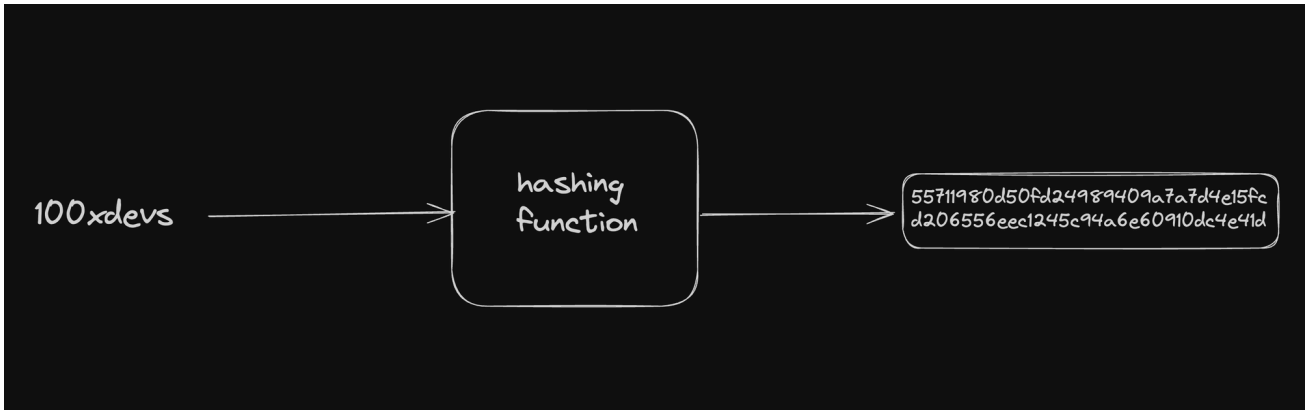
Right now, currencies can only be issued by central governments. You can't create your own `Kirat coin` and ask users to use it.

Even if I do issue a `Kirat coin` , no one would use it, and for good reasons -

1. I can print any number of Kirat coins, making myself richer
2. I become the central mint and verification athority for the coin.
3. No one would (or should) trust me



Intro to hashing



Hashing is a process that transforms input data (of any size) into a fixed-size string of characters.

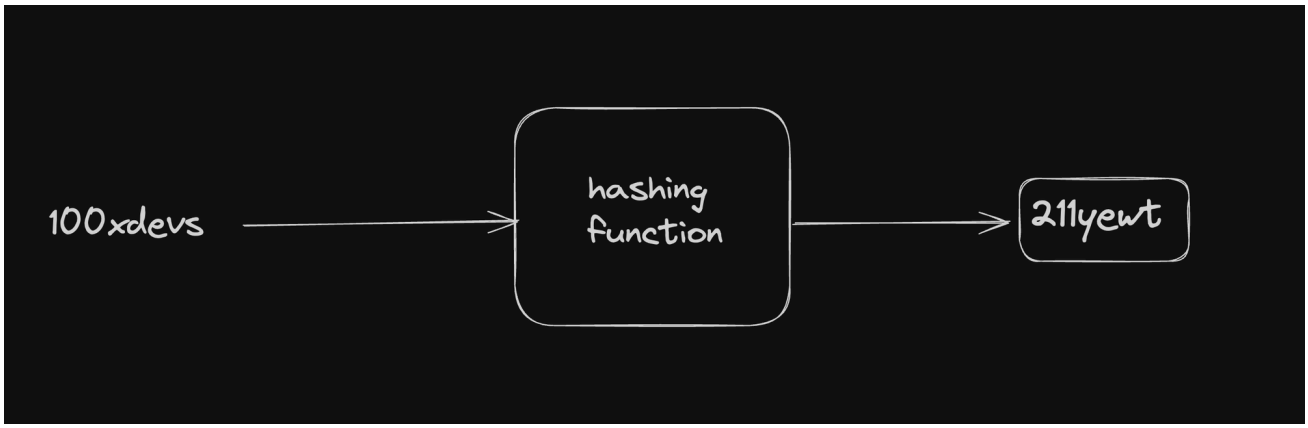
Hash functions have several important properties:

1. **Deterministic**: The same input will always produce the same output.

2. **Fast computation**: The hash value can be quickly computed for any given data.
3. **Pre-image resistance**: It should be computationally infeasible to reverse the hash function (i.e., find the original input given its hash output).
4. **Small changes in input produce large changes in output**: Even a tiny change in the input should drastically change the hash output.
5. **Collision resistance**: It should be computationally infeasible to find two different inputs that produce the same hash output.

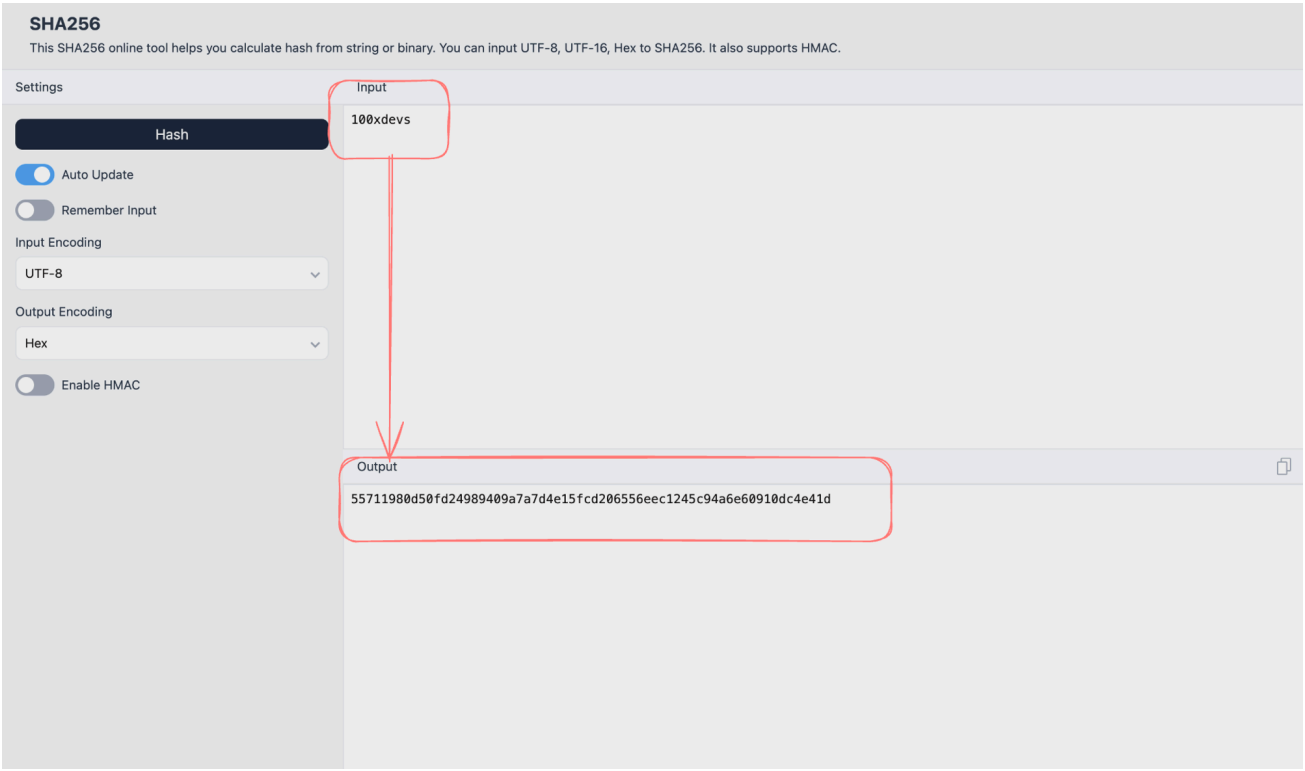
Is this a hashing algorithm?

What if I try "hashing" a string by increasing each alphabet's value by one. Do you think this follows all the rules we've written above?



SHA-256

Lets try out a famous hash function, SHA-256 here - https://emn178.github.io/online-tools/sha256.html



Node.js code for generating SHA-256

```
const crypto = require('crypto');

const input = "100xdevs";
const hash = crypto.createHash('sha256').update(input).digest('hex');

console.log(hash)
```

Copy

```
➜  ~ node index.js
55711980d50fd24989409a7a7d4e15fcd206556eec1245c94a6e60910dc4e41d
```

Intro to Proof of work

Assignment #1

What if I ask you the following question — Give me an input string that outputs a SHA-256 hash that starts with `00000` . **How will you do it?**

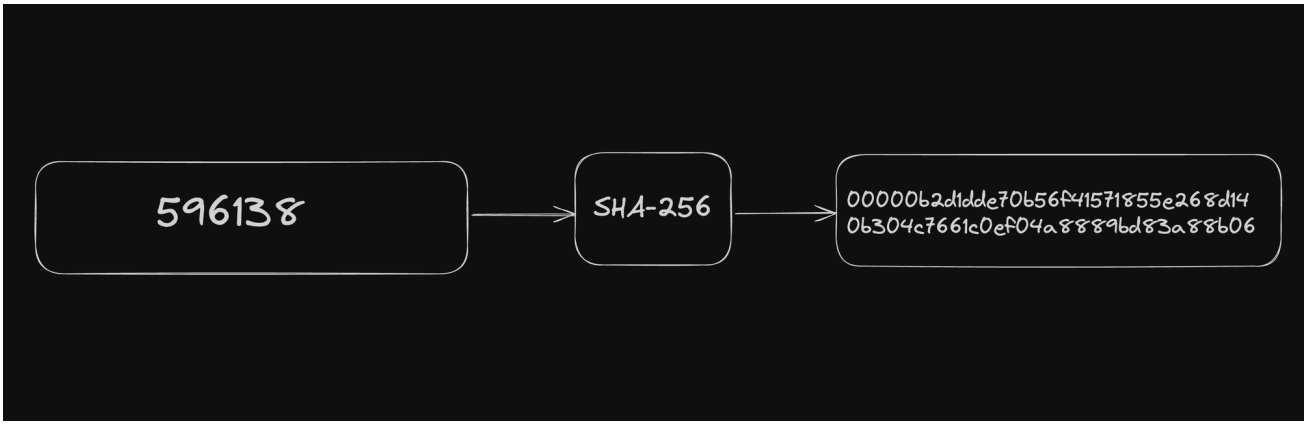
A: You will have to brute force until you find a value that starts with `00000`

▼ Node.js code

```
const crypto = require('crypto');

// Function to find an input string that produces a hash start
function findHashWithPrefix(prefix) {
    let input = 0;
    while (true) {
        let inputStr = input.toString();
        let hash = crypto.createHash('sha256').update(inputStr
        if (hash.startsWith(prefix)) {
            return { input: inputStr, hash: hash };
        }
        input++;
    }
}

// Find and print the input string and hash
const result = findHashWithPrefix('00000');
console.log(`Input: ${result.input}`);
console.log(`Hash: ${result.hash}`);
```



Assignment #2

What if I ask you that the `input string` should start with `100xdevs` ? How would the code change?

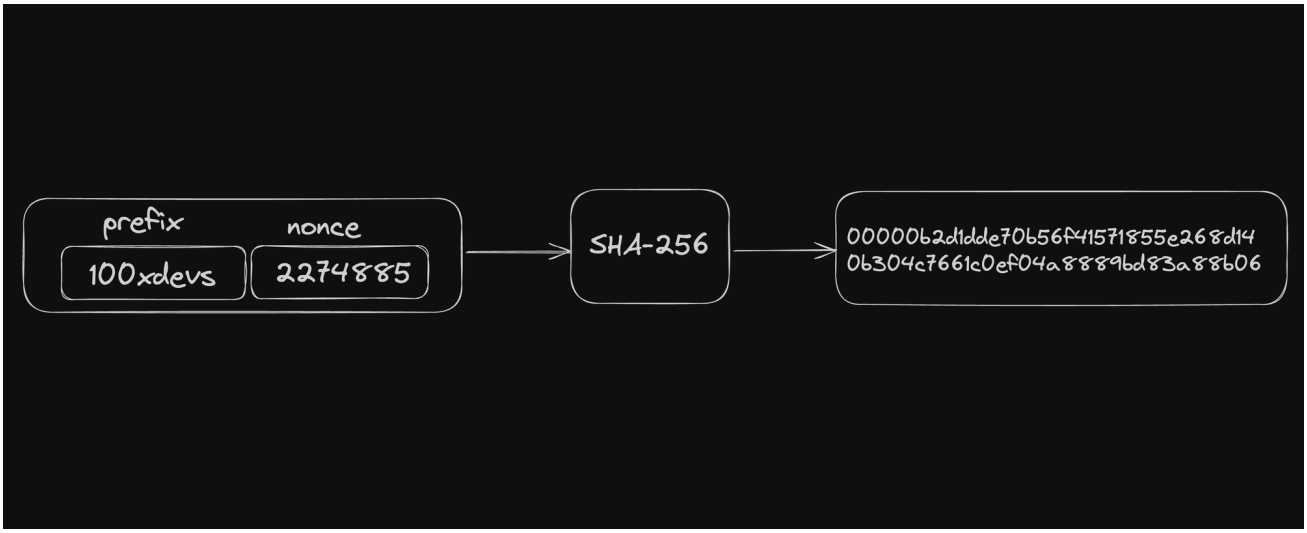
▼ Node.js code

```
const crypto = require('crypto');

// Function to find an input string that produces a hash start
function findHashWithPrefix(prefix) {
    let input = 0;
    while (true) {
        let inputStr = "100xdevs" + input.toString();
        let hash = crypto.createHash('sha256').update(inputStr
        if (hash.startsWith(prefix)) {
```

```
            return { input: inputStr, hash: hash };
        }
        input++;
    }
}

// Find and print the input string and hash
const result = findHashWithPrefix('00000');
console.log(`Input: ${result.input}`);
console.log(`Hash: ${result.hash}`);
```



Assignment #3

What if I ask you to `find` a nonce for the following input -

```
harkirat => Raman | Rs 100
Ram => Ankit | Rs 10
```

Copy

▼ Node.js code

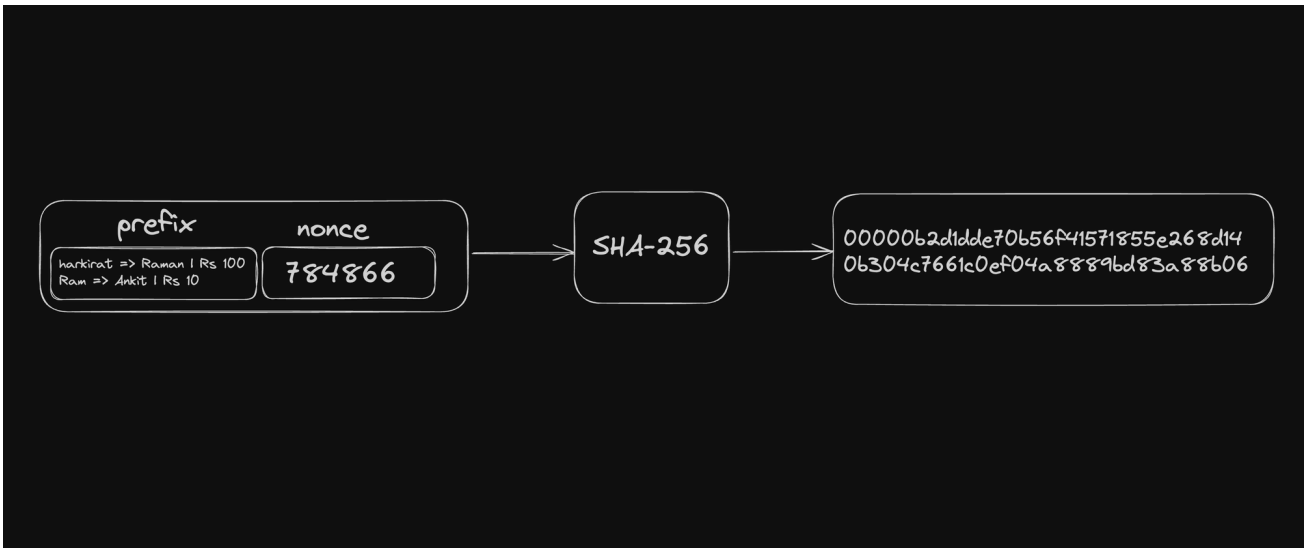
```
const crypto = require('crypto');

// Function to find an input string that produces a hash start
function findHashWithPrefix(prefix) {
    let input = 0;
    while (true) {
        let inputStr = `
harkirat => Raman | Rs 100
Ram => Ankit | Rs 10
` + input.toString();
        let hash = crypto.createHash('sha256').update(inputStr
        if (hash.startsWith(prefix)) {
```

Copy

```
            return { input: inputStr, hash: hash };
        }
        input++;
    }
}

// Find and print the input string and hash
const result = findHashWithPrefix('00000');
console.log(`Input: ${result.input}`);
console.log(`Hash: ${result.hash}`);
```



Assignment #4

Lets explore https://andersbrownworth.com/blockchain/

Intro to Bitcoin

Bitcoin white paper was released in 2008 - https://bitcoin.org/bitcoin.pdf

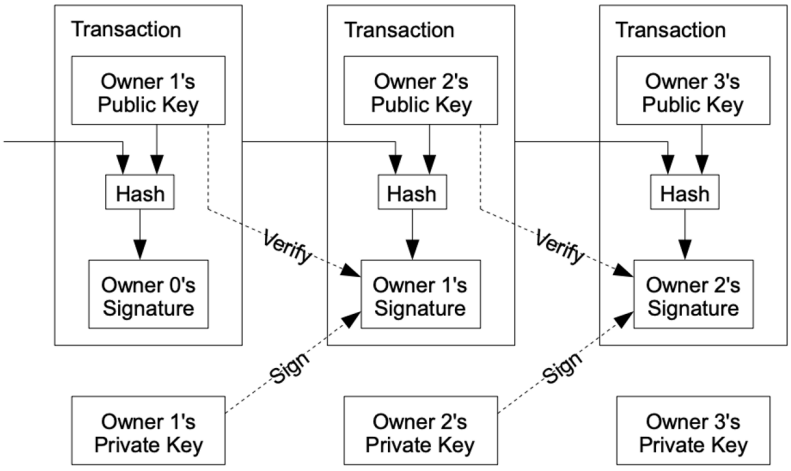
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

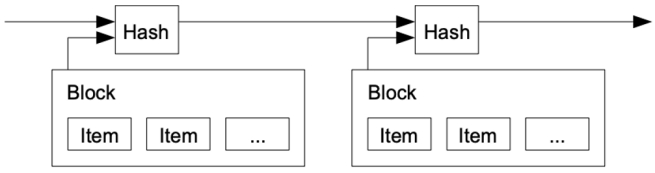


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

3. Timestamp server

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

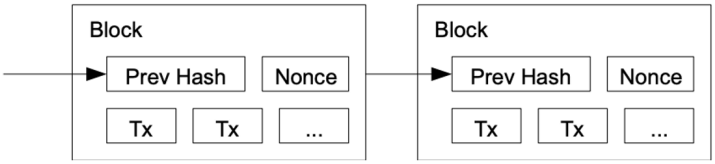


4. Proof of work

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up

5. Network

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

6. Incentive

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.