Timeline

2008 - Global Financial Crisis.

2009 - Bitcoin whitepaper

https://bitcoin.org/bitcoin.pdf

2010 - The 10k BTC Pizza

https://www.youtube.com/watch?v=j28hkTJMuTA

2015 – Ethereum Launch

2018 – Solana Founded

2020 – Solana Mainnet Beta

2020 – COVID-19 & Infinite Money printing

2021 – NFT, DeFi & Solana Breakout

2025 - Trump coin/Memecoin saga/Hyperliquid/Lighter

Problems with currencies

Inflation

Last year

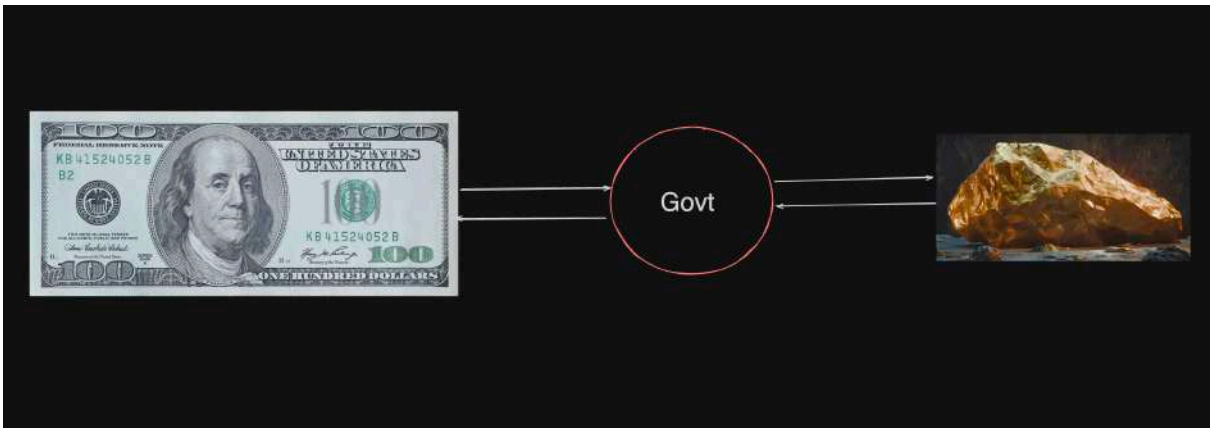
This year

Centralized

The govt/RBI gets to decide when to mint, where to use the newly minted supply, if there is a need for stimulus cheques.

Not backed by anything

There was a time when currencies were backed by gold. You could always take `$X` to the govt and they would return you 10 grams of gold. That no longer holds true.



Gold as a store of value

People consider **gold a store of value** because, across history and economics, it has several qualities that help it *preserve purchasing power over long periods of time*. The main reasons are:

1. Scarcity and Limited Supply

Gold is rare and costly to extract. Unlike paper money, it **can't be created at will** by governments or central banks. This scarcity helps prevent rapid devaluation.

2. Durability

Gold doesn't rust, corrode, or decay. A gold coin from 2,000 years ago is still essentially the same today, which makes it reliable for long-term value storage.

3. Universal Acceptance

Across cultures and civilizations, gold has been recognized as valuable. Even today, it's traded globally and accepted almost everywhere, making it highly liquid.

4. Intrinsic Demand

Gold has value beyond investment:

- Jewelry
- Electronics
- Dentistry
- Industrial uses

This ongoing demand supports its price even when financial systems are unstable.

---

5. Inflation Hedge (Historically)

Over long periods, gold has tended to **retain purchasing power** when currencies lose value due to inflation or excessive money printing.

---

6. Independence from Governments

Gold isn’t tied to any single country’s economy or political system. When trust in governments, banks, or fiat currencies declines, people often turn to gold.

Bitcoin as a store of value/currency/digital gold

1. Gold is hard/invonvinient to carry
2. Very hard to tell the purity of gold w/o getting it checked.
3. Inefficient markets (India price is different from US price, gold shops have >3-5% making charges)

Wouldn't it be nice if something `digital` holds the same properties as gold?

Where price discovery was better than gold, purity could be judged digitally.

Where it would be easy to create global marketpalces rather than localized gold shops.

This is what was introduced in the BTC whitepaper

Bits and bytes

What is a Bit?

A bit is the smallest unit of data in a computer and can have one of two values: 0 or 1.

Think of a bit like a light switch that can be either off (0) or on (1).

What is a byte?

A byte is a group of `8` bits. It's the standard unit of data used to represent a single character in memory. Since each bit can be either 0 or 1, a byte can have 2^8 (256) possible values, ranging from 0 to 255

Assignment

What is the `11001010` converted to a `decimals` ?

▼ Answer

- **2^7:** 1×27=1×128=128
- **2^6:** 1×26=1×64=64
- **2^5:** 0×25=0×32=0
- **2^4:** 0×24=0×16=0
- **2^3:** 1×23=1×8=8

- **2^2:** 0×22=0×4=0
- **2^1:** 1×21=1×2=2
- **2^0:** 0×20=0×1=0
  = 202

Representing bits and bytes in JS

- Bit

```
const x = 0;
console.log(x);
```

- Byte

```
const x = 202
console.log(x);
```

- Array of bytes

```
const bytes = [202, 244, 1, 23]
console.log(bytes);
```

UInt8Array

A better way to represent an array of bytes is to use a `UInt8Array` in JS

```
let bytes = new Uint8Array([0, 255, 127, 128]);
console.log(bytes)
```

Why use `UInt8Array` over `native arrays` ?

- They use less space. Every number takes 64 bits (8 bytes). But every value in a `UInt8Array` takes 1 byte.
- UInt8Array Enforces constraints - It makes sure every element doesn't exceed 255.

Assignment -

What do you think happens to the first element here? Does it throw an error?

```
let uint8Arr = new Uint8Array([0, 255, 127, 128]);
uint8Arr[1] = 300;
```

Encodings

Bytes are cool but highly unreadable. Imagine telling someone

> Hey, my name is 00101011101010101020

It's easier to `encode` data so it is more `human readable` . Some common encodings include -

1. Ascii
2. Hex
3. Base64
4. Base58

Ascii

`1 character = 7 bits`

Every byte corresponds to a `text` on the `computer` .
Here is a complete list - https://www.w3schools.com/charsets/ref_html_ascii.asp#:~:text=The ASCII Character Set&text=ASCII is a 7-bit,are all based on ASCII.

▼ Bytes to Ascii

```
function bytesToAscii(byteArray) {
  return byteArray.map(byte => String.fromCharCode(byte)).join('');
}

// Example usage:
const bytes = [72, 101, 108, 108, 111]; // Corresponds to "Hello"
const asciiString = bytesToAscii(bytes);
console.log(asciiString); // Output: "Hello"
```

▼ Ascii to bytes

```
function asciiToBytes(asciiString) {
  const byteArray = [];
```

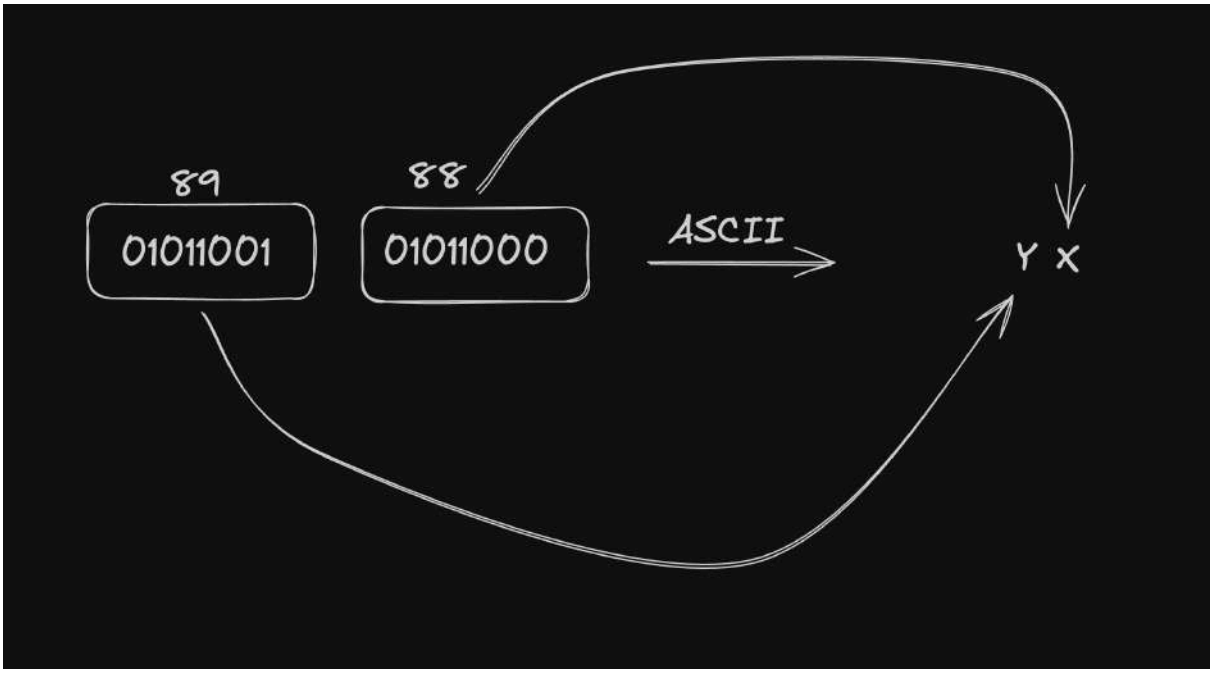
```
  for (let i = 0; i < asciiString.length; i++) {
    byteArray.push(asciiString.charCodeAt(i));
  }
  return byteArray;
}

// Example usage:
const ascii = "Hello";
const byteArray = asciiToBytes(ascii);
console.log(byteArray); // Output: [72, 101, 108, 108, 111]
```

▼ UInt8Array to ascii

```
function bytesToAscii(byteArray) {
  return new TextDecoder().decode(byteArray);
}

// Example usage:
const bytes = new Uint8Array([72, 101, 108, 108, 111]); // Corresponds to
"Hello"
const asciiString = bytesToAscii(bytes);
console.log(asciiString); // Output: "Hello"
```

▼ Ascii to UInt8Array

```
function asciiToBytes(asciiString) {
  return new Uint8Array([...asciiString].map(char ⇒ char.charCodeAt
(0)));
}

// Example usage:
const ascii = "Hello";
const byteArray = asciiToBytes(ascii);
console.log(byteArray); // Output: Uint8Array(5) [72, 101, 108, 108, 111]
```



Hex

1 character = 4 bits

A single hex character can be any of the 16 possible values: `0-9` and `A-F`.

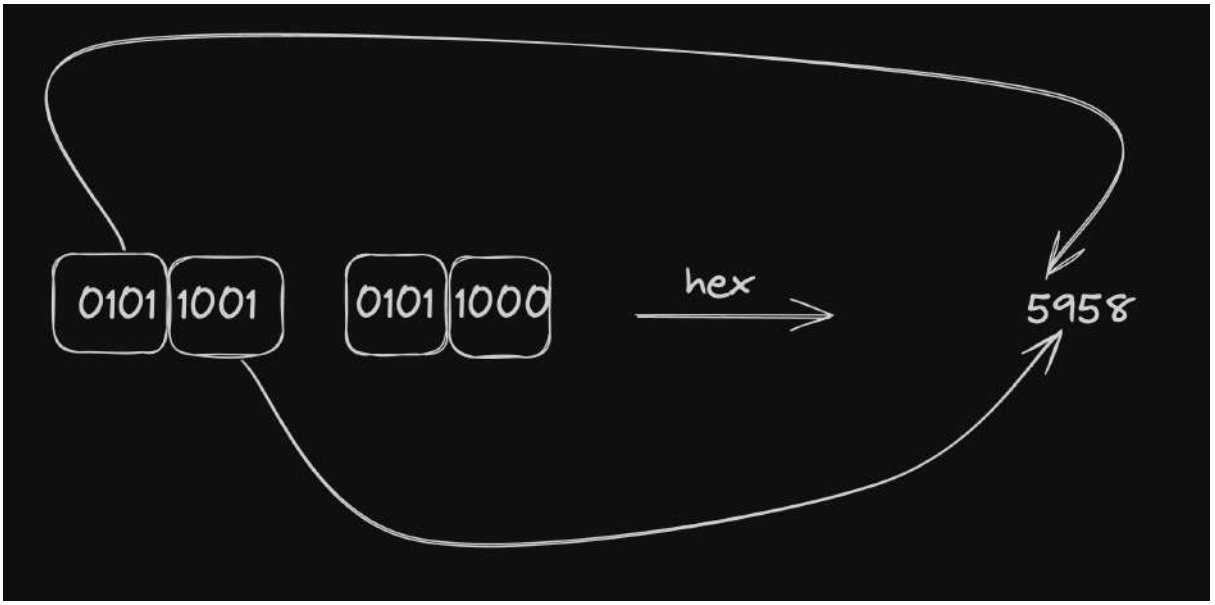
▼ Array to hex

```
function arrayToHex(byteArray) {
  let hexString = '';
  for (let i = 0; i < byteArray.length; i++) {
    hexString += byteArray[i].toString(16).padStart(2, '0');
  }
  return hexString;
}

// Example usage:
const byteArray = new Uint8Array([72, 101, 108, 108, 111]); // Correspon
ds to "Hello"
const hexString = arrayToHex(byteArray);
console.log(hexString); // Output: "48656c6c6f"
```

▼ Hex to array

```
function hexToArray(hexString) {
  const byteArray = new Uint8Array(hexString.length / 2);
  for (let i = 0; i < byteArray.length; i++) {
    byteArray[i] = parseInt(hexString.substr(i * 2, 2), 16);
  }
  return byteArray;
}

// Example usage:
const hex = "48656c6c6f";
const byteArrayFromHex = hexToArray(hex);
console.log(byteArrayFromHex); // Output: Uint8Array(5) [72, 101, 108, 1
08, 111]
```

Ref - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt



Base64

1 character = 6 bits

Base64 encoding uses 64 different characters (`A-Z`, `a-z`, `0-9`, `+`, `/`), which means each character can represent one of 64 possible values.

https://www.base64encode.org/

https://www.base64decode.org/

▼ Encode

```
const uint8Array = new Uint8Array([72, 101, 108, 108, 111]);
const base64Encoded = Buffer.from(uint8Array).toString("base64");
console.log(base64Encoded);
```

Base58

It is similar to Base64 but uses a different set of characters to avoid visually similar characters and to make the encoded output more user-friendly

Base58 uses 58 different characters:

- Uppercase letters: `A-Z` (excluding `I` and `O`)
- Lowercase letters: `a-z` (excluding `l`)
- Numbers: `1-9` (excluding `0`)
- `+` , `/`

▼ Encode

```
const bs58 = require('bs58');

function uint8ArrayToBase58(uint8Array) {
  return bs58.encode(uint8Array);
}

// Example usage:
const byteArray = new Uint8Array([72, 101, 108, 108, 111]); // Correspon
ds to "Hello"
const base58String = uint8ArrayToBase58(byteArray);
console.log(base58String); // Output: Base58 encoded string
```

▼ Decode

```
const bs58 = require('bs58');

function base58ToUint8Array(base58String) {
```

```
  return bs58.decode(base58String);
}

// Example usage:
const base58 = base58String; // Use the previously encoded Base58 st
ring
const byteArrayFromBase58 = base58ToUint8Array(base58);
console.log(byteArrayFromBase58); // Output: Uint8Array(5) [72, 101, 1
08, 108, 111]
```

Ascii vs UTF-8

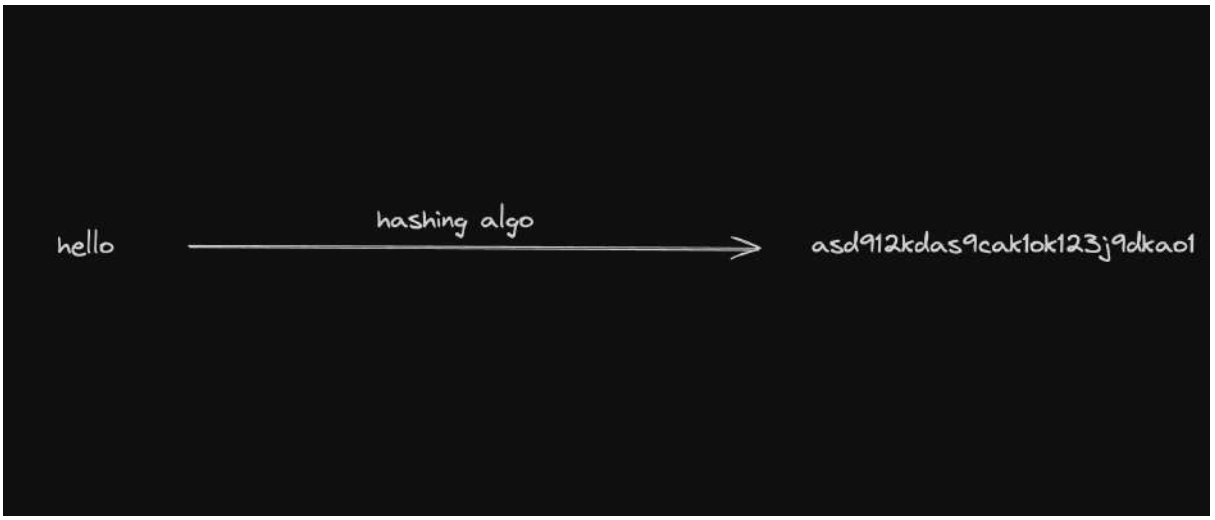
- ASCII uses a 7-bit encoding scheme.
- UTF-8 uses 1 to 4 bytes to encode each character.

https://www.fileformat.info/info/charset/UTF-8/list.htm

Hashing vs encryption

Hashing

Hashing is a process of converting data (like a file or a message) into a fixed-size string of characters, which typically appears random.



Common hashing algorithms - SHA-256, MD5

▼ Differences b/w md5 and sha-256

MD5

- **Output size:** 128 bits (32 hex characters)
- **Speed:** Very fast
- **Security:** ❌ **Broken**
- **Collisions:** Easy to generate (two different inputs with same hash)
- **Use today:**
  - ❌ Cryptography
  - ❌ Password hashing
  - ⚠️ Only for non-security uses (e.g., quick checksums)

Status: Obsolete for security purposes.

SHA-256

- **Output size:** 256 bits (64 hex characters)
- **Speed:** Slower than MD5 (but still fast)
- **Security:** ✅ **Strong**
- **Collisions:** Computationally infeasible with current technology
- **Use today:**
  - ✅ Cryptography
  - ✅ Digital signatures
  - ✅ Blockchain (Bitcoin)

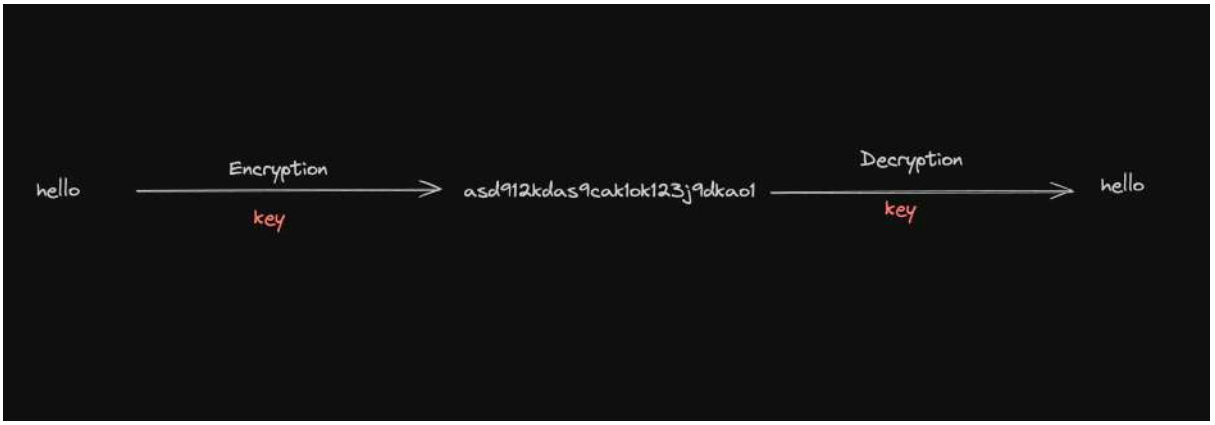
Encryption

Encryption is the process of converting plaintext data into an unreadable format, called ciphertext, using a specific algorithm and a key. The data can be decrypted back to its original form only with the appropriate key.

Key Characteristics:

- **Reversible**: With the correct key, the ciphertext can be decrypted back to plaintext.
- **Key-dependent**: The security of encryption relies on the secrecy of the key.
- **Two main types**:
  - **Symmetric encryption**: The same key is used for both encryption and decryption.
  - **Asymmetric encryption**: Different keys are used for encryption (public key) and decryption (private key).

Symetric encryption

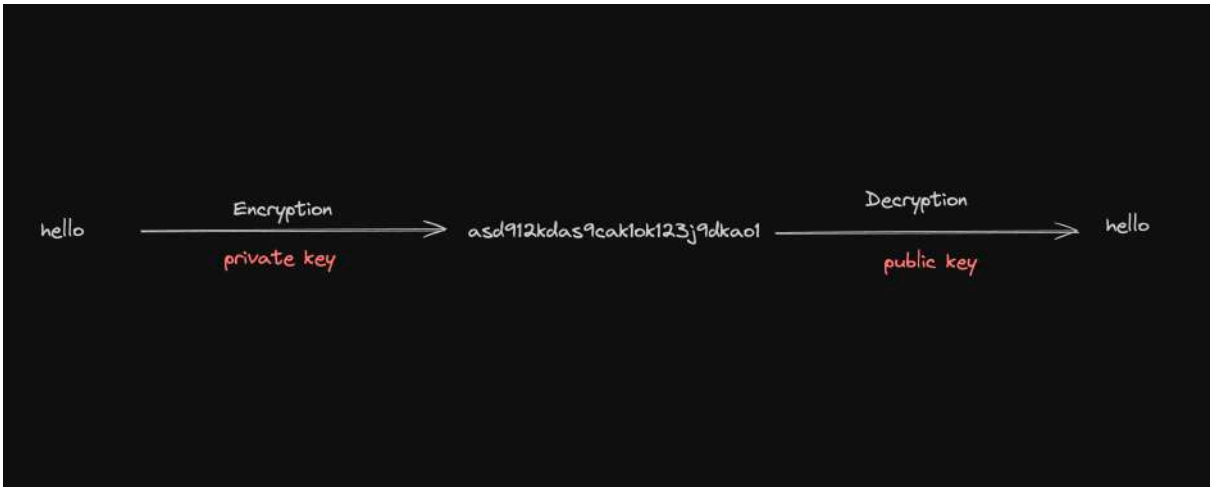


▼ Code

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

// Generate a random encryption key
const key = crypto.randomBytes(32); // 32 bytes = 256 bits
const iv = crypto.randomBytes(16); // Initialization vector (IV)

// Function to encrypt text
function encrypt(text) {
    const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);
    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    return encrypted;
}

// Function to decrypt text
function decrypt(encryptedText) {
    const decipher = crypto.createDecipheriv('aes-256-cbc', key, iv);
    let decrypted = decipher.update(encryptedText, 'hex', 'utf8');
    decrypted += decipher.final('utf8');
    return decrypted;
}

// Example usage
const textToEncrypt = 'Hello, World!';
const encryptedText = encrypt(textToEncrypt);
const decryptedText = decrypt(encryptedText);
```

```
console.log('Original Text:', textToEncrypt);
console.log('Encrypted Text:', encryptedText);
console.log('Decrypted Text:', decryptedText);
```

Asymetric encryption

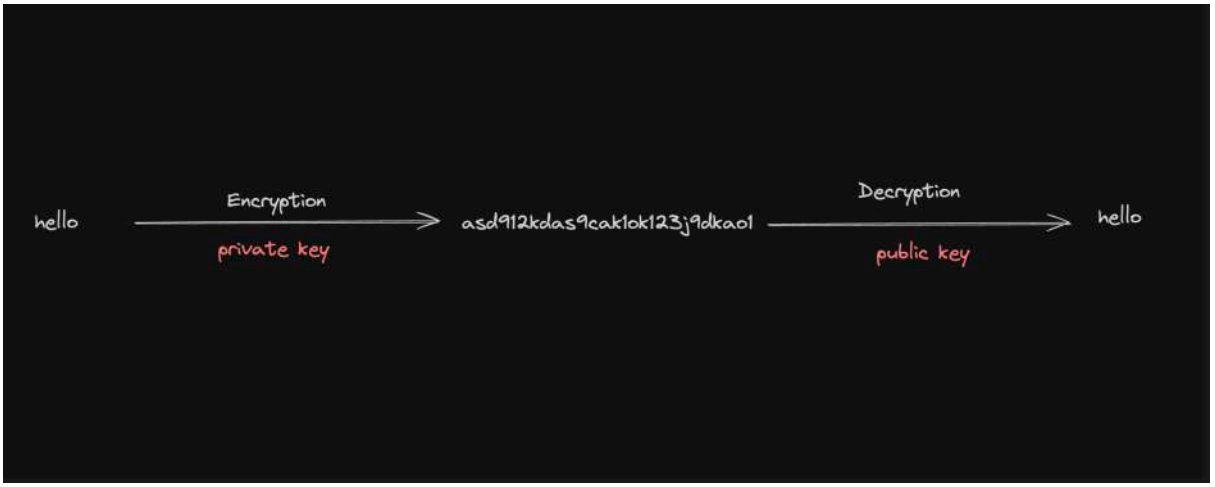


Asymetric Encryption

Asymmetric encryption, also known as public-key cryptography, is a type of encryption that uses a pair of keys: a public key and a private key. The keys are mathematically related, but it is computationally infeasible to derive the `private key` from the `public key`.

Public Key: The public key is a string that can be shared openly

Private Key: The private key is a secret cryptographic code that must be kept confidential. It is used to decrypt data encrypted with the corresponding public key or to create digital signatures.



Common Asymmetric Encryption Algorithms:

1. RSA - Rivest–Shamir–Adleman
2. ECDSA - Elliptic Curve Digital Signature Algorithm - ETH and BTC
3. **EdDSA -** Edwards-curve Digital Signature Algorithm - SOL

How eliptic curves work - https://www.youtube.com/watch?v=NF1pwjL9-DE&

Common eleptic curves

1. secp256k1 - BTC and ETH
2. ed25519 - SOL

Few usecases of public key cryptography -

1. SSL/TLS certificates

2. SSH keys to connect to servers/push to github
3. Blockchains and cryptocurrencies

Creating a public/private keypair

Let's look at various ways of creating public/private keypairs, signing messages and verifying them

EdDSA - Edwards-curve Digital Signature Algorithm - ED25519

▼ Using `@noble/ed25519`

```
import * as ed from "@noble/ed25519";

async function main() {
  // Generate a secure random private key
  const privKey = ed.utils.randomPrivateKey();

  // Convert the message "hello world" to a Uint8Array
  const message = new TextEncoder().encode("hello world");

  // Generate the public key from the private key
```

```
  const pubKey = await ed.getPublicKeyAsync(privKey);

  // Sign the message
  const signature = await ed.signAsync(message, privKey);

  // Verify the signature
  const isValid = await ed.verifyAsync(signature, message, pubKey);

  // Output the result
  console.log(isValid); // Should print `true` if the signature is valid
}

main();
```

▼ Using `@solana/web3.js`

```
import { Keypair } from "@solana/web3.js";
import nacl from "tweetnacl";

// Generate a new keypair
const keypair = Keypair.generate();

// Extract the public and private keys
const publicKey = keypair.publicKey.toString();
const secretKey = keypair.secretKey;

// Display the keys
console.log("Public Key:", publicKey);
console.log("Private Key (Secret Key):", secretKey);

// Convert the message "hello world" to a Uint8Array
const message = new TextEncoder().encode("hello world");

const signature = nacl.sign.detached(message, secretKey);
const result = nacl.sign.detached.verify(
  message,
  signature,
  keypair.publicKey.toBytes(),
```

```
);

console.log(result);
```

ECDSA (Elliptic Curve Digital Signature Algorithm) - secp256k1

▼ Using `@noble/secp256k1`

```
import * as secp from "@noble/secp256k1";

async function main() {
  const privKey = secp.utils.randomPrivateKey(); // Secure random private key
  // sha256 of 'hello world'
  const msgHash =
    "b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9";
  const pubKey = secp.getPublicKey(privKey);
  const signature = await secp.signAsync(msgHash, privKey); // Sync methods below
  const isValid = secp.verify(signature, msgHash, pubKey);
  console.log(isValid);
}

main();
```

▼ Using `ethers`

```
import { ethers } from "ethers";

// Generate a random wallet
const wallet = ethers.Wallet.createRandom();

// Extract the public and private keys
const publicKey = wallet.address;
const privateKey = wallet.privateKey;

console.log("Public Key (Address):", publicKey);
```

```
console.log("Private Key:", privateKey);

// Message to sign
const message = "hello world";

// Sign the message using the wallet's private key
const signature = await wallet.signMessage(message);
console.log("Signature:", signature);

// Verify the signature
const recoveredAddress = ethers.verifyMessage(message, signature);

console.log("Recovered Address:", recoveredAddress);
console.log("Signature is valid:", recoveredAddress === publicKey);
```

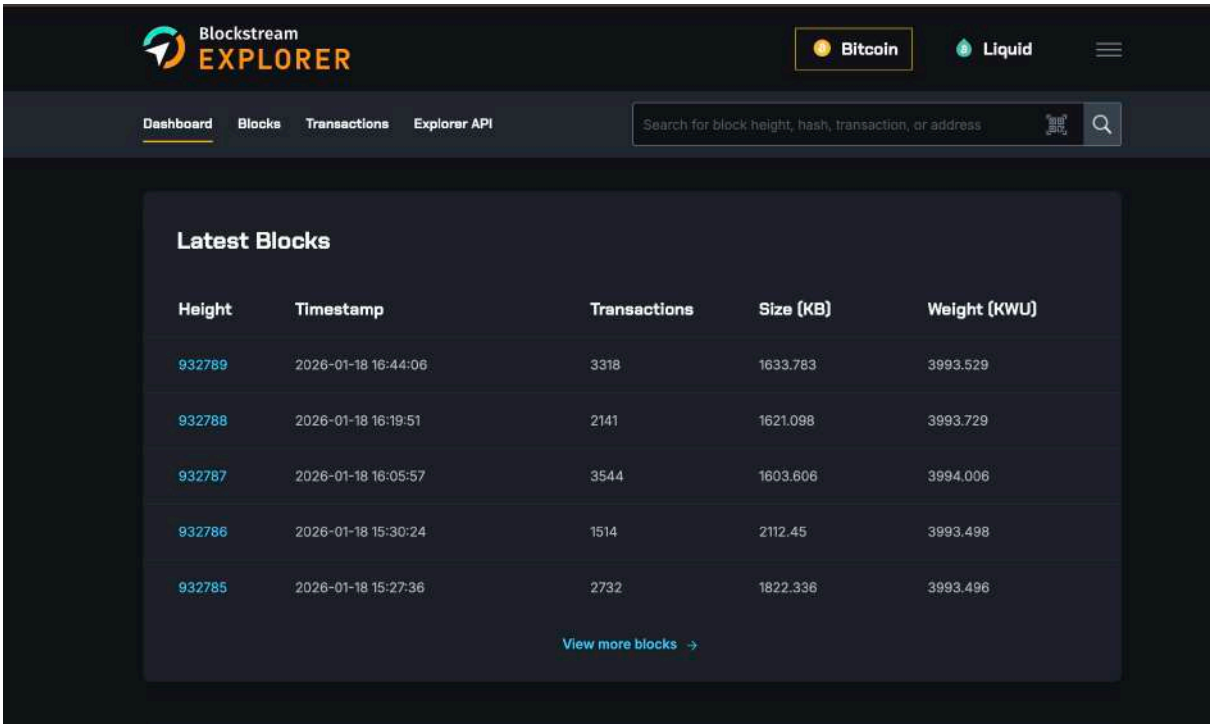
How would you create a decentralized blockchain/currency?

1. Write a program (./bitcoin_miner.exe) that people can run on their machine to become `miners` of the network
2. Write code in the program that dictates how the currency is minted (new currency is created) and who gets the initial supply (developers, initial investors, airdrop)
3. Let anyone join the network/be a miner by running the program (bitcoin_miner.exe)
4. If people want to send BTC/receive BTC, they should create a `public private keypair` (no username password like banks)
5. To send BTC from your wallet (your public key) to a different wallet, `sign a transaction/message` and send the transaction to one/many miners (btc-miner-india.100xdevs.com)
6. Miners bundle multiple transactions (lets say all txns that came in the last 10s) in a block. The block is transmitted by the miner to everyone else in the network. This creates a permanent record/ledger of all the blocks since the starting (genesis).
7. A chain of these blocks is called the blockchain.

There are a lot of problems in this approach that we will discuss/fix in the upcoming slides.

Ref - https://www.canva.com/design/DAG-wwcLjb0/-PJvxmw65928wfsLQDPOrQ/edit?utm_content=DAG-wwcLjb0&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Latest blocks - https://blockstream.info/



Aims to create 1 block per 10 minutes by varying difficulty of proof of work

Problems with current approach

▼ Can a miner move money from one random address to another?

No, only the user who holds the private key can sign the transaction. So even if a miner tries to `mimic` a transaction, other miners will reject the transaction signature.

▼ Which miner should I send my transaction to?

When you hit "send" in your wallet, your transaction gets broadcast to whichever Bitcoin nodes your wallet is connected to. From there, it propagates across the entire network through a gossip protocol—each node that receives your transaction validates it and forwards it to the other nodes it's connected to. Within seconds, your transaction typically reaches most nodes on the network, including those run by miners.

▼ What if a miner doesnt include my transaction?

It can happen. There can be blacklists/less incentive for a miner to include ur txn. That is why the concept of `tipping` exists

▼ Why would a person start a new miner? What do they get?

When you propose a block, you add a txn to the top that gives you the block reward. This is how new BTC is added to the system. You can also earn tips based on who wants their txn to be included the fastest.

1. Block subsidy (the "block reward")

This is newly created bitcoin that gets awarded to whoever mines a valid block. It started at 50 BTC per block in 2009 and halves roughly every four years (every 210,000 blocks). Currently it's 3.125 BTC per block after the April 2024 halving. This subsidy is how all new bitcoin enters circulation.

2. Transaction fees

Every transaction in the block includes a fee paid by the sender. The miner who finds the block collects all the fees from every transaction they included. When the mempool is congested, fees can add up to a meaningful amount—sometimes rivaling or even exceeding the block subsidy during periods of high demand.

▼ Why wouldnt a miner always start to propose a block then? I would want to keep all the block reward for myself. I'll just start a new miner and start proposing blocks every 1 minute.

To be able to propose a block, you need to solve a `cryptographically hard` problem.
If there are 100 miners right now all having 4 cpus, they are all solving the same problem.
On average you will only be able to propose a block once in 100 blocks (simple probability).

This is why people say BTC/crypto is bad for the environment because thousands of machines all around the world are solving the same futile problem just to be the `block leader`

▼ What if two people propose a block at the same time?

This happens occasionally and is totally normal—it's called a **chain split** or **temporary fork**.

What actually happens

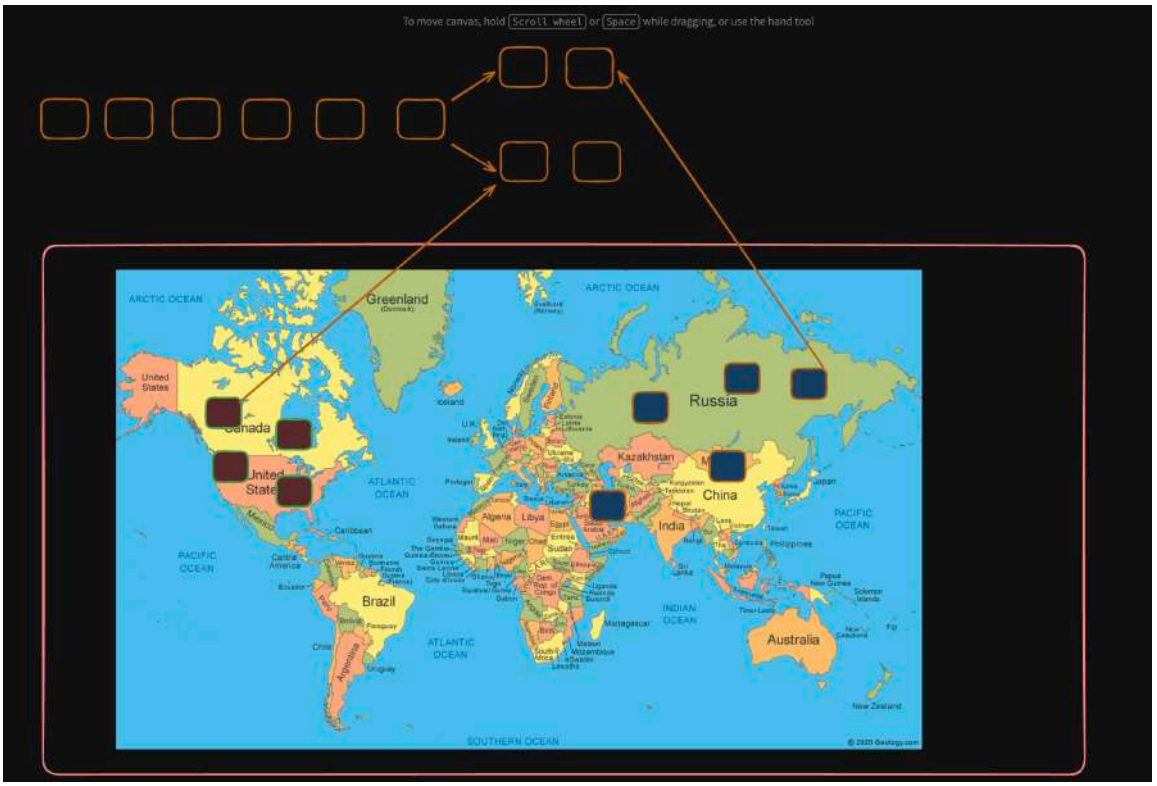
Say miners A and B both find valid blocks at roughly the same height. Some nodes will hear about block A first, others will hear about block B first. Each node considers the first valid block it receives as the current tip of the chain and starts building on that. So now you briefly have two competing versions of the blockchain, each with a portion of the network working on it.

How it resolves

The rule is simple: the longest chain wins (technically, the chain with the most cumulative proof-of-work). As soon as another block gets mined on top of either A or B, that chain becomes longer. Nodes following the shorter chain will immediately abandon it and switch to the longer one. The "orphaned" block gets discarded—its transactions return to the mempool to be included in a future block.

Implications

This is why people wait for "confirmations" before considering a transaction final. One confirmation means it's in a block, but that block could still get orphaned. Six confirmations has become the informal standard for high-value transactions—at that depth, a reorg is extraordinarily unlikely without a deliberate 51% attack.

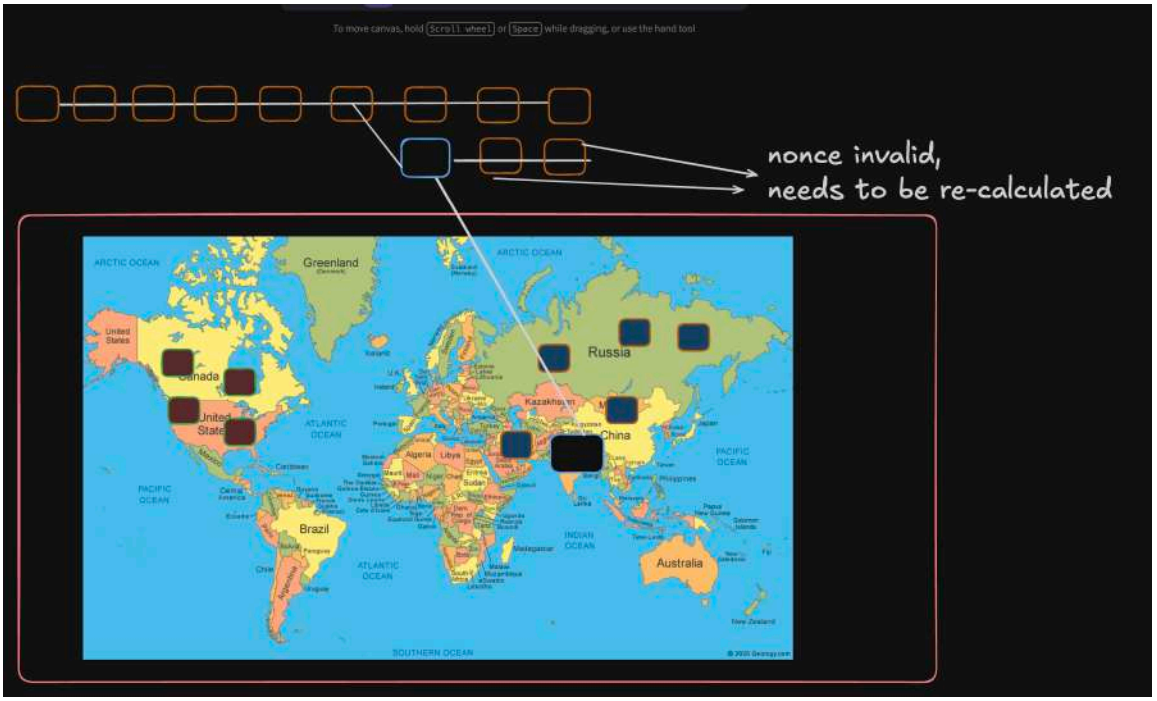


▼ How to forks get resolved?

Eventually one fork will start to become longer, and broadcast this fork to the miners currently maintaining the other fork. The other miners will then accept the longer fork and discard their original fork. This happens automatically and rarely does a fork sustain for more than a few hours

▼ Can a transaction ever be reversed? Can you ever create a new fork that doesnt have a transaction that you want removed from the blockchain?

A blocks nonce is generated with the txn signature of the prev block attached. You will have to compete with everyone else creating the longest chain while you try your best to create a fork.



▼ Visualization

https://andersbrownworth.com/blockchain/