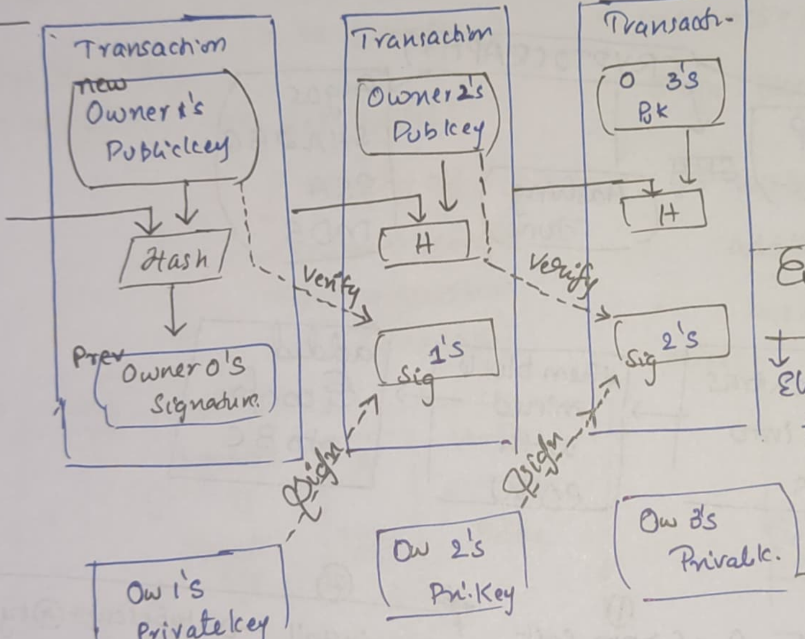
Project BC — Buiding My Own BlockChain.

Abstract →

What's Bitcoin

A peer-peer Electronic Cash Sys — Satoshi Nakamoto 2008 on white Paper

- Modern finSys ∝ TRUST of big centralized banks to hold out fiat currencies & execute transactions.
- Trust is a weakness that eventually req. intervention by lawyers & gov.
- BitC allows 2 parties make reliable transactions ∝ Crypographic proof
  ∝̸ trusty middle man
- Purpose → array 01 → meaningful way.
  ↓
  Protocol that makes it so
  ↓
  Blockchain — allows 2p to make trans



from financial Perspect.
↓
BC
↓
Public ledger that has all has all trans < all BCoin user
& is distrib & sync. around world
↓
Eliminates need of Central Aut to verify & validate

from tech Pers
↓
BC
↓
(Linkedlist) [dataBa]

Each rec/block rep a grp of transactions that have been permanently com → dB

→ Creation goes through v. strict cry. rules
→ each user/wallet — public key → to Receive
— Private key to Send/spend
→ Before u can spend, u need to PROVE u have a Publickey (owner)
& money has been Sent B

→ Each tras → has a
1. Hash: an encry rep of prev. trans
2. New owner's Public key
3. Sign of old owner's Private key

→ to validate ownership without any need to expose private key
→ makes it virtually imp to alter trans

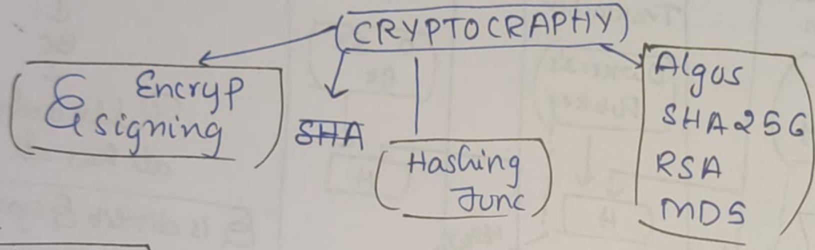
What if someone tries to → 2 diff people @ exact time
or
→ doublespend
?

② ← (DataMining) → ① Sys that allows multiple comp acc. Works to agree on appropriate state of entire Sys

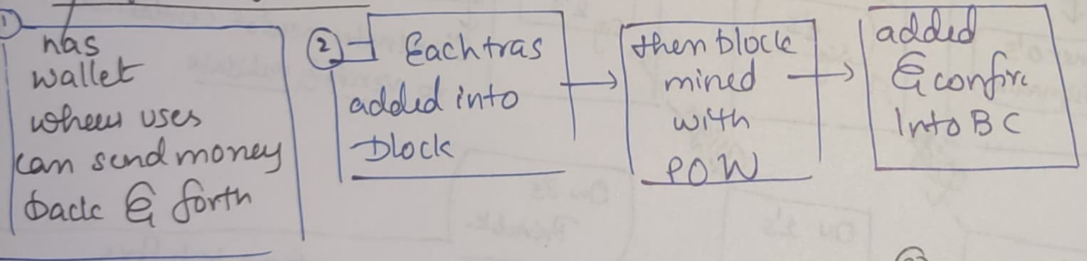
Each new transaction is broadcast to whole network

- The trans r packaged into blocks → miners will spend cp → Mine/solve compute a proof for a random prob thats vrly diff to solve, but easy to verify ← (PROOF OF WORK) ~~validate~~

- He gets a portion of Bco as reward
- Now Block is broadcast back to other nodes → BC



impl-BC
(Using- nodejs, Ts)


①
has wallet where uses can send money back & forth
②
Each tras added into Block
then block mined with POW
added & confirm into BC

Step1 → initialize new node.js proj (npm init ② npm install -D typescript @types/node) ③ npm in node
→ npm install node, typescript
→ create tsconfig.json (c/p) [add how typescript compiler should comp .ts files]

```
"compilerOptions" [

]
```

→ Now to run the .ts files
↳ go to package.json
↳ create a script called 'dev'
↳ (dev : "tsc -w") — watch flag

tsc → typescript compiler

-w → to constantly compile our code to plain java script (in bg)

→ Now we can (npm run dev) → to keep ts running in background.

Step 2 — index.js (OOPS)

1. import * as crypto from 'crypto';

crypto library — inbuit in node.js modular node to handle bunch of func for Cryptogrp

2. we have 4 classes

Transaction

transf funds (objProps)

- amt of trasac.
- payer
- payee

→ convert obj → string
tostring()

Block

Containor for transa [Linked L]

- prevHash
- transaction
- ts (timeStamp)

→ get hash()
↓
stringify obj
specifies a specific hash algorithm (SHA256)

Chain → LL of Blocks

there should only be 1 BC. so, create a singleton instance
↳ usg static instance-

- instance = new Chain()
- chain : Block[] [array of B]
- get lastBlock()
- addBlock( )
- push( )
- mine()

Wallet

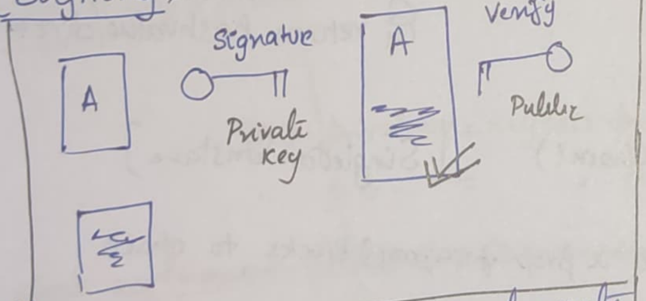
to securely send coin back & fth
wrapper for pubk & prik

- publickey
- privatekey
- keypair = crp.generate

(to generate pub k & pri k
↓
Algo RSA
↓
foll Encry. Algo
↳ Can Encry & Decrp

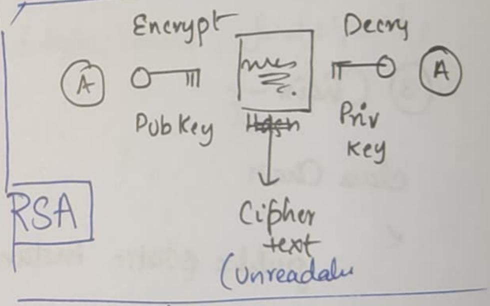
- sendMoney()

Signing



with sh → No need to Encrypt it. instead create a hash of it. then we sign it with private key. Then the msg can be verified with public key.

If someone tries to change msg, hash is diffent → verification fail



Encryption

① import

```
class Transaction
{
    constructor
    (
        public amount : number
               payer  : string
               payee  : string
    ) { }

    toString()
```

②

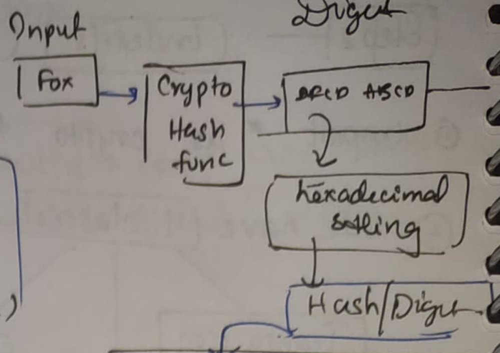
```
class Block
{
  constructor
  ( public prevHash : string
    public transach : Transaction,
    public ts        = Date.now()
  ) { }

  getHash()
  {
    const str = JSON.stringify(this)
    const hash = crypto.createHash('SHA256'),
    hash.update(str).end();
    return hash.digest('hex');
  }
```

convert obj → string

- hash the str [string version of block]
- then return the hashed val or diget

createHash('SHA256') → specifies a specific hash Algo called SHA256 [Secure Hash Algorithm] 256bits length

- A one-way cryptog. func
- Can encrypt ✓
- Can't decrypt back ×
- used to
- hash string version of block & return hashvalue (diget



Can't reconstruct A value from Hash

but can compare this Hashes

③ Chain —

```
class Chain
{
    public static instance = new Chain()        [Singleton Instance]

    chain : Block[];      // declare a prop of array of Blocks to chain

    constructor()       // define 1st block in chain [called genesis]
    {
        this.chain = [new Block(null, new Transaction(100, 'genesis', 'Satwi'))],
    }
                             ↓                ↓
                          prevHash      Transach Obj
                                              ↓
                                  (amount, payer, payee)

    get lastBlock()
    {
        return this.chain[this.chain.length -1]
    }
```

```
addBlock( transaction: Transaction , senderPublicKey: string , signature: string)
                                                                    → to verify later
{
  const newBlock = new Block(this.lastBlock.hash , transaction);
                                        ↓                  ↓
                                     prevHash        transactionObj
  // first create a newBlock
```

Prob → Noway to know this is a legitimate transfer to add Block.
So, rechange

```
class Wallet
{
    public publicKey;  → receive Money
    public privateKey; → send Money

    constructor()
    { const keypair = crypto.generateKeyPairSync('rsa', {
                      ①   generateKeyPairSync → diffalgo   'rsa' — FullEncAlgo
          modulusLength: 2048
          publicKeyEncoding: { type: 'spki', format: 'pem' },
          privateKeyEncoding: { type: 'pkcs8', format: 'pem' },
        }
      );

      this.privateKey = keypair.privateKey;
       "   pub  "     =   "   pub  "  ;
    }
```

format generated keys
⇓
string
# format: 'pem' → save to file/users

Now that we ~~have a publicK & privKey, we can use~~ them to send Money.

```
    sendMoney(amount: number, payeePublicKey: string)
    {
        const transaction = new Transaction(amount, this.publicKey, payeePublicKey);

        const sign = crypto.createSign('SHA256') → create a signature
        sign.update(transaction.toString()).end(); → using trans.data as value

        const signature = sign.sign(this.privateKey);  ← like create 1 time Pass
        Chain.instance.addBlock(transaction, this.publicKey, signature);
    }
}
```

Now, we created a signature & sent it to addBlock() & verify

①

```
const newBlock = new Block (this.LastBlock.hash, transaction);
this.chain.push (newBlock);
```

X → as its not secure & not verified.

⇓

② So, then, we created a signature, & now we create a verifier to verify it

```
addBlock ( transaction: Transaction , sendPublicKey: string , signature: Buffer )
{
    const verifier = crypto.createVerify ('SHA256')
    verifier.update (transaction.toString ());
    const isValid = verifier.verify (senderPublicKey, signature )
    if (isValid)
    {
        const newBlock = new Block(this.lastBlock.hash, transaction);
        this.chain.push (newBlock);
    }
```

→ X → Now we verified that user actually trying to spend that amt

but what if he tries to 2 diff users @ sametime so, they may potentially spend more than they own. [Double Spend issue]

③ POW ⟸

⇓

(mining)

(difficult comp prob is solved inorder to Confirm the block
easy - verify by other nod)

mining → distributed → (multiple nodes competing) [lottery Type] → winner gets port of coin so, competition.

[Cloud Computing Resour ⟹ Money]

POW Implement

```
In  Block
{
    public nonce = Math.round()* 999999999);   ↗ one-time Random No
}
```

Chain

```
{
        add method called mine

    mine (nonce : number)

        {   let solution = 1;
            console.log('⛏ mining...')

            while (true)

            {   const hash = crypto.createHash('MD5');
                hash.update((nonce + solution)).toString()).end();

                const attempt = hash.digest('hex');

                if (attempt.substr(0,4) === '000')

                {  console.log('Solved: ${solution}');
                   return solution;

                }

            solution += 1;

            }

    }
```

→ takes nonce as arg

tries to find a number

[when nonce + number = hash #0000

hash that starts with 0000]

creates a hash with MD5 algo

msg-dijest Algo

≅ SHA256 but is 128 bits

& is faster to compute

when thats found,

Mined!!