22/09/22

Full Domain Hash scheme

```
sig(m, d)
{
    S = (H(m))^d mod n
    return S
}.
```

```
Verify (m, S, e)
{
    compute h' = S^e mod n
    if (h' == H(m)).
        accept
    else
        reject
}.
```

Theorem: Suppose RSA is (t', ϵ') secure Then the Full domain Hash signature Scheme is (t, ϵ) secure where.

$$t = t' - (q_{hash} + q_{sig} + t_{\theta(k^3)}$$

$$\epsilon = (q_{hash} + q_{sig})\epsilon'$$

Hash used to break homomorphism in RSA (to prevent forgery).

(can use padding & instead of hash but it increases message size and hence the complexity)

RSA hard problem :

Given $y = x^e \bmod n$, and e, n it is hard to find x

RSA is (t', ϵ') secure means that RSA hard problem can be solved in (feasible) time t' with probability $\epsilon < \epsilon'$

$\left(\epsilon' = neg(x) = \frac{1}{2^{100}}\right)$.

$\Rightarrow$ RSA is secure

For full domain hash signature scheme to be secure

t should be ~~Exponential~~ polynomial

ϵ should be $\frac{1}{\text{exponential}}$

$q_{hash} + q_{sig} \Rightarrow$ polynomial

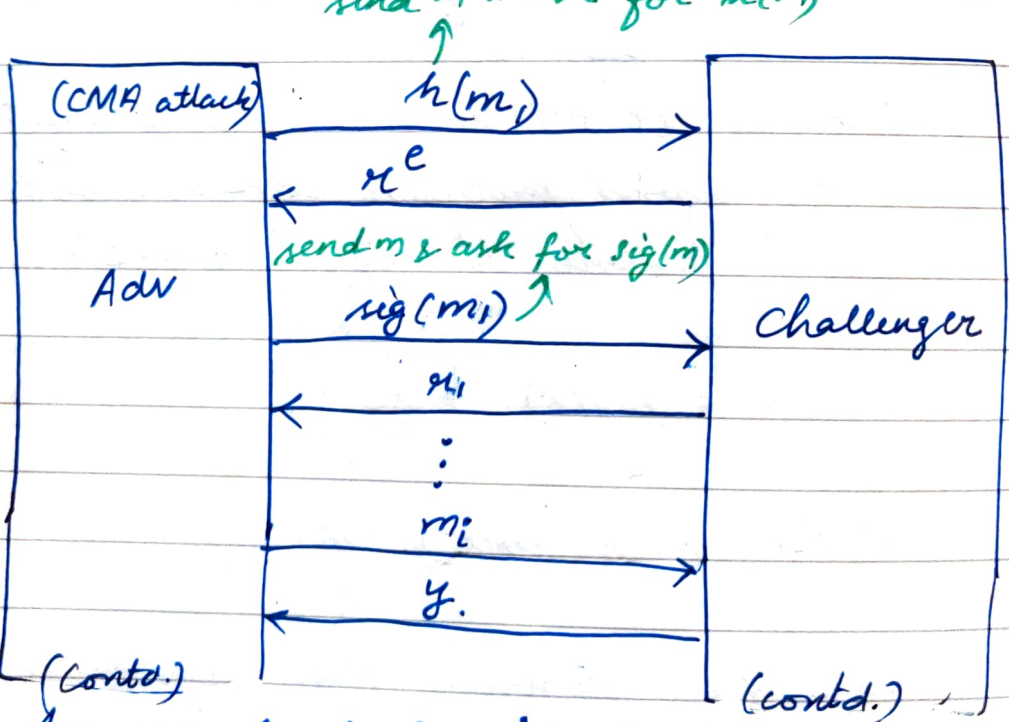
$(q_{rom} + q_{sig})$

$\epsilon = poly \times \frac{1}{exp} \rightarrow \epsilon'$

$= \frac{1}{exp}$

To prove: If ~~an~~ ~~full~~ ~~domain~~ an adversary can break full domain hash signature scheme then there exists another challenger that can solve RSA hard problem.

existential forgery under CMA



Assume hash is ideal :

→ op is completely random (only for this proof).

In CMA, adversary first asks for hash of his choice of message & the signature

Challenger does not know the private key but he can give any value as the hash value (as it is completely random).

So challenger finds $h(m)$ (chosen randomly).

and returns $(h(m))^e$ or say ~~r^e ($r : h(m)$)~~.

Since challenger does not know private key d, ~~but~~ he can simply choose

$r = (h(m))^d$ randomly (since $h(m)$ is random).

and returns r^e

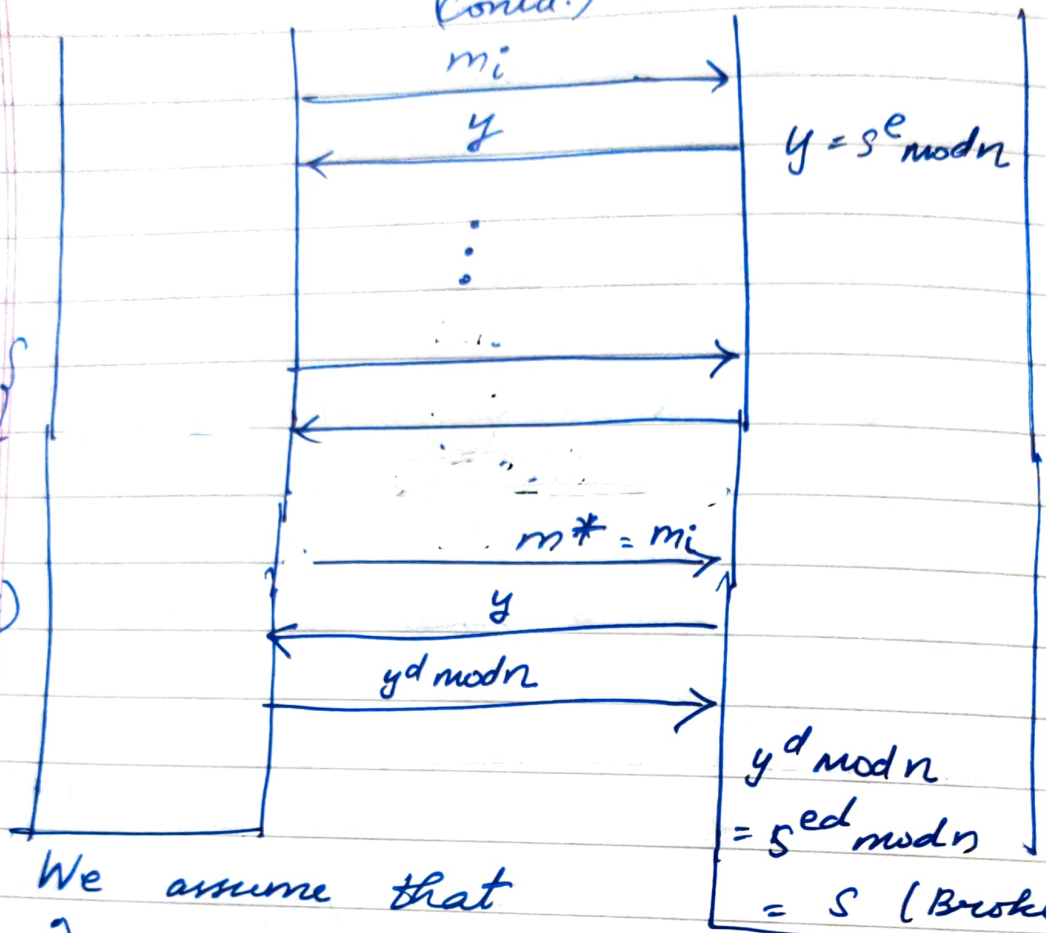
$\Rightarrow$ signature is r

Adv can ask poly #queries (say 1000).

At the i^{th} such query, challenger ~~choose~~ does not choose r, but gives y back

But challenger will not be able to answer the signature of m_i since he does ~~how~~ not know d ($S = y^d \bmod n$).

(contd.)



We assume that

i) adversary will ask hash query on $m^* = m_i$

ii) adversary ~~dto~~ does not ask sig query for some msgs. but he asks hash for all msgs (polynomial #msgs)

(Eg): $q_{hash} = 1000$, $q_{sig} = 500$.

If $m^* = m_1$, challenger already knows the signature on $m_1 = m_i$ (he generated it)

So it does not help him break the RSA-hard problem.

Challenger achieves existential forgery on m_i with prob. $\frac{1}{1000}$ (#queries = 1000)

(In this game adv is helping chal. solve RSA-hard problem and chal is helping adv break full domain hash by giving all that he wants).

If challenger can find

$$\begin{aligned} sig(m_i) &= y^d \\ &= s^{ed \bmod n} \\ &= s \end{aligned}$$

he can break RSA-hard problem

~~Adversary can break~~

Only if adversary breaks Full domain hash signature by making an existential forgery on m_i, will the challenger be able to break RSA-hard problem

(Adv sends y^d to chall!)

↳ s on $m^* = m_i$

Prob. that challenger solves the hard problem $(\epsilon') = \frac{1}{1000} \times$ prob. that adversary breaks the full domain hash signature scheme (ϵ)

$\therefore \epsilon' = \frac{1}{n}\epsilon$ in general with n queries

$$\epsilon' = \frac{1}{q_{hash} + q_{sig}}\epsilon$$

$m^* = m_i$ (otherwise challenger knows the signature already for any message $m \neq m_i$ - he generated $x = (h(m))^d$ and $s = x^e$).

If Now chal. sends y ($(h(m_i))$ back to adv., adv breaks full domain by making a forgery on m^* $\Rightarrow$ he can find $s = y^d$ for m^* and sends it to chal

$$s' = y^d \bmod n$$
$$= s^{ed} \bmod n$$
$$= s.$$

Now chall. has found ~~$y = s^e$~~ s from $y = s^e \bmod n$ using only y, e, n

⇒ He broke RSA-hard problem which is not possible

So Full domain hash is secure

(Adversary ~~asks~~ sends m and asks $h(m)$ for all msgs but not $sig(m)$ for all - because his aim is to forge the signature on one such message whose sig was not asked from the chal → to break Full domain hash signature which is his goal).

(challenger should not know private key d because he is trying to break the RSA-hard problem

⇒ if he knows d, he can simply do the ~~inverse~~ inverse for . $d \rightarrow$ trapdoor ?),

↳ & break RSA-hard prob. using d

23/09/22

Full hash domain signature

Deterministic scheme

$S = H(m)^d \bmod n$

(here $H(m)$ is not random. We assumed it is random only for the previous proof).

verify (m, S, e)

if ($S^e \bmod n == H(m)$)

accept

1. Hash is not secondary image resistant

~~pro~~

⇒ Given m, adversary can compute m' such that

$H(m) == H(m')$

$S(m) = H(m)^d \bmod n$

$S(m') = H(m')^d \bmod n$

$= H(m)^d \bmod n$

$= S(m)$

∴ $sig(m) = sig(m')$

Now adversary can do existential ~~for~~ forgery on m by asking signature on m' from challenger and using the same signature for m

H is not secondary-image resistant

For Selective forgery → Adversary chooses m
Finds m' such that H(m) = H(m') since H
is not secondary image resistant
Asks signature on m' = s' from challenger
and sends (m, s') (forged signature
on this choice of message m).

Schnorr Signature Scheme

Given:
Two primes p, q such that $q | p-1$,
group G with generator g is
Schnorr group

Key Generation: $x \in$

PR: x PB: $y = g^x$

sig(m, x):
choose random k and
compute $r = g^k$ (Introducing randomization).

$e = H(m, r)$ (random).
↳ because r is random

$s = k + ex$

(e maybe given by the verifier)

Sends (m, r, s) to verifier

Verify (m, r, s, y) :

Compute $e' = H(m, r)$

we know that

$$s = k + ex$$

$$\Rightarrow g^s = g^k \cdot g^{ex}$$

$$g^s = g^k \cdot (g^x)^e$$

$$g^s = r \cdot (y)^e$$

He knows $g, s, r, y,$ and can find e' as well. So he can verify the signature

if $(g^s == r \cdot y^{e'})$

accept

else

reject

For security

$p = 1024$ bits, $q = 360$ bits

(cannot be less than 1024 bits)

* Schnorr signature scheme is secure

⇒ CMA adversary cannot make existential forgery

Forking Lemma :

If you can make a forgery once, you can make a forgery again (with lower probability).

To prove : If ~~the~~ an adversary can make existential forgery on in schnorr signature scheme then there exists another adversary that can solve DLP hard problem (⇒ can compute x in $y = g^x \bmod p$ using g, y, p)

Proof :

Assume adversary can make existential forgery

⇒ He can find (m, r, e, s)

By Forking Lemma

adversary can make forgery twice on same message

=> Can find (m, e, r, s) and (m, e', r, s').

$s = k + ex$

$s' = k + ex$

Assume hash is ideal
(Full domain hash, Schnorr hash proofs are based on random oracle model)

So $e = H(m, r)$ and

$e' = H(m, r)$ (since H is random).

Here for each message m, there are many signatures and verifier has to accept any of those signatures.

(H is deterministic => adv can compute H himself

Instead assume H is ideal & adversary gets the hash & value from challenger

=> 3 to the same).

$s = k + ex$

$s' = k + e'x$

$s - s' = (e - e')x$

r is same

$\Rightarrow k$ is same

$$x = \frac{s - s'}{e - e'}$$

~~k, s, s', e, e'~~

$\Rightarrow$ can find private key

8/10/22

BLOCKCHAIN

Cryptocurrency

Bitcoin

→ Invented by Satoshi Nakamuro in 2009?
→ Completely decentralized
(no trusted party like RBI, Govt. of India)
→ Came into existence because people lost trust in centralized financial institutes like bank after the Global Financial Crisis in 2009
→ Not reversible
→ Very low fees
→ All transactions are visible to everyone (not the case with banks)

ECDSA in Bitcoin

$y^2 = x^3 + 7$ over Z_p

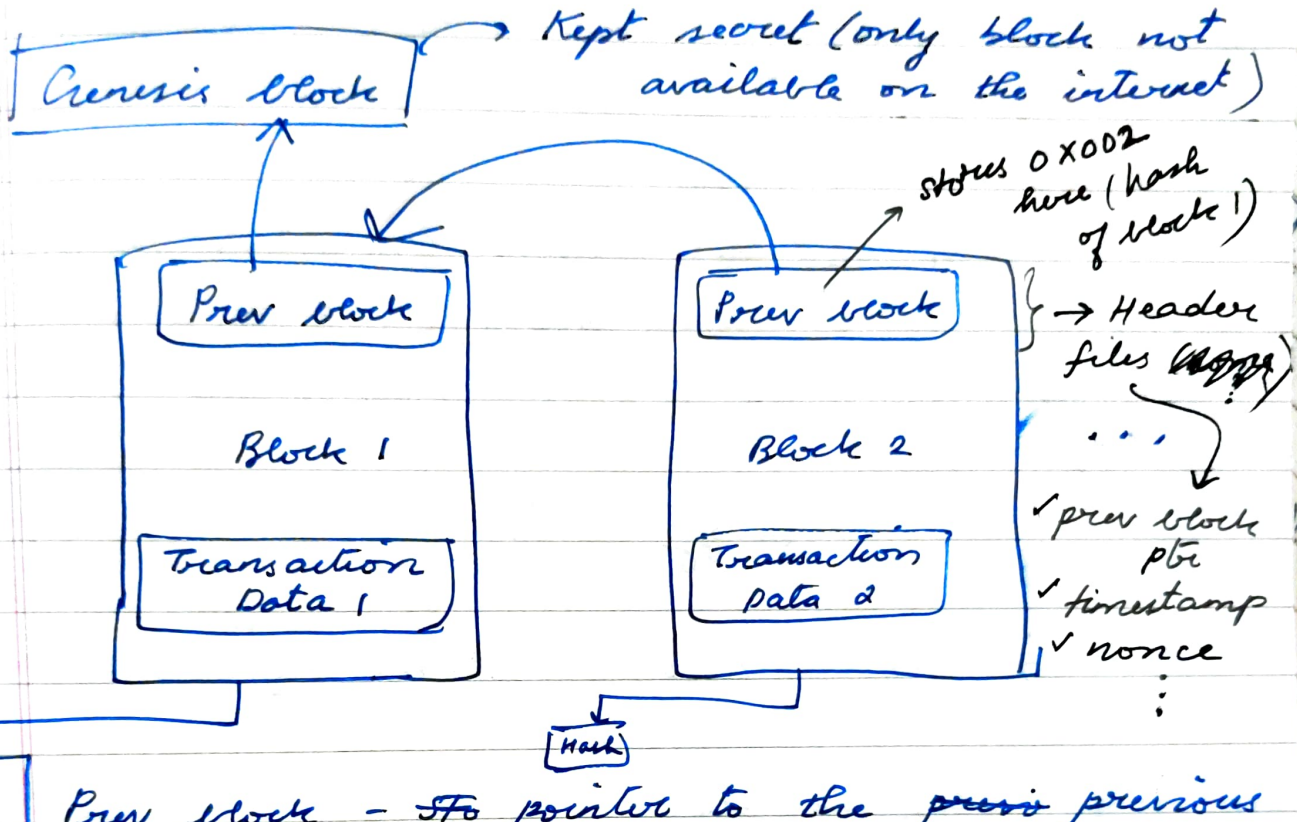
$a = 0$, $b = 7$ in $y^2 = x^3 + ax + b$

$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
(very less)

(p is too, because

→ used in Ethereum also

Blockchain



Prev block - ~~It~~ pointer to the ~~previ~~ previous block's hashed value

If transaction data of block 1 is changed
⇒ its Hash changes
⇒ Prev block value in block 2 changes
⇒ Hash of block 2 changes
and so on

⇒ a change in 1 block changes all the subsequent blocks
⇒ Tamper-resistant

If hash is not secondary image resistant adversary can replace block B_2 with B_2' such that

$$h(B_2) = h(B_2')$$

⇒ He can simply copy the timestamp, prev block hash, ... and easily replace B_2 with B_2'.

Groofy Coin

→ hypothetical cryptocurrency

Steps in doing a transaction in blockchain

- ✓ Create a coin
- ✓ Add details of the transaction (Eg: Pay to Alice)
- ✓ Sender puts a digital signature

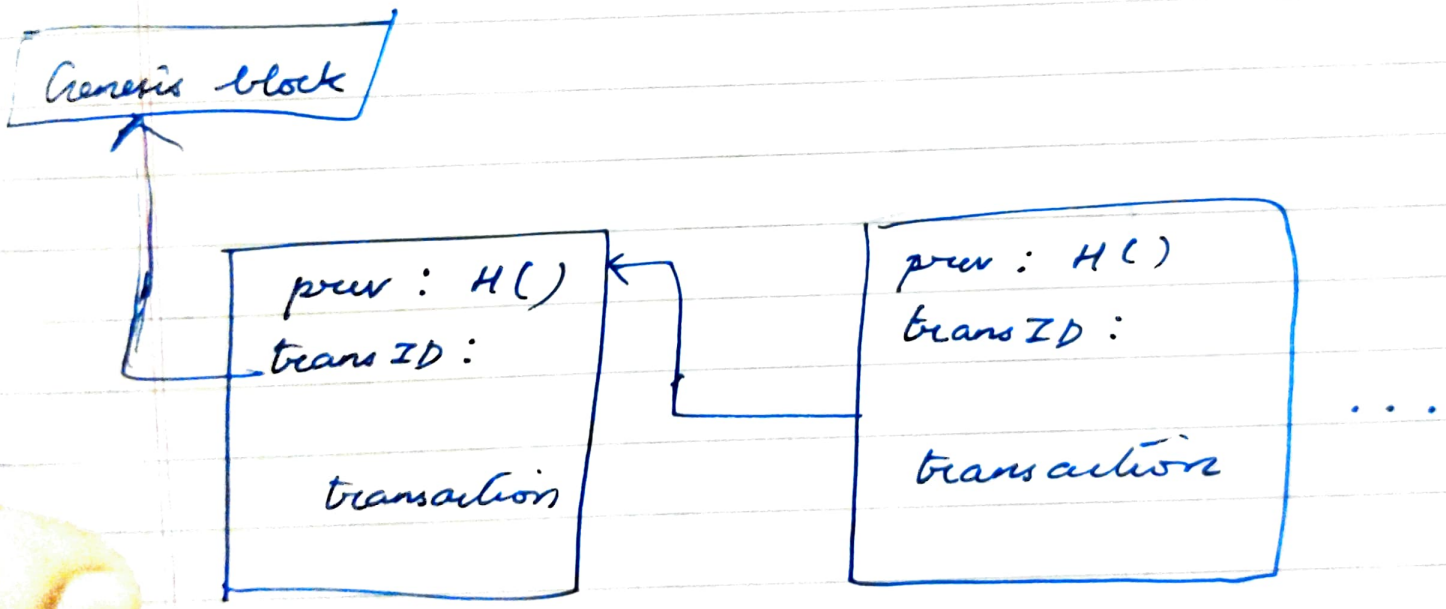
⇒ Prone to double-spending, triple-spending ...

Copy a coin, change the beneficiary
⇒ same coin used to do two different transactions

⇒ arises because all blocks are in arbitrary locations

Do all the transactions in a closed place
⇒ place them all in 1 place
(easy to detect double spending) -
⇓
there will be observers who can detect it!
⇒ This is where blockchain comes into picture - to store all the transactions

Centralised Woofy Coin

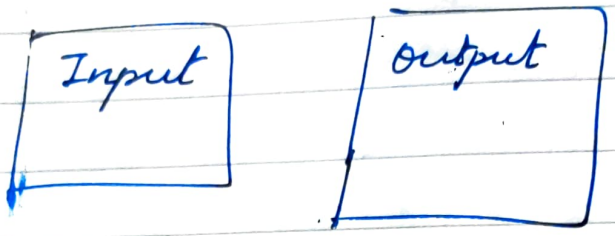


For every transaction, miners check if coin has been spent or not before adding it to the blockchain
⇒ only if the currency received fro source is unspent is the transaction valid

Nonce : Identity of the block

Transaction Hash

??

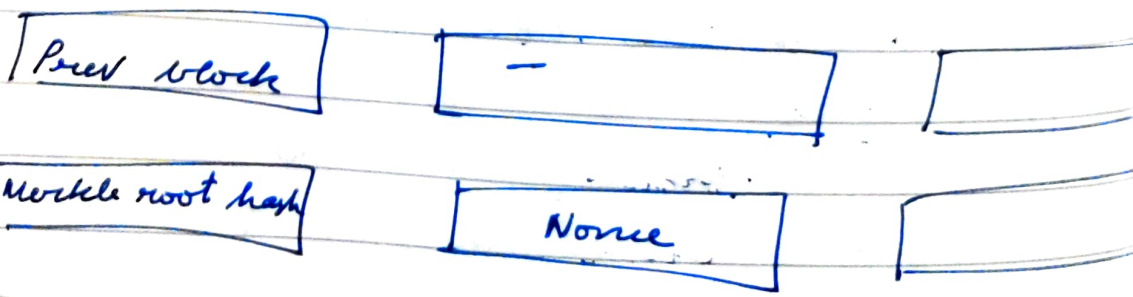


Transaction

Regular
(A gives to B
B gives to C
:)

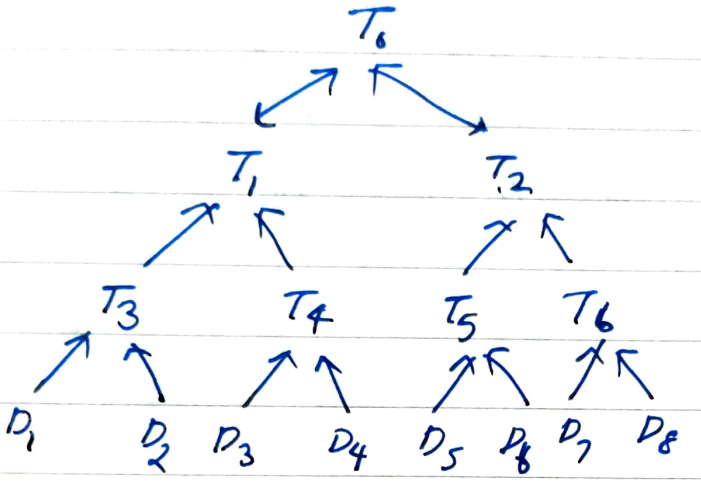
Coin base
(input - nothing
output - some valid destination)
↓
contains miner's reward

Block header



Merkle Hash Tree

→ Binary tree (complete).



$T_3 = H(D_1 \| D_2)$

$T_4 = H(D_3 \| D_4)$

$T_1 = H(T_3 \| T_4)$

$T_0 = H(T_1 \| T_2)$

↳ merkle root hash (stored in header files)

If some block is changed, T_0 also changes and we can detect this

To convince the verifier about my possession of D_2

✓ Normal way - $O(n)$

check one-by-one

↳ all blocks?

✓ only $O(\log n)$ using Merkle Tree

Give D_3, T_3, T_2 to verifier

Verifier finds.

$$H(D_2 \| D_3) = T_4'.$$
$$H(T_3 \| T_4') = T_1'$$
$$H(T_1' \| T_2) = T_0'.$$

If $T_0' = T_0$ (root hash) is correct, then verifier is convinced that D_2 is in the correct block as the sender claims

Drawback:

Bitcoin - no transaction fee and anyone can create a coin/block

⇒ Adversary can add arbitrary useless transactions like

$K_3 \to K_4$ and $K_4 \to K_3$

(no need for these transactions here)

But this wastes others' time

So Satoshi Nakamoto introduced some randomization

⇒ He created a puzzle

(make the problem hard so that even intelligent people find it difficult to

Prob of solving is same for all

solve → to ensure fairness for ~~to~~ x ordinary people).

In block header, ~~nonce~~ everything is almost fixed but nonce can be changed.

Hash puzzle

Say H = SHA 256 hash

Find M such that

$H(M)$ is very small (say ≤ 100)

$H(M) \in \{1, 2^{256} - 1\}$

⇒ This can be done by trying out every M ?

⇒ This problem gives the nonce of the block you want to create & only solving it allows you to create a block

Over time people have been able to solve this faster

⇒ But they wanted only around 1 block for every 10 mins

so they made the problem harder (say $H(M) \leq 50$).

Bitcoin

→ Decentralized Goofy Coin

Consensus ~~Algo~~ (Proof ① of Work)

✓ New transactions - broadcast to all nodes
✓ Nodes ~~that solve the hash puzzle,~~ add the new transaction / block to a transaction pool
✓ special nodes (miners) that solve the puzzle have the right to add a ~~block~~ block from the transaction pool into the blockchain
✓ To motivate miners to solve puzzles they are rewarded with 6.25 BTC (bitcoins)

(1 bitcoin was 5375 $ in ___).

1. Random selection (anyone can solve the puzzle).
2. Incentive to add to longest chain
3. Penalties to those (not?) adding to their chain (solves the puzzle but does not add a block to the chain).

penalty also for adding arbitrary blocks

Reward money = Block subsidy + transaction fee

Rule: After every 210,000 blocks are added, reward money is halved
⇒ takes approx. 4 years

2009 → 50.
late 2012 — 2013

03/01/09 → 50
28/11/12 → 25. (1st halve)
~4 years → 12.5. (2nd halve)
Now → ~~62~~ 6.25.
By 2140 → 0. (64th halve)

21 million bitcoins will be created by then

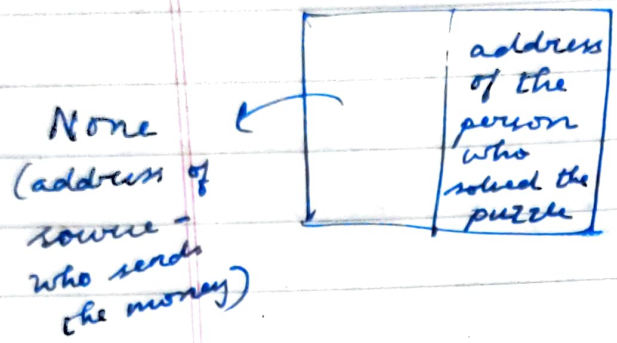
(there will not be any more new bitcoins after that?)
↓
bitcoins will be constant

reward is only the transaction fee

Reward money is the only way money is pumped into the system.

In each block there will be exactly 1 coin base transactions (~~including that~~ for reward money)

Coin base



Disadvantage of bitcoin

⇒ Warming the earth
bcos of Proof of Work
Many companies are working just to solve the hash puzzles - bcos of the huge reward. They continuously compute hash values
Every 10 mins they repeat this on a new system.
Also many companies are working on it simultaneously - but only 1 miner is selected

⇒ uses 1 year of energy requirement of NZ (over 1 year).

↑ Consensus algo

(2) Proof of Stake :

If bitcoin price goes up or down, the one with most stake (coins) is most affected
⇒ why not make him a stronger candidate for miner (he is more probable to be honestly attempting to solve puzzles given he has a large stake involved).

(Eg): Alice – 20% of total coins
Bob – 15%
⇒ P(Alice becoming a miner) = 20%.
P(Bob becoming a miner) = 15%.
One node is randomly chosen as miner from the two

→ Nowadays many people have a large # coins so bitcoin may shift to PoS in the near future (only PoW still).

Bitcoin Wallet

~~Suppose~~ ~~ada~~

Transaction btw 2 nodes happens only using their addresses (computed using public & private key)

⇒ their identities are not involved

⇒ Anyone can claim that the destination address is theirs — ~~we~~ I have to convince the blockchain that it is my address

1. show Private key & verify it

⇒ but private key is revealed

2. put a digital signature on the message determined by the bitcoin using the ~~corresponding~~ Private key (corresponding to the destination address) and give the public key also

⇒ we have to link the Private key, Public key and address (otherwise anyone can put a valid signature even without having the same address).

challenges to be solved in order to spend the BTC you received

Account-based ledger

→ to check validity of transactions, we need to keep track of account balances

→ used in Ethereum

→ not possible in ~~the~~ bitcoin because transactions are not reversible (does not support the notion of accounts).

→ sequence of transactions is also important here → not very efficient

(A $\xrightarrow{25}$ B, A $\xrightarrow{15}$ C

If A has only 30, we must transfer 25 to B but nothing to C → insufficient bal. and cannot transfer 15 to C first).

Bitcoin - transaction based ledger

Book : Arvind Narayanan

Bitcoin & Cryptocurrency Technologies

07/09/22

Bitcoin Transaction

If Alice initially has 25 and wants to transfer 17 to Bob
⇒ she creates a new address and transfers her remaining 8 BTC to that address
(as there is no concept of accounts here)

Multi i/p, multi o/p are also possible

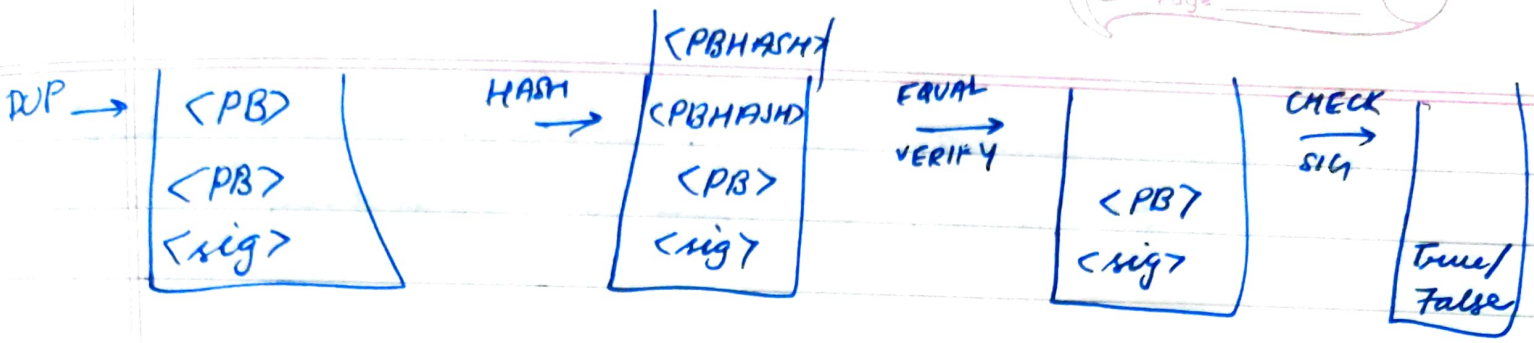
Every address associated with corresponding private key ⇒ can use digital signature to claim the ownership of an address

(sign, PB) included in the output of bitcoin
Response script

Processing takes place using 1 stack
⇒ Not Turing Complete

<sig> <PB> DUP HASH160 <PBHASH> EQUALVERIFY CHECKSIG

- <sig>: Digital signature → pushed onto the stack
- <PB>: Public Key → pushed onto the stack
- DUP: Duplicate top of stack (which is PB)
- HASH160: Apply Hash on top element (PB becomes PBHASH)
- <PBHASH>: address corresponding to PB (given by the hash of PB) → pushed onto the stack
- EQUALVERIFY: verify top 2 elements in stack are = or not
- CHECKSIG: check the signature



Can also use Multisignature

Forking

2 or ~~more~~ more miners solve a challenge & find a block nearly at the same time.

Protocol:

⇒ Both miners broadcast their soln on the network

⇒ nodes accept the solution they receive first and reject the others

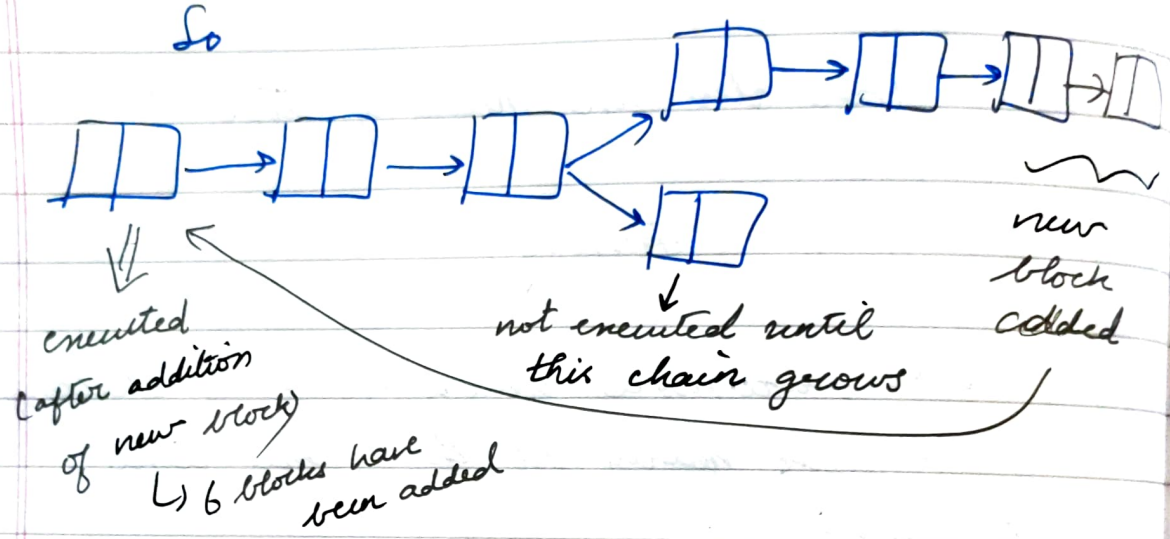
⇒ nodes switch to the longest chain they hear

⇒ network abandons blocks not in the longest chain (orphaned blocks).

longest chain rule?

If some node C is the only miner in the next time, and if its previous block is A, then all B blocks switch to block A (A/B - 2 miners that solved nearly at the same time & broadcast of this solns through the network)

Transaction in a block is executed only when it has 6 forward blocks



Forking :

one - normal

twice by same miners - not normal

Permissionless blockchain - Bitcoin

Permissioned blockchain - Hyperledger

Blockchain is to be used when there is no trusted authority ?

Cryptocurrency - only to transfer money
no other application

Ethereum (Blockchain 2.0) - uses the idea of smart contracts

Ethereum Blockchain

- ✓ Most popular (for developing smart contracts)
- ✓ Account based
- ✓ solidity lang used to write smart contract & Turing - complete
- ✓ Cryptocurrency used - Ether

Idea (by Nick Szabo) → 1994.

Have a code run in a car

Smart contract

⇒ If person doesn't pay EMI for 1 month code issues warning

⇒ another few months code locks the car & people will take the car away after a few days

Smart Contract

→ only one interpretation (the way the code executes)
→ not reversible

Defn: An executable code that runs on blockchain to facilitate, execute & enforce agreement between untrusted parties

I/p - Money / Data

Current major build - 5

Transaction
↙ ↘
money O/p of the function (in the contract?)
↓
also added to the blockchain

Constructor

- ✓ atmost 1
- ✓ can take params
- ✓ same name as contract
- ✓ does not return any data explicitly
- ✓ Typically used to know the owner of a contract / store

Consensus algo (in Ethereum) - Proof of Work (same as bitcoin)

ECC curve - same as bitcoin

Accounts

- External Owned Accounts → owned by people
- Contract account → owned by code

Wallets - a set of 1 or more accounts

Gas and Gas cost

To run a smart contract, you must pay some amount
↓
unit is Gas.

(Eg): 500 Gas.
⇒ Used to prevent DoS attack (running flawed code or program that runs infinitely).

Normal transaction - 21000 Gas

Gas limit : max Gas user is willing to spend

If Gas limit < gas cost, miner collects the gas, but does not change the blockchain (since not enough gas is available)
⇒ unnecessary loss of Gas

gas limit

If gas limit > gas cost, contract is executed and excess gas is returned

IPFS - see ppt.

Instead of storing files directly in blockcha
send it to IPFS, get corresponding hash &
store hash in blockchain
⇒ ~~loss~~ cheaper

transfer()	send()
⇒ transaction fails then exception sent to sender & payment is reverted	⇒ if transaction fails then it returns false but payment not reverted

Types of transactions

1. Money (~~based on ether~~) transfer
   - EOA → EOA
   - EOA → contract
   - contract → EOA
   - contract → contract

   (Involves ether)
2. Execute a fn on a deployed contract
3. Deploy a contract

If there are 100 transactions $T_1, T_2, \ldots, T_{100}$ the order in which they are added to blockchain need not be the same (more amount involved in a transaction ⇒ more miners try to solve the puzzle and add this block).

⇓ So

Can reuse solns since they are available publicly and send a block such that it gets added before the block the miner selects

Front-running attack

07/10/22

~~Bitcoin~~ Wallet does not provide complete anonymity ⇒ Pseudonomity

⇒ Transfer of money is between account numbers not names (or any other identity of the people)

If you repeatedly use the same account no. for many transactions your identity will be revealed

⇒ Use multiple (PR, addresses) and use a different address for ~~any~~ ~~two consecutive~~ successive transactions

PR is lost ⇒ amount is also lost

Reentrancy Vulnerability

Only solidity has fallback fn (no other OOP lang. has)

Fallback fn

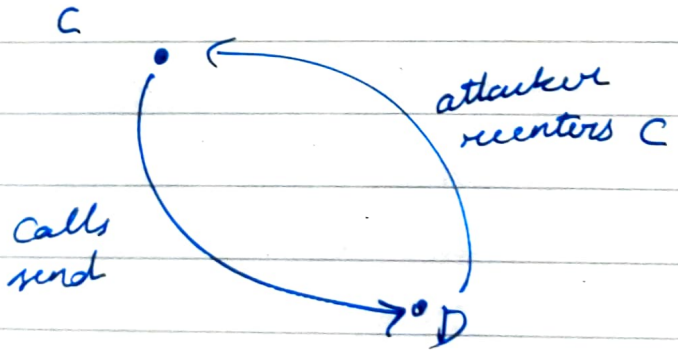
- ✓ method with no name
- ✓ no args, does not return anything
- ✓ a contract has exactly 1 unnamed fn
- ✓ called whenever a ~~you~~ call ~~to~~ an invalid method (non-existent fns) or ~~calling~~ a method without any data.
  ⇒ called whenever contract receives plain Ether (without any data)
  (fallback fn must be marked 'payable' to receive Ether)
- ✓ Fallback fn can also execute complex ops
  only receives money (no other op) - 2300 gas
  Else more gas (for the operation it performs other than receiving Ether)

DAO

⇒ Only real attack on ethereum (others - theoretical?).

⇒ made $150 million but lost $60 million due to their system getting hacked (saved $90 million using hard-forking could have save entire $150M ?).

Reentrancy



⇒ fallback fn reenters the caller fn

```
contract B {
  bool sent = false;
function ping (address c)
{  if (!sent) {
     c.call.value(2)();
   sent = true; }
}
```

```
contract C {
  function() {
  B.ping(this);
  }
```

↳ call fallback fn in C.

Since sent is false, C calls B again & B keeps sending ~~fn~~ & Ethers in an ~~infin~~ in a loop

~~withdraw fn~~

Attack :

fallback fn { Get the balance amount
Withdraw that amount

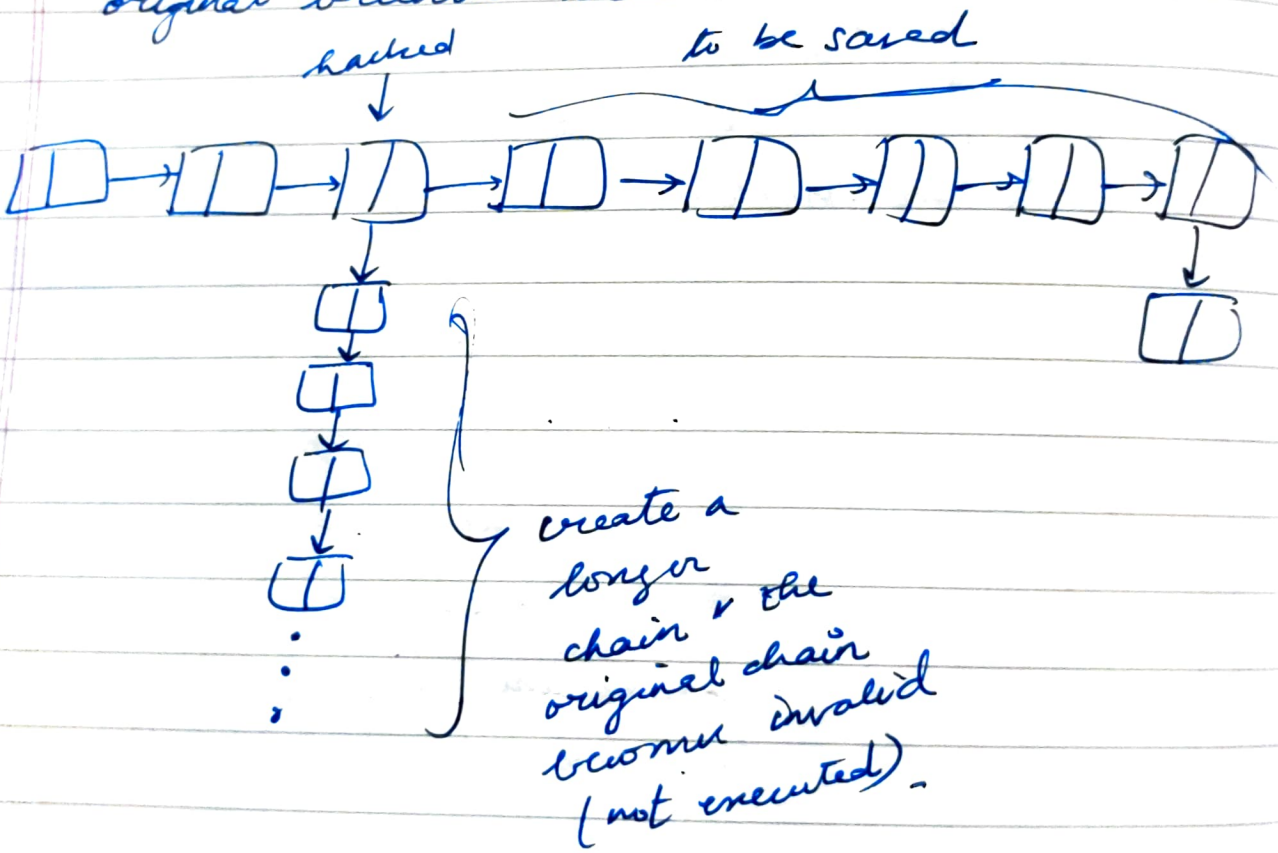
In withdraw of sender
→ first transfer amnt
→ ~~transfer~~ reduce balance

the fn called here will not exist
⇒ calls fallback which again calls ~~repeats~~ the withdraw fn ~~call~~
(So balance not updated but it is transferred infinitely).

Loops until
✓ out of gas
✓ stack limit is reached
✓

Hard-Fork proposal

To save $90M, they solved puzzles & added more blocks to the chain creating another longer chain making the original branch invalid



could have saved all $150M but some do not ~~of~~ agree

Criminal Smart Contracts

⇒ can pay money to a criminal to kill someone without ever revealing my identity

⇒ can also be used to reveal secret info

Protocol:

Divide secret info M into n parts

$M = m_1 \| m_2 \| \ldots \| m_n$

for each m_i, compute

✓ private key $sk_i = h_1(m_i)$

compute corresponding PB using ECC

✓ bitcoin address $a_i = h_2(pk_i)$

✓ symmetric key $K_i = h_3(pk_i)$

✓ ciphertext $e_i = enc_{K_i}(m_i)$

Publishes (n, k, T_{open})

$E = \{e_i\}_{i=1}^{n}$

$A = \{a_i\}_{i=1}^{n}$

Challenge

For the buyer to be convinced of the challenger, challenger# , he ~~random~~ who asks ~~buyer~~ to reveal plaintext of k blocks challenger chooses k blocks randomly out of n blocks (say 3 out of 100) and shows the plaintext, ciphertext corresponding to it

(see ppt)

Pairing

✓ A fn $e: G_1 \times G_2 \rightarrow G_3$

$e(P, Q) = R \qquad P \in G_1, Q \in G_2, R \in G_3.$

✓ generally used with elliptic groups
i.e G_1, G_2 are elliptic groups.

Properties

*. $e(aP, Q) = e(P, Q)^a = e(P, aQ).$

$\Rightarrow e(aP, bQ) = e(P, Q)^{ab} = e(bP, aQ) =$
$e(abP, Q) = e(P, abQ)$

*. $e(P, Q) \neq 1$
(otherwise all powers of $e(P, Q)$ are also 1
$\Rightarrow$ becomes a many-to-one mapping).

*. computing $e(P, Q)$ should only take polynomial time

✓ Can be used for 3-party agreement (to share a PR among 3 of them
2-party - Diffie Hellman Key Exchange Algo)

3-party key exchange/agreement

Consider $e: G_1 \times G_1 \rightarrow G_3$.

G_1 - elliptic group

DLP is hard in additive group also

$\Rightarrow$ Finding a given P, aP is hard.

3 parties $\rightarrow$ X, Y, Z

Private keys $\rightarrow$ a, b, c $(a, b, c \in G_1 ?)$.

Let P be the generator of group G_1 ?

X sends aP to Y, Z

Y sends bP to X, Z

Z sends cP to X, Y.

X $\rightarrow$ knows a, bP, cP but not b, c

He computes $K = e(bP, cP)^a$

$= e(P, P)^{abc}$

Illy Y knows b, aP, cP

he computes $K = e(aP, cP)^b$

$= e(P, P)^{abc}$

Z ~~comps~~ knows c, aP, bP

computes $K = e(aP, bP)^c$

$= e(P, P)^{abc}$

$\Rightarrow$ Now all 3 of them share the common key

Drawback : Pairing does not work on all the groups (works only for some groups like Elliptic groups)

Decisional Diffie Hellman Problem

Given aP, bP, c. It is hard to check whether $c == abP$

But, DDH is not hard in additive groups because of pairing.

if $e(aP, bP) = e(c, P)$ → abP.
then $c = abP$.
else $c \neq abP$.

so DDH is easy in elliptic group.