| | |
|---|---|
| Distributed Trust and Blockchains | Date: *12-September-2019* |
| Instructor: *Sujit Prakash Gujar* | Scribes: Debojit Das<br>Nikhil Pinnaparaju |

# Lecture 5: Bitcoin Ledger and Bitcoin Scripts

## 1 Recap

In the previous lecture by Professor Sujit Gujar, we covered the following topics: Crypto Puzzle, Bitcoin Consensus, and Proof-of-Work and Incentive Engineering. In the next subsections, we provide a brief overview of these topics.

### 1.1 Crypto Puzzle

The objective is to decentralise Bitcoins. This is achieved by creating a crypto puzzle, which when solved will allow the person/organisation that solved it to create the next block in the blockchain of bitcoins. The crypto puzzle is of the form,

$$H(nonce||prev\_hash||tx||tx||...||tx) < target,$$

and to solve the puzzle, you have to find a *nonce* that satisfies the inequality, given a target. For $target = 2^{256}$, any nonce will work. For $target = 1$, only one particular nonce will work, and finding that is hard. Higher the target, easier it is to solve the puzzle. In general, for $target = 2^k$, you need a number with the first $(256-k)$ digits as 0 (in the binary representation). The probability of randomly picking such a number is $\frac{1}{2^k}$. Since the number chosen can be thought of as a *Geometric Random Variable*, the expected number of trials to solve the puzzle is $k$. Keeping the current hash rate ( $\sim 10^7 TH/s$) in mind, $target$ is chosen to be typically of the order $2^{72}$.

In order to prevent centralisation, the difficulty of solving the problem by a particular individual/organisation is kept proportional to their computation power. This is ensured by including their previous transactions in the computation. Since the key is different for all users, chances of solving them are different. So, say Company A owns 40% of all computing power, A will be able to solve the puzzle only 40% of the time.

### 1.2 Bitcoin Consensus

The *Bitcoin Consensus Algorithm* for selecting a random node is as follows:

- Broadcast transactions to all nodes.

- Each node collects new transactions into a block
- A random node gets the chance to write to the blockchain ledger. It broadcasts its block.
- Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)
- Nodes express their acceptance of the block by including its hash in the next block they create

### 1.3 Proof-of-Work and Incentive Engineering

Proof-of-Work helps all users agree on the correct chain. This is done by taking following measures:

- The longest chain is deemed correct.
- Appending to the longest chain is rewarded (Game Theory)
- Appending elsewhere is penalised (Game Theory)

## 2 Overview of Lecture

The following topics were covered in the lecture:

1. Bitcoin as an immutable ledger
2. Bitcoin ledger architecture
3. Bitcoin transactions
4. Bitcoin scripts and important commands
    - Bitcoin scripts for transfer
5. Bitcoin payment scripts

## 3 Bitcoin as an Immutable Ledger

*Immutability* is a property of Blockchains that tells us that once data has been written to a blockchain no one can change it. This ensures that our data has not been altered by sharing between various parties. This is useful for databases of financial transactions. Blockchains are in essence databases with some inbuilt pre-agreed technical and business logic criteria, kept in sync via peer-to-peer mechanisms and pre-agreed rules about what new data can be added. The two key ideas that help us detect tampering easy are: cryptographic functions and blocks.

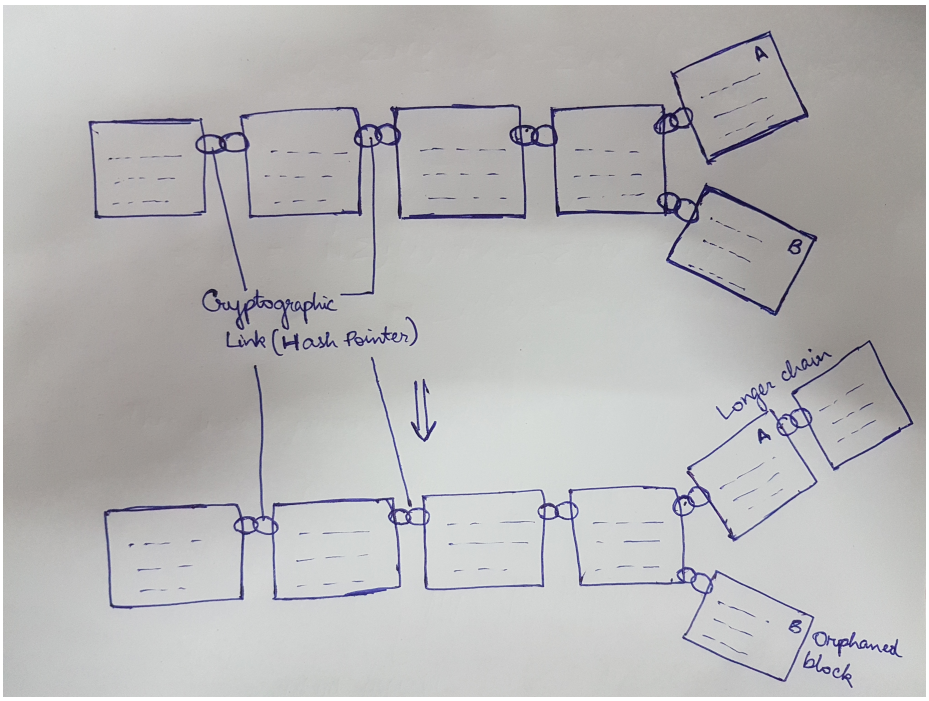


Figure 1: Blockchain as a Ledger

Consider the case where two people solved the crypto puzzle almost simultaneously. Node A and node B are chosen but node A eventually becomes part of the longer chain. Once node B is rendered invalid, it is added to the longer chain. There is no loss of information. However, the person who solved the puzzle and chose B does not receive his 12.5 Bitcoin payment.

# 4 Bitcoin Ledger Architecture

Generally, in banks, individual transactions are added to the ledger one at a time. We cannot maintain bitcoin transactions using such ledger because anyone who wants to determine if a transaction is valid will have to keep track of these account balances and there can be millions of people per block.

Bitcoin uses a ledger that keeps track of transactions. These transactions specify a number of inputs and outputs. We can think of inputs as coins being consumed and outputs as coins being created. For transactions in which new currency is being minted, there are no coins being consumed. Each transaction has a unique identifier and is signed by the owner to authorize the transaction.

# 5 Bitcoin Transactions

Public keys are the identities in bitcoin transactions. Here is an example of a bitcoin transaction:

Amit pays 1BTC to Mohit and Mohit wants to pay 0.5BTC out of this to Rohit. Suppose the transaction fee for the first transaction is 0.2BTC, Amit ends up with 0.2BTC. If Amit also was the miner for the said block, his total reward is 12.7BTC. Amit signs the 1BTC block and pays himself 0.2BTC and the rest goes to Mohit. Now Mohit can sign on the remaining amount to give it to Rohit.

$TransactionFee = InputValue - OutputValue$

# 6 Bitcoin Scripts and Important Commands

Payment of bitcoins work by writing scripts. There is no address to pay to, instead the payer writes a script stating he/she is willing to pay the stated amount to the payee. Following are some important properties of bitcoin scripts:

- Forth-like, which is an old and simple programming language
- stack-based i.e. every instruction is executed exactly once, in a linear manner
- Processed from left to right
- Theres only room for 256 instructions, because each one is represented by one byte
- Supports cryptographic operations
- 15 instructions are currently disabled, 75 are reserved for future

You redeem your coins by executing the scripts in which you received the coins. $< scriptSigKey >< scriptPubKey >$ (concatenated) is how the scripts are executed.

The reserved instruction codes havent been assigned any specific meaning yet, but might be instructions that are added later. There are crypto instructions which include hash functions, instructions for signature verification, as well as a special and important instruction called CHECKMULTISIG that lets you check multiple signatures with one instruction. The following are common Script instructions:

- *DUP:* Duplicates the top item on the stack
- *HASH160:* Hashes twice: first using SHA-256 and then RIPEMD-160
- *PUSHDATA:* Push the number of bytes mentioned after this to stack
- *EQUALVERIFY:* Returns true if the top two elements of the stack are equal. Returns false and marks the transaction as invalid if they are unequal
- *CHECKSIG:* Checks that the input signature is a valid signature using the input public key for the hash of the current transaction, *Verify(Sig,pk,H(Transaction))*
- *CHECKMULTISIG:* Checks that the k signatures on the transaction are valid signatures from k of the specified public keys.

Some other common control statements include IF, NOT IF, NEGATE and AND.

A script may be executed as follows: $< Signature_{MOHIT} >< PubKey_{MOHIT} >$ *DUP HASH160 PUSHDATA[20][MOHITx2018] EQUALVERIFY CHECKSIG.*

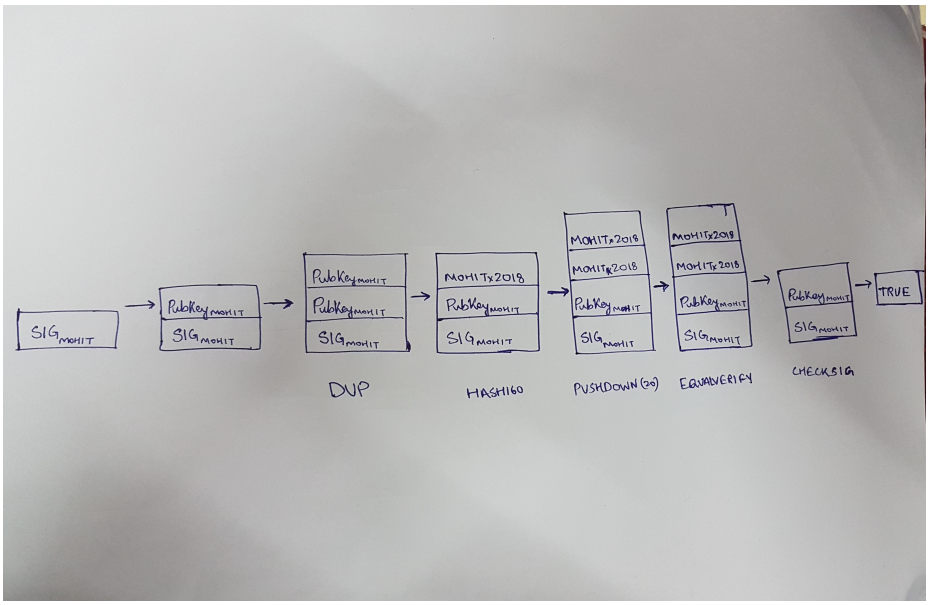


Figure 2: Script Execution

## 6.1 Bitcoin Script for Transfer

**"in"**: [
{
"prev_out" :
{
"hash" : "3be4ac9728a0823cf5e2deb2e86fc0bdb2aa503a91d307b42ba76117d79280260"
"n" : 0 },
<scriptSigKey> "scriptSig" : "30440..."
},
]

**"out"**: [
{
"value" : "1"
<scriptPubKey> "scriptPubKey" : "DUP HASH160 MOHIT BITCOIN ADDRESS EQUALVERIFY CHECKSIG"
}
]

### 6.1.1 Mohit Redeems

**"in"**: [
{
"prev_out" :{
"hash" : "0x1234"scriptPubKey{DUP HASH160 MOHIT BITCOIN ADDRESS EQUALVERIFY CHECKSIG}
"n" : 0 },
<scriptSigKey> "scriptSig" : < Signature$_{Mohit}$¿¡$PubKey_{Mohit}$¿
},
]

**"out"**: [
{
"value" : "0.5"
<scriptPubKey> "scriptPubKey" : "DUP HASH160 ROHIT BITCOIN ADDRESS EQUALVERIFY CHECKSIG"
},
{
"value" : "0.5"
"scriptPubKey" : "DUP HASH160 MOHIT BITCOIN ADDRESS EQUALVERIFY CHECKSIG"
}
]

$< Signature_{Mohit} >$ $< PubKey_{Mohit} >$ DUP HASH160 MOHIT BITCOIN ADDRESS EQUALVERIFY CHECKSIG

# 7 Bitcoin Payment Scripts

Following are some common Bitcoin Payment Scripts:

- *Pay-to-Pub Key*
  - scriptPubKey: $< pubKey > OP_{CHECKSIG}$
  - scriptSig: $< SIG$
  - $< SIG_{MOHIT} >$ $< pubKey_{MOHIT} > CHECKSIG$ This has become obsolete.
- *Pay-to-Pub Hash.* This is the most commonly used one.
- *Pay-to-Script Hash.*
  - Often used in organisations with multiple people in authority.

- Only $t$ out of $n$ signatures are needed.
- Useful when a transaction needs to be made urgently but the availability of all people can not be guaranteed at all times.

# References


[1] Chapter 2: How Bitcoin Achieves Decentralisation, Bitcoin and Cryptocurrency Technologies,by Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

[2] Chapter 3: Mechanics of Bitcoin, Bitcoin and Cryptocurrency Technologies,by Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

[3] https://en.bitcoin.it/wiki/Script