Distributing Trust and Blockchains		Date:	*16th September, 2019*
Instructor: *Sujit Prakash Gujar*		Scribes:	Vineeth Chelur, Hemanth Vemuri

Lecture 6: Bitcoin Mechanism and Scripting

1 Recap

1.1 Bitcoin Scripts

The Bitcoin scripting language has the following properties

- Based on Froth (a stack-based scripting language)
- Inbuilt support for cryptographic functions
- Absence of looping constructs
- 8-bit instructions $\implies$ A total of 256 instruction out of which 75 are reserved for future use and 15 are banned
- Two types of instructions - data instructions and OP_CODE (Acts on the value(s) on top of the stack and puts the result on the stack as well)
- Common OP_CODE instructions are
  - DUP: Duplicates the data on the top of the stack
  - HASH160: RIPEMD-160(SHA1(Top of the stack)) - Gives the bitcoin address if the top of the stack is the public key
  - PUSHDATA: Push the number of bytes mentioned after this to the stack
  - EQUALVERIFY: Compares the top two elements of the stack and returns true only if they are equal
  - CHKSIG: Verify(Signature, Public Key, Hash(Transaction))
  - CHKMULTISIG: Verifies if at least t signatures out of n are valid (Multiple public keys are specified)
  - Other conditional statements like IF ELSE but no loops to prevent submission of arbitrary scripts with infinite loops, hence protecting the miner
- scriptPubKey consists of taking the public key on the stack, duplicating it, hashing it and comparing it to the hash of the public key that the transaction was destined for
- scriptSig contains a signature and a public key, which is used to prove that the transaction is from the owner and hence allows for spending of bitcoins

1.2 Bitcoin payment methods

- Pay-to-Public-Key-Hash (P2PKH)
  - Basic and most common form of making a transaction in the bitcoin network
  - The transaction specifies an address to which the payment is to be sent, in the form of the hash of the public key
  - Validation of the script is done as shown below and can lead to either True or False <Sign><PubKey> DUP HASH160 <BitcoinAddress> EQUALVERIFY CHECKSIG

- Pay-to-Public-Key (P2PK)
  - Very similar to Pay-to-Public-Key-Hash
  - The public key is presented as a point on an elliptic curve and therefore hashing and duplication operators are not needed
- Pay-to-Script-Hash (P2SH)
  - An extension of the multi-signature idea but reduces the burden on the Bitcoin infrastructure in terms of storage requirements by storing the hash of the details of the transaction
  - Removes complexity from the sender, so the recipient can just specify a hash that the sender sends money to and it can only be redeemed by the script having the same hash
  - P2SH output scripts are small and all the complexity is pushed to the input scripts

2 Applications of Bitcoin Scripts

Now that we understand how Bitcoin scripts work, let's take a look at some of the powerful applications that can be realized with this scripting language. We have many applications of Bitcoin scripts that in a way justify the complexity of having the scripting language instead of just specifying the public key.

2.1 Escrow Transactions

2.1.1 Usage

Escrow Transactions are used when the two parties between which the transaction takes place don't trust each other. Let us assume that a customer is willing to pay only after he/she has received the goods and the merchant doesn't want to send the goods until he/she has been paid. Escrow transactions introduce a third trustworthy party who verifies the whole transaction and then confirms it.

2.1.2 Implementation

- An escrow transaction can be implemented using the MULTISIG data instruction. M-of-N MULTISIG tells us that at least M signatures are needed out of N for the transaction to be valid. 2-of-3 is the most common safety measure for multi-signature transactions (customer, merchant, judge)
- Let the customer initiate an escrow transaction (2-of-3 MULTISIG) where he/she sends some coins and specifies that they can only be spent if any 2 of the 3 parties (customer, merchant, judge) sign the transaction.
- The transaction is included in the blockchain and the coins are now held in escrow among the 3 parties
- The merchant can now safely send the goods to the customer and in the case when both the merchant and customer are honest, they sign the escrow transaction and the judge need not get involved. This way, the customer receives the goods and the merchant receives the money.
- If either the customer or merchant is dishonest, then the judge gets involved. Say, the customer lies about not receiving the goods or the merchant lies about sending the goods, the judge will decide which of the 2 parties deserves the money. Based on this, the judge signs the escrow transaction depending on who gets the money. Hence, the judge is present to resolve a conflict, if it ever occurs.

.

2.2 Green Addresses

The green address is a third party trust trick and can help resolve most problems related to the need of wait for confirmations. It also solves the problem, that one can not go and check the block chain (for any reason, example: he does not have an internet connection) for whether that transaction is added or not. To solve this problem of being able to send money using Bitcoin without recipient being able to access the block chain, green address were introduced.

- Green address is also just a Bitcoin address. Essentially, instead of directly paying to merchant, we ask the green address to pay to merchant.
- Here green address is trusted by both merchant and customer. It guarantees that green address will not double spend this money. Hence we can see that this is not a Bitcoin enforced guarantee rather it is a real world guarantee that uses Bitcoin scripts.

However, the concept of green address has a few drawbacks,

1. Loss of anonymity; (rather no anonymity at all)
2. Redundant transactions in the Blockchain network
3. The entire concept of green address depends on the trust people have on the the address and hence the green address has to be trusted by everyone to be useful.

In fact, the two most prominent online services that implemented green addresses were Instawallet and Mt.Gox, and both ended up collapsing. Today green addresses aren't used very much as people have become quite nervous about the idea and are worried it puts too much trust in the green address.

2.3 Efficient micro-payments

2.3.1 Usage

A third example of Bitcoin scripts is a way to do efficient micro-payments. Say that Alice is a customer who wants to continually pay Bob small amounts of money for some service that Bob provides. Creating a Bitcoin transactions every time the customer pays a small amount is not feasible as it creates too many transactions, and the transactions fees add up. If the value of each one of these transactions is on the order of what the transaction fees are, customer is going to be paying quite a high cost to do this.

2.3.2 Implementation

Another way to do this is, to combine all these small payments into one big payment and pay at the end. We can do this efficiently by using micro-payments.

1. Customer will start a MULTISIG transaction that pay the maximum amount he would ever need to spend to an output requiring both customer and merchant to sign to release the coins.
2. After every service(that merchant provides), customer will sign the transaction paying the amount of service used till now from the start.
3. Note that the input script used in all these transactions will all point towards the same hash i.e. same previous transaction is used as input to pay every time. Even though all these transactions are technically a double spend attack, only one of these transaction will be valid after signing.

Hence when the customer stops using the service, then the merchant will sign the last transaction(it will have largest amount) sent by the customer and publish it to the block chain.

2.4 Lock Time

In the above mentioned scenario, consider the case when the merchant never signs the last transaction. In this case, not only will the merchant will not get money for the services he provided, but also the customer will lose the full value that she paid at the beginning as the coins will just sit there in escrow forever. To avoid this, lock time was introduced.

Definition 1 (Lock Time)**.** *Lock time is the time at which a particular transaction can be added to the Blockchain. This is the earliest time that miners can include the transaction in their hashing of the Merkle root to attach it in the latest block to the Blockchain.*

Hence in the above example, even before the micro-payment protocol can even start, customer and merchant will both sign a transaction which refunds all of customer's money back to him/her, but the refund is "locked" until some time in the future. This guarantees that if the merchant is malicious and doesn't sign the last transaction in the micro-payment protocol until a certain time in the future, the customer can still get back the money he put in the escrow at the beginning.

2.5 Smart Contracts

The general term for contracts like the ones we saw in this section is smart contracts. These are contracts for which we have some degree of technical enforcement in Bitcoin, whereas traditionally they are enforced through laws or courts of arbitration. It's a really cool feature of Bitcoin that we can use scripts, miners, and transaction validation to realize the escrow protocol or the micro-payment protocol without needing a centralized authority.

3 Bitcoin Blocks

In this section, we will discuss the need to group multiple transactions into a block and then see the structure of these blocks.

1. If miners had to come to consensus on each transaction individually, the rate at which new transactions could be accepted by the system would be much lower.

2. Also, a hash chain of blocks is much shorter than a hash chain of transactions would be, since a large number of transactions can be put into each block.

Hence we can see that creating blocks is basically an optimization by creating a single unit of work for the miners.

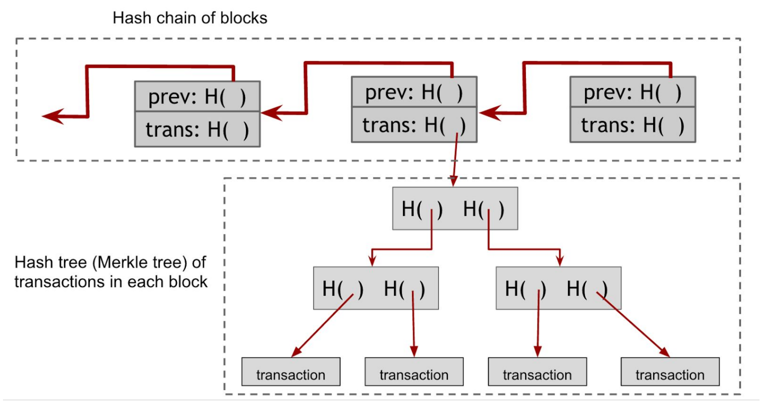


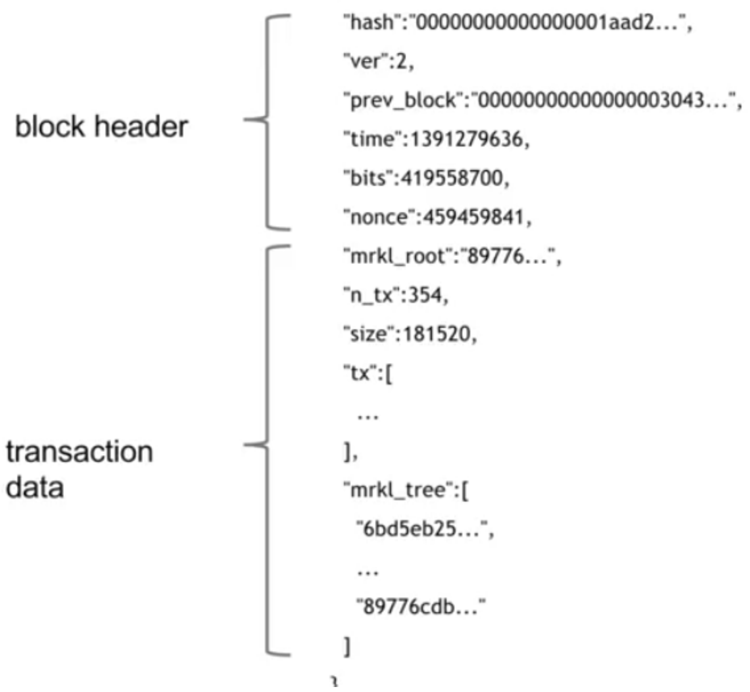
Figure 1: Bitcoin Blockchain structure containing two different hash structures.

3.1 Block Structure

Bitcoin block chain is a clever combination of two different hash-based data structure as shown in the figure.

1. Hash chain of blocks in which each block has a block header, a hash pointer to some transaction data and a hash pointer to the previous block in the sequence.
2. Per-Block Merkle tree of all of the transactions that are included in that block

Merkle trees helps us to have a digest of all the transactions in the block in an efficient way and can find transactions in logarithmic time. Figure 2 gives a high level view of the Bitcoin Blockchain. Now, lets discuss about the low level structure of each block in the chain of blocks. A block consists of the following,



```
"hash":"00000000000000001aad2...",
"ver":2,
"prev_block":"00000000000000003043...",
"time":1391279636,
"bits":419558700,
"nonce":459459841,
"mrkl_root":"89776...",
"n_tx":354,
"size":181520,
"tx":[
  ...
],
"mrkl_tree":[
  "6bd5eb25...",
  ...
  "89776cdb..."
]
}
```

Figure 2: Low level Bitcoin Block Structure with "block header" and "transaction data"

- Block header
  - Header mainly contains the metadata for the block. It contains the hash of the block, hash of the previous block, a nonce field that miners can change, time field (time stamp) and a "bits" field which is an indication of how difficult this block was to find.
  - Only the headers are hashed during mining the block. So, to verify the chain of blocks we just need to look at headers.
- Hash pointer to Transaction data
  - The only transaction data that's included is the root of the transanction tree - the "mrkl_root" tree.
  - It contains a long list of transactions, number of transactions, size of the block.
  - It contains all the hashes in the mrkl tree field which are arranged in tree structure. This helps in efficiently proving which transactions are in the block.
  - There is a special transaction in the Merkle tree in the form of a "Coinbase Transaction". We will discuss these transactions in the next section.

4 Coinbase Transactions

The miner gets his reward for mining a block in two different forms, the block reward and the transaction fee. The block reward is called Coinbase.

Definition 2 (Coinbase Transaction)**.** *The transaction that the miner is allowed to place inside each of his mined block as his block reward is called the Coinbase transaction.*

4.1 Properties

1. This is where creation of new coins in Bitcoin happens. It is analogous to CreateCoin in TrustMe coin discussed in earlier classes.
2. Key difference between a Coinbase and a normal transaction are:
    - it always has a single input and a single output
    - the input doesn't redeem a previous output and thus contains a null has pointer since it is creation of new bitcoins.
    - the output value is the miner's revenue from the block and consists of: a flat mining reward (constant and set by the system), and the transaction fees collected from every transaction included in the block.
    - there is a special "Coinbase" parameter, which is completely arbitrary i.e. miners can put whatever they want in there.
3. The flat mining reward part of the output value of each Coinbase transaction is currently at 12.5 Bitcoins. This value started at 50 Bitcoins in 2009 and halves every 210,000 blocks (or 4 years)
4. As a political commentary on the motivation for starting Bitcoin, the genesis block refers a story in the Times of London newspaper involving the Chancellor bailing out banks in its Coinbase parameter.

5 The Bitcoin Network

5.1 Introduction

- The bitcoin network is a peer-to-peer payment network that operates on a cryptographic protocol. Users send and receive bitcoins, the units of currency, by broadcasting digitally signed messages to the network using bitcoin cryptocurrency wallet software
- The network changes over time and is dynamic due to nodes entering and leaving at any time. There isnt an explicit way to leave the network. Instead, if a node hasn't been heard from for three hours, other nodes start to forget it. In this way, the network gracefully handles nodes going offline
- It runs over TCP and has a random topology, where each node peers with other random node

5.2 Joining the network

- To join the network, you need to know how to contact a seed node, which is a node that is already on the network. There are a few different ways to look up lists of seed nodes
- A special message is sent to the seed node where it is asked to tell the addresses of all the other nodes in the network that it knows about
- Repeat the above process with the new nodes that you learn about as many times as you want. Choose the nodes you want to peer with and you become a fully functioning member of the network
- You will have to answer similar queries as other nodes once you become a member of the network and if you are inactive for more than 3 hours, the network slowly forgets you

5.3 Gossip Protocol

- To maintain the blockchain, transactions have to be published and the entire network has to hear about it
- This happens through a simple flooding algorithm, called the Gossip Protocol
- Suppose if Alice wants to pay Bob some money, her client creates and her node sends this transaction to all the nodes it's peered with
- Each of those nodes executes a series of checks to determine whether or not to accept and relay the transaction. If the checks pass, the node in turn sends it to all of its peer nodes
- Nodes that hear about a transaction put it in a pool of transactions which they've heard about but that aren't on the blockchain yet
- If a node hears about a transaction that's already in its pool, it doesn't further broadcast it. This ensures that the flooding protocol terminates and transactions don't loop around the network forever
- Every transaction is identified uniquely by its hash, so its easy to look up a transaction in the pool

5.4 Options for joining

i Can join as a thin client

ii Or as a miner or a full node

iii About 90% of nodes run Core Bitcoin (C++) (serves as a bitcoin node and provides a wallet to verify bitcoin payments)

iv Other implementations running successfully are:

   (a) BitcoinJ (Java)
   (b) Libbitcoin (C++)
   (c) btcd (Go)

v **Original Satoshi Client**: Bitcoin Core is free open-source software that serves as a bitcoin node (the set of which form the bitcoin network) and provides a bitcoin wallet which fully verifies payments

6 Thin or Simple Payment Verification (SPV) Clients

Definition 3 (Lightweight node)**.** *A lightweight node, also called thin or SPV client does not download the whole blockchain, instead it downloads the block headers only to validate the authenticity of the transactions that it cares about and proof of work*

- Majority of the bitcoin network consists of SPV clients and most bitcoin wallets maintain a lightweight node
- They use the Simple Payment Verification (SPV) method to verify the transaction
- As per the SPV protocol, lightweight nodes can only verify the proof of work by the miner but cannot verify the transactions inside the block as it only stores the header of a block
- The full nodes allow the lightweight nodes to connect to the bitcoin network and also allows for verification of transactions and thus they are essential for functioning of a SPV node

Advantages: The cost savings of being a lightweight node are huge. The block headers are only about $\frac{1}{1000}$ the size of the block chain. So instead of storing a few tens of gigabytes, its only a few tens of megabytes. Even a smart phone can easily act as a lightweight node in the Bitcoin network

Disadvantages: If a lightweight node is not connected to its trusted full node, then it cannot verify its transaction and hence may accept invalid or used bitcoins. Lightweight wallets typically send addresses to a trusted third party and receive wallet balance and history. This allows the trusted third party to spy on all the users past and future transactions.

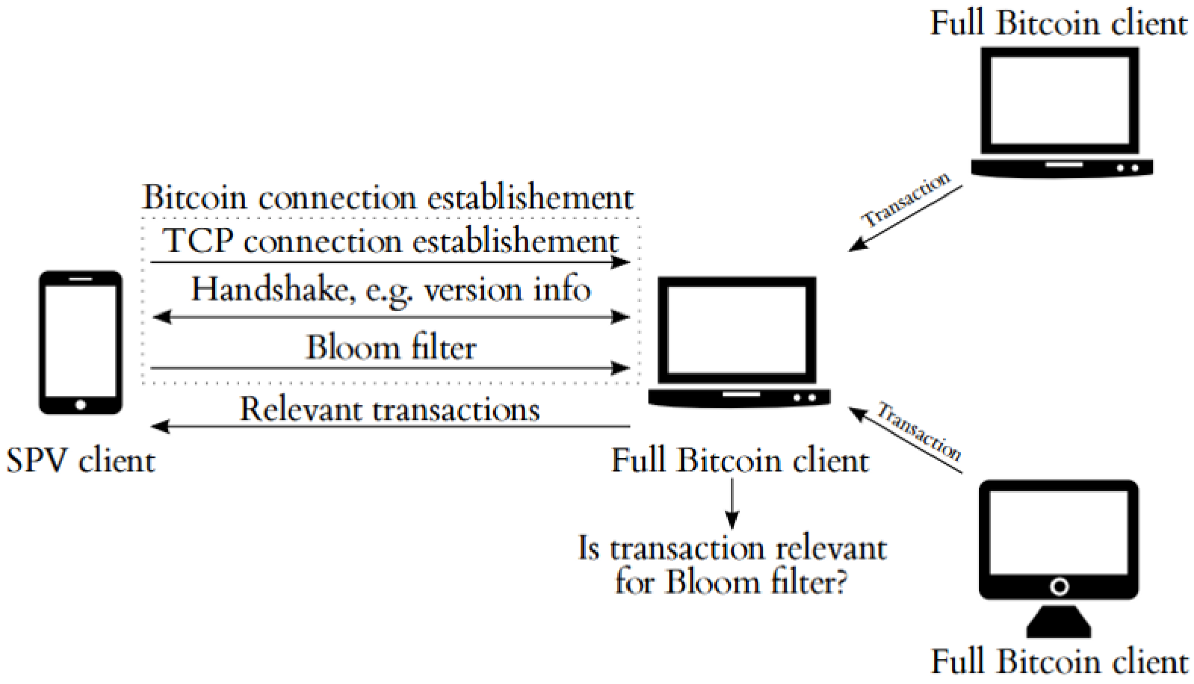

Full Bitcoin client
Transaction
Bitcoin connection establishement
TCP connection establishement
Handshake, e.g. version info
Bloom filter
Relevant transactions
SPV client
Full Bitcoin client
Transaction
Is transaction relevant for Bloom filter?
Full Bitcoin client

Figure 3: Sketch of the operation undergone by an SPV client. SPV clients connect to a full Bitcoin node, which only forwards to the SPV clients the transactions relevant to their Bloom filters

7 Role of a Bitcoin Node

Definition 4 (Bitcoin Node)**.** *A full node or Bitcoin node is a program that fully validates transactions. Almost all full nodes also help the network by accepting transactions from other full nodes, validating those transactions, and then relaying them to other full nodes*

- Most full nodes also serve lightweight clients by allowing them to transmit their transactions to the network and by notifying them when a transaction affects their wallet
- If not enough nodes perform this function, clients won't be able to connect to the peer-to-peer network and will have to use centralized services instead. Hence, full nodes maintain the peer-to-peer network by relaying transactions

Full nodes relay transactions only after performing the following checks

i **Check** if the transaction inputs are in UTXO (Unspent Transactions Output)

ii **Ensure** that it is not a double spending transaction

iii **Execute** the output script of input transactions along with ScriptSig. The transaction is added to the transaction pool only if it is true

iv **Broadcast** the transaction only if it is true in the previous step. Else if the transaction is repeated, then do not broadcast the transaction to avoid flooding

8 Unspent Transaction Output

During a transaction we are both destroying old bills of various denomination (in the form of inputs) and creating new bills of other denominations (in the form of outputs). UTXOs are created when our outputs are more than our inputs.

Definition 5 (UTXO)**.** *Unspent Transaction Output (UTXO) is the set of all output transactions that can be used as inputs to new transactions.*

- All full nodes need the UTXO set in order to validate new transactions hence the UTXO set's size is directly proportional to how expensive it is to validate new transactions.
- The size of UTXO decreases when the inputs are more than outputs. However throughout Bitcoin's history, the size of UTXO has been consistently increasing as shown in figure 4.

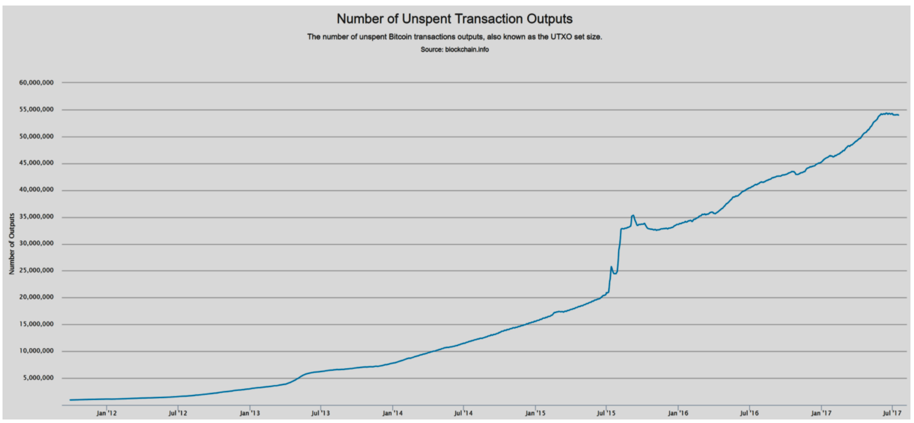


Figure 4: Evolution of UTXO set size through bitcoin history.

References

[1] Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton University Press.

[2] `https://en.wikipedia.org/wiki/Bitcoin_Core`

[3] `https://en.wikipedia.org/wiki/Gossip_protocol`