Distributed Trust and Blockchains		Date:	*28 August 2018*
Instructor: *Sujit Prakash Gujar*		Scribes:	M Chitra, Kunal Garg, Shipta Golechha

Lecture 8: Bitcoin Ledger Architecture and Transactions

1 Recap

In Lecture 7, we discussed about how can we select a random node which will propose the next block and add that block to the Bitcoin Blockchain. We also calculated the probability of a successful double-spend-attack by attacker and how many blocks should a merchant for conformation before offering the services in exchange of Bitcoins. In Lecture 8, we revised all the content covered form Lectures 1-7 and started discussion about Bitcoin Ledger Architecture.

2 Introduction

- *Trust* is an arrangement where by a person(a trustee) holds property as its nominal owner for the good of one or more beneficiaries.
- *Blockchain* is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block consists of footprints of the previous blocks. It is maintained by users over a network, so it is also considered as a distributed ledger.
- *Bitcoin* is the first decentralized digital currency, an application of the blockchain network in which the network is peer-to-peer and transactions take place between users directly without an an intermediary such as, a central bank or single administrator.
- *Smart Contract* is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. In general, it is a piece of code which is stored in the blockchain network and defines the conditions to which all parties using contract agrees.

3 Evolution in currency system

1. *Barter system* : System of exchange where goods or services are directly exchanged for other goods or services without using a medium of exchange such as money.
2. *Credit system* : System for allowing people to purchase things on credit i.e., borrowed money that you can use to purchase goods or services.
3. *Cash system* : Currently we use fiat currency and Banks are responsible for maintaining the ledger. Issues in Banking system: No anonymity or Decentralization.
4. *Digital Cash* : This was proposed by David Chaum in 1982, who introduced the idea of *Blind Signature* which is a form of digital signature in which the content of a message is disguised (blinded) before it is signed. The resulting blind signature can be publicly verified against the original, unblinded message in the manner of a regular digital signature [3]. It solved the issue of anonymity but could solve issue of double spending.
5. *HashCash* : It was proposed by Adam Back in 1997. HashCash is a proof-of-work algorithm that requires a selectable amount of work to compute, but the proof can be verified efficiently.It was used to limit email spam and denial-of-service attacks. Currently, Bitcoin also uses this for its mining function

   Bitcoin is able to provide anonymity, prevent double spending and is also decentralized.

4 Hash Functions

A hash function is a function that can be used to map data of arbitrary size to data of a fixed size. The value of the hash is unique to the hashed data. Any change in the data, even changing or deleting a single character, results in a different value.

4.1 Properties

- *Deterministic* : It should always give same output for same input.
- *Easily Computable* : Given x, $H(x)$ should be very easy to compute where $H()$ is a hash function.
- *Pre-image resistant(Hiding)* : Given a hash h, difficult to compute message such that $h = H(m)$, also called one way function where $H()$ is a hash function and m is message.
- *Second pre-image resistant* : Given a message x and hash function $H()$, it is difficult to compute message y such that $H(x) = H(y)$ where $x \neq y$.
- *Collision resistant* : Difficult to find two messages $x \neq y$ s.t. $H(x) = H(y)$ where $H()$ is a hash function.

The number of distinct values Q we need to try to crack hash function of M bits with probability of success is greater than 1/2 are proportional to square root of M. i.e. $Q \geqslant 1.17\sqrt[2]{M}$
Thus to achieve 128 bit security against collision attacks, hashes of length at-least 256 is required

4.2 Digital signatures

Digital Signature is a mathematical technique to validate authenticity and integrity of a message, software or a digital document.

4.2.1 Properties of digital signatures

1. Only you should be able to make your signature, but everyone should be able to verify.
2. Signature should be tied to a particular document so that the signature cannot be used to indicate your agreement or endorsement of a different document.

4.2.2 Algorithms

1. *RSA :* is based on the fact that finding the factors of a large composite number is difficult: when the integers are prime numbers.
2. *ElGamal signature scheme :* is based on the difficulty of computing Discrete logarithms.
3. *DSA* is a sort of hybrid of the ElGamal and Schnorr signature schemes.
4. *ECDSA :* Bitcoin uses a particular digital signature scheme thats called the Elliptic Curve Digital Signature Algorithm (ECDSA). Its an update of the earlier DSA algorithm adapted to use elliptic curves

5 Simple Cryptocurrencies

5.1 Goofy Coin

Goofy can create new coins whenever he wants. Whoever owns a coin can transfer it on to someone else by signing a statement that saying, Pass on this coin to X (where X is specified as a public key). Anyone can verify the validity of a coin by following the chain of hash pointers back to its creation by Goofy, verifying all of the signatures along the way. GoofyCoin does not solve the double−spending attack and therefore its not secure.

5.2 Scrooge Coin

Scrooge Coin is another cryptocurrency like GoofyCoin which solves the problem of double spending introduced in GoofyCoin by maintaining an *append-only ledger* which contains the history of all the transactions that have happened.

To implement append only functionality, Scrooge can build a block chain, which consists of a series of data blocks, each with one transaction in it. Each block has the Id of a transaction, the transaction's contents, and a hash pointer to the previous block. Scrooge will digitally sign the final hash pointer and a transaction is valid only if it is in the block chain signed by Scrooge. Anybody can verify that transaction was endorsed by Scrooge by checking Scrooge's signature on the block that records the transaction. Scrooge will have sure that he doesn't endorse a transaction that attempts to double spend an already spent coin.

The problem with Scrooge Coin is *centralization* as Scrooge has too much influence. He can't create fake transactions, because he can't forge other people's signatures. But he can stop endorsing transactions from some users,denying them service and making their coins unspendable. He can create new coins for himself whenever he wants and can also start charging the transaction fee to publish transactions.

6 Distributed Consensus Protocol

There are n nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:

- It must terminate with all honest nodes in agreement on the value.
- The value must have been generated by an honest node.

Impossibility Results in Consensus

1. *Byzantine Generals Problem* - The nodes which are malicious or faulty and try to prevent the consensus of a distributed system are called byzantine nodes. It is proven that it is impossible to achieve consensus if one-third or more byzantine nodes are present.
2. *Fischer-Lynch-Paterson* - It showed that under some conditions, which include the nodes acting in a deterministic manner, it is impossible to reach consensus even a single faulty process.
3. *Paxos* - Paxos makes certain compromises. On one hand, it never produces an inconsistent result. On the other hand, it accepts the trade-off that under certain conditions, albeit rare ones, the protocol can fail to make any progress.

7 Bitcoin Consensus Algorithm

Bitcoin violates many assumptions built into the traditional model of consensus, so it works much better in practice than in theory. It introduces the idea of incentives and does away the with the notion of a specific starting point and ending point for consensus so consensus can happen over a long period of time. The algorithm is as follows :

1. New transactions are broadcast to all nodes.
2. Each node collects new transactions into a block.
3. In each round a random node gets to broadcast its block.
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures).
5. Nodes express their acceptance of the block by including its hash in the next block they create.

8 Malicious Things A Random Node Can Do?

1. *Stealing Bitcoins* - A random node 'X' cannot steal the bitcoins belonging to another user as to create a valid transaction that spends the coin of other user, it would require to forge the owner's signature which is impossible to do so as the secret key used for digital signature is retained by the owner only.
2. *Denial Of Service* - If a random node 'X' does not include any transactions originating from user 'Y', in the block node 'X' proposed in the chain then some other honest node will include the transactions of 'Y' in it's proposed block to be added in chain.Thus the transaction will just appear a bit late but wouldn't be denied.
3. *Double Spending* - A node 'X' can double spend if somehow it creates two transactions (one in which the node is paying to the merchant and second in which node is paying the same coins to itself) and broadcasts them to two different parts of network. If the block containing the 1st transaction is seen by merchant he will conclude that 'X' has paid him and allows 'X' to download the software. But this if this block doesn't get included in the main chain (becomes orphan block), but the block with 2nd transaction makes it to the main chain, then it would be a successful double spent attack as he got the service without paying money. We will see why this can't happen in 11.

9 Random node selection criteria

We select random node by selecting nodes in proportion to a resource that we hope that nobody can monopolize. In Bitcoin, we select nodes in proportion to their computing power i.e we use proof-of-work system. Nodes are allowed to compete with one another by using their computing power, which will result in nodes automatically being picked in proportion to that capacity. Bitcoin achieves proof of work using *hash puzzles.*

To create a block, the node that proposes that block is required to find a number (a nonce) , such that when you concatenate the nonce, the previous hash, and the list of transactions that make up the block and then take the hash of this whole string, then that hash output should be a number that falls in a target space that is quite small in relation to the much larger output space of that hash function. Once in a while, one of them will find a random nonce that satisfies this property. That lucky node then gets to propose the next block. By this means, the system is completely decentralized. Nobody is deciding which node gets to propose the next block.

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

There are few methods when we *cannot* use for selection of random node :

1. *Public Keys* - There is no central authority to assign identities to participants and verify that theyre not creating new nodes at will. So, anyone can create a new identity whenever he wants. A malicious adversary can create copies of nodes and make them look like there are a lot of different participants, when in fact all those pseudo-participants are really controlled by the same adversary. The technical term for this is a *Sybil attack.*
2. *IP address* - The pseudonymity is inherently a goal of Bitcoin. So, if we use Public IP addresses, then one has to revel his real-life identity.

10 Block Rewards

If a miner mines a new block, they're given a reward in the form of the Block reward. This is the main incentive for Bitcoin miners as currently the Block reward is 12.5 BTC which is around $150,000, a significant amount of money.The block reward is halved every 210,000 blocks, which is approximately every 4 years.

11 Probability of an Attacker Winning Over Honest Nodes

p = probability an honest node finds the next block
q = probability the attacker finds the next block
z = number of blocks attacker has to cover in order to beat the longest chain

$$\lambda = z(\frac{q}{p})$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress attacker could have made by the probability attacker could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{(\lambda)^k e^{-\lambda}}{k!} = \begin{cases} (\frac{q}{p})^{z-k} & k \leq z \\ 1 & k > z \end{cases} \tag{1}$$

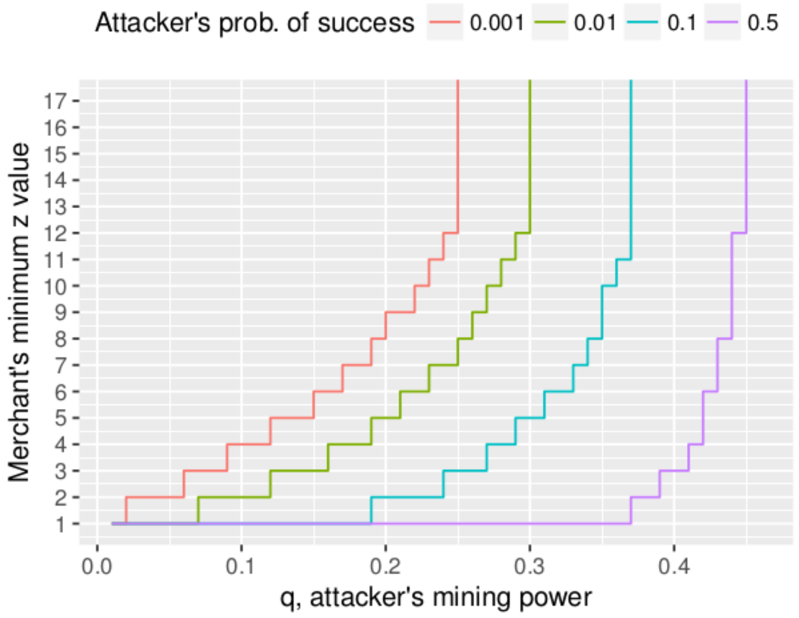

Figure 1: Given the attacker's mining power, and a desired probability of success (e.g, 0.001,0.01, 0.1, and 0.5), the plot shows the minimal value of z required of the merchant.

Figure 1 [1] shows the plot of equation 1 rather than values. We can use Figure 1 to calculate the minimum value of z, that merchant needs to have to prevent double spend attack from the Attacker whose probability of success and mining power are known.

At an average interval of 10 minutes, a new block is added to the system. A cautious merchant would not release the software to Alice even after the transaction was included in one block;he would continue to wait. If Bob sees that Alice successfully launches a double-spend attack, he realizes that the block containing Alices payment to him has been orphaned. He should abandon the transaction and not let Alice download the software. In general, the more confirmations a transaction gets, the higher the probability that it is going to end up on the long-term consensus chain as honest nodes always extend the longest valid branch that they find. The double-spend probability decreases exponentially with the number of confirmations.

[1]Image credits : Research paper on "An Explanation of Nakamoto's Analysis of Double-spend Attacks" [2]

12 Bitcoin Transaction

In Bitcoin transaction we don't maintain the what balance does each person has but instead each bitcoin has its own entity and is transferred from one address to another address. Transactions contain one-or-more inputs and one-or-more outputs.

- An input is a reference to an output from a previous transaction.
- An output specifies an amount and a address.

An input always references a previous transaction's output. This continual pointer of inputs to previous transactions outputs allows for an uninterrupted, verifiable stream of value (represented by the bitcoin currency) amongst addresses.

13 Bitcoin Scripting Language

The Bitcoin scripting language is very small and many similarities to a language called Forth, which is an old, simple, stack-based, programming language. The scripting language is stack-based i.e. every instruction is executed exactly once, in a linear manner. Theres only room for 256 instructions, because each one is represented by one byte. Of those 256, 15 are currently disabled, and 75 are reserved. The reserved instruction codes havent been assigned any specific meaning yet, but might be instructions that are added later in time. 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.

- A list of common Script instructions and their functionality:

  DUP : Duplicates the top item on the stack
  HASH160 : Hashes twice: first using SHA-256 and then RIPEMD-160
  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.

Bitcoin script to transfer 1 BTC from Amit to Mohit :

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

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

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

- *Inputs :* The transaction inputs form an array, and each input has the same form. An input specifies a previous transaction, so it contains a hash of that transaction, which acts as a hash pointer to it. The input also contains the index of the previous transactions outputs thats being claimed. And then theres a signature. Remember that we have to sign to show that we actually have the ability to claim those previous transaction outputs.
- *Outputs :* The outputs are again an array. Each output has just two fields. They each have a value, and the sum of all the output values has to be less than or equal to the sum of all the input values. If the sum of the output values is less than the sum of the input values, the difference is a transaction fee to the miner who publishes this transaction.
- To validate that a transaction redeems a previous transaction output correctly, we combine the new transactions input script and the earlier transactions output script. We simply concatenate them, and the resulting script must run successfully in order for the transaction to be valid. These two scripts are called *scriptPubKey* and *scriptSig* because in the simplest case, the output script just specifies a public key (or an address to which the public key hashes), and the input script specifies a signature with that public key.

References

[1] *Bitcoin and Cryptocurrency Technologies*, Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

[2] *An Explanation of Nakamoto's Analysis of Double-spend Attacks*, A. Pinar Ozisik and Brian Neil Levine

[3] https://en.wikipedia.org/wiki/Blind_signature