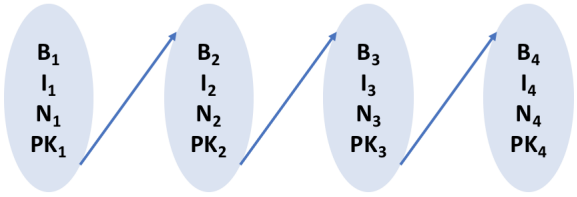
Introduction to Cryptography 03/29/2018

Lecture 18: Bitcoin and Blockchain (continued)

Instructor: Vipul Goyal *Scribe: Eipe Koshy*

1 Review

Q1. How does a miner check if a newly downloaded copy of the public ledger is legitimate?
Ans. The miner checks if the downloaded public ledger is legitimate by computing hashes and comparing them to existing hashes one-by-one for successive blocks on the public ledger.

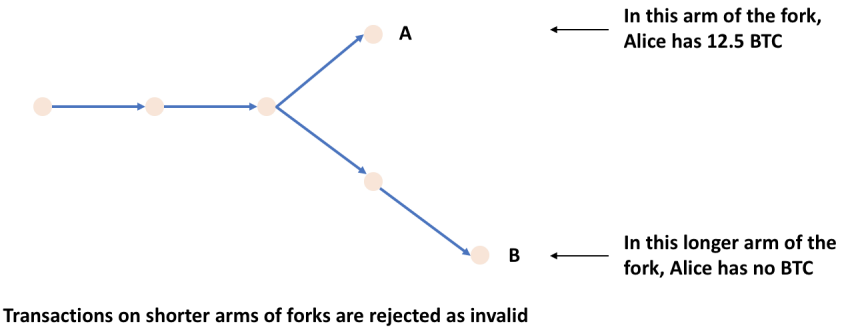


Hashes of successive blocks can be checked one by one for validation

Q2. What is a key problem with transactions using Bitcoin? Detail a potential workaround for this problem and show that it does not solve the problem completely.
Ans. Transactions using Bitcoin on the blockchain are not instantaneous, which can pose a problem for people receiving payments (such as merchants.) This problem is referred to as the "Double-spending" problem, since the payer can transfer the same payment to multiple payees at the same time and only one payee will receive the payment once confirmation comes through.

A potential workaround is to use signed contracts (for e.g. If Alice has to pay Bob 1 Bitcoin BTC, Alice, who has 1 BTC associated with her public key, will send Bob a signed contract that states "Alice will transfer 1 BTC to Bob", which Bob can redeem at any time to receive 1 BTC.) Using these signed contracts appears to make transactions instantaneous. However, this workaround also suffers from the double-spending problem, since the payer can send multiple payees signed contracts for the same amount, and only one of the payees will receive payments once they redeem the contract.

Q3. What happens to Bitcoins (BTC) mined and transactions made which are made on one arm of a fork, if the other arm of the fork becomes longer (and hence, valid)? **Ans.** Both BTC mined and transactions made on the shorter arm of the fork become invalid, since blocks with invalid blocks are rejected by the network as being invalid.



Q4. How can replay attacks be prevented on the blockchain?
Ans. Unique information specific to transactions, such as timestamps or nonces, can be used by miners to distinguish valid transactions from invalid ones.

2 Applications built on top of Bitcoin and the Blockchain

There are many applications that can be built on top of Bitcoin and the Blockchain. In this section, we will attempt to enumerate a few examples of such applications:

- **Bitcoin scripting language**

  The Bitcoin scripting language enables custom transactions based on certain criteria being met (which could occur in the future.) Examples of potential transactions enables by the scripting language include:

  1. "A will transfer 1 BTC to anyone who publishes the value of $f^{-}(y)$, where f^{-} is a hard-to-invert function."
  2. "A will transfer 1 BTC to B if B publishes a signed statement that 'B will ship a TV to A'"
  3. "A will transfer 1 BTC to a script being run at a particular address"
     Note, the script at this address will be run by a miner (which will be suitably rewarded for running the script.) Possible use-cases for such scripts include rent-collection, subscriptions, etc.

- **Smart Assets**

  Smart Assets are either a virtual representation of a physical asset, or represent virtual goods, such as an equity share. Examples of smart assets include:

1. *Gold-backed cryptocurrency coins*
   The basic idea is that each coin represents a value of gold (for e.g. 1 coin represents 1 gram of gold.) The value of gold is stored by a trusted third-party custodian and can be traded with other coin holders. Such coins can be used in smart contracts, and can reduce volatility in coin value (since the coins are backed by gold, the value of which is relatively more stable.)
2. *Virtual representations of real-world goods*
   The basic idea is that real-world goods, such as shipping insurance, land records, etc. are represented as abstract assets (using virtual currency tokens) on the blockchain. These abstract assets act as a public record of ownership (using a mechanism similar to that used to represent Bitcoin ownership) and can be transferred between owners securely and safely.

3 Information Verification on the Blockchain - Merkle trees

Verification on the blockchain is facilitated by the use of cryptographic primitives called Merkle trees. Merkle trees have been around since the 1980's and simplify the problem of key management for digital signatures. Given that exchanging a public key can be complex, a scheme is needed that reduces both the number of public keys needed to sign messages as well as their total size. Merkle trees are such a scheme, which uses a single public key to sign many messages.

A use-case that exemplifies the need for such a scheme is an email provider that signs email messages. Given that the provider has to sign and send millions of email messages everyday, it needs a signature scheme that generates millions of signatures, each of which are easy to verify. Merkle trees provide a solution to this use-case, as we shall see further below.

Our goal in developing the Merkle tree scheme is to build a cryptographic scheme that can generate a required number of signatures. Essentially, we are looking to develop a batch signature scheme, where each signature is individually verifiable.

A naive implementation of such as signature scheme is to sign a hash f concatenation of n messages, i.e.,

1. $Gen(1^n) \rightarrow (p_k, s_k)$

2. $\sigma = Sign_{s_k}(H(m_1|m_2|m_3|m_4|....|m_n))$

3. $Verify_{p_k}(m_1, \sigma)$ requires that we know the other $n-1$ messages, so that we can hash and verify.

While such a scheme works in practice, the length of the signature generated is of the order of the sum of the length of the messages, which makes management difficult. Our goal is to do better.

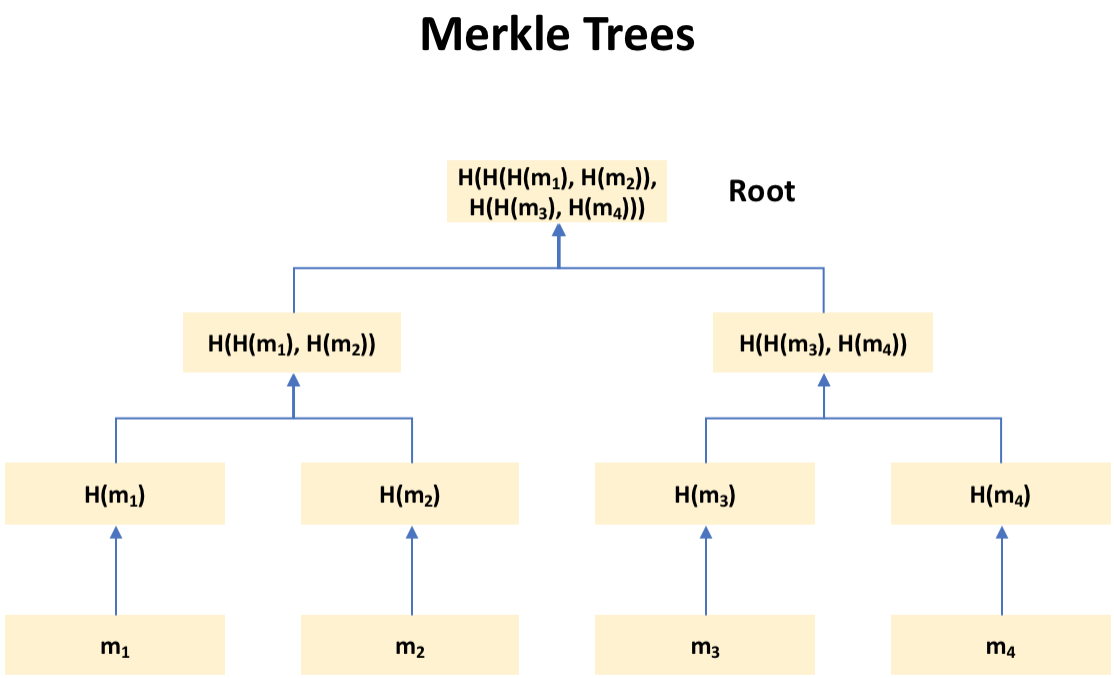


Image source: https://en.wikipedia.org/wiki/Merkle_tree

The key idea behind Merkle trees is to arrange messages as the leaf nodes of a tree, then repeatedly hash pairs of nodes till there is only one hash left at the root of the tree, which is then signed.

Assume m_2 is the message whose signature needs to be verified. For verification, we need the signature, as well as the hashes of the siblings at every level. Referring to the diagram above, we need the signature σ, as well as the hashes of the siblings at every level, i.e. $H(m_1)$, $H(H(m_3), H(m_4))$. Using m_2 and the hashes of the siblings, we can generate the hash at the root, $H(H(H(m_1), H(m_2)), H(H(m_3), H(m_4)))$ which can then be verified against the signature σ using the public key.

Correctness

Part 1: By signing the root, all messages are effectively signed.
Part 2: Size of the signature is related to $log(n)$, where n is the number of messages.

Security

An adversary should not be able to produce message m', such that message m'=m_i $\forall i$ messages covered by signature σ. Security relies on the Collision Resistance of the hash functions used. Suppose the adversary generates a message m', say, in place of message m_2 as shown in the figure. Given this new message, and the original siblings at every level, the adversary must be able to generate the same hash at the root. Doing so will require the hashes generated with the new message, m' at multiple levels must be the same as that using the old message, m_2, which is ruled out by the collision resistance property of the hash functions used. In our example, for the same hash to be generated at the root, the following hashes to all be equal, which is difficult given the collision resistance property of the hash functions:

1. $H(m') \neq H(m_2)$
2. $H(H(m'), H(m_2)) \neq H(H(m_1), H(m_2))$
3. $H(H(H(m'), H(m_2)), H(H(m_3), H(m_4))) \neq H(H(H(m_1), H(m_2)), H(H(m_3), H(m_4)))$

Properties

1. Able to sign messages as a block.
2. Enables light weight verification on low-memory devices such as mobile devices.

Usage in Bitcoin

Merkle trees are used for lightweight verification of transactions in Bitcoin. To verify all the transactions in a block, one doesn't need to download all the transactions in the block. One can instead just download the hashes going down a particular branch of the Merkle tree and compute the hashes all the way to the root.

4 Limitations of Bitcoin

Bitcoins have several limitations including:

1. Transactions are not instantaneous.
2. Wastage of computational resources.
   Bitcoin uses Proof of Work as the mechanism for validation, which consumes a lot of resources. As a solution, we can use alternate energy-efficient validation mechanisms, such as:
   - *Proof of Stake*: Validation is performed by those who already own units of the cryptocurrency.
   - *Proof of Storage*: Validation is performed by those who have storage that can be used for the public ledger of the cryptocurrency.
3. Scalability
   Based on the current block sizes and mining rates, Bitcoin can process 7 transactions per second. In comparison, Visa can process > 5000 transactions per second.

5 Types of forks in Bitcoin

There are two types of possible of forks in Bitcoin:

1. **Soft forks**
   Soft forks are a backward compatible means of upgrading software on the nodes of the blockchain, where the original chain (which contains non-upgraded software and contains original blocks) also accepts non-updated blocks, while the upgraded nodes on the forked chain only accepts updated blocks.

   Examples of soft forks[1] include:

   - *BIP 66*: A soft fork on Bitcoins signature validation
   - *P2SH*: A soft fork that enabled multi-signature addresses in Bitcoins network

2. **Hard forks**
   Hard forks are non-backward compatible means of upgrading software on nodes of the blockchain, where the original and forked chain run different software and rules, which are not compatible with each other. Blocks mined on any one chain are not compatible with the other chain.

   Examples of hard forks[1] include:

   - *Ethereum's Byzantium*: Represents a planned multi-phase upgrade of Ethereum's blockchain base to deliver features such as better scalability and integration of private transactions
   - *Monero hard fork*: Represents an upgrade to Monero network to implement a feature called Ring Confidential Transactions (RCT) to improve privacy and security

[1] `https://masterthecrypto.com/guide-to-forks-hard-fork-soft-fork/`