# COSC-373 Final Report: Bitcoin

Lee Jiaen, Hyery Yoo, and Alexander Lee

May 17, 2022

## 1 Introduction

Before Bitcoin, the only commerce system on the internet relied heavily on trusted third parties to process electronic payments. Suppose Alice, a buyer, wants to buy an item from Bob's e-commerce store. Alice would need to make an online payment that relies on a financial institution or some trusted third party to process the transaction. However, Alice may not trust such third parties. Alice wishes for a way where her online payment can be sent directly to Bob without going through a third party. While the previous system worked for most transactions, it had a few significant inefficiency problems. Trusted third-party financial intuitions cannot avoid mediating disputes, making it impossible to have completely non-reversible transactions. Therefore, the need for trust increases, and "a certain percentage of fraud is accepted as unavoidable" (Nakamoto 2008). One could avoid these issues with the use of physical currency. However, like Alice, many people would prefer a decentralized version of electronic cash that allows any two willing parties to transact directly without a trusted third party. This is the goal and motivation of Bitcoin. The following paper describes how Bitcoin, an electronic payment system, is based on cryptographic proof instead of trust, allowing for direct transactions without trusted third parties or financial institutions.

## 2 Preliminaries

To understand Bitcoin, we first need to develop an understanding of the fundamental building-blocks behind it: digital signatures and cryptographic hash functions.

### 2.1 Digital Signatures

Suppose for example that Alice wants to send a message to Bob over a network. How can Bob ensure that the message he received was from Alice and not from a malicious actor like Eve? What Alice can do is append a *digital signature* to her message and send the message and signature to Bob, which allows Bob to verify that the message he received was indeed from Alice.

The main idea behind digital signatures is as follows. Both Alice and Bob each generate a *private key* and *public key* pair for themselves, where each key is a array of bits and private keys are kept secret (i.e., only Alice knows her private key and only Bob knows his private key). Producing a signature involves a function sign(msg, priKey), which takes Alice's message msg and private key priKey as inputs and outputs Alice's signature sig. Similarly, verifying the authenticity of the received message involves a function verify(msg, sig, pubKey), which takes the received message msg, the signature sig, and Alice's public key pubKey as inputs and outputs *true* if the signature was produced using Alice's message and private key, and *false* otherwise. We will not discuss how the sign() and verify() functions work in depth since such details are not the focus of this work.

Digital signatures have two required properties. First, the authenticity of Alice's signature generated from her message and private key can be verified easily using her corresponding public key. Secondly, it should be computationally infeasible for someone like Eve to generate a valid signature for Alice without knowing Alice's private key. That is, Eve has no better strategy than guessing and checking if random signatures are valid using Alice's message and public key. Assuming a 256 bit signature, the aforementioned brute force strategy would require Eve to check $2^{256}$ signatures in the worst case.

### 2.2 Cryptographic Hash Functions

A *cryptographic hash function* is a hash function that takes as input data of an arbitrary size, known as the *message*, and outputs a bit array of a fixed size, known as the *hash*. In the case of the SHA256 cryptographic hash function, the hash has a length of 256 bits.

Cryptographic hash functions ideally should have the following properties. First, computing the hash of a given message should be quick. Second, generating the message from only the hash should be computationally infeasible. Third, finding two different messages that map to the same hash under the function should also be infeasible. Fourth, slightly changing the message should change the hash so much such that the old and new hashes seem uncorrelated. Similar to digital signatures, the only way to find a message that produces a given hash is to via a brute force approach of guessing and checking possible messages. Again, assuming a 256 bit hash, this strategy would require $2^{256}$ hashes to check in the worst case.

## 3 Bitcoin

The *Bitcoin network* is a randomly connected overlay network of a few thousand nodes, where nodes are controlled by various owners and perform the same operation. In other words, the network is a homogeneous network without central control. Each user of Bitcoin generates a private/public key pair. A user is identified by their *address*, which is derived from their public key and

used to receive funds in Bitcoin. All nodes in the Bitcoin network work together to track each address' balance in bitcoins.

# 4 Transactions

Suppose we have a *transaction*, where Alice is the sender and Bob is the recipient. What is a transaction, and how does it work?

A *transaction* is a data structure that describes the transfer of bitcoins from spenders to recipients. The transaction consists of several inputs and outputs. More specifically, the inputs are a tuple consisting of a reference to a previously created output and a signature to the spending condition, proving that the transaction creator has the permission to spend the referenced output. Meanwhile, the outputs are a tuple composed of an amount in bitcoins and a spending condition requiring a valid signature associated with the private key of an address. The outputs of a transaction may assign less than the sum of inputs, in which case the difference is called the transaction fee. The transaction fee is used to incentivize other participants in the system.

An output has two states, unspent and spent, and can only be spent at most once. When we sum up the bitcoin amounts of unspent outputs associated with an address, we get the address balance, and the set of these unspent transaction outputs is the shared state of Bitcoin.

Transactions can be in one of two states: unconfirmed or confirmed. Unconfirmed transactions are incoming transactions from the broadcast that are added to a pool of transactions called the memory pool. All transactions are broadcasted to the entire Bitcoin network, with each node deciding whether or not to accept the transaction. As a result, every node in the Bitcoin network holds a complete replica of the shared state. The local replicas of this shared state may temporarily diverge, but consistency is eventually reestablished.

Thus, for the transaction between Alice to Bob, the inputs are Alice's previous unspent outputs and Alice's digital signature, which was created using her private key. Meanwhile, the outputs of this transaction are the amount in Bitcoins for Bob and the spending condition requiring a valid signature associated with the private key of Bob's address. The transaction is then broadcasted in the Bitcoin network and processed by every node that receives it.

# 5 Doublespending Problem

Suppose we now have another transaction from Alice to Charlie that attempts to spend the same output as the transaction from Alice to Bob. *Doublespending* is a situation in which multiple transactions attempt to spend the same output. However, only one transaction can be valid because outputs can only be spent at most once. Doublespending occurs because the order in which transactions are seen may not be the same for all nodes, and the validity of transactions depends on the order in which the transactions arrive. If a node sees two con-

flicting transactions, it considers only the first transaction it sees valid and the second invalid. The second seen transaction is invalid since it tries to spend an already present output. As a result, different nodes in the network accept different transactions, making their shared states inconsistent. If the problem of doublespending is not resolved, the shared states of the Bitcoin network diverge. The problem is that there is nothing preventing nodes from locally accepting different transactions that spend the same output. Therefore, a conflict resolution mechanism is needed to decide which of the conflicting transactions (only one transaction) should be accepted by every node to achieve eventual consistency in the network.

## 6 Proof-of-Work

## 7 Network

## 8 Conclusion

## 9 References

https://bitcoin.org/bitcoin.pdf