# A review of the Blockchain literature

George Pîrlea
george.pirlea.15@ucl.ac.uk

November 14, 2016

# Contents

# 1 Bitcoin

## 1.1 History

On October 31st, 2008, Satoshi Nakamoto announces [1] that he has published a design paper for Bitcoin [2], "a new electronic cash system that's fully peer-to-peer, with no trusted third party".

Block 0 of the Bitcoin blockchain, the Genesis Block, is established at 18:15:05 GMT on January 3rd, 2009 [3]. The block contains the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" [4].

Version 0.1 of the Bitcoin software was released on January 9th, 2009 [5].

## 1.2 Rationale

The current system of electronic payments relies on trusted financial institutions to process transactions. This *trust-based* model has inherent weaknesses:

- Non-reversible transactions are not possible; disputes need to be mediated
- Small payments are infeasible; cost of mediation increases transaction costs
- Transactions are not anonymous; third party must collect personal information

Bitcoin, on the other hand, is an electronic payment system based on *cryptographic proof of work* instead of trust, allowing two parties to transact directly with each other without the need for a trusted third party.

## 1.3 Technical analysis

### Foreword

The Bitcoin paper gives a very high level overview of the system. It does not describe how Bitcoin is actually implemented. If you only read the paper, you will have an inexact (and in some cases incorrect) view of how Bitcoin works. For example, Bitcoin uses a stack-based scripting language for transactions [6], but the paper does not mention it at all.

To get an accurate understanding of Bitcoin internals, check the source code [7], the developer guide [8] and *Mastering Bitcoin* by Andreas Antonopoulos [9].

### 1.3.1 Block chain

Bitcoin keeps a public ledger of all transactions, called the *block chain*. This is, in essence, a peer-to-peer distributed timestamp server. Its role is to prevent double spending and modification of previous transaction records.

Each full node in the Bitcoin network keeps a complete copy of the block chain, containing all blocks validated by that particular node. Consistency across the network is guaranteed by following a series of *consensus rules*. When several nodes in the Bitcoin network independently arrive at identical block chains, they are considered to be in *consensus*.

#### Overview

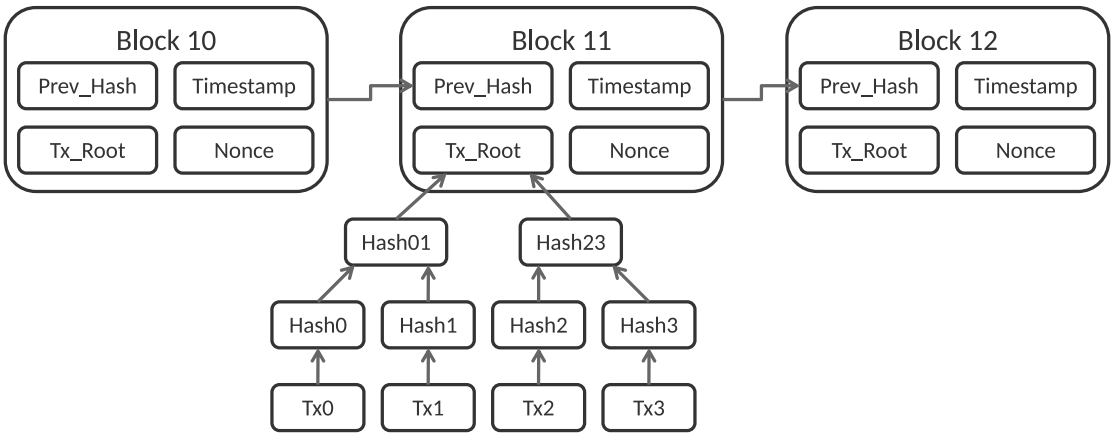


Figure 1.1: Simplified depiction of the Bitcoin block chain [10]

A block chain, as the name suggests, is a digital chain of *blocks*. Each block contains a timestamp, a *nonce* (a number whose purpose will be described later) and a Merkle Tree [11] that stores transactions.

The blocks are cryptographically chained together using hashes. Each block contains the hash of the previous block, which in turn contains the hash of the previous, and so on back to the first block. Modifying any part of a past block would invalidate all the subsequent hashes. This is critical for preserving the integrity of the ledger.

Computing a hash, however, is not very difficult. A usual CPU can compute roughly a million SHA-256 hashes per second. GPUs can compute hundreds of millions of hashes per second [12]. ASICs (application-specific integrated circuits) are even faster. Under normal circumstances, it would be possible to modify previous blocks, as you could simply recompute all subsequent hashes.

### 1.3.2 Proof of work

The block chain is maintained collaboratively by peers on the Bitcoin network. To prevent malicious peers from modifying past blocks, the network requires a significant

amount of work to be invested in the generation of each block. Chaining blocks together makes it impossible to modify transactions included in any block without modifying all following blocks.

When there are multiple chains competing for acceptance in the network (for example, when someone tries to rewrite a past block), consensus rules dictate that nodes will pick as valid the chain that is longest, i.e. the chain that has most proof of work behind it [13].

To successfully rewrite a past block, a malicious actor needs to re-generate all following blocks and overtake the honest nodes in the network. This is computationally infeasible, as long as honest peers control a large share of the total hashing power.

#### Implementation

The network only accepts a block as valid if the hash of its header is less than a certain *hash target*, which is computed based on the network *difficulty*. The network sets the difficulty and updates it every 2016 blocks [14, 15] so a block is generated, on average, every 10 minutes.

Remember the nonce in the block header that we did not explain? This is where it comes in. The nonce is a 32 bit integer that you can increment to change the block's hash. When mining, you try different nonces until the block header hashes to something less than the hash target.

In practice, since the network difficulty is high, a 32 bit nonce is not enough to beat the hash target, so miners also change `extraNonce`, a field included in the coinbase transaction of the block [16, 17, 18]. As the coinbase transaction is included in the Merkle Tree, any change propagates upwards into the Merkle root in the header.

Bitcoin uses the Hashcash proof-of-work function [19, 20].

### 1.3.3 Transactions

A transaction has at least one input and one output. For normal transactions, each input has to be the output of a previous transaction. When a transaction is made, each output waits as an Unspent Transaction Output (UTXO) until a later input spends it.

It is important to note that coins are **not fungible**, i.e. they can be differentiated from one another, based on their chain of ownership. When your Bitcoin wallet tells you that you have a balance of 1,000 satoshis, it really means you have a total of 1,000 satoshis waiting in one or more UTXOs.

#### Coinbase transactions

New coins are created in special transactions called *coinbase transactions* [21]. These have a special input field called *coinbase* [22], which isn't the output of a previous transaction — it creates coins.

The Bitcoin network allows miners to include a coinbase transaction at the beginning of every block they mine. Miners claim the *block reward* by creating new coins for themselves.

### Transaction fees

If the total value of a transaction's outputs is greater than the total value of its inputs, the difference can be collected by miners as a *transaction fee*, on top of the block reward they get for mining the block. If a transaction doesn't have a fee, miners have no incentive to add it to their blocks.

### Transaction types

Bitcoin uses a scripting system to process and define transactions. This language, called Script, is stack-based and is purposefully not Turing-complete [6].

A script is a list of instructions recorded with each transaction that describes how a person who wants to spend the transaction's outputs can gain access to them in the future. There are several main types of transactions:

- **P2PKH** – Pay to Public Key Hash

  This is the most common type of transaction. To gain access to its outputs, the recipient needs to prove that he can sign a message with the private key (S) which corresponds to public key (P) that hashes to H.

  To create a transaction, the payer needs to know the recipient's address.

  P2PKH addresses are the Base58Check encoding [23] of version number `0`, a hash of the recipient's public key (H), and a checksum. Due to the way they are constructed, all P2PKH addresses start with the character "`1`" [24].

  Example: `1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2`

- **P2PK** – Pay to Public Key

  With this transaction type, the public key itself is stored in the locking script, rather than a public-key-hash.

  P2PKH is an improvement on this, as it has shorter addresses and provides some security benefits (assuming addresses are not re-used). [9]

- **OP_RETURN** – Data Output

  Allows developers to add 40 bytes of non-payment data to a transaction output, which is stored on the block chain. [9]

- **P2SH** – Pay to Script Hash

  To gain access to outputs sent via P2SH, the recipient must provide a script which matches the given hash and data which makes that script evaluate to `True`.

  This allows the recipient to implement various unusual security schemes. For example, one can thus send bitcoins to *multi-signature addresses*, which require the signatures of multiple people to spend the outputs [25].

P2SH addresses are the Base58Check encoding [23] of version number `5`, a hash of the recipient's public key (H), and a checksum. Due to the way they are constructed, all P2SH addresses start with the character "`3`" [24].

Example: `3EktnHQD7RiAE6uzMj2ZifT9YgRrkSgzQX`

There is space for expansion, so more transaction types may be added in the future.

### 1.3.4 Contracts

Distributed contracts are a way of using Bitcoin transactions to enforce financial agreements. They can be used to formalise and guarantee agreements in a way that does not rely on the traditional court system. Example contracts include Escrow, Micropayment channels and CoinJoin [8].

Several different types of contracts can be implemented in Script, but the language is not Turing-complete, so not all possible contracts can be implemented. For a block chain system specifically designed for smart contracts, check out Ethereum and its Turing-complete contract language, Solidity.

## 1.4 Proposed improvements

Various proposals to improve certain aspects of Bitcoin have been published. Some of them have been collected as BIPs (Bitcoin Improvement Proposals) [26]. Others proposals are stand-alone papers or forum posts. Some propose improvements of the existing Bitcoin system, while others create entirely new crypto-currencies.

| **Weakness** | **Proposed solutions** |
|---|---|
| Scalability | Increase block size [27]<br>Ultimate blockchain compression [28]<br>Mini-Blockchain Scheme [29]<br>Lighting Network [30]<br>Sharding [31] |
| Lack of anonymity | Ring signatures [32]<br>zkSNARKs [33] |
| Miner centralisation | Uncle blocks [34]<br>Egalitarian proof of work [32] |
| Electricity waste | Proof of stake [35, 36]<br>Proof of elapsed time [37]<br>Proof of capacity [38]<br>Proof of burn [39] |

# 2 Ethereum

## 2.1 Previous work

The Bitcoin protocol does facilitate a weak version of *smart contracts*, however the scripting language implemented in Bitcoin has several important limitations [40]:

- **Lack of Turing-completeness** – the scripting language doesn't have loops
- **Value blindness** – cannot control what amount is withdrawn; either the entire UTXO is withdrawn, or none of it is
- **Lack of state** – UTXOs scripts can only be used for one-off contracts; "stateful" multi-stage contracts such as decentralised organisations are impossible to implement
- **Blockchain blindness** – cannot access blockchain data such as block headers or transaction history

## 2.2 Rationale

Ethereum aims to build "the ultimate abstract foundational layer": a blockchain with a built-in Turing-complete programming language, value awareness, blockchain awareness and state [40].

Smart contracts, decentralised applications and even entirely new protocols (currencies, reputation systems etc.) can be developed on top of the Ethereum block chain.

This new wave of blockchain-based technology is called "Blockchain 2.0".

## 2.3 Technical analysis

For a high level overview of Ethereum, the original white paper [40] by Vitalik Buterin is a good resource.

The Ethereum Yellow Paper [41] by Gavin Wood provides a formal specification of the system. Indeed, this was the first formal specification of *any* blockchain protocol.

### 2.3.1 Accounts

The state in Ethereum consists of objects called *accounts*. Each account has a 20-byte address. Ethereum has two types of accounts: *externally owned accounts*, controlled by private keys, and *contract accounts*, controlled by their contract's code [40].

An Ethereum account contains 4 fields [41]:

- `nonce` – for externally owned accounts, the number of transactions sent; for contract accounts, the number of contract-creations made; used to make sure each transaction can only be processed once
- `balance` – number of Wei in this account
- `codeHash` (immutable) – for contract accounts, stores the hash of the EVM code of this account; the actual code is stored in the global state database
- `storageRoot` – a hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account

### 2.3.2 Transactions

A *transaction* is a cryptographically-signed instruction created by an external actor. There are two types of transactions: message-call transactions and contract-creation transactions.

A transaction contains a number of standard fields [41]:

- `nonce` – (sender account's nonce at the time the transaction was sent) + 1
- `gasPrice` – the number of Wei to be paid per unit of *gas* for all computation costs incurred as a result of the execution of this transaction
- `gasLimit` – maximum amount of gas that should be used in executing this transaction; this is paid up-front, before any computation is done and may not be increased later
- `to` – 20-byte address of the message-call recipient or, in the case of a contract-creation transaction, empty
- `value` – number of Wei to be transferred to the message-call recipient or to the newly-created account
- `v`, `r`, `s` – values corresponding to the signature of the transaction and used to determine the sender of the transaction

Contract transactions additionally contain:

- `init` – EVM-code fragment for the account initialisation procedure

  `init` is EVM-code that returns the `body`, a second EVM-code fragment that executes each time the contract account receives a message. `init` is executed only once, at account creation, and is discarded immediately afterwards.

Message transaction contain:

- `data` – a byte array specifying the input data of the message call

### 2.3.3 Messages

*Messages* are used to pass `data` and `value` (Ether) between two accounts. A message can be created either by external entities through a transaction, or by contracts through their EVM-code.

# Bibliography


[1] Satoshi Nakamoto. *Bitcoin P2P e-cash paper*. Oct. 2008. URL: http://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html.

[2] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[3] *Block #0*. URL: https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f.

[4] *Genesis block*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/Genesis_block.

[5] Satoshi Nakamoto. *Bitcoin v0.1 released*. Jan. 2009. URL: http://www.metzdowd.com/pipermail/cryptography/2009-January/014994.html.

[6] *Script*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/Script.

[7] *Bitcoin Core*. Source code. URL: https://github.com/bitcoin/bitcoin.

[8] *Bitcoin Developer Guide*. URL: https://bitcoin.org/en/developer-guide.

[9] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, Dec. 2014. URL: http://chimera.labs.oreilly.com/books/1234000001802/index.html.

[10] Matthäus Wander. Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.svg.

[11] *Merkle Tree*. Wikipedia. URL: https://en.wikipedia.org/wiki/Merkle_tree.

[12] *GPU Mining Hardware Comparison Table*. Coin Police. URL: http://coinpolice.com/gpu/.

[13] *What does the term "Longest chain" mean?* Bitcoin StackExchange. URL: BitcoinSEConsensus.

[14] *Difficulty*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/Difficulty.

[15] *Bitcoin Difficulty and Hashrate Chart*. Bitcoin Wisdom. URL: https://bitcoinwisdom.com/bitcoin/difficulty.

[16] Andreas M. Antonopoulos. *Mastering Bitcoin*. Extra Nonce. O'Reilly Media, Dec. 2014. URL: http://chimera.labs.oreilly.com/books/1234000001802/ch08.html#extra_nonce.

[17] *What is the extraNonce?* Bitcoin StackExchange. URL: https://bitcoin.stackexchange.com/questions/5048/what-is-the-extranonce.

[18] *Determining a Block's Extranonce Value*. Bitcoin StackExchange. URL: https://bitcoin.stackexchange.com/questions/36455/determining-a-blocks-extranonce-value.

[19] *What is Hashcash?* Bitcoin Mining. URL: https://www.bitcoinmining.com/what-is-hashcash/.

[20] Adam Back. *Hashcash-a denial of service counter-measure*. 2002. URL: http://www.hashcash.org/papers/hashcash.pdf.

[21] *Coinbase / Generation Transaction*. Bitcoin Glossary. URL: https://bitcoin.org/en/glossary/coinbase-transaction.

[22] *Coinbase, Coinbase Field*. Bitcoin Glossary. URL: https://bitcoin.org/en/glossary/coinbase.

[23] *Base58Check encoding*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/Base58Check_encoding.

[24] *List of address prefixes*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/List_of_address_prefixes.

[25] *Address*. Bitcoin Wiki. URL: https://en.bitcoin.it/wiki/Address.

[26] *Bitcoin Improvement Proposals*. URL: https://github.com/bitcoin/bips.

[27] Grace Caffyn. *What is the Bitcoin Block Size Debate and Why Does it Matter?* CoinDesk. URL: http://www.coindesk.com/what-is-the-bitcoin-block-size-debate-and-why-does-it-matter/.

[28] *Ultimate blockchain compression with trust-free lite nodes*. Bitcoin Talk. URL: https://bitcointalk.org/index.php?topic=88208.0.

[29] J.D. Bruce. "The Mini-Blockchain Scheme". In: (July 2014). URL: http://cryptonite.info/files/mbc-scheme-rev2.pdf.

[30] Joseph Poon and Thaddeus Dryja. "The Bitcoin Lighting Network: Scalable Off-Chain Instant Payments". In: (Jan. 2016). Version 0.5.9.2. URL: https://lightning.network/lightning-network-paper.pdf.

[31] Vlad Zamfir. *Sharding the Blockchain*. URL: http://diyhpl.us/wiki/transcripts/scalingbitcoin/sharding-the-blockchain/.

[32] Nicolas van Saberhagen. "CryptoNote v2.0". In: (Oct. 2013). URL: https://cryptonote.org/whitepaper.pdf.

[33] Eli Ben-Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy* (2014). URL: http://zerocash-project.org/paper.

[34] *Ethereum Block Protocol 2.0*. URL: https://github.com/ethereum/wiki/wiki/Block-Protocol-2.0.

[35] *Proof of stake instead of proof of work*. Bitcoin Talk. URL: https://bitcointalk.org/index.php?topic=27787.0..

[36] Sunny King and Scott Nadal. "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake". In: (Aug. 2012). URL: https://www.peercoin.net/assets/paper/peercoin-paper.pdf.

[37] *Proof of Elapsed Time (PoET)*. URL: https://intelledger.github.io/introduction.html#proof-of-elapsed-time-poet.

[38] *BURST's Proof of Capacity mining*. Bitcoin Talk. URL: https://bitcointalk.org/index.php?topic=731923.0.

[39] P4Titan. "Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn". In: (May 2014). URL: http://www.slimcoin.club/whitepaper.pdf.

[40] Vitalik Buterin. "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform". In: (2013). URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[41] Gavin Wood. "Ethereum: a secure decentralised generalised transaction ledger". In: (2014). URL: http://gavwood.com/Paper.pdf.