Blockchain Quick Reference

A guide to exploring decentralized blockchain application development

By Brenn Hill, Samanyu Chopra and Paul Valencourt

Packt

www.packt.com

Blockchain Quick Reference

A guide to exploring decentralized blockchain application development

Brenn Hill
Samanyu Chopra
Paul Valencourt

BIRMINGHAM - MUMBAI

Blockchain Quick Reference

Commissioning Editor: Sunith Shetty
Acquisition Editor: Tushar Gupta
Content Development Editor: Rhea Henriques
Technical Editor: Dharmendra Yadav
Copy Editor: Safis Editing
Project Coordinator: Manthan Patel
Proofreader: Safis Editing
Indexer: Rekha Nair
Graphics: Jisha Chirayil
Production Coordinator: Nilesh Mohite

First published: August 2018

Production reference: 1070818

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78899-578-8

www.packtpub.com

Foreword

Considering the frenzy around Blockchain and it's first killer app, Bitcoin, it is notable how few people actually understand not only what it is, but why it is. At its core, it is a ledger—not necessarily sexy, so why all the fuss?

We can go back to the invention of double-entry bookkeeping during the Renaissance to see the impact of a trustworthy ledger. Likewise, we can look to the more recent financial crisis of 2008 to study the impact of non-trustworthy ledgers. A secure, distributed, verifiable, and programmable account of transactions between parties has the flexibility to impact on a wide variety of businesses. Banking and finance are the most obvious businesses to benefit from this. Indeed this is what gave rise to Bitcoin—utilizing Blockchain to avoid the traditional model of a centralized bank to manage transactions between entities and reduce costs by removing the middleman.

Really, any supply chain is impacted, whether it be money, livestock, precious gems, or classic cars. Both physical and virtual assets can take advantage of Blockchain as a way to ensure that you're getting what you are told you're getting. This certainly justifies the fuss.

This brings us to *Blockchain Quick Reference*. The name belies the thoroughness of the book while emphasizing it's concise, clear-eyed approach. Brenn, Samanyu, and Paul have laid out a methodical manual that avoids hyperbole. Starting with *Blockchain 101* and progressing in-depth into topics at the core of Blockchain such as consensus, smart contracts, and the fundamental cryptographic concepts that form the foundation for any secure system.

The first half of the book gives a reader a solid foundation for understanding the what and why of this process. I say process rather than technology because it is not a single item but a collection of technologies combined with a systematic and consistent methodology. They then dive into real-world examples that give context to the hysteria. They, of course, cover Bitcoin and some of its competitors such as Ethereum along with an explanation of the often-heard but rarely understood **Initial Coin Offerings**. The authors are not wide-eyed academics but seasoned technologists who address the practical need for understanding and realistic dialog. The 1990's saw unrealistic, unjustified growth for companies founded upon the nascent World Wide Web. The aftermath of the bubble burst left in place the foundation of the internet and a move towards broadband that drives much of the economy today. Similarly, the Bitcoin craze has exposed the world to the benefits and mysteries of Blockchain.

Blockchain Quick Reference explores these benefits and makes it less mysterious and it certainly warrants a bit of fuss.

By Rino Lupetin,
Managing Partner, P&I
6D Global Technologies, Inc.

Contributors

About the authors

Brenn Hill is a senior software engineer who has worked with such clients as NASCAR, PGA Tour, Time Warner Cable, and many others. He has experience leading international teams on cannot fail engineering projects. He strives to work with business to ensure that tech projects achieve good ROI and solve key business problems. He has a master's degree in Information Science from UNC-CH and currently travels the world as a digital nomad.

Samanyu Chopra is a developer, entrepreneur, and Blockchain supporter with wide experience of conceptualizing, developing, and producing computer and mobile software's. He has been programming since the age of 11. He is proficient in programming languages such as JavaScript, Scala, C#, C++, Swift, and so on. He has a wide range of experience in developing for computers and mobiles. He has been a supporter of Bitcoin and blockchain since its early days and has been part of wide-ranging decentralized projects since a long time. You can write a tweet to him at `@samdonly1`.

Paul Valencourt is CFO of BlockSimple Solutions. He currently helps people launch STOs and invest in cryptocurrency mining. Harass him on LinkedIn.

I would like to thank Jacob Gadikian for introducing me to the madness of the STO space and Tim Sandau for showing me the comparative calm of the cryptocurrency mining world. I would also like to thank my fiancée Maria for supporting me in my interminable visits to coffee shops in order to complete my drafts.

About the reviewers

Sanket Thodge is an entrepreneur and speaker by profession based out of Pune, India. Sanket is an author of *Cloud Analytics with Google Cloud Platform*. Sanket is founder of Pi R Square Digital Solutions and has expertise in big data. Sanket has explored Cloud, IoT, machine learning and blockchain. Sanket has also applied for a patent in IoT and has worked with numerous start-ups and MNCs in providing consultancy, architecture building, development, and corporate training across the globe.

Rajib Bhattacharya currently works as a Pre Sales Consultant with a premier IT solution provider in the Middle East. He has 13 years of experience in IT Consulting and architecture solutions. He is a seasoned architect focusing on analytics, data warehousing, big data, and cognitive computing. He holds multiple patents and is an author of 2 books in the field of analytics. He is passionate in helping clients to combine their enterprise data, which enables them to make valuable business decisions using analytics solutions. He holds a Master of Computer Applications degree from West Bengal University of Technology and is certified in technologies. He has developed and delivered various training programs on analytics and cognitive computing.

Table of Contents

Preface

This is *Blockchain Quick Reference*, a book designed to introduce newcomers to the world of blockchain in a holistic fashion. It takes you through the electrifying world of blockchain technology, and it is aimed at those who want to polish their existing knowledge of the various pillars of the blockchain ecosystem.

This book is your go-to guide that teaches you how to apply principles and ideas that making your life and business better. You will cover the architecture, **Initial Coin Offerings** (**ICOs**), tokens, smart contracts, and terminology, before studying how they work. All you need is a curious mind to get started with blockchain technology. Once you have grasped the basics, you will explore components of Ethereum, such as ether token, transactions, and smart contracts, in order to build simple Dapps. You will then move on to learning why Solidity is used specifically for Ethereum-based projects, followed by exploring different types of blockchain with easy-to-follow examples. All this will help you tackle challenges and problems. In addition to this, you will learn how blockchain has been revolutionizing IoT and machine learning, and how it can affect business processes.
By the end of this book, you will not only have solved current and future problems relating to blockchain technology but also be able to build efficient decentralized applications.

Who this book is for

This is a book for those who are interested in blockchain but are overwhelmed by the sudden explosion of options in the space. No longer limited to just Bitcoin, blockchain technology has spread into many sectors and a significant number of different technologies. This book is primarily aimed at business people who want an introduction to the technology, how it works, the major types of blockchains out there, and some help getting started by using it in your organization or planning an ICO.

This book is aimed primarily at business users and developers who are considering a blockchain based project. This book will help orient you to the current world of blockchain introduces the major software projects and packages and covers some of the legal background currently affecting the field.

What this book covers

`Chapter 1`, *Blockchain 101*, explains what blockchain technologies are and how they work. We also introduce the concept of the distributed ledger.

`Chapter 2`, *Components and Structure of Blockchain*, takes a closer look at the technical underpinnings of a blockchain and peeks under the hood to understand what a block is and how the chain is created.

`Chapter 3`, *Decentralization Versus Distributed Systems*, covers different types of decentralized and distributed systems and cover the often-overlooked differences between them.

`Chapter 4`, *Cryptography and Mechanics Behind Blockchain*, discusses over the fundamentals of cryptographic systems which are critical to the proper functioning of all blockchains.

`Chapter 5`, *Bitcoin*, examine Bitcoin, the first blockchain, and it's specific mechanics in depth.

`Chapter 6`, *Altcoins*, covers the major non-bitcoin cryptocurrency projects that have gained popularity over the last few years.

`Chapter 7`, *Achieving Consensus*, looks into the different ways blockchains help achieve consensus. This is one of the most important aspects of blockchain behavior.

`Chapter 8`, *Advanced Blockchain Concepts*, covers the interactions between blockchain technology, privacy, and anonymity along with some of the legal side effects of the blockchain technology.

`Chapter 9`, *Cryptocurrency Wallets*, covers the different wallet solutions that exist for keeping your cryptocurrency secure.

`Chapter 10`, *Alternate Blockchains*, examine blockchain creation technologies such as Tendermint and Graphene, and other non-currency based blockchain technologies.

`Chapter 11`, *Hyperledger and Enterprise Blockchains*, examine the Hyperledger family of distributed ledger technologies aimed at corporate and enterprise use.

`Chapter 12`, *Ethereum 101*, look at Ethereum, the second most dominant blockchain technology today.

`Chapter 13`, *Solidity 101*, cover the basics of Solidity, the Ethereum programming language.

`Chapter 14`, *Smart Contracts*, covers the smart contracts, which are enabled in different ways by different blockchain technologies.

`Chapter 15`, *Ethereum Development*, look at writing applications for the Ethereum blockchain.

`Chapter 16`, *Ethereum Accounts and Ether Token,* in this chapter, we look at the mechanics of Ethereum accounts and the token itself in the Ethereum system.

`Chapter 17`, *Decentralized Applications*, discusses decentralized applications as a whole, including ones that operate without a blockchain or in tandem with blockchain technologies.

`Chapter 18`, *Mining*, we cover blockchain mining and how this is used to secure blockchains, the different types of hardware used in mining, and the different protocols involved.

`Chapter 19`, *ICO 101*, we cover the basics of launching an Initial Coin Offering or Initial Token Offering.

`Chapter 20`, *Creating Your Own Currency*, we cover the creation of your own blockchain based cryptocurrency.

`Chapter 21`, *Scalability and Other Challenges*, covers the difficulties and limitations currently facing blockchain technology.

`Chapter 22`, *Future of Blockchain*, we examine the possible future developments of the industry technologically, legally, and socially.

To get the most out of this book

1. Let go of the hype and really try to understand what blockchain does and doesn't do.
2. Check out the projects mentioned and see which, if any, apply to your plans. You might save yourself a lot of work.
3. Think about how decentralization might help or hurt your goals.

Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Blockchain-Quick-Reference`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: " Various options and commands can be checked by using the `geth --help` command."

A block of code is set as follows:

```
difficulty = difficulty_1_target/current_target
difficulty_1_target =
0x00000000FFFF0000000000000000000000000000000000000000000000000000
```

Any command-line input or output is written as follows:

```
COMMANDS:
 list Print summary of existing accounts
 new Create a new account
 update Update an existing account
 import Import a private key into a new account
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

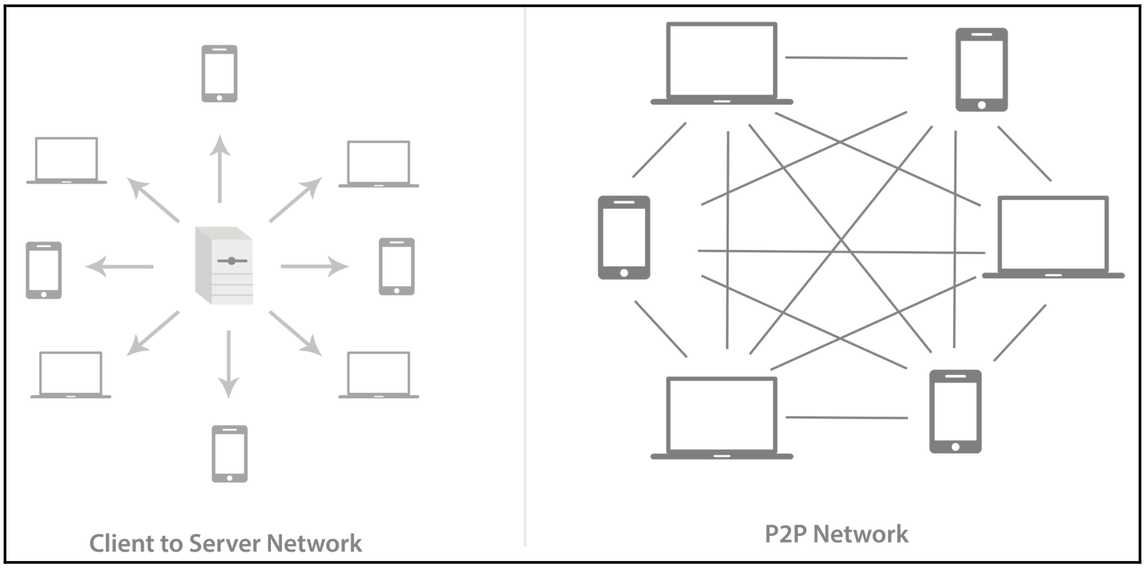
If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

1
Blockchain 101



Since its inception in 2008, blockchain has been a keen topic of interest for everyone in finance, technology, and other similar industries. Apart from bringing a new overview to record keeping and consensus, blockchain has enormous potential for disruption in most industries. Early adopters, enthusiasts, and now governments and organizations are exploring the uses of blockchain technology.

In this book, we will discuss the basics of financial transactions using fiat money to create our own cryptocurrency based on Ether tokens, and, in so doing, we will try to cover the majority of topics surrounding blockchain. We will be discussing Ethereum-based blockchains, Hyperledger projects, wallets, altcoins, and other exciting topics necessary to understand the functioning and potential of blockchain.

In this chapter, we will discuss the following:

- Financial transactions
- Financial ledger
- P2P networks
- General elements of a blockchain
- Uses and benefits of blockchain
- Various types of blockchain
- Introducing the consensus system
- Challenges faced by a blockchain network

Let's start by discussing each of the preceding listed topics and other important details surrounding them.

Processing a financial transaction

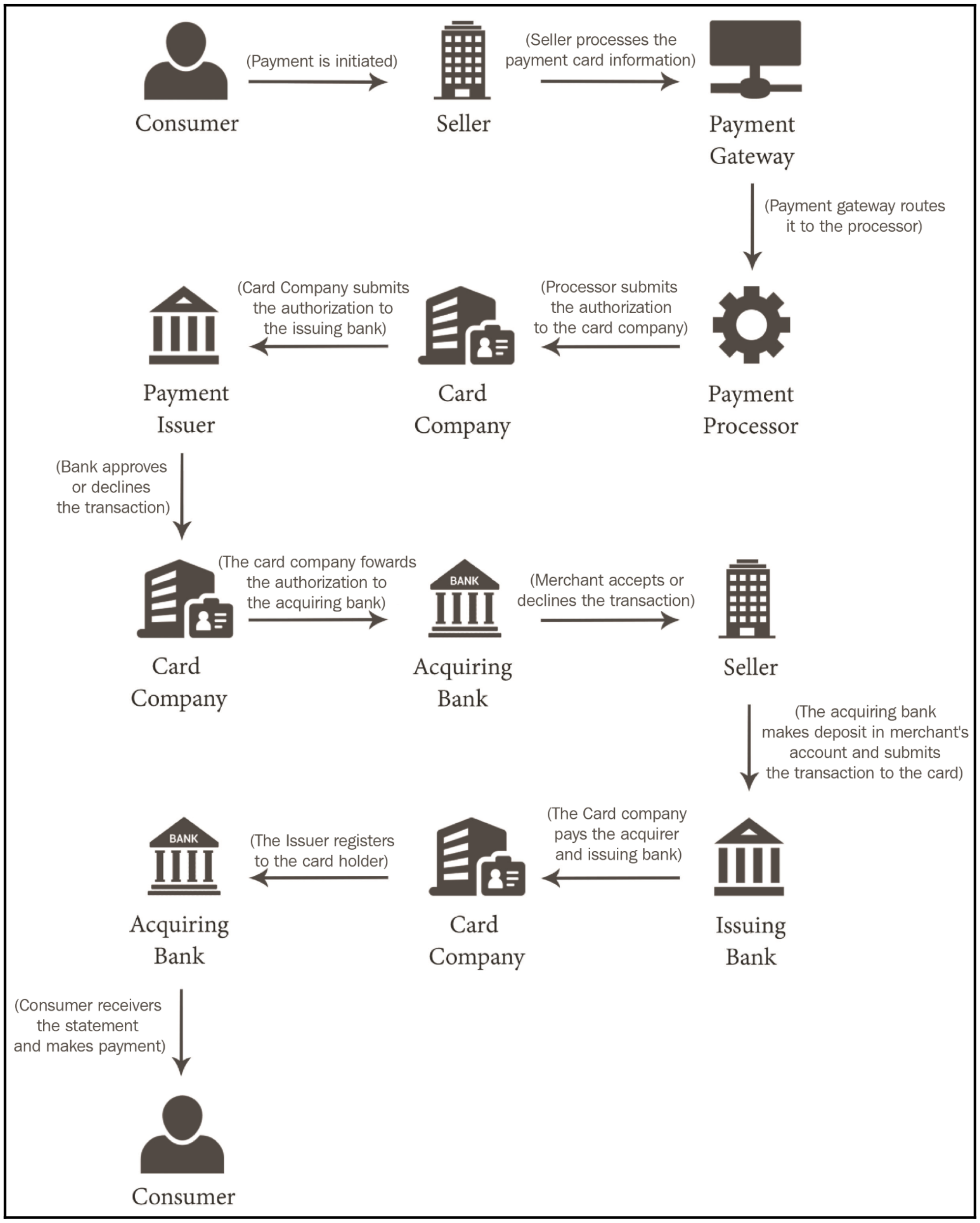
Before we dig deeper into blockchain-based transactions, it is helpful to know about how financial transactions actually happen and the functioning of fiat money.

Fiat money is entirely based on the credit of the economy; by definition, it is the money declared legal tender by the government. Fiat money is worthless without a guarantee from the government.

Another type of money is **commodity money**; it is derived from the commodity out of which the good money is made. For example, if a silver coin is made, the value of the coin would be its value in terms of silver, rather than the defined value of the coin. Commodity money was a convenient form of trade in comparison to the barter system. However, it is prone to huge fluctuations in price.

Commodity money proved to be difficult to carry around, so, instead, governments introduced printed currency, which could be redeemed from the government-based banks for actual commodity, but then, even that proved to be difficult for the government to manage, and it introduced fiat-based currency, or faith-based currency.

Having fiat-based currencies incurred a lot of third-party consensus during its time; this would help eradicate fraud from the system. It is also necessary to have a stringent consensus process to make sure that the process, as well as the privacy, is maintained within the system. The following diagram depicts the process of a credit card-based payment process:


Consumer
(Payment is initiated)
Seller
(Seller processes the payment card information)
Payment Gateway
(Payment gateway routes it to the processor)
Payment Processor
(Processor submits the authorization to the card company)
Card Company
(Card Company submits the authorization to the issuing bank)
Payment Issuer
(Bank approves or declines the transaction)
Card Company
(The card company fowards the authorization to the acquiring bank)
BANK
Acquiring Bank
(Merchant accepts or declines the transaction)
Seller
(The acquiring bank makes deposit in merchant's account and submits the transaction to the card)
Issuing Bank
(The Card company pays the acquirer and issuing bank)
Card Company
(The Issuer registers to the card holder)
BANK
Acquiring Bank
(Consumer receivers the statement and makes payment)
Consumer

The process of a credit card-based payment and the need for multiple third-party reference points to maintain trust.

Ledger

A ledger is a record for economic transactions that includes cash, accounts receivable, inventory, fixed assets, accounts payable, accrued expenses, debt, equity, revenue, costs, salaries, wages, expenses, depreciation, and so on. In short, the book in which accounts are maintained is called a **ledger**. It is the primary record used by banks and other financial institutions to reconcile book balances. All the debits and credits during an accounting period are calculated to make the ledger balance.

The financial statements of banks, financial institutions, and enterprises are compiled using ledger accounts.

Concept of a trustless system

While doing a financial transaction using fiat currency, we have a third-party ledger that maintains information about every transaction. Some of these third-party trust systems are VISA, MasterCard, banks, and so on.

Blockchain has changed the landscape of this trustless system by making everyone part of the ledger. Hence, it is sometimes even called a **distributed ledger**; everybody doing a transaction in blockchain has a record of other transactions that have happened or are happening in the blockchain-based Bitcoin system. This decentralized ledger gives multiple authenticity points for every transaction that has happened; plus, the rules are pre-defined and not different for each wallet user.

On a further note, blockchain does not actually eliminate trust; what it does is minimize the amount of trust and distributes it evenly across the network. A specific protocol is defined using various rules that automatically encourage patrons on the basis of the rules followed. We will be discussing this in depth in later chapters.

Introducing blockchain

The whitepaper released by Bitcoin's founder or a group of founders called Satoshi Nakamoto, in 2008, described Bitcoin as a purely peer-to-peer version of electronic cash. Blockchain was introduced along with Bitcoin. During the initial stages, blockchain was only used with Bitcoin for Bitcoin-based financial transactions.

Blockchain not only restricts financial transactions in Bitcoin, but in general any transaction between two parties that is maintained by the open, decentralized ledger. Most importantly, this underlying technology can be separated and can have other applications create a surge in the number of experiments and projects surrounding the same.

Numerous projects inspired by blockchain started, such as Ethereum, Hyperledger, and so on, along with currencies such as Litecoin, Namecoin, Swiftcoin, and so on.

Blockchain at its core is a distributed and decentralized open ledger that is cryptographically managed and updated various consensus protocols and agreements among its peers. People can exchange values using transactions without any third party being involved, and the power of maintaining the ledger is distributed among all the participants of the blockchain or the node of the blockchain, making it a truly distributed and decentralized system.

Some of the industry verticals using blockchain are as follows:

- **Cryptocurrency**: Bitcoin is the prime example of this. Apart from this, there are various alternative cryptocurrencies, such as Ethereum, Litecoin, Peercoin, and so on.
- **Cybersecurity**: There are various companies and projects harnessing the distributed nature of blockchain to create special authentication protocols for mass usage.
- **Healthcare**: Scalability is one of the best catalysts of blockchain; a lot of projects are securely storing data and using analytics to come to the perfect solution. Providing decentralized patient records and authentication is already being used at various facilities.

- **Financial services**: A lot of insurance and financial institutions are using blockchain to maintain the vast bank data, such as financial records, ledgers, guarantees, bonds, and so on.
- **Manufacturing**: Supply chain, prototyping and proof-of-concept, along with the tracking and tracing of goods is utilizing blockchain at the best efficiency.
- **Governance**: A lot of governments are racing toward becoming the world's first blockchain-powered state. Blockchains are being used across most government departments and not restricted only to public safety, transport, health, and shipping.
- **Retail**: A lot of startups and projects are aiming to introduce open no-middle man-based ecosystems; some are working on loyalty systems and blockchain-derived gift card systems.
- **Media**: Record labels can use blockchains to keep their ownership network and intellectual property rights secure.
- **Travel**: Projects are being worked on to introduce and revolutionize vehicle leasing, ride sharing, and other travel-related queries.
- **Legal**: Blockchain can bring transparency and solve the scaling issues in the ever-complex legal systems that are in place.

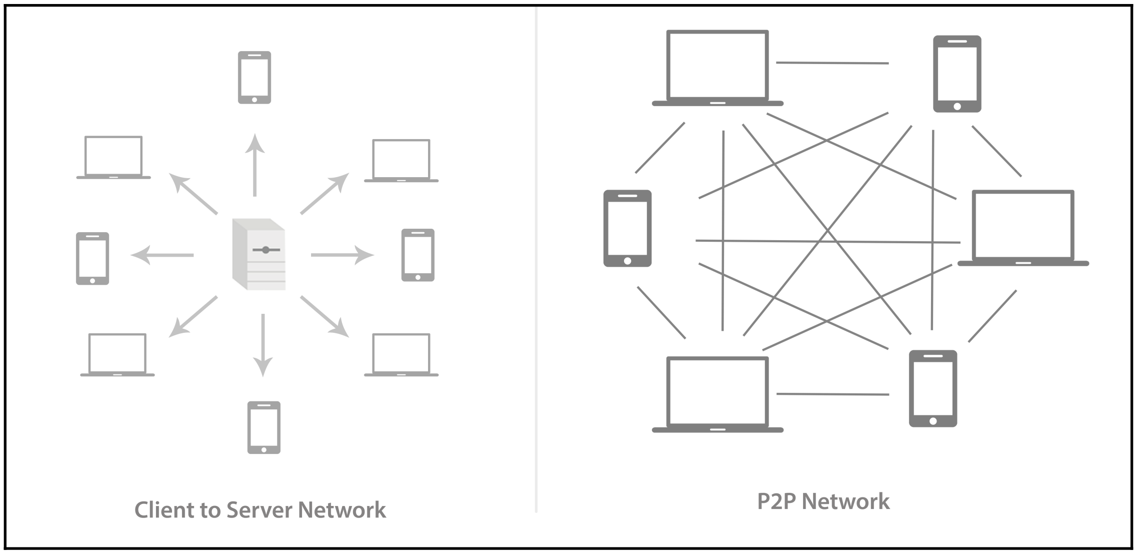
Furthermore, we will be discussing various other elements of blockchain and what other problems blockchain can solve.

General elements of blockchain

It is time to discuss the general elements of blockchain, starting from its basic structure to its formation and further details on the same.

Peer-to-peer network

This is a type of network whereby all peers can communicate with one another and are equally entitled, without the need for central coordination by servers or hosts. In conventional networks, the systems are connected to a central server, and this server acts as a central point for communication among the systems. On the other hand, in a peer-to-peer network, all the systems are connected to one another evenly, with no system having central authority. Look at this diagram:



The pictorial difference between a client-to-server network and a peer-to-peer network

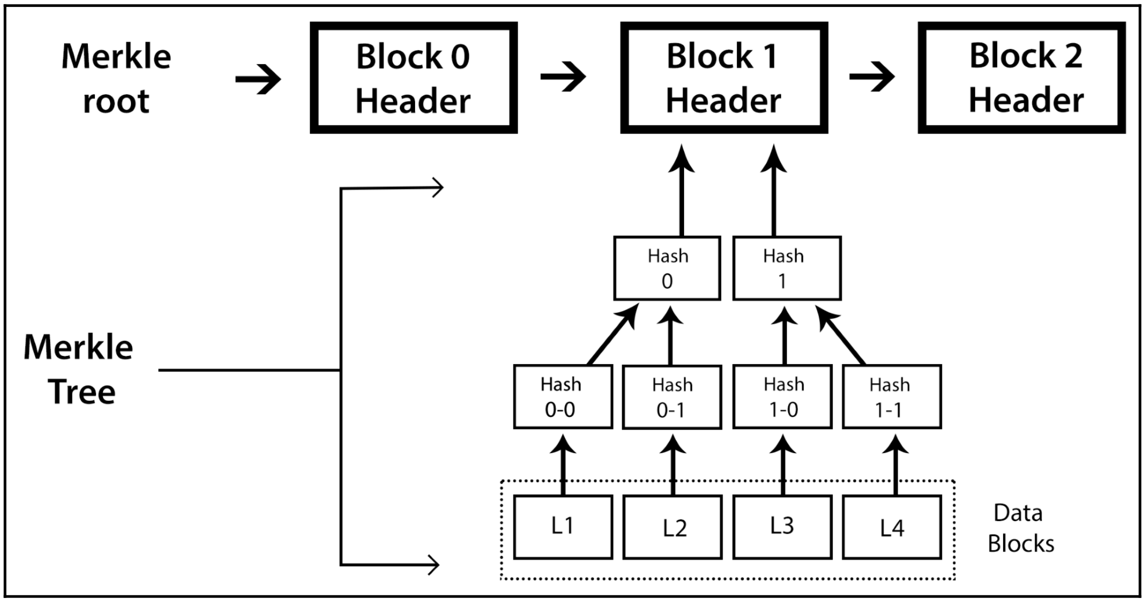
Block

A **block** is the smallest element of a blockchain; the first block is called the **genesis block**. Each block contains batches of hashed and encoded transactions. The blocks are stored in a Merkle tree formation. Every block includes the hash of the previous block in the chain that links all blocks to one another. In Bitcoin, a block contains more than 500 transactions on average. The average size of a block is around 1 MB. A block is comprised of a header and a list of transactions.

Block header

The block header of a block in Bitcoin comprises of metadata about the block. Consider the following:

- **Bitcoin version**: This field contains the Bitcoin version number.
- **Previous block hash**: The previous block's hash is required to create the new block's hash.
- **Merkle root**: This is the hash of the root of the Merkle tree of the current block's transactions.
- **Timestamp**: This is the timestamp of the block in UNIX.
- **Mining difficulty**: Mining is a crucial part of the blockchain ecosystem in Bitcoin. There is a difficulty target for mining, which is mentioned in the header.
- **Nonce**: Blockchain adds deviations in each block; these are known as nonce. Take a look at this diagram:



The formation of block headers and what comprises the Merkle root and the Merkle tree

Addresses

Addresses are unique identifiers that are used in a transaction on the blockchain to send data to another address; in the case of Bitcoins, addresses are identifiers that are used to send or receive Bitcoins. Bitcoin blockchain addresses have evolved from time to time. Originally, IP addresses were used as the Bitcoin address, but this method was prone to serious security flaws; hence, it was decided to use P2PKH as a standard format. A P2PKH address consists of 34 characters, and its first character is integer 1. In literal terms, **P2PKH** means **Pay to Public Key Has**. This is an example of a Bitcoin address based on P2PKH: 1PNjry6F8p7eaKjjUEJiLuCzabRyGeJXxg.

Now, there is another advanced Bitcoin protocol to create a **P2SH** address, which means **Pay to Script Hash**. One major difference with a P2SH address is that it always starts with integer 3 instead of 1.

Wallets

A **wallet** is a digital wallet used to store public or private keys along with addresses for transaction. There are various types of wallets available, each one offering a certain level of security and privacy.

Here is a list of the various types of wallets, based on their functions:

- **Software**: This wallet is installed on the actual computer; the private key access is with the owner of the machine on which the wallet's software is installed.
- **Web Wallets**: These wallets are based on the cloud and can be accessed anywhere. The private keys are shared with the wallet service.
- **Paper Wallets**: The private keys of this wallet are printed on paper.
- **Hardware**: These are physical wallets and are small and portable in nature. The private keys are with the hardware users of each wallet.

It is important to understand the functioning and the need for various wallets along with the requirement for each.

Transaction

A **transaction** is the process of transferring data from one address in blockchain to another address. In Bitcoin, it is about transferring Bitcoins from one address to another address. All the transactions happening in blockchain are registered from the start of the chain till the current time; this information is shared across the network and all the P2P nodes. The transaction is confirmed by miners, who are economically compensated for their work.

Each transaction in a blockchain goes through a number of confirmations, as they are the consensus of a transaction. Without confirmation, no transaction can be validated.

Nodes

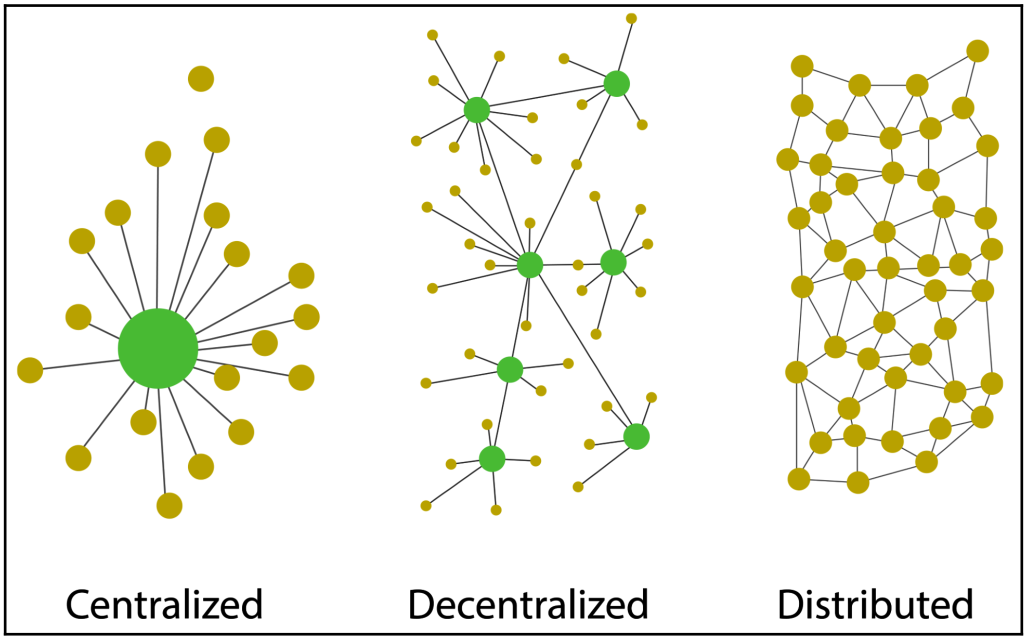
Nodes are part of a blockchain network and perform functions as assigned to them. Any device connected to the Bitcoin network can be called a **node**. Nodes that are integral components of the network and validate all the rules of the blockchain are called **full nodes**. Another type of Node is called a **super node**, which acts as a highly connected redistribution point, as well as a relay station.

What does blockchain solve?

A blockchain performs various functions. We will discuss each of them briefly here and in detail later:

- **Security**: Due to its consensus structure and multiple failback points, there is minimum chance of failure. Its distributed nature gives better security and reliability.
- **Faster settlements**: Traditional banking protocols are very time-consuming and incur fairly large forex charges; on the other hand, Bitcoins based on blockchain offer near-instant speed, saving time and costs for the entire financial industry.
- **Transparency**: Being decentralized in nature, there is no need for a third party, as blockchain is shared with everyone with a wallet, making it a transparent system with trust.
- **Economical**: Having no third party and the ledger being shared by everyone means no overhead costs or auditing expenses.

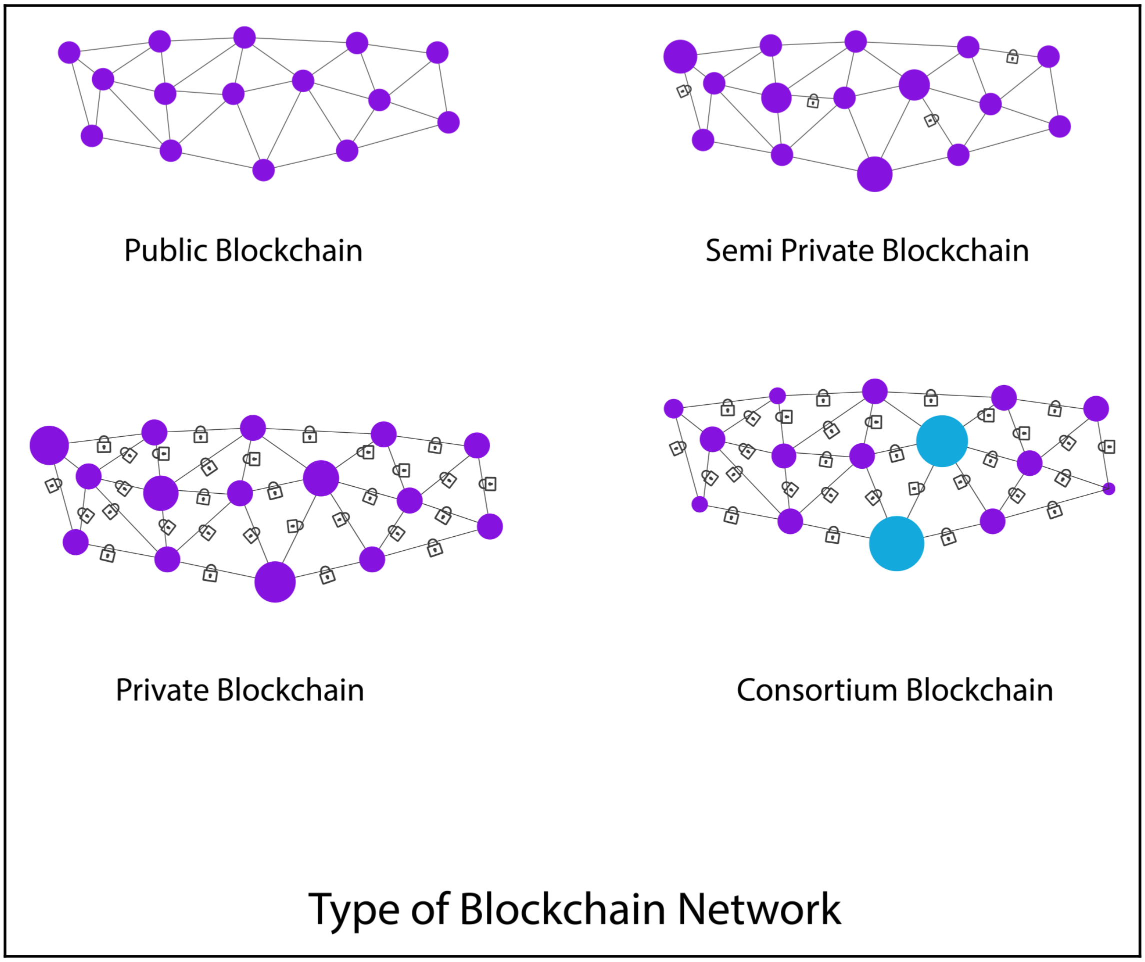
The following diagram depicts the difference between centralized, decentralized, and distributed networks:



Being distributed in nature, blockchain offers lots of out-of-the-box features, such as high stability, security, scalability, and other features discussed previously.

Types of blockchains

Considering the way blockchain has evolved, we can classify blockchain into multiple types; these types define the course of blockchain and make it go beyond the use of P2P money. The following diagram displays the different types of blockchain networks currently available or proposed.



We will now discuss each type of blockchain network in detail.

Public blockchain

A **public blockchain** is a blockchain where anyone in the world can become a node in the transaction process. Economic incentives for cryptographic verification may or may not be present. It is a completely open public ledger system. Public blockchains can also be called **permissionless ledgers**.

These blockchains are secured by crypto economics, that is, economic incentives and cryptographic verification using mechanisms such as PoW or PoS or any other consensus mechanism. Some popular examples of this type of blockchain are Bitcoin, Ethereum, Litecoin, and so on.

Semi-private blockchain

A semi-private blockchain is usually run by a single organization or a group of individuals who grant access to any user, who can either be a direct consumer or for internal organizational purposes. This type of blockchain has a public part exposed to the general audience, which is open for participation by anyone.

Private blockchain

In private blockchains, the write permissions are with one organization or with a certain group of individuals. Read permissions are public or restricted to a large set of users. Transactions in this type of blockchain are to be verified by very few nodes in the system.

Some prime examples of private blockchain include Gem Health network, Corda, and so on.

Consortium blockchain

In this type of blockchain, as the name suggests, the consensus power is restricted to a set of people or nodes. It can also be known as a **permission private blockchain**. Transaction approval time is fast, due to fewer nodes. Economic rewards for mining are not available in these types of blockchains.

A few examples of consortium-based blockchains are Deutsche Boerse and R3 (financial institutions).

Byzantine generals problem

This is one of the classic problems faced by various computer networks, which until recently had no concrete solution. This problem is called **Byzantine Generals' Problem** (**BGP**). The problem at its root is about consensus, due to mistrust in the nodes of a network.

Let's imagine that various generals are leading the Byzantine army and are planning to attack a city, with each general having his own battalion. They have to attack at the same time to win. The problem is that one or more of generals can be disloyal and communicate a duping message. Hence, there has to be a way of finding an efficient solution that helps to have seamless communication, even with deceptive generals.

This problem was solved by Castro and Liskov, who presented the **Practical Byzantine Fault Tolerance** (**PBFT**) algorithm. Later, in 2009, the first practical implementation was made with the invention of Bitcoin by the development of PoW as a system to achieve consensus.

We will be discussing in detail the BGP in later chapters.

Consensus

Consensus is the process of reaching a general agreement among nodes within a blockchain. There are various algorithms available for this especially when it is a distributed network and an agreement on a single value is required.

Mechanisms of consensus: Every blockchain has to have one mechanism that can handle various nodes present in the network. Some of the prime mechanisms for consensus by blockchain are the following:

- **Proof of Work** (**PoW**): This is the most commonly used consensus mechanism, also used by the first ever cryptocurrency, Bitcoin. This algorithm has proven most successful against Sybil attacks.
- **Proof of Stake** (**PoS**) this makes the mining of new blocks easier for those who have the highest amount of cryptocurrency.
- **Delegated Proof of Stake** (**DPOS**) one small change it has over PoS is that each node that has a stake can delegate the validation of a transaction to other nodes by means of voting.
- **Proof of Importance** (**POI**) this is designed to be energy efficient and can also run on relatively less powerful machines. It relies on stake as well as the usage and movement of tokens to establish trust and importance.

- **Proof of Elapsed Time** (**PoET**) this is a blockchain algorithm created by Intel, using **Trusted Execution Environment** (**TEE**) to have randomness and security in the voting process using a guaranteed wait time.
- **Proof of burn** (**PoB**) this is mostly used for bootstrapping one cryptocurrency to another. The basic concept is that miners should prove that they have burned coins, that is, they have sent them to a verifiable unspendable address.
- **Proof of activity** (**PoA**): A random peer is selected in this from the entire network to sign a new block that has to be tamper-proof.

All the preceding algorithms and a host of already available or currently under research make sure that the perfect consensus state is achieved and no possible security threats are present on the network.

Blockchain in a nutshell

It is time to discuss the benefits as well as the challenges or limitations faced by blockchain technology, and what steps are being taken by the community as a whole.

Benefits

If it's all about trust and security, do we really need a trusted system, even after everything is already highly secure and private? Let's go through the limitations in each of the existing ecosystems where blockchain is a perfect fit.

Banking records

Record keeping and ledger maintenance in the banking sector is a time and resource-consuming process, and is still prone to errors. In the current system, it is easy to move funds within a state, but when we have to move funds across borders, the main problems faced are time and high costs.

Even though most money is just an entry in the database, it still incurs high forex costs and is incredibly slow.

Medical records

There are lot of problems in record keeping, authentication and transferring of records at a global scale, even after having electronic records, are difficult when implemented practically. Due to no common third party, a lot of records are maintained physically and are prone to damage or loss.

During a case of epidemiology, it becomes essential to access and mine medical records of patients pertaining to a specific geography. Blockchain comes as a boon in such situation, since medical records can be easily accessible if stored in the blockchain, and they are also secure and private for the required users.

Government records

Any government agency has to deal with a lot of records for all of its departments; new filings can be done on blockchain, making sure that the data remains forever secure and safe in a distributed system.

This transparency and distributed nature of data storage leads to a corruption-free system, since the consensus makes sure the participants in the blockchain are using the required criteria when needed.

Creative and copyright records

Copyright and creative records can be secured and authenticated, keeping a tab on copyright misuse and licensing.

One premier example of this is KodakCoin, which is a photographer-oriented cryptocurrency based on blockchain, launched to be used for payments of licensing photographs.

University degree records

Verification, authentication, and inspection is hard. It is highly prone to theft and misuse. Blockchain can offer a great semi-private access to the records, making sure signing of degrees is done digitally using blockchain.

Gradual record keeping of degrees and scores will benefit efficient utilization of resources as well as proper distribution and ease in inspection process.

The preceding are just some of the varied use cases of blockchain, apart from Bitcoins and alternative cryptocurrencies. In the coming chapters, we will be discussing these points in much more detail.

Challenges

As with any technology, there are various challenges and limitations of blockchain technology. It is important to address these challenges and come up with a more robust, reliable, and resourceful solution for all. Let's briefly discuss each of these challenges and their solutions.

Complexity

Blockchain is complex to understand and easy to implement.

However, with widespread awareness and discussions, this might be made easier in the future.

Network scalability

If a blockchain does not have a robust network with a good grid of nodes, it will be difficult to maintain the blockchain and provide a definite consensus to the ongoing transactions.

Speed and cost

Although blockchain-based transactions are very high in speed and also cheaper when compared to any other conventional methods, from time to time, this is becoming difficult, and the speed reduces as the number of transactions per block reduces.

In terms of cost, a lot of hardware is required, which in turn leads to huge network costs and the need for an intermittent network among the nodes.

Various scaling solutions have been presented by the community. The best is increasing the block size to achieve a greater number of transactions per block, or a system of dynamic block size. Apart from this, there are various other solutions also presented to keep the speed reduced and the costs in check.

Double spending

This is a type of attack on the blockchain network whereby a given set of coins is spent in more than one transaction; one issue that was noted here by the founder/founders of Bitcoin at the time of launch is **51 attacks**. In this case, if a certain miner or group of miners takes control of more than half of the computing power of blockchain, being open in nature, anyone can be a part of the node; this triggers a 51 attack, in which, due to majority control of the network, the person can confirm a wrong transaction, leading to the same coin being spent twice.

Another way to achieve this is by having two conflicting transactions in rapid succession in the blockchain network, but if a lot of confirmations are achieved, then this can be avoided.

There are various other features that will be discussed in the coming chapters, it should be noted that all of these features exist in the present systems but considering active community support, all these limitations are being mitigated at a high rate.

Summary

This chapter introduced us to blockchain. First, ideas about distributed networks, financial transactions, and P2P networks were discussed. Then, we discussed the history of blockchain and various other topics, such as the elements of blockchain, the types of blockchains, and consensus.

In the coming chapters, we will be discussing blockchain in more detail; we will discuss the mechanics behind blockchain, Bitcoins. We will also learn about achieving consensus in much greater detail, along with diving deep into blockchain-based applications such as wallets, Ethereum, Hyperledger, all the way to creating your own cryptocurrency.

2 Components and Structure of Blockchain

Blockchain is not a single technology, but more of a technique. A blockchain is an architectural concept, and there are many ways that blockchains can be be built, and each of the variations will have different effects on how the system operates. In this chapter, we are going to cover the aspects of blockchain technology that are used in all or most of the current implementations.

By the end of this chapter, you should be able to describe the pieces of a blockchain and evaluate the capacities of one blockchain technology against another at the architectural level.

Here are the concepts we will be covering:

- Blocks
- The chain between blocks
- Hashing and signatures
- Block structure
- Block miners
- Block validators
- Smart contracts
- Blockchain speed

Blocks

Blockchain is a specific technology, but there are many forms and varieties. For instance, Bitcoin and Ethereum are proof-of-work blockchains. Ethereum has smart contracts, and many blockchains allow custom tokens. Blockchains can be differentiated by their consensus algorithm (PoS, PoW, and others)—covered in `Chapter 7`, *Achieving Consensus*, and their feature set, such as the ability to run smart contracts and how those smart contracts operate in practice. All of these variations have a common concept: the block. The most basic unit of a blockchain is the block. The simplest way of thinking of a block is to imagine a basic spreadsheet. In it, you might see entries such as this:

Account	**Change**	**New Balance**	**Old Balance**	**Operation**
Acct-9234222	–$2,000	$5,000	$7,000	Send funds to account-12345678
Acct-12345678	$2,000	$2,000	0	Receive funds from account-9234222
Acct-3456789	-$200	$50	$250	Send funds to account-68890234
Acct-68890234	$200	$800	$600	Receive funds from account-3456789

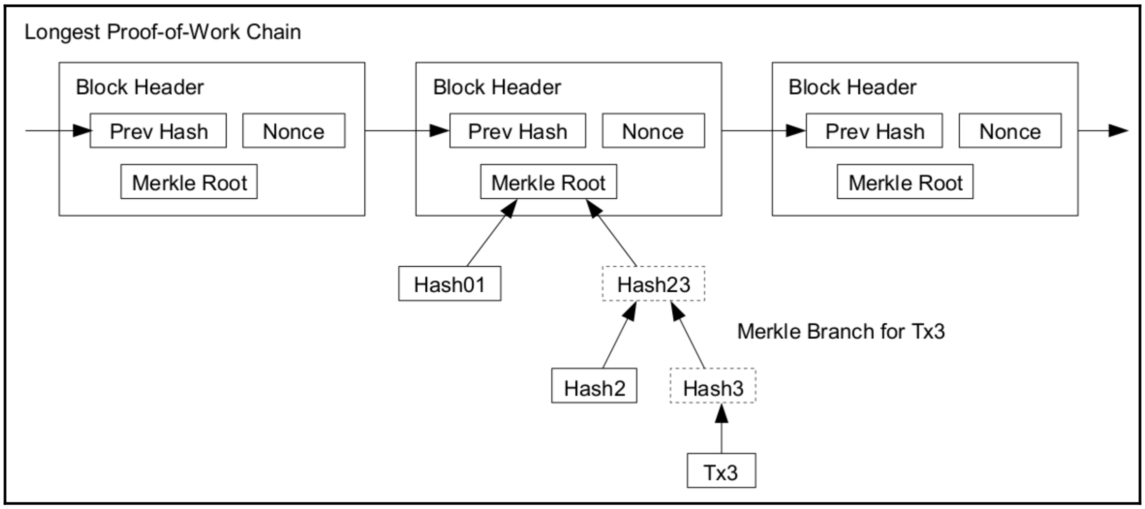
A block is a set of transaction entries across the network, stored on computers that act as participants in the blockchain network. Each blockchain network has a block time, or the approximate amount of time that each block represents for transactions, and a block size: the total amount of transactions that a block can handle no matter what. If a network had a block time of two minutes and there were only four transactions during those two minutes, then the block would contain just those four transactions. If a network had 10,000,000 transactions, then there may be too many to fit inside the block size. In this case, transactions would have to wait for their turn for an open block with remaining space. Some blockchains handle this problem with the concept of network fees. A network fee is the amount (denominated in the blockchain's native token) that a sender is willing to pay to have a transaction included in a block. The higher the fee, the greater the priority to be included on the chain immediately.

The chain between blocks

In addition to the transaction ledger, each block typically contains some additional metadata. The metadata includes the following:

- A reference to the prior block
- Metadata about the network
- The Merkle root of the transactions, which acts as a check of the validity of the block

These basics tend to be common for all blockchains. Ethereum, Bitcoin, Litecoin, and others use this common pattern, and this pattern is what makes it a chain. Each chain also tends to include other metadata that is specific to that ecosystem, and those differences will be discussed in future chapters. Here is an example from the Bitcoin blockchain:



By Satoshi Nakamoto: http://Bitcoin.org/Bitcoin.pdf, MIT, https://commons.wikimedia.org/w/index.php?curid=24542868

If you are asking, What is a Merkle root?, that brings us to our next set of key concepts: hashing and signature.

Hashing and signatures

Let's say you have two text files that are 50 pages long. You want to know whether they are the same or different. One way you could do this would be to hash them. Hashing (or a hashing function) is a mathematical procedure by which any input is turned into a fixed-length output. There are many of these functions, the most common being SHA-1, SHA-2, and MD5. For instance, here is the output of a hashing function called **MD5** with an input of two pages of text:

```
9a137a78cf0c364e4d94078af1e221be
```

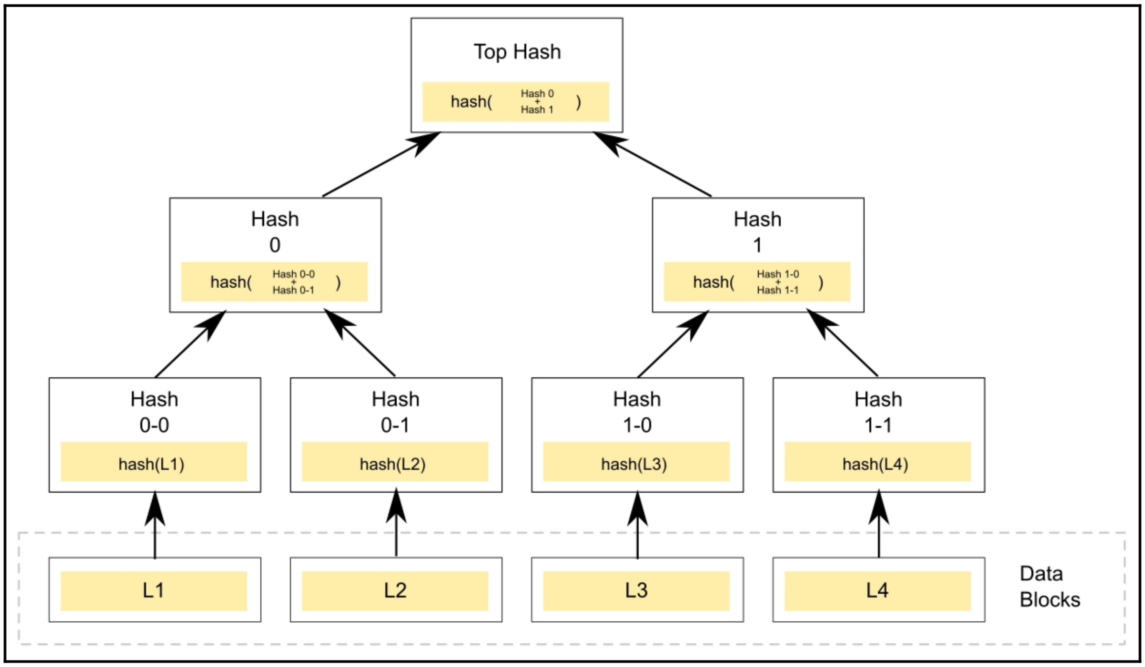
What's powerful about hashing functions is what happens when I add a single character to the end and run the same function:

```
8469c950d50b3394a30df3e0d2d14d74
```

As you can see, the output is completely different. If you want to quickly prove that some data has not been changed in any way, a hash function will do it. For our discussion, here are the important parts of hashing functions:

- They are very fast for computers to run.
- The function is one way. You can get the hash easily, but you cannot realistically use the hash to restore the original.
- They can be used recursively. For instance, I can take the hash of the hash; for example, MD5(`8469c950d50b3394a30df3e0d2d14d74`) becomes `705b003fc9b09ecbeac0b852dfc65377`.

This recursive property to hashing is what brings us to the concept of a **Merkle tree**, named after the man who patented it. A Merkle tree is a data structure that, if your were to draw it on a whiteboard, tends to resemble a tree. At each step of the tree, the root node contains a hash of the data of its children. The following is a diagram of a Merkle tree:



Original illustration by David Göthberg, Sweden, released to the public domain

In a blockchain, this means that there is a recursive hashing process happening. A recursive hash is when we take a hash of hashes. For instance, imagine we have the following words and their hashes. Here, we will use the MD5 algorithm, because it is easy to find MD5 hashing code on the web, so you can try for yourself:

```
Salad: c2e055acd7ea39b9762acfa672a74136
Fork: b2fcb4ba898f479790076dbd5daa133f
Spoon: 4b8e23084e0f4d55a47102da363ef90c
```

To take the recursive or the root hash, we would add these hashes together, as follows:

```
c2e055acd7ea39b9762acfa672a74136b2fcb4ba898f479790076dbd5daa133f4b8e23084e0
f4d55a47102da363ef90c
```

Then we would take the hash of that value, which would result in the following:

```
189d3992be73a5eceb9c6f7cc1ec66e1
```

This process can happen over and over again. The final hash can then be used to check whether any of the values in the tree have been changed. This root hash is a data efficient and a powerful way to ensure that data is consistent.

Each block contains the root hash of all the transactions. Because of the one-way nature of hashing, anyone can look at this root hash and compare it to the data in the block and know whether all the data is valid and unchanged. This allows anyone to quickly verify that every transaction is correct. Each blockchain has small variations on this pattern (using different functions or storing the data slightly differently), but the basic concept is the same.

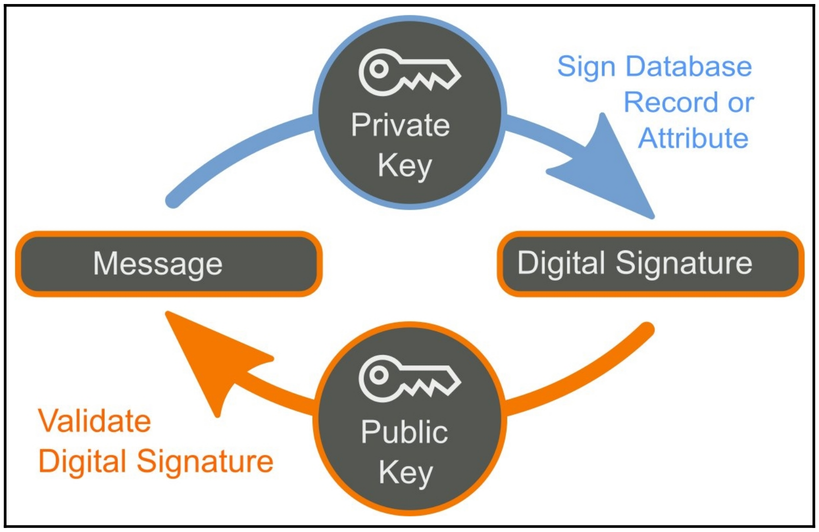
Digital signatures

Now that we've covered hashing, it's time to go over a related concept: **digital signatures**. Digital signatures use the properties of hashing to not only prove that data hasn't changed but to provide assurances of who created it. Digital signatures work off the concept of hashing but add a new concept as well: **digital keys**.

What are digital keys?

All common approaches to digital signatures use what is called **Public Key Cryptography**. In Public Key Cryptography, there are two keys: one public and one private. To create a signature, the first hash is produced of the original data, and then the private key is used to encrypt that hash. That encrypted hash, along with other information, such as the encryption method used to become part of the signature, are attached to the original data.

This is where the public key comes into play. The mathematical link between the public key and the private key allows the public key to decrypt the hash, and then the hash can be used to check the data. Thus, two things can now be checked: who signed the data and that the data that was signed has not been altered. The following is a diagrammatic representation of the same:



By Engelbert Niehaus with SVG color scheme by user Bananenfalter —Used SVG color of illustration by Bananenfalter and edit with Inkscape, CC BY-SA 3.0, https://en.wikiversity.org/w/index.php?curid=226250

This form of cryptography is critical to blockchain technology. Through hashing and digital signatures, a blockchain is able to record information both on actions (movement of tokens) as well as prove who initiated those actions (via digital signatures).

Let's create an example of how this would look. Jeremy and Nadia wish to send messages to each other securely. Each publishes a public key. Jeremy's will look something as follows (using an RSA algorithm with 1,024 bits):

```
-----BEGIN PUBLIC KEY-----
MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgH+CYOAgKsHTrMlsaZ32Gpdfo4pw
JRfHu5d+KoOgbmYb0C2y1PiHNGEyXgd0a8iO1KWvzwRUMkPJr7DbVBnfl1YfucNp
OjAsUWT1pq+OVQ599zecpnUpyaLyg/aW9ibjWAGiRDVXemj0UgMUVNHmi+OEuHVQ
ccy5eYVGzz5RYaovAgMBAAE=
-----END PUBLIC KEY-----
```

With that key, he will keep private another key, which looks as follows:

```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgH+CYOAgKsHTrMlsaZ32Gpdfo4pwJRfHu5d+KoOgbmYb0C2y1PiH
NGEyXgd0a8iO1KWvzwRUMkPJr7DbVBnfl1YfucNpOjAsUWT1pq+OVQ599zecpnUp
yaLyg/aW9ibjWAGiRDVXemj0UgMUVNHmi+OEuHVQccy5eYVGzz5RYaovAgMBAAEC
gYBR4AQYpk8OOr9+bxC6j2avwIegwzXuOSBpvGfMMV3yTvW0AlriYt7tcowSOV1k
YOKGqYdCflXwVTdtVsh//KSNiFtsLih2FRC+Uj1fEu2zpGzErhFCN2sv1t+2wjlk
TRY78prPNa+3K2Ld3NJse3gmhodYqRkxFFxlCmOxTzc4wQJBAOQ0PtsKCZwxRsyx
GAtULHWFIhV9o0k/DjLw5rreA8H3lb3tYZ5ErYuhS0HlI+7mrPUqzYaltG6QpJQY
YlMgktECQQCPClB1xxoIvccmWGoEvqG07kZ4OBZcBmgCzF6ULQY4JkU4k7LCxG4q
+wAeWteaP+/3HgS9RDQlHGITAmqhW6z/AkBaB16QzYnzC+GxmWAx//g2ONq0fcdw
eybf4/gy2qnC2SlDL6ZmaRPKVUy6Z2rgsjKj2koRB8iCIiA7qM8Jmn0xAkBzi9Vr
DqaNISBabVlW89cUnNX4Dvag59vlRsmv0J8RhHiuN0FT6/FCbvetjZxUUgm6CVmy
ugGVaNQgnvcb2T5pAkEAsSvEW6yq6KaV9NxXn4Ge4b9lQoGlR6xNrvGfoxto79vL
7nR29ZB4yVFo/kMVstU3uQDB0Pnj2fOUmI3MeoHgJg==
-----END RSA PRIVATE KEY-----
```

In the meantime, Nadia will do the same, resulting in the following two keys:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHWwgTfI5Tic41YjUZqTmiKt+R
s5OMKIEdHPTyM8FZNaOBWIosFQbYk266V+R7k9odTnwCfi370GOt0k5MdTQilb9h
bK/lYiavIltgBd+1Em7xm7UihwO4th5APcg2vG4sppK41b1a9/I5E6P/jpQ320vF
BMuEtcnBoWawWcbXJwIDAQAB
-----END PUBLIC KEY-----
```

This is her private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDHWwgTfI5Tic41YjUZqTmiKt+Rs5OMKIEdHPTyM8FZNaOBWIos
FQbYk266V+R7k9odTnwCfi370GOt0k5MdTQilb9hbK/lYiavIltgBd+1Em7xm7Ui
hwO4th5APcg2vG4sppK41b1a9/I5E6P/jpQ320vFBMuEtcnBoWawWcbXJwIDAQAB
AoGBAKz9FCv8qHBbI2H1f0huLQHInEoNftpfh3Jg3ziQqpWj0ub5kqSf9lnWzX3L
qQuHB/zoTvnGzlY1xVlfJex4w6w49Muq2Ggdq23CnSoor8ovgmdUhtikfC6HnXwy
PG6rtoUYRBV3j8vRlSo5PtSRD+H4lt2YGhQoXQemwlw+r5pRAkEA+unxBOj7/sec
3Z998qLWw2wV4p9L/wCCfq5neFndjRfVHfDtVrYKOfVuTO1a8gOal2Tz/QI6YMpJ
exo9OEbleQJBAMtlimh4S95mxGHPVwWvCtmxaFR4RxUpAcYtX3R+ko1kbZ+4Q3Jd
TYD5JGaVBGDodBCRAJALwBv1J/o/BYIhmZ8CQBdtVlKWCkk8i/npVVIdQB4Y7mYt
Z2QUwRpg4EpNYbE1w3E7OH27G3NT5guKsc4c5gcyptE9rwOwf3Hd/k9N10kCQQCV
YsCjNidS81utEuGxVPy9IqWj1KswiWu6KD0BjK0KmAZD1swCxTBVV6c6iJwsqM4G
FNm68kZowkhYbc0X5KG1AkBp3Rqc46WBbpE5lj7nzhagYz5Cb/SbNLSp5AFh3W5c
sjsmYQXfVtw9YuU6dupFU4ysGgLBpvkf0iU4xtGOFvQJ
-----END RSA PRIVATE KEY-----
```

With these keys, Jeremy decides to send a message to Nadia. He uses her key and encrypts the following message: I love Bitcoin, which results in the following data:

```
EltHy0s1W1mZi4+Ypccur94pDRHw6GHYnwC+cDgQwa9xB3EggNGHfWBM8mCIOUV3iT1
uIzD5dHJwSqLFQOPaHJCSp2/WTSXmWLohm5EAyMOwKv7M4gP3D/914dOBdpZyrsc6+a
D/hVqRZfOQq6/6ctP5/3gX7GHrgqbrq/L7FFc=
```

Nobody can read this, except Nadia. Using the same algorithm, she inputs this data and her private key, and gets the following message:

```
I love Bitcoin.
```

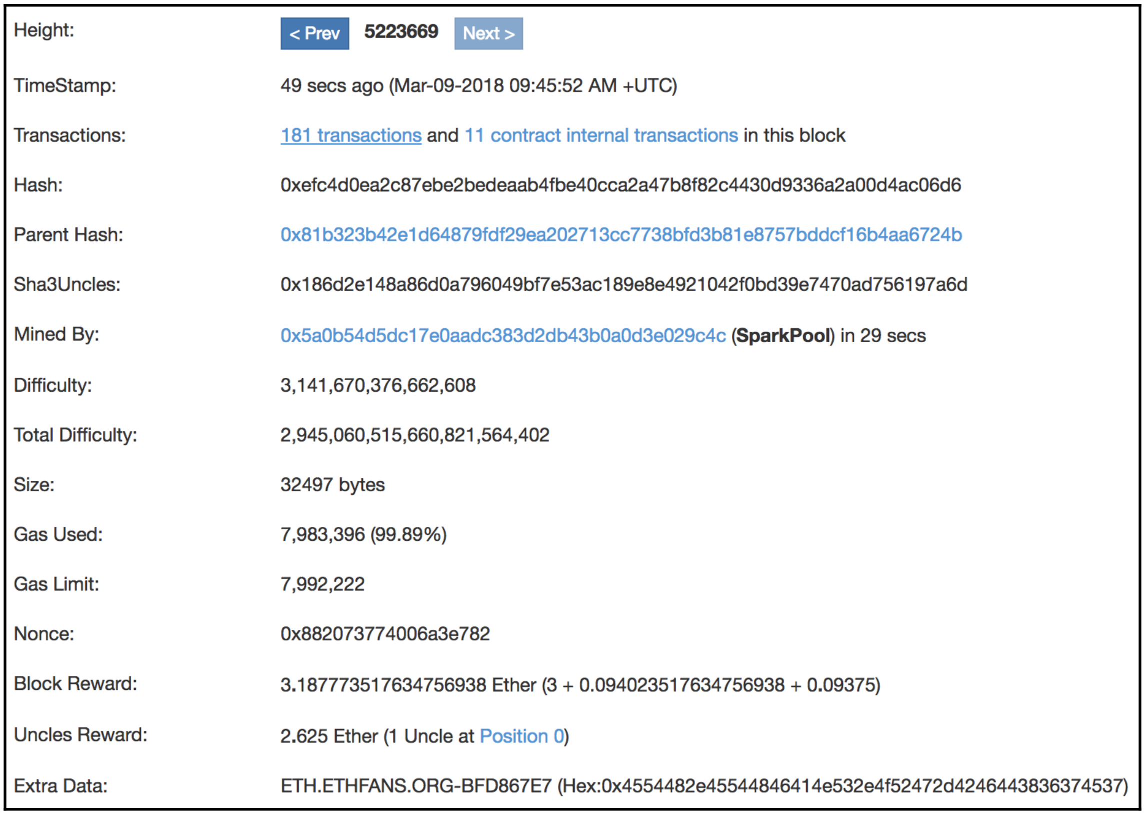
We'll discuss more about this topic in `Chapter 4`, *Cryptography and the Mechanics Behind Blockchain.*

Example block data

In this section, we are going to examine the data structures that are used in blockchains. We will be looking primarily at Ethereum, Bitcoin, and Bitshares blockchains to see key commonalities and differences.

Example Ethereum block

Here is the data from an example Ethereum block, from block **5223669**:

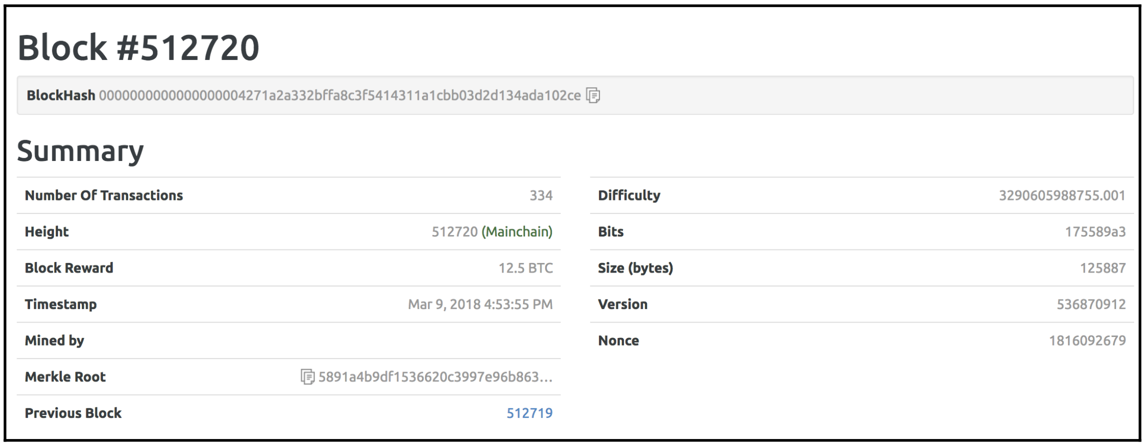


Height:	< Prev **5223669** Next >
TimeStamp:	49 secs ago (Mar-09-2018 09:45:52 AM +UTC)
Transactions:	181 transactions and 11 contract internal transactions in this block
Hash:	0xefc4d0ea2c87ebe2bedeaab4fbe40cca2a47b8f82c4430d9336a2a00d4ac06d6
Parent Hash:	0x81b323b42e1d64879fdf29ea202713cc7738bfd3b81e8757bddcf16b4aa6724b
Sha3Uncles:	0x186d2e148a86d0a796049bf7e53ac189e8e4921042f0bd39e7470ad756197a6d
Mined By:	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c (**SparkPool**) in 29 secs
Difficulty:	3,141,670,376,662,608
Total Difficulty:	2,945,060,515,660,821,564,402
Size:	32497 bytes
Gas Used:	7,983,396 (99.89%)
Gas Limit:	7,992,222
Nonce:	0x882073774006a3e782
Block Reward:	3.187773517634756938 Ether (3 + 0.094023517634756938 + 0.09375)
Uncles Reward:	2.625 Ether (1 Uncle at Position 0)
Extra Data:	ETH.ETHFANS.ORG-BFD867E7 (Hex:0x4554482e45544846414e532e4f52472d4246443836374537)

If you remember, at the beginning of the chapter, we said there were three things common to blockchains: the reference to the prior block, the **Hash** of the transactions in the block, and network-specific metadata. In this block from the Ethereum network, all three are present. The reference to the prior block is contained by the block height and parent hash values. The **Hash** of the transactions is the hash entry, and the metadata is everything else, which will be network specific.

Bitcoin block

Here is a snapshot of a Bitcoin block:



Block #512720

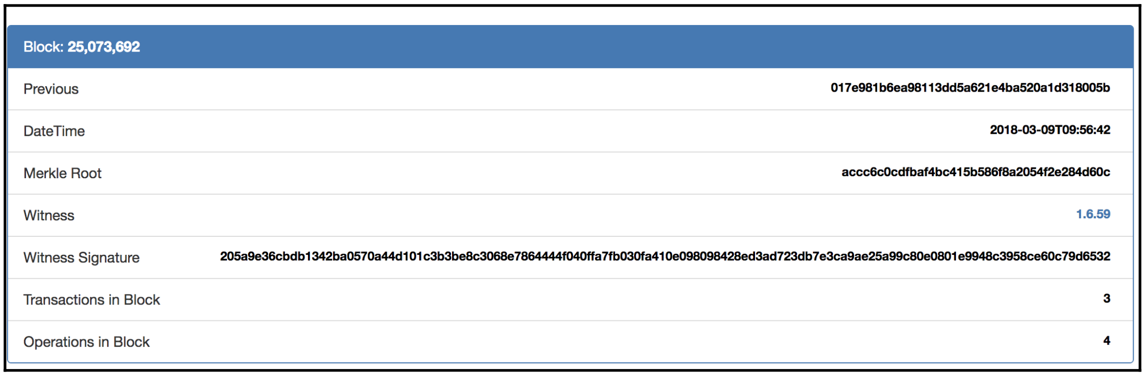
BlockHash 000000000000000000004271a2a332bffa8c3f5414311a1cbb03d2d134ada102ce

Summary

Number Of Transactions	334	**Difficulty**	3290605988755.001
Height	512720 (Mainchain)	**Bits**	175589a3
Block Reward	12.5 BTC	**Size (bytes)**	125887
Timestamp	Mar 9, 2018 4:53:55 PM	**Version**	536870912
Mined by		**Nonce**	1816092679
Merkle Root	5891a4b9df1536620c3997e96b863...		
Previous Block	512719		

Both Bitcoin and Ethereum are PoW chains; let's look now at a **Proof of Stake** (**POS**) ecosystem: Bitshares.

Here is some data from a Bitshares block:



Block: **25,073,692**	
Previous	**017e981b6ea98113dd5a621e4ba520a1d318005b**
DateTime	**2018-03-09T09:56:42**
Merkle Root	**accc6c0cdfbaf4bc415b586f8a2054f2e284d60c**
Witness	**1.6.59**
Witness Signature	**205a9e36cbdb1342ba0570a44d101c3b3be8c3068e7864444f040ffa7fb030fa410e098098428ed3ad723db7e3ca9ae25a99c80e0801e9948c3958ce60c79d6532**
Transactions in Block	**3**
Operations in Block	**4**

Despite a radically different architecture, the fundamentals remain: references to a previous block, Merkle root, and network metadata. In Bitshares, you can also see that there is a **Witness Signature**. As a PoS blockchain, Bitshares has validators (they are called witnesses). Here, we see the witness and signature of the computer responsible for calculating this block.

Global state

One of the key properties of blockchain technology is that it can act as a trusted global state. There are many applications where a trusted global state is important but difficult, such as financial technology and logistics.

For instance, a few years ago, I ordered some camera equipment online. A few days later, I came home and was surprised to find that my equipment had arrived. I was so thankful that the expensive equipment sitting outside had not been stolen. It was only the next day that I received an email from the seller alerting me that the package had been shipped.

Here is a clear breakdown of the global state. The truth was that the camera was already on a truck, but neither I nor the shipper had that information stored properly. If the camera equipment had been stolen from my porch, it would have been very hard to discover what had happened.

If the seller, the logistics company, and I were all writing and reading data from a blockchain, this would have been impossible. When the logistics company registered the shipment, the state of the object would have changed, and both the seller and I would have known as soon as the next block was finalized.

Block time and block size

As discussed before, each blockchain has a **block time** and a **block size** . Each network can have very different values and ways of handling block time. In Bitcoin, for instance, the block time is 10 minutes, while with Ethereum the block time is 20 seconds. In Stellar, the block time is about 4 seconds. These block times are determined by the code that runs the network. For networks such as Bitcoin, Litecoin, and Ethereum, the block time is actually an average. Because these are PoW networks, the block is finished once a miner solves the mining puzzle, which allows them to certify the block. In these networks, the difficulty of the puzzle is automatically adjusted so that on average the desired block time is reached.

The block size is the maximum amount of information that can be stored in each block. For Bitcoin, this is 1 MB of data's worth of transactions. For Ethereum, the limit is actually measured in GAS, a special unit of measuring both processing power (since Ethereum has smart contracts) as well as storage. Unlike Bitcoin, the GAS/storage limit for each block is not fixed but is instead adjusted by the miners dynamically.

It's important to note that blocks contain only possible information until they are finalized by the network. For instance, 1,000 transactions might happen, but if only 500 make it on to the next block, then only those 500 are real. The remaining transactions will continue to wait to be included into a future block.

Blockchain miners

Blockchain miners and blockchain validators (see the upcoming sections) both have to do with consensus, which will be explored in depth in `Chapter 7`, *Achieving Consensus*. Generally, blockchain miners are associated with blockchains. A PoW chain functions by having the computers that are miners compete to do the work needed to certify a block in the chain. Currently, the only major PoW blockchains are Bitcoin, Litecoin, and Ethereum. Most other systems use a variation of PoS consensus, which we will discuss in the next *Blockchain validators* section. We'll cover how mining works in detail in `Chapter 18`, *Mining.*

Blockchain validators

Blockchain validators are used by PoS systems. A PoS system works by requiring computers that wish to participate in the network to have **stake**—a large number of tokens—to assist in the blockchain. Unlike PoW algorithms, computers cannot join the network and expect to have any say in consensus. Rather, they must *buy in* through token ownership. Depending on the network, the naming convention for validators might be different. Tendermint has validators, Steemit and Bitshares have witnesses, Cardano has stakeholders, and so on. A validator is a computer with a positive stake (number of tokens) that is allowed to participate in the network and does so. Each chain has its own rules for how this works, and these will be covered more in-depth in `Chapter 7`, *Achieving Consensus*.

Smart contracts

Some blockchains are said to have smart contracts when they are able to perform actions and behavior in response to changes to the chain. These will be covered in depth in `Chapter 14`, *Smart Contracts* and `Chapter 17`, *Decentralized Applications*.

Blockchain speed

One ongoing concern for blockchain systems is performance. Public blockchains are global systems, with their system resources shared by all the users in the world simultaneously. With such a large user base, resource constraints are a real concern and have already caused real problems. For instance, a popular game called **CryptoKitties** was launched on Ethereum and caused the network to become congested. Other applications became nearly unusable, as the load from CryptoKitties overwhelmed the network.

How to calculate blockchain throughput

The quick and dirty way of calculating the throughput of a blockchain is as follows:

$$(blocksize/averagetransactionsize) = transactionsperblock$$

$$(1/blocktimeinseconds) = fractionsofablockpersecond$$

$$transactionsperblock * fractionsofablockpersecond = transactionsperblock$$

For Bitcoin, the transaction throughput is about *7tx/second*. This is because of the relatively small block and the very long block time. Ethereum has short block times but very tiny blocks and ends up at about *14tx/second*. Blockchains such as Stellar, Bitshares, and Waves can reach speeds of over, *1000tx/second*.

Comparisons with traditional networks

VISA is the premier payment-processing network worldwide. In one of the company's blogs, it was revealed that VISA can process over 40,000 transactions a second. This is peak capacity, and it usually processes nowhere near that, except around times such as Christmas. Nevertheless, it should be clear that blockchains have a way to go before they can compete for processing global payments on the same scale as VISA. Newer networks, such as EOS and COSMOS, are trying, however, with innovative multi-threaded designs and segmented blockchain zones.

Summary

Now you should understand the basic components of a blockchain. Blocks are groups of transactions grouped together and act as the fundamental unit of a blockchain. Miners are computers that create new blocks on PoW blockchains. Validators, also called witnesses and other names, are computers that create blocks on PoS blockchains. Digital signatures are composed of public and private keys and use mathematics to prove the author of the data.

The key ideas of hashing is to use a mathematical function that maps arbitrary data to a single, simple to deal with value. Any change to the data will make the end value very different

- It's essentially impossible to construct the original data from the hash, but it's easy to create the hash from the original data
- You can use these properties to prove that data has not been changed

In the next chapter, we will learn what these systems are and how blockchain counts as both. We will learn how to differentiate between the two systems and why these concepts are so important to blockchain.

3
Decentralization Versus Distributed Systems

One of the biggest misconceptions in the blockchain space is between distributed systems and decentralized systems. In this chapter, we are going to discuss both types of systems, why they matter, their similarities, their differences, and how blockchain technology can fit into both categories.

By the end of this chapter, you should be able to do the following:

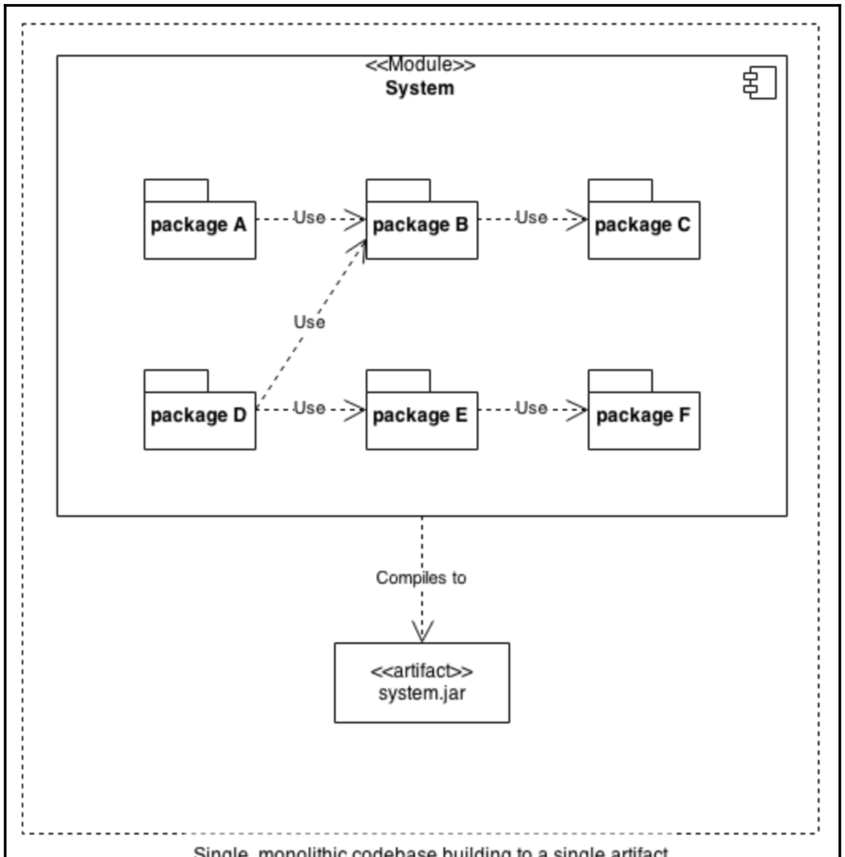
- Define a distributed system
- Define a decentralized system
- Understand the benefits and drawbacks of distributed systems
- Understand the benefits and drawbacks of decentralized systems

Distributed systems

A **distributed system** is one in which the application and its architecture is distributed over a large number of machines and preferably physical locations. More simply, a distributed system is one where the goal of the system is spread out across multiple sub-systems in different locations. This means that multiple computers in multiple locations must coordinate to achieve the goals of the overall system or application. This is different than monolithic applications, where everything is bundled together.

Let's take the example of a simple web application. A basic web application would run with processing, storage, and everything else running on a single web server. The code tends to run as a monolith—everything bundled together. When a user connects to the web application, it accepts the HTTP request, uses code to process the request, accesses a database, and then returns a result.

The advantage is that this is very easy to define and design. The disadvantage is that such a system can only scale so much. To add more users, you have to add processing power. As the load increases, the system owner cannot just add additional machines because the code is not designed to run on multiple machines at once. Instead, the owner must buy more powerful and more expensive computers to keep up. If users are coming from around the globe, there is another problem—some users who are near the server will get fast responses, whereas users farther away will experience some lag. The following diagram illustrates a single, monolithic code base building to a single artifact:



Single, monolithic codebase building to a single artifact

What happens if the computer running this application has a fault, a power outage, or is hacked? The answer is that the entire system goes down entirely. For these reasons, businesses and applications have become more and more distributed. Distributed systems typically fall into one of several basic architectures: client–server, three-tier, n-tier or peer-to-peer. Blockchain systems are typically peer-to-peer, so that is what we will discuss here.

The advantages of a distributed system are many, and they are as follows:

- **Resiliency**: If part of the system fails, the entire system does not fail
- **Redundancy**: Each part of the system can be built to have backups so that if it fails another copy can be used instead, sometimes instantly
- **Parallelism**: Work can be divided up efficiently so that many inexpensive computers can be used instead of a single (very expensive) fast computer

Resiliency

Resiliency is the ability of a system to adapt and keep working in response to changes and challenges. Resiliency can only be discussed in the context of the types of events that a system is resilient towards. A system might be resilient to a few computers getting turned off but may not be resilient to nuclear war.

Resiliency can be broken down into different sub-categories:

- **Fault tolerance**: The ability of the system to deal with invalid states, bad data, and other problems
- **Failure isolation**: A problem in one part of the system does not infect other parts of the system. Bad data or system failure in one place does not result in problems elsewhere
- **Scalability**: A scalable system under heavy use is able to provide additional capacity and is thus resilient to load
- **Complexity management**: A system that has ways of managing complexity helps it be resilient against human errors

We will now discuss fault tolerance in more detail.

Fault tolerance and failure isolation

A system is said to be fault tolerant when it is capable of operating even if some of the pieces fail or malfunction. Typically, fault tolerance is a matter of degree: where the level of sub-component failure is either countered by other parts of the system or the degradation is gradual rather than an absolute shutdown. Faults can occur on many levels: software, hardware, or networking. A fault tolerant piece of software needs to continue to function in the face of a partial outage along any of these layers.

In a blockchain, fault tolerance on the individual hardware level is handled by the existence of multiple duplicate computers for every function—the miners in bitcoin or proof of work systems or the validators in PoS and related systems. If a computer has a hardware fault, then either it will not validly sign transactions in consensus with the network or it will simply cease to act as a network node—the others will take up the slack.

Consensus and coordination

One of the most important aspects of blockchain is the concept of consensus. We will discuss the different ways blockchains achieve consensus in `Chapter 7`, *Achieving Consensus*. For now, it is enough to understand that most blockchain networks have protocols that allow them to function as long as two thirds to slightly over one-half of the computers on the network are functioning properly, though each blockchain network has different ways of ensuring this which will be covered in future chapters.

Backups

In most blockchains, each computer acting as a full participant in the network holds a complete copy of all transactions that have ever happened since the launch of the network. This means that even under catastrophic duress, as long as a fraction of the network computers remains functional, a complete backup will exist.

In PoS chains, there tends to be far fewer full participants so the number of backups and distribution is far less. So far, this reduced level of redundancy has not been an issue.

Consistency

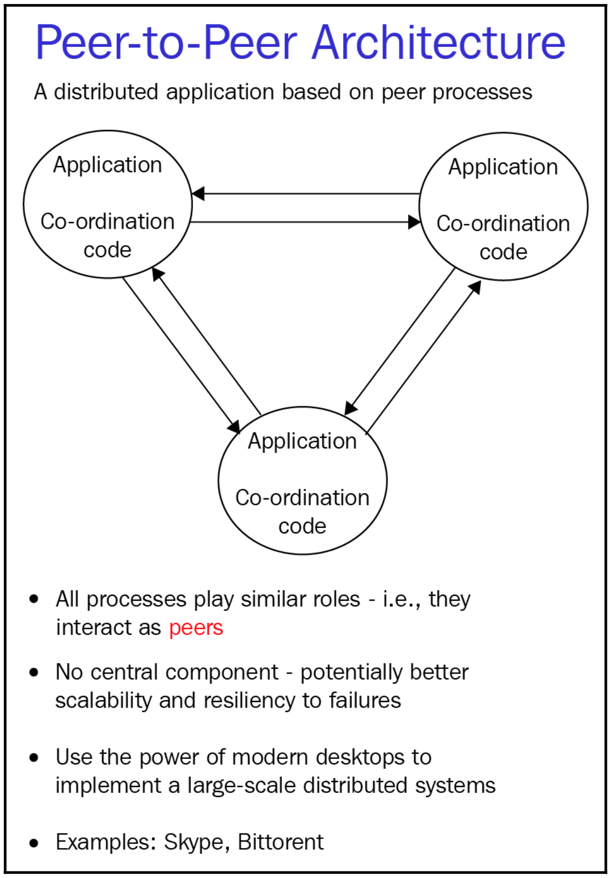
As discussed in prior chapters, hashing and the Merkle root of all transactions and behaviors on the blockchain allow for an easy calculation of consistency. If consistency is broken on a blockchain, it will be noticed instantly. Blockchains are designed to never be inconsistent. However, just because data is consistent does not mean it is accurate. These issues will be discussed in `Chapter 21`, *Scalability and Other Challenges*.

Peer-to-peer systems

Most computer systems in use today are client–server. A good example is your web browser and typical web applications. You load up Google Chrome or another browser, go to a website, and your computer (the client) connects to the server. All communication on the system is between you and the server. Any other connections (such as chatting with a friend on Facebook) happen with your client connected to the server and the server connected to another client with the server acting as the go-between.

Peer-to-peer systems are about cutting out the server. In a peer-to-peer system, your computer and your friend's computer would connect directly, with no server in between them.

The following is a diagram that illustrates the peer-to-peer architecture:



Peer-to-Peer Architecture

A distributed application based on peer processes

- All processes play similar roles - i.e., they interact as peers
- No central component - potentially better scalability and resiliency to failures
- Use the power of modern desktops to implement a large-scale distributed systems
- Examples: Skype, Bittorent

Decentralized systems

All decentralized systems must be distributed. But distributed systems are not necessarily decentralized. This is confusing to many people. If a distributed system is one spread across many computers, locations, and so on, how could it be centralized?

The difference has to do with location and redundancy versus control. Centralization in this context has to do with control. A good example to showcase the difference between distributed and decentralized systems is Facebook. Facebook is a highly distributed application. It has servers worldwide, running thousands of variations on its software for testing. Any of its data centers could experience failure and most of the site functionality would continue. Its systems are distributed with fault tolerance, extensive coordination, redundancy, and so on.

Yet, those services are still centralized because, with no input from other stakeholders, Facebook can change the rules. Millions of small businesses use and depend on Facebook for advertising. Groups that have migrated to Facebook could suddenly find their old messages, work, and ability to connect revoked—with no recourse. Facebook has become a platform others depend on but with no reciprocal agreement of dependability. This is a terrible situation for all those groups, businesses, and organizations that depend on the Facebook platform in part or on the whole.

The last decade has brought to the forefront a large number of highly distributed yet highly centralized platform companies —Facebook, Alphabet, AirBnB, Uber, and others—that provide a marketplace between peers but are also almost completely unbeholden to their users. Because of this situation, there is a growing desire for decentralized applications and services. In a decentralized system, there is no central overwhelming stakeholder with the ability to make and enforce rules without the permission of other network users.

Principles of decentralized systems

Like distributed systems, decentralization is a sliding scale more than an absolute state of being. To judge how decentralized a system is, there are a number of factors to consider. We're going to look at factors that have particular relevance to blockchain and decentralized applications and organizations. They are the following:

- Open access
- Non-hierarchy
- Diversity
- Transparency of operation

Open access

By definition, any system that is practically or logically closed will be at least somewhat centralized. A system that is closed is automatically centralized to the pre-existing actors. As with all other aspects of decentralized systems, this is not a binary yes/no but more of a sliding scale of possibility.

The early internet was seen as revolutionary in part because of its open access nature and the ability for anyone (with a computer, time, and access) to get online and begin trading information. Similarly, blockchain technologies have so far been open for innovation and access.

Non-hierarchical

A hierarchical system is the one commonly found within companies and organizations. People at the top of a hierarchy have overwhelming power to direct resources and events. A hierarchy comes in different extremes. At one extreme, you could have a system wherein a single arbiter holds absolute power. At the other extreme, you could have a system where each member of the system holds identical direct power and therefore control exists through influence, reputation, or some other form of organizational currency.

In blockchain, a few forms of non-hierarchical patterns have emerged. The first is in proof-of-work mining systems. All miners are fundamentally equal participants in the blockchain, but their influence is proportional to the computing resources they make available to the network.

In PoS blockchain systems, the power is distributed based on the level of investment/stake in the protocol of a specific. In this case, decentralization is achieved both through mass adoption as well as competition with other chains. If one chain becomes too centralized, nothing stops users from migrating to a different one.

How decentralized these systems will remain over time is an open question.

Ecosystem diversity

Open access naturally leads to another trait of decentralized systems: diversity. A diverse system stands in opposition to monoculture. In technology, a monoculture is the overwhelming prevalence of a single system, such as the dominance of Windows, which persisted for a long time in corporate America.

Transparency

One of the ways power can be centralized in a system is through information dominance, where one set of actors in a system has access to more or greater information than other actors. In most current blockchain technology, each participant on the chain gets the same amount of information. There are some exceptions. Hyperledger Fabric, for instance, has the capacity to have information hiding from participants.

The ability to have perfectly enforced transparency is one of the drivers of interest in blockchain systems. By creating transparent and unforgettable records, blockchain has an obvious utility for logistics and legal record keeping. With records on a blockchain, it is possible to know for certain that data was not altered. A transparent blockchain also ensures a level of fairness—participants can all be sure that at a minimum there is a shared level of truth available to all which will not change.

Downsides

Decentralized systems are not without their downsides. Here are a few key issues with decentralized systems that have specific relevance to blockchain:

- Speed
- Censorship resistance
- Chaos/non-determinism

Speed

Centralized systems and decentralized systems tend to be faster or slower at dealing with certain types of events. Blockchains are decentralized systems of record keeping. One way to think about a basic blockchain such as bitcoin is that it is an append-only database. Bitcoin can handle approximately seven transactions a second. By comparison, Visa and MasterCard are distributed (but not decentralized) transaction-handling systems that can handle more than 40,000 transactions a second. Blockchain systems continue to increase in speed but typically at with the trade-off of some amount of centralization or restrictions on access. Some PoS systems such as Tendermint or Waves have a theoretical throughput of over 1,000 tx/second but are still far from the peak capacity of their traditional counterparts.

Censorship resistance

Decentralized systems tend to be much harder to censor because of a lack of a central authority to do the censoring. For free-speech and free-information purists, this is not seen as a downside in the slightest. However, some information (child pornography, hate speech, bomb-making instructions) is seen as dangerous or immoral for public dissemination and therefore should be censored. As a technology, anything actually written into the blockchain is immutable once the block holding that information is finished. For instance, Steemit is a blockchain-based social blogging platform where each post is saved to the chain. Once each block is finalized, the data cannot be removed. Clients of the system could choose not to show information, but the information would still be there for those who wanted to look.

The desire for censorship extends to self-censorship. Content written to the change is immutable—even for its author. For instance, financial transactions done via bitcoin can never be hidden from authorities. While bitcoin is anonymous, once a person is attached to a bitcoin wallet, it is possible to easily track every transaction ever done since the beginning of the blockchain.

Because of this, a blockchain-based national currency would allow perfect taxation—due to perfect financial surveillance of the chain. Censorship resistance is thus a double-edged sword.

Chaos and non-determinism

Decentralized systems tend to be much more chaotic than centralized ones by their nature. In a decentralized system, each actor works according to their own desires and not the demands of an overarching authority. Because of this, decentralized systems are difficult to predict.

Summary

In this chapter, we have discussed the difference between distributed systems and decentralized systems and gone over some of the key features. You should now understand how each decentralized system is also a distributed system and some of the key aspects of each concept.

In the next chapter, we will start looking at how these things work in practice.

4 Cryptography and Mechanics Behind Blockchain

The use of blockchain hinges on cryptography. Numeric cryptography can be regarded as a recent invention, with the ciphers of the past relying on exchanging words for words and letters for letters. As we'll see, modern cryptography is a very powerful tool for securing communications, and, importantly for our topic, determining the provenance of digital signatures and the authenticity of digital assets.

In this chapter, the following topics will be covered:

- Principles of security
- Historical perspective – classical cryptography
- Cryptographic signatures
- Hashing

Principles of security

Cryptography safeguards the three principles of information security, which can be remembered by the mnemonic device **Central Intelligence Agency** (**CIA**):

- **Confidentiality**: Ensures that information is shared with the appropriate parties and that sensitive information (for example, medical information, some financial data) is shared exclusively with the consent of appropriate parties.

- **Integrity**: Ensures that only authorized parties can change data and (depending on the application) that the changes made do not threaten the accuracy or authenticity of the data. This principle is arguably the most relevant to blockchains in general, and especially the public blockchains.

- **Availability**: Ensures authorized users (for example, holders of tokens) have the use of data or resources when they need or want them. The distributed and decentralized nature of blockchain helps with this greatly.

The relevance to blockchain and cryptocurrency is immediately evident: if, for instance, a blockchain did not provide integrity, there would be no certainty as to whether a user had the funds or tokens they were attempting to spend. For the typical application of blockchain, in which the chain may hold the chain of title to real estate or securities, data integrity is very important indeed. In this chapter, we will discuss the relevance of these principles to blockchain and how things such as integrity are assured by cryptography.

Historical perspective – classical cryptography

Cryptography is the term for any method or technique used to secure information or communication, and specifically for the study of methods and protocols for secure communication. In the past, cryptography was used in reference to encryption, a term that refers to techniques used to encode information.

At its most basic, encryption might take the form of a substitution cipher, in which the letters or words in a message are substituted for others, based on a code shared in advance between the parties. The classic example is that of the Caesar Cipher, in which individual letters are indexed to their place in the alphabet and shifted forward a given number of characters. For example, the letter *A* might become the letter *N*, with a key of 13.

This specific form of the Caesar Cipher is known as **ROT13**, and it's likely the only substitution cipher that continues to see any regular use—it provides a user with a trivially reversible way of hiding expletives or the solutions to puzzles on static websites (the same, of course, could be implemented very simply in JavaScript).

This very simple example introduces two important concepts. The first is an algorithm, which is a formal description of a specific computation with predictable, deterministic results. Take each character in the message and shift it forward by *n* positions in the alphabet. The second is a key: the *n* in that algorithm is 13. The key in this instance is a pre-shared secret, a code that the two (or more) parties have agreed to, but, as we'll see, that is not the only kind of key.

Types of cryptography

Cryptography is principally divided into symmetric and asymmetric encryption. Symmetric encryption refers to encryption in which the key is either pre-shared or negotiated. AES, DES, and Blowfish are examples of algorithms used in symmetric encryption.

Symmetric cryptography

Most savvy computer users are familiar with WEP, WPA, or WPA2, which are security protocols employed in Wi-Fi connections. These protocols exist to prevent the interception and manipulation of data transmitted over wireless connections (or, phrased differently, to provide confidentiality and integrity to wireless users). Routers now often come with the wireless password printed on them, and this is a very literal example of a pre-shared key.

The algorithms used in symmetric encryption are often very fast, and the amount of computational power needed to generate a new key (or encrypt/decrypt data with it) is relatively limited in comparison to asymmetric encryption.

Asymmetric (public-key) cryptography

Asymmetric cryptography (also called public-key cryptography) employs two keys: a public key, which can be shared widely, and a private key, which remains secret. The public key is used to encrypt data for transmission to the holder of the private key. The private key is then used for decryption.

The development of public-key cryptography enabled things such as e-commerce internet banking to grow and supplement very large segments of the economy. It allowed email to have some level of confidentiality, and it made financial statements available via web portals. It also made electronic transmissions of tax returns possible, and it made it possible for us to share our most intimate secrets in confidence with, maybe, perfect strangers—you might say that it brought the whole world closer together.

As the public key does not need to be held in confidence, it allows for things such as certificate authorities and PGP key servers—publishes the key used for encryption, and only the holder of the private key will be able to decrypt data encrypted with that published key. A user could even publish the encrypted text, and that approach would enjoy some anonymity—putting the encrypted text in a newsgroup, an email mailing list, or a group on social media would cause it to be received by numerous people, with any eavesdropper unable to determine the intended recipient. This approach would also be interesting in the blockchain world—thousands or millions of nodes mirroring a cipher text without a known recipient, perhaps forever, irrevocably, and with absolute deniability on the part of the recipient.

Public-key cryptography is more computationally expensive than symmetric cryptography, partly due to the enormous key sizes in use. The NSA currently requires a key size of 3,072 bits or greater in commercial applications for key establishment, which is the principal use of public-key cryptography. By comparison, 128-bit encryption is typically regarded as adequate for most applications of cryptography, with 256-bit being the NSA standard for confidentiality.

For the most part, although it is possible to use the public-key algorithm alone, the most common use of public-key cryptography is to negotiate a symmetric key for the remainder of the session. The symmetric key in most implementations is not transmitted, and, as a consequence, if an attacker were to seize one or both of the private keys, they would be unable to access the actual communications. This property is known as forward secrecy.

Some protocols, such as SSH, which is used to remotely access computers, are very aggressive. Over the course of a session, SSH will change the key at regular intervals. SSH also illustrates the essential property of public-key cryptography—it's possible to put your public key on the remote server for authentication, without any inherent confidentiality issue.

Most cryptography in use today is not unbreakable, given extremely large (or infinite) computing resources. However, an algorithm suited to the task of protecting data where confidentiality is required is said to be computationally improbable—that is, computing resources to crack the encryption do not exist, and are not expected to exist in the near future.

Signatures

It is notable that, although when encrypting data to send it to a given recipient, the private key is used for decryption, it is generally possible to do the reverse. For cryptographic signing, private keys are used to generate a signature that can be decrypted (verified) with the public key published for a given user. This inverted use of public-key cryptography allows for users to publish a message in the clear with a high degree of certainty that the signer is the one who wrote it. This again invokes the concept of integrity—if signed by the user's private key, a message (or transaction) can be assumed to be authentic. Typically, where Blockchains are concerned, when a user wishes to transfer tokens, they sign the transaction with the private key of the wallet. The user then broadcasts that transaction.

It is now also fairly common to have multisignature wallets, and, in that instance, a transaction is most often signed by multiple users and then broadcast, either in the web interface of a hosted wallet service, or in a local client. This is a fairly common use case with software projects with distributed teams.

Hashing

Distinct from the concept of encryption (and present in many mechanisms used in cryptography, such as cryptographic signatures and authentication) is hashing, which refers to a deterministic algorithm used to map data to a fixed-size string. Aside from determinism, cryptographic hashing algorithms must exhibit several other characteristics, which will be covered in this section.

As we'll see in the following section, a hash function must be difficult to reverse. Most readers who got through high school algebra will remember being tormented with factoring. Multiplication is an operation that is easy to complete, but difficult to reverse—it takes substantially more effort to find the common factors of a large number as opposed to creating that number as a product of multiplication. This simple example actually enjoys practical application. Suitably large numbers that are the product of the multiplication of two prime numbers—called **semiprimes** or (less often) **biprimes**—are employed in RSA, a widely used public-key cryptography algorithm.

RSA is the gold standard in public key cryptography, enabling things such as SSH, SSL, and systems for encrypting email such as PGP. Building on operations such as this — easy to do one way and very hard to do in the other—is what makes cryptography so robust.

The avalanche effect

A desirable feature of robust hashing algorithms is known as **the avalanche effect**. A small change in the input should result in a dramatic change in the output. Take for instance the following three examples using output redirection and the GNU `md5sum` utility present in most distributions of Linux:

```
$ echo "Hills Like White Elephants by Ernest Hemingway" | md5sum
86db7865e5b6b8be7557c5f1c3391d7a -
$ echo "Bills Like White Elephants by Ernest Hemingway" | md5sum
ccba501e321315c265fe2fa9ed00495c -
$ echo "Bills Like White Buffalo by Ernest Hemingway"| md5sum
37b7556b27b12b55303743bf8ba3c612 -
```

Changing a word to an entirely different word has the same result as changing a single letter: each hash is entirely different. This is a very desirable property in the case of, say, password hashing. A malicious hacker cannot get it close enough and then try permutations of that similar password. We will see in the following sections, however, that hashes are not perfect.

Collisions

An ideal hash function is free of **collisions**. Collisions are instances in which two inputs result in the same output. Collisions weaken a hashing algorithm, as it is possible to get the expected result with the wrong input. As hashing algorithms are used in the digital signatures of root certificates, password storage, and blockchain signing, a hash function having many collisions could allow a malicious hacker to retrieve passwords from password hashes that could be used to access other accounts. A weak hashing algorithm, rife with collisions, could aid in a man-in-the-middle attack, allowing an attacker to spoof a **Secure Sockets Layer** (**SSL**) certificate perfectly.

MD5, the algorithm used in the above example, is regarded as inadequate for cryptographic hashing. Blockchains thankfully largely use more secure hash functions, such as SHA-256 and RIPEMD-160.

Hashing a block

In the PoW systems, new entries to a blockchain require hashes to be computed. In Bitcoin, miners must compute two SHA-256 hashes on the current transactions in the block—and included therein is the hash of the previous block.

This is pretty straightforward for a hashing algorithm. Let's briefly reiterate: an ideal hash function takes the expected input and then outputs a unique hash. It is deterministic. There is only one possible output and it is impossible (or computationally improbable) to achieve that output with a different input. These properties ensure that miners can process a block and that each miner can return the same result. It is through hashing that Blockchains attain two properties that are crucial to their adoption and current popularity: decentralization and immutability.

Linking the current block to the previous block and the subsequent block is in part what makes the blockchain an ever-growing linked list of transactions (providing it with the property of immutability), and the deterministic nature of the hash algorithm makes it possible for each node to get the same result without issue (providing it with decentralization).

Hashing outside PoW

Aside from proof of work, PoS and DPoS also make use of hashes, and largely for the same purpose. Plenty of discussion has been dedicated to whether PoS will replace PoW and prevent us from running thousands of computers doing megawatts' worth of tedious hashing with enormous carbon footprints.

PoW systems seem to persist in spite of the power consumption and environmental impact of reasonably difficult hashing operations. Arguably, the reason for this is the very simple economics: miners have an incentive to validate transactions by computing hashes because they receive a share of new tokens minted into the system. More complex tokenomics schemes for proof of stake or distributed proof of stake often fail the smell test.

Take, for example, the idea of a stock photo blockchain project—we'll call it Cannistercoin. Users contribute photos to a stock photo website, and in return they receive tokens. The token is also used to buy stock photos from the website, and this token is traded on exchanges.

This would seem to work, and it's a complete market—Cannistercoin has identified buyers and sellers and has a mechanism to match them—but it is perhaps not a functional market. The barriers to entry here are significant: a buyer could use any ordinary stock photo site and use their credit card or bank account. In this model, the buyer needs to sign up for an exchange and send cryptocurrency in exchange for the token.

To be truly decentralized, a big piece is missing from this economic model—that is, this system of incentives. What provides an incentive for witnesses or validators to run their machines and validate transactions?

You can give them some share of the tokens, but why wouldn’t many of them sell their tokens immediately in order to recoup the costs of running their machines? It can be reasonably expected that that constant selling pressure holds down the price of tokens in many of these appcoin cryptocurrencies, and this is a shame. Proof of stake systems are often more elegant with regard to processing power (at the expense of less elegant economic models).

Proof of stake (or another mechanism) may well still take over the world, but, one way or the other, you can safely expect the crypto world to do plenty of hashing.

Summary

The world of blockchain and cryptocurrency exists thanks largely to the innovations of the last century in cryptography. We've covered how cryptography works conceptually and how cryptographic operations, specifically hashing, form a large part of what happens behind the scenes in a blockchain.

In the next chapter, we'll build on this foundation and introduce Bitcoin, the first (and most notable) blockchain application.

5 Bitcoin

In earlier chapters, we discussed blockchain, its components, and its structure in detail. We also discussed cryptography, the mechanics behind blockchain, and how blockchain is revolutionizing the network world. In this chapter, we will be discussing Bitcoin's origins.

We will discuss the introduction of Bitcoin, its history, and how it became one of the biggest revolutions of financial history in such a short space of time. We will also dive deep into other aspects of Bitcoin, such as its encoding system, transaction process, network nodes, and we'll briefly cover the mining of Bitcoins.

The topics that we will cover in this chapter include the following:

- The history of Bitcoin
- Why Bitcoin is volatile
- Keys and addresses
- Transactions
- Blocks
- Bitcoin network
- Wallets

The history of Bitcoin

Bitcoin is the first and, to date, the most successful application of blockchain technology. Bitcoins were introduced in 2008, in a paper on Bitcoin called **Bitcoin: A Peer-to-Peer Electronic Cash System** (`https://bitcoin.org/bitcoin.pdf`), which was authored by Satoshi Nakamoto.

Bitcoin was the world's first decentralized cryptocurrency; its introduction heralded a revolution, and, in just about a decade, it has proved its strengths, with huge community backing and widespread adoption.

From 2010, certain global businesses have started to accept Bitcoins, with the exception of fiat currencies. A lot of currency exchanges were founded to let people exchange Bitcoin with fiat currency or with other cryptocurrencies. In September 2012, the Bitcoin Foundation was launched to accelerate the global growth of Bitcoin through standardization, protection, and promotion of the open source protocol.

A lot of payment gateways such as BitPay came up to facilitate merchants in accepting Bitcoin as a payment method. The popular service WordPress started accepting Bitcoins in November 2012.

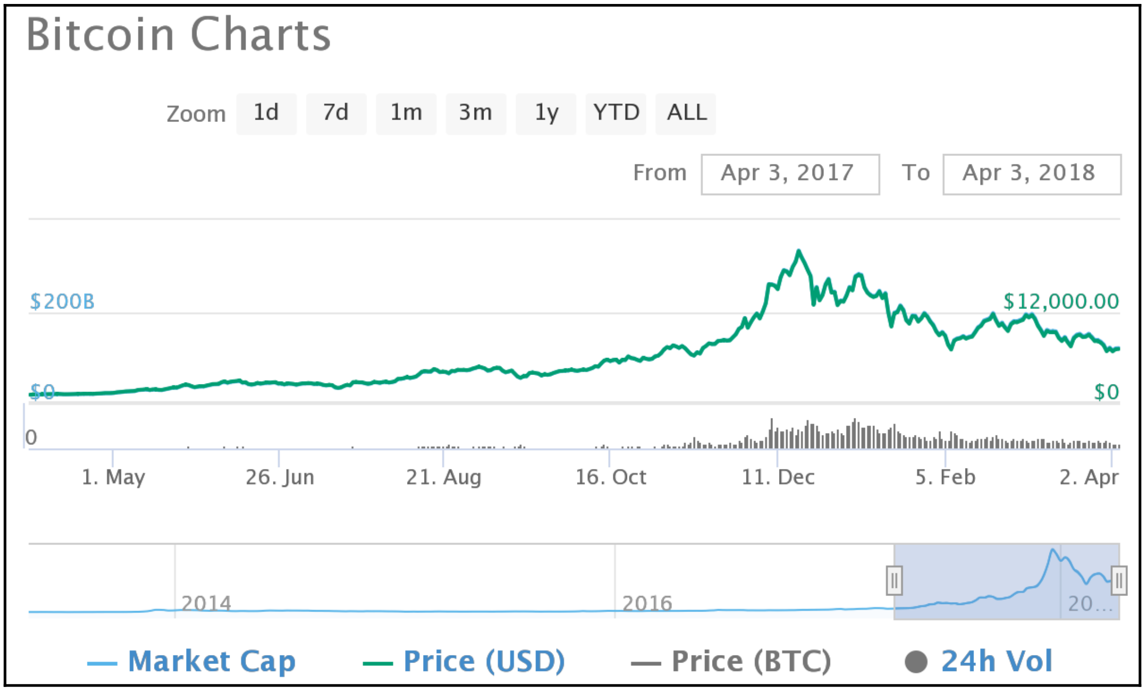
Bitcoin has been growing as a preferred payment option in global payments, especially business-to-business supply chain payments. In 2017, Bitcoin gained more legitimacy among financial companies and government organizations. For example, Russia legalized the use of cryptocurrencies, including Bitcoin, Norway's largest bank announced the inception of a Bitcoin account, and Japan passed a law to accept Bitcoin as a legal payment method. The world's largest free economic zone, Dubai, has started issuing licenses to firms for trading cryptocurrencies.

On August 1, 2017, Bitcoin split into two derivative digital currencies; one kept the legacy name Bitcoin, and the other with an 8 MB block size is known as **Bitcoin Cash** (**BCH**). After this, another hard fork happened on October 24, 2017, with a new currency known as **Bitcoin Gold** (**BTG**). Then, again, on February 28, 2018, another hard fork happened, with the new currency known as **Bitcoin Private** (**BTCP**). There was another hard fork due in November 2017, but this was canceled due to lack of consensus from the community.

However, there is a single major concern of the promoters of Bitcoin, with regards to price volatility and slowing of transaction due to a large number of confirmations required to approve a transaction.

Why Bitcoin is volatile

When we say Bitcoin is volatile, we mean the price of Bitcoin is volatile. The spot rate of Bitcoin at various exchanges changes every moment and, moreover, it functions 24/7. Hence, any user or community member of Bitcoin is perturbed by the regularly changing price of Bitcoin. The following chart shows the price fluctuation of Bitcoin over the last financial year:



The volatility of Bitcoin is the most discussed topic and has been a concern for investors, miners, and supporters of Bitcoin since the exchanges of Bitcoin came up. Some prime reasons for this as follows:

- **Security breaches**: This has been a major issue in the price fluctuation of Bitcoin in the past; whenever news of security flaws at an exchange's end comes up, the price of Bitcoin takes a hit, since this makes the investors start to doubt a certain exchange or the Bitcoin network.
- **Legal issues**: There have been many doubts by global lawmakers who make predictions on Bitcoin's price or even try to label Bitcoin as a non-legal entity. Various statements by the governments also hamper the price.
- **Psychological shift in adaption**: Bitcoin, although getting adopted and supported very quickly, is still a new currency and is very different from anything that came before; this leads people to be wary of adopting it. Plus, any negative press directly impacts the adoption of Bitcoin, which sometimes creates a downward spiral in price until any positive news surrounding Bitcoin surfaces.

The preceding points are just some of the prime points that are causing a huge volatility in the Bitcoin market. There are various other factors that play a vital role in the price fixtures of Bitcoin from time to time.

Keys and addresses

Bitcoin, being a purely digital currency, can be owned by people by keeping or storing it in files or in a **Bitcoin Wallet**. Addresses are used to transfer Bitcoins from one wallet to another, and keys are used to secure a transaction.

Keys in Bitcoins are used in pairs. One is a public key, and the other is a private key. The private key is to be kept securely, since it gives you control over a wallet. The keys are stored and controlled by a Bitcoin wallet.

Addresses are alphanumeric strings that are shared for sending or receiving Bitcoins from one wallet to another. The addresses are mostly encoded as **Base58Check**, which uses a Base58 number for address transcription. A Bitcoin address is also encoded in a QR code for quick transactions and sharing.

Currency units

Bitcoin has a widely used metric system of denominations that are used as units of Bitcoins. The smallest denomination of Bitcoin is called a **Satoshi**, after the name of its creator. The following table shows the units of Bitcoin, from its smallest unit, **Satoshi**, to **Megabit**:

DENOMINATION	ABBREVIATION	Common NAME	VALUE IN BTC
Satoshi	SAT	Satoshi	0.00000001 BTC
Microbit	µBTC (uBTC)	Microbitcoin or Bit	0.000001 BTC
Millibit	mBTC	Millibitcoin	0.001 BTC
Centibit	cBTC	Centibitcoin	0.01 BTC
Decibit	dBTC	Decibitcoin	0.1 BTC
Bitcoin	BTC	Bitcoin	1 BTC
DecaBit	daBTC	Decabitcoin	10 BTC
Hectobit	hBTC	Hectobitcoin	100 BTC
Kilobit	kBTC	Kilobitcoin	1000 BTC
Megabit	MBTC	Megabitcoin	1000000 BTC

Vanity addresses

These are valid addresses that contain readable addresses. For example: `1BingoAuXyuSSoYm6rH7XFZc6Hcy98zRZz` is a valid address that contains a readable word (Bingo). Generating a vanity address needs creation and testing of millions of private keys, until the desired Base58 letter address is found.

The vanity addresses are used for fun and offer the same level of security as any other address. The search time for a vanity address increases as the desired pattern's length increases.

Base58 check encoding

This encoding takes the binary byte arrays and converts them into a human-readable format. This string is created by using a set of 58 alphanumeric characters.

Instead of Base58, Base64 could also be used, but that would have made some characters look identical, which could have resulted in identical-looking data. The Base58 symbol chart used in Bitcoin is specific to Bitcoins and was used only by Bitcoins at the time of creation. The following table shows the value and the character corresponding to it in the Base58 encoding:

Value	**Character**	**Value**	**Character**	**Value**	**Character**	**Value**	**Character**
0	1	1	2	2	3	3	4
4	5	5	6	6	7	7	8
8	9	9	A	10	B	11	C
12	D	13	E	14	F	15	G
16	H	17	J	18	K	19	L
20	M	21	N	22	P	23	Q
24	R	25	S	26	T	27	U
28	V	29	W	30	X	31	Y
32	Z	33	a	34	b	35	c
36	d	37	e	38	f	39	g
40	h	41	i	42	j	43	k
44	m	45	n	46	o	47	p
48	q	49	r	50	s	51	t
52	u	53	v	54	w	55	x
56	y	57	z	-	-	-	-

Transactions

This is the primary part of the Bitcoin system. Transactions are not encrypted, since Bitcoin is an open ledger. Any transaction can be publicly seen in the blockchain using any online blockchain explorer. Since addresses are encrypted and encouraged to be unique for every transaction, tracing a user becomes difficult.

Blocks in Bitcoin are made up of transactions that are viewed in a blockchain explorer; each block has the recent transactions that have happened. Every new block goes at the top of the blockchain. Each block has a height number, and the height of the next block is one greater than that of the previous block. The consensus process is commonly known as **confirmations** on the blockchain explorer.

Types

There are various types of scripts available to manage the value transfer from one wallet to another. Some of the standard types of transactions are discussed here for a clear understanding of address and how transactions differ from one another.

Pay-to-Public-Key Hash

The **Pay-to-Public-Key Hash** (**P2PKH**) majority of the transactions on the Bitcoin network happen using this method. This is how the script looks:

```
OP_DUP OP_HASH160 [Pubkey Hash] OP_EQUALVERIFY OP_CHECKSIG
```

This is how the signature script looks:

```
[Sig][PubKey]
```

These strings are concatenated together to be executed.

Pay-to-Script Hash

The **Pay-to-Script Hash** (**P2SH**) process is used to send transactions to a script hash. The addresses to pay using script hash have to start with 3. This is how the script looks:

```
OP_HASH160 [redeemScriptHash] OP_EQUAL
```

The signature looks like this:

```
[Sig]...[Sig][redeemScript]
```

As with P2PKH, these strings are also concatenated together to create the script signature.

Blocks

The transaction data is recorded in files, and these files are known as **blocks**. The blocks are stacked on top of one another, the most recent block being at the top. The following table depicts the structure of the block and the size of the elements in a block:

Size	Field	Details
4 bytes	Magic no.	Value is always 0xD9B4BEF9
4 bytes	Blocksize	number of bytes following up to end of block
80 bytes	Blockheader	consists of 6 items
1 - 9 bytes	Transaction counter	positive integer VI = VarInt
-	transactions	list of transactions

Every block in the Bitcoin network has almost the same structure, and each of the blocks is chained to the most recent block. These are the fields of the block:

- **Magic number**: This number is an identifier for the blockchain network. Its value is always constant at **0xD9B4BEF9**. It confirms the start of the block and verifies that the data is from the production network.
- **Block size**: This signifies the size of the block.
- **Block header**: A header contains the metadata of the block. It comprises multiple items of data, such as the Bitcoin version, the previous block hash, Merkle root, timestamp, mining difficulty, and nonce.
- **Transaction counter**: It is the count of the transactions in the block.
- **Transaction list**: It stores the hash of the transactions in that block.

Genesis block

The **genesis block** is the first block in the blockchain of Bitcoin. The creation of the genesis block marked the start of Bitcoin. It is the common ancestor of all the blocks in the blockchain. It is statically encoded within the Bitcoin client software and cannot be altered. Every node in the blockchain of Bitcoin acknowledges the genesis block's hash and structure, the time of its creation, and the single transaction it contains. Following is the static code written in the Bitcoin source code, which describes the creation of the genesis block with static parameters `pszTimestamp`, `genesisOutputScript`, `nTime`, `nNonce`, `nBits`, and `nVersion`. Here is the snippet of this code in the Bitcoin repository:

```
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce,
uint32_t nBits, int32_t nVersion, const CAmount& genesisReward)
 {
 const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on
brink of second bailout for banks";
 const CScript genesisOutputScript = CScript() <<
ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0e
a1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6b
f11d5f") << OP_CHECKSIG;
 return CreateGenesisBlock(pszTimestamp, genesisOutputScript,
nTime, nNonce, nBits, nVersion, genesisReward);
 }
```

Bitcoin network

The network is based on a **peer-to-peer** (**P2P**) protocol. Various nodes exchange transactions and blocks in this network. Every node in this Bitcoin network is treated equally. One advantage of this is that each node has the option of taking different roles, depending on each person's preference on how they want to participate in the Bitcoin network.

Types of nodes

Before we discuss the types of nodes, let's discuss some of the primary functionalities that the nodes perform:

- Wallet
- Mining
- Full blockchain
- Routing

Majorly, there are two types of nodes in the Bitcoin network. We'll now go into some brief details on each.

Full node

A full node is made up of the wallet, miner, complete blockchain, and the routing network. These nodes maintain a complete up-to-date record of the blockchain. The full nodes verify every transaction on the blockchain network.

Lightweight nodes

Lightweight nodes perform transactions on the blockchain. They do not contain the entire blockchain, instead just a subset of the blockchain. They verify the transactions using a system called **Simplified Payment Verification** (**SPV**). These nodes are also sometimes called **SPV nodes**.

Other nodes

There are various other nodes on the Bitcoin network, each of them performing a specific set of functionalities from the offered functionalities in the Bitcoin network. Some nodes contain only the blockchain and routing functionalities. Some nodes only work as miners and do not contain the wallet.

There are other nonstandard nodes called **pool protocol servers**. These Nodes work on alternative protocols such as the **stratum protocol**. The stratum protocol works on TCP sockets and JSON-RPC to communicate among the nodes.

Network discovery

The **network discovery** in Bitcoin is required by any node when it is first started; a node has to discover other nodes in the network to participate in the blockchain. At the start of a node, it has to connect with at least one existing node in the network.

For this, the nodes establish a connection over the TCP protocol, over port `8333` or any other port if there is one. Next a handshake is performed by transmitting a certain message. That message is called the **version message**, which contain basic identification information.

Finding peers

Peers are found in the network primarily by two methods. One is by querying DNS using **DNS seeds**, which are basically DNS servers that provide a list of the IPs of Bitcoin nodes. The other method is a list of IPs that Bitcoin core tries to connect to. Another method, which was used earlier, was seeding nodes through IRC, but that method was discontinued due to security concerns.

DNS seeds

DNS seeds are servers which contains lists of IP addresses. These seeds are custom implementations of **Berkeley Internet Name Daemon** (**BIND**) and return random subsets collected by a Bitcoin node. Most of the Bitcoin clients use DNS seeds to connect while trying to establish to first set of connection. It is better to have various seeds present so that a better connection can be established by the client with the peers present over the network. In the Bitcoin core client, the option to use DNS seeds is controlled by the `-dnsseed` parameter, which is set to `1` by default. Here is how the DNS seeds are represented in the `chainparams.cpp` file of the Bitcoin source:

```
 vSeeds.push_back(CDNSSeedData("bitcoin.sipa.be",
"seed.bitcoin.sipa.be")); // Pieter Wuille
 vSeeds.push_back(CDNSSeedData("bluematt.me",
"dnsseed.bluematt.me")); // Matt Corallo
 vSeeds.push_back(CDNSSeedData("dashjr.org",
"dnsseed.bitcoin.dashjr.org")); // Luke Dashjr
 vSeeds.push_back(CDNSSeedData("bitcoinstats.com",
"seed.bitcoinstats.com")); // Christian Decker
 vSeeds.push_back(CDNSSeedData("xf2.org", "bitseed.xf2.org")); //
Jeff Garzik
 vSeeds.push_back(CDNSSeedData("bitcoin.jonasschnelli.ch",
"seed.bitcoin.jonasschnelli.ch")); // Jonas Schnelli
```

The preceding seeds are currently being used in Bitcoin core, for connecting with the seed client for establishing the connection with the first node.

Static IPs

These are static lists of IP addresses. If the Bitcoin client is successfully able to connect with one IP address, it will be able to connect to other nodes by sharing the node's IPs. The command-line argument `-seednode` is used to connect to one node. After initial connection to the seed node, the client will discover new seeds using that seed itself.

Wallets

Bitcoin wallets are an important function of the Bitcoin node; they contain private and/or public keys and Bitcoin addresses. There are various types of Bitcoin wallets and each one offers a varied level of security and functions, as required.

There is a common misconception that an e-wallet can contain Bitcoins but a Bitcoin wallet will only contain keys. Each Bitcoin is recorded on the blockchain in the Bitcoin network. A Bitcoin wallet contains keys, and these keys authorize the use of Bitcoins that are associated with the keys. Users or wallet owners sign a transaction with the keys in the wallet, proving that they own the Bitcoins. In reality, these Bitcoins are stored on the blockchain in the form of **transaction outputs** that are denoted as **`txout`**.

Types

Primarily, there are two types of wallets, which is based on whether the keys contained by the wallets are related to one another.

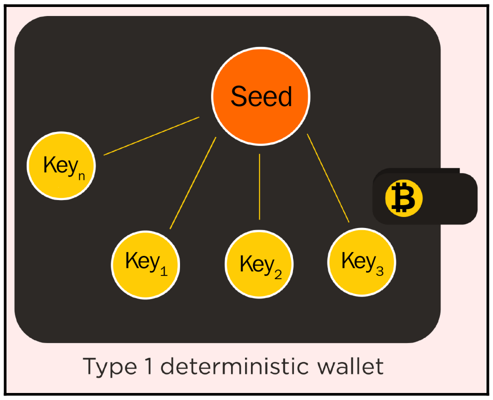
Deterministic wallet

This is a type of wallet in which all the keys are derived from a single master key, which is also known as a **seed**. All the keys in this type of wallet are linked with one another and can be easily generated again with the help of a seed. In some cases, a seed allows the creation of public key addresses without the knowledge of private keys. Mostly, seeds are serialized into human-readable words known as a mnemonic phrase.

There are multiple key derivation methods used in deterministic wallets, which are described in the following subsections.

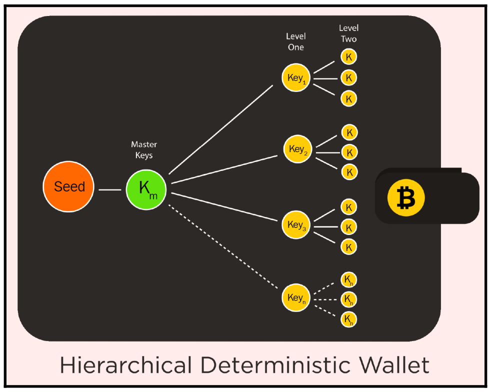
Deterministic wallets

Deterministic wallets hold private keys that are derived from a common seed. A one-way hash function is used for this. In a deterministic wallet, this seed is essential to recover all the derived keys, and hence a single backup at the time of creation is sufficient. The following diagram depicts how a single seed is connected/related to all the keys generated by the wallet:



HD wallets

HD wallets are one of the most advanced form of deterministic wallets. They contain keys derived from a tree structure, such that the master key can have multiple level-1 keys that can further contain multiple level-2 keys of up to an infinite depth. The following diagram depicts how a seed generates master keys that further create multiple keys in a hierarchical formation:



Non-deterministic wallet

In this type of wallet, each key is independently generated from a random number. The keys generated in this wallet are not related to one another. Due to the difficulty in maintaining multiple non-related keys, it is very important to create regular backups of these keys and protect them to prevent theft or loss.

Summary

In this chapter, we discussed the basics of Bitcoin, its history, and its pricing in comparison to fiat currency. We also discussed Bitcoin addresses, their encoding, their transaction types, and blocks. Finally, we discussed the Bitcoin network and the types of nodes the network contains.

Now that we have discussed the world's first cryptocurrency in this chapter, in the next chapter, we will discuss various other cryptocurrencies that were inspired by Bitcoin and are also known as **alternative currencies**. *W*e will discuss the alt-currencies, which are sometimes also called **Altcoins**.

6 Altcoins

After the release of Bitcoin, there has always been a large community supporting cryptocurrencies. Also, since Bitcoin and blockchain are open source, a lot of people within the community started to create their cryptocurrencies offering similar services that provide different consensus methods and so on, in comparison to Bitcoin.

Since Bitcoin is the first and, by far, most dominant cryptocurrency, all other cryptocurrencies are known as **Alternative Coins** or **Altcoins**. The first Altcoin was **Namecoin**, introduced in 2011. After that, a lot of Altcoins have been introduced; a few of them were popular and also started to be used as mainstream coins, whereas others were less popular. As of now, there are more than 1,500 cryptocurrencies in existence. There are two broad categories in the alternative cryptocurrency sphere. If an alternative blockchain platform is to be created, they are commonly called **Alternative Chains**, but if the purpose of the coin is to introduce a new currency, then it can be known as an **Altcoin**.

Many of the Altcoins are forked directly from Bitcoin's source, and some are even written from scratch. Altcoins are looking to solve some limitation or other of Bitcoins such as the consensus method, mining limitations, block time, distribution, privacy, or sometimes even adding a feature on top of the coin.

The topics that we will be covering in this chapter are as follows:

- Introducing Altcoins
- Discussing tokens and various token platforms
- Alternative currency and a brief introduction to widely used alt currencies
- How to acquire cryptocurrency

Introducing Altcoins

There are more than 1,600 cryptocurrencies in circulation; each one of them provides an improvement or a modification over Bitcoins. There are various online exchanges where cryptocurrencies can be exchanged amongst one another or for fiat currencies (such as USD, EUR, GBP, and so on) too, similar to a currency exchange or a stock-exchange portal. Here is a list of some popular Altcoins:

- Litecoins
- Ethereum
- Zcash
- Dash
- Ripple
- Monero
- EOS
- IOTA
- TRON
- Tether
- Ethereum Classic
- Bitcoin Cash
- Stellar
- NEO
- NEM
- Cardano

There are two broad categories of Altcoin on the basis of their blockchain, since the blockchain defines features, security, and other aspects of the coin.

It is certain that having so many cryptocurrencies in the market, along with currency exchanges, means that, now, a lot of key financial and market metrics are required to differentiate among the coins. The following are some of the key factors to consider when discussing an Altcoin:

- The total market of the coin
- The unity price of the coin when purchasing it using fiat currency
- The total number of the Altcoins in circulation
- The volume of coins that are transacted hourly as well as within a 24-hour cycle

- The change in the unit price of the coin on an hourly, a daily, and a weekly basis
- Exchanges and merchants accepting the Altcoin
- Total wallets supporting the coin
- Any recent news that could affect the price or reputation of the coin

Either an Altcoin can have its own blockchain or it can be built on top of another blockchain, usually known as **tokens**.

Tokens

Altcoins built on top of another blockchain are called tokens. Tokens cannot exist without another platform on which they are built. Here is list of platforms on top of which tokens can be created:

- Ethereum
- Omni
- NEO
- Omni
- Qtum
- Counterparty
- Waves
- Stellar
- Bitshares
- Ubiq
- Ardor
- NXT

Ethereum

Ethereum is the most widely used option to create tokens; there are more than 600 tokens based on the Ethereum platform that are available. Being public and open source makes it easy for anyone to create tokens based on the Ethereum blockchain; furthermore, being backed by a huge community makes it more secure and easy for any cryptocurrency-based exchange to accept the tokens.

Some of the popular tokens built on the Ethereum platform are EOS, TRON, VeChain, and so on.

Omni Layer

Starting out in the form of a MasterCoin in 2013, the **Omni Layer protocol** is one of the most popular meta-protocols based on the Bitcoin blockchain. The Omni Layer offers a different wallet similar to the Bitcoin client and an Omni Layer similar to the Bitcoin core.

Some of the popular tokens built on the Omni Layer platform are Tether, MaidSafeCoin, Synereo, and so on.

NEO

NEO started as Antshares; it was built by Onchain, a company based in China. It started out in early 2014, providing services to the banking and finance sectors. Antshares was rebranded as NEO in June 2017.

Some of the popular tokens built on the NEO platform are Ontology, Gas, DeepBrain Chain, and so on.

Waves

Often described as the open blockchain platform, **waves** is a platform where not just cryptocurrencies but all types of real-world commodities can also be exchanged, issued, and transferred in a completely decentralized way.

Some of the tokens built on the waves platform are Wager, Mercury, Incent, and so on.

Counterparty

Counterparty is another protocol layer implemented on top of Bitcoin Protocol Layer, just like Omni Layer and released in 2014, it claims to offer various features apart from Bitcoin, which makes it a valuable token creation platform.

Some of the tokens built on Counterparty are Triggers, Pepe Cash, Data bits, and so on.

Alternative currency

Apart from tokens, which are built on top of existing blockchains, there are various alt currencies fueled by either having their own blockchain and/or any other improvement or differentiating factor.

Here are some of the factors that have been modified on the Altcoins, when compared to Bitcoins:

- Monetary alternatives
- Consensus alternatives

We will discuss each of these alternatives in detail, and then afterward we will discuss some of the widely used Altcoins and what modifications they have over Bitcoins.

Monetary alternatives

Bitcoin is limited to 21 million coins, and it has a declining issuance rate, including a 10-minute block generation time, which takes lot of confirmation time; it also takes a lot of time in further coin generation. A lot of Altcoins have modified a number of primary parameters to achieve escalated results. Some of the prime coins that have modified monetary parameters are Litecoin, Dogecoin, Ethereum, NEO, and so on.

Consensus alternatives

A consensus mechanism is the root of transactions of Bitcoins; the mechanism used in Bitcoins is based on proof-of-work, using an SHA256 algorithm. Since the consensus mechanism requires the mining of blocks, it became very computation-intensive, leading to the creation of specified **Application-Specific Integrated Circuit Chips** (**ASICs**), which are Bitcoin-mining hardware created specifically to solve Bitcoin blocks.

This led to the creation of Altcoins with innovative algorithms such as these:

- **Scrypt**: This is widely used in Litecoins, Dogecoin, and so on
- **Scrypt - N**: This is used in vertcoin, Execoin, and so on
- **Skein**: This is used in DigiByte, Myraid, and so on
- **Groestl**: This is used in Groestlcoin, securecoin, and so on
- **SHA3**: This is used in Maxcoin, Slothcoin, and so on
- **X11**: This is used in Dash, CannabisCoin, and so on
- **Blake**: This is used in Netko, Trumpcoin, and so on
- **X13**: This is used in Navcoin, Cloakcoin, and so on
- **CryptoNight**: This is used in Monero, Bytecoin, and so on
- **QuBit**: This is used in Geocoin, DGB-Qubit, and so on

- **Dagger Hashimoto**: This is used in Ethereum, Ethereum Classic, and so on
- **Lyra2RE**: This is used in Verge, Zcoin, and so on
- **X15**: This is EverGreenCoin, Kobocoin, and so on

Apart from innovative algorithms, there are also a lot of innovative consensus types commonly known as proof types, being used, such as these:

- **Proof of Work** (**PoW**): This is the most commonly used proof type; in this proof type, the calculations are required by the network node to form the ledger by the process of mining.
- **Proof of Stake** (**PoS**): This was first used in Peercoin. With Proof of Stake, instead of mining a node stakes (holds) coins and validates ownership. A new block creator is predicted by using algorithms by searching for the lowest hash value combined with the size of the stake. The nodes can predict which stake will create the next block. No mining is required in this type of system.
- **Proof of Stake Anonymous** (**PoSA**): This is a variation of the POS algorithm, first introduced in Cloakcoin; with Proof of Stake, the transactions are cloaked by other nodes that receive a reward for helping in the process. The nodes provide inputs and outputs to the transaction, making it impossible to figure out the destination or the source of the specified transaction.
- **Delegated Proof of Stake** (**DPoS**): With delegated proof of stake, users act as delegates who are given power to earn from running a full node of the blockchain. Bitshares introduced this type of proof; some more coins using it are Steem, EOS, Lisk, and so on.
- **Proof of Importance** (**POI**): With Proof of Importance, every account on the Blockchain is given an importance score; this score influences how each node can harvest the blockchain. The task of the harvesters on the blockchain is to add transactions on the blockchain in exchange for a small economic incentive. As the importance score increases, the chance of getting the rewards also increases.
- **Proof of Capacity or Proof of Space** (**PoSpace**): In this type of proof, instead of using blockchain, storage is used. This type is considered an economical and greener alternative due to the general-purpose nature of storage and the cheaper energy cost required by storage. Some of the theoretical and practical implementations of PoC are Burstcoin, SpaceMint, and others.
- **Proof of Stake Time** (**PoST**): In this approach, consensus is formed by introducing a stake-time component, wherein the stake increases over time. This provides an incentive staking process and boosts the security of the network. The goal of this type of proof is to avoid common Proof-of-Stake methods, where nodes with higher numbers of coins used to get more economic incentives.

- **Proof of Stake Velocity** (**PoSV**): In this method, rewards are distributed based on how many coins a node has and how actively the transactions happen on the node.
- **Proof of Activity** (**POA**): In this proof type, a hybrid approach is followed, combining Proof of Work with a Proof of stake-based system. In this system, mining occurs just like in PoW systems. Although the blocks mined do not contain any transactions, only header and the miner's reward address. Then, the system switches to PoS, where a random group of validators are chosen; the block then becomes a fully fledged block as soon as all validators sign it.
- **Proof of Burn** (**PoB**): In this proof, we have to prove that coins are burned in the process of sending a transaction to an address that is undependable. Although this only works with coins mined from PoW currencies. In short, coins are bootstrapped by destroying the value of another PoW coin.

Due to the large amount of Altcoins and larger number of coins being released regularly, it is important to know about the differences each of the coins offers. It is now time to discuss various Altcoins and what each one of them offers.

Litecoin

This is one of the initial Altcoins, released in 2011. The prime modification of Litecoin over Bitcoin is the use of the script algorithm instead of SHA-256 used in Bitcoins. Also, the limit of coins for Litecoin is 84 million, compared to 21 million Bitcoins.

Litecoin was created in October, 2011, by a former Google engineer, with its prime focus to reduce the block generation time from Bitcoins from 10 minutes to 2.5 minutes. Due to faster block generation, the confirmation of a transaction is much faster in Litecoin compared to Bitcoin.

Ether

Ether is based on an Ethereum platform, which provides scripting functionality. Represented by the symbol ETH, it has a block time of 14 to 15 seconds. It is based on PoW using the Ethash algorithm. The circulation limit is suggested to be around 120 million coins.

Ripple

Ripple is a cryptocurrency backed by a company by the name of Ripple, which is a system of **Real-Time Gross Settlements (RTGS)**. It is denoted by the symbol XRP. It uses the **Ripple Protocol Consensus Algorithm (RPCA)**, which is applied every few seconds by all the nodes in the network to maintain the agreement of the network. The organization backing Ripple plans to create no more than 100 billion ripples. As per the plan, half of those will be released for circulation, and the other half will be retained by the company.

Bitcoin Cash

On August 1, 2017, the community of Bitcoin developers went ahead and split the blockchain into two. This kind of split is known as a fork, and is introduced to add new features to the blockchain. Bitcoin Cash is a result of the first hard fork to split Bitcoins. Bitcoin Cash has a relatively lower transaction fee, as well as a lower mining difficulty, compared to Bitcoin.

Acquiring cryptocurrency

There are various ways in which Altcoins or Bitcoins can be obtained; if the coin supports mining, primarily the coins that work on PoW algorithms fall into this category, those coins can be obtained by a mining process. Coins such as Bitcoin, Litecoin, Ether, Bitcoin Cash, Monero, and others support mining. As discussed earlier, there are various exchanges where cryptocurrencies are exchanged, for fiat currency, or even other cryptocurrency; this is another widely used method.

Mining of cryptocurrency

Mining is a process by which new blocks are added to the blockchain. Transactions are validated by the mining process by mining nodes and kept in the blocks, then these blocks are added to the blockchain. This is a highly resource-intensive process to ensure the required resources are spent by the miners for the block to be accepted.

Each cryptocurrency has a different mining difficulty, which is defined by the block height, the number of miners mining that cryptocurrency, the total transactions of the currency, and block time.

Bitcoin mining

The block creation time for Bitcoin is 10 minutes. Miners get rewards when new blocks are created and are also paid the transaction fees considering the blocks contain the transactions that are included in the block mined. The block time is maintained at 10 minutes, which makes sure blocks are created at a fixed rate. The reward for the mining of blocks is halved every 210,000 blocks, roughly every four years. When Bitcoins were initially introduced, the block reward was 50 Bitcoins, which was halved in 2012 to 25 Bitcoins. In July 2016, the reward for mining each block was again halved to 12.5 coins. The next date for reducing the block reward is July 2020, which will reduce the coin reward to roughly 6.25 coins.

Due to the halving event, the total number of Bitcoins in existence will not be more than 21 million. The decreasing supply was chosen as it is similar to other commodities such as gold and silver. The last Bitcoin will be mined in 2140, and no new mining reward will be present after that, although the transaction fees would still be awarded for generations of blocks containing transaction fees.

Mining difficulty

This is a measure of the mining hash rate; the Bitcoin network has a global block difficulty. Valid blocks need to have a hash rate below this target. The difficulty in network changes every 2,016 blocks. Other coins have their own difficulty or have implemented a modified version of the Bitcoin difficulty algorithm. Here is the difficulty adjustment formula for Bitcoin:

```
difficulty = difficulty_1_target/current_target
difficulty_1_target =
0x00000000FFFF0000000000000000000000000000000000000000000000000000
```

Here, `difficulty_1_target` is the maximum target used by SHA256, which is the highest possible target and is the first difficulty used for mining the genesis block of Bitcoin.

The reason for difficulty regulation in Bitcoin is that 2,016 blocks take around two weeks, since block time is maintained at around 10 minutes. If it takes longer than two weeks to mine 2,016 blocks, then there is a need to decrease the difficulty, and if it takes less than two weeks to mine 2,016 blocks, then the difficulty should be increased.

Difficulty of genesis block of Bitcoin, here is the block header:

```
$ Bitcoin-cli getblockhash 0
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

$ Bitcoin-cli getblockheader
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
{
 ...
 "height": 0,
 ...
 "bits": "1d00ffff",
 "difficulty": 1,
 ...
}
```

As you can see, the genesis block has `1` as difficulty and `1d00ffff` bits, which is the target hash value. Here is the code for this re-targeting difficulty algorithm in the `pow.cpp` file in the Bitcoin source:

```
 // Go back by what we want to be 14 days worth of blocks
 int nHeightFirst = pindexLast->nHeight -
(params.DifficultyAdjustmentInterval()-1);
 assert(nHeightFirst >= 0);
 const CBlockIndex* pindexFirst = pindexLast->GetAncestor(nHeightFirst);
 assert(pindexFirst);
```

Here is the limit adjustment step in the same `pow.ccp` file of the Bitcoin source code:

```
// Limit adjustment step
int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
if (nActualTimespan < params.nPowTargetTimespan/4)
nActualTimespan = params.nPowTargetTimespan/4;
if (nActualTimespan > params.nPowTargetTimespan*4)
nActualTimespan = params.nPowTargetTimespan*4;


   // Retarget
    const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
    arith_uint256 bnNew;
    bnNew.SetCompact(pindexLast->nBits);
    bnNew *= nActualTimespan;
    bnNew /= params.nPowTargetTimespan;

    if (bnNew > bnPowLimit)
        bnNew = bnPowLimit;
```

The re-targeting adjustment should be less than a factor of 4 in a single cycle of two weeks. If the difficulty adjustment is more than a factor of 4, it won't be adjusted by the maximum factor. Further adjustment should be achieved in the next two weeks of the cycle. Hence, very large and sudden hashing rate changes take many two-week cycles to balance in terms of difficulty.

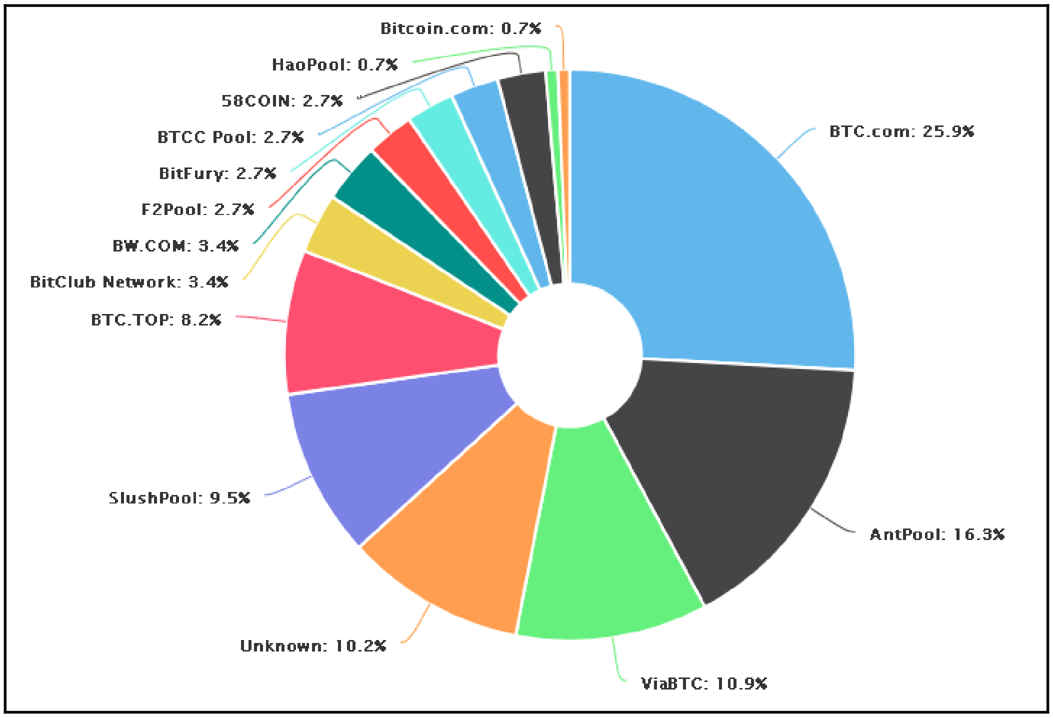
Due to the introduction of ASICs, the hashing power increased exponentially, following which the difficulty of mining increased.

Mining pools

Due to the huge mining hash rate, people from the mining community came together to mine blocks together and decided to split the reward depending on how much hash power each was contributing to the pool. In a mining pool, everyone gets a share of the block reward directly proportional to the resources invested. Many mining pools exist nowadays. Some of the popular mining pools are these:

- **BTC.com**: This is one of the largest mining pools, responsible for around 25% of Bitcoin blocks mined on a regular basis as of June, 2018. The pool was launched in September, 2016, and is currently owned by Bitmain Technologies Ltd.
- **Antpool**: This is owned by the same company that owns Btc.com, one of the largest mining pools. Antpool and Btc.com share the top positions among the largest mining pools. Bitmain is based in Beijing, China.
- **ViaBTC**: This was founded in May, 2016, and is owned by Viabtc Technology. It is based in China.
- **SlushPool**: This is the world's first mining pool. It started on November 27, 2010, and went by the name of Bitcoin Pooled Mining Server. The pool has mined over 1 million BTC since its launch. It is a reliable and stable pool with a history of accurate payouts.

The following diagram shows the percentage of Bitcoins mined by the various Bitcoin mining pools between June 1 and June 2, 2018:



Apart from the ones we have mentioned, there are many mining pools that actively mine and keep on adding more and more features in the race to become the largest mining pool.

Altcoin mining

As we have discussed, various Altcoins have different algorithms; each of them has corrections and improvements to increase difficulty and avoid centralization. Currently, apart from Bitcoin mining, various other Altcoins are also mined regularly. Almost every mining pool right now supports Altcoins; some of the most popularly mined Altcoins as of June, 2018, are Ether, Litecoin, Zcash, Dash, Bitcoin Cash, Ethereum classic, and so on.

There are various discussions about mining profitability; due to the very high difficulty of Bitcoin mining, other alt currencies are becoming popular options for miners, as well as mining pools. Nowadays, ASIC miners are also coming up for Altcoins such as Litecoins, Cash, Ether, and so on.

Due to major GPU and resource requirements by Bitcoin and other SHA-256 based coins, this lot of CPU-friendly mining coins were created based on script.

Cryptocurrency exchanges

There are many exchanges where users can buy or sell Bitcoins and other Altcoins. Exchanges can deal in fiat money, Bitcoin, Altcoins, commodities, or all of these. These exchanges usually charge a small fee for the trade done on their platform. Some of the popular Cryptocurrency exchanges are these:

- **Binance**: One of the top exchanges in the world, having more than 150 Altcoins listed on it, increasing day by day. It is one of the most active cryptoexchanges.
- **Huobi**: Another popular exchange, founded in September, 2013. It is one of the largest exchanges in China. More than 250 coins are listed on the Huobi exchange.
- **Bithumb**: One of the largest cryptoexchanges based in South Korea, with a huge daily trading volume. Bithumb does not list a vast number of coins; as of now, only 20 coins are listed on Bithumb.
- **Bitfinex**: Hong Kong-based cryptocurrency exchange owned by iFinex, Inc. More than 75 coins are listed on Bitfinex.
- **OKEx**: One of the largest cryptoexchanges, with more than 500 coins listed on its platform. Started in 2014, it is also known for BTC futures trading, in which it generates huge daily volumes.
- **Kraken**: One of the first Bitcoin exchanges, started in 2011. It has been through lot of security breaches in mid of 2017 but has been stable post that. Kraken provides Bitcoin pricing to the Bloomberg terminal.
- **Cex.io**: A popular cryptocurrency exchange founded in 2013. Formerly known for cloud mining services. The mining pool of Cex.io know as Ghash.io contributed over 42% in Bitcoin mining hash rate in 2014.
- **Bitstamp**: One of the oldest exchanges started in 2011. It is most popular for fiat to cryptoexchanges.
- **HitBTC**: A UK based cryptocurrency exchange started in 2015. It supports a vast list of currencies.
- **Upbit**: A south korea based cryptocurrency exchange offering fiat deposit in South korean won.
- **Bittrex**: It is a US based cryptocurrency exchange founded in 2013. It works with a vast amount of crypto currencies.

Apart from the ones mentioned here, there are many other popular exchanges, some focusing primarily on fiat to cryptocurrency purchase, and others working on only cryptocurrencies. Some other prominent exchanges are Upbit, Bittrex, Lbank, Bit-Z, HitBTC, coinbase, BCEX, GDAX, Gate.io, Bitstamp, EXX, OEX, Poloniex, Kucoin, Cobinhood, Yobit and so on.

Cryptocurrency wallets

A **cryptocurrency wallet** is a collection of private keys to manage those keys and transfer them from one wallet to another. Bitcoin wallets are compared on the basis of security, anonymity, ease of use, features, platforms available, and coins supported. Usually, all of the cryptocurrencies have their own official wallets, but other third-party wallets can also be chosen based on requirements. Some of the prominent cryptocurrency wallets are these:

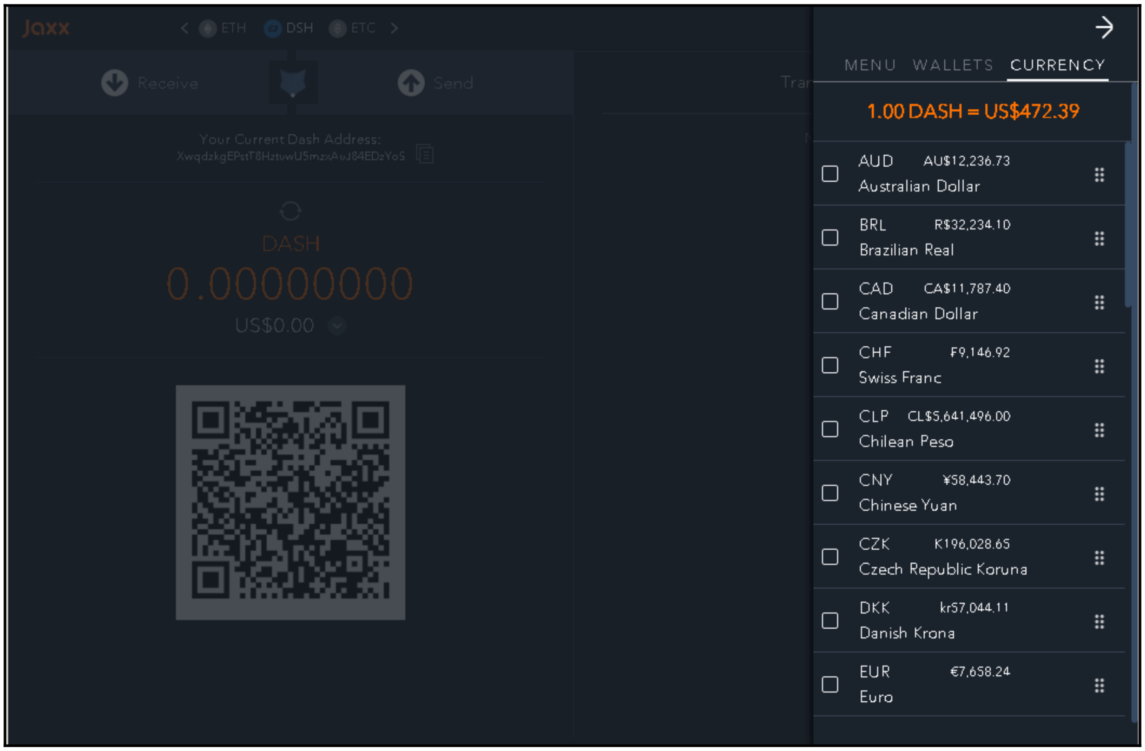
- **Coinbase**: This is a digital wallet that started in July 2011, apart from storing currencies, they also offer the selling and buying of cryptocurrencies. The currencies supported are Bitcoin, Bitcoin Cash, Ethereum, and Litecoin.
- **Blockchain.info**: This is a widely used Bitcoin wallet, as well as a Block Explorer service. It was launched in August, 2011. It supports Bitcoin, Bitcoin Cash, and Ethereum.
- **Jaxx**: A multicurrency wallet, created in 2014, supporting multiple coins such as Bitcoin, Ether, Ethereum classic, Dash, Litecoin, Zcash, Augur, and others.

Here is a list of some more cryptocurrency wallets which offer multi-currency support:

- Trezor
- Blockchain wallet
- Metamask
- Openledger
- Exodus
- Ledger Nano

Apart from the third-party wallets we have mentioned, there are many other wallets offering different features. It should be noted that some wallets charge higher transaction fees, compared to actual network fees, to cover their development cost.

The following screenshot shows the Jaxx cryptocurrency wallet:



Your Current Dash Address:
XwqdzkgEPstT8HztuwU5mzxAuJ84EDzYoS

0.00000000

1.00 DASH = US$472.39

AUD	AU$12,236.73	Australian Dollar
BRL	R$32,234.10	Brazilian Real
CAD	CA$11,787.40	Canadian Dollar
CHF	F9,146.92	Swiss Franc
CLP	CL$5,641,496.00	Chilean Peso
CNY	¥58,443.70	Chinese Yuan
CZK	K196,028.65	Czech Republic Koruna
DKK	kr57,044.11	Danish Krona
EUR	€7,658.24	Euro

Summary

In this chapter, we discussed alternative currency and the difference between a coin and a token. We discussed in detail the various platforms, based on which platforms can be created. Furthermore, we discussed various alternatives that Altcoins provide over Bitcoins. We read in detail about currencies such as Litecoin, Ether, Ripple, and Bitcoin Cash.

We also discussed acquiring a cryptocurrency and the various ways to do so. We read about mining a cryptocurrency and the differences between Bitcoin and Altcoins, in terms of mining. We discussed exchanges and how we can store Altcoins other than Bitcoins in wallets. We also learned about the difficulty of re-targeting an algorithm in Bitcoin.

7
Achieving Consensus

The concept of consensus is straightforward: consensus is when the network agrees on what information stored in the network is true and should be kept, and what is not true and should not be kept. For Bitcoin, achieving consensus is a simple matter of coming to agreement on the set to send and receive of Bitcoin across the network. For other networks, achieving consensus would also involve coming to an agreement on the final state of smart contracts, medical records, or any other network information stored on the blockchain.

Consensus algorithms have been the subject of research for decades. The consensus algorithms for distributed systems have to be resilient to multiple types of failures and issues, such as corrupt messages, parts of a network connecting and disconnecting, delays, and so on. In financial systems, and especially in blockchains, there is a risk of selfish and malicious actors in the system seeking profit. For each algorithm in a blockchain network, achieving consensus ensures that all nodes in the network agree upon a consistent global state of the blockchain. Any distributed consensus protocol has three critical properties:

- **Safety**: The ability of all the nodes on a distributed network to guarantee that they have the same state or consistency
- **Liveness/Availability**: The protocol is guaranteed to succeed and have the different nodes produce a final result
- **Fault tolerance**: The ability of the protocol to deal with nodes producing faulty or hostile results

As it happens, a famous paper by Fischer, Lynch, and Paterson stated that it's impossible for all three to be true in the same asynchronous distributed system. Hence, any and all blockchain designs must make trade-offs between these properties. These trade-offs are typically between safety and liveness, as fault tolerance is generally seen as a must-have for a globally distributed network.

In blockchain systems, there are currently four primary methods of achieving consensus. They are as follows:

- Practical Byzantine fault tolerance algorithm
- PoW algorithms
- PoS algorithms
- Delegated PoS algorithms (DPoS)

These approaches will each be covered in turn in this chapter.

Practical Byzantine fault tolerance algorithm

Practical Byzantine fault tolerance (**PBFT**) algorithm. Many algorithms are called **Byzantine fault tolerant**. The name comes from the allegory that presented the original problem.

Imagine an ancient Byzantine army moving to capture a city. The idea is to attack from all sides. Once the generals of the army reach the city, they must agree on when and how to attack. The difficulty is in how to agree. The generals can communicate only by messenger, but the messengers could be captured by the enemy, and there is the additional fear that one or more of the generals or their commanders are traitors.

The generals need a method to ensure that all the loyal generals agree on the same plan, and that a small number of possible traitors cannot cause the mission to fail.

The loyal generals will all do what the method says they will do, but the traitors might do anything. How can the generals create a method that ensures that, as long as most of them are loyal, their plans will succeed?

This allegory is also sometimes called the **Chinese general's** problem, as well as a few other names, but the issue remains the same: how can different parties securely communicate and reach an agreement about something when communication channels may not be secure, and when there may even be traitors in their midst.

In the case of blockchain, the generals in the story are the computers that are participating in the distributed network running the blockchain. The messengers represent the digital network that these machines are running on and the message protocols used by those machines. The goal is for the good computers or generals to decide which information on the network is valid while rooting out bad actors and preventing false information from being recorded on the blockchain.

The loyal generals in the story represent the operators of honest nodes that are interested in ensuring the integrity of the blockchain and the applications based on it, and that are therefore invested in making sure that only correct data is recorded. The traitors represent the many bad actors of the world who would love to falsify data (especially financial data) for personal gain or on behalf of some other antagonistic party. The motivations of the bad actors could be varied, from spending Bitcoin they do not truly possess or getting out of contractual obligations, or even trying to destroy the network as a form of currency control by a hostile government.

Byzantine faults

To understand PBFT and all the other consensus algorithms that come afterward, it's important to first define what a Byzantine fault is. A Byzantine fault is any event or result that would disrupt the consistency of a distributed system, such as the following:

- Failure to return any result
- Return of an incorrect or inconsistent result
- Return of a deliberately misleading result
- Any behavior that isn't defined beforehand

If any of these events happen, a Byzantine fault is said to have occurred. A Byzantine fault tolerant system is therefore able to handle some level of inconsistent input, but still provide a correct result at the end. The key here is that such systems are fault tolerant, not fault immune. All fault tolerant systems can only tolerate so much before their tolerance is exhausted and the system fails in some way.

How PBFT works

Hyperledger is the primary blockchain that uses PBFT. Here is how PBFT works in Hyperledger. Each validating node (a computer running the blockchain software and working to maintain consistency) keeps a copy of the internal state of the blockchain. When a node receives a message, it uses the message in conjunction with their internal state to run a computation on what the new state should be. Then the node decides what it should do with the message in question: treat it as valid, ignore it, or take another course of action. Once a node has reached its decision about the new message, that node shares that decision with all the other nodes in the system. A consensus decision is determined based on the total decisions submitted by all nodes:

1. **Submission**: One or more of the nodes on the network submit a transaction that is sent to the other nodes. For instance, if there are ten nodes participating and three of them send messages, it looks as follows:
    - All ten computers see three transactions
    - These transactions are distributed so that each node has a full copy of all the transactions
    - These transactions may arrive at nodes at different times, so the order may not be consistent
2. **Ordering**: One of the validating nodes is elected as the leader by a vote of the other nodes. This validating leader chooses the order of transactions and sends this to the other participating nodes. Each of the other validating nodes then rearranges the transactions they already had into the order set by the validating leader.
3. **Execution**: The validating nodes then execute the newly ordered transactions. Each node independently executes all changes and adds these changes to the global state from previous blocks. If consensus cannot be reached, the transactions are rolled back and rejected.

This process is repeated for each block. The advantage of PBFT is that it is very fast and scales relatively well. The downside is that the participants must be known—not just anyone can join the network.

Proof of Work

The first consensus algorithm used in blockchains was Bitcoin's **proof-of-work** (**PoW**). Proof-of-work fundamentally functions by exploiting a feature of certain cryptographic functions: there are mathematical problems that are very hard to solve, but once they are solved, they are very easy to check. As discussed before, one of these problems is hashing: it's very easy to take data and compute a hash from it, but extremely difficult to take a hash and discover the input data. PoW is most notably used by Bitcoin, Litecoin, and Ethereum.

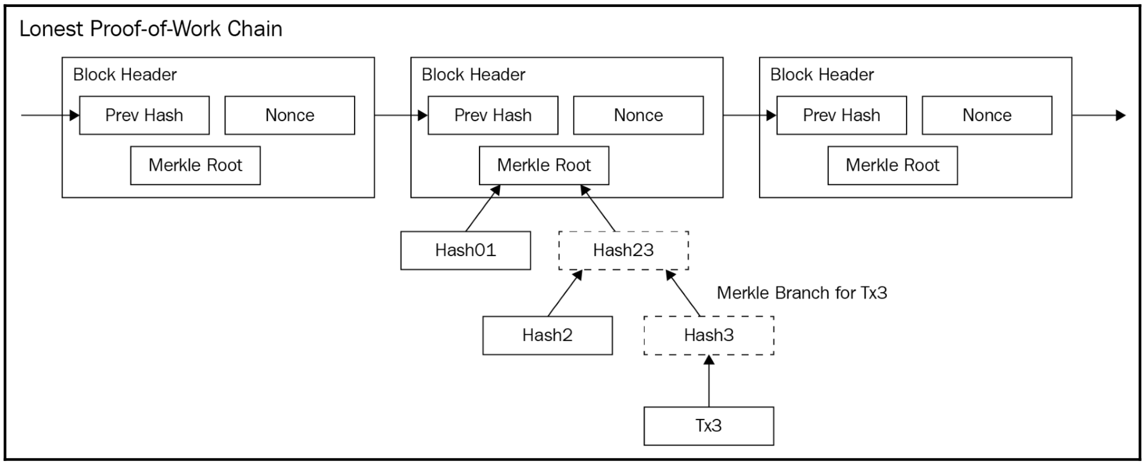
PoW has the following characteristics:

- **Relatively predictable time to solution**: Bitcoin's network protocol expects each block to take about ten minutes to solve. If the network starts to solve the proof-of-work problem too quickly, the network will automatically increase the difficulty.
- **Resistant to large increases or decreases in computing power**: Moore's law suggests that the amount of work computers can do is expected to double every two years. In addition, because the network is open, anyone can add vast computing resources to the network at any time. For the network to remain stable, the algorithm must automatically adjust itself. Similarly, if the network ceases to be profitable, then the amount of computing power being used will drop as those resources are redirected. This is achieved through the automatically adjusting difficulty in the algorithm. It must be easy for any network participant to quickly check that they have the right chain and that the chain is valid. This is achieved through the use of hashing functions.

The proof-of-work algorithm maintains network integrity as long as no group of actors controls more than 50% of the overall network computing power. The possibility of bad actors being able to control the chain is called the **51% attack**. If a single group ever controls more than half the network power, they can control the network and network transactions by halting payments or even doubling spending. The attacking group would be able to prevent new transactions from being confirmed (halting payments for users as they see fit) and even reverse transactions that happened after they had started controlling the network.

How the PoW problem works in Bitcoin

The PoW algorithm starts by taking the longest chain. In Bitcoin, there are multiple ways blocks can be finalized (depending on the included transactions). Thus, there may be multiple available "solved" chains that could be selected as a base by the Bitcoin nodes. As part of the algorithm, Bitcoin takes the chain that is the longest and thus has had the most computing power applied to it. The following diagram illustrates a PoW chain:



The difficult puzzle in Bitcoin is to find an input that, when added to the prior block hash and the list of transactions, produces a hash that starts with a certain number of zeros.

Typically, the input to the function is the Merkle root of all transactions and the prior block hash. To simplify this for illustrative purposes, imagine that we have a simple input, such as *I love Blockchains*. Let's also assume that the system has the easiest possible difficulty: a single zero at the start of the hash. The SHA-256 hash of *I love Blockchains* is as follows:

```
ef34c91b820b3faf29104f9d8179bfe2c236d1e8252cb3ea6d8cb7c897bb7d96.
```

As you can see, it does not begin with `0`. To solve the block for this input, we need to find a string (called a **nonce**) that we can add to this string so that hashing the combination (*nonce* + *I love Blockchains*) results in a string starting with `0`. As it turns out, we can only do this through testing. For instance, if we add `1` to the beginning, we get *1I love Blockchains*, and the hash is as follows:

```
b2fc53e03ea88d69ebd763e4fccad88bdb1d7f2fd35588a35ec6498155c702ed
```

No luck. What about 2 and 3? These will also fail to solve the puzzle. As it happens, *4I love Blockchains* has a hash that starts with `0`:

```
    0fd29b2154f84e157d9f816fa8a774121bca253779acb07b07cfbf501825415d
```

It only took four tries, but this is a very low difficulty. Each additional zero doubles the challenge of finding a proper input that will compute a proper hash. As of writing, a valid Bitcoin block requires 18 zeros to be valid.

This process of trying to find the nonce that results in a proper hash is called mining. Every computer mining a PoW chain is competing to see who can find a proper nonce first. The winner gets to create the next block in the chain and is rewarded in tokens. For more details, see `Chapter 18`, *Mining*.

The advantage of PoW is that anyone can join a PoW network, and it is well established as a functional consensus mechanism. The primary downsides of PoW networks are slow speeds and financial costs: running all the computers to do these computations is very expensive, and the output is not put to any real productive use. This is considered bad for the environment, and can result in increased energy prices wherever large amounts of blockchain mining are done. In some areas, blockchain mining has been banned for this reason.

As a result of these downsides, **proof-of-stake** (**PoS**) was invented.

Proof of Stake

PoS has the same objectives as PoW to secure the network against attack and to allow consensus to occur in an open network. The first digital currency to use this method was Peercoin, and was followed by many others, such as NXT, Dash, PIVX, and so on. In PoW networks, solving the puzzle is what determines which node gets to create the next block in the chain. In PoS networks, blocks are said to be forged instead of mined, as they are in proof-of-work blockchains. In PoS chains, the validators get rewarded by getting paid the transaction fees for each block, and sometimes in additional coins created automatically each time a block is created. In PoS chains, the chance to be the creator of the next block is determined by the amount of investment a node has in the network.

Have a look at the following example:

There are five nodes in a PoS network. They have the following balances:

1. 10,000 coins
2. 200 coins
3. 300 coins
4. 4,000 coins
5. 20,500 coins

The total number of tokens is 35,000 coins. Assuming that each node is staking 100% of their coins, every block and the nodes they contain will have the following likelihoods of being the next block signer:

1. 28.57%
2. 0.57%
3. 0.86%
4. 11.4%
5. 58.6%

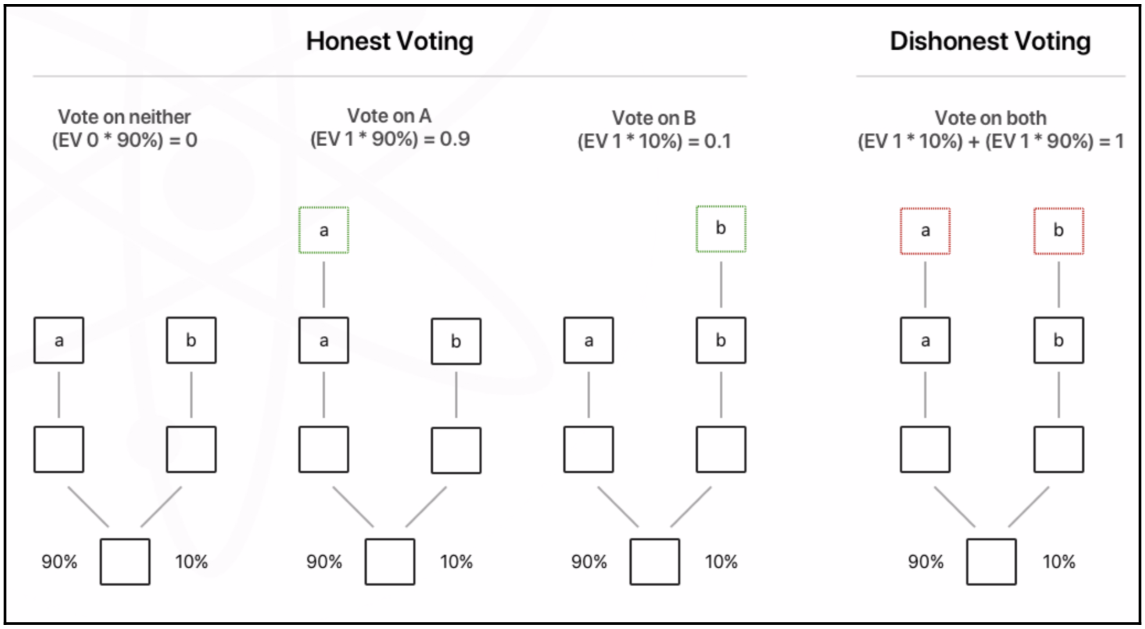
It should be pretty obvious that, if a single node ever controls the majority of tokens (or even a large fraction), then they will have substantial control over the network. In this case, node #5 would end up creating more than half the blocks. Moreover, because node #5 would be regularly signing blocks, it would also get the majority of the transaction fees and new coins that are created. In a way, PoS rewards validators with interest on their investment in the form of additional tokens. One criticism of PoS networks is that the *rich get richer*, which can lead to increasing network centralization and control.

The nothing-at-stake attack

One of the issues in PoS systems is the threat of nothing-at-stake attacks. In a nothing-at-stake attack, a validator actually creates multiple blocks in order to spend tokens multiple times. Because of the low cost of creating blocks in PoS systems, there is no financial incentive to the network not to approve all the transactions, causing consensus to break down.

For instance, imagine a bad actor, Cain, who only has 100 tokens. He decides to try and to cheat, and sends two messages to the network: one in which he sends his 100 tokens to Sanjay, and another where he sends his 100 tokens to Eliza. The network should accept either transaction, but not accept both. Typically, the nodes would have to come to consensus about which transaction is valid or reject both of them. However, if a validator is cooperating with Cain (or is run by Cain himself), then it turns out it is to their financial advantage to approve both blocks.

In the following diagram, **expected value** stands for the **EV**. It shows that if a validator accepts both blocks, it can effectively double spend without penalty:


Honest Voting
Dishonest Voting
Vote on neither (EV 0 * 90%) = 0
Vote on A (EV 1 * 90%) = 0.9
Vote on B (EV 1 * 10%) = 0.1
Vote on both (EV 1 * 10%) + (EV 1 * 90%) = 1
a
b
90%
10%

To avoid this problem, PoS systems have introduced various countermeasures, such as staking deposits. In the case of a blockchain fork or a double-spend attack, the validators that participate risk losing their tokens. Through financial penalties and loss of staked tokens, the incentive to double spend and validate all blocks is thought to be reduced or eliminated.

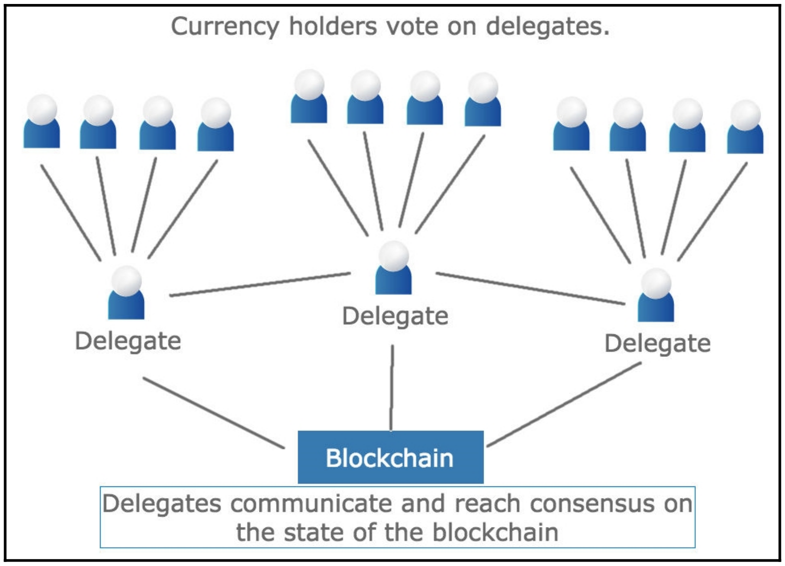
Variations

There are numerous variations on the basic PoS approach. Each variation will have different requirements, such as the minimum balance needed to have a stake, the potential penalties for bad behavior, the rights and abilities of the stakeholders of the network, and modifiers, such as how long an account needs to have had a staked balance in order to be counted.

Delegated Proof of Stake

DPoS is related to PoS consensus, but with some critical differences. This new system is the creation of Dan Larimer of Bitshares, Steemit, and currently EOS. Both these networks and Lisk (another commonly used blockchain) are currently the only major blockchains that use this approach. In DPoS, the holders of tokens are not the ones doing block validation. Instead, they can use their tokens to elect a node to validate on their behalf—their delegate (also called a **validator**). It is this delegate/validator that helps operate the network. The number of available validator slots tends to be locked to a specific number, typically 21. In order to become a delegate, the owner of the node must convince the other users of the network to trust them to secure the network by delegating their share of the overall tokens on the network to them. Essentially, each token on the network acts as a vote, and the top vote holders are allowed to operate the network. Currently, only Bitshares, Steemit, EOS, and Lisk are the major blockchains that use this approach.

In DPoS, each delegate has a limited, designated time in which to publish a new block. If a delegate continually misses their block creation times or publishes invalid transactions, the token holders using their stake can vote them out and replace them with a better delegate. The following diagram shows what this structure looks as follows:

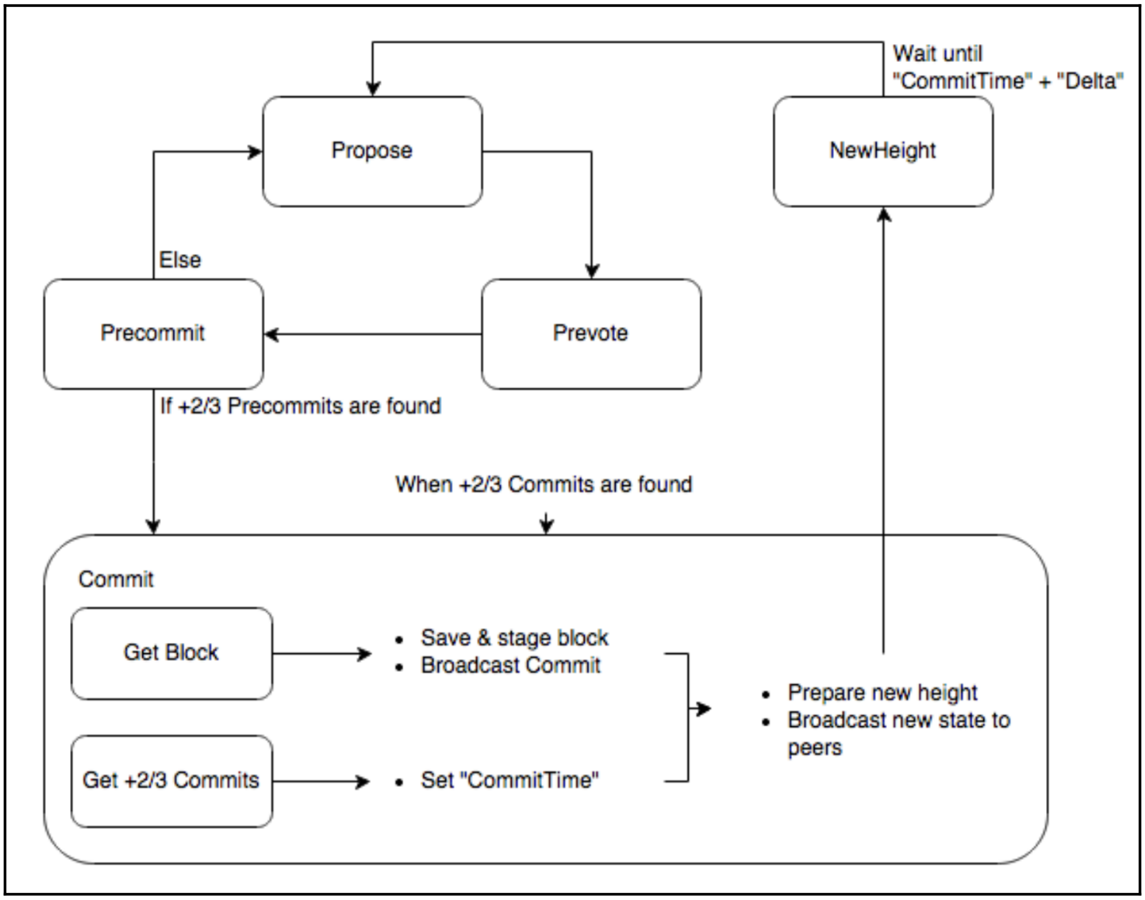

Currency holders vote on delegates.
Delegate
Delegate
Delegate
Blockchain
Delegates communicate and reach consensus on the state of the blockchain

The primary criticism of DPoS is that it is partially centralized and has no real immediate financial penalty for betraying the network. The consequence of violating network rules is to be voted out by the token holders. It is thought that the cost to reputation and the loss from campaigning for delegated shares will outweigh the financial benefit of trying to negatively influence the network. By only having a small number of delegate slots, it is easier for the token holders to pay attention to the behavior of individual validator nodes.

Tendermint consensus

Tendermint uses a custom consensus engine, designed as part of a doctoral thesis by Jae Kwon. It is similar to DPoS in that participants in the network can delegate their voting power to a validating account. However, to do so, they must bond or lock their tokens. To do this, they must issue a special bonding transaction in which their coins are locked to a validating node. In the event that their delegate misbehaves, both the delegate and the accounts lending their coins forfeit some portion of their bonded tokens. To release their tokens, another special unbonding transaction must be posted to the network, and such withdrawals are subject to a long delay.

Let's look at how these transactions happen. The following diagram is from the Tendermint documentation:



Let's look at the preceding figure in more detail. Delegates signal the next block by signing votes. There are three types of votes: prevotes, precommits, and commits. Each block has a special validator called a **proposer**. The proposer goes first, suggesting a valid block state based on a prior locked block. This proposal is shared peer-to-peer among the other validators, and if 2/3 or more vote in agreement with the locked block (in the prevote stage) then they move to the next stage: precommit. In the precommit stage, again, if 2/3 agree with the prevote condition, they will signal that they are ready to commit. Finally, the actual commitment of the block takes place: the node must have received the block, and it must have received 2/3 valid votes to precommit.

If this sequence of 2/3 votes seems unusual, it is because of the nature of asynchronous networks, where the validators may receive blocks and votes at different times. This sequence, and the edge cases that are handled when the 2/3 majority is not reached, allow for effective and fast consensus on unreliable networks.

Proof of Authority

Proof-of-authority (**PoA**) networks are used only when all blockchain participants are known. In proof-of-authority, each participant is known and registered with the blockchain. Such a blockchain is called a **permissioned chain**, as only computers that are part of this approved list of authorities are able to forge blocks. It is critical, therefore, that none of the authority computers is compromised, and each operator must take pains to ensure the integrity of their validator. This approach was originally shared by Gavin Wood of Parity Technologies as a different way of running an Ethereum-based blockchain.

Establishing authority

The three main conditions that must be fulfilled for a validator to be established are as follows:

- Identity needs to be formally verified and on chain.
- Eligibility should be difficult to obtain. Examples are things such as becoming a notary public, submitting to background checks, and posting a bond.
- The set of things that must be required of each authority should be well documented, uniform, and worthy of the network's trust.

Once an authority has been established, the right to forge new blocks might be granted by adding the authority to the list of valid validators for the blockchain.

While PoA is mostly used in private chains, it can be used in public chains as well. Two public Ethereum test networks, Rinkleby and Kovan, are public blockchain networks that use PoA as their consensus mechanism.

The obvious downside of PoA is that the identity of each validator operator must be known and trusted, and the penalties for abusing that trust must be real. For global blockchains, this may not be preferred, as one of the appeals of blockchain technology is the ability to anonymously exchange value.

Proof of Elapsed time

The Hyperledger Sawtooth project introduced a new consensus mechanism called **proof-of-elapsed-time** or **PoET**. Hyperledger deals mostly with permissioned blockchains, chains in which only a specified number of participants are allowed on the network, similar to PoA chains.

The basic approach is simple:

- Each node must wait a random amount of time
- The first node to stop waiting gets to create a block

There are two things that we must be able to do for this to work. First, we must be able to verify that the waiting time for all participants was actually random, or else a simple attack would be to pretend to wait a random time and then just immediately create a new block. Second, it must be verifiable that not only was the length of time chosen random, but that the node actually waited the full period of time before acting.

The solution to these issues comes from Intel (who created the PoET algorithm), and relies on special CPU instructions to ensure that only trusted code is run. By forcing trusted code to be in charge of handling block timing, the system ensures that the lottery is fair.

Summary

At this point, you should have a solid foundation in the different mechanisms that blockchains use to reach consensus. Each consensus algorithm makes certain trade-offs between speed, availability, consistency, and fault tolerance. The most common consensus mechanisms are still PoW and PoS, but blockchain development continues at a very rapid pace, and new and improved approaches are likely to be developed. Improvements to consensus algorithms will improve blockchain scalability and reliability, and the scope of the potential applications for the technology.

References

1. https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf
2. https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf
3. https://github.com/tendermint/tendermint.com/blob/5c111743a03d2c6ed2e0b14bd3091cac8974c8da/docs/tendermint_v02.pdf
4. https://peercoin.net/assets/paper/peercoin-paper.pdf
5. https://github.com/ethereum/guide/blob/master/poa.md
6. https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256

8 Advanced Blockchain Concepts

Privacy is discussed very frequently in the tech world—especially now that social media executives are grudgingly suiting up and being paraded in front of US Senate committees.

At the same time, as blockchain advocates are happy to see this technology advancing human welfare and decentralizing money transfers, it's natural to wonder whether a user can actually have any privacy, what with all transactions being public on the chain.

In this chapter, the following topics will be covered:

- Blockchain as a tool for corporate governance
    - Unbanked companies
    - The DAO
- Social-purpose uses of blockchain
- Privacy concerns within the context of the social purpose of blockchain
- Zero-knowledge crypto-systems as a solution to privacy concerns

Blockchain and banks

Privacy is sorely needed in the cryptocurrency ecosystem. Cryptocurrencies could help raise people out of poverty in developing countries and boost economies with increased money transfers—or they could be a way for oppressive regimes to track down every transaction and have more opportunities to accuse innocents of wrongdoing.

The appeal of blockchain technology to people with antiauthoritarian streaks is obvious. Many in the US have a bone to pick with the banking system. In the thirties, the Federal Housing Administration, which insures mortgages, drew maps of areas in which it would do business, excluding poor and minority communities. This practice continued for a generation, and it was devastating to the core of many American cities, destroying hundreds of billions of dollars in wealth:

More recently, the 2008 global financial crisis resulted from speculative bets on future derivatives and dodgy lending practices, which drove real-estate prices sky high (and then very low).

Some regulation was passed in response to this malfeasance, but unlike the savings and loan crisis in the 1980s, few faced criminal penalties. Since then, numerous candidates for public office have decried the perceived unfairness in the treatment of the bankers (who were bailed out) relative to others said to have committed crimes against society, such as low-level drug offenders.

Less politically, it's not difficult to find widespread dissatisfaction with the available solutions for banking and payments. Anyone who lives abroad for any length of time will have some difficulty getting access to their money. Anyone who travels will have their credit or debit card locked up with a fraud alert. How about setting up banking for a new business without a domestic office? Good luck. For added difficulty, try sending an international wire transfer from that new business bank account.

It's easy to say that banking hasn't caught up with the times, but that's not quite right. Establishing a business bank account is harder than it has been in the past. As a consequence, some startups have chosen to go unbanked, including some with which several of this book's authors were involved. This approach provides some advantages as far as administrative expense is concerned (no bank fees, no endless declarations and disclosures, no locking out users of payment cards because of fraud alerts). It also carries risk, and as we'll see in subsequent sections, going unbanked will require some development to become a viable alternative.

Unbanked – going pure crypto

It seems natural to consider a purely crypto company as the answer to the ills of banking—why not just run the entity with nothing but Bitcoin or Ethereum? As we'll see, most of the solutions involve unacceptable levels of uncertainty.

Fixing pegs

Access to fiat currency to pay certain expenses is important—most of a business' ultimate expenses are not going to be based on the price of Bitcoin or Ethereum, but the price of one or both of those currencies in US dollars or euros. Until you can pay for power and food in cryptocurrency, it's quite unlikely that the price of cryptocurrency in US dollars or euros will become irrelevant.

Some have heralded the development of a so-called **stablecoin** as the solution that would fix this problem. Various mechanisms can be used to ensure buying pressure or selling pressure in an asset, such as the backer buying back tokens when the value is below what was promised. Somewhat more simply, however, a token pegged to a currency issued by a sovereign government could be the solution.

A peg is a specific kind of fixed exchange rate system, in which the issuer of a currency asserts that their currency has a value based on that of another currency—and ideally is willing to exchange it for the same.

Economists tend not to speak of pegs favorably. The so-called **tequila crisis** in Mexico in 1994 is a famous example of a currency being overvalued. The Mexican government had issued bonds in pesos, which were redeemable in US dollars, at a valuation of about 3 pesos per dollar. It is doubtful that anyone believed the Mexican peso was worth that much. Eventually, short selling (borrowing the peso and selling it for dollars) and capital flight (investors unloading their peso-denominated assets) led to a precipitous drop in the value of the peso, and then to a severe recession.

Mexico's central bank attempted to control inflation by raising interest rates. Unlike the US and continental Europe, many mortgages in Mexico were adjustable rate, which meant that interest as a proportion of a homeowner's payment went up drastically, leading in many cases to default and foreclosure.

A handful of countries, such as Cambodia, do continue to use pegs in spite of their downsides, but it's difficult to find many other things so widely regarded as a bad idea.

Attempts to peg cryptocurrency to US dollars invariably raise money in cryptocurrency and then use it to buy sovereign debt. The assertion is that one unit of the token is equal to a dollar, or some fraction of a dollar. The tokens are backed by the debt held by the issuer, and in theory, the issuer is able to make money on the float.

If the issuer tries to minimize risk and only buys treasury bills at 1-3%—well, 1-3% of a few billion dollars is probably enough to pay your staff. The business model is pretty straightforward. PayPal famously also attempted to go fee-free and pay all of their expenses out of the float—and didn't succeed.

The most famous attempt at a stablecoin is Tether (Ticker: USDT), the price of which hovers around $1. Tether retained Friedman LLP, a New York accounting firm, and promised regular audits. These audits didn't materialize, and Friedman (likely owing to respect for the interests of the principal) has not made any statements concerning the validity of Tether's claim that each unit is backed by cash or a cash equivalent.

A peg of this nature would be a little odd in the cryptospace; ultimately underlying this sort of transaction is trust. It is almost antithetical to the idea of a trustless, decentralized ecosystem; but an alternative exists.

Buying options

It is likely that the exchange-rate risk relative to the dollar or euro will be managed in the future not by a stablecoin but instead by financial options—derivatives giving the user the option, but not the obligation, to buy or sell an asset. Users looking to manage risk will pay a slight premium to buy protective puts against their cryptocurrency positions.

A put is the right to sell an asset at a specific price. A protective put is a put purchased against an asset you already own, for less than it's worth now. In the event that the price of an asset drops below the strike price, which is the price at which an option can be executed, the option can be used to secure a greater price for the underlying asset.

The problem with existing exchanges, where options are concerned, is that there is no mechanism to enforce the option in the event that something goes wrong. Commodities were traditionally considered the most risky sort of investment. You could trade at 20:1 leverage, and if the market moved against you would have only 5% margin call! Either put in more money or lose all of it.

At least in the case of commodities, however, if a producer fails to deliver their contract of wheat or soybeans or oil, the exchange is liable. Options will need to be enforceable by escrow (the exchange holds the crypto against which the option is written) or another mechanism. Buying an option against Ethereum held in someone else's local wallet would have what is known as counterparty risk, or the risk that the other party not live up to their expectations in the event that the market moves against them.

Other contracts, such as shorts and futures, also have counterparty risk, and it might be that, in the future, rather than depending on a Tether or a bit USD to handle our fiat hedges, we have regulated exchanges with crypto or cash in escrow ready to pay out if things don't go our way.

Why regulated exchanges?

The universe tends towards maximum irony—people want to say *damn the man* and get away from sovereign currency, but most of the same people agree that government has a legitimate interest in protecting people against fraud and violence.

The trustless nature of cryptocurrency fails at the point of exchanges. If you want to exchange one cryptocurrency for another, that transaction frequently requires you to trust the exchange. Likewise, if you want another party to follow through on a transaction on which they might lose money, the other party either needs to be known to you so you can pursue recourse in the courts, or the law needs to put the onus on the exchange to pay out the claim.

The most likely outcome is that in the US, the SEC or CFTC will begin licensing exchanges, and exchanges will have certain liquidity requirements. This will make unbanked companies a less risky proposition.

Unbanked and unincorporated?

We've established a company without a bank account as a possibility (and a legal one—there's no specific requirement in many jurisdictions for a legal entity to have a bank account). But what about an organization without a legal structure, or with a highly unconventional one?

Anyone considering investing in an organization or asset without a legal entity should be very careful, because the assumption is that your personal assets are at risk. In general, people use entities in order to protect their personal assets and attain some sort of organizational or tax benefit.

Without an entity, an organization may not have limited liability, and the law may choose to treat the owners as partners. As far as liability is concerned, partnership without an entity is just about the worst treatment you can ask for. This holds true in both common law jurisdictions (which have LLCs, LLPs, corporations, trusts, foundations, and other forms) and in civil law jurisdictions (which have analogous forms such as the Luxembourgish SARL, the German GmbH, and the French SA).

Under US law, members of a partnership can be jointly and severally liable for transgressions committed by the partnership. People involved in a crypto entity, even a little bit, don't have the benefit of a corporation or other entity that could take the hit and fall over dead if someone else had a claim against it.

Limited liability is just that—your liability is limited to your investment and organizations usually distinguish between people who are passively involved in an entity (limited partners or shareholders) and those who are actively involved (general partners or officers).

Limited liability doesn't protect members of an organization against criminal liability. If the officer of a corporation commits fraud either against a shareholder or against the public, limited liability doesn't protect them.

Given that extreme caution should be exercised in considering whether the operation of any crypto project is legal, regulatory action so far has mainly been concerned with fraudulent raises fleecing investors rather than investors being involved in an organization that fleeced the public. This does not mean the investors are safe.

Consideration should be given to whether the management team is identifiable (for instance, can you at least look at someone's LinkedIn and confirm that they exist?), and whether the entity is identifiable (can you find the jurisdiction the company claims to be registered in so I can check that it really exists?).

The DAO

The **Decentralized Autonomous Organization** (**DAO**) was an attempt at an unbanked entity, though not unincorporated. It was organized as a Swiss SARL, and raised approximately 11.5 million **Ethereum** (**ETH**). The DAO was to be a fund that ran on smart contracts, and would make loans to entities that would use the funds in productive ways.

The reasoning behind the DAO was that investors would vote on all decisions, and with governance done entirely with smart contracts for the benefit of investors, it would defeat any perverse incentive that a director or manager might have. All decisions would be in the interest of investors.

A series of vulnerabilities in one of the DAO's smart contracts allowed malicious hackers to steal $50 million in ETH, or a third of the DAO's funds.

The DAO was delisted from major exchanges by the end of 2016, and that was the end of it. In order to recover the funds, the Ethereum blockchain went through a controversial hard fork.

A faction of the Ethereum community was opposed to refunding the funds from the DAO, claiming that a hard fork violated the principle of immutability and the notion that *code is law*—that users should be made to abide by. This led to the creation of Ethereum Classic.

About a year after delisting, the SEC stated that the raise was probably in violation of US securities law, although it was declining to take action against the DAO. No one suffered criminal penalties.

This was likely the first instance in which professional managers were removed from the process of running an organization. Disintermediating corporate boards and directors and allowing stakeholders to have a say directly seems to be a good idea, at least on the face of it. Blockchain provides a variety of consensus mechanisms, such as proof of stake and delegated proof of stake (discussed in greater detail in the previous chapter), that are natural fits for this sort of problem, as well as the limitless possibilities allowed by smart contracts.

Decentralizing an organization

The possibility of running an organization by aid of a smart contract remains. It is likely that the organization would still need one or more directors to carry out the directives of users and otherwise carry out the duties of running an organization. The DAO claimed zero employees, and that may have been true to a point. Given that it's effectively gone, it actually is zero now.

For a decentralized asset organization to be legal, a good start would be for it to have a legal entity, as suggested previously—something to provide limited liability to investors. For it to raise money, it would be a good start for it to admit that it's a securities offering, instead of claiming to be a utility token. The offering would then have to register as a security with the appropriate regulatory authority, or file for an exemption.

Much of the legal trouble with offerings such as the DAO perhaps could have been avoided by adhering to forms—there's nothing inherently wrong with a token as a security, except that to start, we should probably admit that most of the ones that people are buying speculatively are, in fact, securities.

Doing an issue under Rule 506(c) in the US allows unlimited fundraising among accredited investors, and general solicitation is allowed—you can advertise your offering. If you want to raise money among investors other than accredited investors, you can do that with a different exemption, the Reg A+, albeit with certain limitations and substantially more cost.

The core concept of users voting on corporate governance is a good one. The notion that organizations such as the DAO formed legal entities only as compatibility measures with real life speaks to the enormous arrogance of certain elements of the crypto community.

Corporations exist for the benefit of shareholders. In practice, layers of management and pervasive rent-seeking suck tons of productivity out of organizations. The thinking that eliminating people from an organization would fix its problems, however, is fundamentally flawed.

Algorithms are written by people. Smart contracts are written by people. True, an organization with zero employees has no payroll expense but if it's replaced by $50 million in oops-our-money-was-stolen expense, it's hard to say that that's an improvement.

Still, the basic concept of an organization truly run by stakeholders, where every deal is vetted by the people whose capital is supposed to be grown and protected—that really may work in some instances. Much has been written about the popular wisdom averaging out to be quite good.

In much the same way that actively managed mutual funds, net of all fees, tend to underperform index funds, professional managers of a fund that invests in a given asset might not earn their keep. A crypto real estate fund, in which the investors are primarily seasoned real estate investors who had previously been in syndication deals and so forth, would likely yield good results.

The value of blockchain in this instance is that stakeholders, running specialized software or clicking around a web interface, have a fast, consistent, verifiable way to participate in board elections or run the entire organization themselves.

Putting a corporation on a blockchain

We've discussed blockchain as a mechanism for stakeholders to have greater control over corporate governance. What could shareholders control?

Men like Benjamin Graham delighted in taking over corporations that didn't perform and forcing them to pay dividends. This is an example of shareholder activism. The corporation exists for your benefit—if it's not working for you, make it work for you.

Running a big cash surplus? Make it pay cash dividends.

Executive compensation out of control? Cut compensation, renegotiate, and fire, as the law allows.

Your corporation is a Radio Shack or a Blockbuster, woefully under-equipped to compete today? Liquidate it. Turn it into Ether.

Corporations with activist shareholders tend to outperform those that do not have them. If GE had had activist shareholders with good visibility into company finances, the CEO would not have had an opportunity to have an extra private jet follow his private jet in case his private jet broke down. (Yes, seriously.)

Institutional inertia and the slow death of capital seems less of a risk in an environment in which the shareholders have the option of pulling the entity off of life support at a moment's notice.

Cutting out the middle man

Shareholder activism is widely assumed to improve returns, and some evidence exists for this—but what if you have a highly technical business model and don't expect your investors to understand?

Suppose your business involves mortgage-backed securities, distressed debt, swaps, or other contrivances? Some of these involve fiendishly complicated financial models, and there's a real risk that the shareholders (or token holders) won't understand. What does that look like in this brisk, brutal environment? Could the stakeholders kill the entity prematurely?

Fortunately, the tokens need not be shares. No technical barrier exists that would prevent the tokens from being other securities.

A token could be a revshare (a form of debt that represents a share of the entity's revenue until some multiple of the principal is paid back), a note, a bond, or another interest-yielding instrument. The interest could be paid based on a price in fiat or a fixed rate in Ether or another common, fungible cryptocurrency.

It could even represent a share of revenue or profits from a specific asset, a future derivative providing a return from an underlying crypto portfolio, a hard asset portfolio, rents or royalties. At the time of writing, a firm called **Swarmsales** is currently preparing to create an asset that would take a percentage of the income from the sales of thousands of units of software sales by a large and growing decentralized, freelance sales staff. The expectation is that paying that staff with this instrument will give sales professionals a new source of income, replacing their base salaries and providing a source of income extending even beyond their term of employment. Swarmsales expects to be able to use this approach to create the largest B2B salesforce in the world.

This is what blockchain makes possible. Raises that would involve securities transfer agents, endless meetings, and very specialized, expensive staff can now be accomplished on the internet with a modest amount of technical and legal work and marketing budgets that would be laughable on Wall Street.

Providing capital

There remains a perception that **Initial Coin Offerings** (**ICOs**), and increasingly **Security Token Offerings** (**STOs**), are a simple, low-cost way to raise money. This may be true for the right team, but budgets have greatly grown. One project, called **Financecoin**, simply copy-pasted the README file from DogeCoin on BitcoinTalk and succeeded in raising over $4 million. Whether or not they did anything useful with the money is not known to the author, invoking again another problem with the space.

Those days are gone—it's now widely agreed that it costs between $200,000 and $500,000 to do an ICO or STO in this increasingly crowded and regulated space. It's not clear, though, that many of the good projects would have had a problem raising capital anyway.

The great majority of token sales are to support *très chic* tech projects, some of which would be as well suited to conventional fundraising. It's impossible to do an **initial public offering** (**IPO**) or reg A+ without the assistance of an investment bank or brokerage, and those firms largely herd towards tech and healthcare. It's safe to say that blockchain hasn't improved access to capital for projects with strong teams with good concepts and previous startups under their belts.

Certainly DAOs are largely not of interest to investment banks; those remain solely within the purview of distributed teams doing ICOs or STOs.

Some projects with a social purpose have succeeded in raising money via conventional channels—Lemonade, an unconventional insurer organized as a New York B Corp, closed a very successful Series C last year. Benefit corporations seem to be the new darlings of mainstream venture capital firms, such as Andreessen Horowitz and Founders Fund.

This seems to be a good fit for blockchain as well—benefit corporations and charities similarly need cash but may not be suited to the multiple rounds of fundraising and eventual exit required of them by VC. Fidelity Charitable collected $22 million in Bitcoin and other cryptocurrencies in 2017, and that segment of charitable contributions is expected to continue to grow.

Social purpose – blockchain as leveler

As a means of raising capital, blockchain has the potential to improve environmental impacts and long-standing disparities between the haves and the have-nots. An Australian firm called **African Palm Corp** is working towards launching a cryptocurrency with each unit backed by a metric ton of sustainable-source palm oil.

As this book goes to publication, the EU is preparing to ban the import of Indonesian and Malaysian palm oil—85% of the current supply. The means of farming palm oil causes habitat destruction on a massive scale. Conversely, this STO aims to purchase four million hectares of land on which palm trees already stand, potentially reducing unemployment in four West African countries on a massive scale.

Another project headed by the founder of a Haitian orphanage has the potential to improve tax collection and provide physical security for a great number of vulnerable people. The Caribean Crypto Commission is a project to create a coin employed by a sort of cryptoentity; governments would allow the formation of a crypto corporation with a greatly reduced tax rate in exchange for that entity having public financials. Recording that business' transactions publicly on a blockchain would allow the company to create verifiable balance sheets and income statements. An entity subject to a 15% tax rate would be an improvement over an entity that nominally has a 40% tax rate (and which, in practice, pays nothing).

This concept also provides physical security to people living in temporary housing in earthquake-rattled Haiti, where banks are expensive and cash is kept in tents and houses—putting the occupants in danger of robbery. The alternative? An SMS wallet on your smartphone. Via a gateway, a program on your phone can send and receive text messages that can credit an offline wallet, even without mobile data.

Banking the unbanked

A lack of access to lending and banking lead people to use predatory lenders. Payday loans promise to provide money to address problems needing attention in the immediate term—an impounded car, a bond for release from jail, a mortgage near to foreclosure.

The loans are expensive to the borrower, and to the lender too—it's thought that 25% of principal is lost to default. The right to collect on those defaulted loans is then sold to collection agencies, which enjoy a largely deserved reputation for being nasty and abusive.

Loans paid back on time are likewise not subjected to treatment that could be considered fair or equitable. Rates of interest range from 50% to over 3000% annualized, with some lenders treating each payment as a refinance and writing a new loan, allowing them to again to charge the maximum rate of interest.

Some are said to be altogether unbanked—aside from lending, the fractional reserve model of banking provides the option for the lender to avoid charging fees for some services. People without checking accounts because of prior overdrafts, geographic isolation, or high costs are cut off from this system, and generally have to resort to using check-cashing services and prepaid debit cards. This is another product that is not cheap, with services charging between one and four percent—a hefty sum if a worker is forced to cash his paycheck this way. For the poorest of the poor, after all credits, it is not unusual to pay more in check-cashing fees than in income tax, however hard those same people may rail against taxes.

Some firms are already using alternatives to traditional banking, for better or worse. It is now regrettably not unusual for firms to pitch prepaid debit cards to their employees as a convenient alternative to banking, at substantially greater cost for lesser service.

Another alternative, however, exists in the form of various electronic tenders. DHL now pays its contractors in Kenya with Safaricom, M-PESA, a sort of mobile phone wallet pegged to the Kenyan shilling. This represents a huge leap forward in the case of people for whom the nearest bank may be 50 or 60 miles away.

This is less security than having cryptocurrency—the properties of cryptocurrency, specifically decentralization, ensures that there is no central authority easily able to seize or revoke ownership of funds. This is not the case with what is essentially a bank account used exclusively through an app.

Likewise, a very large part of the promise of cryptocurrency is cash that is free from such specters as asset forfeiture, in which law enforcement agencies may seize property in so called **rem proceedings**, in which the onus is on the owner of the property to show that the ownership thereof is legitimate. Blockchain provides not only a means by which ownership can be shown, but also that property's chain of title—stretching back, in most instances, to the date at which the asset was created.

Silk road LLC – privacy and ethics

The curious contradiction of an emerging technology is that it's simultaneously lauded and condemned. Possibilities good and bad are considered and weighted.

Probably we tend a bit too much toward moralism—almost everyone interested in blockchain has been told that it's just for drugs. There's something to this: many people first heard of Bitcoin as something used on dark net markets, such as the Silk Road, and this use of Bitcoin is arguably responsible for the popularity of cryptocurrency. The same people might then be surprised that you have, in fact, been receiving your salary in Bitcoin for six or seven months.

Each evangelist has to genuinely consider the drawbacks—in Bitcoin, for instance, privacy remains a large problem. As of this writing, this seems very unlikely to change. Take any idea that would do so much for transparency—say, a crypto corporation, with its transactions, or at least its balance sheet, on the record, available for all to see. The idea is conceptually sound, and it would improve tax collection and compliance.

Employees who were previously off the books might enjoy social security credit. Wage parity between people of different racial and ethnic groups might be achieved. Organizations that serve as fronts for money laundering would have to do so under greater scrutiny. Financial statement fraud would be easily detectable. Insurance rates would plummet—or at least my errors and omissions insurance would get cheaper.

All of this sounds pretty great until you consider the obvious drawbacks of everyone being in everyone else's business, and one hopes that the people building and purveying any technology, not just blockchain, stop and carefully consider the value of privacy.

Take another example: electric smart meters offer improvements in efficiency to utility companies. It eliminates obscene amounts of human labor. Rather than having a person come by to read the meter on every single house regularly, the meter transmits data on power usage on a regular basis. In theory, this would also provide benefit to the consumer: unplug your refrigerator for 20 minutes and see how much juice it's using. Less directly (and perhaps out of naïveté), we would also assume that the utility's savings are passed on to the consumer in the form of lower power tariffs.

So, where's the downside? Some of the least sophisticated of these meters could identify, say, the use of a microwave. A sudden increase in power usage in exactly the amount of 1000W for about a minute, for the sake of argument. If you find that a given household uses 1000W right around sundown every day during the month of Ramadan, you've probably identified a Muslim family observing the ancient tradition of fasting. If you find that from Friday night to Saturday night there's no power usage at all, perhaps you've identified an observant Orthodox Jew.

However benevolent an authority may seem now, information tends to stick around, and it is terribly empowering to people with ill intent. Consider that the Holocaust was especially bad in the Netherlands, but killed only a single family in Albania. The former country was great at keeping records. The latter, terrible.

Tracking all the things

Blockchain as a solution for social problems offers promise. Everything from preventing blood diamonds from entering circulation to guaranteeing a minimum income could be implemented with blockchain. Any tool has the potential for abuse, and this is no different.

Consider the pitch—put a tiny chip in every diamond. It has a little chip with a revocable private key on it, something akin to the EMV chip on a credit card. In theory, this could be used to ensure that each diamond in your rings is not a so-called **conflict diamond**, one sold to fund a war.

It also may have the effect of freezing everyone else out of the market. Jewelers getting stones certified through organizations such as the **Gemological Institute of America** (**GIA**) now have to sign a declaration that they don't knowingly sell blood diamonds, as well as submit personal information related to anti-money laundering regulations.

Purveyors of diamonds not wanting to adhere to this scheme largely don't have to, at the expense of being consigned to the rubbish heap—selling exclusively uncharted diamonds implies a proprietor offering a product of lesser quality. Whatever the expense, businesses go to great lengths to appear legitimate, and it's unlikely that even the savviest of consumers will care much for the proprietor's privacy.

Defeating some privacy issues with zero-knowledge proofs

Solutions that respect human limitations, with regard to knowledge or moral character, are more ethical and more robust than the sort of hand-wave everything will be fine attitude that pervades the tech world. It seems far safer to acknowledge that, although some people will abuse the anonymity, cash and barter transactions have taken place for thousands of years without any sort of central authority regulating them, and people often have legitimate reasons for wanting privacy or even anonymity.

Zero-knowledge cryptosystems may offer a solution in some use cases—cryptocurrencies such as Zcash and Monero more truly resemble cash. Bitcoin and (presently) Ethereum do not offer the sort of anonymity and privacy that detractors would have you believe they do.

Blockchain inherently has many desirable properties for people seeking privacy and anonymity, decentralization being the foremost of those. More broadly, outside of blockchain, this author suspects that users will be willing to pay somewhat of a premium for technology that ensures ephemerality, the property of being transitory, temporary, perhaps forgettable.

We've entrusted the most intimate details of our personal lives to mechanisms designed by fallible human beings, and despite the constant data breaches and the occasional happening, the public largely trusts things such as Facebook chats and devices with always-on microphones. However zealous blockchain evangelists may seem, trust in companies with centralized IoT and mobile junk remains the most remarkable arrogance.

Zero-knowledge proofs (and zero-knowledge arguments) allow the network to verify some computation (for example, ownership of tokens) without knowing anything about it. In addition to the properties for which we use blockchain (blockchains are consistent, immutable, and decentralized), specific implementations of zero knowledge may also be deniable, and avoid recording information on a blockchain that may compromise a user's privacy.

Unwrapping the concept of zero-knowledge proofs

Conceptually, a zero-knowledge proof is similar to a randomized response study. Researchers are understandably concerned about whether people will honestly answer a question about a taboo behavior—such as drug use or interactions with sex workers.

In order to eliminate bias, statisticians came up with a method that introduced randomness in individual responses while keeping the meaning of overall results. Imagine you're trying to determine the prevalence of abortion by interviewing women, in a jurisdiction in which abortion is illegal. Have the interviewee flip a coin. If heads, answer the question honestly. If tails, just say Yes.

The researcher doesn't need to know the results of the coin flip, or each individual's true response—they only need need to know that, given a sufficient sample size, taking the margin above 50% and doubling it gives you the actual prevalence of the practice. This approach preserves the privacy of the individual respondents while not compromising the quality of the data.

Zero-knowledge proofs (and arguments) are highly technical and the nuts and bolts are beyond the scope of this publication, but they are conceptually similar to what we're discussing. Depending on the specific implementation, zero knowledge might allow a user to spend the contents of their wallet without other users of the network knowing the contents of the wallet. This is one respect in which a cryptocurrency can truly be similar to cash: short of a mugger taking your wallet, there's no way for another person to know what you have until you've revealed it. This approach succeeds in addressing one of the largest problems in cryptocurrencies such as Bitcoin, and at the time of writing, Ethereum.

Summary

Blockchain is a transformational technology with an impact similar to that of the internet, vaccinations, or powered flight: the social implications are extensive, subtle, and perhaps in some ways pernicious. It has the potential to further degrade privacy, drastically improve corporate governance, or lift billions out of poverty. The specific application of this technology will define it as a tool that will help to shape the world.

In the next chapter, we will discuss some applications of blockchain, starting with the most fundamental of them, cryptocurrency wallets.

9
Cryptocurrency Wallets

In this chapter, we will discuss cryptocurrency wallets in details. In earlier chapters, we have been introduced to wallets as well as the types of crypto wallets; in this chapter, we will further discuss wallets in detail, their source, and how the security of wallets can be strengthened.

Wallets are used to store private and public keys along with Bitcoin addresses. Coins can be sent or received using wallets. Wallets can store data either in databases or in structured files. For example, the Bitcoin core client wallets use the Berkeley DB file.

The topics we will cover in this chapter are as follows:

- Importance of cryptocurrency wallets
- Software wallets
- Hardware wallets
- Paper wallets
- Brain wallets
- Key derivation methods in a cryptocurrency wallet
- Mnemonic codes

Introduction to cryptocurrency wallets

A wallet of any cryptocurrency can store multiple public and private keys. The cryptocurrency is itself not contained in the wallet; instead, the cryptocurrency is de-centrally stored and maintained in the public ledger. Every cryptocurrency has a private key with which it is possible to write in the public ledger, which makes spending in the cryptocurrency possible.

It is important to know about the wallets, since keeping the private key secure is crucial in order to keep the currency safe. Wallet is a collection of public and private keys, and each of them is important for the security and anonymity of the currency holder.

Transactions in cryptocurrency wallets

Transactions between wallets are not a transfer of value; instead, the wallets store the private key in them, which is used to sign a transaction. The signature of a transaction is generated from the combination of private and public keys. It is important to store the private key securely.

Wallets can store multiple private keys and also generate multiple public keys associated with the private key.

Hence it is important to keep the wallet secure so that the private key is safe; if the private key is lost the coins associated with that private key are lost forever, with no feasible way to recover the coins.

Types of cryptocurrency wallets

Wallets can be categorized in various types based on their traits. Wallets can be categorized by amount of currencies they supports, availability, software or hardware, key derivation method, and so on. We will look at the types of cryptocurrency wallets to be covered in the following subsections.

Currency support

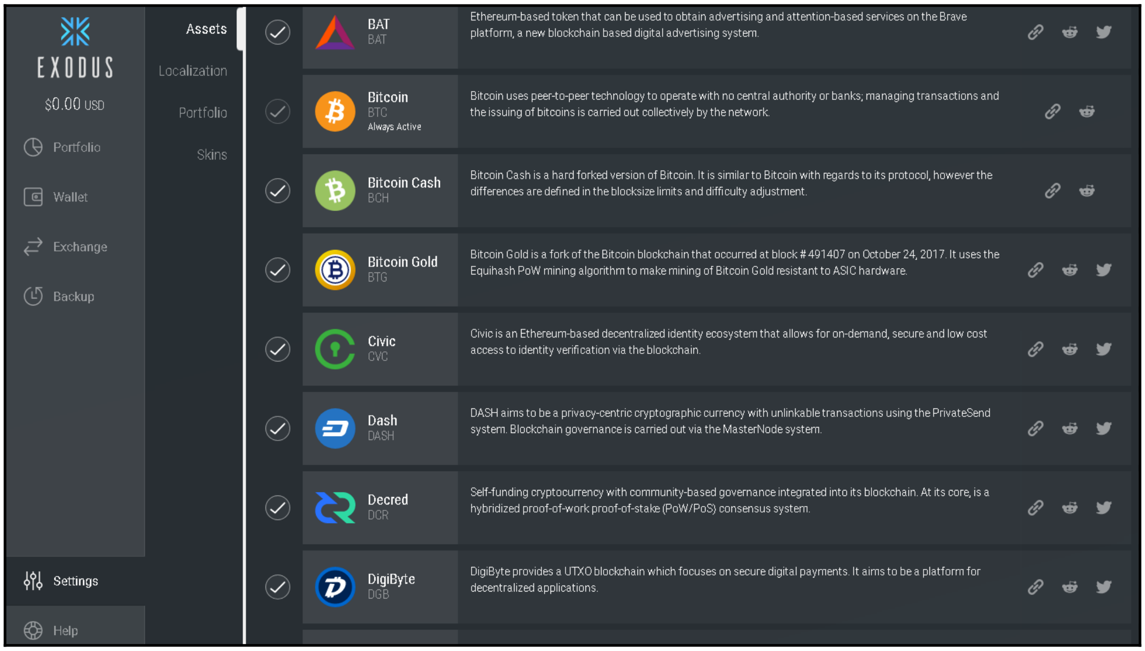
One primary trait on which the wallets can be distinguished is the number of currencies the wallets support; for example, there can be wallets with single currency support or multiple currency support. Each of the coins has a core client which includes a wallet too.

The official wallets usually support single currencies, but nowadays, lot of third-party wallets have appeared support multiple wallets, these wallets performing the same functions as regular wallets, the only difference being the number of currencies they support.

Some of the wallets that support multiple currencies are as follows:

- Infinite wallet
- Jaxx wallet
- Electrum
- Trezor
- Coinbase wallet

The following is a screenshot of the **EXODUS** wallet, which supports various currencies:



BAT BAT — Ethereum-based token that can be used to obtain advertising and attention-based services on the Brave platform, a new blockchain based digital advertising system.

Bitcoin BTC Always Active — Bitcoin uses peer-to-peer technology to operate with no central authority or banks; managing transactions and the issuing of bitcoins is carried out collectively by the network.

Bitcoin Cash BCH — Bitcoin Cash is a hard forked version of Bitcoin. It is similar to Bitcoin with regards to its protocol, however the differences are defined in the blocksize limits and difficulty adjustment.

Bitcoin Gold BTG — Bitcoin Gold is a fork of the Bitcoin blockchain that occurred at block #491407 on October 24, 2017. It uses the Equihash PoW mining algorithm to make mining of Bitcoin Gold resistant to ASIC hardware.

Civic CVC — Civic is an Ethereum-based decentralized identity ecosystem that allows for on-demand, secure and low cost access to identity verification via the blockchain.

Dash DASH — DASH aims to be a privacy-centric cryptographic currency with unlinkable transactions using the PrivateSend system. Blockchain governance is carried out via the MasterNode system.

Decred DCR — Self-funding cryptocurrency with community-based governance integrated into its blockchain. At its core, is a hybridized proof-of-work proof-of-stake (PoW/PoS) consensus system.

DigiByte DGB — DigiByte provides a UTXO blockchain which focuses on secure digital payments. It aims to be a platform for decentralized applications.

There are various other wallets coming up from time to time that offer multiple currencies. Sometimes, existing wallets that support a single currency start to introduce multiple currencies to increase their popularity or even to support another cryptocurrency.

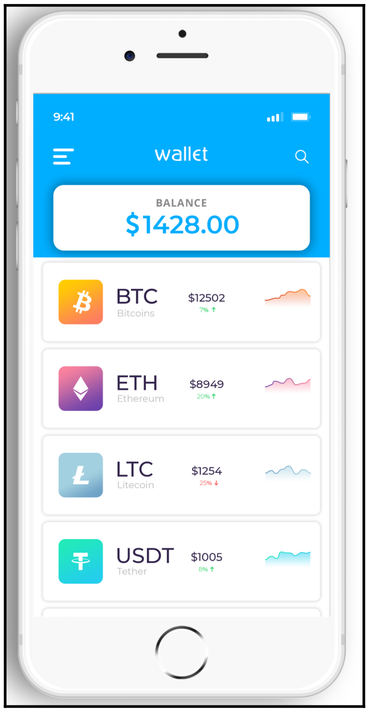
Tenancy

The wallets can also be differentiated based on whether they are software, hardware, paper, or cloud based. Let's discuss each of these wallets in detail.

Software wallets

These wallets are based on the local computer or mobile. These can be further divided into desktop or mobile wallets. These wallets are confined within the local machine; they usually download the complete blockchain in order or keep record of the public ledger:

- **Desktop software**: They store the keys locally on the desktop or laptop. It offers complete control over the cryptocurrency, although responsibility for security rests with the person hosting the wallet on their machine, so, even if the hard drive of the machine fails without any backup being in one place, there is a chance of loosing the coins forever. It is not necessary to have the blockchain updated at every moment, hence, a machine not connected to a computer can also be used in such cases.
- **Mobile wallets**: These wallets run via an app on the mobile phone. This allows users to easily access their coins. These wallets store the coins or keys locally on the mobile phone just like a desktop software wallet, or they are just an interface to a cloud wallet;, we will discuss that type of wallet later. The following screenshot is an example of a mobile wallet:



The core clients of most of the cryptocurrencies offer software wallets to start with, with initial support for desktop wallets.

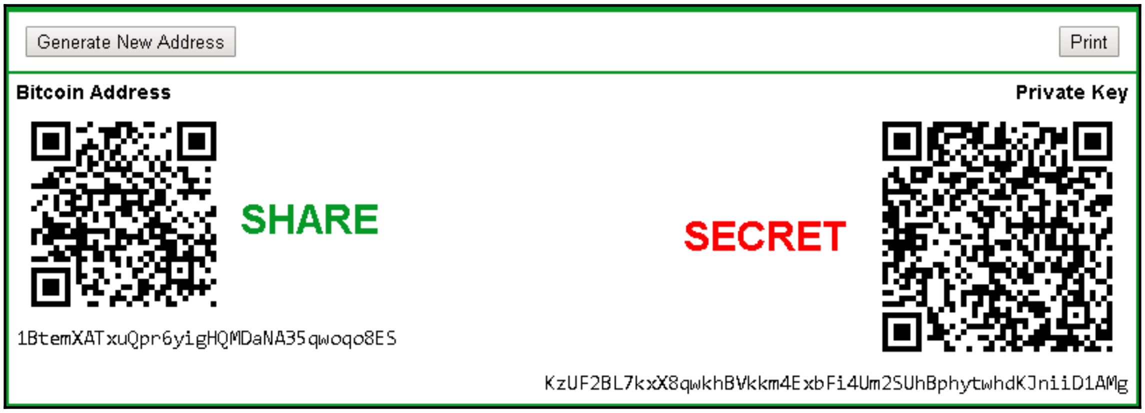
Hardware wallets

The private key is to be stored in the most secure location possible. Hardware wallets store the private key in a custom hardware designed to store private keys. It is not possible to export the private key in plain text, which gives it another layer of security. Hardware is connected to a computer only when required, and at all other times the private key is kept secure. Hardware wallets were first introduced in 2012 by Trezor.

Some of the popular hardware wallets currently available are Trezor, Ledger, and KeepKey wallets. The following snap shows an example of a hardware wallet connected to a computer:

Paper wallets

Paper wallets, as the name suggests, are simply public and private keys printed together. The keys are usually printed in QR form, also serving as the address. Anybody can create a paper wallet by printing the keys but also making sure they remove the keys from the computer, or anyone could have access to the keys. Paper wallets are meant to be stored only on paper with no backup elsewhere. There are various online services that generate paper wallets such as `www.walletgenerator.net`. The following screenshot is an example of a paper wallet, the following Image can be printed, to receive payments, public address is shared but the private key marked secret and is to be kept safe:

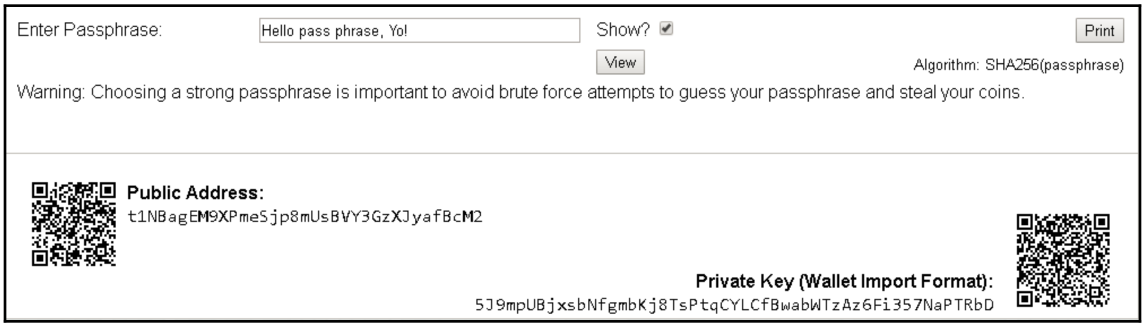


1BtemXATxuQpr6yigHQMDaNA35qwoqo8ES

KzUF2BL7kxX8qwkhBVkkm4ExbFi4Um2SUhBphytwhdKJniiD1AMg

Brain wallet

A brain wallet is a simple wallet that creates addresses by hashing passphrases to generate private and public keys. To generate a brain wallet, we choose a simple passphrase to generate a public and private key pair. The following screenshot shows how the **Public Address** and **Private Key** are generated. The passphrase is entered which is to be reminded as shown in the following screenshot:



Enter Passphrase: Hello pass phrase, Yo! Show? ✓

Algorithm: SHA256(passphrase)

Warning: Choosing a strong passphrase is important to avoid brute force attempts to guess your passphrase and steal your coins.

Public Address:
t1NBagEM9XPmeSjp8mUsBVY3GzXJyafBcM2

Private Key (Wallet Import Format):
5J9mpUBjxsbNfgmbKj8TsPtqCYLCfBwabWTzAz6Fi357NaPTRbD

Usage frequency

The wallets can also be differentiated on the basis of usage; there are primarily two types of wallets on this basis: cold wallet and hot wallet. In simple terms, cold wallets are not connected to the internet while hot wallets are connected to the internet, at all times and can be used for sending the respective cryptocurrency at any time. Cold wallets can be used to receive the currency even when not connected to the internet but it is not possible to send the currency to other address before connecting it to internet.

A hardware wallet is not connected to the internet unless plugged-in to a device; they can be considered cold wallets.

Key derivation

The private keys are generated by a wallet to be present on the blockchain, and there are primarily two methods by which the key can be generated. The key generation method is crucial for the security of the wallet and also important for recovery of the wallet in case the wallet is lost.

Non-deterministic wallet

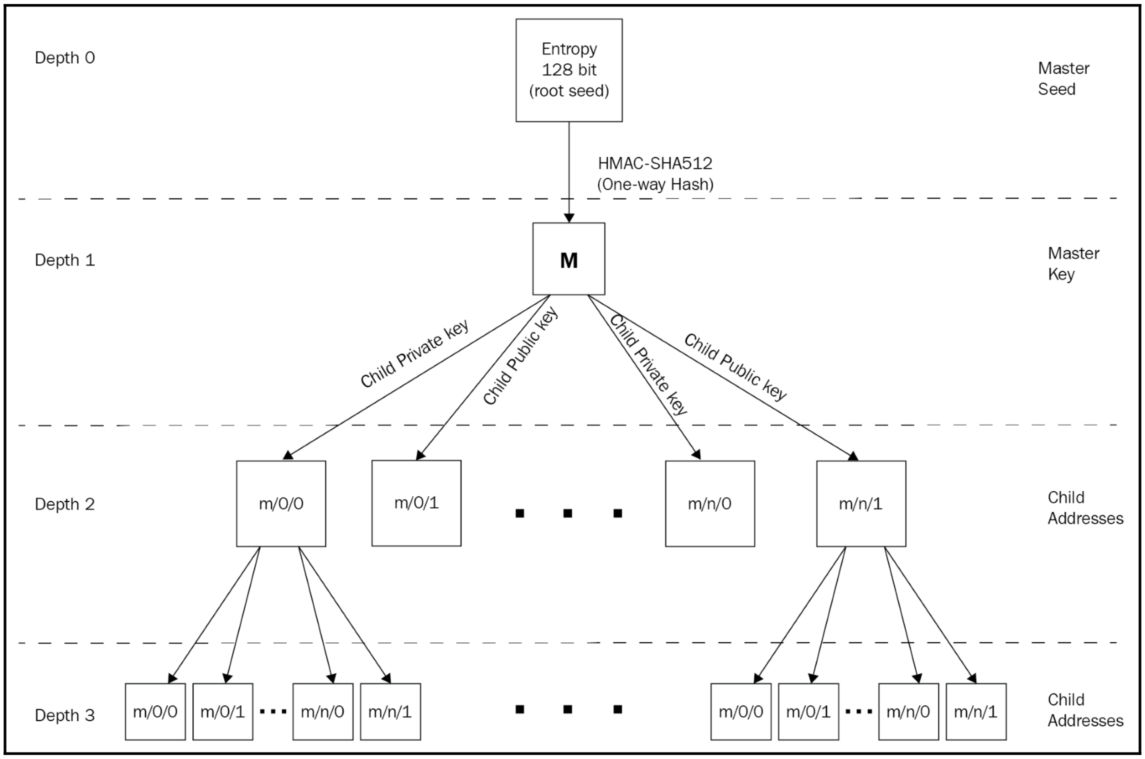
These were the initial iterations of the Bitcoin clients; the wallets had randomly generated private keys. This type of wallet is being discontinued due to a major disadvantage, that the random keys are inaccessible if the wallet is lost. Since it is advisable to use a different address for every transaction in order to maintain anonymity on the network, with so many random keys, it becomes difficult to maintain and, hence, addresses became prone to re-use. Even though in the Bitcoin core-client, there is a wallet that is implemented as a type-0 wallet, its use is widely discouraged.

Deterministic wallets

In these wallets, the keys are derived from a single master key or one can say a common seed. All the private keys in this type of wallet are linked to a common seed. Backup of only the seed is sufficient to recover all the keys derived in this type of wallet.

Hierarchical deterministic wallets

This is the most advanced form of deterministic wallets. These were introduced in the BIP0032 of the Bitcoin Improvement Proposal system. These wallets follow a tree structure, that is, the seed creates a master key, which further creates child keys, and each of the keys can derive further grandchildren keys. Thus, in these types of wallets, there can be multiple branches of keys, and each key in the branch is capable of creating more keys as required. The following diagram shows the keys and the hierarchy of addresses created in such wallets:



Mnemonic codes

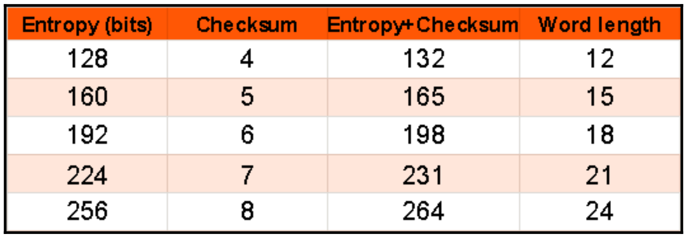
These are English words used to represent the random number used to derive the seed in a deterministic wallet. The words act as a password; the words can help in recovering the seed and subsequently the keys derived from it. The mnemonic codes act as a good backup system for the wallet user.

The wallet shows a list of 12 to 24 words when creating a wallet. This sequence of words is used to back up the wallet and recover all the keys in the event of a wallet becoming inaccessible.

Here is the process of generation of mnemonic code and seed as per the BIP0039 standard:

1. Initial random **Entropy** of ENT bits are generated between the allowed size of 128-256 bits.
2. Checksum is generated by taking the first few bits of its SHA256 hash. The checksum length is defined by `ENT/32` formula.
3. The checksum denoted by `CS` is added at the end of initial **Entropy**.
4. The sequence created is split into 11-bits; each is encoded a number between 0 and 2,047 which acts as an index to a pre-defined wordlist.
5. A list of 12-24 words is created representing the mnemonic code.

The length mnemonic code, also known as a mnemonic sentence (`MS`), is defined by `MS = (ENT + CS) / 11`. The following screenshot shows the word length and the **Entropy** associated with that word length:



Entropy (bits)	Checksum	Entropy+Checksum	Word length
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

The seed that is 512 bits is generated from the mnemonic sequence using the `PBKDF2` function where the mnemonic sentence is used as the password and the *string mnemonic* + *passphrase* is used as a salt. The passphrase is something that a user can use to protect their mnemonic; if it's not set, then `""` is used.

The length of the derived key from this process is 512-bits; different wallets can use their own process to create the wordlist and also have any desired wordlist. Although it is advised to use the mnemonic generation process specified in the BIP, wallets can use their own version of wordlist as they require.

Key generation process in HD wallets

We will be discussing key generation processes in detail, starting from master keys through to private keys, and the various addresses a wallet creates for transaction purposes.

The initial process is to create the root seed which is a 128, 256, or 512 bit random. The root seed is represented by the mnemonic sentence, which makes it easier to restore the complete wallet in the case of losing access to the wallet.

Root seed is generated using the mnemonic sentences and the root seed which is of a chosen length between 128 and 512-bits, although 256 bits is advised. Generated using (P) RNG. The resultant hash is used to create the master private key and the master node. Generation of the master key is the depth level 0 in the hierarchy system; subsequent wallets or keys are denoted by depth 1, 2, and so on.

Child key derivation

HD wallets extensively use the **Child Key derivation** (**CKD**) function to create child keys from the parent keys. Keys are derived using a one-way hash function using the following elements:

- Parent key
- Chain code which works as a seed, 256-bits
- Index number, 32-bits

There are various ways in which child keys can be generated from already present keys; the following are the key derivation sequences:

- Generation of private child key from private parent key
- Generation of public child key from public parent key
- Generation of public child key from private parent key

Let's discuss each of the previously mentioned sequences in detail.

Private key generation

The parent key, chain code, and index number are combined and hashed with the `HMAC-SHA512` algorithm to produce a 512-bit hash using the following formula:

$$I = \text{HMAC-SHA512}(\text{Key} = C_{par}, \text{Data} = ser_p(\text{point}(k_{par})) \,||\, ser_{32}(i))$$

The resultant hash is split in two hashes, I_L and I_R The right-hand half of the hash output becomes the chain code for the child, and the left-hand half of the hash and its index number are used to produce the child private key as well as to derive the child public key.

By changing the index, we can create multiple child keys in the sequence.

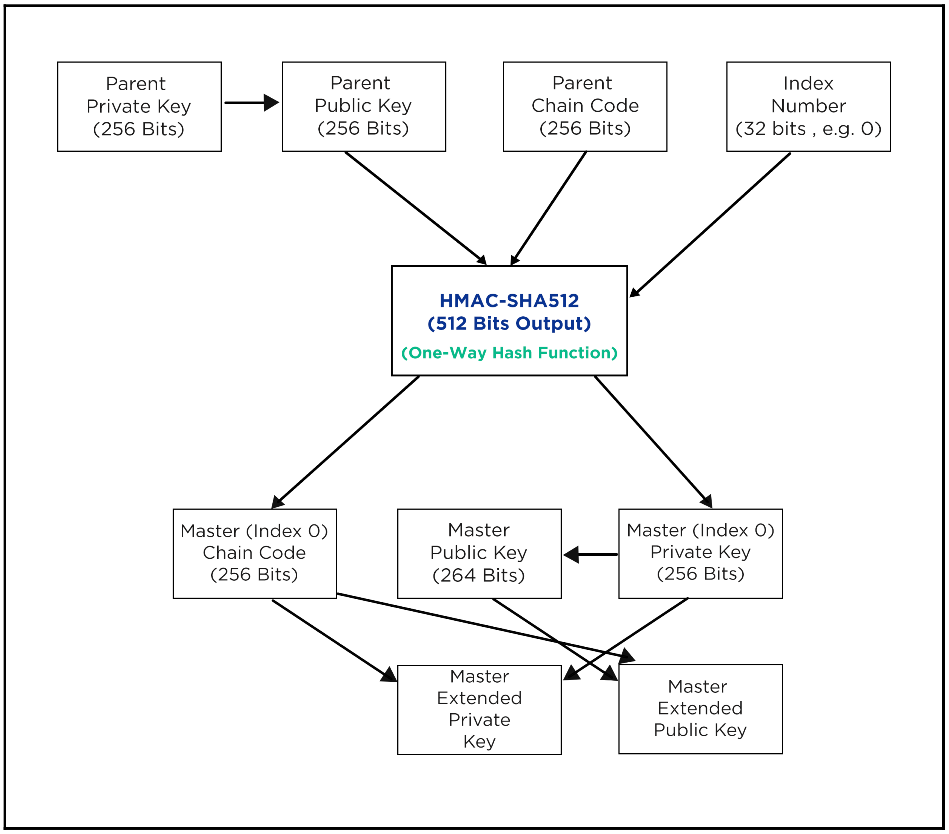
Extended keys

As discussed a number of child keys can be derived from parent key considering the three required inputs are available. We can also create another type of key, called the **extended key**, which consists of the parent key and the chain code.

Furthermore, there are two types of extended keys, distinguished by whether the parent key used is a private key or a public key. An extended key can create children which can further create children in the tree structure.

Extended keys are encoded using `Base58Check`, which helps to easily export and import between wallets. These keys are basically extensions of the parent keys, hence sharing any extended key gives access to entire branch in the branch.

An extended private key has the `xprv` prefix in the key's `Base58Check`, while an extended public key has the `xpub` prefix in the key's `Base58Check`. The following diagram show's how extended keys are formed:


Parent Private Key (256 Bits)
Parent Public Key (256 Bits)
Parent Chain Code (256 Bits)
Index Number (32 bits , e.g. 0)
HMAC-SHA512 (512 Bits Output)
(One-Way Hash Function)
Master (Index 0) Chain Code (256 Bits)
Master Public Key (264 Bits)
Master (Index 0) Private Key (256 Bits)
Master Extended Private Key
Master Extended Public Key

Summary

In this chapter we discussed in detail cryptocurrency wallets and various types of crypto wallets, we read about various characteristics based on which the crypto wallets can be distinguished, and we talked about the tenancy of the wallets, benefits of each of the wallet types, and the issues one can face while using the specific wallet type. We discussed key derivation methods and its importance with regard to security, accessibility, and other aspects of a wallet.

10
Alternate Blockchains

In the previous chapters, we learned about blockchain, its structure, components, mechanism and the biggest use case of blockchain, Bitcoins. In the last chapter, we discussed cryptocurrency wallets, and their role and usage with a blockchain. Most of our discussion surrounds Bitcoins and other cryptocurrency.

The success of Bitcoin brought a lot of attention to the technology and the underlying blockchain ledger system, and the community started creating alternate cryptocurrency based on blockchain, making slight modifications to the parameters of Bitcoin, each time trying to improve in one way or another. Subsequently, various organizations started creating alternatives to blockchain by making slight modifications or changes but keeping the core definition of blockchain, that being a public ledger, intact. Some of the projects trying to create alternative blockchain did not gain much attention, but others managed to get a lot of attention and community support.

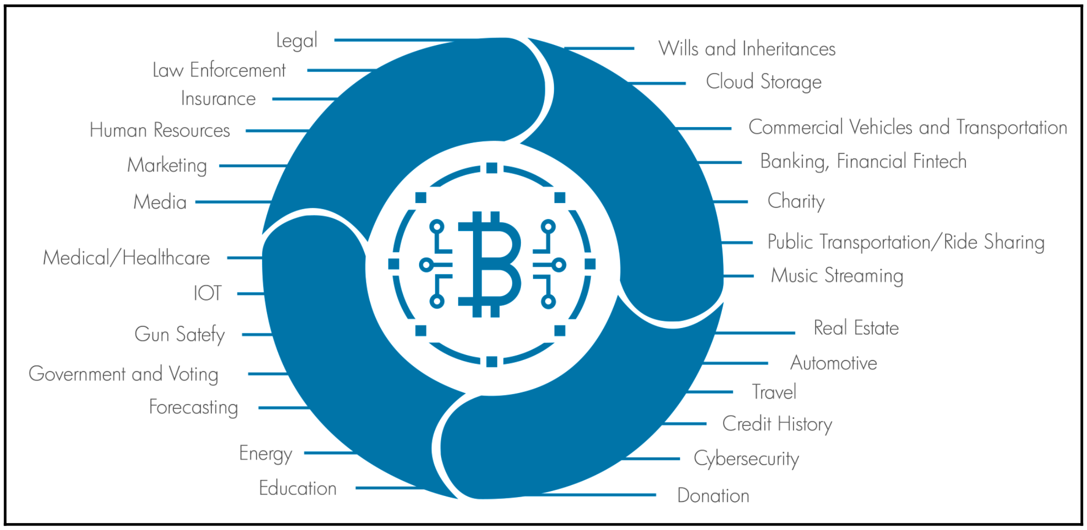
In this chapter, we will discuss the following topics:

- Uses of blockchain in various verticals such as government, healthcare, medical research, supply chain, fine art, shipping, energy, enterprise, and so on
- The Ripple payment system
- The Stellar payment network
- Tendermint
- Monax

Various uses of blockchain

Distributed ledger technology is said to be the biggest revolution in computers after the internet; blockchain is and will be revolutionizing and impacting on each individual in years to come.

Blockchain is used in currency-related applications such as Bitcoins and Altcoins, but, apart from that, there are various other use cases of blockchain in other industries that entail completely different monetary usage. The following diagram depicts some of the industries where blockchain is being used:



Government

Various governments across the globe are using blockchain to store public records or any other information across various government sectors, such as healthcare, identity management, taxation, voting, and financial services.

By having a decentralized database, it will make it easy for the governments to reduce fraud and also introduce certain checks before the data is entered into the distributed ledger system.

Healthcare

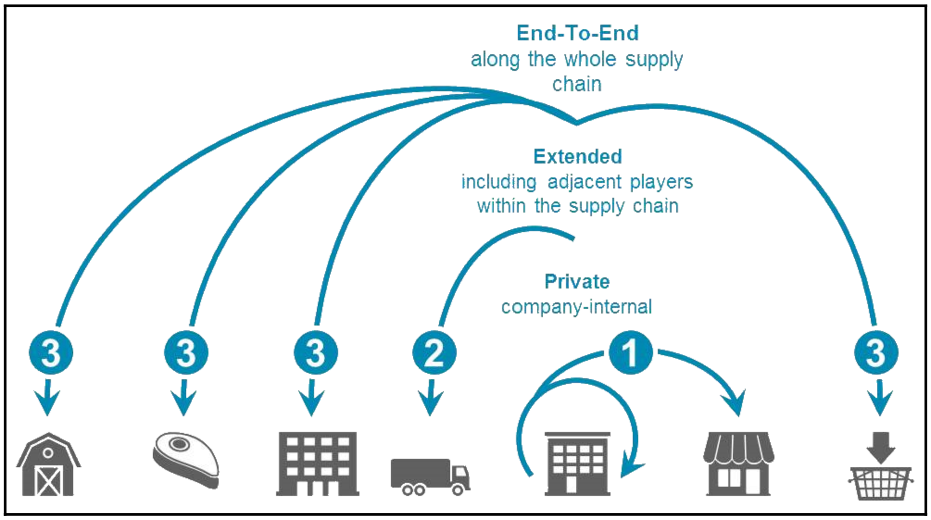
Medical records of an individual need authentication for correct information, and it is important to have access to health records that are complete in all aspects. Blockchain can be used in facilitating data sharing and record keeping. Sensitive medical data can become easily accessible to doctors and other relevant people of the healthcare community.

Medical research

Researchers in the medical community are always thriving to make better innovations and techniques that can improve clinic care. With data being present on blockchain, researchers can access the authentic data with ease and also add theories/results based on an apt approval cycle. The system's interoperability can help in multiple levels, along with offering precision and authenticity.

Supply chain

Supply chain management is one of the most scattered bottlenecks of a business process. There has always been a need for efficiency in supply chain management. There is a lack of compatibility because of the use of multiple software systems, with each one having various data points required for smoother movement. Blockchain can provide each participant in the supply chain process with access to relevant information reducing communication or data errors, as seen in the following diagram:


End-To-End
along the whole supply chain
Extended
including adjacent players within the supply chain
Private
company-internal
3
3
3
2
1
3

Copyright

A blockchain can be used to resolve copyright claims, since any entry in the blockchain-based system can only be introduced after it has been approved by the consensus system, thus making sure that the copyright is maintained.

Fine art

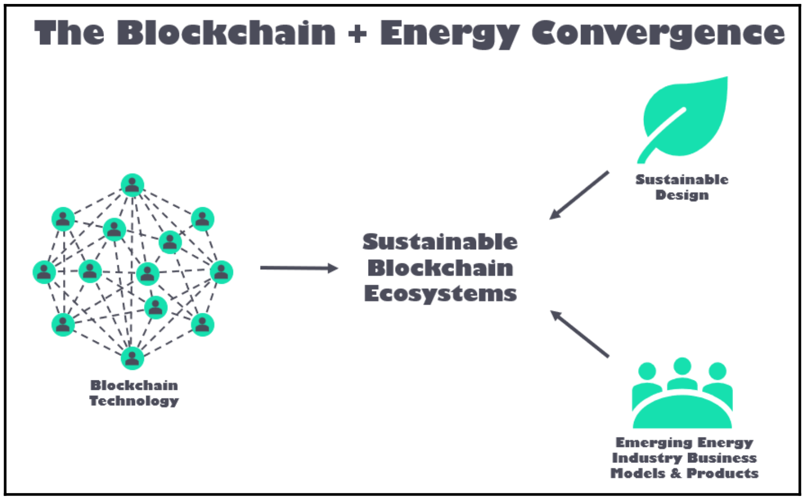
The art industry depends on the authentication of artwork; although blockchain cannot authenticate artwork or whether a painting is original or forged, it can be used to authenticate ownership.

Shipping

A number of projects have come up that are using blockchain within the maritime logistics industry to bring transparency in international trade. Various global shippers are using blockchain for the same reason, to introduce blockchain-based techniques and to weed out any bottlenecks that distributed ledger technology solved for the industry.

Energy

Blockchain can help to maximize efficiency in the energy distribution sector by keeping a track on energy allocation and implementing efficient distribution. Energy production and research for new sustainable resources can be monitored by using a blockchain to maintain authenticity and consensus in the same sense, as can be seen here:


The Blockchain + Energy Convergence
Blockchain Technology
Sustainable Blockchain Ecosystems
Sustainable Design
Emerging Energy Industry Business Models & Products

Computation and data storage

Computation resources get wasted around the globe. Data centers and data lakes are always in need of efficient data maintenance. Using a blockchain can ensure security and improvement.

Identification and social security

User identification is an important use case of blockchain being used in governments, but can also be used by other organizations for social security and other identification processes, as required.

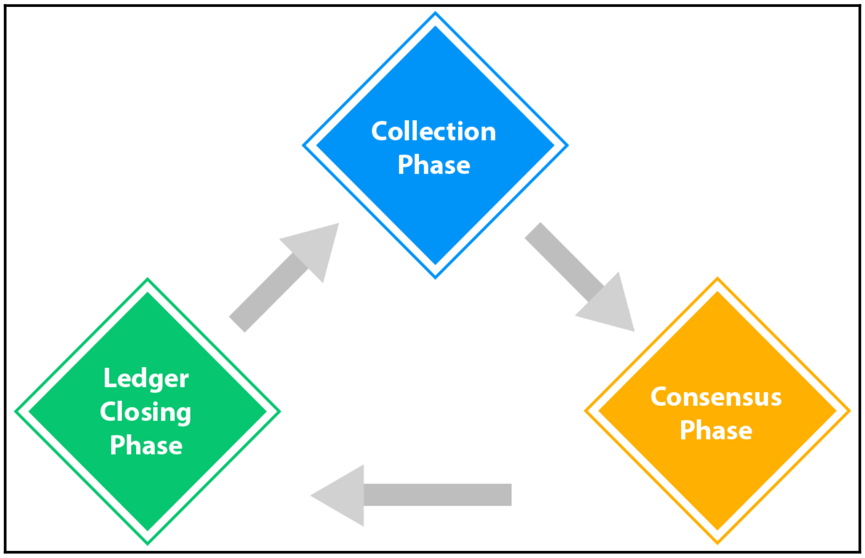
Enterprise

Enterprises can use blockchain in various cases such as coordination among departments, intra-office and inter-office communication, data migration, and various other tasks. Microsoft, IBM, Google, Amazon, and other companies have already started beta testing the usage of blockchain in various enterprise departments.

Ripple

Ripple acts as a real-time gross settlement and remittance network built by Ripple company, and founded in 2012. It allows payments between parties in seconds. It operates with its own coin, known as **Ripple** (**XRP**), and also supports non-XRP payments. Ripple has proposed a new decentralized global network of banks and payment providers, known as RippleNet. This network uses Ripple's transaction settlement technology at its core. RippleNet is proposed to be independent of banks and payment providers, setting a standardized network for real-time payment settlement.

Ripple networks consist of various nodes that perform each of their own defined tasks. The first nodes that facilitate the system are called **user nodes**. The user nodes use Ripple for payment and transactions, such as to make and receive payments. The second type of node in Ripple is the validator node. These nodes are part of a consensus mechanism in the Ripple network. Nodes in the **unique node list** (**UNL**) are part of the Ripple network and trusted for the consensus mechanism. Anyone can become a validator node or a user node. The following diagram displays the flow of transactions that take place in the Ripple network. The transaction begins in the collection phase, and then proceeds to move through the consensus phase. The final phase, which is the ledger closing phase, creates the block of the certain transaction for the next set of transactions to be received:



For the consensus mechanism, Ripple uses the **Ripple Protocol Consensus Algorithm** (**RPCA**). RPCA works neither on **Proof of Work** (**PoW**) nor **Proof of Stake** (**PoS**) systems; instead, its consensus mechanism works on a correctness-based system. The consensus process works on a voting system by seeking acceptance from the validator nodes in an iterative manner so that a required number of votes is received. Once the required number of votes is received, the changes are validated and the ledger is closed. Once the change in the ledger is accepted and the ledger is closed, an alert is sent to the network.

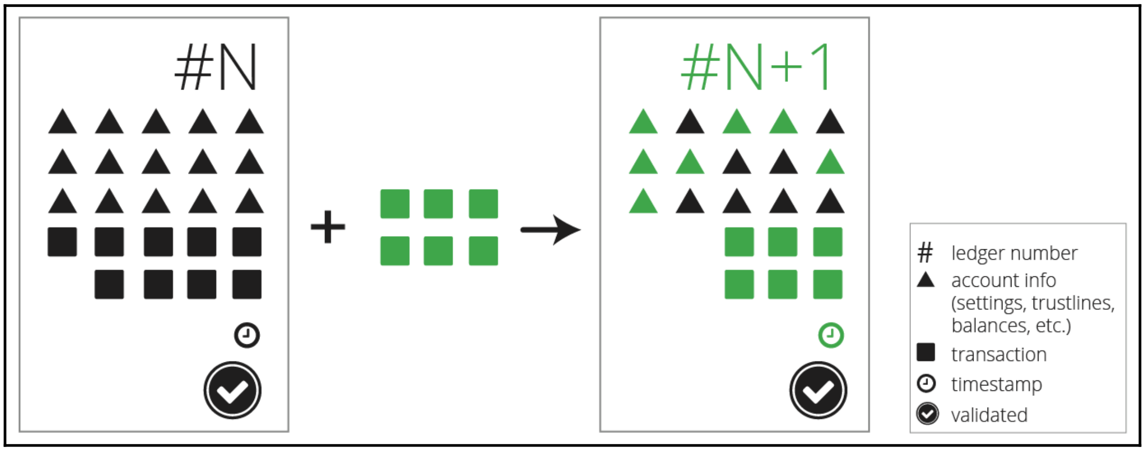
A Ripple network consists of various elements, which together make the transactions in Ripple successful:

- **Validator**: This element is the participant in the consensus protocol.
- **Ledger**: This contains elements such as the ledger number, account settings, transactions, timestamp, and ledger validation flag.
- **Open ledger**: This is the ledger on which the voting takes place. An open ledger contains the proposed transactions.
- **Unique node list**: This is the list of nodes that a validating server uses to seek votes and move the consensus ahead.
- **Proposer**: This element proposes new transactions that are to be made part of the consensus process.

Transactions

Transactions are created by the Ripple network nodes in order to update the ledger. A transaction is required to be digitally signed and validated for it to be part of the consensus process. Each transaction costs a small amount of XRP, just like Gas in Ethereum. There are various types of transactions in Ripple Network: payments related, order related, and account related.

There are also various developer APIs available in the Ripple network to work with the transactions and payments, along with integration on RippleNet. Interledger works with RippleNet to enable compatibility with different networks. The following diagram depicts what a block consists of in a Ripple network transaction:



Stellar

The Stellar network is for the exchange of any currency, including custom tokens. Stellar has a consensus system that is more commonly known as the **Stellar consensus protocol** (**SCP**), which is based on the **Federated Byzantine Agreement** (**FBA**). SCP is different from PoW and PoS with its prime focus to offer lower latency for faster transactions.

It has four main properties:

- **Decentralized property**: It allows participation by anyone without any central party
- **Low latency**: It addresses the much-desired requirement of fast transaction processing
- **Flexible trust**: It allows users to choose which parties they trust for a specific purpose
- **Asymptotic security**: It makes use of digital signatures and hash functions for providing the required level of security on the network

The Stellar network maintains a distributed ledger that saves every transaction and is replicated on each Stellar server connected to the network. The consensus is achieved by verifying a transaction among servers, and updating the ledger with updates of the very same transaction. The Stellar ledger can also act as a distributed exchange order book as users can store their offers to buy or sell currencies.

Tendermint

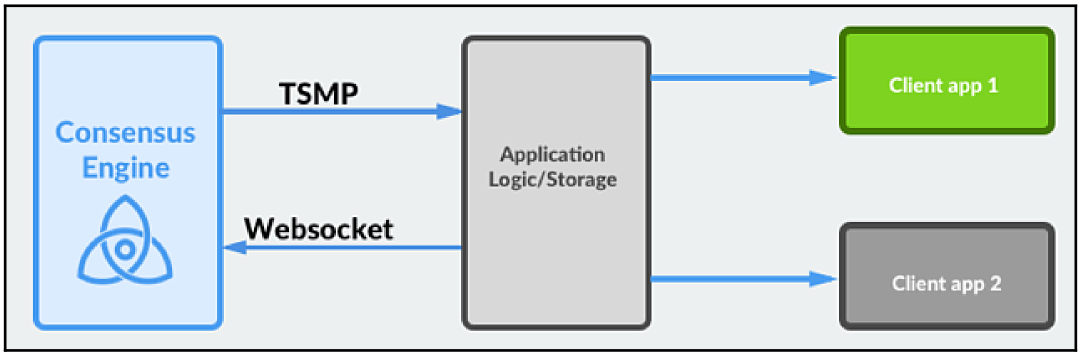
Tendermint provides a secure and consistent state machine replication functionality. Its main task is to develop a secure and high-performance, easily replicable state machine. It is Byzantine Fault Tolerant, that is, even if one in three of the machines fail, Tendermint will keep on working.

The two prime components of Tendermint are as follows:

- **The Tendermint Core**: The Tendermint Core enables secure replication of transactions on each node present in the network. It is a consensus engine.
- **The Tendermint Socket Protocol** (**TMSP**): TMSP is an application interface protocol that allows interfacing with any programming language and helps to process transactions.

The Tendermint consensus algorithm is a round-based mechanism where validator nodes initiate new blocks in each round being done. A locking mechanism is used to ensure protection against a scenario when two different blocks are selected for closing at the same height of the blockchain. Each validator node syncs a full local replicated ledger of blocks that contain transactions. Each block contains a header that consists of the previous block hash, timestamp of the proposed block, present block height, and the Merkle root hash of all transactions present in that block.

The following diagram shows the flow between the consensus engine and the client apps via the Tendermint Socket protocol:



Participants of the Tendermint protocol are generally referred to as **validators**. Each validator takes turns to propose blocks of transactions. They also vote on them just like the Ripple voting system discussed previously. If a block is unable to commit, the protocol moves on to the next round. A new validator then proposes a block for that same height. Voting requires two stages to successfully commit a block. These two stages are commonly known as the pre-vote and pre-commit stage. A block is committed only if more than two in three of the validators pre-commit for the same block and in the same round.

Validators are unable to commit a block for a number of reasons. These would include the current proposer being offline, or issues with the quality or speed of the network. Tendermint also allows the validators to confirm if a validator should be skipped. Each validator waits for a small amount of time to receive a proposal block from the concerned proposer. Only after this voting can take place can they move to the next round. This dependence on a time period makes Tendermint a synchronous protocol despite the fact that the rest of the protocol is asynchronous in nature, and validators only progress after hearing from more than two-thirds of the validator sets. One of the simplifying elements of Tendermint is that it uses the same mechanism to commit a block as it does to skip to the next round.

Tendermint guarantees that security is not breached if we assume that less than one-third of the validator nodes are Byzantine. This implies that the validator nodes will never commit conflicting blocks at the same height. There are a few locking rules that modulate the paths that can be followed. Once a validator pre-commits a block, it is locked on that block. In such cases, it must pre-vote for the block it is to be locked on, and it can only unlock and pre-commit for a new block.

Monax

Monax is a blockchain and a smart contract technology that was founded in 2014. It started its journey as Eris Industries, but changed its name to Monax in October of 2016.

Monax has a lot to offer. Some of these include various frameworks, SDKs, and tools that allow accelerated development of blockchains and their deployment for businesses. The idea behind the Monax application platform is to enable development of ecosystem applications that use blockchains in their backend. It also allows integration with multiple blockchains and enables various third-party systems to interact with other blockchain systems, and offers a high level of compatibility. This platform makes use of smart contracts written in solidity language. It can interact with blockchains such as Ethereum or Bitcoin. All commands are standardized for different blockchains, and the same commands can be used across the platform.

Monax is being actively used for the following applications:

- Fleet leasing
- Corporate governance
- Intellectual property rights
- Legal processes

Summary

In this chapter, we were introduced to alternative blockchains. We discussed the various use cases of blockchains other than cryptocurrency. Some of these included government, healthcare, medical research, supply chains, copyright, fine art, shipping, energy, and so on. As well as this, we discussed Ripple, which is a new blockchain used for fast payments and offers various modifications and improvements compared to the Bitcoin blockchain. After that, we discussed the Stellar payment protocol and its prime properties, which help to accelerate payments in Stellar. Tendermint is another blockchain software, which was discussed and brought to our attention.

In the next chapter, we will discuss, in detail, Hyperledger and some of the prominent projects based on the Hyperledger protocol. We will also discuss detailing and other parameters of the Hyperledger protocol.

11
Hyperledger and Enterprise Blockchains

Unlike most of the other blockchain systems discussed in this book, Hyperledger never had an **Initial Coin Offer** (**ICO**) and has no tokens that are publicly traded. This is because Hyperledger isn't a blockchain itself, but instead a collection of technologies used for the creation of new blockchains. Moreover, these blockchain technologies were explicitly designed and built for enterprise use cases and not the public markets.

In this chapter, we will cover the following items:

- The history of Hyperledger
- Critical details about each of the six current Hyperledger projects
- An overview of the Hyperledger tools used to support the ecosystem
- Suggestions on how to select Hyperledger technologies for a project

The Hyperledger name doesn't apply to a single technology, but rather a collection of blockchain technologies all donated to the Linux Foundation under the Hyperledger name.

Members of the Hyperledger project include major blockchain companies such as Consensys, R3, and Onchain, as well as a large number of enterprise-focused technology companies such as Baidu, Cisco, Fujitsu, Hitachi, IBM, Intel NEC, Red Hat, and VMware. In addition to these companies, a number of financial services firms have come on board due to the clear application of blockchain in fintech. Financial services members include ANZ Bank, BYN Mellon, JP Morgan, SWIFT, and Wells Fargo. Seeing the opportunity to be on the next wave of business software consulting, major integrators joined as well—Accenture, CA Technology, PWC, and Wipro, along with many others.

Recently, Amazon, IBM, and Microsoft have all revealed blockchain-as-a-service offerings featuring Hyperledger technology.

History of Hyperledger

The Hyperledger project was founded in 2015, when the Linux Foundation announced the creation of the Hyperledger project. It was founded in conjunction with a number of enterprise players, including IBM, Intel, Fujitsu, and JP Morgan. The goal was to improve and create industry collaboration around blockchain technology so that it would be usable for complex enterprise use cases in the key industries most suitable to blockchain disruption: technology, finance, and supply chain.

The project gained substance in 2016, when the first technology donations were made. IBM donated what was to become known as **Hyperledger Fabric**, and Intel donated the code base that became Hyperledger Sawtooth.

Unlike most projects in the blockchain space, Hyperledger has never issued its own cryptocurrency. In fact, the executive director of Hyperledger has publicly stated that there never will be one.

Hyperledger projects

As mentioned, a hyperledger isn't a single blockchain technology, but rather a collection of technologies donated by member companies. While there is a long-term goal of greater integration between projects, currently most of the Hyperledger projects function independently. Each project's core code base was donated by one or more of the Hyperledger member organizations, based on problems they were trying to solve internally before open sourcing the code and handing ownership to the Linux Foundation.

Hyperledger Burrow

Hyperledger Burrow is a re-implementation of the **Ethereum Virtual Machine** (**EVM**) and blockchain technology, but with a few key changes. First, instead of using the proof-of-work consensus algorithm used by the public Ethereum chain, Burrow is designed around the Tendermint consensus algorithm (See `Chapter 7`, *Achieving Consensus*). This means there are no miners and no mining to be done on Burrow-based projects.

Second, Hyperledger Burrow is permissioned—the computers allowed to participate in a Hyperledger Burrow network are known and granted access, and the computers signing blocks (called **validators**, as in Tendermint) are all known. This is very different than Ethereum, where anyone can anonymously download the Ethereum software and join the network.

Smart contracts written for the EVM will still mostly work. Due to the change in consensus, there are also changes in the way Gas is used. In the public Ethereum blockchain, each transaction costs Gas, depending on the complexity of the transaction, and each block has a Gas limit. Depending on the network load, participants have to pay a variable cost in Ether for the Gas needed for their transactions. In Burrow, these complexities are mostly dispensed with. Each transaction is gifted a basic amount of Gas automatically. Because the Gas is still limited, Burrow is able to guarantee that all transactions eventually complete—either by succeeding or failing—because they run out of Gas.

For more on the EVM, solidity language, and other aspects of Ethereum shared with Hyperledger Burrow, please see `Chapter 12`, *Ethereum 101*, to `Chapter 15`, *Ethereum Development*.

Hyperledger Sawtooth

Hyperledger Sawtooth, like the rest of the Hyperledger family, is built for permissioned (private) networks rather than public networks, such as Ethereum, Bitcoin, and so on. As an enterprise-oriented blockchain system, it is designed around allowing different companies to coordinate using a blockchain and smart contracts. Originally developed by Intel, Sawtooth uses a unique consensus algorithm called **Proof of Elapsed Time**, or **PoET**.

PoET uses a lottery-based system for leader election. Using special Intel technology called the **Trusted Execution Environment** (**TEE**), along with **Software Guard Extensions** (**SGX**), available on some Intel chipsets, the leader is elected by each node generating a random wait time, with the shortest wait time going first. Because the code to generate the wait time is in the TEE, it can be verified that each node is running appropriate code and not skipping in line to become leader by not waiting the amount of time generated by the random time generator. Therefore, the election of the leader (and block issuer) is very fast, which in turn allows the blockchain to operate quickly.

Sawtooth architecture

Sawtooth has a pluggable architecture, comprised of the Sawtooth core, the application-level and transaction families, and the consensus mechanism (typically PoET, but hypothetically pluggable with others). We will study them in detail in the following sections.

Transaction families

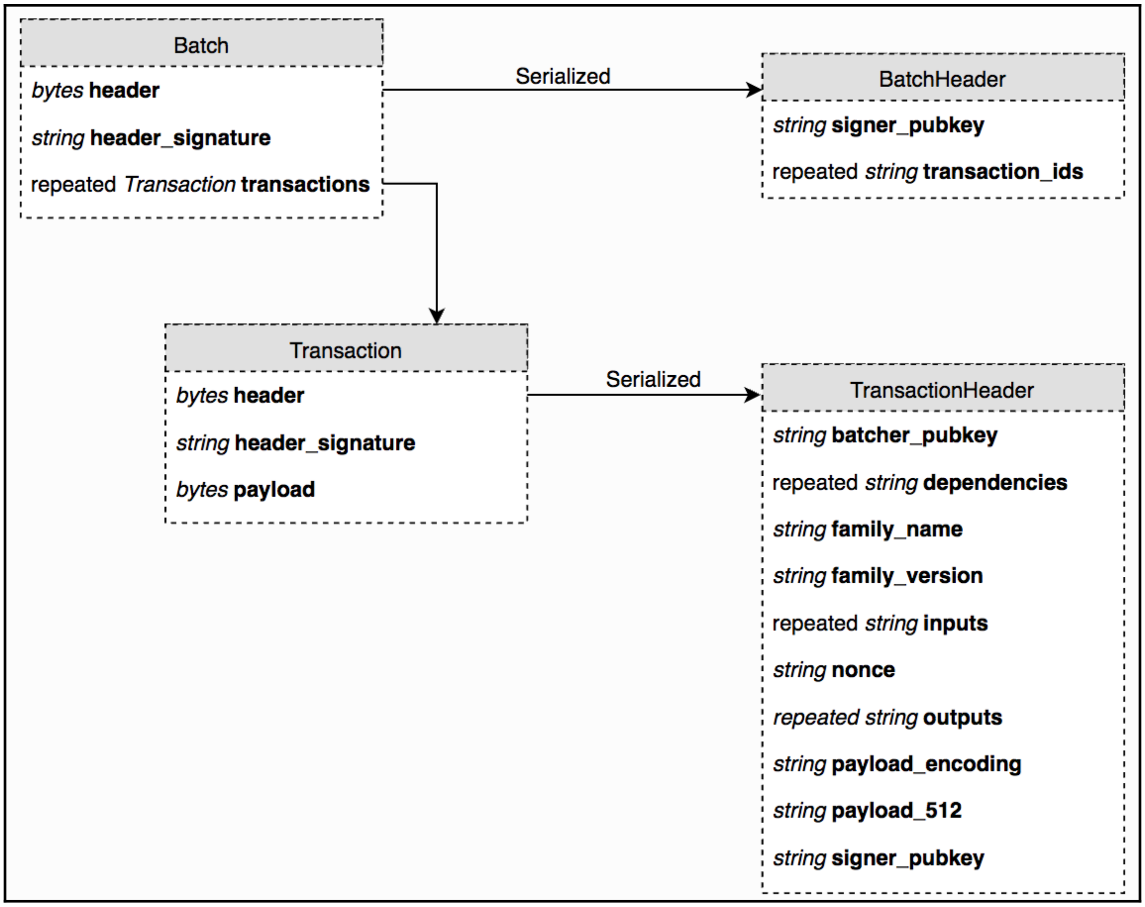
Because Sawtooth is meant to be a pluggable, enterprise-oriented architecture, the application layer is highly configurable. Each Sawtooth-based blockchain allows transactions to be made based on what are called **transaction families**. Transaction families determine what sorts of operations are permissible on the Sawtooth blockchain. For instance, it is possible to allow smart contracts, such as with Ethereum, using the **Seth** transaction family. Under Seth, all possible Ethereum-based contracts and Ethereum contract-based transactions would be permissible, along with all the possible mistakes and issues such freedom creates.

A Sawtooth-based blockchain can have multiple transaction families operating at once. In fact, this is common, as one of the transaction families that ships with Sawtooth is the settings family, which stores system-wide configuration settings directly on to the blockchain. In most cases, this transaction family and a few others, comprised of business use cases, will be operating in parallel. Furthermore, because multiple transaction families can be running at the same time, this means that business logic can be isolated and reused as an independent transaction family across multiple blockchain implementations.

Because many businesses have only a few valid business rules and business outcomes, it is possible to customize the available operations on the blockchain through the creation of a custom transaction family. For instance, a shipping company may use Sawtooth to track the location of packages, and the only valid transactions might be new package, package accepted, package released, package in transit, update package location, and package delivered. By restricting the available transactions, the number of errors and mistakes can be reduced. Using the shipping company example, network participants could be trucking companies, warehouses, and so on. For a package to move between a truck and a warehouse, the two network participants would issue package released and package accepted transactions, respectively, in a batch on to the blockchain. This brings us to the next concept in Sawtooth: **Transaction Batches**.

Transactions and batches

In Sawtooth, transactions are always part of batches. A *batch* is a set of transactions that come together and either all succeed or all fail. If a transaction needs to process by itself, then it would be in a single batch containing only that one transaction. Using the shipping company example, the transactions package released and package accepted may be programmed so that they only succeed if their counter-transaction is part of the same batch, forcing a successful handoff or throwing an error. The following diagram shows the data structure of a transaction batch:


Batch
bytes **header**
string **header_signature**
repeated *Transaction* **transactions**
Serialized
BatchHeader
string **signer_pubkey**
repeated *string* **transaction_ids**
Transaction
bytes **header**
string **header_signature**
bytes **payload**
Serialized
TransactionHeader
string **batcher_pubkey**
repeated *string* **dependencies**
string **family_name**
string **family_version**
repeated *string* **inputs**
string **nonce**
repeated string **outputs**
string **payload_encoding**
string **payload_512**
string **signer_pubkey**

Transactions and batches in Sawtooth are abstracted at a high level so that they can be created by custom transaction families and by arbitrary programming languages. Because of this, it is possible to program smart contracts and transaction families in Java, Python, Go, C++, and JavaScript. To code in any language, there is another restriction on transactions: serialization, or the transition from an in-memory structure on a computer to a fixed binary that can be sent across the network. No matter what the language is, the approach to serialization must have the same output. In Sawtooth, all transactions and batches are encoded in a format called **protocol buffers**, a format created by Google internally, and released in 2008. Protocol buffers are a solution to having a fixed and high-performance method of data-exchange between computers that is programming language and computer architecture independent.

The key pieces

Transaction families and transactions in Sawtooth require a few things to be created by developers. Consider the following:

- First, you need the protocol buffer definitions for each transaction the data model of what will be stored
- Secondly, you need a transaction handler that will handle incoming transactions that are part of the transaction family
- Lastly, it is necessary to register the handler using the core SDK with the transaction processor

Sawtooth includes Python-based sources to serve as examples in both the settings and identity-based transaction families on GitHub. Next, we'll cover Hyperledger Fabric, another enterprise-oriented blockchain technology.

Hyperledger Fabric

Hyperledger Fabric, like Sawtooth, is designed to be an enterprise-oriented blockchain solution that is highly modular and customizable. Hyperledger Fabric is both private and permissioned. This means that, like Sawtooth, by default Hyperledger blockchains are not observable to the public at large and are not and will not have tokens that are tradeable on exchanges. Users of the blockchain must have validated identities and join the blockchain by using a **Membership Service Provider** (**MSP**). These MSPs are configured on the system, and there can be more than one, but all members must successfully be granted access by one or more MSPs. Fabric also has a number of special tools that make it particularly full-featured, which we will cover later.

Architecture choices and features

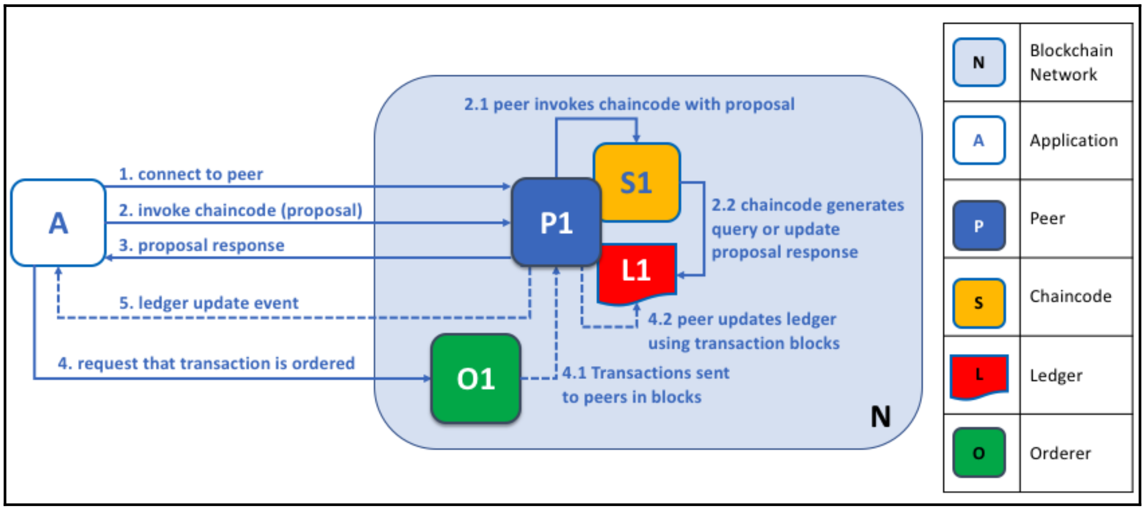
Hyperledger Fabric was designed around a few key features and use cases that were seen as critical for enterprise users.

At the core is the ledger itself. The ledger is a set of blocks, and each block holds a set of transactions. A transaction is anything that updates the state of the blockchain. Transactions, in turn, are performed by smart contract code installed on to the blockchain (called **Chaincode**). Let's look at how blocks and transactions are formed.

Each block is ordered in sequence, and inside each block is a set of transactions. Those transactions are also stored as happening in a specific sequence. Unlike other blockchains, the creation of the transaction and the sequence it is eventually given are not necessarily performed at the same time or on the same computer. This is because the ordering of transactions and the execution of transactions are separated. In Hyperledger Fabric, computers being used to operate the blockchain can run in three different modes (node types); these are as follows:

1. **Client**: A client acts on behalf of users of the blockchain and submits actions and events to the network as part of an application.
2. **Peer**: Peers process the incoming transactions for validating and handle updating state changes as a result of transactions and chain-code execution. Once they have executed a transaction, they broadcast the result to the network so that the transaction can be handled by an orderer (see the next role).
3. **Orderer**: While peer nodes execute the transactions, orderer nodes take a look at all the executed transactions and decide on the final order in which they are considered to have occurred in the blockchain. The ordering service nodes decide the final order of events, and thus decide the final set of events that will be written on to the next block of the blockchain.

It is important to note that a single computer can act as up to all three of these node types on a Fabric blockchain, but this is not necessary. While it is possible for the same computer on a Hyperledger network to both execute transactions and order their sequence, Hyperledger is able to scale more by providing these as distinct services. To illustrate, look at the following diagram (from the Hyperledger documentation):

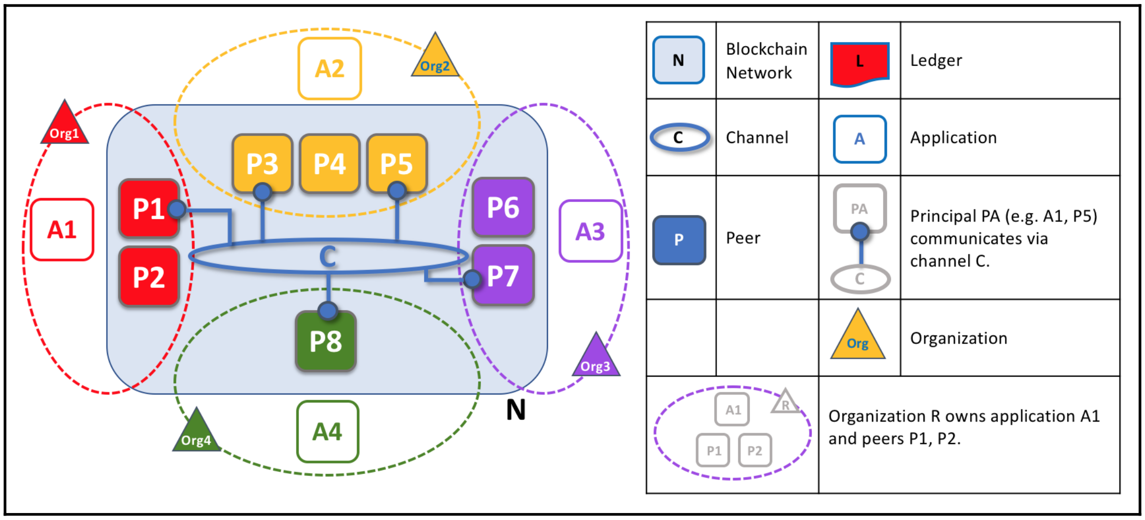


2.1 peer invokes chaincode with proposal
1. connect to peer
2. invoke chaincode (proposal)
3. proposal response
A
P1
S1
L1
2.2 chaincode generates query or update proposal response
5. ledger update event
4.2 peer updates ledger using transaction blocks
4. request that transaction is ordered
O1
4.1 Transactions sent to peers in blocks
N
N Blockchain Network
A Application
P Peer
S Chaincode
L Ledger
O Orderer

As you can see, incoming transaction first go to peers, who execute transactions using chaincode/smart contracts and then broadcast successful transactions to the ordering service. Once accepted, the ordering service decides on a final order of transactions, and the resulting transactions sets are re-transmitted to peer nodes, which write the final block on to the chain.

Organizational focus

As an enterprise-oriented system, Fabric distinguishes between peers and orderers (nodes on a blockchain network) and the organization that owns them. Fabric is meant to create networks between organizations, and the nodes running the blockchain do so as agents of that organization. In this way, each node and its permissions are related to the organization it helps represent. Here is another diagram from Hyperledger:



As you can see, each network node operates the blockchain network on behalf of the contributing organization. This is different than networks such as Ethereum and Bitcoin, where the network is created by a set of computers that contribute resources to the network independently, or at least are perceived by the network as doing so independently, no matter who owns them. In Hyperledger Fabric, it is the organizations that create the shared ledger that contribute to the network by contributing resources in the form of peers and ordering nodes. The distinction is subtle but crucial. In most public networks, the idea is to allow computers to coordinate, but in Fabric the idea is to allow companies to coordinate. Owning organizations provides each of their peers a signed digital certificate proving their membership of a certain organization. This certificate then allows each node to connect to the network through an MSP, granting access to network resources. The organizations versus private computers focus brings us to another enterprise-oriented feature of Hyperledger Fabric, one that is necessary for a number of corporate requirements: **Private Channels**.

Private channels

Hyperledger Fabric has a critical and unique functionality called *private channels*. A private channel allows a subset of members on a Fabric-based blockchain to create a new blockchain that is observable and intractable only for them. This means that, while Fabric is already private and permissioned, members of that private blockchain can create a smaller, yet more exclusive, chain to trade information that cannot be traded across the full membership network. As a result, Fabric is able to support critical use cases (such as legal communications) that would be impossible or inappropriate to broadcast, even on a relatively exclusive network.

For instance, if Hyperledger Fabric were used to set up a logistics network, the primary blockchain could be used for the tracking of packages, but pricing bids could be done on private channels. The participants of the network would be a number of shipping providers, materials providers, and a set of buyers. A buyer could issue to the blockchain a notice that they were accepting bids for a transfer of some supplies, and they could then create private channels between themselves and all transporters and suppliers. The suppliers and shipping companies could give the buyer time and cost prices, without making that information public to their competitors. While private, all of these exchanges would be encoded on to the blockchain for record keeping, legal compliance, and so on. Moreover, if corporate policy were something like taking the second-lowest bid, the entire process could be automateable through smart contracts.

Assets

In Hyperledger Fabric, assets are defined as anything that can be given a value. While this could be used to exchange different fiat currencies, an asset could be designed to denote something abstract, such as intellectual property, or something more tangible, such as a shipment of fresh fish.

In Fabric, assets are processed internally as simple key-value pairs, with their state stored on the ledger and modifiable via the chaincode. Assets in Hyperledger can fulfill all the duties performed in Ethereum by ERC-20 and ERC-721 tokens, and beyond. Anything that can be described in a token format can be stored as an asset in Hyperledger Fabric.

Smart contracts

In Hyperledger Fabric, **Smart Contracts** are called **chaincode**. Unlike Ethereum, chaincode is not embedded directly inside the ledger itself. Instead, chaincode is installed on each peer node and interacts with the ledger to read and update state information about the assets the chaincode controls. Because the chaincode is signed and approved by all peers, and because each peer that uses a piece of chaincode must validate any state changes on the ledger, this system still allows distributed and trusted consensus, using smart contracts. To provide consistency, chaincode itself runs in an isolated Docker container.

Because of the modular nature of both the distributed ledger and the chaincode, multiple programming languages can be used to develop smart contracts; however, currently, the supported options are limited. There are full-featured SDK packages only for Go and Node.js, with eventual plans to add Java.

Advantages of Fabric

Fabric is one of the most popular of the Hyperledger projects for good reason. It's highly modular, designed for coordination across companies, and the private channel feature enables secure functionality that's impossible on public chains, and even most private ones. Moreover, Hyperledger Fabric has **Composer**—a visual tool for architecting blockchain applications. We'll discuss Composer later in the section on Hyperledger tools.

Next, we'll cover **Iroha**, a Hyperledger project aimed at bringing blockchain to mobile devices.

Hyperledger Iroha

Hyperledger Iroha is a project written in C++ and contributed to by Sorimitsu. The goals of the project were to provide a portable C++ based blockchain implementation that could be used in mobile devices. Both iOS and Android operating systems, along with small computers such as Raspberry Pi, are all capable of efficiently running tightly written C++ code efficiently. To make things even easier, Iroha provides iOS, Android, and JavaScript libraries for developers.

One major difference from Ethereum, in particular, is that Hyperledger Iroha allows users to perform common functions, such as creating and transferring digital assets, by using prebuilt commands that are in the system. This negates the need to write cumbersome and hard-to-test smart contracts for the most common functionalities, enabling developers to complete simple tasks faster and with less risk. For instance, to create a new token type on Iroha, it takes just a single command—`crt_ast`. To make things even easier, Iroha has a command-line interface that will guide a new user through creating the asset, without writing code at all.

If the goals of Sawtooth and Fabric are completeness, Iroha is oriented more towards ease of use and device compatibility.

Hyperledger Indy

One of the more common use cases for blockchain technology is identity verification and authorization. You have probably experienced the issues around the web, where you either need to remember many usernames and passwords to confirm your identity to another provider, such as Google or Facebook. The problem here is that you must trust Google, Facebook, or other providers to manage your identity and keep it safe. This creates a single point of failure and allows a centralized authority to control whose identities are valid and what rights they have. This ecosystem is an obvious target for disruption and decentralization.

Hyperledger Indy is a blockchain project built around decentralized, self-declared identity. The goal of Indy is to provide tools and libraries for creating digital identities that can be managed on a blockchain and made interoperable with other applications and use cases that require identity verification.

While Fabric, Sawtooth, and Iroha all have some level of identity mechanism built in, Indy is specifically oriented around identity management and for use by applications that may not run on a blockchain. Thus, Indy could be used to provide identity services to web applications, company resources, and so on. Existing companies include Sovrin (who donated the original Indy codebase) and Every.

Tools in Hyperledger

An often overlooked aspect of any application is the need for helpful tools to manage the life cycle of that application. Tools, such as software to ease deployment, debugging, and design, can make a tremendous difference in the ease of use of a system, for developers and users alike. Most public blockchains are severely hampered by the lack of high-quality tools and support. The Hyperledger ecosystem, however, continues to invest in building excellent support tools.

Hyperledger Caliper

One of the common needs for any system is benchmarking. **Hyperledger Caliper** is a blockchain-oriented benchmarking tool, designed to help blockchain architects ensure that the system can perform fast enough to meet the needs of the hosting organizations. Using a set of pre-defined, common use cases, Hyperledger Caliper will report an assortment of critical performance measurements, such as resource usage, **Transactions Per Second** (**TPS**), transaction latency, and so on.

Using Caliper, a team working on blockchain applications can take continuous measurements as they build out smart contracts and transaction logic and use those measurements to monitor performance changes. Caliper is compatible with Sawtooth, Indy, and Fabric blockchain systems.

Hyperledger Composer

Hyperledger Composer is a design tool for building smart contracts and business applications on the blockchain. It is designed for rapid prototyping of chaincode and asset data models for use with Hyperledger Fabric. As a Fabric-specific tool (so far), it is primarily designed around helping with Hyperledger Fabric specific concepts, such as assets, identity management, transactions, and the resultant chaincode used to power the business rules between all these items.

It is not designed as a "fire-and-forget" tool, where someone can build an entire ecosystem from scratch to production, rather it is designed for rapid visual prototyping to get testable applications up and running quickly, with finer details iterated directly in the codebase. IBM hosts a demo online at `https://composer-playground.mybluemix.net/editor`.

The primary users of Composer will be blockchain developers (especially new developers) and some technical business users. It sits well as part of an agile process to developing blockchain applications, allowing developers, network administrators, and technical business users to visualize the network and the code operating on it.

Hyperledger Cello

If Composer is used to assist with building aspects of a Fabric-based blockchain, then **Cello** is a tool to assist with the deployment of that blockchain to various servers and cloud services. Cello can be used to manage the blockchain infrastructure or launch new blockchains in a blockchain-as-a-service approach. Common life cycle and deployment tasks include starting, stopping, and deleting a blockchain, deploying new nodes to an existing blockchain, and abstracting the blockchain operation so that it can run on local machines, in the cloud, in a virtual machine, and so on. Cello also allows monitoring and analytics.

Cello is primarily a tool for what is called **DevOps**, or the connection between development teams and production operations. It is primarily aimed at the Hyperledger Fabric project, but support for Sawtooth and Iroha is intended for future development.

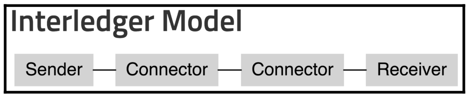
Hyperledger Explorer

Hyperledger Explorer is a blockchain module and one of the Hyperledger projects hosted by the Linux Foundation. Designed to create a user-friendly web application, Hyperledger Explorer can view, invoke, deploy, or query blocks, transactions, and associated data, network information (name, status, list of nodes), chain codes and transaction families, as well as any other relevant information stored in the ledger. Hyperledger Explorer was initially contributed by IBM, Intel and DTCC.

Hyperledger Quilt

There are times when it makes sense for multiple blockchains to be able to communicate. This is where **Hyperledger Quilt** comes in. Quilt is a tool that facilitates cross-Blockchain communication by implementing an **Interledger protocol** (**ILP**). The ILP is a generic specification available to all blockchains to allow cross-ledger communication, originally created by ripple labs. With ILP, two ledgers (they do not have to be blockchains) can coordinate, to exchange values from one ledger to the other.

ILP is a protocol that can be implemented using any programming language or technology, as long as it conforms to the standard. Because of this, it can be used to join multiple completely independent ledgers, even ones with radically different architecture. These ledgers do not need to be blockchains, but can be any system of accounting. In ILP, cross-ledger communication occurs primarily through actors called *connectors*. See the following diagram from `interledger.org`:



The ILP bridges ledgers with a set of connectors. A connector is a system that provides the service of forwarding interledger communications towards their destination, similar to how packets are forwarded across the internet—peer to peer. ILP communication packets are sent from senders to a series of connectors that finally land at receivers.

The connectors are trusted participants in this sequence, and the sender and all intermediate connectors must explicitly trust one another. Unlike other blockchain-oriented technology, ILP does not involve trustless exchange. However, the sender and each connector need trust only their nearest links in the chain for it to work.

Quilt is the implementation of the ILP that has been donated to the Hyperledger project, on behalf of ripple labs, Everis, and NTT DATA. These organizations have also sponsored ongoing dedicated personnel to help improve the Quilt codebase, which is primarily in Java.

Relationships between the tools

The distinctions between Fabric, Cello, Composer, Explorer, and Caliper can be described as follows:

- Fabric is the application itself, and where the business logic will finally reside

- Composer is a tool to help build this logic and the final chaincode

Both Fabric and Composer are going to be principally involved in the development phase of a blockchain project, followed shortly after by Caliper for performance testing:

- Cello and Explorer are tools dealing more with the operations side of a project. Cello is used by DevOps teams to deploy and extend blockchain applications across servers and virtual machines
- Explorer is a monitoring tool to inspect what is going on across different ledgers

Finally, Hyperledger Quilt can be used to connect different ledgers and blockchains together. For instance, Quilt could be used to communicate from a Fabric-based system to the public Ethereum network, or to the ACH banking system, or all of the above.

Thus, the Hyperledger project has tools for end-to-end creation, operation, and interoperability of blockchain-based application ecosystems.

Which Hyperledger project should you use?

Given the numerous sub-projects inside Hyperledger that are all focused on business use cases, it would not be surprising if there was some confusion about which to use. This is understandable, but the good news is that for most cases the proper project to build on is clear.

By far the most popular and well-documented framework is Hyperledger Fabric. Fabric also has blockchain-as-a-service support from Amazon and Microsoft. In addition, Composer, Cello, and Caliper tools all work with the latest versions of Fabric. For the vast majority of projects, Hyperledger Fabric will be the project of most interest.

The second most-obvious choice is Sawtooth. For supply chain solutions, Sawtooth already has a reference implementation. In addition to this, Sawtooth has better support for writing smart contracts in multiple languages, whereas Hyperledger has support only for Go and JavaScript. In addition to this, Sawtooth core is written in Python. Python is a very popular language in data science, a field that is regularly paired with blockchain technology.

The final choices are Burrow, which would make a good match for technologies migrating from Ethereum, or needing to interface with the public Ethereum network, and Iroha, which would be a better match for projects that need to run a blockchain across mobile devices or other small machines.

Using Hyperledger

Like much blockchain technology, the Hyperledger ecosystem is relatively new, and many projects have not even hit a full 1.0 release yet. While there is a large amount of development activity and multiple working systems already in production use, the system as a whole is fractured. For instance, Sawtooth is written in Python, Fabric in Go, Quilt in Java, and so on. Even staying inside the Hyperledger family, it would be difficult to use a homogeneous set of technologies for end-to-end implementations.

Moreover, Hyperledger's focus on private networks is a problem for the projects that may wish to have a public component. One of the appeals of blockchain technology is transparency. A project that seeks maximum transparency through public usage of their technology may need to look elsewhere or find a way to bridge between Hyperledger and public networks—possibly by using Quilt and ILP.

Similarly, projects looking to raise funds through an ICO should probably look elsewhere. Few projects have tried to use Hyperledger as part of an ICO, and, as far as we know, none of those have actually succeeded in fundraising. Hyperledger remains oriented strongly toward private networks—where it has succeeded tremendously.

Summary

Now you have a good idea of the different subprojects that make up Hyperledger and an awareness of the tooling you can use to build Hyperledger-based projects. Hyperledger is a set of technologies for building private blockchain networks for enterprises, versus the public and tradeable networks, such as Ethereum and Bitcoin. The Hyperledger family is made of six projects and a set of support tools, all with subtly different focuses and advantages to suit different projects.

Over time, the different projects are expected to become more consistent and interoperable. For instance, Hyperledger Burrow and Hyperledger Sawtooth have already cross-pollinated with the Seth transaction family, which allows Sawtooth to run Ethereum smart contracts. It is expected that tools such as Cello and Composer will be extended to support additional Hyperledger projects in time, leading to an increasingly robust ecosystem.

Next, we will discuss Ethereum in depth. Ethereum is a public blockchain network and the first and most popular of the public networks to support fully-programmable smart contracts.

12
Ethereum 101

In the previous chapters, we have studied in detail blockchain, Bitcoin, alternative cryptocurrencies, and crypto wallets. We discussed blockchain usage and benefits in not only currency-based applications, but other similar areas. We also discussed how Bitcoin has changed the landscape of blockchain usage for monetary benefits and how it has shaped the global economy.

In this chapter, we will be studying Ethereum blockchain in depth. It is currently the largest community-backed blockchain project, second to Bitcoin, with supporters and various projects and tokens running on top of it. In this chapter, we will discuss the following topics:

- Ethereum accounts
- Ethereum network
- Ethereum clients, such as Geth
- Execution environment for Ethereum blockchain projects
- Ethereum block
- Ethereum virtual machine
- Using gas in an Ethereum transaction
- Ether and Ether procurement methods

Introducing Ethereum

Ethereum is a blockchain-based system with special scripting functionality that allows other developers to build decentralized and distributed applications on top of it. Ethereum is mostly known among developers for the easy development of decentralized applications. There are differences between Ethereum and blockchain. The most important difference is that Ethereum blockchain can run most decentralized applications.

Ethereum was conceptualized in late 2013 by Vitalik Buterin, cryptocurrency researcher and developer. It was funded by a crowd sale between July and August 2014. Ethereum has built in Turing, a complete programming language, that is, a programming language meant to solve any computation complexity. This programming language is known as Solidity and is used to create contracts that help in creating decentralized applications on top of Ethereum.

Ethereum was made live on July 30, 2015, with 11.9 million coins pre-mined for the crowd sale, to fund Ethereum development. The main internal cryptocurrency of Ethereum is known as **Ether**. It is known by the initialism **ETH**.

Components of Ethereum

Let's discuss some general components of Ethereum, its primary currency, network, and other details. This will help in understanding Ethereum in a much better way and also help us see how it is different to Bitcoin and why it has a huge community, currently making it the most important cryptocurrency and blockchain project, second only to the Bitcoin blockchain.

Ethereum accounts

Ethereum accounts play a prime role in the Ethereum blockchain. These accounts contain the wallet address as well as other details. There are two types of accounts: **Externally Owned Accounts** (**EOA**), which are controlled by private keys, and **Contract Accounts**, which are controlled by their contract code.

EOAs are similar to the accounts that are controlled with a private key in Bitcoin. Contract accounts have code associated with them, along with a private key. An externally owned account has an Ether balance and can send transactions, in the form of messages, from one account to another. On the other hand, a contract account can have an Ether balance and a contract code. When a contract account receives a message, the code is triggered to execute read or write functions on the internal storage or to send a message to another contract account.

Ethereum network

Two Ethereum nodes can connect only if they have the same genesis block and the same network ID. Based on usage, the Ethereum network is divided into three types:

- **MainNet**: This is the current live network of Ethereum; as of now the latest version of MainNet is called **homestead**.
- **TestNet**: This is used for testing purposes, for testing smart contracts and DApps by developers, before the contracts are deployed on to the blockchain. The latest version of TestNet is called **Ropsten**.
- **PrivateNet**: This is used to create a permissioned blockchain by generating a new genesis block.

All of the preceding types are the same, apart from the fact that each of them has a different genesis block and network ID; they help to differentiate between various contract accounts and externally owned accounts, and if any contract is running a different genesis, then they use a different network ID to distinguish it from other contract accounts.

There are some network IDs that are used officially by Ethereum. The rest of the network IDs can be used by contract accounts. Here are some of the known IDs:

- **0**: This is the Ethereum public prerelease Testnet, and it is known by the project name Olympic.
- **1**: This ID is the Ethereum public main network; it has had many versions: Frontier, Homestead, and Metropolis. We will be discussing public MainNet more in future sections.
- **2**: This was the ID used by initial Testnet; it was deprecated in late 2016.
- **3**: This is the most recent Testnet at the time of writing this book. It was launched in late 2016, since the older Testnet was having multiple issues.
- **4**: This is a public Testnet, in which a PoA consensus is being tried out. It is known as **Rinkeby**.

Ethereum public MainNet

The public MainNet has a network ID of 1, but since Ethereum has a very active community backing it, there are various updates and upgrades happening to the Ethereum blockchain; primarily, there are four stages of the Ethereum network; let's discuss each of them in detail:

- **Frontier**: This is the first official public main network; it was launched in mid 2015.
- **Homestead**: This was one of the major upgrades in March 2016.
- **Metropolis**: This upgrade stage will be bringing a lot of upgrades to the Ethereum blockchain. This upgrade will be implemented by a hard fork and is divided into two phases, **Byzantium** and **Constantinople**.
- **Serenity**: This release of Ethereum will move the consensus from PoW to **Proof of Stake** (**PoS**). This is essentially being introduced to reduce the power consumption of the Ethereum network.

Ethereum clients

Clients are implementations of the Ethereum blockchain; they have various features. In addition to having a regular wallet, a user can watch smart contracts, deploy smart contracts, clear multiple Ether accounts, store an Ether balance, and perform mining to be a part of the PoW consensus protocol.

There are various clients in numerous languages, some officially developed by the Ethereum Foundation and some supported by other developers:

- **Geth**: Based on the Go programming language, sometimes also known as **go-ethereum**
- **Parity**: Based on the Rust programming language, and developed by Ethcore
- **cpp-ethereum**: Built on C++ and officially developed by the Ethereum Foundation
- **Pyethapp**: Based on the Python programming language, and officially developed by Ethereum
- **ethereumjs-lib**: Based on the JavaScript programming language
- **Ethereum(j)**: A Java-based client, developed by Ether camp
- **ruby-ethereum**: A Ruby-based client, developed by Jan Xie
- **ethereumH**: A Haskell-based client, developed and maintained by BlockApps

The preceding list consists of some of the most prominent Ethereum-specific clients currently in production. There are many other clients apart from these that are not heavily community-backed or are in their development phase. Now let's discuss the most prominent Ethereum client—**Geth**, or **go-ethereum**.

Geth

This is one of the most widely used Ethereum clients built on Golang; it is a command-line interface for running a full Ethereum node. It was part of the Frontier release and currently also supports Homestead. Geth can allow its user to perform the following various actions:

- Mining Ether
- Creating and managing accounts
- Transferring Ether between two Ethereum accounts
- Creating, testing, and deploying smart contracts
- Exploring block history

Installing Geth

Geth can be installed using the following commands on Ubuntu systems:

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
```

After installation, run `geth account new` to create an account on your node. Various options and commands can be checked by using the `geth --help` command.

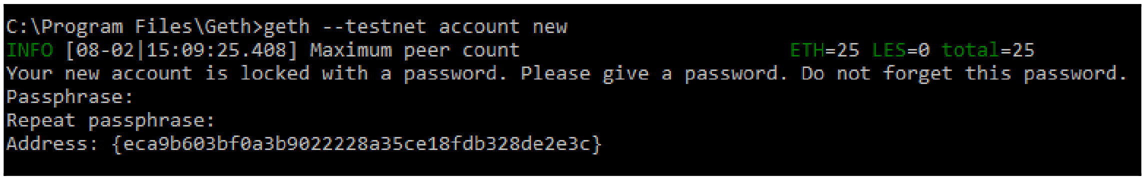
On Windows-based systems, it is much easier to install Geth by simply downloading the latest version from `https://geth.ethereum.org/downloads/` and then downloading the required `zip` file, post-extracting the `zip` file, and opening the `geth.exe` program.

Managing accounts

Geth provides account management, using the `account` command. Some of the most-used commands related to account management on Geth are as follows:

```
COMMANDS:
 list Print summary of existing accounts
 new Create a new account
 update Update an existing account
 import Import a private key into a new account
```

The following screenshot is the output that will be generated after executing the preceding code:



```
C:\Program Files\Geth>geth --testnet account new
INFO [08-02|15:09:25.408] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {eca9b603bf0a3b9022228a35ce18fdb328de2e3c}
```

When we run the command to create a new account, Geth provides us with an address on our blockchain:

```
$ geth account new
Your new account is locked with a password. Please give a password. Do not
forget this password.
Passphrase:
Repeat Passphrase:
Address: {168bc315a2ee09042d83d7c5811b533620531f67}
```

When we run the `list` command, it provides a list of accounts that are associated with the custom keystore directory:

```
$ geth account list --keystore /tmp/mykeystore/
Account #0: {5afdd78bdacb56ab1dad28741ea2a0e47fe41331}
keystore:///tmp/mykeystore/UTC--2017-04-28T08-46-27.437847599Z-
-5afdd78bdacb56ab1dad28741ea2a0e47fe41331
Account #1: {9acb9ff906641a434803efb474c96a837756287f}
keystore:///tmp/mykeystore/UTC--2017-04-28T08-46-52.180688336Z-
-9acb9ff906641a434803efb474c96a837756287f
```

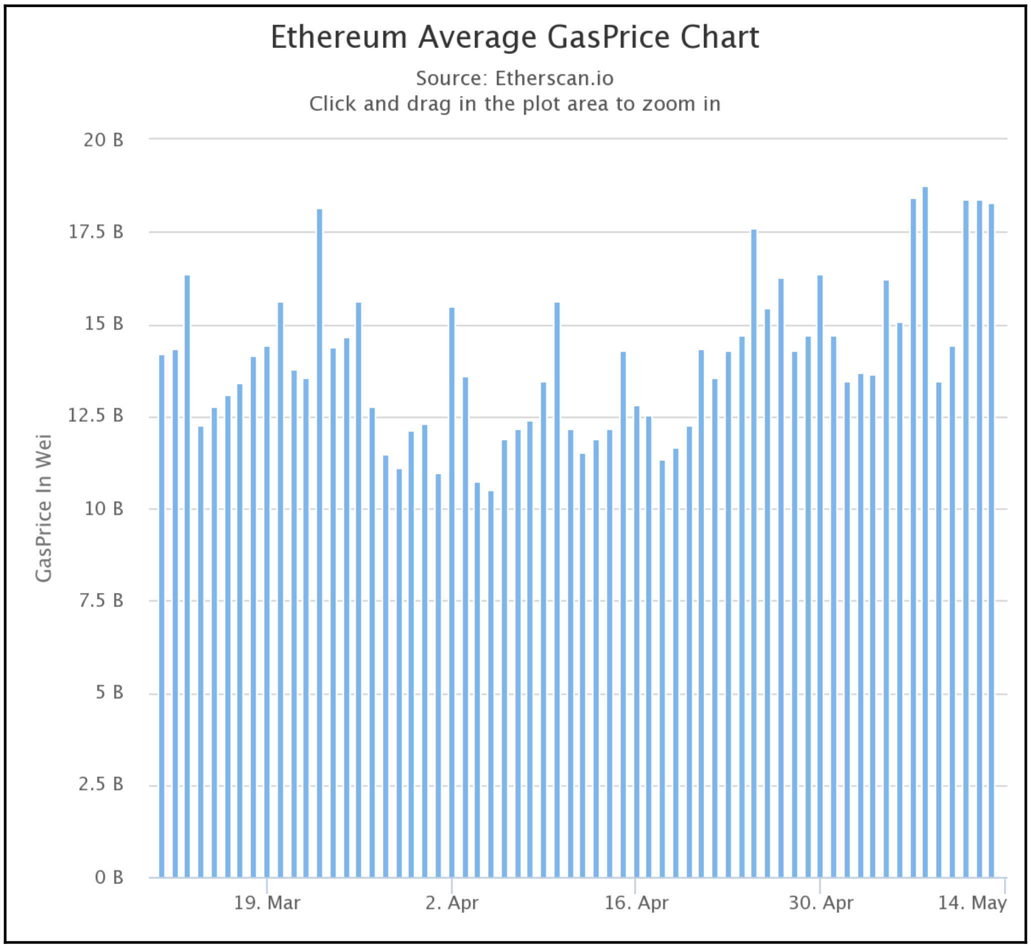
We will be discussing mining and contract development in later chapters.

Ethereum gas

Every transaction on the Ethereum blockchain is required to cover the computation cost; this is done by paying *gas* to the transaction originator. Each of the operations performed by the transaction has some amount of gas associated with it.

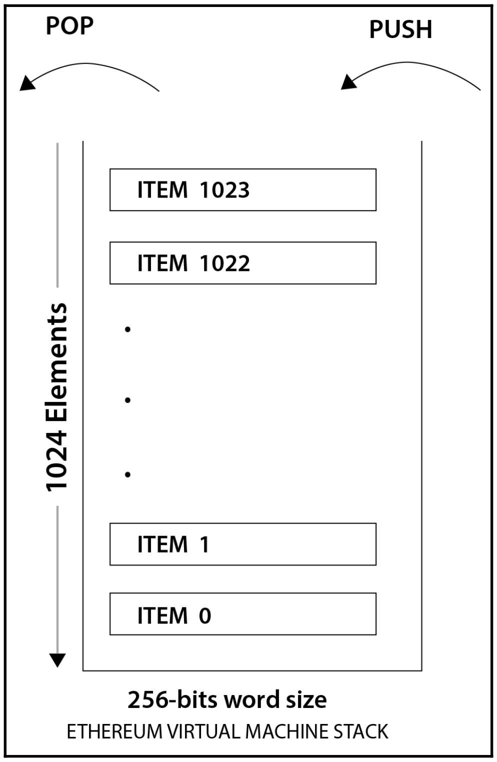
The amount of gas required for each transaction is directly dependent on the number of operations to be performed—basically, to cover the entire computation.

In simple terms, gas is required to pay for every transaction performed on the Ethereum blockchain. The minimum price of gas is 1 Wei (smallest unit of ether), but this increases or decreases based on various factors. The following is a graph that shows the fluctuation in the price of Ethereum gas:



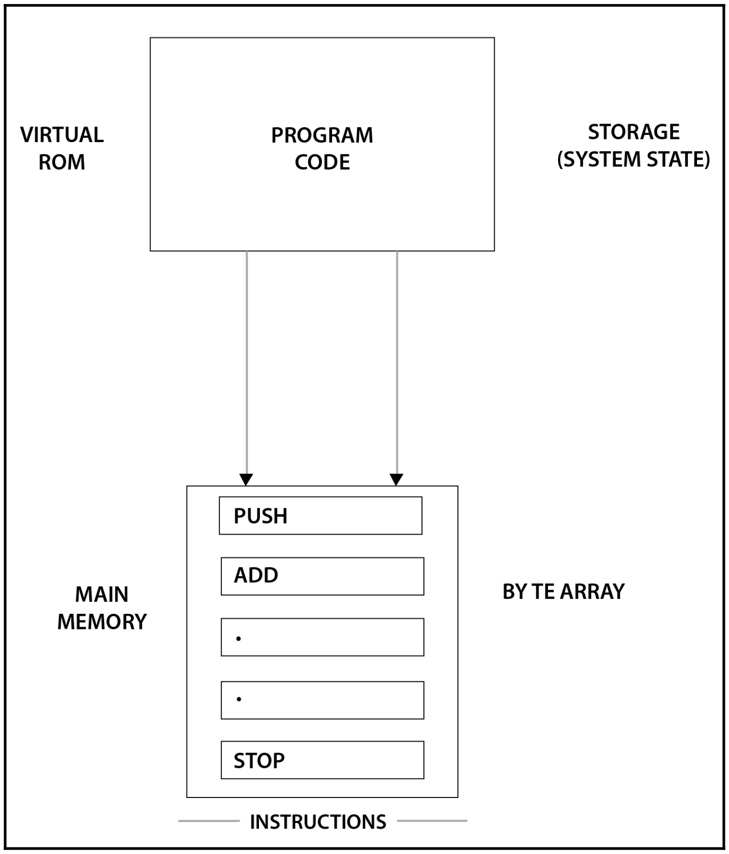
Ethereum virtual machine

Ethereum virtual machine (**EVM**) is a simple stack-based execution machine and acts as a runtime environment for the smart contracts. The word size of EVM is 256-bits, which is also the size limit for each stack item. The stack has a maximum size of 1,024 elements and works on the **Last in First Out** (**LIFO**) queue system. EVM is a Turing-complete machine but is limited by the amount of gas that is required to run any instructions. Gas acts as a propellant with computation credits, which makes sure any faulty code or infinite loops cannot run, as the machine will stop executing instructions once the gas is exhausted. The following diagram illustrates an EVM stack:



EVM supports exception handling in case of an exception occurring, or if there is insufficient gas or invalid instructions. In such cases, the EVM halts and returns an error to the executing node. The exception when gas is exhausted is commonly known as an **Out-of-Gas** (**OOG**) exception.

There are two types of storage available to contracts and EVM: one is memory, and the other is called storage. Memory acts just like RAM, and it is cleared when the code is fully executed. Storage is permanently stored on the blockchain. EVM is fully isolated, and the storage is independent in terms of storage or memory access, as shown in the following diagram:



The storage directly accessible by EVM is **Word Array**, which is non-volatile and is part of the system state. The program code is stored in virtual ROM, which is accessible using the **CODECOPY**, which basically copies the code from the current environment to the memory.

Execution environment

Apart from system state and gas, there are various other elements and information that is required in the execution environment where the execution node must be provided to the EVM:

- Address of the account that owns the execution code.
- Sender address of the transaction that originated the execution.
- Originating address of the execution.
- Gas price of the transaction that initiated the execution.
- Input data or the transaction data, depending on the executing agent type. If the execution node is a transaction, then the transaction data is included as input data.
- Address of the account that initiated the code execution or the transaction sender. This is the address of the sender if the initiation is by a transaction, or else the address of the account.
- Transaction value—this amount is in Wei (the smallest unit of Ether). If the execution agent is a transaction, then it is the value of the transaction.
- The code to be executed, presented as a byte array that the iterator function can execute in cycles.
- Block header of the present block.
- Number of message calls or contract-creation transactions in execution—that is, the number of CALLs or CREATEs being executed in the current cycle of execution.

Ethereum block

The Ethereum blockchain is a collection of required parameters similar to a Bitcoin blockchain; here are the primary elements of an Ethereum block:

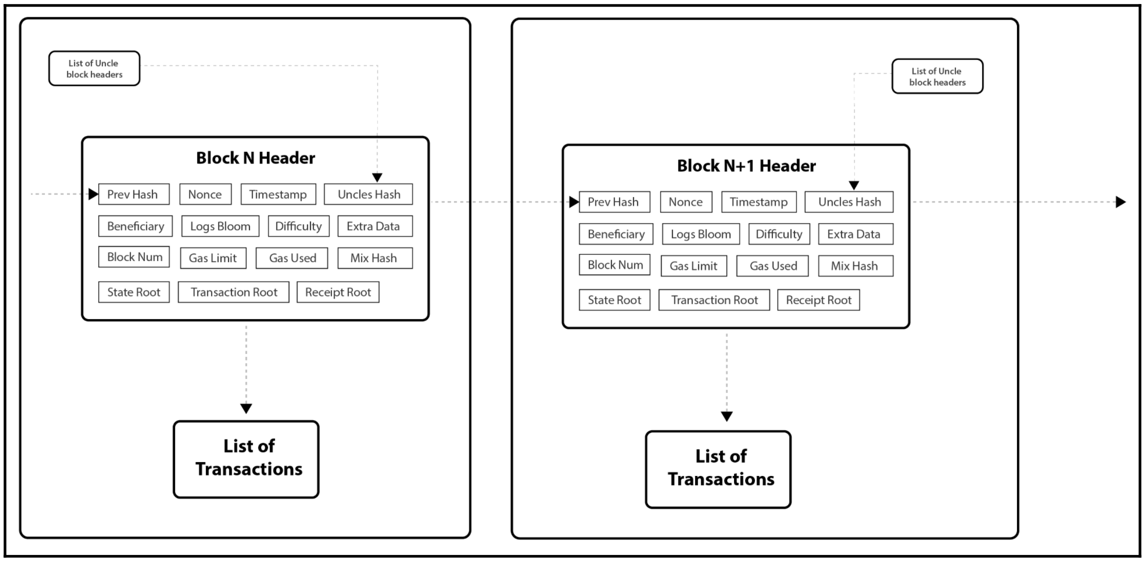
- Block header
- Transaction list
- List of headers of uncles/ommers

Block header

A block header is a collection of various valuable information, which defines the existence of the block in the Ethereum blockchain. Take a look at the following:

- **Parent hash**: The Keccak 256-bit hash of the parent block's header.
- **Ommers hash**: The Keccak 256-bit hash of the ommers (uncles) list of this block.
- **Beneficiary**: The 160-bit address to which all fees from the mining of this block are collected.
- **State root**: The Keccak 256-bit hash of the root node of the state trie. It is calculated once all the transactions are processed and executed.
- **Transactions root**: The keccak 256-bit hash of the root node of the trie structure. The transaction trie represents the list of transactions included in this block.
- **Receipts root**: This is the keccak 256-bit hash of the root node of the trie structure. This trie is formed of receipts of all the transactions included in the block. The receipts are generated after a successful execution of each transaction.
- **Logs bloom**: This is a bloom filter that is composed of the logger address and the log topics from the logs of each transaction receipt of the included transaction list of the block.
- **Difficulty**: This is the difficulty level of each block. This is calculated by the previous block's difficulty and timestamp.
- **Number**: This is the total number of the previous blocks; the genesis block has a number of zero.
- **Gas limit**: The limit set on the gas consumption of the current block.
- **Gas used**: The total gas consumed by the transactions included in the block.
- **Timestamp**: This is the epoch Unix time of the time of the block initialization.
- **Extra data**: This file can be used to store arbitrary data related to this block. It must be 32 bytes max in size.
- **Mixhash**: This contains a 256-bit hash that is combined with the nonce; it is used to prove that sufficient computation power is consumed to mine the block, part of the PoW mechanism.
- **Nonce**: This is a 64-bit hash that is used to prove (along with the mix hash field) that adequate computation effort has been spent to create this block.

The following diagram shows the structure of a block's headers:



Ommers or uncles

Ethereum incentivizes miners to include a list of uncles or ommers when a block is mined, up to to a certain limit. Although in Bitcoin, if a block is mined at the same height, or if a block contains no transaction, it is considered useless; this is not the case with Ethereum. The main reason to include uncles and have them as an essential part of the Ethereum blockchain is that they decrease the chance of an attack occurring by 51%, because they discourage centralization.

Messages

The message is the data and the value that is passed between two accounts. This data packet contains the data and the value (amount of ether). A message can be sent between contract accounts or externally owned accounts in the form of transactions.

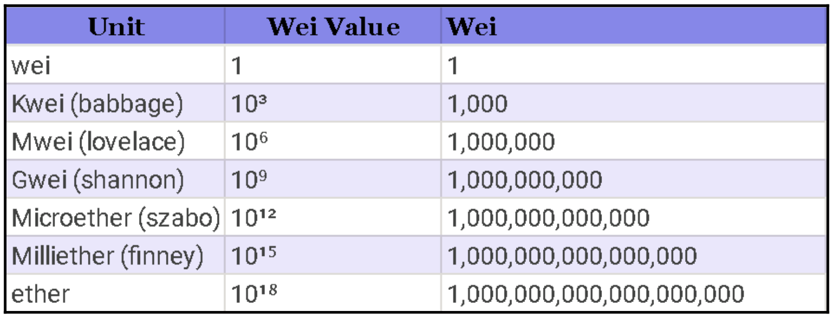
Ethash

Ethash is the **Proof of Work** (**PoW**) algorithm used in Ethereum. It is the latest version of the Dagger–Hashimoto algorithm. It is similar to Bitcoin, although there is one difference—Ethash is a memory-intensive algorithm; hence, it is difficult to implement ASICs for the same. Ethash uses the Keccak hash function, which is now standardized to SHA-3.

Ether

Ether is the main cryptocurrency associated with the Ethereum blockchain; each of the contract accounts can create their own currency, but Ether is used within the Ethereum blockchain to pay for the execution of the contracts on the EVM. Ether is used for purchasing gas, and the smallest unit of Ether is used as the unit of gas.

Since **Wei** is the smallest unit of Wei, here is a table, listing the denominations and the name commonly used for them, along with the associated value:



Unit	Wei Value	Wei
wei	1	1
Kwei (babbage)	10^3	1,000
Mwei (lovelace)	10^6	1,000,000
Gwei (shannon)	10^9	1,000,000,000
Microether (szabo)	10^{12}	1,000,000,000,000
Milliether (finney)	10^{15}	1,000,000,000,000,000
ether	10^{18}	1,000,000,000,000,000,000

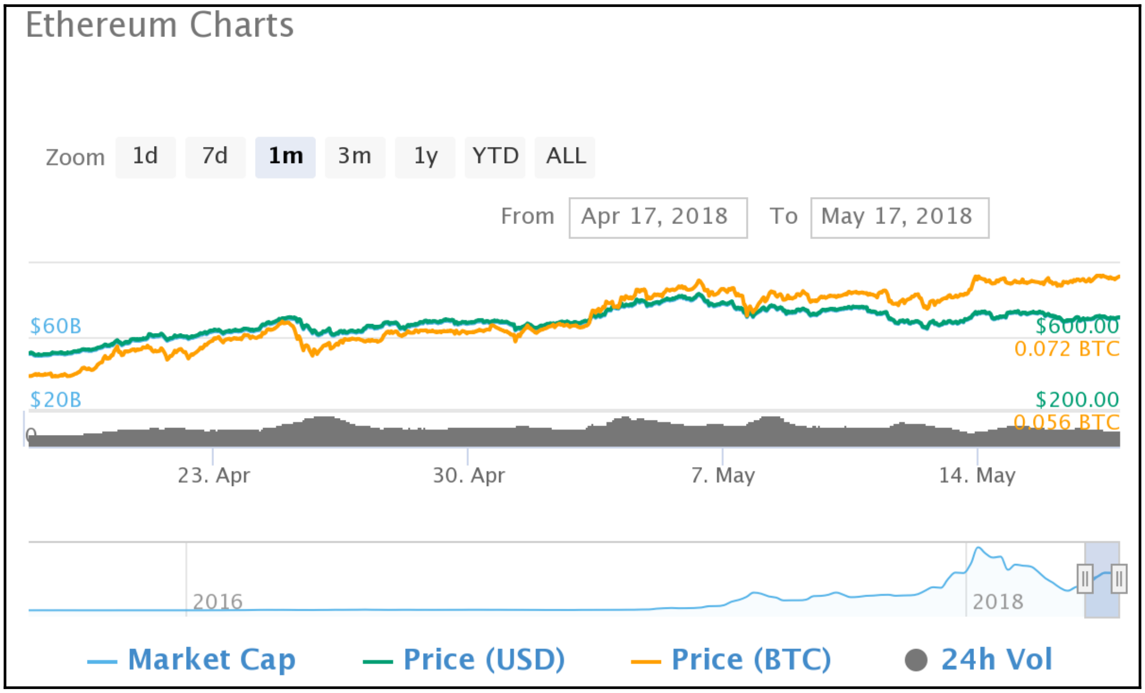
Procuring ether

There are various ways by which Ether can be procured for trading, building smart contracts, or decentralized applications:

- Mining of Ethers, by either joining a mining pool or doing solo mining
- Trade using a crypto exchange platform, and exchange from any other cryptocurrency
- Buying Ether from any fiat currency—there are various exchanges that allow the purchasing of Ether from fiat currency

Trading

Due to its vast community support and major active development, Ether has always been a preferred investment opportunity for everyone. There are more than 500 known exchanges that support the exchange of Ether among other cryptocurrencies or fiat currencies. Here is a price chart, showing the fluctuation of Ether price from April 17, 2018 to May 17, 2018:



Summary

In this chapter, we discussed various components of Ethereum, its execution, network, and accounts, and there was a detailed study of Ethereum's clients. We also discussed gas and EVM, including its environment and how an execution process works. Finally, we discussed the Ethereum block and its block header, the Ethereum algorithm, and the procurement of ether.

In the next chapter, we will learn about **Solidity**, the official and standard language for contract writing on Ethereum blockchain. Learning about Solidity will help us gain a better understanding of smart contract development and deployment.

13
Solidity 101

In the previous chapters, we learned about Ethereum in detail. We read about the Ethereum network, clients, gas, Ethereum virtual machine, and other elements of the Ethereum blockchain. One of the interesting facts about Ethereum is that anyone can create their own blockchain using Ethereum.

Ethereum runs smart contracts on its platform; these are applications that use blockchain technology to perform the required action, enabling users to create their own blockchain and also issue their own alternative cryptocurrency. This is made possible by coding in **Solidity**, which is a contract-oriented programming language used for writing smart contracts that are to be executed on the Ethereum blockchain and perform the programmed tasks.

Solidity is a statically typed programming language that runs on the Ethereum virtual machine. It is influenced by C++, Python, and JavaScript, was proposed in August 2014 and developed by the Ethereum project's solidity team. The complete application is deployed on the blockchain, including smart contract, frontend interface, and other modules; this is known as a **DApp** or a **Decentralized Application.**

We will be covering following topics in this chapter:

- Basics of solidity
- Layout of a Solidity file
- Structure of a smart contract
- Variables and functions
- Value types
- Reference types
- Key to value mapping

Basics of Solidity

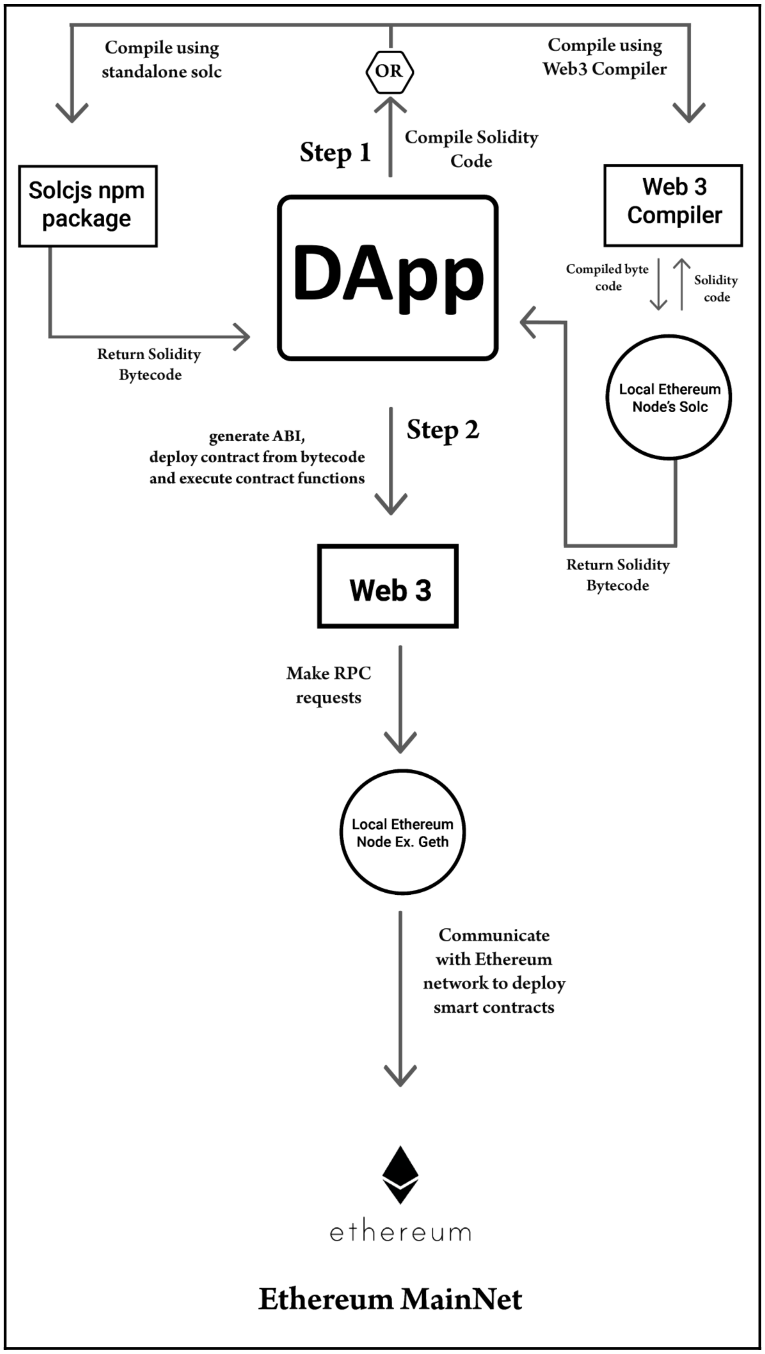
Solidity is not the only language to work on Ethereum smart contracts; prior to solidity, there were other languages that were not as successful. Here is a brief list of languages currently (as of August 2018) compatible with Ethereum:

- **Mutan**: Inspired from Golang and deprecated in March 2015.
- **LLL**: Short for lisp-like language. While it is still supported, it is rarely used.
- **Serpent**: While this language is similar to Python, it is no longer recommended for use.
- **Solidity**: The fourth language introduced by the Ethereum foundation, and so far the most successful language for developing smart contracts. It is the most documented, stable, and has a large community support.
- **Vyper**: Newly introduced, much simpler and easier than Solidity, although it does not have much community support yet. It is influenced by Python.

Solidity is also known as a contract-oriented language, since contracts are similar to classes in object-oriented languages. The Solidity language is loosely based on ECMAScript (JavaScript); hence, a prior knowledge of the same would be helpful in understanding Solidity. Here are some tools required to develop, test, and deploy smart contracts programmed in Solidity:

- **TestNet**: The choice of TestNet to work on, the specified network ID to be used.
- **Compiler**: Choosing the required compiler, for example `solc`, is a solidity compiler; it is included in most of the nodes and also available as a standalone package.
- **Web3.js**: The library that helps in the connection between the Ethereum network and our DApp via HTTP or the IPC network connection.
- **Framework**: It is important to choose a framework from those available for contract compilation and the deployment and other tasks. Some of the frameworks used are Truffle, Embark, Dapple, and others.

Apart from the crucial tools we've already mentioned, there are also various other tools that help in the development of a smart contract to be run on an Ethereum blockchain for tasks such as understanding the contract flow, finding security vulnerabilities, running the test application, writing documentation, and so on. Take a look at the following diagram:


Compile using standalone solc
OR
Compile using Web3 Compiler
Step 1
Compile Solidity Code
Solcjs npm package
DApp
Web 3 Compiler
Compiled byte code
Solidity code
Return Solidity Bytecode
Local Ethereum Node's Solc
Step 2
generate ABI, deploy contract from bytecode and execute contract functions
Web 3
Return Solidity Bytecode
Make RPC requests
Local Ethereum Node Ex. Geth
Communicate with Ethereum network to deploy smart contracts
ethereum
Ethereum MainNet

The basics of working on Solidity

If you program regularly, you are already aware of code editors or **Integrated Development Environments** (**IDEs**). There is a list of integrations available for various IDEs already present; apart from this, Ethereum foundation has also released a browser-based IDE with integrated compiler and a Solidity runtime environment, without the server components for writing and testing smart contracts. It can be found at `remix.ethereum.org`.

Using the compiler

For small and learning-based DApps projects, it is suggested to work on the browser-based compiler by the Ethereum foundation: **Remix**. Another way is to install the Solidity compiler on to your machine. `solc` can be installed from `npm` using the following command:

```
npm install -g solc
```

Solidity can also be built from the source by cloning the Git repository present on the GitHub link: `https://github.com/ethereum/solidity.git`.

Programming in Solidity

In this section, we will be discussing the structure and elements of a Solidity source file; we will discuss the layout, structure, data types, its types, units, controls, expressions, and other aspects of Solidity. The format extension of a solidity file is `.sol`.

Laying out a Solidity file

Solidity is going through active development and has lot of regular changes and suggestions from a huge community; hence, it is important to specify the version of a solidity file at the start of the source file, to avoid any conflict. This is achieved by the Pragma version. This is defined at the start of the solidity file so that any person looking to run the file knows about the previous version. Take a look at this code:

```
pragma solidity ^0.4.24;
```

By specifying a version number, that specific source file will compile with a version earlier or later than the specified version number.

Importing files

Similar to ECMAScript, a Solidity file is declared using the `import` statement as follows:

```
import "filename.sol";
```

The preceding statement will import all the symbols from the `filename.sol` file into the current file as as global statements.

Paths are also supported while importing a file, so you can use `/` or `.` or `..` similar to JavaScript.

Commenting

Single line (`//`) comments and multi-line(`/* ... */`) comments are used, although apart from this there is another type of comment style called **Natspec Comment**, which is also possible; in this type of comment, we either use `///` or `/** ... */`, and they are to be used only earlier function declaration or statements.

Natspec is short for natural specification; these comments as per the latest solidity version (0.4.24) do not apply to variables, even if the variables are public. Here is a small code snippet with an example of such these types of comments:

```
pragma solidity ^0.4.19;

/// @title A simulator for Batman, Gotham's Hero
/// @author DC-man
/// @notice You can use this contract for only the most basic simulation
/// @dev All function calls are currently implement without side effects
contract Batman {
 /// @author Samanyu Chopra
 /// @notice Determine if Bugs will accept `(_weapons)` to kill
 /// @dev String comparison may be inefficient
 /// @param _weapons The name weapons to save in the repo (English)
 /// @return true if Batman will keep it, false otherwise
 function doesKeep(string _weapons) external pure returns (bool) {
 return keccak256(_weapons) == keccak256("Shotgun");
 }
}
```

Tags

They are used in the Natspec comments; each of the tags has its own context based on its usage, as shown in the following table:

Tag	Used for
`@title`	Title for the Smart Contract
`@author`	Author of the Smart Contract
`@notice`	Explanation of the function
`@dev`	Explanation to developer
`@param`	Explanation of a parameter
`@return`	Explanation of the return type

Structure of a contract

Every contract in Solidity is similar to the concept of classes. Contracts can inherit from other contracts, in a fashion similar to classes. A contract can contain a declaration of the following:

- State variables
- Functions
- Function modifiers
- Events
- Struct types
- Enum types

State variables

These are the values that are permanently stored in the contract storage, for example:

```
pragma solidity ^0.4.24;

contract Gotham {
 uint storedData; // State variable
 // ...
}
```

Functions

Functions can be called internally or externally, for example:

```
pragma solidity ^0.4.24;

contract Gotham {
 function joker() public Bat { // Function
 // ...
 }
}
```

Function modifiers

Function modifiers can be used to amend the semantics of functions in a declaration. That is, they are used to change the behavior of a function. For example, they are used to automatically check a condition before executing the function, or they can unlock a function at a given timeframe as required. They can be overwritten by derived contracts, as shown here:

```
pragma solidity ^0.4.24;

contract Gotham {
 address public weapons;

modifier Bank() { // Modifier
 require(
     msg.sender == coins,
     "Only coins can call this."
     );
     _;
 }

    function abort() public coinsbuyer { // Modifier usage
 // ...
 }
}
```

Events

Events allow convenient usage of the EVM, via the frontend of the DApp. Events can be heard and maintained. Take a look at this code:

```
pragma solidity ^0.4.24;

contract Attendance {
      event Mark_attendance(string name, uint ID); // Event

    function roll_call() public marking {
             // ...
                 emit Mark_attendance(Name, ID); //Triggering event
         }
}
```

Types

In Solidity, the type of each variable needs to be specified at compile time. Complex types can also be created in Solidity by combining the complex types. There are two categories of data types in Solidity: **value types** and **reference types**.

Value types

Value types are called **value types** because the variables of these types hold data within its own allocated memory.

Boolean

This type of data has two values, either true or false, for example:

```
bool b = false;
```

The preceding statement assigns `false` to boolean data type `b`.

Operators in Solidity are similar to JavaScript operators, like arithmetic operators, assignment operators, string operators, comparison operators, logical operators, types operators and bitwise operators. These operators can be used with various value types, depending on allowed usage.

Integers

This value type allocates integers. There are two sub-types of integers, that is `int` and `uint`, which are signed integer and unsigned integer types respectively. Memory size is allocated at compile time; it is to be specified using `int8` or `int256`, where the number represents the size allocated in the memory. Allocating memory by just using `int` or `unit`, by default assigns the largest memory size.

Address

This value type holds a 20-byte value, which is the size of an Ethereum address (40 hex characters or 160 bits). Take a look at this:

```
address a = 0xe2793a1b9a149253341cA268057a9EFA42965F83
```

This type has several members that can be used to interact with the contract. These members are as follows:

- `balance`
- `transfer`
- `send`
- `call`
- `callcode`
- `delegatecall`

`balance` returns the balance of the address in units of wei, for example:

```
address a = 0xe2793a1b9a149253341cA268057a9EFA42965F83;
uint bal = a.balance;
```

`transfer` is used to transfer from one address to another address, for example:

```
address a = 0xe2793a1b9a149253341cA268057a9EFA42965F83;
address b = 0x126B3adF2556C7e8B4C3197035D0E4cbec1dBa83;
if (a.balance > b.balance) b.transfer(6);
```

Almost the same amount of gas is spent when we use `transfer`, or `send` members. `transfer` was introduced from Solidity 0.4.13, as send does not send any gas and also does not propagate exceptions. `Transfer` is considered a safe way to send ether from one address to another address, as it throws an error and allows someone to propagate the error.

The `call`, `callcode`, and `delegatecall` are used to interact with functions that do not have **Application Binary Interface** (**ABI**). `call` returns a Boolean to indicate whether the function ran successfully or got terminated in the EVM.

When `a` does call on `b`, the code runs in the context of `b`, and the storage of `b` is used. On the other hand, when `a` does `callcode` on `b`, the code runs in the context of `a`, and the storage of `a` is used, but the code of and storage of `a` is used.

The `delegatecall` function is used to delegate one contract to use another contract's storage as required.

All these members: `call`, `delegatecall`, and `callcode` are not advised to be used unless really necessary, as they tend to break the type-safety of Solidity. It is possible that `callcode` will be deprecated in the near future.

Array value type

Solidity has a fixed and dynamic array value type. Keywords range from `bytes1` to `bytes32` in a fixed-sized byte array. On the other hand, in a dynamic-sized byte array, keywords can contain bytes or strings. `bytes` are used for raw byte data and `strings` is used for strings that are encoded in `UTF-8`.

`length` is a member that returns the length of the byte array for a fixed-size byte array or for a dynamic-size byte array.

A fixed-size array is initialized as `test[10]`, and a dynamic-size array is initialized as `test2[`.

Literal

Literals are used to represent a fixed value; there are multiple types of literals that are used; they are as follows:

- Integer literals
- String literals
- Hexadecimal literals
- Address literals

Integer literals are formed with a sequence of numbers from 0 to 9. Octal literals and ones starting with `0` are invalid, since the addresses in Ethereum start with `0`. Take a look at this:

```
int a = 11;
```

String literals are declared with a pair of double(`"..."`) or single(`'...'`) quotes, for example:

```
Test = 'Batman';
Test2 = "Batman";
```

Hexadecimal literals are prefixed with the keyword `hex` and are enclosed with double (`hex"69ed75"`) or single (`hex'69ed75'`) quotes.

Hexadecimal literals that pass the address checksum test are of `address` type literal, for example:

```
0xe2793a1b9a149253341cA268057a9EFA42965F83;
0x126B3adF2556C7e8B4C3197035D0E4cbec1dBa83;
```

Enums

Enums allow the creation of user-defined type in Solidity. Enums are convertible to and from all integer types. Here is an example of an enum in Solidity:

```
enum Action {jump, fly, ride, fight};
```

Function

There are two types of functions: internal and external functions. Internal functions can be called from inside the current contract only. External functions can be called via external function calls.

Function Modifiers

There are various modifiers available, which you are not required to use, for a Solidity-based function. Take a look at these:

- `pure`
- `constant`
- `view`
- `payable`

The `pure` functions can't read or write from the storage; they just return a value based on its content. The `constant` modifier function cannot write in the storage in any way. Although, the post-Solidity Version 0.4.17 `constant` is deprecated to make way for `pure` and `view` functions. `view` acts just like `constant` in that its function cannot change storage in any way. `payable` allows a function to receive ether while being called.

Multiple modifiers can be used in a function by specifying each by white-space separation; they are evaluated in the order they are written.

Reference types

These are passed on by reference; these are very memory heavy, due to the allocation of memory they constitute.

Structs

A struct is a composite data type that is declared under a logical group. Structs are used to define new types. It is not possible for a struct to contain a member of its own type, although a struct can be the value type of a mapping member. Here is an example of a struct:

```
struct Gotham {

address Batcave;
uint cars;
uint batcomputer;
uint enemies;
string gordon;
address twoface;

}
```

Data location

This specifies where a particular data type will be stored. It works with arrays and structs. The data location is specified using the `storage` or `memory` keyword. There is also a third data location, `calldata`, which is non-modifiable and non-persistent. Parameters of external functions use `calldata` memory. By default, parameters of functions are stored in `memory`; other local variables make use of `storage`.

Mapping

Mapping is used for key-to-value mapping. Mappings can be seen as hash tables that are virtually initialized such that every possible key exists and is mapped to a default value. The default value is all zeros. The key is never stored in a mapping, only the `keccak256` hash is used for value lookup. Mapping is defined just like any other variable type. Take a look at this code:

```
contract Gotham {

    struct Batman {
        string friends;
        string foes;
        int funds;
        string fox;
    }

    mapping (address => Batman) Catwoman;
   address[] public Batman_address;
}
```

The preceding code example shows that `Catwoman` is being initialized as a `mapping`.

Units and global variables

Global variables can be called by any Solidity smart contract. They are mainly used to return information about the Ethereum blockchain. Some of these variables can also perform various functions. Units of time and ether are also globally available. Ether currency numbers without a suffix are assumed to be wei. Time-related units can also be used and, just like currency, conversion among them is allowed.

Summary

In this chapter, we discussed Solidity in detail, we read about the compiler, and we did a detailed study programming in solidity that included studying about the layout of a solidity file, the structure of a contract, and the types of values and reference. We also learned about mapping.

In the next chapter, we will apply our new knowledge from this chapter to develop an actual contract and deploy the same on a test network.

14
Smart Contracts

The concept of smart contracts was first conceived by researcher Nick Szabo in the mid 1990s. In his papers, he described smart contracts as a set of promises, specified in digital form, including protocols within which the parties perform these promises. This description can be broken into four pieces:

- A set of promises
- Digital form
- Protocols for communication and performance
- Performance of actions triggered automatically

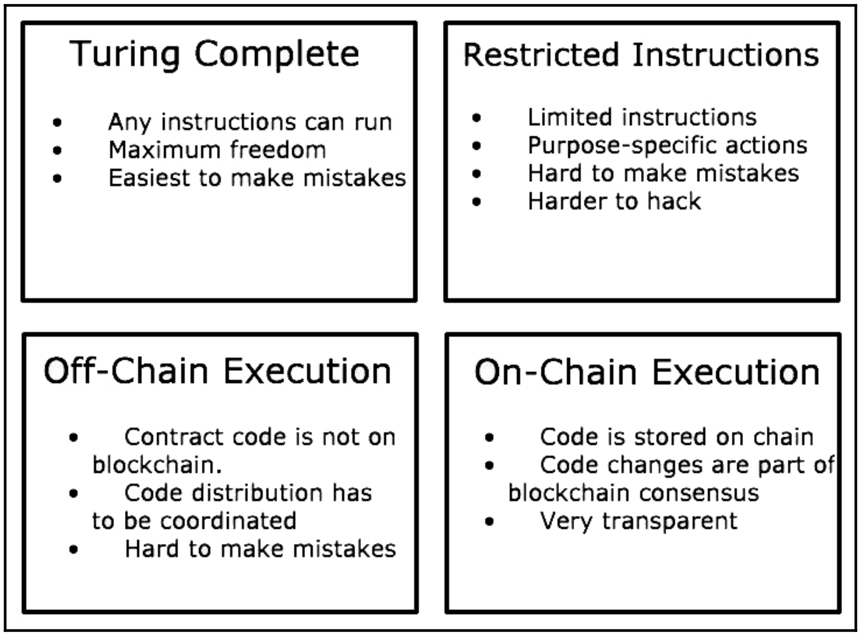
As you can see, nowhere in this is the blockchain directly specified, as blockchain technology had not yet been invented and would not be invented for another 13 years. However, with the invention of blockchain technology, smart contracts were suddenly much more achievable.

Smart contracts and blockchain technology are independent ideas. A blockchain can exist without smart contracts (Bitcoin, for instance, has no real smart contract ability built in), and smart contracts can be built without a blockchain. However, blockchain is a technology particularly well-suited for the development of smart contracts because it allows trustless, decentralized exchange. Essentially, the blockchain provides two out of the four necessary items for smart contracts: digital form and protocols for the communication and performance of actions between distinct parties.

In this chapter, we will go over some of the different blockchain networks and their approaches to smart contract technology. In this chapter, we will cover the following topics:

- Why use smart contracts?
- Approaches to smart contracts
- Limitations of smart contracts

In general, the various smart contract approaches can be divided into different types: **Turing Complete**, **Restricted Instructions**, **Off-Chain Execution**, and **On-Chain Execution**, as shown in the following figure:



The types of smart contracts that are executed on a system determine performance, what can and cannot be executed on the system, the complexity, and of course, the level of security.

Before we go further, let's discuss why smart contracts are desired and even revolutionary.

Why smart contracts?

The world before smart contracts was one that was fraught with uncertainty. Legal contracts, even simple ones, need not be followed, and the cost of recourse using most legal systems was and is extremely expensive, even in countries where the legal system is not corrupt. In many areas of the world, contracts are barely worth the paper they are written on, and are usually enforceable only by parties with substantial political or financial power. For weaker actors in an economic or political system, this is a terrible and unfair set of circumstances.

The issues that we mentioned previously come primarily from the human factor. As long as a person is involved in the enforcement of a contract, they can be corrupt, lazy, misinformed, biased, and so on. A smart contract, in contrast, is written in code, and is meant to execute faithfully no matter what parties are involved. This provides the opportunity for safer, cheaper, faster, and far more equitable outcomes.

Let's look at the key advantages of smart contracts in more depth in the following subsections.

Automating processes and resolutions between parties

The most immediate advantage of smart contracts is that they reduce the labor and pain involved in even successful and faithfully carried out agreements. Take for example, a simple purchase order and invoice between companies. Imagine a company called **FakeCar Inc.** that decides they need 1,000 wheels from their supplier, Wheelmaster. They agree between them that each wheel will cost $20, with payment made when the wheels are delivered to FakeCar. At the beginning, the wheels might be shipped by freight, passing through multiple hands on the way to FakeCar. Once they arrive, FakeCar would need to scan and inspect each wheel, make notes, and then issue a check or wire transfer to Wheelmaster. Depending on the distance involved, the wheels may be in the custody of multiple companies: a trucking company, intercontinental shipping, another trucking company, and finally FakeCar's manufacturing facility. At each stage, there is a chance of damage, loss, or misdelivery. Once delivered, FakeCar would need to issue a transfer to cover the invoice. Even if all goes well, this process can take weeks. In the meantime, both FakeCar and Wheelmaster have to worry whether they will get their wheels or their money, respectively.

Now let's look at how this process might work with smart contracts:

1. FakeCar issues a purchase order on the blockchain for 1,000 wheels at $20 a wheel, valid for 1 month.
2. Wheelmaster issues a shipping request from their suppliers to deliver in one month, and accepts the purchase order.
3. FakeCar funds the purchase order in a smart contract escrow; Wheelmaster can be assured that they will be paid if the wheels arrive.

4. Wheelmaster sees that funds are available to pay for the wheels, ships with a company that tracks each step on a blockchain, and accepts the terms of paying for any wheels that are lost. They (or their insurer) also fund a shipping escrow contract with enough money to cover the event of lost shipping. The contract will automatically refund the shipper once FakeCar signs off on the receipt.
5. The wheels are shipped and delivered, FakeCar's escrow is released, and the insurance bond returns to the shipping company. This happens the moment FakeCar registers receipt and the shipping company signs off on the change in custody.

In this scenario, payments and insurance can be verified and handled instantly—even across international boundaries, and across cultures and languages—if all the parties participate in a blockchain-based ecosystem of smart contracts. The result is a great increase in the certainty of outcomes across all parties, and a subsequent increase in efficiency. For instance, if Wheelmaster can be certain that their invoice will be paid, then they can make business decisions with vastly more efficiency.

Real-world example

As of writing, the first major logistics transaction using blockchain and smart contracts was completed on the Corda blockchain between HSBC and ING, and involved the shipment of soybeans from Argentina to Malaysia. According to the banks, such a transfer used to be very time consuming, and would take five to ten days. With blockchain, the whole issue of finance was handled in under 24 hours.

The use of smart contracts is still in its infancy, and yet the technology has already resulted in an 80–90% reduction in the cross-border friction of financial services. As the technology and surrounding ecosystem improves, the advantages may become yet more extreme.

Increased transparency

As mentioned earlier, one of the negative factors experienced by organizations worldwide is that, for many transactions, trust is a necessity. This is especially true in financial transactions, where purchase orders, invoices, and shipments move between multiple parties. The trust issues here are many. There is a question of not only whether someone will pay, but whether they can pay at all? Do they have a history of on-time payment and, if not, just how bad is their payment history? In many cases, buyers and sellers in any marketplace have very limited information. This is particularly true internationally. This is where blockchain and smart contracts can help.

Ending centralized data

In the United States, each person has a credit score that is calculated by three large credit agencies. These agencies and their methods are opaque. Neither the buyers of this information nor the people who are reported on are allowed deep insight into how the score is calculated, nor are they able to update this information directly. A mistake by a credit agency can be devastating to someone's ability to finance a home or a car, costing a consumer valuable time and money. Nevertheless, if a consumer finds mistakes on their credit report, they must beg the issuer to update it, and they have few options if that organization refuses. Worse, those same issuers have proven bad stewards of the private financial information they collect. For instance, in 2017, Experian suffered a massive data breach that exposed the records of over 100 million people. If these agencies were replaced by a blockchain system and smart contracts, people would be able to see the rules and update records directly, without having to pay an intermediary that may or not be honest themselves.

Large companies have an advantage in the current marketplace: They can both afford to pay these third-party services for financial data, as well as the personnel and systems needed to track information themselves over time. Smaller companies are not granted such economies of scale, putting them at a competitive disadvantage and increasing their overhead, or even putting them out of business if they make a bad decision because they have less information. However, even larger companies stand to benefit, as the cost and expense of compiling this data adds up for them as well. As more data concerning trust becomes public and automated by smart contracts, the playing field will level and, hopefully, will crowd dishonest actors out of the marketplace. This should result in increased confidence across the market, along with reduced overheads and, by extension, higher profits, lower prices, or both.

Increased fairness

In the United States, there used to be a process known as **red lining**, where people of certain ethnic groups were denied loans and access to financial services—particularly mortgages. These unfair practices continue to some extent, as the criteria and process for granting loans and the way interest rates are calculated are hidden inside centralized organizations. This phenomenon is not contained within the USA; there are many areas in the world where ethnic, religious, and other biases distort what are meant to be objective decisions. With a smart-contract-based system, the rules would be public and auditable to ensure fairness and accuracy.

Smart contract approaches

One approach to smart contracts is to allow full-featured software to be embedded either inside or alongside a blockchain, able to respond to blockchain events. This is an approach taken by Hyperledger Fabric, Ethereum, NEO, and other such companies. This approach gives maximum flexibility, as there is essentially nothing that cannot be written into the blockchain system. The downside of this power is the risk of making errors. The more options available, the more possible edge cases and permutations that must be tested, and the higher the risk that there will be an undiscovered vulnerability in the code.

The other approach to smart contracts is to greatly reduce the scope of what is possible in return for making things more secure and costly mistakes more difficult. The trade-off is currently flexibility versus security. For instance, in the Stellar ecosystem, smart contracts are made as sets of operations. In Stellar, there are only eleven operations:

- Create account
- Payment
- Path payment
- Manage offer
- Create passive offer
- Set options
- Change trust
- Allow trust
- Account merge
- Inflation
- Manage data

These operations themselves have multiple options and permutations, and so enable quite a large amount of behavior. However, it is not possible to easily use these operations to execute something such as the DAO, or some other on-chain governance organization. Instead, such functionality would have to be hosted off the chain. Similarly, there is no clear way in Stellar to manage the equivalent of ERC-721 tokens, which would track the equivalent of something such as trading cards or even pieces of real estate. Stellar's smart contract system is geared toward the transfer of fungible assets, such as currencies. As a result, it can scale very quickly, easily handle multisignature accounts and escrow, and process transactions in just a few seconds with high throughput. Ethereum is more flexible, but the multisignature capability, the tokens themselves, and so on would need to be created with software written in Solidity. Ethereum is obviously more flexible, but requires more code, and thus runs a higher risk of defects.

Example Ethereum smart contracts

The blockchain with the most widespread use of smart contracts is Ethereum. Of all the smart-contract-capable networks presented here, it is not only the one with the largest use, but also has the largest ecosystem of public distributed applications. One of the reasons that Ethereum is so popular is that its representation of smart contracts is relatively intuitive and easy to read. In this section, we are going to look at a common Ethereum-based smart contract that fulfills all four of the preceding criteria and is relatively easy to understand: a token sale contract. The following code will be written in Solidity; for more details, please see `Chapter 13`, *Solidity 101*, and `Chapter 15`, *Ethereum Development*.

The promises

The first aspect of a smart contract is that it must make a set of programmatic promises. The reason we have chosen a token sale contract to look at is that it has a very simple promise to make: if you send the contract Ethereum, the contract will in turn automatically send your account a new token. Let's look at some basic code, which is explicitly not for production; this is simplified code to make certain concepts clearer. This code comes from the `StandardToken` contract, part of the OpenZeppelin (You'll find a link for the same in the *References* section) project on which this is based, which has full-featured and audited code to achieve the same effect, but is more complicated to understand.

First, here is an interface contract for an `ERC20` token, which we will save as a file called `ERC20.sol`:

```
pragma solidity ^0.4.23;
interface ERC20 {
 function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view returns
(uint256);
  function transferFrom(address from, address to, uint256 value) public
returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed _owner, address indexed _spender, uint256
_value);
}
```

Next, we will reference that token interface in our crowdsale contract, which will send an `ERC20` token in response to a payment in ether:

```
pragma solidity ^0.4.23;
import "./ERC20.sol";

contract Crowdsale {

 // The token being sold, conforms to ERC20 standard.
 ERC20 public token;

 // 1 tokens per Eth, both have 18 decimals.
 uint256 public rate = 1;

 constructor(ERC20 _token) public {
    token = _token;
 }

 function () external payable {
    uint256 _tokenAmount = msg.value * rate;
    token.transfer(msg.sender, _tokenAmount);
 }
}
```

This is a very simplified contract, but again, it is not sufficient for a complete, real-world `Crowdsale`. However, it does illustrate the key concepts for a smart contract. Let's look at each piece. The `constructor` method requires a reference to an `ERC20` token, which is the token that will be given to buyers who send in Ethereum, as shown in the following code:

```
constructor(ERC20 _token) public {
    token = _token;
}
```

Because of the way Solidity works, this contract cannot function unless a token has been loaded. So this is the first promise implicitly made by this code: there must be an `ERC20` token available for purchase. The second promise is the conversion rate, which is placed at the very simple 1. For each wei (the smallest unit of currency in Ethereum), a person buying this token will get 1 unit of the new token. Ethereum has 18 decimal places, and by convention so do most tokens, so it would be presumed that this would make the conversion of Ethereum to this token now 1:1. This brings us to item #4 in the necessary aspects of a smart contract: automatic fulfillment. The following code handles this:

```
function () external payable {
    uint 256 _tokenAmount = msg.value * rate; //Calculate tokens purchased
    token.transfer(msg.sender, _tokenAmount); //Execute send on token
contract.
```

```
    }
```

As this is code, the requirement that the smart contract should be in digital form is obvious. The automatic aspect here is also straightforward. In Ethereum, `msg.value` holds the value of the ether currency that is sent as part of the command. When the contract receives Ethereum, it calculates the number of tokens the purchaser should receive and sends them: no human interaction needed, and no trusted party necessary or possible. Similarly, no one can intervene, as once it is deployed to the network, the code in Ethereum is immutable. Therefore, a sender who is using this smart contract can be absolutely assured that they will receive their tokens.

Security considerations

It is important to understand smart contracts in the domain in which they live: decentralized, asynchronous networks. As a result of living in this ecosystem, there are security considerations that are not always obvious, and can lead to issues. To illustrate, we are going to look into two related functions of the `ERC20` standard: `approve` and `transferFrom`. Here is code for the `approve` function from OpenZeppelin:

```
function approve(address _spender, uint256 _value) public returns (bool) {
   allowed[msg.sender][_spender] = _value;
   emit Approval(msg.sender, _spender, _value);
   return true;
 }
```

The `approve` function allows a token owner to say that they have approved a transfer of their token to another account. Then, in response to different events, a future transfer can take place. How this happens depends on the application, but such as the token sale, by approving a transfer, a blockchain application can later call `transferFrom` and move the tokens, perhaps to accept payment and then perform actions. Let's look at that code:

```
function transferFrom(address _from,address _to,uint256 _value) public
returns (bool) {
   require(_to != address(0)); // check to make sure we aren't transfering
to nowhere.

   // checks to ensure that the number of tokens being moved is valid.
   require(_value <= balances[_from]);
   require(_value <= allowed[_from][msg.sender]);

   // execute the transfer.
   balances[_from] = balances[_from].sub(_value);
   balances[_to] = balances[_to].add(_value);
   allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
```

```
        //Record the transfer to the blockchain.
        emit Transfer(_from, _to, _value);
        // let the calling code or app know that the transfer was a success.
        return true;
    }
```

The two functions work together. The user wishing to use the app uses `approve` to allow payment, and the app calls `transferFrom` in order to accept. But because of the asynchronous nature of the calls, it is possible for flaws to exist.

Imagine an app where users can pay tokens in order to join a digital club—40 tokens for a basic membership and 60 tokens for an enhanced membership. Users can also trade the tokens to other people or sell them as they wish. The ideal case for these two functions is where a user approves 40 tokens and the application registers this and calls `transferFrom` to move the 40 tokens, and then grants access as part of the smart contract. So far so good.

It's important to keep in mind that each action here takes time, and the order of events is not fixed. What actually happens is that the user sends a message to the network, triggering `approve`, the application sends another message, triggering `transferFrom`, and then everything resolves when the block is mined. If these transactions are out of order (`transferFrom` executing before `approve`), the transaction will fail. Moreover, what if the user changes their mind and decides to change their approval from 40 to 60? Here is what the user intends:

1. **User**: `approve` 40 (block 1)
2. **User**: `approve` 60 (block 1)
3. **App**: `transferFrom` 60 to App (block 1)
4. **App**: Grant enhanced membership (block 2)

In the end, the user paid 60 tokens and got what they wanted. But because each of these events are asynchronous and the order is decided by the miners, this order is not guaranteed. Here, is what might happen instead:

1. **User**: `approve` 40 (block 1)
2. **App**: `transferFrom` 40 to App (block 1)
3. **User**: `approve` 60 (block 2, as the miners did not include it in block 1)
4. **App**: `transferFrom` 60 to App (Block 2)

Now the user has paid 100 tokens without meaning to. Here is yet another permutation:

1. **User**: `approve` 40 (block 1)
2. **User**: `approve` 60 (block 1)
3. **App**: `transferFrom` 40 to app (block 2)
4. **App**: Grants basic membership (block 2)
5. **App**: `transferFrom` 60 to app (block 3) | fails

At the end of this sequence, the user still has 20 tokens approved, and the attempt to get the enhanced membership has failed. While an app can and should be written without these issues by doing such things as allowing upgraded membership for 20 tokens and checking the max approval before `transferFrom` is called, this attention to detail is not guaranteed or automatic on the part of application authors.

The important thing to understand is that race conditions and ordering issues are extremely important in Ethereum. The user does not control the order of events on a blockchain, nor does an app. Instead, it is the miners that decide which transactions occur in which blocks and in which order. In Ethereum, it is the gas price that affects the priority that miners give transactions. Other influences can involve the maximum block gas limit, the number of transactions already in a block, and whether or not a miner that successfully solves a block has even seen the transaction on the network. For these reasons, smart contracts cannot assume that the order of events is what is expected.

Dealing with threats in smart contracts

Every decentralized network will have to deal with race conditions caused by different orderings. It is critical that smart contracts be carefully evaluated for possible race conditions and other attacks. To know whether a race condition bug is possible is as simple as knowing whether more than one function call is involved, directly or indirectly. In the preceding case, both the user and the app call functions; therefore, a race condition is possible, and so is an attack called front running. It is also possible to have race conditions inside a single method, so smart contract developers should not let their guard down.

Each network has a different model for contract execution, and as a result, each network had different best practices. For Ethereum, Consensys maintains a list of smart contract best practices at `https://consensys.github.io/smart-contract-best-practices/`.

Before shipping any smart contract, it is strongly suggested that an organization write extensive unit tests and simulation tests, and then audit the smart contracts against the best practices for that network.

Limitations of smart contracts

Smart contracts hold tremendous power, but they do have limitations. It is important to note that these systems are only as good as the people building them. So far, many smart contract systems have failed due to unforeseen bugs and events that were not part of the initial design. In many cases, these were merely technical flaws that can at least be fixed in time. However, with the recent rush to use blockchain technology for everything, we are likely to start seeing more substantial failures as people fail to understand the limits of the technology. For blockchain to truly have maximum business impact, both its advantages and limitations have to be addressed.

Data quality and mistakes

Like all systems, smart contracts are only as good as the data they act on. A smart contract that receives bad or incorrect information from the network will still execute. On blockchain systems, this can be a huge issue as most transactions initiated by a human or a contract are irrevocable. Thus, if information is placed on a blockchain that is in error, fraudulent, or has some other deficiency, then a smart contract will still execute faithfully. Instead of expediting the proper functioning of the network, the smart contract would now be assisting in propagating an error.

To use the earlier example of shipping tires between FakeCar and Wheelmaster, what if during transit the boxes holding the tires were broken into and the tires replaced? If the worker at the FakeCar building scanned the boxes as received without checking each and every one, the smart contract would see this update and release escrow. The shipper would have their insurance bond returned, Wheelmaster would get paid, and FakeCar would still no longer have the wheels they ordered. To smart contract purists, this is how things should be. But in these cases, companies may instead refuse to use smart contracts or require additional layers of approval—essentially recreating the systems of old.

In designing smart contract systems, it is therefore critical that designers try and imagine every possible way things could go wrong. As with the DAO and other smart contract systems that have been used so far, small mistakes can have big consequences.

Many smart contracts involve some level of human interaction. For instance, multisignature wallets require multiple people to authorize a transaction before they will execute. These touchpoints introduce the same possibility for errors as old systems, but with the possibility of irrevocable consequences.

Legal validity

Smart contracts do what they are programmed to do. If a smart contract is deemed invalid in a court, how is this resolved? The answer right now is that nobody really knows, but it could happen—and probably will. Most countries in the world have limits on what can and cannot be contractually agreed to and the terms that can be legally used in a contract. For instance, in the USA, there is a limit to the amount of interest that can be charged on certain financial products. Other regulations control the conditions and terms of payment in specific industries. Smart contracts that violate local and national laws run the risk of being canceled, resulting in repayment, damages, or other consequences to the participating organizations, and possibly even the contract authors.

Stability of meaning

In the token sale contract we looked at earlier, a user can be sure that they will receive the tokens they purchase. What they cannot be sure of is that those tokens will be valuable or still be useful in the future. Moreover, if those tokens represent something else (access to a system, real-world assets, or something else), then the mere existence of the tokens does not bring any guarantees that this access will remain, that people will continue to accept the tokens for assets (see the previously mentioned issues with legal validity), and so on. With national currencies, the use and acceptance of that currency is mandated by a government with substantial power. With tokens, the acceptance and use of the token has no mandate. To some, this is the very appeal—that the value of a token is more trustable because it is built not on enforcement by a government, but by social approval and use.

It is likely that over time, legal frameworks and trade will become more stable, and this will be less of an issue.

Summary

Smart contracts are agreements written into code between different parties. The critical aspects of smart contracts is that they contain promises that are in digital form. All of these promises can be executed using digital protocols for communication performance. The outcomes of the contracts are triggered automatically.

At this point, you should have a solid understanding of what smart contracts are, how they work, and their strengths and limitations. You should be able to understand the dangers inherent in smart contract ecosystems and be able to gauge possible risks in the development of smart-contract-based systems. At a minimum, you should recognize the need for careful and thorough evaluation of smart contracts for security reasons. Remember, with smart contracts, the code is executed with little or no human intervention. A mistake in a smart contract means the damage done by the mistake will multiply as fast as the code can be run.

Next, we are going to dive more deeply into smart contracts with a chapter devoted to development in Ethereum.

References

1. `http://firstmonday.org/ojs/index.php/fm/article/view/548`
2. Nick Szabo, Smart Contracts: Building Blocks for Digital Markets, 1996
3. `https://www.cnbc.com/2018/05/14/hsbc-makes-worlds-first-trade-finance-transaction-using-blockchain.html`
4. `http://fortune.com/2017/12/22/experian-data-breach-alteryx-amazon-equifax/`
5. `https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol`

15
Ethereum Development

For the past few chapters, we have been learning about Ethereum blockchain, its uses, and how it has shaped the decentralized technology, not for just currency based uses but also for other industry verticals. Further, we learned about development on top of the Ethereum blockchain, using smart contracts.

In the previous chapter, we discussed Solidity—the programming language introduced by the Ethereum foundation. Solidity is the language that makes it possible to create decentralized applications on top of Ethereum blockchain, either to be used for creating another cryptocurrency token or for any other use case in which blockchain can have an essential role.

In this chapter, we will be specifically working on Ethereum development, which constitutes creating a token that represents a smart contract. We will be creating a coin or a token that will run on the Ethereum blockchain platform. We will be learning about the development, testing, and deployment of the token using Solidity and, along with this, we will be discussing important aspects of the smart contract development phase, which will help in using smart contracts for other applications and industries.

In brief, we will be covering the following topics:

- Preparing for smart contract development
- Working on smart contracts

Introduction

In this chapter, we will go ahead and create a token on top of Ethereum blockchain. We will also briefly look into the development of DApp, which will use the smart contract to its full potential. Here is a list of tools required for the development process:

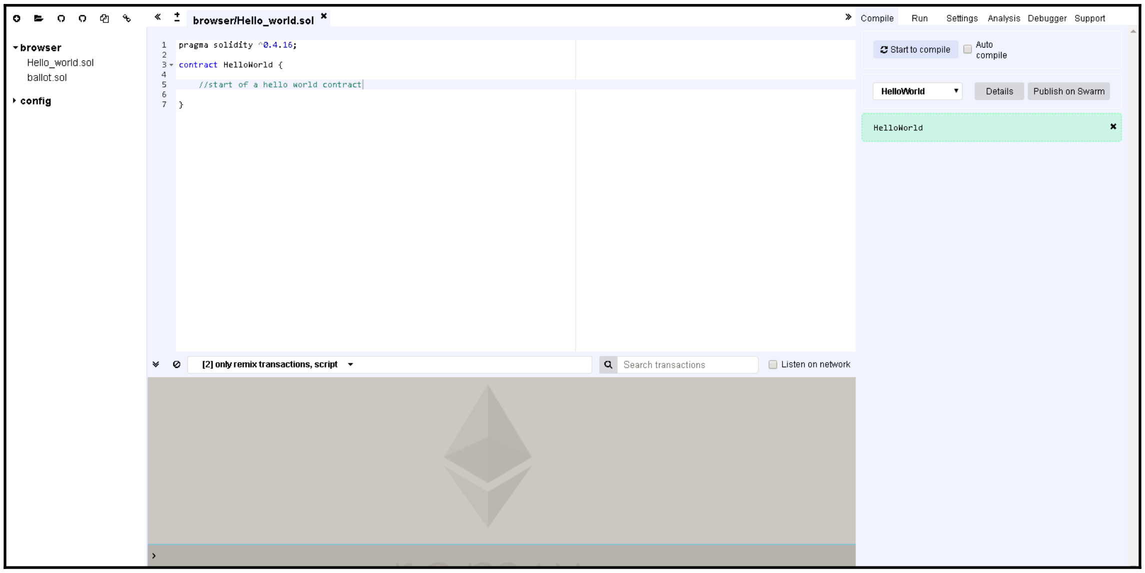
- **Code Editor or IDE**: Just as in the previous chapter, pick a code editor you are comfortable using; in this chapter, we are choosing Sublime text. Although, the official online IDE remix can also be used.
- **Ethereum Wallet**: This is required for the deploying and testing of a smart contract on the TestnNet.
- **Solc compiler**: This is for the compiling of a smart contract that is converting Solidity language code into bytecode for EVM to read.
- **Truffle framework**: This makes it easier to test and deploy the smart contracts.
- **Web3**: This is a JavaScript library and is used to communicate with the Ethereum node; for example, we have a frontend interface in our application for users to interact with the blockchain, then `Web3` allows the user interface to interact with the contracts deployed over the blockchain.

Preparing for smart contract development

Before we dive into development of smart contracts and further testing and deployment procedures, it is important to be ready with the tools and other modules, along with the preparation of the development process that will help create the desired results.

Remix

Remix is currently the official IDE available online; if you are more comfortable in another IDE, you can check whether Solidity is a supported language and start working on the same. The following is a screenshot of what Remix looks like:



```
pragma solidity ^0.4.16;

contract HelloWorld {

    //start of a hello world contract

}
```

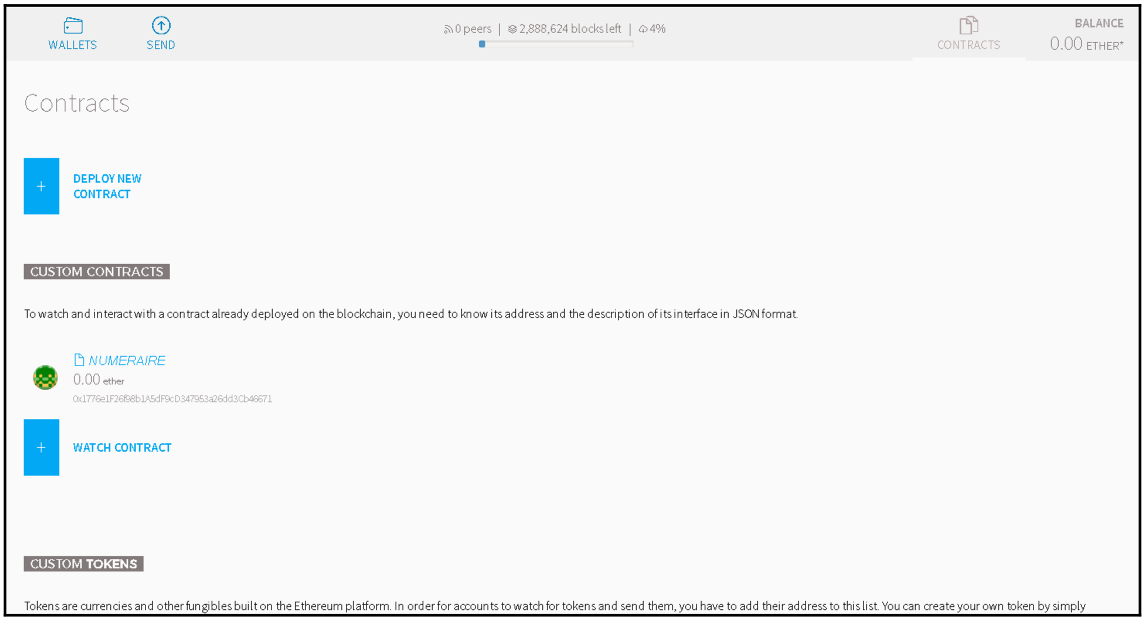
Remix is a fully functional online IDE, with the required features such as the compiling of code, running, connecting with localhost, publishing code on GitHub, and so on.

When Remix is opened for the first time, the `Ballot` contract is loaded.

Geth and Mist

We can run test nodes using Geth, although the task of connecting and mining of Ether in `TestNet` or `private net` can also be done using the official Ethereum client, which is a Mist browser working on top of a Geth node.

The Mist client has various features such as being able to create Ethereum accounts and connecting with `MainNet` or `TestNet`. We can deploy or watch an already-deployed contract on blockchain using Mist. The following screenshot illustrates what Mist looks like:



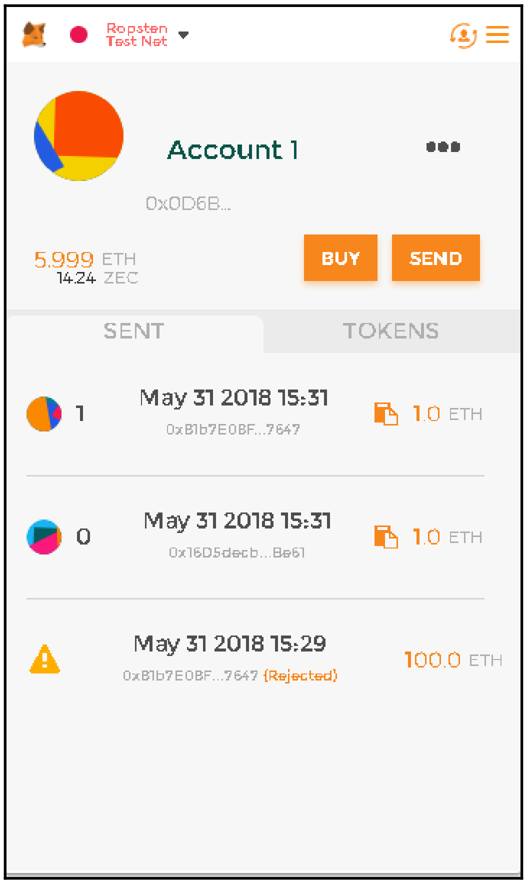
To watch and interact with a contract already deployed on the blockchain, you need to know its address and the description of its interface in JSON format.

Tokens are currencies and other fungibles built on the Ethereum platform. In order for accounts to watch for tokens and send them, you have to add their address to this list. You can create your own token by simply

MetaMask

MetaMask is an Ethereum Wallet and an Ethereum browser. MetaMask enables us to interact with DApps and smart contracts via our web browser without downloading the complete Ethereum blockchain. It is available as a browser extension for Chrome, Firefox, and other browsers.

It lets websites fetch data from blockchain and also allows users to send transactions from the extension itself. With MetaMask, it becomes easier to interact with Ethereum DApp within the browser. The following screenshot is an example of an Ethereum Wallet:



Token standard

With Ethereum, decentralized applications can be created, and we can also issue tokens on the Ethereum blockchain. Just such as **Bitcoin Improvement Proposals** (**BIPs**), there is an online list of **Ethereum Improvement Proposals** (**EIPs**) hosted on GitHub, and this can be found here: `https://github.com/ethereum/EIPs` . In EIP-20, a new token standard was introduced around the end of 2015. This token standard is known as **ERC20**. *ERC* stands for **Ethereum Request for Comments.** ERC is authored by a community developer; post the community and core team approval, the proposal becomes a standard. There are various other standards such as ERC223 and ERC721 that are also being used in addition to ERC20.

There are methods defined in the EIP that must be followed so that the token can be a part of the ERC20 token contract.

Methods in ERC20

Here are the methods specified in the ERC20 contract type; it is important to follow the given methods/events so that the contract can be in the family of ERC20. Although some of these specifications are options and not required to be there. Take a look at this list:

- `name`: It should return the name of the contract, for example: `BaToken`.
- `symbol`: Returns the symbol of the token, for example: `BATN`.
- `decimals`: Returns the number of decimals that the token uses, for example: 18.
- `totalSupply`: Returns the total supply of the tokens. It is a compulsory method.
- `balanceOf(address _owner)`: Returns the account balance of another balance with an address `_owner`. It is a compulsory method.
- `transfer(address _to, unit256 _value)`: It sends the token of `_value` amount to address `_to` . It is a compulsory method.
- `transferFrom(address _from, address _to, uint256 _value)`: This method transfers the `_value` amount of tokens from address `_from` to address `_to`. It is compulsory to have this method in the smart contract for it to be a valid ERC20 contract.

- `approve(address _spender, uint256 _value)`: This method allows the address `_spender` to withdraw from the given account multiple times; the maximum amount it can withdraw is `_value`. It is also a compulsory method; allowance is suggested to first be set to 0 so that any vulnerability is not there.
- `allowance(address _owner, address _spender)`: It returns the amount that `_spender` is still allowed to withdraw from `_owner`. It is compulsory to have this method in the smart contract.
- `event Transfer(address indexed _from, address indexed _to, uint256 _value)`: This event must be triggered when tokens are transferred, even if the value of tokens being transferred is zero.
- `event Approval(address indexed _owner, address indexed _spender, uint256 _value)`: This event is also compulsory and is called whenever the `approve` method is called.

Web3

`Web3.js` is a JavaScript Library; it can communicate with the Ethereum node through JSON RPC calls, by connecting to the HTTP or IPC connection. `Web3.js` works by exposing methods that have been enabled over the RPC.

`Web3` can be installed via `npm` using the following command:

```
npm install web3
```

Truffle framework

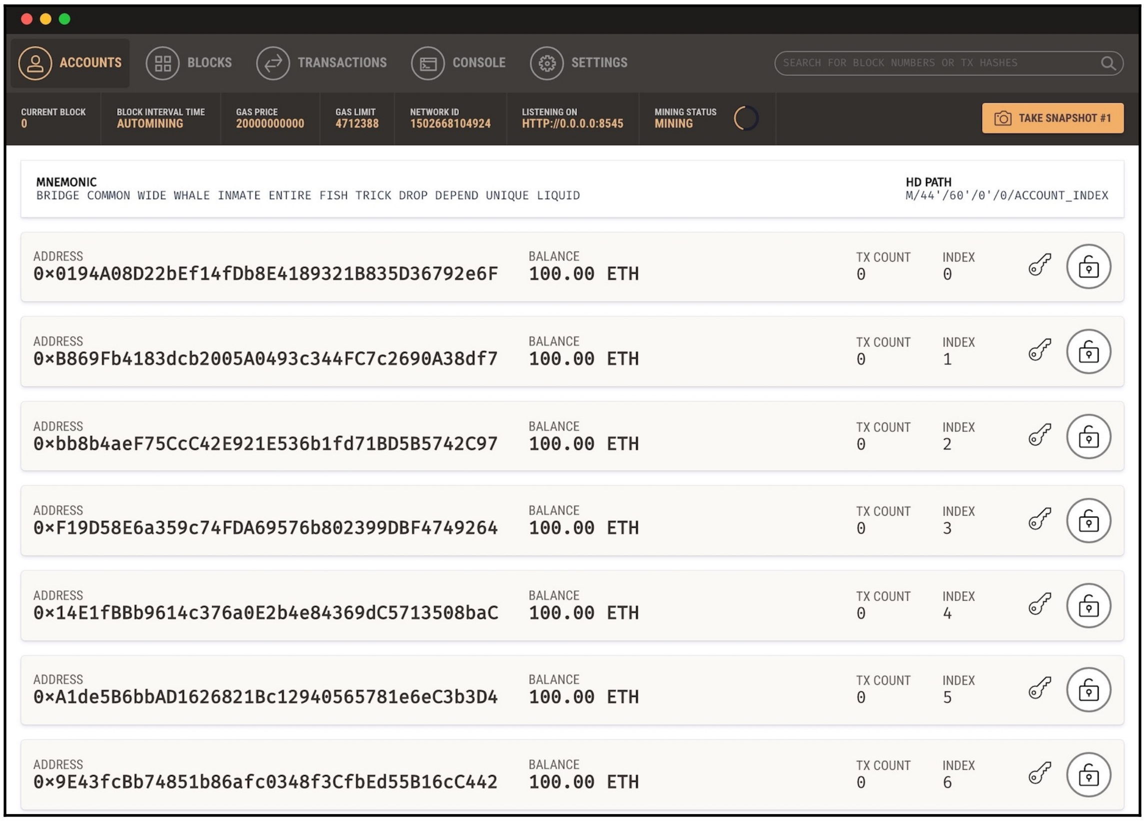
Truffle is a development framework that makes it simple to test and deploy Ethereum smart contracts. Truffle can be installed via `npm` using the following command:

```
npm install -g truffle
```

Truffle helps in contract compilation and linking along with automated testing framework using Mocha and Chai. With Truffle, we can easily deploy contracts to any `estNet`, MainNet, or private Network ID. Truffle does everything in a DApp, such as the compiling of contracts, injecting them into the user interface and testing to check for vulnerabilities.

Ganache

Ganache is a tool introduced by the Truffle Foundation to run tests on the Ethereum blockchain created. It has a command-line interface, but a graphical interface is also available when required. The following screenshot depicts the **Ganache** tool:



CURRENT BLOCK	BLOCK INTERVAL TIME	GAS PRICE	GAS LIMIT	NETWORK ID	LISTENING ON	MINING STATUS
0	AUTOMINING	20000000000	4712388	1502668104924	HTTP://0.0.0.0:8545	MINING

MNEMONIC
BRIDGE COMMON WIDE WHALE INMATE ENTIRE FISH TRICK DROP DEPEND UNIQUE LIQUID

HD PATH
M/44'/60'/0'/0/ACCOUNT_INDEX

ADDRESS	BALANCE	TX COUNT	INDEX
0×0194A08D22bEf14fDb8E4189321B835D36792e6F	100.00 ETH	0	0
0×B869Fb4183dcb2005A0493c344FC7c2690A38df7	100.00 ETH	0	1
0×bb8b4aeF75CcC42E921E536b1fd71BD5B5742C97	100.00 ETH	0	2
0×F19D58E6a359c74FDA69576b802399DBF4749264	100.00 ETH	0	3
0×14E1fBBb9614c376a0E2b4e84369dC5713508baC	100.00 ETH	0	4
0×A1de5B6bbAD1626821Bc12940565781e6eC3b3D4	100.00 ETH	0	5
0×9E43fcBb74851b86afc0348f3CfbEd55B16cC442	100.00 ETH	0	6

Working on a smart contract

Let's dive into the task of creating a smart contract, along with the process of testing the smart contract on `TestNet`. The easiest way to run the code discussed in the following is on Remix. Just follow these steps:

1. When you open Remix in the browser, by default, it opens the `ballot.sol` file; you can create a new file and start editing your first smart contract. Take a look at this:

```
pragma solidity ^0.4.24;

//This is a test comment, details about the contract can be added
here
/*details like Total supply, contract address, Name, Symbol an
decimals which will help someone knowing about the contract instead
of finding these details within the source code

*/

contract Gotham{

 string public name;
    string public symbol;
    uint8 public decimals;
    //most suggested decimal places in 18
   uint256 public totalSupply;


}
```

2. In the previous code snippet, we have created a contract with Solidity Version 0.4.24 and with the name of `Gotham`. Now let's create a constructor that will initialize the variables we have defined. This is how our `contract` code should look now:

```
contract Gotham{

    string public name;
    string public symbol;
    uint8 public decimals;
    //most suggested decimal places in 18
    uint256 public totalSupply;

//Constructor
```

```
    constructor() public {
        symbol = "GOTH";         //This is the symbol of our Token
        name = "GothCoin";       //This is the name of our Token
        decimals = 12;           /* These are the number of decimal
places it can have,  it is suggested to have 18 decimal places */
        totalSupply = 100000;    //total supply of coins allowed
        }

}
```

3. Now, let's add the standard ERC20 methods to a separate contract; these are required to be present in an Ethereum smart contract. Take a look at this code:

```
contract ERC20 {

     function totalSupply() public constant returns (uint);
     function balanceOf(address tokenOwner) public constant returns
(uint balance);
     function allowance(address tokenOwner, address spender) public
constant returns (uint remaining);
     function transfer(address to, uint tokens) public returns
(bool success);
     function approve(address spender, uint tokens) public returns
(bool success);
     function transferFrom(address from, address to, uint tokens)
public returns (bool success);

     event Transfer(address indexed from, address indexed to, uint
tokens);
     event Approval(address indexed tokenOwner, address indexed
spender, uint tokens);
}
```

4. Furthermore, to avoid any overflow or similar issues, it is important to implement some basic mathematics functions that have checks in place to avoid any function overflow. Here is the contract that can be created:

```
contract Arithmetic {

   function Addition(uint a, uint b) public pure returns (uint c) {
        c = a + b;
        require(c >= a);
    }
    function Subtraction(uint a, uint b) public pure returns (uint
c) {
        require(b <= a);
        c = a - b;
    }
    function Multiplication(uint a, uint b) public pure returns
(uint c) {
        c = a * b;
        require(a == 0 || c / a == b);
    }
    function Division(uint a, uint b) public pure returns (uint c)
{
        require(b > 0);
        c = a / b;
    }
}
```

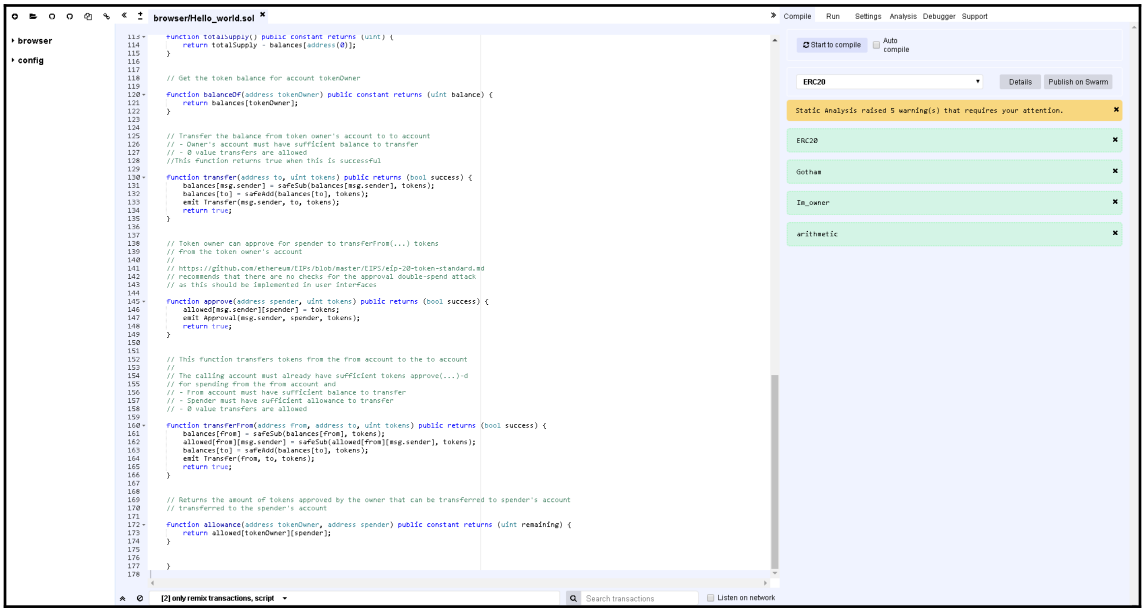
5. Our contract has to have address of the owner; this will help with transferring the coins when the contract is deployed. For this, we create a contract in the Solidity file by the name `Im_owner`. Take a look at the following:

```
contract Im_owner {

    address public owner;
    address public newOwner;
    event OwnershipTransferred(address indexed _from, address
indexed _to);

    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner
{
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}
```

6. Now it is time to work on the `contract Gotham` to add various functions that will return total supply, get the balance of the tokens available in the account, transfer tokens, and other important functions required. Also, since this `contract Gotham` is the main contract, the other contracts in our solidity file should be inherited in this contract. The `contract Gotham` will now look as follows:

```
contract Gotham is ERC20, Im_owner, arithmetic {
    string public symbol;
    string public name;
    uint8 public decimals;
    uint public totalSupply;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;


    // Constructor
    constructor() public {
        symbol = "GOTH"; //This is the symbol of our Token
        name = "GothCoin"; //This is the name of our Token
        decimals = 12; /* These are the number of decimal places it
can have, it is suggested to have 18
decimal places */
        totalSupply = 100000; //total supply of coins allowed
        }


    // Total supply
    function totalSupply() public constant returns (uint) {
        return totalSupply - balances[address(0)];
    }


    // Get the token balance for account tokenOwner

    function balanceOf(address tokenOwner) public constant returns
(uint balance) {
        return balances[tokenOwner];
    }


    // Transfer the balance from token owner's account to to
account
    // - Owner's account must have sufficient balance to transfer
    // - 0 value transfers are allowed
    //This function returns true when this is successful
    function transfer(address to, uint tokens) public returns (bool
success) {
        balances[msg.sender] = safeSub(balances[msg.sender],
tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit Transfer(msg.sender, to, tokens);
        return true;
    }


    // Token owner can approve for spender to transferFrom(...)
tokens
    // from the token owner's account
    //
    //
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-stan
dard.md
    // recommends that there are no checks for the approval double-
spend attack
    // as this should be implemented in user interfaces

    function approve(address spender, uint tokens) public returns
(bool success) {
        allowed[msg.sender][spender] = tokens;
        emit Approval(msg.sender, spender, tokens);
        return true;
    }


    // This function transfers tokens from the from account to the
two account
    //
    // The calling account must already have sufficient tokens
approve(...)-d
    // for spending from the from account and
    // - From account must have sufficient balance to transfer
    // - Spender must have sufficient allowance to transfer
    // - 0 value transfers are allowed

    function transferFrom(address from, address to, uint tokens)
public returns (bool success) {
        balances[from] = safeSub(balances[from], tokens);
        allowed[from][msg.sender] =
safeSub(allowed[from][msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit Transfer(from, to, tokens);
        return true;
    }
    // Returns the amount of tokens approved by the owner that can
be transferred to spender's account
    // transferred to the spender's account
    function allowance(address tokenOwner, address spender) public
constant returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }

    }
}
```

The previous code has the minimum required functions for a Solidity file to be a valid ERC20 standard token. Once the code is complete, you can run it by pressing the **Start to compile** button on the right-hand-side panel in Remix. There is another tab next to the compile tab by the name **Run**, which helps in deploying the smart contract on the `TestNet` as shown in the following screenshot:



```
    function totalSupply() public constant returns (uint) {
        return totalSupply - balances[address(0)];
    }


    // Get the token balance for account tokenOwner

    function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
    }


    // Transfer the balance from token owner's account to to account
    // - Owner's account must have sufficient balance to transfer
    // - 0 value transfers are allowed
    //This function returns true when this is successful

    function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = safeSub(balances[msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit Transfer(msg.sender, to, tokens);
        return true;
    }


    // Token owner can approve for spender to transferFrom(...) tokens
    // from the token owner's account
    //
    // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
    // recommends that there are no checks for the approval double-spend attack
    // as this should be implemented in user interfaces

    function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        emit Approval(msg.sender, spender, tokens);
        return true;
    }


    // This function transfers tokens from the from account to the to account
    //
    // The calling account must already have sufficient tokens approve(...)-d
    // for spending from the from account and
    // - From account must have sufficient balance to transfer
    // - Spender must have sufficient allowance to transfer
    // - 0 value transfers are allowed

    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = safeSub(balances[from], tokens);
        allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit Transfer(from, to, tokens);
        return true;
    }


    // Returns the amount of tokens approved by the owner that can be transferred to spender's account
    // transferred to the spender's account

    function allowance(address tokenOwner, address spender) public constant returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }


    }
```

```
Static Analysis raised 5 warning(s) that requires your attention.
ERC20
Gotham
Im_owner
arithmetic
```

BrowserHello_World.sol file running on your browser using Remix

Summary

In this chapter, we learned about the tools required in smart contract development, testing, and deployment. We learned about Remix, the online official IDE by the Ethereum Foundation; Mist, the official Ethereum Wallet and browser; and MetaMask, the tool to help in Ethereum smart contract development. We learned about the ERC20 token standard, `Web3`, and the Truffle framework. Furthermore, we worked on smart contract creation, along with various details of deployment of a smart contract on Ropsten.

In the coming chapters, we will learn more about decentralized applications and Ethereum Accounts. We will also study in depth about mining, ICO, creating our own currency by forking the Bitcoin repository, and we will discuss the challenges facing blockchain technology.

16 Ethereum Accounts and Ether Tokens

In the previous chapters, we discussed Ethereum, smart contracts, and how the development of a smart contract takes place, including the development of the ERC20 token, based on Ethereum blockchain. We also discussed how the development of Ethereum takes place along with the deployment of a smart contract and token on the Ethereum blockchain.

In this chapter, we will discuss the Ethereum account in detail and also study an ether token; this discussion will help us to better understand decentralized applications. We will also briefly discuss some popular Ethereum tokens and smart contracts. We will also discuss some important topics such as the **transaction sub state**, the validation of an Ethereum Block, and the various steps involved in the process of block finalization. Following this, we will briefly discuss some disadvantages of an Ethereum-based smart contract and currencies, toward the end of this chapter.

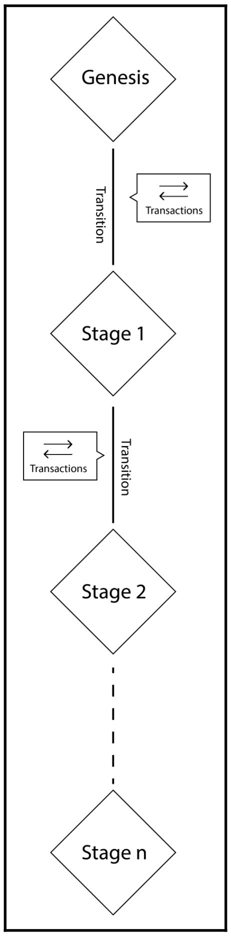
These are the topics which are covered in this chapter:

- The Ethereum state transition function
- The genesis block
- Transaction receipts
- The transaction sub state
- Validating a block
- The disadvantages of Ethereum-based tokens

Introducing Ethereum accounts

The state in Ethereum is made up of objects, each known as an account. Each account in Ethereum contains a 20-byte address. Each state transition is a direct transaction of value and information between various accounts. Each operation performed between or on the accounts is known as a *state transition*. The state transition in Ethereum is done using the Ethereum state transition function.

The state change in Ethereum blockchain starts from the genesis block of the blockchain, as shown in this diagram:



Each block contains a series of transactions, and each block is chained to its previous block. To transition from one state to the next, the transaction has to be valid. The transaction is further validated using consensus techniques, which we have already discussed in previous chapters.

To avoid stale blocks in Ethereum, **GHOST** (**Greedy Heaviest Observed Subtree**) protocol was introduced. This was introduced to avoid random forking by any nodes and inapt verification by other nodes. Stale blocks are created when two nodes find a block at the same time. Each node sends the block in the blockchain to be verified. This isn't the case with Bitcoin, since, in Bitcoin, block time is 10 minutes and the propagation of a block to approximately 50% of the network takes roughly 12 seconds. The GHOST protocol includes stale blocks also known as uncles, and these are included in the calculation of the chain.

As discussed in the previous chapters, there are two types of accounts in Ethereum blockchain. Namely, **Contract Accounts** (**CA**) and **Externally Owned Accounts** (**EOA**s). The contract accounts are the ones that have code associated with them along with a private key. EOA has an ether balance; it is able to send transactions and has no associated code, whereas CA has an ether balance and associated code. The contract account and the externally owned accounts have features of their own, and a new token can only be initiated by the contract account.

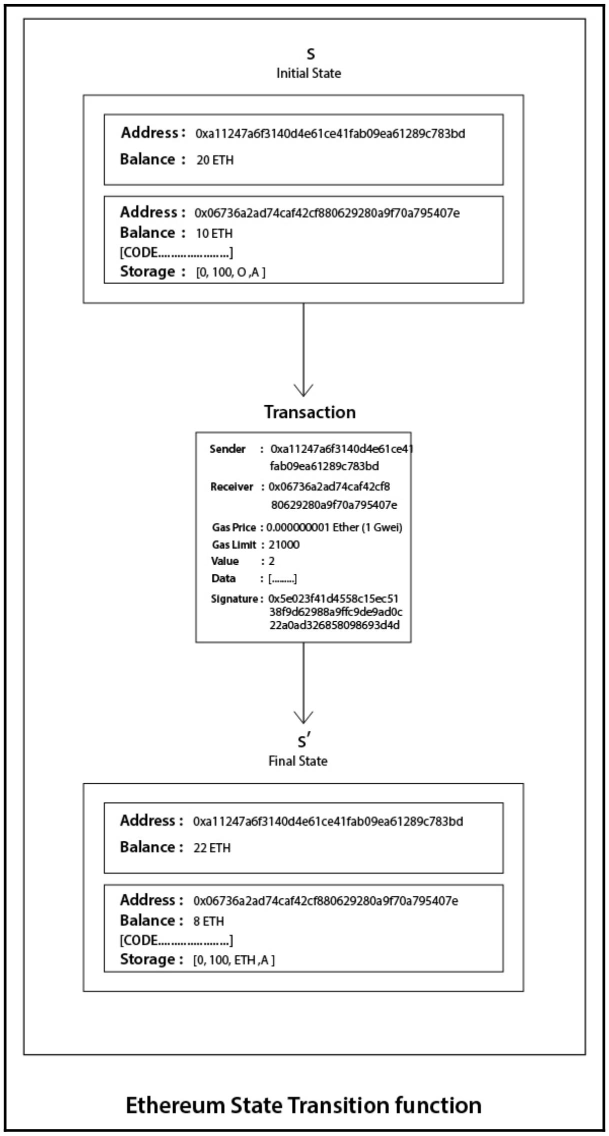
Ethereum state transition function

In the state transition function, the following is the process that every transaction in Ethereum adheres to:

- Confirming of the transaction's validity and structure, by making sure that a signature is valid and nonce matching the nonce in the sender's account and syntax. If there are any issues, then an error is returned.
- The transaction fee calculation is done using the price of gas, and the sender address is determined using the signature. Then, the sender's account balance is checked and reduced accordingly, along with the increment of the nonce. In the case of an insufficient balance, an error occurs.
- Certain gas is taken to cover the cost of the transaction. It is charged per byte incrementally, according to the size of the transaction.
- In this step, the actual transfer of value occurs. The flow happens from the sender's account to the receiver's account. If the receiving account does not exist, then it is created. Also, if the receiving account is a contract account, then the code of the contract is executed. If there is enough gas available, then the contract code runs fully or to the point the gas runs out.

- If the transaction failed due to insufficient balance or gas, all the state changes are reverted, apart from the payment of fees, which is transferred to the miners.
- If there is reminder fee available, then it is sent back to the sender, as change after paying the miners as required. Function returns the resulting state at this point.

The following diagram depicts the state transition flow:

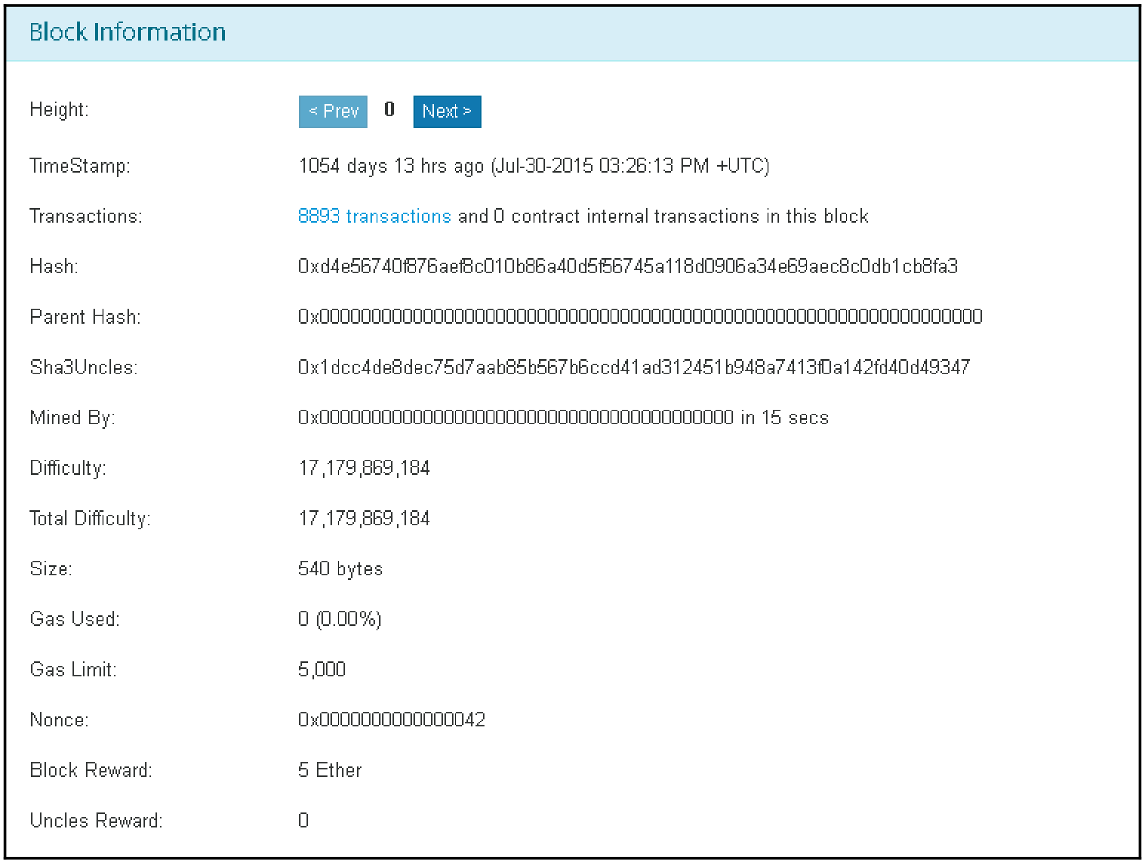

S
Initial State
Address : 0xa11247a6f3140d4e61ce41fab09ea61289c783bd
Balance : 20 ETH
Address : 0x06736a2ad74caf42cf880629280a9f70a795407e
Balance : 10 ETH
[CODE.....................]
Storage : [0, 100, O ,A]
Transaction
Sender : 0xa11247a6f3140d4e61ce41 fab09ea61289c783bd
Receiver : 0x06736a2ad74caf42cf8 80629280a9f70a795407e
Gas Price : 0.000000001 Ether (1 Gwei)
Gas Limit : 21000
Value : 2
Data : [........]
Signature : 0x5e023f41d4558c15ec51 38f9d62988a9ffc9de9ad0c 22a0ad326858098693d4d
S′
Final State
Address : 0xa11247a6f3140d4e61ce41fab09ea61289c783bd
Balance : 22 ETH
Address : 0x06736a2ad74caf42cf880629280a9f70a795407e
Balance : 8 ETH
[CODE.....................]
Storage : [0, 100, ETH ,A]
Ethereum State Transition function

The function is implemented independently in each of the Ethereum clients.

Genesis block

This is the first block of the Ethereum blockchain, just like the genesis block of the Bitcoin blockchain. The height of the genesis block is 0.

The Genesis block was mined on Jul 30, 2015 and marks the first block of the Ethereum blockchain. The difficulty of the genesis block was at 17,179,869,184, as shown in the following screenshot:



Block Information

Height:	< Prev 0 Next >
TimeStamp:	1054 days 13 hrs ago (Jul-30-2015 03:26:13 PM +UTC)
Transactions:	8893 transactions and 0 contract internal transactions in this block
Hash:	0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3
Parent Hash:	0x0000000000000000000000000000000000000000000000000000000000000000
Sha3Uncles:	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
Mined By:	0x0000000000000000000000000000000000000000 in 15 secs
Difficulty:	17,179,869,184
Total Difficulty:	17,179,869,184
Size:	540 bytes
Gas Used:	0 (0.00%)
Gas Limit:	5,000
Nonce:	0x0000000000000042
Block Reward:	5 Ether
Uncles Reward:	0

Transaction receipts

Receipts are used to store the state, after a transaction has been executed. These structures are used to record the outcome of the transaction execution. Receipts are produced after the execution of each transaction. All receipts are stored in an index-eyed trie. This has its root placed in the block header as the receipts root.

Elements

Elements is composed of four primary elements; let's discuss each element of Ethereum's transaction receipts, before we look at the structure of a receipt.

Post-transaction state

Post-transaction state is a trie structure that holds the state, after the transaction has been executed. It is encoded as a byte array.

Gas used

Gas used represents the total amount of gas used in the block that contains the transaction receipt. It can be zero, but it is not a negative integer.

Set of logs

The **set of logs** shows the set of log entries created as a result of transaction execution. Logs contain the logger's address, log topics, and other log data.

The bloom filter

The **bloom filter** is created form the information contained in the logs discussed. Log entries are reduced to a hash of 256 bytes, which is then embedded into the header of the block as a logs bloom. Log entries are composed of the logger's address, log topics, and log data. Log topics are encoded as a series of 32-byte data structures, and log data is composed of a few bytes of data.

Structure

This is what the structure of a transaction receipt looks like:

```
     Result: {
       "blockHash":
"0xb839c4a9d166705062079903fa8f99c848b5d44e20534d42c75b40bd8667fff7",
         "blockNumber": 5810552,
         "contractAddress": null,
         "cumulativeGasUsed": 68527,
         "from": "0x52bc44d5378309EE2abF1539BF71dE1b7d7bE3b5",
         "gasUsed": 7097057,
         "logs": [
           {
             "address": "0x91067b439e1be22196a5f64ee61e803670ba5be9",
             "blockHash":
"0xb839c4a9d166705062079903fa8f99c848b5d44e20534d42c75b40bd8667fff7",
             "blockNumber": 5810552,
             "data":
"0x0000000000000000000000000000000000000000000000000000000576eca9400000000
0000000000000000fd8cd36bebcee2bcb35e24c925af5cf7ea9475d0100000000000000000
0000000000000000000000000000000000000000000",
             "logIndex": 0,
             "topics": [
"0x72d0d212148041614162a44c61fef731170dd7cccc35d1974690989386be0999"
             ],
             "transactionHash":
"0x58ac2580d1495572c519d4e0959e74d70af82757f7e9469c5e3d1b65cc2b5b0b",
             "transactionIndex": 0
           }
         ],
         "root":
"7583254379574ee8eb2943c3ee41582a0041156215e2c7d82e363098c89fe21b",
         "to": "0x91067b439e1be22196a5f64ee61e803670ba5be9",
         "transactionHash":
"0x58ac2580d1495572c519d4e0959e74d70af82757f7e9469c5e3d1b65cc2b5b0b",
         "transactionIndex": 0
       }
       Transaction cost: 7097057 gas.
```

Also, it is to be noted that the receipt is not available for pending transactions.

Transaction sub state

A **transaction sub state** is created during the execution of the transaction. This transaction is processed immediately after the execution is completed. The transaction sub state is composed of the following three sub items.

Suicide set

A **suicide set** contains the list of accounts that are disposed after a transaction execution.

Log series

A **log series** is an indexed series of checkpoints that allow the monitoring and notification of contract calls to the entities external to the Ethereum environment. Logs are created in response to events in the smart contract. It can also be used as a cheaper form of storage.

Refund balance

A **refund balance** is the total price of gas in the transaction that initiated the execution of the transaction.

Messages

Messages are transactions where data is passed between two accounts. It is a data packet passed between two accounts. A message can be sent via the **Contract Account** (**CA**). They can also be an **Externally Owned Account** (**EOA**) in the form of a transaction that has been digitally signed by the sender.

Messages are never stored and are similar to transactions. The key components of a message in Ethereum are:

- Sender of the message call
- Transaction originator
- Transaction recipient
- Contract account, whose code is the be executed during the message call, if there is an account for this purpose
- Maximum available gas
- Gas price

Messages are generated using `CALL` or `DELEGATECALL` methods.

Calls

A **CALL** does not broadcast anything in the blockchain; instead, it is a local call to any contract function specified. It runs locally in the node, like a local function call. It does not consume any gas and is a read-only operation. Calls are only executed locally on a node and do not result in any state change. If the destination account has an associated EVM code, then the virtual machine will start upon the receipt of the message to perform the required operations; if the message sender is an independent object, then the call passes any data returned from the EVM.

Ethereum block validation

After being mined by the miners, an Ethereum block goes through several checks before it is considered valid; the following are the checks it goes through:

- All the ommers/uncles have to verify its identity, considering the PoW for the uncles is valid.
- The existence of the previous block and its validity
- The validity of the timestamp of the block, that is, the current block's timestamp must be higher than the parent block's timestamp. Also, the current block and the parent block should be less than 15 minutes apart from each other. All the block times are calculated in Unix time.

If any of the preceding checks fails, the block gets rejected.

Uncles validation

In this process, the uncles or ommers are validated. Firstly, a block can contain a maximum of two uncles, and, secondly, whether the header is valid and the relationship of the uncle with the current block satisfies the maximum depth of six blocks.

Block difficulty

Block difficulty in Ethereum runs parallel to the calculation of block difficulty in the Bitcoin blockchain. The difficulty of the block increases if the time between two blocks decreases. This is required to maintain consistent block generation time. The difficulty adjustment algorithm in the Ethereum Homestead release is as follows:

```
block_diff = parent_diff + parent_diff // 2048 * max(1 - (block_timestamp -
parent_timestamp) // 10, -99) + int(2**((block.number // 100000) - 2))
```

In this algorithm, the difficulty of the block is adjusted based on the block generation time. According to this algorithm, if the time difference between the generation of the parent block and the current block is less than 10 seconds, the difficulty increases. If the time difference is between 10 and 19 seconds, then the difficulty remains same. The difficulty decreases when the time difference between two block's generation is more than 20 seconds. The decrease in difficulty is directly proportional to the time difference.

Apart from the timestamp-based difficulty increment, as per the algorithm, the difficulty increases exponentially after every 100,000 blocks. This is known as the *difficulty time bomb*, introduced in the Ethereum network, since this will make it very hard to mine on the Ethereum blockchain network. This is the reason why PoS is the proposed consensus mechanism for Ethereum in the near future.

Block finalization

The finalization of a block in Ethereum involves the following four stages:

- The validation of ommers
- The validation of transactions
- The application of rewards
- The verification of the state and the block nonce

Disadvantages of Ethereum-based tokens

We have discussed the advantages and uses of Ethereum and Ethereum blockchain-based currencies throughout the previous chapters; let's now discuss some disadvantages of Ethereum-based tokens:

- **Unstable**: The Ethereum blockchain is still undergoing lot of changes; this also includes moving the consensus method from the PoW to the PoS system.

- **Dependability**: The ERC20 tokens or any other tokens are based and built on top of Ethereum. This poses a threat, since Ethereum is itself undergoing frequent changes at most times.
- **Hard forks**: Being dependent on another blockchain means, due to hard forks, unforeseen issues can occur.
- **Sovereignty**: Since the tokens are based on Ethereum blockchain, there is no say in future development of these tokens. The decision of whether to choose ERC20 should be based on the application and use case.
- **Support**: Any improvements or requirements for support after the smart contract is published on the blockchain can only be done on the Ethereum blockchain and any changes/improvements suggested in the blockchain have to be approved by the whole Ethereum community.
- **Ether as the main currency**: Although with one can create their own currency, symbol and supply limit, ether is still the main currency used in the Ethereum blockchain, hence at all times Ether gets enriched for the efforts added by your token.

Summary

In this chapter, we discussed the Ethereum state transition function, the genesis block, and transaction receipts. We also discussed the transaction sub state. In addition to these topics, we discussed Ethereum block validation and the steps involved in the same as discussed in the Ethereum Yellow paper. Finally, we briefly discussed a few of the disadvantages of using an Ethereum-based token.

In the next chapter, we will discuss **Decentralized Applications**, and we will learn how to create a **DApp**, and how to publish one. We will also discuss the future of DApp and its effects on its users.

17 Decentralized Applications

Decentralized Applications (**DApps**) are applications that run across a decentralized network and are not owned or controlled by a centralized authority. They differ from distributed applications primarily in terms of ownership. A distributed application may run on thousands of computers, but those computers and the management of the software running on them are controlled by a central authority—Amazon, Microsoft, and so on. A decentralized application runs on what is typically a peer-to-peer network and is designed in such a way that no one person or organization can control the functioning of the application. A decentralized application does not require a blockchain. There were multiple decentralized applications before blockchain: BitTorrent, Tor, and Mastodon are all decentralized applications that exist without the use of a blockchain.

In this chapter, we are going to cover the following:

- The major features of a decentralized application
- The largest decentralized applications in operation today
- Non-blockchain based decentralized applications

The goal of this chapter is to give you an understanding of decentralized applications and their development, as well as making you aware of ecosystems and code that already exists. If you are interested in building a decentralized application, interoperability with the existing ecosystems out there will greatly improve your odds.

Let's start by taking a look at what makes an application decentralized.

What makes an application decentralized

Earlier in this book, we discussed distributed versus decentralized systems. A distributed system is one that is made up of a number of computers, with the work of the system distributed across all of these machines. Typically, the computers in a distributed network are placed in different geographical regions to protect the system from outages such as power failures, natural disasters, or military events. A decentralized network is not only distributed geographically, but also in terms of authority and control. A distributed system such as the Amazon cloud can be worldwide in scope but still under the control of a central authority. A decentralized system has no central authority.

Defining a decentralized application

A well-known resource to blockchain-based decentralized applications is the whitepaper written by David Johnson entitled *The General Theory of Decentralized Applications, DApps*. In this whitepaper, he identifies four key criteria to be a DApp:

1. The application must be completely open source; it must operate autonomously, and with no entity controlling the majority of its tokens. The application may adapt its protocol in response to proposed improvements and market feedback, but all changes must be decided by consensus of its users.

2. The application's data and records of operation must be cryptographically stored in a public, decentralized blockchain, so as to avoid any central points of failure.

3. The application must use a cryptographic token (Bitcoin or a token native to its system) that is necessary for access to the application and any contribution of value from miners/farmers should be rewarded with the application's tokens.

4. The application must generate tokens according to a standard cryptographic algorithm acting as proof of the value that the nodes are contributing to the application (Bitcoin uses the PoW algorithm).

However, this definition is very limited. David is thinking only of decentralized applications running on a blockchain, and only ones that can be incentivized through a token. There are a number of decentralized applications that predate blockchain that do not use or require tokens. In this chapter, we will discuss both blockchain and non-blockchain decentralized applications, but with a focus on those that are relevant to the blockchain ecosystem. We will also discuss blockchain applications that are not decentralized, despite running on top of a decentralized network.

For this book, we will use the following four criteria to describe a decentralized application:

- The application must be completely open source.
- It must operate autonomously, with no individual, organization, or organized group controlling the functioning of the application. The application may adapt its protocol in response to proposed improvements and market feedback, but all changes must be decided by consensus of its users.
- The application's data must be stored in a public, decentralized way that is open to inspection and without a single point of failure.
- The application must provide a way of aligning incentives for the use of the application across all key user groups.

As you can see, this simplified definition retains all the key principles of decentralization without relying on a blockchain or tokens, as there are many ways that decentralized applications can be structured or used with or without blockchain. As we will see when we look at IPFS, it is entirely possible to have a decentralized application without blockchain, and incentives without having tokens.

Decentralized apps and blockchain

A decentralized application is a purpose-specific decentralized system. For instance, while Ethereum is a decentralized network, because anyone can join and the nodes are all peer-to-peer, a decentralized application will run on top of the network to provide a specific service or set of services to users. To some extent, the distinction is moot—you could see Ethereum as a distributed application that provides smart contract services and native token transfers. In any case, the key distinction is about power.

Using blockchain does not make an application decentralized

The more a single entity or small group maintains power over the application, the more centralized it is. The less any one group is able to control the fate of the application and its functioning, the more decentralized it is. Just as decentralized applications do not require a blockchain, running on a blockchain does not make an application decentralized. This means that many applications running on blockchains today may still not be true decentralized applications. This is true, even if the application is entirely open source.

To illustrate, let's consider a small sample application called `SpecialClub`, written in Solidity. It is very simple, merely keeping a list of members (stored as addresses) that are part of the `Special Club`:

```
pragma solidity ^0.4.23;

contract SpecialClub {
    // Centralized owner of this application
    address public owner;
    // we set members to true if they are a member, false otherwise.
    mapping(address => bool) public members;
    mapping(address => bool) internal requests;
    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner() {
      require(msg.sender == owner);
      _;
    }
    function approveMembership(address _address) onlyOwner external {
        members[_address] = true;
        requests[_address] = false;
        emit GrantedMembership(_address);
    }
    function requestOwnership() external {
        requests[msg.sender] = true;
        emit RequestToJoin(msg.sender);
    }
    event RequestToJoin(address _address);
    event GrantedMembership(address _address);
}
```

Despite being written in Solidity and deployed on a blockchain, this code is entirely centralized. It is still distributed, as the list of members will be publicly distributed across the entire Ethereum network if deployed. However, control remains with a single address—the `owner`. The `owner` address has absolute control over who is allowed to be added to the membership list of `SpecialClub`. Any further functionality based on this membership list will generally be centralized as a result. One advantage that continues to exist over traditional applications is transparency—by having both the code and its state written to the blockchain, everyone is clear about the rules and the list of members. However, to be a truly decentralized application, this app would need to be modified so that, for example, existing members could vote on who to accept or reject.

Here is a very basic example of how that might look:

```
pragma solidity ^0.4.23;

contract SpecialClub {
    address public owner;
    // we set members to true if they are a member, false otherwise.
    mapping(address => bool) public members;
    mapping(address => bool) internal requests;
    mapping(address => mapping(address => bool)) votedOn;
    mapping(address => uint8) votes;
    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner() {
      require(msg.sender == owner);
      _;
    }
    modifier onlyMember() {
        require(members[msg.sender] == true);
        _;
    }
    function approveMembership(address _address) onlyOwner external {
        members[_address] = true;
        requests[_address] = false;
        emit GrantedMembership(_address);
    }
    function requestOwnership() external {
        requests[msg.sender] = true;
        emit RequestToJoin(msg.sender);
    }
    function voteInMember(address _address) onlyMember external {
        //don't allow re-votes
        require(!votedOn[_address][msg.sender]);
        votedOn[_address][msg.sender] = true;
        votes[_address] = votes[_address] + 1;
        if (votes[_address] >= 5) {
            members[_address] = true;
            requests[_address] = false;
            emit GrantedMembership(_address);
        }
    }
    event RequestToJoin(address _address);
   event GrantedMembership(address _address);
}
```

This version allows new members to be added if at least five existing members vote to make it happen. While such an application would start out centralized, after five members, the owner would no longer be able to exert any control over the membership list. Over time, the level of decentralization would grow.

Major decentralized blockchain applications

It is assumed the reader of this book may be contemplating launching their own decentralized application project or contributing to an existing one. When building new decentralized applications, it is important to be aware of what already exists so that your application can take advantage of the existing functionality. There are a number of existing decentralized applications that are running and in production and that have already undergone substantial development. In general, these applications provide services to help other decentralized applications flourish. In addition, these applications are all open source. If you evaluate one of these projects and find it is missing something, substantially less effort is likely required to either contribute to or for existing functionality.

IPFS

IPFS stands for **InterPlanetary File System**. IPFS is designed to be a decentralized, peer-to-peer file system that can be accessed from the web. IPFS is a decentralized application, but does not involve a blockchain. Instead, it works on a related technology called a **Directed Acyclic Graph**, or **DAG**.

Typical web communications

To understand IPFS, it is easier to start with the way typical web communications work and then establish the differences. In ordinary web communications, there are two primary actors: clients and servers. The servers host an application, files, and so on, and the clients connect to the server. Clients generally do not connect to each other. If two clients communicate at all, they do so by first connecting to the server and having it act as an intermediary.

This approach is simple to understand, and companies have learned how to scale it reliably. Businesses turn to services such as Amazon Web Services, or use software such as OpenStack, to handle large amounts of traffic by adding server capacity.

However, this approach is also very inefficient. For instance, imagine that 10 people in a building are all watching the same movie on their laptops. Instead of having all 10 clients hit the server and download 10 copies, why not have each of the clients download 1/10th of the movie and then share it with the others for a dramatic drop in overall bandwidth and a similar increase in speed?

Peer-to-peer file transfer

This approach of peers downloading different pieces of a single file and then sharing it is the core of a technology called BitTorrent. BitTorrent allows peers to trade files and file fragments amongst one another. The downside of this is that there is little incentive not to download files you desire and then disconnect—preventing others from doing the same. To avoid this, IPFS took inspiration from the BitTorrent protocol and generalized it into a technology they call **Bitswap**. Bitswap works similar to BitTorrent, but with further inspiration from Bitcoin and its proof-of-work incentive system. Bitswap has an incentive system of its own where if a client downloads and stores rare data—data that isn't already highly available on the network—in order to keep global accessibility high.

Value to blockchain projects

While blockchains provide a decentralized method of value and data exchange; they are not well suited for storing large files or content, such as images and videos. Given the expectations of modern customers and web users, this is a real issue. Most storage solutions for these types of content are centralized and expensive. With IPFS, decentralized applications have a way of storing and accessing this content without losing decentralization.

Aragon

Aragon is a project oriented around **Distributed Autonomous Organizations**, or **DAO**s. Here is an excerpt from the Aragon whitepaper:

- Aragon is a DApp that lets anyone create and manage any kind of organization (companies, open source projects, NGOs, foundations, hedge funds...) on the Ethereum blockchain.

- Aragon implements basic features of an organization such as a cap table, token transfers, voting, role assignments, fundraising, and accounting. The behavior of an Aragon organization is easily customized by changing the bylaws. In addition, Aragon organizations are extensible through third-party modules that interact with the organizations' contracts.
- Aragon Core is a set of smart contracts written in Solidity that allow a distributed team to self-organize and handle activities that are part of a typical centralized organization, easily, for instance, handling payroll, internal arbitration, and so on.
- The smart contracts in Aragon Core have been used by other projects, such as districtOx, as a foundation for more advanced functionality. If your project involves any sort of distributed governance, it would make sense to look into Aragon Core.

district0x

The districtOx network is built on top of Ethereum, Aragon, and IPFS. It takes the capabilities of all of those systems and extends them for more specific functionality. In this case, districtOx provides core functionalities that are necessary to operate an online marketplace or community in a decentralized manner using the Ethereum blockchain and decentralized governance.

What is a district?

A district is a decentralized marketplace/community that is built on top of the districtOx `d0xINFRA` code base. The `d0xINFRA` code base is comprised of a set of Ethereum smart contracts and browser-based tools that can interact with both Ethereum and IPFS. These two code bases interact and present a set of key functionalities necessary for experiences people are used to on the centralized web: posting and listing content, searching, reputation management, payments, and invoicing.

Every district build on top of `d0xINFRA` will have these basic functionalities. This baseline code base makes it much easier and faster for future projects to develop into full-featured products.

Ethereum name service

Ethereum addresses are usually written as a hexadecimal string, for instance, `0x86fa049857e0209aa7d9e616f7eb3b3b78ecfdb0`. This is not very readable for a human and would be very prone to error if you had to read it to someone over the phone, type it, and so on. It would also be easy to replace with another address without someone knowing. In fact, this is what has happened in a few ICO attacks. The Ethereum name service, or ENS, is a smart contract based system for resolving human readable names such as `mytoken.ens` to addresses. By registering an ENS address, compatible applications and wallets on the Ethereum network could map a readable name, such as `MyTokenContract.eth`, to an Ethereum address, similar to the way DNS maps domain names to IP addresses on the internet.

It is strongly recommended that any project built on Ethereum secures an appropriate ENS name. Not only does it look cleaner for users, but it will help to prevent hackers from using the name to attempt to steal from your users.

Civic/uPort

Civic and uPort are both identity provider DApps on Ethereum. Currently, your identity information is generally held by a centralized entity: a government, Facebook, Google, and so on. On many places throughout the web, you can be asked to log in by using Facebook or Google. Behind the scenes, the website reaches out to one of these providers, it hands over identity credentials, and then the site lets you in. The downside to this is that the information that you relinquish is under the control and management of a third-party. If one of these providers decided to stop serving you, or provided false information, there would be little you could do about it.

Civic and uPort are both decentralized solutions to identity where the identity owner manages their identity on the blockchain and can grant and revoke permissions through a provider service under their control.

Many upcoming DApps have a blend of blockchain behavior, web activity, and mobile application behaviors. By using one of these providers, you can plug into the emerging decentralized identity ecosystem in addition to supporting centralized providers.

Gnosis

Gnosis was the first major decentralized application to launch on Ethereum. Gnosis provides a decentralized prediction market and governance tool. Prediction markets can work in various industries, such as stock markets, event betting, and so on. By using a decentralized approach to prediction, the hope is that such predictions are more accurate because of a greater variety of information entering the market to adjust projections.

As one of the earliest adopters of Ethereum, the Gnosis team also puts a lot of energy into various tools such as multi-signature wallets.

Steemit

Steemit is a social blogging application, similar to Blogger or Tumblr. However, content, comments, and votes are stored and secured on the blockchain itself. Steemit is a DApp that has its own blockchain. The core is adapted from Bitshares v2.0, but substantially modified to be purpose-suited to the application.

In Steemit, users are rewarded tokens for submitting and voting on content. Each user's vote carries power equal to their shares in the network, called **Steem Power**. If a user attracts a large following, or a following of users with a large amount of Steem Power, then the rewards can be substantial. Rewards are typically small (from pennies to a few dollars), but some authors have been able to get payouts in the thousands because they attracted the favor of power users referred to as **whales**. In some cases, the vote of a whale can be worth hundreds or thousands of dollars on its own.

As mentioned earlier, Steemit is not on Ethereum and cannot be programmed with Solidity. It does, however, have a growing ecosystem of apps that talk to the Steemit blockchain and are used to curate, display, and monetize Steemit content in different ways. Anyone considering blogging, social media, or a similar content app, should carefully evaluate Steemit and its code base. It is all open source and a few modified clones have already been created, such as Golos (for the Russian market) and Serey (for the Cambodian market).

CryptoKitties

CryptoKitties is another Ethereum-based decentralized application. CryptoKitties is a virtual pet simulator, where users can buy, trade, and breed kitties on the blockchain. CryptoKitties was an important landmark for Ethereum, as the techniques developed for CryptoKitties have applications for all video games that may use blockchain. Using techniques similar to Cryptokitties, player equipment, characters, and so on, can be stored on a blockchain.

This is important, because many online video games, such as Minecraft, World of Warcraft, and so on, have suffered from bugs where certain equipment in the games could be duped, and people could make unlimited clones. Using blockchain, each item is assigned a unique reference and can be tracked and traded just like real goods.

Inspired by CryptoKitties, a number of video games are coming to market using these systems to create worlds with genuine scarcity and real economies.

Summary

You should now understand the difference between a decentralized application and a distributed application. A distributed application is one that is spread across many servers and systems, and ideally, the computers involved are also spread across multiple geographic regions for purposes of backup, processing, and availability. A DApp is one in which no single company, person, or group has control over the operation of the application.

While there are many blockchain applications coming to market, not all of them are truly decentralized. In many cases, these applications are merely distributed differently than prior applications by piggybacking on a public blockchain network. If a company or a few key users still control the operation and function of an application, then that application is not truly decentralized, even if it runs on a decentralized network.

You should now be aware of many of the largest blockchain-oriented decentralized applications currently in existence, their interrelationships, and you should be able to use this knowledge to inform future projects and avoid the re-invention of existing code.

Next, we'll discuss in detail how the two largest blockchain networks—Bitcoin and Ethereum—secure the network in `Chapter 18`, *Mining*.

References

[1] `https://github.com/DavidJohnstonCEO/DecentralizedApplications`

18 Mining

In the previous chapters, we discussed the **Proof of Work** (**PoW**) consensus system and the importance of mining. We also discussed Bitcoin and other Altcoins, and how miners play an important role in PoW-based coins.

In this chapter, we will discuss mining in depth, and the need for mining in PoW-based coins and tokens. Then, we will discuss mining pools and how they brought a revolution to the mining ecosystem. Further, we will go ahead with learning how to start mining using the various miners that are available. We will learn in depth about CPU and GPU mining, along with researching about setting up a mining rig and the concept of dual mining. Finally, we will study each of the PoW algorithms available for mining, and discuss those best chosen based on hardware resources available.

In this chapter, we will cover the following topics:

- The mining process and its stages
- Mining algorithms
- Various types of mining hardware
- Types of miners
- Mining rigs
- Types of mining pools
- Mining softwares

Cryptocurrency mining

Cryptocurrency mining is performed by full nodes, that are part of the blockchain; mining is performed only by blockchains with a PoW based consensus system. Transactions are confirmed by the consensus system, and blocks of these transactions are created to be added to the blockchain; once a new block is added to the blockchain, which is commonly known as **block is found**, there is a certain reward, which is given to the miner for performing the task of adding the block in the blockchain; the process is not that simple, though. These blocks are added after performing a resource-intensive validation process to validate a transaction. The resource-intensive task is basically the hashing of certain algorithms associated with the currency.

Since the block generation time is kept to around 10 minutes, when the hashing power of miners increases, the difficulty has to be increased in the same proportion. This is done by difficulty adjustment and re-targeting algorithms, as discussed in the previous chapters.

When a miner connects with the network, there are various tasks that the miner performs to keep up with the network. Each coin has a different specification for miners; shown here, in the context of Bitcoins, are some of the prime tasks performed by the miners:

- **Validation of transaction**: This is the process in which the transactions are validated by verifying the signatures and outputs.
- **Validation of block**: Once each transaction in a block is validated, the nonce is validated to make the process of block validation complete.
- **New block creation**: Miners can create a new block at a higher height in the blockchain by adding the transactions that are not part of any other block after the transactions are verified over the network.
- **Proof of work algorithm solution**: In the task, the miners find a block on the blockchain by solving the required algorithm: SHA256 in the case of Bitcoins. The block header contains a 32-bit nonce field, and miners have to hash it using brute force techniques till the hash matches.
- **Supporting the rewarding system**: When a node hashes the algorithm, the results are sent over the blockchain network. Once this is done, then other miners in the network verify the block to make sure the height of the transactions in the block do not conflict with any prior block. Once the block is accepted by the network, the miner gets a certain reward in terms of coins.

Let's discuss each of these steps in detail, as well as the process involved in mining cryptocurrency.

The mining process

Mining is mostly done in PoW-based blockchains, but as discussed earlier, PoW is not the only consensus system that is in use; there are various other consensus mechanisms as well. Proof of Work, however, is the most widely used consensus system used in cryptocurrencies.

The concept of PoW existed long before its use in Bitcoin. These systems were used previously to restrain denial-of-service attacks, spams, and other networking-related issues that currently persist in the system, since they require proof of computational work from the requester before delivering the required service. This makes such networking-related attacks infeasible.

For PoW systems to be cost-effective enough, the computational task is moderately difficult to perform by the service requester but easy to check by the service provider. Hashcash is one of the systems that first started using PoW-based protocols utilizing the SHA-256 algorithm. With it, users had to submit the proof of calculating thousands of hashing operations before providing them the required service; this in turn limited DoS and spam attacks.

Bitcoin also uses the SHA-256 hashing algorithm, although it is a random algorithm, and is deterministic in nature, which means for any given input the output will always be the same and can be easily verified by anyone using the same algorithm and the same input.

In cryptocurrency mining, the miner needs two things to get the input for the SHA-256 hashing algorithm:

- Header of the newly generated block
- Nonce

The miner uses the brute force method until the hash output matches the difficulty target; it is a 256-bit number that serves as an upper limit, and the SHA-256 output must be lower than or equal to the current difficulty target for the block so that it can be accepted by the network. For example, this is the hash of a block at height `528499` of a Bitcoin blockchain:

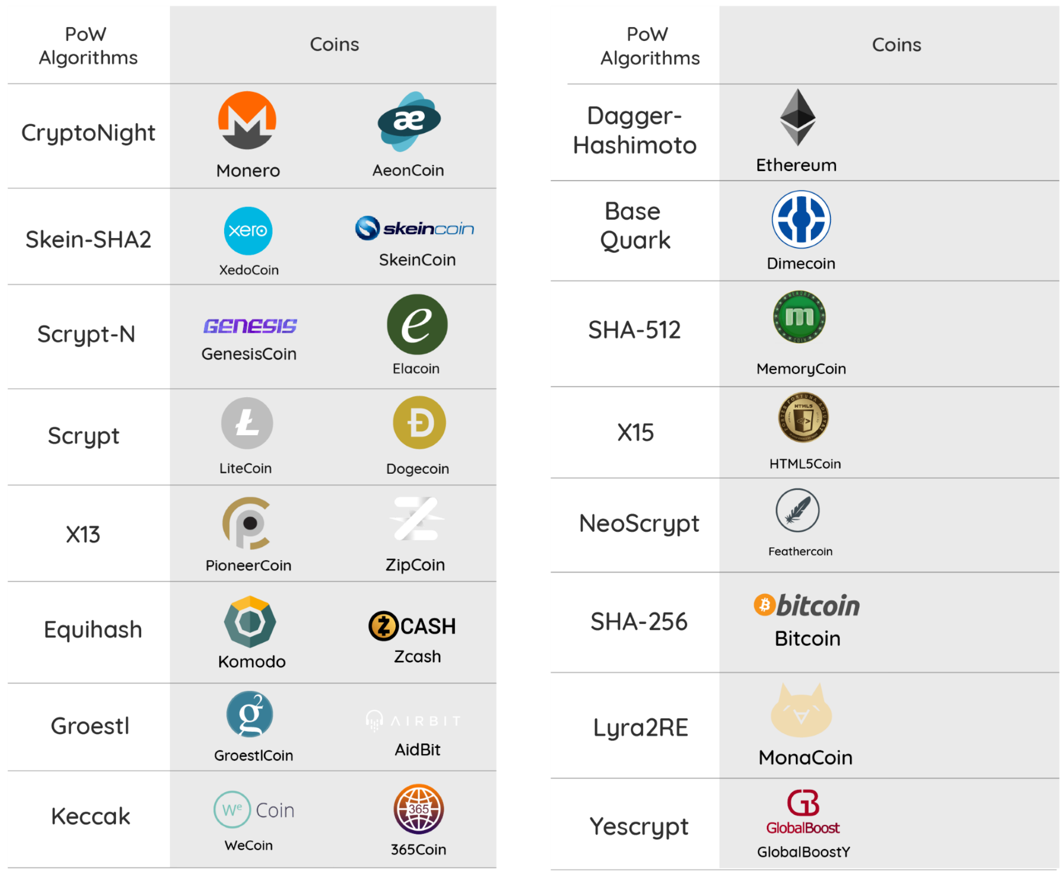
```
00000000000000000021524523382d300c985b91d0a895e7c73ec9d440899946
```

The first transaction in every block is the mining reward, hence it does not have the input address from which funds are to be deducted in the transaction, these are the coins that are created to be a part of the blockchain network. This unique type of transaction is known as a coinbase transaction. Also, in a Bitcoin blockchain, the coins created in the coinbase transaction cannot be spent until it receives at least 100 confirmations in the blockchain. Since the block time is 10 mins, 100 transactions would roughly take 16 hours and 40 minutes. The coinbase transaction can happen on the miner's own address only.

Algorithms

Bitcoin uses SHA-256, but there are various algorithms that can be used in PoW consensus types, and some of these algorithms are listed as follows and illustrated in the next screenshot:

- **base Quark**: Dimecoin, AnimeCoin
- **CryptoNight**: AeonCoin, Monero
- **Dagger-Hashimoto**: Ethereum, Krypton
- **Equihash**: Zcash, Komodo
- **Groestl**: AidBit, GroestlCoin
- **Keccak**: 365Coin, Wecoin
- **Lyra2RE**: Lyrobar, MondaCoin
- **SHA-512**: MemoryCoin, ProtoShares
- **NeoScrypt**: Feathercoin, Ufocoin
- **NXT**: CoinoIndex, SuperNet
- **Scrypt**: Litecoin, Dogecoin
- **Scrypt-jane**: QQcoin, ThorCoin
- **Scrypt-N**: Caishen, Elacoin, GenesisCoin
- **Skein-SHA2**: SkeinCoin, XedoCoin
- **X11**: Dash, Chipcoin, Karmacoin
- **X13**: Cloakcoin, Zipcoin, PioneerCoin
- **X15**: CrystalCoin, Halcyon, HTML5coin
- **Yescrypt**: BitZeny, GlobalBoostY



PoW Algorithms	Coins	
CryptoNight	Monero	AeonCoin
Skein-SHA2	XedoCoin	SkeinCoin
Scrypt-N	GenesisCoin	Elacoin
Scrypt	LiteCoin	Dogecoin
X13	PioneerCoin	ZipCoin
Equihash	Komodo	Zcash
Groestl	GroestlCoin	AidBit
Keccak	WeCoin	365Coin

PoW Algorithms	Coins
Dagger-Hashimoto	Ethereum
Base Quark	Dimecoin
SHA-512	MemoryCoin
X15	HTML5Coin
NeoScrypt	Feathercoin
SHA-256	Bitcoin
Lyra2RE	MonaCoin
Yescrypt	GlobalBoostY

Example's of coins' using various PoW Algorithms

Each of these algorithms has had specific modifications over other algorithms.

Mining hardware

The cryptocurrency mining community has gone through a lot of innovation and resistance hand-in-hand to take care of the core principles of blockchain. Mining can be done using home-based computers and specialized hardware. The types of hardware commonly used for cryptocurrency mining are discussed in the following sections.

CPU-based mining

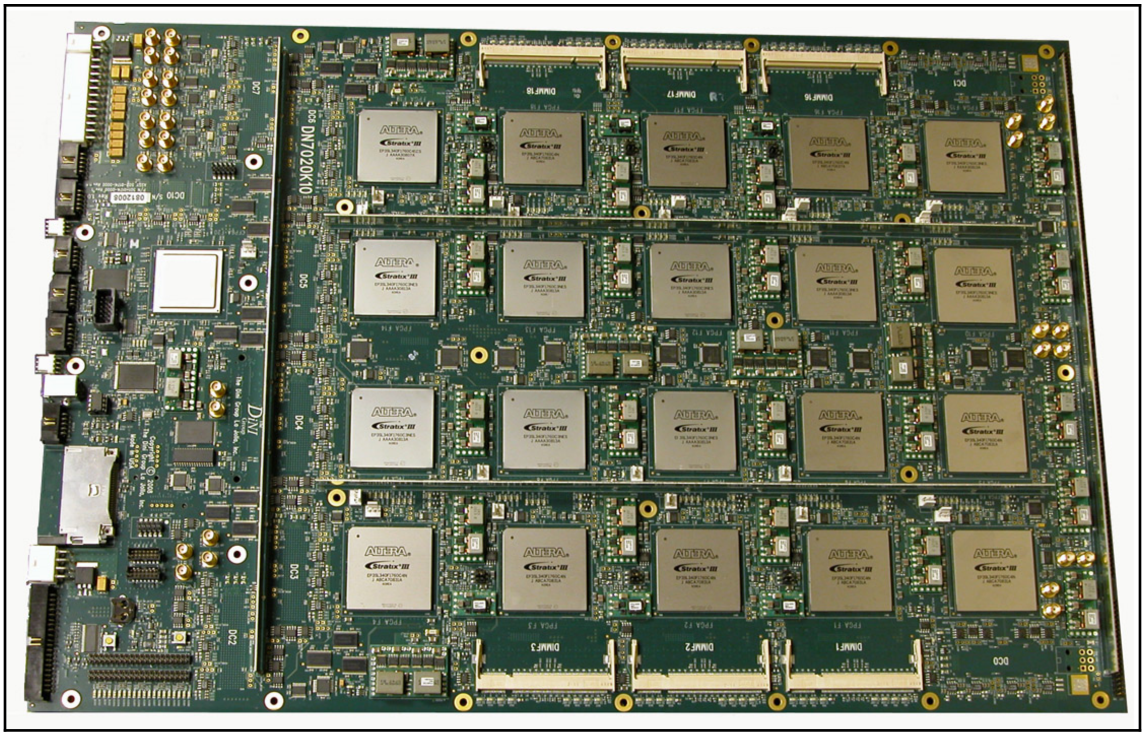
This was the first type of mining available in the official Bitcoin client. During the initial days of Bitcoin, home-based computers were able to mine coins. With the advent of more powerful and specialized hardware, Bitcoin mining is no longer preferred for mining Bitcoins. Other coins still support CPU mining, but as the coins' difficulty grows with time, mining of those types of coins also becomes infeasible.

GPU-based mining

Since the difficulty of the blockchain network increases incrementally over time, CPU mining becomes infeasible, or it sometimes becomes impossible to mine the coin using a CPU. Considering this, miners started to use GPUs since they offer faster and much higher parallel processing. GPU manufacturing companies such as AMD and Nvidia are releasing new hardware from time to time, which can produce excellent mining results apart from gaming performance:

FPGA-based mining

The **field-programmable gate array** (**FPGA**) consists of integrated circuits that can be configured after manufacture. The programming for configuration is specified using **hardware description language** (**HDL**). A field-programmable device can be modified without disassembling the device; this device contains a series of gate arrays that create truth tables to calculate inputs from a data stream as shown in the following screenshot:



As FPGAs support parallelism from the core, they are claimed to be fifteen times more efficient compared to GPU-based mining.

ASIC-based mining

Application-specific integrated circuit (**ASIC**) miners are a lot better compared to CPU-, GPU-, and FPGA-based mining since they are designed to perform one specific task only, that is, the mining of cryptocurrencies. An ASIC miner, pictured below, is algorithm-specific:

These hardwares are specifically built to produce high hashing rates. There are various companies that are popular in producing some of the best-performing miners, such as Bitmain, Avalon, Pangolin, ASICminer, and so on.

Miner types

There are two types of miners these days—classified on the basis of procuring the hardware themselves, or on purchasing the hashing power online.

Cloud mining

In this type of mining, the miners do not own the hardware, and instead they purchase hashing power remotely from other miners. Cloud mining has various pros over mining by procuring one's own hardware, such as low cost of entry, and minimal risks. For people who want to invest in cryptocurrency but do not want to purchase from an exchange or have enough technical knowledge, this is the best possible option for them. Now, there are various organizations that have large data centers at various places with GPU-, ASIC-, and FPGA-based miners available for people to purchase. Some of these organizations are Genesis Mining, SkyCoinLabs, Nicehash, hashflare, and so on.

Hardware mining

Enthusiasts are always interested in setting up self-hosted hardware for mining; mining can be done by a high-end home computer or acquiring an ASIC or FPGA device. Nowadays, people are also setting up mining rigs that are specialized setups with options to connect multiple GPUs, ASICs, or FPGAs to a single machine. People mostly make a rig from scratch by purchasing extended casings, and attaching multiple hardware together to achieve the required results. The best part of a rig is you can add more hardware and try out a new ASIC to see if the desired results are achieved.

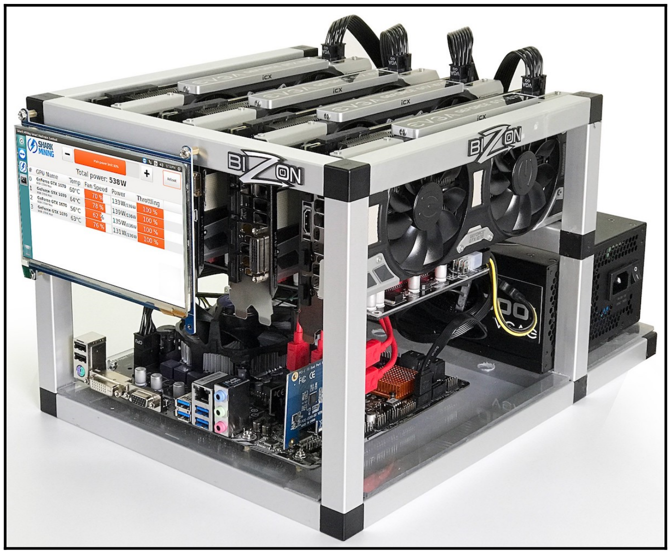
Mining rigs

Since a lot of GPUs or ASICs are rigged together, they tend to produce a large amount of heat, hence it is important to have proper airflow available. Here are the requirements of a basic rig, which one can set up on their own:

- **Motherboard**: A specialized motherboard is required that can support multiple PCI-E slots for multiple GPUs to be connected.
- **HD**: A minimum hard drive is enough, but better to opt for an SSD for better performance.
- **Memory**: A minimum of 8 GB or 4 GB of RAM is enough for a mining rig, as mining is more about computation-intensive work.
- **GPU, ASIC, or FPGA**: These are the most important components in a mining rig, either any one of the configurations can be opted such as GPU-based rig or ASIC-based rig or FPGA-based rig. A mix of all these can also be tried on rigs to figure out which device produces the highest result.
- **Case**: Since a large number of computational devices are rigged together, they tend to produce a huge amount of heat, hence a proper casing is required to connect these together; airflow inside should be adequate since a large amount of heat can result in loss of hardware or reduction in system-resource usage.
- **Power Supply**: A standard power supply in a desktop computer cannot work in a mining rig; multiple GPUs require a large amount of electricity, and hence a power supply that can support such a large amount of electricity is required.

Nowadays, pre-built mining rigs are also available; these rigs are of plug-and-play types, with no setup required. Here are some of the most widely used pre-built rigs:

- **Shark mining**: This is a popular vendor creating various GPU-based rigs supporting numerous coins to mine. They offer various options, from compact to large size rigs.
- **Mining Cave**: They have options for various rigs, for Nvidia- or AMD-based graphic cards:



Mining pools

As more and more miners start to mine for coins, the difficulty of the coin increases. Pools are groups of miners who come together to mine a block, and once a reward is given for successfully mining the block, the reward is split among the miners who mined in the pool; there are various ways in which the payment of the reward is split, and we will be discussing these methods in the next section. The reason for having various reward split methods is because hashing is a purely brute-force-based mechanism, hence it is pure luck for any miner to find the correct nonce and then go ahead with the process of successfully submitting a block in the blockchain, so it would be unfair for other miners in the pool if their hashing and mining efforts go unaccounted for. Hence, on the basis of hashing power, the reward is split, but still there are various methods by which the exact calculation of each miner's share is done.

Pay-per-share – PPS

PPS is a method that transfers the risk to the mining pools. It is the most-preferred method for miners as they are paid on the number of shares of hashes they mine; the reward of the share mined is guaranteed for each and every share, and nowadays very few pools support this system. The miners are paid from the pool's existing balance of coins.

Proportional – PROP

This is known as the proportional approach and, as the name suggests, in this method the reward is proportionally distributed among the miners based on the number of shares of blocks each miner has found.

Pay-per-last-N-shares – PPLNS

PPLNS is similar to the proportional method, although instead of counting all the shares in a round, in this method the last *N* shares are looked, irrespective of the total shares contributed by the miner.

The double geometric method – DGM

DGM is a hybrid approach in which risk is divided between the pool and the miners. In this, the pool receives part of the mining reward when short rounds are going on, and the same is returned when longer rounds are underway.

Shared maximum pay per share – SMPPS

SMPPS is similar to PPS, but the pool does not pay more than the total coins rewarded to the pool in a single round. This removes the risk that the pool takes in the PPS method.

Equalized shared maximum pay per share – ESMPPS

ESMPPS is similar to SMPPS; it distributes payments equally among all the miners who are part of the pool and were mining for the current round.

Recent shared maximum pay per share – RSMPPS

RSMPPS is similar to SMPPS, but this method prioritizes the recent miners first.

Capped pay per share with recent backpay – CPPSRB

CPPSRB uses a Maximum Pay per Share such as system in such a way that it pays the miners the maximum reward using the income from the block rewards, considering that no backup funds are required by the pool.

Bitcoin pooled mining – BPM

BPM is also known as **slush's system**, since it was first used in the mining pool called **slush's pool**. In this payment calculation method, the older shares from the beginning of the block round are given less significance as compared to the recent shares. This system was introduced as community members started reporting that miners were able to cheat the mining pools by switching pools when a round was underway.

Pay on target – POT

POT is a high-variance PPS such as system which pays out based on the difficulty of work returned to the pool by a miner, instead of the difficulty of work done by the whole pool miners mining the block.

SCORE

This is a proportional reward system, based on the time a share was submitted. This process makes later shares worth a lot more than the earlier shares; these shares are scored by time, and the rewards that are to be given to individual miners are calculated based on the proportion of the scores and not the shares submitted in the system.

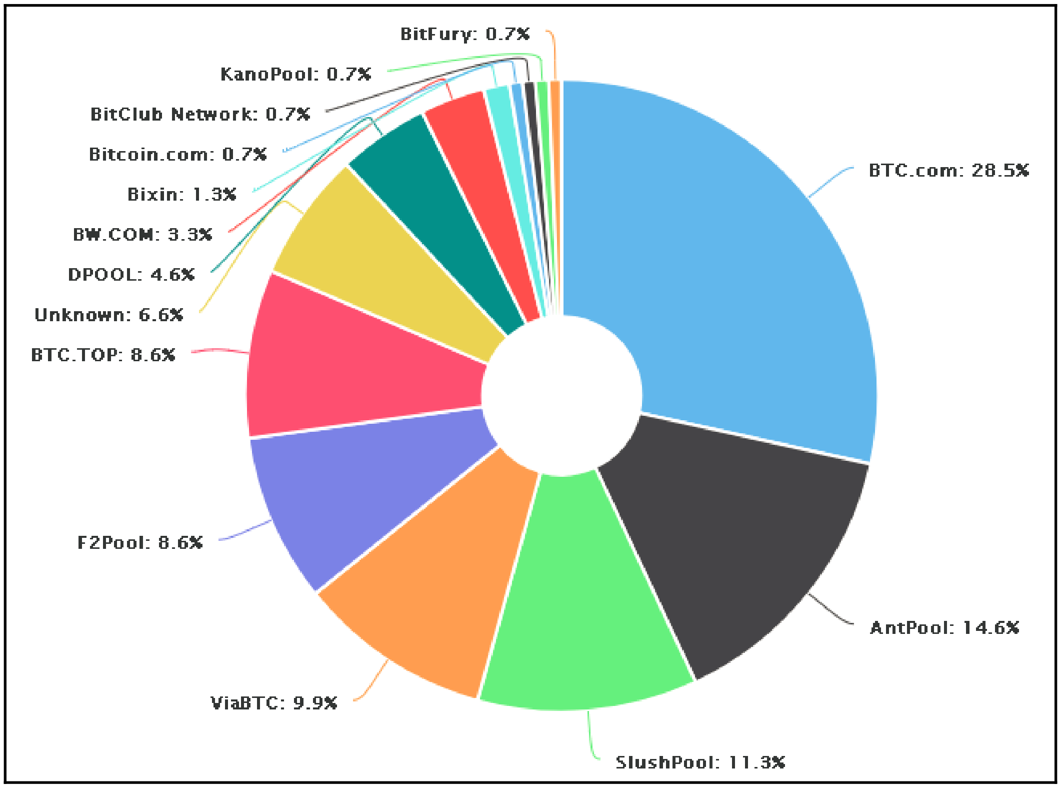
Apart from these, there are still new reward systems being proposed by mining pools and the community; systems such as ELIGUIS, Triplemining, and so on still exist and are being used by various developers.

Popular pools

There are various mining pools present, and anyone can be part of an interested pool and start mining right away. Pools can support anyone cryptocurrency or multiple currencies at a time. Here is a list of mining pools, along with the currencies they support:

- **BTC.com**: This is a China-based pool, part of the Bitmain organization. It supports Bitcoin and Bitcoin Cash.
- **Antpool**: Another pool owned by the Bitmain organization. It supports Bitcoin, Litecoin, Ethereum, Ethereum Classic, Zcash, Dash, Bitcoin Cash, Siacoin, Monero Classic, and Bytom.
- **BTC.TOP**: A China-based mining pool supporting Bitcoin, as of now.
- **SlushPool**: The world's first mining pool, and the most reliable to date. It mines about 3% of all Bitcoin blocks. It supports Bitcoin and Zcash.
- **F2Pool**: A China-based pool, supporting various currencies such as Bitcoin, Litecoin, Zcash, Ethereum, Ethereum Classic, Siacoin, Dash, Monero, Monero Classic, Decred coin, Zcoin, and Aion coin.
- **ViaBTC**: A China-based pool targeted toward China-based miners. It supports currencies such as Bitcoin, Bitcoin Cash, Litecoin, Ethereum, Ethereum Classic, Zcash, and Dash:

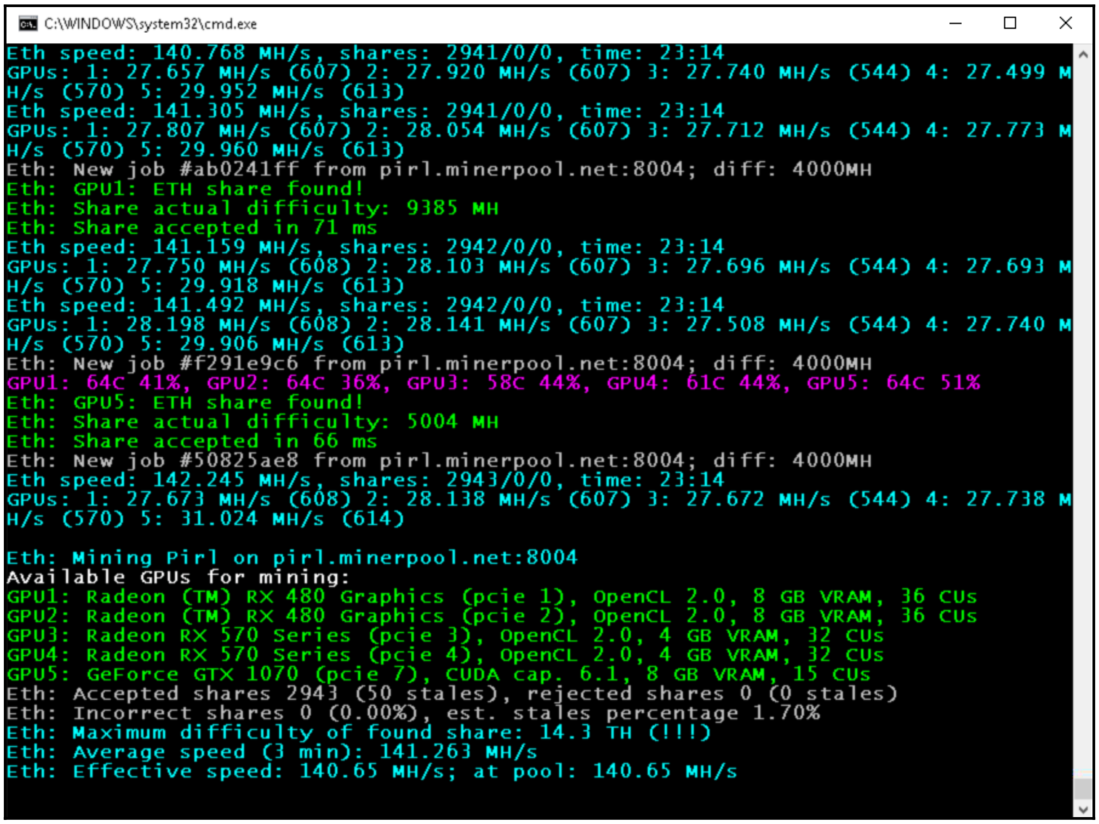
Apart from the listed pools, there are various other pools, some supporting a single coin and some supporting multiple coins. Some of them are BTCC, Bitfury, BW Pool, Bitclub.network, Suprnova, minergate, and so on. The following diagram shows the Hashrate distribution of the Bitcoin Network among the various mining pools, and it can be found at `www.Blockchain.info/pools`:



Mining software

Mining hardware takes care of the mining process, but it is also important to have efficient software for the best results and the removal of bottlenecks, if any.

The task of the mining software is to share the mining task of the hardware over the network. Apart from this, its task is to receive work from other miners on the network. Based on the operating system the hardware is running, there is various mining softwares available, such as BTCMiner, CGMiner, BFGMiner, Nheqminer, and so on:



```
C:\WINDOWS\system32\cmd.exe
Eth speed: 140.768 MH/s, shares: 2941/0/0, time: 23:14
GPUs: 1: 27.657 MH/s (607) 2: 27.920 MH/s (607) 3: 27.740 MH/s (544) 4: 27.499 M
H/s (570) 5: 29.952 MH/s (613)
Eth speed: 141.305 MH/s, shares: 2941/0/0, time: 23:14
GPUs: 1: 27.807 MH/s (607) 2: 28.054 MH/s (607) 3: 27.712 MH/s (544) 4: 27.773 M
H/s (570) 5: 29.960 MH/s (613)
Eth: New job #ab0241ff from pirl.minerpool.net:8004; diff: 4000MH
Eth: GPU1: ETH share found!
Eth: Share actual difficulty: 9385 MH
Eth: Share accepted in 71 ms
Eth speed: 141.159 MH/s, shares: 2942/0/0, time: 23:14
GPUs: 1: 27.750 MH/s (608) 2: 28.103 MH/s (607) 3: 27.696 MH/s (544) 4: 27.693 M
H/s (570) 5: 29.918 MH/s (613)
Eth speed: 141.492 MH/s, shares: 2942/0/0, time: 23:14
GPUs: 1: 28.198 MH/s (608) 2: 28.141 MH/s (607) 3: 27.508 MH/s (544) 4: 27.740 M
H/s (570) 5: 29.906 MH/s (613)
Eth: New job #f291e9c6 from pirl.minerpool.net:8004; diff: 4000MH
GPU1: 64C 41%, GPU2: 64C 36%, GPU3: 58C 44%, GPU4: 61C 44%, GPU5: 64C 51%
Eth: GPU5: ETH share found!
Eth: Share actual difficulty: 5004 MH
Eth: Share accepted in 66 ms
Eth: New job #50825ae8 from pirl.minerpool.net:8004; diff: 4000MH
Eth speed: 142.245 MH/s, shares: 2943/0/0, time: 23:14
GPUs: 1: 27.673 MH/s (608) 2: 28.138 MH/s (607) 3: 27.672 MH/s (544) 4: 27.738 M
H/s (570) 5: 31.024 MH/s (614)

Eth: Mining Pirl on pirl.minerpool.net:8004
Available GPUs for mining:
GPU1: Radeon (TM) RX 480 Graphics (pcie 1), OpenCL 2.0, 8 GB VRAM, 36 CUs
GPU2: Radeon (TM) RX 480 Graphics (pcie 2), OpenCL 2.0, 8 GB VRAM, 36 CUs
GPU3: Radeon RX 570 Series (pcie 3), OpenCL 2.0, 4 GB VRAM, 32 CUs
GPU4: Radeon RX 570 Series (pcie 4), OpenCL 2.0, 4 GB VRAM, 32 CUs
GPU5: GeForce GTX 1070 (pcie 7), CUDA cap. 6.1, 8 GB VRAM, 15 CUs
Eth: Accepted shares 2943 (50 stales), rejected shares 0 (0 stales)
Eth: Incorrect shares 0 (0.00%), est. stales percentage 1.70%
Eth: Maximum difficulty of found share: 14.3 TH (!!!)
Eth: Average speed (3 min): 141.263 MH/s
Eth: Effective speed: 140.65 MH/s; at pool: 140.65 MH/s
```

Mining software should be chosen on the basis of the operating system, hardware type, and other factors. Most mining software is open source and has a large amount of active community to clarify any doubts in choosing the correct software for the available hardware to the miner.

Summary

In this chapter, we learned about the mining of cryptocurrency; starting with studying various algorithms, we discussed mining hardware and the various types available. Then, we discussed mining pools, how the pools split rewards among miners, and various popular pools currently available.

In the next chapter, we will discuss **Initial Coin offering** (**ICO**), which is the process of raising funds for a coin or token that is launched. ICO is an important part of the blockchain community, and helps the blockchain project stakeholder to raise funds from the community itself.

19
ICO 101

ICO stands for **Initial Coin Offering**, also called a **token sale** or **initial token offering**. An ICO is an event where a new blockchain project raises money by offering network tokens to potential buyers. Unlike IPOs, no equity is for sale. Buyers receive tokens on the network but do not own the underlying project intellectual property, legal ownership, or other traditional equity traits unless specifically promised as part of the sale. The expectation of profit (if there is one) comes from holding the token itself. If demand for use of the new network increases, then presumably so will the value of owning the token.

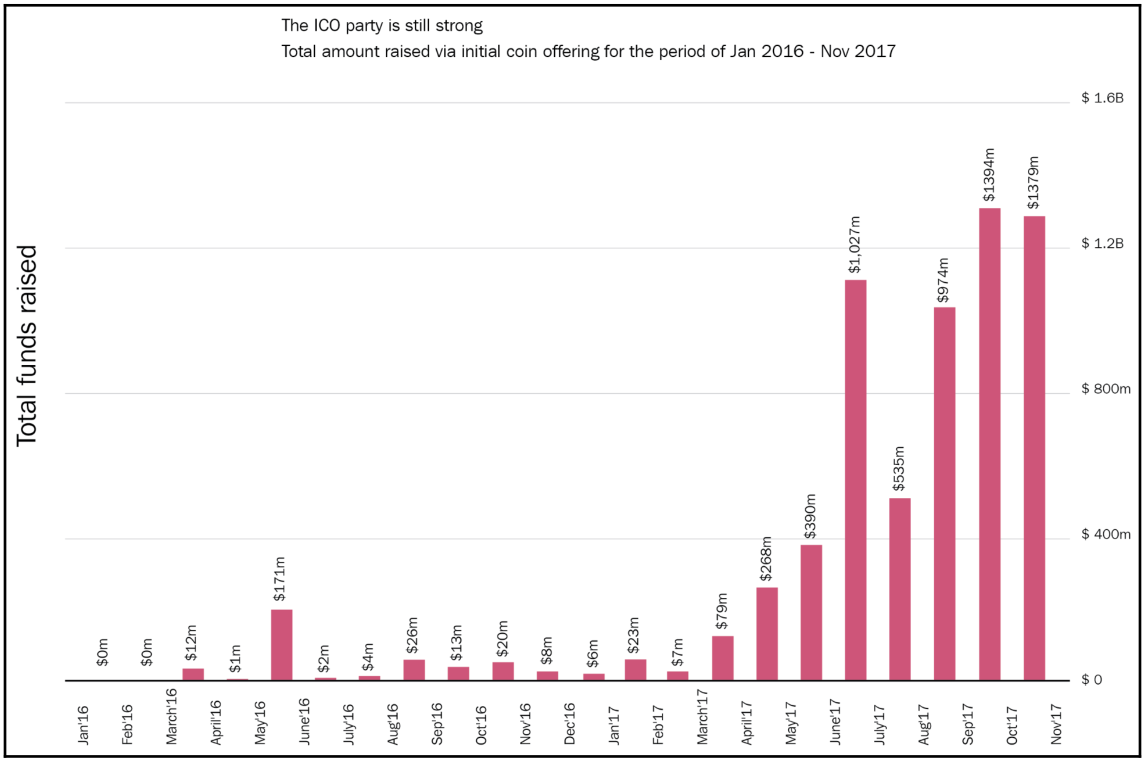
In this chapter, we are going to cover ICOs, how they came about, and the critical aspects that happen as part of executing one. ICOs continue to evolve, but many events and deliverables have become expected and even mandatory for success.

The first ICO was developed in 2013 by Mastercoin. Mastercoin held that their token, such as bitcoin, would increase in value and at least a few others agreed. Mastercoin held a month-long fundraiser and ended up raising about $500,000, while afterwards the Mastercoin overall market cap appreciated to as high as $50 million. The ability to raise substantial capital without going through traditional channels began to spark a flurry of activity.

The following year, the Ethereum network was conceived and held its token sale. Wth the birth of this network, the difficulty of launching a new token decreased substantially. Once the Ethereum network was stable and had established a critical mass, ICOs began to happen regularly. During the next two years, ICOs began to happen more and more frequently.

Some notable projects from this early period include:

- **Ethereum**: $18 million
- **ICONOMI**: $10.6 million
- **Golem Project**: $10 million
- **Digix DAO**: $5.5 million



In 2017 the pace of ICOs accelerated, as did the amount of money raised. Here are some of the major projects from 2017:

1. **Filecoin**: ~$257 Million USD
2. **Tezos**: ~$236 Million USD
3. **EOS**: ~$200 Million USD
4. **Bancor**: ~$153 Million USD

There are now over, 1500 cryptocurrencies and more are released regularly. New projects being released at a rate of around 100/month. The topics that we will be covering in this chapter are as follows:

- The current state of the ICO market
- Typical aspects of an ICO campaign
- Issues with ICO's and blockchain projects

The current state of the ICO market

The difference between the early ICO market and the current state of the industry is stark. In the beginning, there were only a few ICOs and those were held by teams that were relatively well known inside the blockchain community that spent considerable time and effort bringing a project to life before running the ICO. After the launch of Ethereum, the barrier to entry for doing an ICO fell substantially and the number of new tokens swelled.

Increasing volume of ICOs

Before the Ethereum network, most ICOs were for a new blockchain. With Ethereum, tokens could now launch using smart contracts instead of creating the entire blockchain infrastructure from scratch. For more on how this is done, see the chapters on Solidity and Smart Contracts (See `Chapter 13`, *Solidity 101*, and `Chapter 14`, *Smart Contracts*).

Currently, 2018 is on track to have over 1,000 new ICOs:

- The amount of money raised by ICOs as a whole keeps going up
- Extremely large raises are becoming rarer
- Existing companies are starting to ICO

Typical aspects of an ICO campaign

Most ICOs have a typical trajectory for their marketing activities. Each of these activities exists to attract interest and investment in the company and present the project to the world at large.

Whitepaper

For most projects, the most critical piece is the whitepaper. The project whitepaper introduces the purpose of the project, the problems it tries to solve, and how it goes about solving it.

A good white paper will discuss the utility of the token and the market. Key sections of most whitepapers include:

- An introduction to the project
- A history of the market and prior solutions
- An introduction to the new solution using blockchain
- The utility of the token and tokenomics
- Future applications and synergies
- Addressing concerns and risks
- Team and team backgrounds
- A summary

Most whitepapers will include sections such as these, as well as others, depending on the exact nature of the project and the target market. The whitepaper will be used extensively for all future marketing efforts, as it will be the source of information for slide decks, pitches, and so on.

Private placement

Private placement is the art of selling large blocks of tokens to private investors, usually before those tokens are available on the common market. There are a number of reasons for this practice. First, private investors tend to be more sophisticated and are able to place large buys. The literal buy-in of a well-established investor, especially one with a successful track record, will encourage future buying as the highest risk purchase is the first.

Private buyers also provide early funds, possibly before the whitepaper is finalized if they really believe in the strength of the team. In addition, accredited larger investors also have a special legal status in many jurisdictions, including in the United States. This status makes it much safer to sell to them in the uncertain legal environment that faces ICOs.

Private placements can happen in a number of ways:

1. The founders either already know or meet high net-worth individuals or organizations that do private buys, such as family offices
2. The founders hire advisors who can connect them to private buyers
3. The project gets enough buzz and interest pre-investment that buyers seek them out

If the founders do not have an extensive network, they will need to build one as best they can. This can be done by going to conferences, giving talks, attending meetups, and building genuine relationships. This process can take a long time, and founders are encouraged to begin this activity immediately. Networking such as this should happen if you are even considering a blockchain startup or ICO, even before the company is founded. It's also important that a genuine relationship is formed—once these people buy in, they are essentially partners. Their success is the company's success, so they will be highly motivated to help promote the company. At the same time, the company's failure is their failure and loss, so they will want to have a strong trust in the abilities of the founders. Relationships such as this are rarely made instantly.

One step removed from this process is finding the right advisors. Advisors who have access to investors will absolutely wish to be paid for their services, often up front and with a percentage of the raise. Too many people want access to investors to be worth bothering with anyone with no resources at all. Moreover, these advisors must also believe in the project. For an advisor to bring investors bad projects is to lose access to that investor and sour their relationship. Responsible advisors will, therefore, refuse to introduce projects until those projects are truly ready for investment.

The last way for private sales to happen is for a project to get enough interest publicly that investors seek out the project. Because project hype corresponds strongly to token prices, buying into a token with a lot of hype and PR is seen as a safer investment. Smart investors will still strongly vet the project and team, but this evidence of early traction makes things much easier.

Many teams start their fundraising approach with a private placement round. In a few cases, this may be all the team needs.

Well known funds that do private placement are the Crypto Capital Group, the Digital Currency Group, Blockchain Capital, Draper Associates, Novotron Capital, and Outlier Ventures.

Pre-sale

A token pre-sale is usually done before the official launch date of the public token sale. It is half-way between private placement and a full public sale. Tokens are typically sold at some level of discount from the official public price and there may be a higher minimum purchase amount than the public sale.

Such as private placement sales, pre-sales are often somewhat restricted in terms of who is allowed to buy. If a project has been able to build up some community engagement, the pre-sale is typically offered to that community first.

For instance, if the public sale per token will be $0.10, with a minimum purchase of $300 (or equivalent in Ethereum or Bitcoin), the private sale price may be $0.08, but with a minimum purchase of $1,000.

Good pre-sale practices

In an ideal world, a token project doing a pre-sale should have the following:

- A demo or alpha ready to be tested and used
- An established community
- A fully fleshed out team with appropriate backgrounds
- Strong cybersecurity measures in place to prevent theft
- Legal opinions and proper regulatory compliance secured
- Systems for monitoring all communications 24/7 to prevent fraud and bad actors from injecting themselves into the presale
- A strong marketing team and marketing plan in place
- A well-thought-out plan for using any funds
- A clear and reasonable hard-cap on fundraising, both in general and for the pre-sale

Projects that have all of these are in a stronger position for a successful pre-sale. Projects that also have well-known private investors or funds behind them have an additional advantage.

Public sale

The public sale is the end stage of an ICO. By this point, a team should have spent substantial time in community building, early fundraising, PR activities (see in the succeeding sections), and so on. The reason for all this is that the most successful ICOs tend to be over very fast, selling out in minutes or hours. There are three major different approaches to structuring the sale.

Capped sale

The capped sale is the most common ICO structure. In a capped sale, there is typically a soft cap and a hard cap. The soft cap is the minimum raise the team is looking for to build their project. The hard cap is the maximum they will accept as part of the raise. Once the token sale begins, it is usually executed on a first come, fist served basis. Tokens are sold at a pre-set rate. In some cases, that rate may include bonuses. For example, the first 20 buyers (or first $20,000) receives a 10% bonus in terms of the number of tokens purchased. As the sale continues, the bonus drops. This structure is designed to reward initial movers and inspire fomo for fear of missing out.

Uncapped sale

The uncapped sale is designed to raise as much capital as possible. This is done by fixing the time of the sale and, typically, the number of tokens available. As people buy in, they receive a share of tokens equal to their share of the total capital invested. The number of tokens each person receives is often not known until the end of the ICO. For instance, if, on day 1 an investor puts in $10,000 and no one else bids, they would own all of the available tokens. However, the next day, another investor puts in $10,000 as well. If this were the end of the sale, the two investors would each own half the available tokens.

Such as the capped sale, there are many variations. The EOS token sale is probably the best-known version of the uncapped sale. The EOS token was sold over 341 days, with a few more tokens being made available for purchase each day.

Dutch auction

The Dutch auction is the rarest form of ICO offering, but one of the fairest. A Dutch auction works with the auction beginning with a high price, with the price slowly lowered until participants jump in or a reserve is hit. A reverse Dutch auction starts with a low price, and the price is then slowly raised over time at fixed intervals. Either approach is good for finding a proper market price for a token, provided that there are a sufficient number of buyers. The most famous ICO to use the Dutch auction approach was the Gnosis project.

The Gnosis team did not set out to sell a fixed percentage of the total tokens. Instead, the number of tokens released would increase the longer the auction took to hit the cap of $9 million GNO tokens sold, or $12.5 million raised. Despite the auction setup designed to slow down participation, the sale was over in under 15 minutes.

Influencer marketing

Blockchain is a new technology and startups are, by nature, quite risky. For both of these reasons, investors often seek out the input, advice, and viewpoints of established experts in the field. In a field this new, an expert is typically just someone well known, or someone who has made a few, very public correct predictions in the recent past.

In an era of social media, a multitude of YouTube channels, podcasts, and other organizations have sprung up to act as gatekeepers and commentators on the ICO ecosystem. The more successful of these shows have over 100,000 subscribers interested in cryptocurrencies and ICO projects.

Successful ICOs are therefore highly motivated to get on these shows to raise awareness of their project. In turn, the operators of these shows can charge whatever the market will bear in order to get a presence. It's not unusual for high-visibility shows and podcasts to charge $5-20,000 USD for a single appearance, paid in a combination of fiat, crypto, and project tokens.

PR campaigns

Mainstream press attention helps every project. PR can be one of the more time-consuming aspects of ICO marketing, because few ICO teams already have the requisite media contacts. PR agencies can help, but tend to be expensive. Whether or not a PR agency is used, an ICO should have everything else carefully lined up first. Mainstream media attention will bring a lot of attention, which also means a lot more people looking for flaws. A well-executed whitepaper, solid website, a good and clear plan, and fantastic team bios are all important.

Some PR outreach should generally start as soon as the project is solid. Typically, smaller venues and publications are more welcoming than larger ones. A smart ICO team will work with local media, local podcasts, regional business hubs, newsletters, and so on to start with. Once an ICO has some media exposure, it will be easier to interest larger publications.

Naturally, this PR activity needs to go hand-in-hand with content marketing and community building activities.

Content marketing

ICOs usually have at least one person, and often the whole team, doing at least some level of content marketing. The goal of content marketing is to provide information about the project, its goals, and how it will impact the world in a way that is relevant but rarely about sales. Content marketing is usually done through company blogs and platforms, such as Medium or Steemit, where there will be many casual users who may get introduced to the project through the different articles.

Another aspect of content marketing is social content—Tweeter, LinkedIn posts, Facebook, and so on. The purpose is the same—to connect with people who may not otherwise know about the project and build a sense of connection, trust, and purpose.

Content marketing is usually published as coming from the team members, whether or not they actually wrote the content. Doing this helps make the team feel accessible and real—a serious problem when some ICO scams have made up team members.

Good content marketing also helps drive community growth and engagement. ICO teams that have excellent content marketing are much more likely to succeed than ones who do not.

ICO reviewers

As ICO marketing grew, and the number of projects increased, the market started looking for better ways to evaluate projects. A number of companies sprang up to offer impartial reviews of ICO projects. Self-described experts rate projects on different scales to recommend how good a project is for technical and investment success. However, many projects have taken to paying reviewers, or at least offering to pay, in order to get a higher rating.

While there are a large number of ICO rating sites, the two best known are currently **ICOBench** and **ICORating**. Many projects work hard to get a high score on these sites so that they can feature this rating in their marketing materials. Whether or not these ratings are accurate or not is still being worked out and will change. However, every marker of trust an ICO can get surely helps so many projects work hard (fairly or unfairly) to get good ratings.

Smart contract and prototype development

At the start of the ICO frenzy, it was common for projects to be able to raise money with only a whitepaper, a team, and an idea that involved the blockchain. Very quickly, a number of projects raised money and either failed or were drawn into protracted legal disputes. As a result, the market matured and, increasingly, investors wish to see some level of working prototype before they will buy into an ICO. In many jurisdictions, having a working product makes the case for a utility token versus a security token more plausible and reduces the potential for legal issues later on.

The need for prototypes creates a barrier to entry for under-funded or non-funded projects, making them closer to traditional fundraising with VC and angel investors. In many cases, such projects need to get angel or seed funding before they are in a position to do an ICO.

Because of the demand for prototypes, a number of firms have begun offering blockchain prototyping services where they work with business stakeholders who do not have their own development team to build something they can take to market.

Depending on the blockchain system being used, there is also the need for smart contract and token development work. With Ethereum in particular, there is a need for careful testing because, once deployed, code is immutable—bugs cannot be fixed.

Code audits

For many Ethereum ICOs, one of the final steps is a code audit. In a code audit, a trusted third-party is brought in to inspect the code for any possible security issues or violations of best practices. Typically, this audit is publicly released, along with updated code that fixes any important issues found.

Bounty campaigns

A bounty campaign is when a project promises payment for services in their native tokens. These services are typically promotional in some way, such as translating the whitepaper into different languages, writing articles on medium or Steemit, or other tasks to help spread the word. By offering a bounty, the team both spreads the token (and thus the ecosystem), as well as incentivizes people to evangelize the project. After all, by holding the token, the people performing the bounties stand to benefit if the project takes off.

There is no real limit to what can and can't be a bounty. The important part for every team is to make sure that the people who perform the bounties are rewarded, otherwise it's easy to turn a supporter into an enemy.

Airdrops

Airdrops are a promotional approach where a team sends free tokens to all accounts on a network that meet certain criteria, typically involving minimum balance and activity requirements. The airdrop is meant to achieve two goals: spread awareness and interest in the project, and also build a userbase for the ecosystem.

One of the reasons Airdrops became popular is that they were also seen as a way to distribute tokens without being seen as a security, since no money changed hands. This approach is still untested legally.

Road shows

A road show is when the core team for a project travels from conference to conference and to other events across the world promoting their project. The goal is to put the project in front of as many people as possible and to meet as many investors and other influencers as possible. A road show is draining and expensive, but often necessary for projects to gain strong investor support. Many investors, especially large investors, want to get to know the founders and see what they are made of. If investors and influencers have strong confidence in project leadership, they are more likely to invest. Building relationships in person is one way to do that, and the easiest way for many projects to meet investors is through attending multiple events over time, sometimes months. Ideally, this process begins long before the ICO, to give the project team members a long time to build these relationships and get feedback on ideas without seeming needy or asking for money.

Issues with ICOs and blockchain projects

The ICO market is going through growing pains. As a global, mostly unregulated market, there are numerous issues and challenges that do not exist with traditional fundraising. There have been a number of criticisms of ICO projects.

Proof of product-market fit and traction

In more traditional VC-backed or bootstrapped projects, it would be necessary for a project to show value through some form of traction. Typically, this would involve having customers, revenue, and growth, if not profitability. However, with ICOs, the money is raised before most projects launch or show an MVP. In many cases, there is no proof of the team's ability to deliver whatsoever. Because of this, it is impossible for ICO buyers to be sure that the project will address a real problem in the market successfully.

The result of all these weak projects is a high rate of failure. A survey from `bitcoin.com` found that almost half of all projects had already failed, ceased operations, or simply vanished in 2017 alone.

Low barrier to entry

One of the attractions to ICOs is that traditional VC funding is very hard to come by for most of the world. If someone does not live in New York City, San Francisco, or some other major technology hub, then they are far less likely to have access to a serious investor network. With ICOs, it is possible for anyone, anywhere, to attempt to attract early-stage investment. The downside, however, is that the level of due diligence and skill of those doing the investment is questionable and the amount of credible information is low.

An ICO typically starts with the release of a whitepaper, a relatively short document going over the value of the network, the token, and future plans. Because the technology is new, and a typical ICO campaign is short (3-4 months), there is very little time and opportunity for investors to do extensive due diligence. Moreover, because blockchain based startups are not required to show financials (they don't have any), traction, or product-market fit (there is no product yet), there would be little data on which to base evaluations.

In time, consistent standards for evaluating blockchain startups and ICOs may evolve, but, currently, they do not exist.

Does a project really need the blockchain?

Because of the low barrier to entry and the large amounts of available capital, there is strong financial pressure on companies to do an ICO if at all possible. The ability to raise substantial capital without issuing equity, and without well-defined legal obligations, is an opportunity that's seen as too good to pass up.

In many cases, projects don't truly need a blockchain or a token. In fact, unless the features of blockchain are absolutely vital to a project, it is cheaper and easier to go forward without it. To require a blockchain, a project will need to make quality use of blockchain-enabled features: decentralized governance, immutability, true digital scarcity, and so on. Similarly, projects that release tokens on public blockchains must sacrifice privacy and speed. It is imperative that anyone evaluating blockchain projects keeps these issues in mind.

Misleading token practices

Despite ICO and IPO being similar from the point of view of their names there is no real connection between blockchain tokens and company shares. Ownership of a blockchain token backed by a company provides no ownership, influence, or rights to the company's profits, behavior, or business in any way. The value of the token is instead driven entirely by the value of the network itself and what that network enables. Moreover, if a company creates a blockchain project and then abandons it, the token holders likely have no recourse.

Because of this, the way tokens are released can have a huge effect on how a company supports a blockchain project. For instance, if a team were to sell 90% of network tokens in an ICO, then in the future, they will receive only 10% of the benefit of the rise in value of the token. In comparison, they may have millions of dollars in cash on hand. Because of this, the team may decide to give only a limited amount of attention to the blockchain project and feel little urgency in terms of having improve the network. A small team with tens of millions of dollars could pay themselves large salaries until they die and be very secure.

On the other hand, a team releasing only 10% of tokens would be strongly incentivized to increase the value of the token, but there would be a different issue: centralization. A small group would overwhelmingly control the network. In most cases, this would defeat most of the purpose of blockchain technology, leading to the preceding issue—do they really need a blockchain to begin with?

Another problematic issue with tokens is their liquidity. With high token liquidity, a team of investors may be more incentivized to create hype for the network rather than substance. If a project can create enough buzz, the value of their token may increase tenfold. If the token is then liquid, team members and early investors could dump the token for a huge profit and then move on. No longer having anything at stake, the project might be abandoned.

For these reasons, high-quality projects often have some system of investor and team lockup, preventing team members and large investors from selling the token at all for a period of time followed, by a slow vesting where tokens are released over time. By preventing liquidity, the team must focus on long-term value versus short-term manipulation.

Legality

Before we discuss the legalities of blockchain ICOs, it's important to make clear that none of the authors of this book are lawyers and that nothing in this book can constitute or replace quality legal advice. The rules around blockchain technology vary radically between countries and continue to evolve rapidly. For any blockchain project, we suggest that you consult with a skilled local lawyer who has experience in the sector.

The creation of blockchain tokens resulted in an entirely new asset class, one that did not fit well into the existing categories of stock, currency, or equity. Moreover, because public blockchains are global, it is not clear how to apply local laws to the use of blockchains.

Utility versus Security

In the United States, there is a distinction between a token that is a utility and one that is a security. A utility token could be described as something such as a carnival token, a comic book, or a baseball card. While the market value of the item may go up or down, the fundamental reason for purchasing it (and the way it is advertised) is not directly related to profit-seeking. Someone buys a carnival token to play a game, a comic book to read, and a baseball card to collect.

Many tokens try to position themselves as utilities in order to avoid invoking US securities law which greatly restricts sales and requires formal registration with the SEC along with requirements for other disclosures to investors.

In other cases, ICOs issue a **SAFT** or **Simple Agreement for Future Tokens**. The SAFT is absolutely a security and tissuers accept this in order to sell to accredited investors before the launch of the network. Once the network has launched, these SAFT agreements are converted to tokens on the network.

The security versus utility token classification has been complicated by statements from the SEC, that a token can become more or less of a security over time—beginning life as a security or utility and then drifting into the other category. For projects trying to stay on the right side of the law, such fuzzy definitions can be maddening.

Other considerations

Security laws aside, blockchain projects can also be seen as currency and thus run into another set of laws: those governing banks and other money-handling businesses. In the United States and elsewhere, companies that deal with remittances, payments, and other common blockchain use-cases could be seen as money transmitters. In the United States, such businesses must be licensed on a state-by-state basis. Other use cases may require a banking license.

For a global enterprise using blockchain, the regulatory burden to fully comply with the mix of laws and regulations—know-your-customer laws, anti-money laundering laws, money transmitter laws, banking laws, security sales and marketing laws, and so on can be immense.

For these reasons, many blockchain companies tend to defensively headquarter themselves in friendly jurisdictions with the hope of being legal in their primary home and dealing with the rest of the world later, if ever.

Sustainability

Projects in the ICO space, such as blockchain itself, operate in a unique fashion. Depending on the project, they may operate along the lines of a traditional business, but also as a central bank—adding and removing currency from circulation. For a traditional company, profitability is paramount and straightforward, in that a company offers a product and earns a return from sales. However, if you are also creating a currency from nothing and selling and trading it for some other product, or if this currency IS the product, then the situation is more complicated. If the sustainability of the project comes from selling currency, then a downturn can be catastrophic as currency buyers dry up and immense swings in the value of the network can take place nearly instantly. Just as was the case with failed state economy, trust in the currency could evaporate and, as a result, hyper-inflation ensues as individual tokens become nearly worthless as a medium of exchange.

Because company, based blockchain projects are so new, it's still not clear what the long-term sustainable models look like for these companies. Earlier projects, such as Bitcoin and Ethereum, did not need to turn a traditional project but just attract enough currency interest to pay the small team of developers.

Advantages of ICOs

Despite all the issues, ICOs have advantages that traditional fundraising does not. Indeed, one of the takeaways from the booming ICO market is that traditional avenues to capital for startups have been far too restrictive. There is a clear hunger for access to early-stage tech companies.

Liquidity

The biggest advantage for ICO investors is potential liquidity. In traditional equity funding, backers needed to wait for a liquidity event—either a merger or acquisition or a public offering. Any of these options could be a long time coming, if ever they arrived at all. It was not unusual for years and years to pass with no liquidity. With tokens, the sale can happen rapidly after the initial raise is finished. Many tokens would come onto the market mere months after the initial offering. This allowed successful investors to exit part of their position, collect profit, and reinvest. The ability to exit also allowed those investors to re-enter, increasing the pool of available early-stage money overall.

Lack of gatekeepers

Traditional equity investment is highly regulated. This is meant to protect investors, but also acts as a barrier to entry for smaller actors who may have spare capital to invest, but cannot because they are insufficiently wealthy. As a result, those investors cannot invest at all until a company goes public. As a result, smaller investors miss out on the high-risk, high-return options that venture capitalists have access to.

The ability of smaller investors to become involved means that investor-fans, people who invest for reasons other than a pure profit motive, can be a far more powerful force. It's worth asking which is the better investment—a company that can convince a VC to give them $10 million, or a company that can convince 1,000 people to part with $10,000.

Minimal investment sizes

Many ICOs have a minimum investment size in the hundreds of dollars, or even the low thousands. This is far, far below the hundreds of thousands or millions needed at even the seed level in traditional equity raises. As a result, the pool of possible investors increases exponentially. Moreover, by spreading the investment over a much larger pool of people, the risk to any one investor is more limited. Overall, ICOs provide a funding model that is more prone to scams but also more democratic and better suited to projects that may not have profit motive as their #1 priority. Projects for the social good, which could never make economic sense to a venture capitalist, might still find support in an ICO from people who wish to support the cause and take the opportunity for profit as just a bonus.

Notable scams

The ability to raise vast sums of money in only a few months with nothing more than a whitepaper and some advertising naturally gave rise to bad actors looking to cash in. Numerous ICOs began to spring up with fake teams, fake projects, and projects that were questionable at best. Once the projects had collected the money, the teams would vanish. There have been a number of arrests, but there are still plenty of scammers and frauds who have so far escaped with investor money.

Onecoin

Onecoin was an international Ponzi scheme posing as a blockchain project. The project was labeled a scheme by India, while authorities from China, Bulgaria, Italy, Vietnam, Thailand, Finland, and Norway have warned investors about the project. There have been a number of arrests and seizures throughout the world but the Onecoin (`https://www.onecoin.eu/en/`) website continues to operate.

Pincoin and iFan

Pincoin and iFan were two projects claiming to be from Singapore and India, but were secretly backed by a company in Vietnam called Modern Tech, based out of Ho Chi Minh City. It is currently believed to be the largest scam in ICO history, having managed to scam 32,000 people for over $660 million dollars. The leaders of the scam held events and conferences and went to extreme lengths to convince investors of the viability of the projects. In truth, it was a massive Ponzi scheme with early investors paid with the money from later investors. Once the money had been gathered, the team vanished, along with the money.

Bitconnect

Bitconnect was long accused of being a Ponzi scam, as they promised tremendous returns to the owners of their BCC coin. Bitconnect operated an exchange and lending program where users could lend BCC coins to other users in order to make interest off the loans. The company ceased operations after cease-and-desist letters from state lawmakers. The scheme collapsed, along with the value of the coin, leaving most investors with massive losses. A class-action lawsuit is in progress.

Other problems

Beyond outright fraud, ICOs have been plagued with other issues. Because of the amount of money involved and the fact that most funds are raised via cryptocurrency transactions that cannot be reversed or intercepted, ICOs are a perfect target for hackers.

Major hacks

To the author Please add something here

The DAO

One of the earliest decentralized projects was known as the **DAO**, or **Decentralized Autonomous Organization**. The DAO was meant to function as a VC fund of sorts for blockchain projects and was built so that people could buy a stake in the organization (and its profits) using Ethereum. The project itself was run using the Ethereum blockchain and smart contracts to control funds and manage voting. The project was a massive success, managing to raise about $250 million in funds at a time when Ethereum was trading for around $20.

Unfortunately, a flaw in the smart contract code used to operate the DAO resulted in a loophole that a hacker was able to exploit. Because of the subtle bugs in the smart contracts, the attacker was able to withdraw funds multiple times before balances would update. The hacker was then able to withdraw as much as they liked, quickly draining the DAO of tens of millions of dollars worth of ether. This hack radically affected the course of the Ethereum project by causing a hard fork—the majority of users voted to split the network and refund the stolen funds. The minority started Ethereum classic, where all the attackers, actions were allowed to stand. Both networks still operate independently.

Parity

The Parity team is one of the most respected in the entire Ethereum ecosystem. Led by Gavin Wood, one of the founders of Ethereum, they are some of the most experienced and skilled blockchain developers in the world. Unfortunately, everyone is human and the Parity wallet product had a flaw. This flaw allowed an attacker to drain wallets remotely, resulting in millions of dollars of ether being stolen. Thankfully, the attack was not automated, which gave the ecosystem time to notice and respond.

The flaw was fixed, but the fix itself created a new bug. This bug allowed a new attacker to issue a kill command to the wallet, freezing all funds. As of the time of writing, over $260 million in ether remains locked. The community is still trying to figure out a way to rescue the funds.

The moral of the story with the Parity wallet hacks is that even the best teams can make mistakes, and that it's vital that any code on Ethereum has some sort of upgrade path. It's also clear that until the ecosystem improves, any code running on Ethereum should be seen as risky. If even the founders aren't perfect, that should tell you the difficulty involved in doing it right.

Securing an ICO

If contemplating an ICO, it is critical that every effort is made to secure the funds against attack. Here, we will discuss some common attacks and how to defend against them.

SSH key locked servers

Ideally, all servers used by an ICO team should have login access restricted to a known whitelist of ssh keys. This means that only the computers known to the team can log into the server. Even better is if these same machines remain shut down and disconnected from the internet immediately prior to and during the ICO. Thus, even if that computer were compromised, it could not be used to hack the ICO.

DNS security

One of the ways that ICOs have been hacked is by attackers creating a clone of the ICO website and then hacking DNS to redirect the domain name to a computer the attackers control. The new site looks exactly such as the official one, except the address where to send funds is changed. Because most addresses are hexadecimal and not effortlessly human distinguishable, this is an easy detail to overlook. If this happens for even a few minutes during a busy ICO, hackers can make off with hundreds of thousands or millions of dollars.

Many DNS providers have different ways of locking DNS, and these security measures should all be enabled. Moreover, a DNS checking service where such as DNS Spy, should be used to regularly check for any changes in real time. Taking these countermeasures helps ensure that attackers trying to hijack DNS to steal funds will not be successful.

Intrusion detection

Any machines used to run the website for an ICO should have some level of intrusion detection software installed as a backup. If, for some reason, something was missed, it's critical that the team be notified of any suspicious behavior on key servers. There are numerous intrusion detection products on the market. Key features to look for are modifications of permissions detection and file change detection (for instance, altering an HTML file to change the Ethereum address as to where to send funds).

Purchase related domains

Another way ICOs can be attacked is through the purchase of similar sounding or looking domains. For instance, if a team is running on `myIcoDomain.io`, the attacker might buy `myIcoDomain.net`. It's important for teams to make this harder by buying as many related domains as they can, especially with the most common extensions. If an attacker owns one of these related domains, they can easily send emails to possible buyers, post messages on social media, and so on, in a way that may confuse some buyers. Just as with DNS attacks, attackers will often put up an identical looking site except for the critical payment details.

Moreover, ICO teams should make very clear what the official domain is and regularly and proactively warn users to mistrust anything else.

Monitor social channels

Attackers also may try and steal funds by injecting messages into Facebook groups, telegram groups, discord channels, and other communication platforms to steal funds. For instance, a user may show up and declare that they are helping the team and could everyone please fill out some survey or other. The survey collects email addresses and, naturally, those email addresses are then sent emails that look as though they are coming from a related domain to send funds or get a special sale.

Who can and cannot speak for the ICO should be extremely clear, and anyone who claims to speak for the ICO team but doesn't should be flagged and banned immediately.

Multi-signature wallets

Funds collected by an ICO should be moved into a multi-signature wallet. This is because another possible attack is to compromise the wallet holding the funds, allowing the attacker to transfer everything to themselves. Multi-signature wallets exist for most major crypto ecosystems, and radically increase the difficulty for attackers. Instead of stealing the private key for one person, they must now steal multiple keys from multiple computers and use them all simultaneously. In the Ethereum ecosystem, the Gnosis wallet has proven robust and capable here, used by multiple successful teams without incident.

Code audits

While a best practice in general, having a public code audit for any crowd sale code helps to decrease the likelihood of a hack. By being proactive in looking for and removing vulnerabilities with the help of skilled third-parties, a team both increases trust in the project and decreases the chance that something has been missed that a hacker could exploit.

Conclusion

By now, you should have a good idea of what goes into an ICO and some idea of the immense work that goes into it. A typical ICO can go for minutes to months, but it takes an incredible amount of prep work to get marketing materials done for public and private investors. Any ICO project should be extremely careful about security, as an ICO that gets hacked and loses funds will find it very hard to inspire confidence in the project and in future projects.

Many ICOs fail, either because of a lack of marketing, the wrong team, or an inability to engage the community. The early days of easy money are over. Still, ICOs are, on average, vastly more successful than traditional approaches to gaining funds, such as approaching Venture Capitalists.

In the next chapter, we will be looking into how to create your own token or cryptocurrency.

References

- https://www.forbes.com/sites/laurashin/2017/09/21/heres-the-man-who-created-icos-and-this-is-the-new-token-hes-backing/#1191f6431183
- https://www.forbes.com/sites/kashmirhill/2014/06/03/mastercoin-maidsafe-crowdsale/#2b8c819d207d
- https://blog.ethereum.org/2014/07/22/launching-the-ether-sale/
- https://coinmarketcap.com/all/views/all/
- https://news.bitcoin.com/46-last-years-icos-failed-already/
- https://www.bloomberg.com/graphics/2016-who-gets-vc-funding/
- https://www.mirror.co.uk/news/uk-news/who-wants-onecoin-millionaire-you-7346558
- https://spectrum.ieee.org/computing/networks/do-you-need-a-blockchain
- http://www.businessinsider.com/bitcoin-price-what-is-a-saft-blockchain-the-crypto-fundraising-craze-shaking-up-venture-capital-2017-11
- https://coincenter.org/entry/sec-s-clayton-use-of-a-token-can-evolve-toward-or-away-from-being-a-security
- https://cointelegraph.com/news/unpacking-the-5-biggest-cryptocurrency-scams
- http://vietnamnews.vn/economy/426152/city-investigates-fraud-case.html#y6cCVwoFYZDJXsJH.97
-]https://cointelegraph.com/news/investors-file-class-action-lawsuit-against-bitconnect-following-its-shutdown
- http://observer.com/2017/12/cryptocurrency-startup-centra-tech-sued-by-investors-over-ico-fraud/
- https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/

20
Creating Your Own Currency

Up until now, we have talked extensively about blockchain, Bitcoin's, and Altcoins. We have discussed various intricacies involved in Bitcoin's, its blockchain, and other elements that form the blockchain. Ethereum has been a good topic of interest for most parts of the book so far. We have also read about other blockchain-based projects that comprise currency-based projects.

Ethereum has made it possible to make decentralized applications despite having little development knowledge. There are various tokens available at exchanges that are built on top of Ethereum and other similar projects, and are being recognized by backers and the community.

In this chapter, we will be discussing the following topics:

- Types of cryptocurrency
- Setting up Litecoin
- Forking the Litecoin repository
- Creating one's own coin on top of the Litecoin repository

Understanding types of cryptocurrency

There are three ways in which one's own cryptocurrency can be created. Each has its own benefits over the other. Prior to creating a currency, it is important to know all the intricacies involved, and what your coin offers compared to other existing coins currently available at exchanges. A list of popular coins can be found at `https://coinmarketcap.com`.

Tokens based on existing blockchains

These are the tokens that are based on existing blockchain platforms, such as Ethereum, Omni, NEO, and so on. We have already discussed development on token in the previous `Chapter 15`, *Ethereum Development*. A token helps in creating a coin faster, and supports a quick go-to market strategy so that most of the time can be invested in further development of a self-hosted blockchain.

There are examples of various blockchain projects, which start as a token and post a successful ICO once they start their own fully supported blockchain complete with a coin. Since it is faster and easier to develop a token as compared to a coin, it becomes easy for such projects to gain traction and also get launched in no time, and most of the time can be invested in other important tasks such as whitepaper creation, ICO, and so on.

Creating a new blockchain from scratch

In this method, creation of one's own blockchain from scratch can be done. Taking hints from Bitcoin and other blockchain platforms will be helpful to create the coin, which will allow you to integrate new features and also to use different consensus techniques.

A forked blockchain with one's own genesis block

Our prime focus in this chapter is about this type of coin development; we will be creating our own coin complete with its own blockchain, after the introduction of Platforms such as Ethereum, Counterparty, NEO, and so on. There have been limited resources with creating one's own blockchain by taking fork of an existing coin such as Bitcoin, Litecoin, etc. and work on that to create own cryptocurrency.

Litecoin's development

Litecoin is one of the first cryptocurrencies built on top of Bitcoin. Litecoin's source was forked from Bitcoin's core code, including the wallet and other sources of Bitcoin. The major change that Litecoin has over Bitcoin is that the PoW algorithm is script instead of Bitcoin's SHA-256. Apart from this, Litecoin has a coin supply limit of 84,000,000 LTC, and the block time is 2.5 minutes.

The process

In this chapter, we will be forking Litecoin source code and working on top of that. Here is a brief overview of the steps involved:

- Making the layout and the requirements of the project.
- Opting of the algorithm, various algorithms can be opted for this process. In this chapter, we are going forward with the script algorithm, which is used in Litecoin itself.
- **Consensus type**: PoW, PoS, or any other consensus type can be used based on community support. In this chapter, we will be using the PoW consensus type, which will require miners to mine for the coin and also confirm transactions.
- **Coin name**: The name of the coin has to be decided.
- **Coin abbreviation**: Such as for Bitcoin, BTC is used, and for Litecoin, LTC is used; similarly, an abbreviation is required, so better to have the abbreviation be similar to the coin name.
- **Port for connection**: It is important to select the port for connection with the network; this port will be used by every node connected to the blockchain network.
- **Block reward**: It is important to set the block reward, that is, the amount of coins that will be rewarded when a block is mined by the miners.
- **Block halving time**: This is the time when the block reward will be halved, for example, in Bitcoin the block reward is halved every 210,000 blocks, which controls the production of the coins.
- **Coin supply limit**: This is the limit of coins that will be produced by all the miners in total; this is generally controlled by the block halving time, since after a certain amount of blocks, it will not be feasible to mine any more blocks.
- **Coinbase maturity**: It is important to set the number of blocks that are to be mined before the mined coins can be spent which were received in a block mining reward.
- **Confirmations**: This is the number of blocks that are to be mined before a transaction is confirmed.
- **Difficulty re-adjustment time**: For example, Bitcoin has a difficulty re-adjustment time of two weeks; similarly, it is required to set this time during the development process.
- **Block mining time**: Total time which should be spent in the mining of a block.

- **Seed node**: This is the node that is the starting node of the coin; it is important to have a node that is always online till enough nodes are synced and connected to the network. It is acceptable to have multiple seed node addresses available. We can also opt for DNS seeds, which are nothing but DNS servers which contains addresses for the seed nodes on the blockchain network.
- **Wallet UI**: The core wallet is built on a QT framework, and its GUI can be changed as per requirements.
- **Graphic assets**: Icons for the coin and other graphic assets can be chosen and replaced in the Litecoin source; it is also suggested to maintain the sizing properties of the icons.

Creating one's own cryptocurrency

Once the previously listed parameters are defined, it is time to work on the source code and make the required changes, as needed.

Setting up Litecoin

It is important to set up the Litecoin environment in our local machine; the source code is available on GitHub here: `https://github.com/litecoin-project/litecoin`.

Platform selection

It is important to select the build platform on which the environment is to be set up. You can find the required build instruction in the `doc` subfolder of the source code. There, required instruction files for the preferred platforms are present for you to follow the steps to install the Litecoin core and the Wallet.

In this section, we will be working on Max OS X build instructions, although instructions for other platforms are also available in the suggested instruction path.

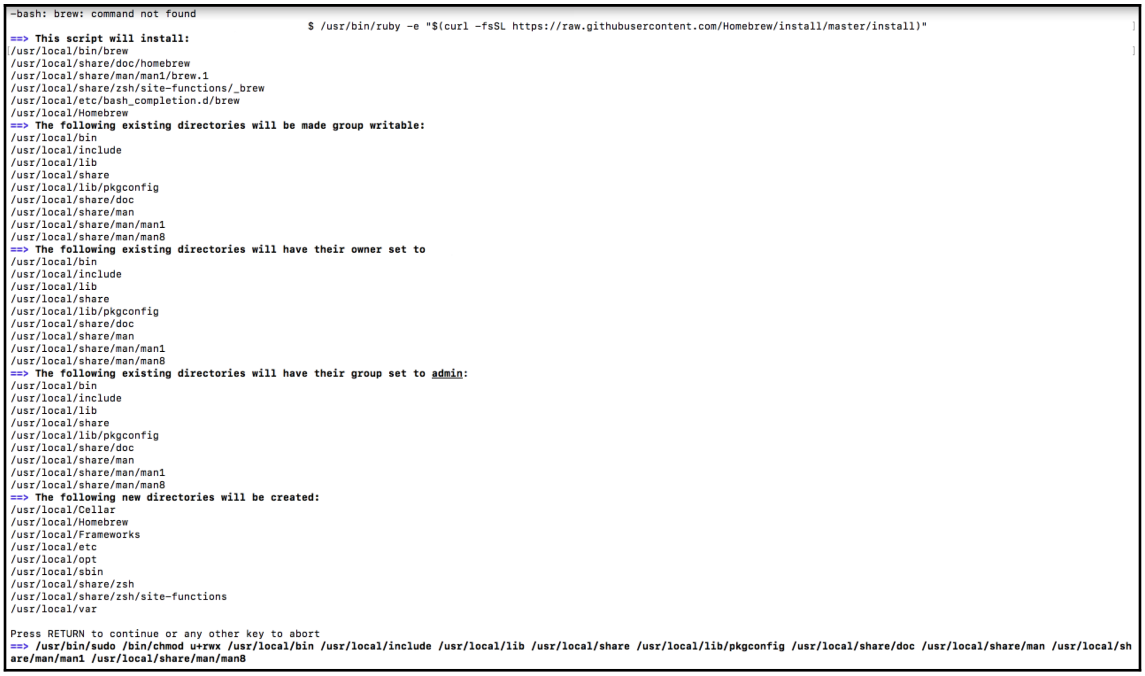
Preparation

Installation of `xcode` is necessary for compilation of required dependencies. The following command should be executed in the terminal:

```
xcode-select --install
```

Installation of `brew`, which is a package manager for macOS, is needed next. The following command is used to install `brew`:

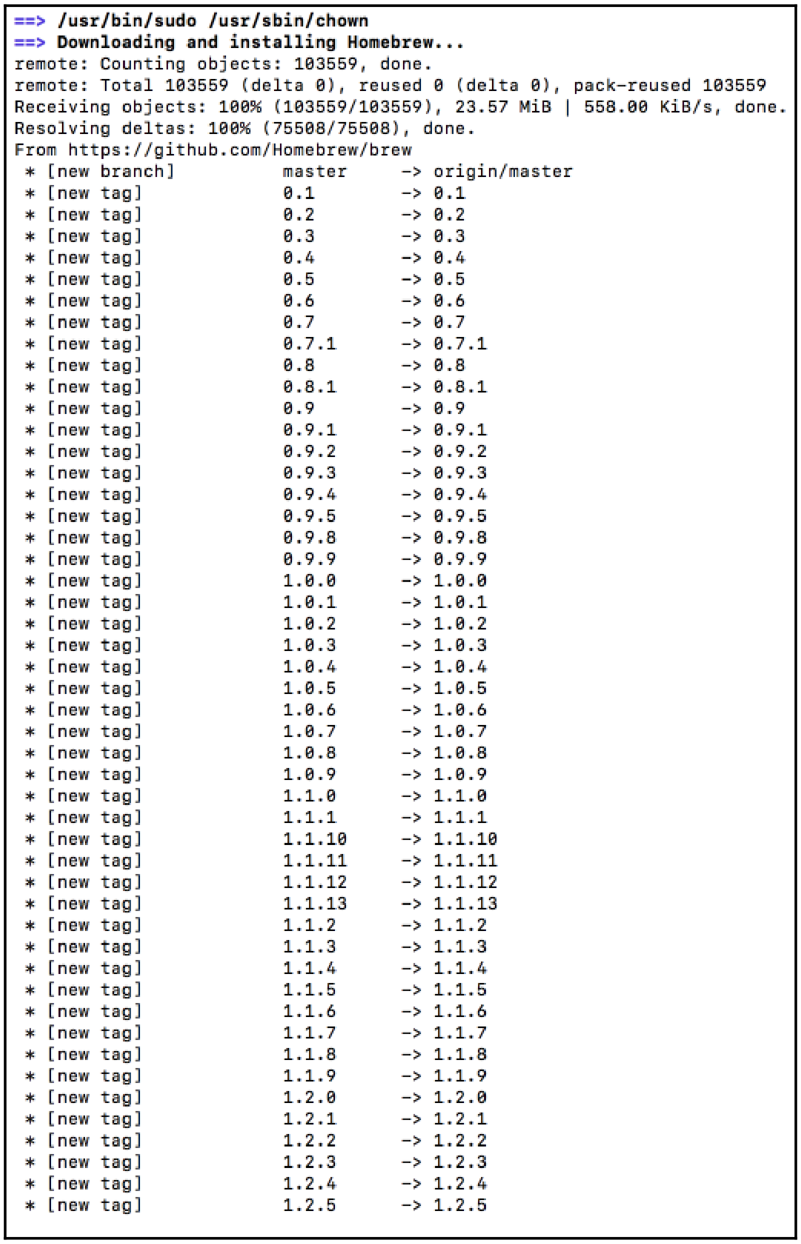
```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```



```
-bash: brew: command not found
                                        $ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
==> The following existing directories will be made group writable:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/lib/pkgconfig
/usr/local/share/doc
/usr/local/share/man
/usr/local/share/man/man1
/usr/local/share/man/man8
==> The following existing directories will have their owner set to
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/lib/pkgconfig
/usr/local/share/doc
/usr/local/share/man
/usr/local/share/man/man1
/usr/local/share/man/man8
==> The following existing directories will have their group set to admin:
/usr/local/bin
/usr/local/include
/usr/local/lib
/usr/local/share
/usr/local/lib/pkgconfig
/usr/local/share/doc
/usr/local/share/man
/usr/local/share/man/man1
/usr/local/share/man/man8
==> The following new directories will be created:
/usr/local/Cellar
/usr/local/Homebrew
/usr/local/Frameworks
/usr/local/etc
/usr/local/opt
/usr/local/sbin
/usr/local/share/zsh
/usr/local/share/zsh/site-functions
/usr/local/var

Press RETURN to continue or any other key to abort
==> /usr/bin/sudo /bin/chmod u+rwx /usr/local/bin /usr/local/include /usr/local/lib /usr/local/share /usr/local/lib/pkgconfig /usr/local/share/doc /usr/local/share/man /usr/local/sh
are/man/man1 /usr/local/share/man/man8
```

Dependency installation

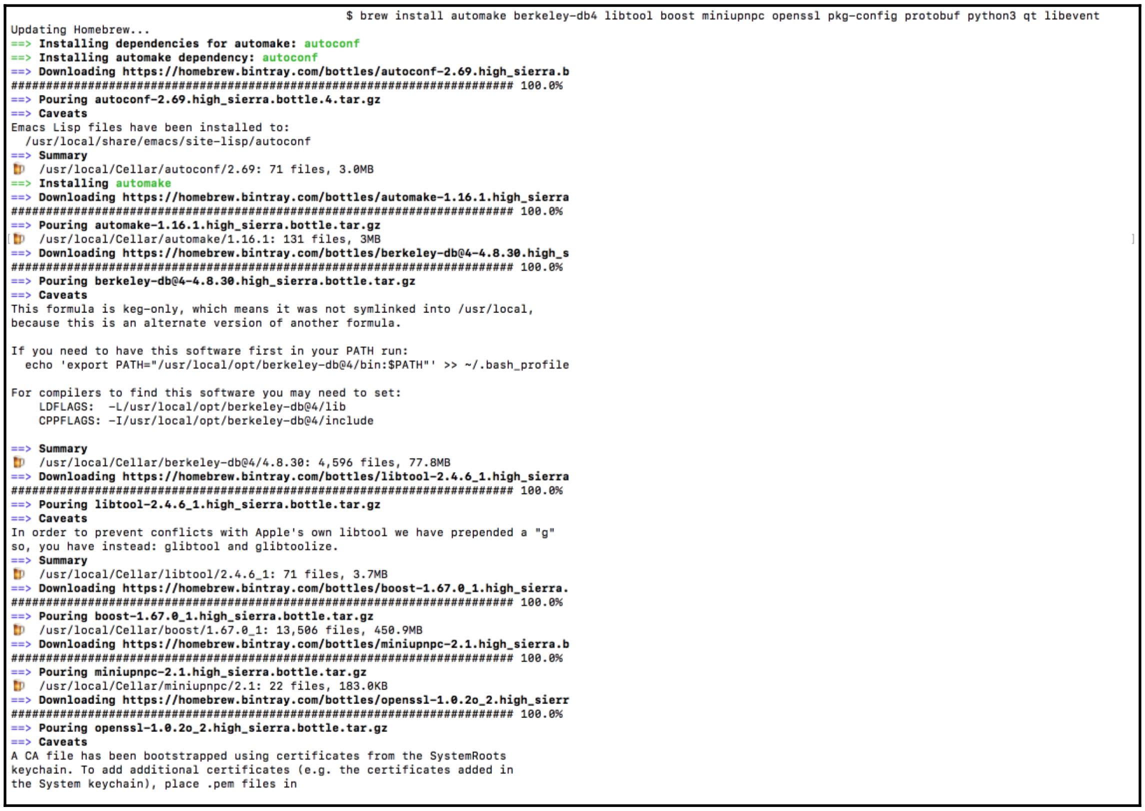
Once `brew` is installed, the next step is to install the required dependencies using the following command:



```
==> /usr/bin/sudo /usr/sbin/chown
==> Downloading and installing Homebrew...
remote: Counting objects: 103559, done.
remote: Total 103559 (delta 0), reused 0 (delta 0), pack-reused 103559
Receiving objects: 100% (103559/103559), 23.57 MiB | 558.00 KiB/s, done.
Resolving deltas: 100% (75508/75508), done.
From https://github.com/Homebrew/brew
 * [new branch]          master     -> origin/master
 * [new tag]             0.1        -> 0.1
 * [new tag]             0.2        -> 0.2
 * [new tag]             0.3        -> 0.3
 * [new tag]             0.4        -> 0.4
 * [new tag]             0.5        -> 0.5
 * [new tag]             0.6        -> 0.6
 * [new tag]             0.7        -> 0.7
 * [new tag]             0.7.1      -> 0.7.1
 * [new tag]             0.8        -> 0.8
 * [new tag]             0.8.1      -> 0.8.1
 * [new tag]             0.9        -> 0.9
 * [new tag]             0.9.1      -> 0.9.1
 * [new tag]             0.9.2      -> 0.9.2
 * [new tag]             0.9.3      -> 0.9.3
 * [new tag]             0.9.4      -> 0.9.4
 * [new tag]             0.9.5      -> 0.9.5
 * [new tag]             0.9.8      -> 0.9.8
 * [new tag]             0.9.9      -> 0.9.9
 * [new tag]             1.0.0      -> 1.0.0
 * [new tag]             1.0.1      -> 1.0.1
 * [new tag]             1.0.2      -> 1.0.2
 * [new tag]             1.0.3      -> 1.0.3
 * [new tag]             1.0.4      -> 1.0.4
 * [new tag]             1.0.5      -> 1.0.5
 * [new tag]             1.0.6      -> 1.0.6
 * [new tag]             1.0.7      -> 1.0.7
 * [new tag]             1.0.8      -> 1.0.8
 * [new tag]             1.0.9      -> 1.0.9
 * [new tag]             1.1.0      -> 1.1.0
 * [new tag]             1.1.1      -> 1.1.1
 * [new tag]             1.1.10     -> 1.1.10
 * [new tag]             1.1.11     -> 1.1.11
 * [new tag]             1.1.12     -> 1.1.12
 * [new tag]             1.1.13     -> 1.1.13
 * [new tag]             1.1.2      -> 1.1.2
 * [new tag]             1.1.3      -> 1.1.3
 * [new tag]             1.1.4      -> 1.1.4
 * [new tag]             1.1.5      -> 1.1.5
 * [new tag]             1.1.6      -> 1.1.6
 * [new tag]             1.1.7      -> 1.1.7
 * [new tag]             1.1.8      -> 1.1.8
 * [new tag]             1.1.9      -> 1.1.9
 * [new tag]             1.2.0      -> 1.2.0
 * [new tag]             1.2.1      -> 1.2.1
 * [new tag]             1.2.2      -> 1.2.2
 * [new tag]             1.2.3      -> 1.2.3
 * [new tag]             1.2.4      -> 1.2.4
 * [new tag]             1.2.5      -> 1.2.5
```

```
brew install automake berkeley-db4 libtool boost miniupnpc openssl pkg-
config protobuf python3 qt libevent
```

The preceding command will install all the required dependencies, as needed:



```
                                          $ brew install automake berkeley-db4 libtool boost miniupnpc openssl pkg-config protobuf python3 qt libevent
Updating Homebrew...
==> Installing dependencies for automake: autoconf
==> Installing automake dependency: autoconf
==> Downloading https://homebrew.bintray.com/bottles/autoconf-2.69.high_sierra.b
######################################################################## 100.0%
==> Pouring autoconf-2.69.high_sierra.bottle.4.tar.gz
==> Caveats
Emacs Lisp files have been installed to:
  /usr/local/share/emacs/site-lisp/autoconf
==> Summary
🍺  /usr/local/Cellar/autoconf/2.69: 71 files, 3.0MB
==> Installing automake
==> Downloading https://homebrew.bintray.com/bottles/automake-1.16.1.high_sierra
######################################################################## 100.0%
==> Pouring automake-1.16.1.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/automake/1.16.1: 131 files, 3MB
==> Downloading https://homebrew.bintray.com/bottles/berkeley-db@4-4.8.30.high_s
######################################################################## 100.0%
==> Pouring berkeley-db@4-4.8.30.high_sierra.bottle.tar.gz
==> Caveats
This formula is keg-only, which means it was not symlinked into /usr/local,
because this is an alternate version of another formula.

If you need to have this software first in your PATH run:
  echo 'export PATH="/usr/local/opt/berkeley-db@4/bin:$PATH"' >> ~/.bash_profile

For compilers to find this software you may need to set:
    LDFLAGS:  -L/usr/local/opt/berkeley-db@4/lib
    CPPFLAGS: -I/usr/local/opt/berkeley-db@4/include

==> Summary
🍺  /usr/local/Cellar/berkeley-db@4/4.8.30: 4,596 files, 77.8MB
==> Downloading https://homebrew.bintray.com/bottles/libtool-2.4.6_1.high_sierra
######################################################################## 100.0%
==> Pouring libtool-2.4.6_1.high_sierra.bottle.tar.gz
==> Caveats
In order to prevent conflicts with Apple's own libtool we have prepended a "g"
so, you have instead: glibtool and glibtoolize.
==> Summary
🍺  /usr/local/Cellar/libtool/2.4.6_1: 71 files, 3.7MB
==> Downloading https://homebrew.bintray.com/bottles/boost-1.67.0_1.high_sierra.
######################################################################## 100.0%
==> Pouring boost-1.67.0_1.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/boost/1.67.0_1: 13,506 files, 450.9MB
==> Downloading https://homebrew.bintray.com/bottles/miniupnpc-2.1.high_sierra.b
######################################################################## 100.0%
==> Pouring miniupnpc-2.1.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/miniupnpc/2.1: 22 files, 183.0KB
==> Downloading https://homebrew.bintray.com/bottles/openssl-1.0.2o_2.high_sierr
######################################################################## 100.0%
==> Pouring openssl-1.0.2o_2.high_sierra.bottle.tar.gz
==> Caveats
A CA file has been bootstrapped using certificates from the SystemRoots
keychain. To add additional certificates (e.g. the certificates added in
the System keychain), place .pem files in
```

Build instructions

The first step is to clone the Litecoin in the root or any other directory:

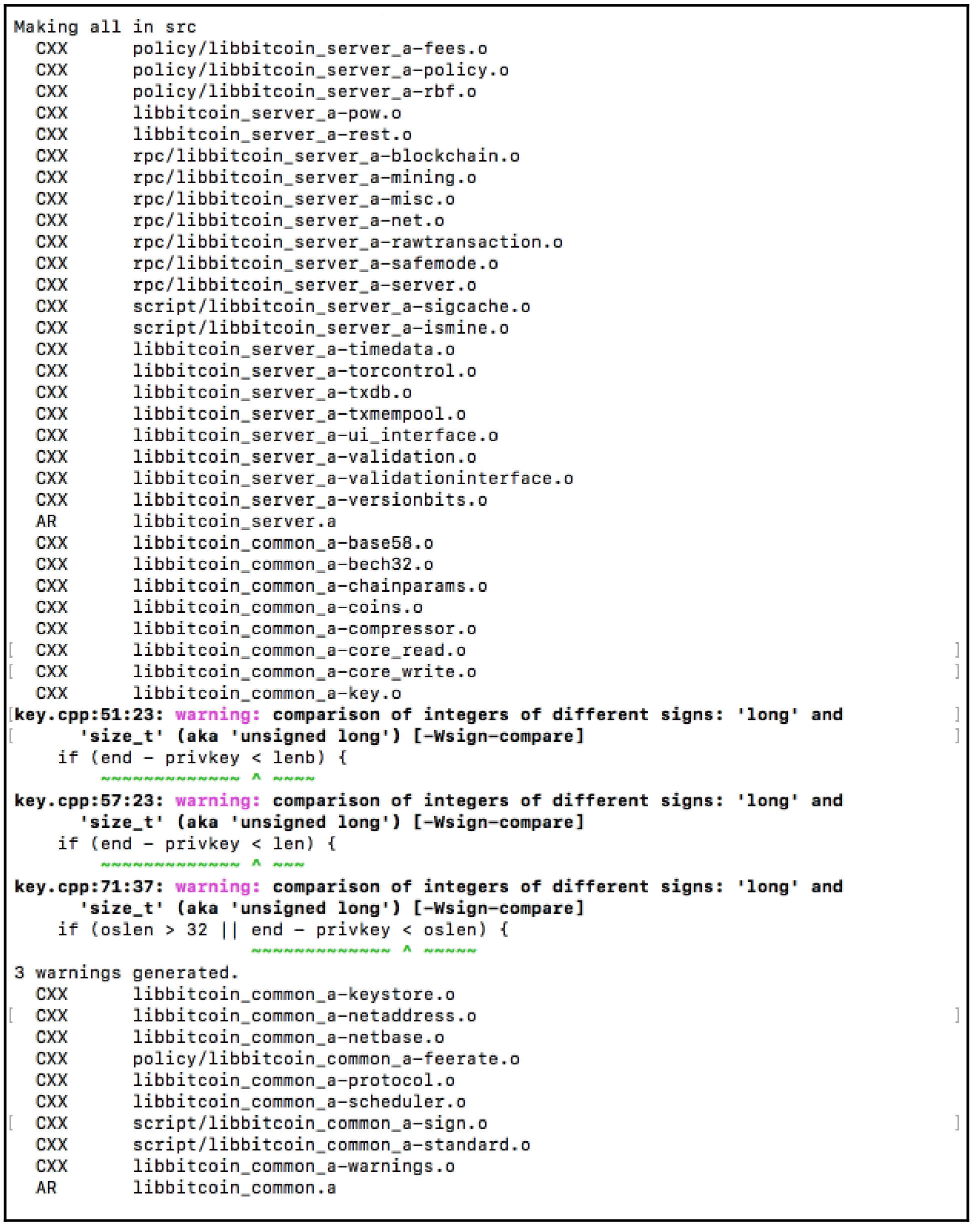
```
git clone https://github.com/litecoin-project/litecoin
cd Litecoin
```

Installation of `BerkleyDb` is performed with the following command:

```
./contrib/install_db4.sh .
```

Building of Litecoin-core uses the following make command:

```
./autogen.sh
./configure
make
```

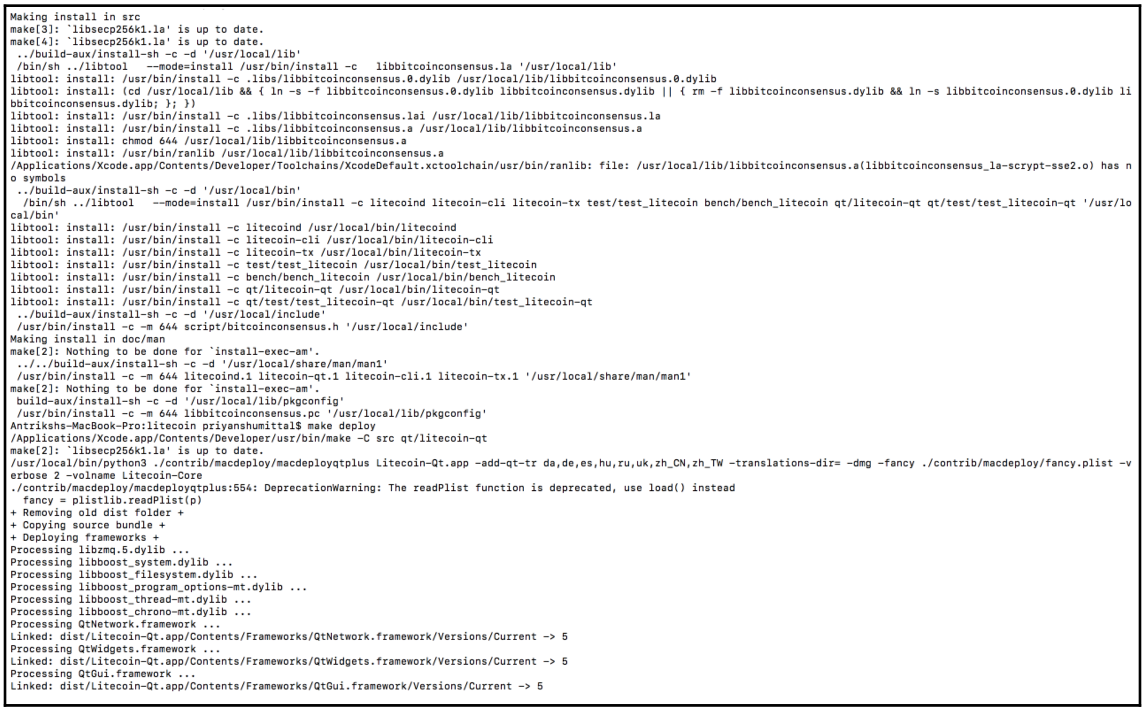


```
Making all in src
  CXX      policy/libbitcoin_server_a-fees.o
  CXX      policy/libbitcoin_server_a-policy.o
  CXX      policy/libbitcoin_server_a-rbf.o
  CXX      libbitcoin_server_a-pow.o
  CXX      libbitcoin_server_a-rest.o
  CXX      rpc/libbitcoin_server_a-blockchain.o
  CXX      rpc/libbitcoin_server_a-mining.o
  CXX      rpc/libbitcoin_server_a-misc.o
  CXX      rpc/libbitcoin_server_a-net.o
  CXX      rpc/libbitcoin_server_a-rawtransaction.o
  CXX      rpc/libbitcoin_server_a-safemode.o
  CXX      rpc/libbitcoin_server_a-server.o
  CXX      script/libbitcoin_server_a-sigcache.o
  CXX      script/libbitcoin_server_a-ismine.o
  CXX      libbitcoin_server_a-timedata.o
  CXX      libbitcoin_server_a-torcontrol.o
  CXX      libbitcoin_server_a-txdb.o
  CXX      libbitcoin_server_a-txmempool.o
  CXX      libbitcoin_server_a-ui_interface.o
  CXX      libbitcoin_server_a-validation.o
  CXX      libbitcoin_server_a-validationinterface.o
  CXX      libbitcoin_server_a-versionbits.o
  AR       libbitcoin_server.a
  CXX      libbitcoin_common_a-base58.o
  CXX      libbitcoin_common_a-bech32.o
  CXX      libbitcoin_common_a-chainparams.o
  CXX      libbitcoin_common_a-coins.o
  CXX      libbitcoin_common_a-compressor.o
  CXX      libbitcoin_common_a-core_read.o
  CXX      libbitcoin_common_a-core_write.o
  CXX      libbitcoin_common_a-key.o
key.cpp:51:23: warning: comparison of integers of different signs: 'long' and
      'size_t' (aka 'unsigned long') [-Wsign-compare]
    if (end - privkey < lenb) {
        ~~~~~~~~~~~~~ ^ ~~~~
key.cpp:57:23: warning: comparison of integers of different signs: 'long' and
      'size_t' (aka 'unsigned long') [-Wsign-compare]
    if (end - privkey < len) {
        ~~~~~~~~~~~~~ ^ ~~~
key.cpp:71:37: warning: comparison of integers of different signs: 'long' and
      'size_t' (aka 'unsigned long') [-Wsign-compare]
    if (oslen > 32 || end - privkey < oslen) {
                      ~~~~~~~~~~~~~ ^ ~~~~~
3 warnings generated.
  CXX      libbitcoin_common_a-keystore.o
  CXX      libbitcoin_common_a-netaddress.o
  CXX      libbitcoin_common_a-netbase.o
  CXX      policy/libbitcoin_common_a-feerate.o
  CXX      libbitcoin_common_a-protocol.o
  CXX      libbitcoin_common_a-scheduler.o
  CXX      script/libbitcoin_common_a-sign.o
  CXX      script/libbitcoin_common_a-standard.o
  CXX      libbitcoin_common_a-warnings.o
  AR       libbitcoin_common.a
```

You can run unit tests to make sure the build is successful without any errors:

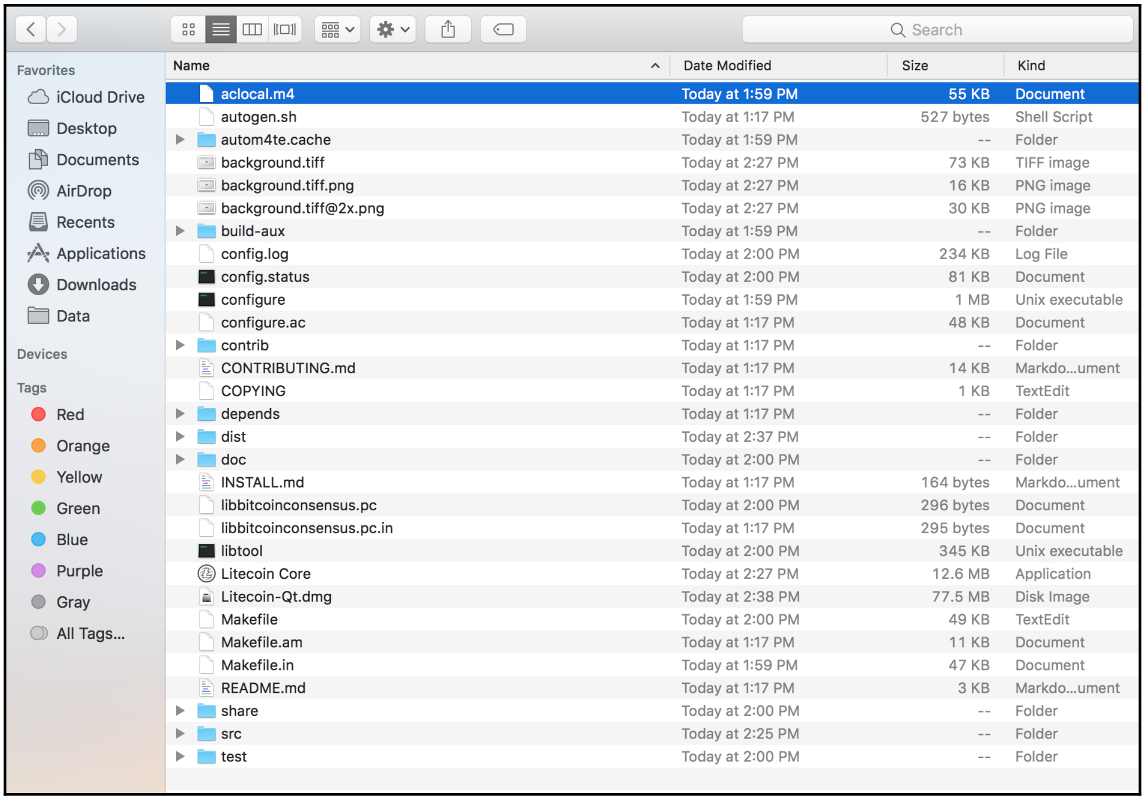
```
make check
```

Deployment of the `.dmg` which contains the `.app` bundles, is shown here:



```
Making install in src
make[3]: `libsecp256k1.la' is up to date.
make[4]: `libsecp256k1.la' is up to date.
 ../build-aux/install-sh -c -d '/usr/local/lib'
 /bin/sh ../libtool   --mode=install /usr/bin/install -c   libbitcoinconsensus.la '/usr/local/lib'
libtool: install: /usr/bin/install -c .libs/libbitcoinconsensus.0.dylib /usr/local/lib/libbitcoinconsensus.0.dylib
libtool: install: (cd /usr/local/lib && { ln -s -f libbitcoinconsensus.0.dylib libbitcoinconsensus.dylib || { rm -f libbitcoinconsensus.dylib && ln -s libbitcoinconsensus.0.dylib li
bbitcoinconsensus.dylib; }; })
libtool: install: /usr/bin/install -c .libs/libbitcoinconsensus.lai /usr/local/lib/libbitcoinconsensus.la
libtool: install: /usr/bin/install -c .libs/libbitcoinconsensus.a /usr/local/lib/libbitcoinconsensus.a
libtool: install: chmod 644 /usr/local/lib/libbitcoinconsensus.a
libtool: install: /usr/bin/ranlib /usr/local/lib/libbitcoinconsensus.a
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/ranlib: file: /usr/local/lib/libbitcoinconsensus.a(libbitcoinconsensus_la-scrypt-sse2.o) has n
o symbols
 ../build-aux/install-sh -c -d '/usr/local/bin'
  /bin/sh ../libtool   --mode=install /usr/bin/install -c litecoind litecoin-cli litecoin-tx test/test_litecoin bench/bench_litecoin qt/litecoin-qt qt/test/test_litecoin-qt '/usr/lo
cal/bin'
libtool: install: /usr/bin/install -c litecoind /usr/local/bin/litecoind
libtool: install: /usr/bin/install -c litecoin-cli /usr/local/bin/litecoin-cli
libtool: install: /usr/bin/install -c litecoin-tx /usr/local/bin/litecoin-tx
libtool: install: /usr/bin/install -c test/test_litecoin /usr/local/bin/test_litecoin
libtool: install: /usr/bin/install -c bench/bench_litecoin /usr/local/bin/bench_litecoin
libtool: install: /usr/bin/install -c qt/litecoin-qt /usr/local/bin/litecoin-qt
libtool: install: /usr/bin/install -c qt/test/test_litecoin-qt /usr/local/bin/test_litecoin-qt
 ../build-aux/install-sh -c -d '/usr/local/include'
 /usr/bin/install -c -m 644 script/bitcoinconsensus.h '/usr/local/include'
Making install in doc/man
make[2]: Nothing to be done for `install-exec-am'.
 ../../build-aux/install-sh -c -d '/usr/local/share/man/man1'
 /usr/bin/install -c -m 644 litecoind.1 litecoin-qt.1 litecoin-cli.1 litecoin-tx.1 '/usr/local/share/man/man1'
make[2]: Nothing to be done for `install-exec-am'.
 build-aux/install-sh -c -d '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 libbitcoinconsensus.pc '/usr/local/lib/pkgconfig'
Antrikshs-MacBook-Pro:litecoin priyanshumittal$ make deploy
/Applications/Xcode.app/Contents/Developer/usr/bin/make -C src qt/litecoin-qt
make[2]: `libsecp256k1.la' is up to date.
/usr/local/bin/python3 ./contrib/macdeploy/macdeployqtplus Litecoin-Qt.app -add-qt-tr da,de,es,hu,ru,uk,zh_CN,zh_TW -translations-dir= -dmg -fancy ./contrib/macdeploy/fancy.plist -v
erbose 2 -volname Litecoin-Core
./contrib/macdeploy/macdeployqtplus:554: DeprecationWarning: The readPlist function is deprecated, use load() instead
  fancy = plistlib.readPlist(p)
+ Removing old dist folder +
+ Copying source bundle +
+ Deploying frameworks +
Processing libzmq.5.dylib ...
Processing libboost_system.dylib ...
Processing libboost_filesystem.dylib ...
Processing libboost_program_options-mt.dylib ...
Processing libboost_thread-mt.dylib ...
Processing libboost_chrono-mt.dylib ...
Processing QtNetwork.framework ...
Linked: dist/Litecoin-Qt.app/Contents/Frameworks/QtNetwork.framework/Versions/Current -> 5
Processing QtWidgets.framework ...
Linked: dist/Litecoin-Qt.app/Contents/Frameworks/QtWidgets.framework/Versions/Current -> 5
Processing QtGui.framework ...
Linked: dist/Litecoin-Qt.app/Contents/Frameworks/QtGui.framework/Versions/Current -> 5
```

```
make deploy
```



Name	Date Modified	Size	Kind
aclocal.m4	Today at 1:59 PM	55 KB	Document
autogen.sh	Today at 1:17 PM	527 bytes	Shell Script
autom4te.cache	Today at 1:59 PM	--	Folder
background.tiff	Today at 2:27 PM	73 KB	TIFF image
background.tiff.png	Today at 2:27 PM	16 KB	PNG image
background.tiff@2x.png	Today at 2:27 PM	30 KB	PNG image
build-aux	Today at 1:59 PM	--	Folder
config.log	Today at 2:00 PM	234 KB	Log File
config.status	Today at 2:00 PM	81 KB	Document
configure	Today at 1:59 PM	1 MB	Unix executable
configure.ac	Today at 1:17 PM	48 KB	Document
contrib	Today at 1:17 PM	--	Folder
CONTRIBUTING.md	Today at 1:17 PM	14 KB	Markdo...ument
COPYING	Today at 1:17 PM	1 KB	TextEdit
depends	Today at 1:17 PM	--	Folder
dist	Today at 2:37 PM	--	Folder
doc	Today at 2:00 PM	--	Folder
INSTALL.md	Today at 1:17 PM	164 bytes	Markdo...ument
libbitcoinconsensus.pc	Today at 2:00 PM	296 bytes	Document
libbitcoinconsensus.pc.in	Today at 1:17 PM	295 bytes	Document
libtool	Today at 2:00 PM	345 KB	Unix executable
Litecoin Core	Today at 2:27 PM	12.6 MB	Application
Litecoin-Qt.dmg	Today at 2:38 PM	77.5 MB	Disk Image
Makefile	Today at 2:00 PM	49 KB	TextEdit
Makefile.am	Today at 1:17 PM	11 KB	Document
Makefile.in	Today at 1:59 PM	47 KB	Document
README.md	Today at 1:17 PM	3 KB	Markdo...ument
share	Today at 2:00 PM	--	Folder
src	Today at 2:25 PM	--	Folder
test	Today at 2:00 PM	--	Folder

Setting up our own coin

Now, it is time to work on our own coin; for this, it is important to check out the clone directory and Litecoin, and make a copy in case any of the steps turn out to be fatal.

As per the parameters we have listed in one of the previous sections, it is time to replace the parameters at the requisite locations.

The `src` directory contains the Litecoin core source code where most of the parameters are to be set. In the `chainparams.cpp` file, change the abbreviation from `LTC` to the abbreviation of our choice. Similarly, all other files where the abbreviation seems fit is to be changed as suggested.

Port selection

Now, the ports need to be changed, so that our blockchain works at the relevant port for all the nodes in the network.

The connection port should be changed in the `init.cpp` file.

The RPC port should be changed in the `bitcoinrpc.cpp` file.

The setting of block-related parameters

In the `validation.cpp` file, the following parameters should be edited:

- Block value
- Block reward
- Block time
- Difficulty re-target time
- Difficulty re-target scale

The block value should be changed in the `GetBlockSubsidy()` function:

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params&
consensusParams)
{
 int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
 // Force block reward to zero when right shift is undefined.
 if (halvings >= 64)
 return 0;

 CAmount nSubsidy = 50 * COIN;
 // Subsidy is cut in half every 210,000 blocks which will occur
approximately every 4 years.
 nSubsidy >>= halvings;
 return nSubsidy;
}
```

Amount limit

Now, it is time to set the coin limit and the minimum value, and the same can be done in the `amount.h` file:

```
typedef int64_t CAmount;

static const CAmount COIN = 100000000;
static const CAmount CENT = 1000000;

static const CAmount MAX_MONEY = 84000000 * COIN;
inline bool MoneyRange(const CAmount& nValue) { return (nValue >= 0 &&
nValue <= MAX_MONEY); }

#endif // BITCOIN_AMOUNT_H
```

The coinbase maturity number

To change the coinbase maturity, in the `qt` subfolder, the `transactionrecord.cpp` file should be changed to set the required number of blocks that are to be found before the mined coins can be spent:

```
   if (nNet > 0 || wtx.IsCoinBase())
     {
         //
         // Credit
         //
         for(unsigned int i = 0; i < wtx.tx->vout.size(); i++)
         {
             const CTxOut& txout = wtx.tx->vout[i];
             isminetype mine = wallet->IsMine(txout);
             if(mine)
             {
                 TransactionRecord sub(hash, nTime);
                 CTxDestination address;
                 sub.idx = i; // vout index
                 sub.credit = txout.nValue;
                 sub.involvesWatchAddress = mine & ISMINE_WATCH_ONLY;
                 if (ExtractDestination(txout.scriptPubKey, address) &&
IsMine(*wallet, address))
                 {
                     // Received by Bitcoin Address
                     sub.type = TransactionRecord::RecvWithAddress;
                     sub.address = EncodeDestination(address);
                 }
                 else
```

```
                {
                    // Received by IP connection (deprecated features), or
a multisignature or other non-simple transaction
                    sub.type = TransactionRecord::RecvFromOther;
                    sub.address = mapValue["from"];
                }
                if (wtx.IsCoinBase())
                {
                    // Generated
                    sub.type = TransactionRecord::Generated;
                }

                parts.append(sub);
            }
        }
    }
    else
    {
        bool involvesWatchAddress = false;
        isminetype fAllFromMe = ISMINE_SPENDABLE;
        for (const CTxIn& txin : wtx.tx->vin)
        {
            isminetype mine = wallet->IsMine(txin);
            if(mine & ISMINE_WATCH_ONLY) involvesWatchAddress = true;
            if(fAllFromMe > mine) fAllFromMe = mine;
        }

        isminetype fAllToMe = ISMINE_SPENDABLE;
        for (const CTxOut& txout : wtx.tx->vout)
        {
            isminetype mine = wallet->IsMine(txout);
            if(mine & ISMINE_WATCH_ONLY) involvesWatchAddress = true;
            if(fAllToMe > mine) fAllToMe = mine;
        }

        if (fAllFromMe && fAllToMe)
        {
            // Payment to self
            CAmount nChange = wtx.GetChange();

            parts.append(TransactionRecord(hash, nTime,
TransactionRecord::SendToSelf, "",
                            -(nDebit - nChange), nCredit - nChange));
            parts.last().involvesWatchAddress = involvesWatchAddress; //
maybe pass to TransactionRecord as constructor argument
        }
```

We have to change the transaction confirm count in the `transactionrecord.cpp` file to set this parameter, as per our requirements.

Genesis block creation

The genesis block is created from the `LoadBlockIndex()` function, which is present inside the `validation.cpp` file:

```
bool CChainState::LoadBlockIndex(const Consensus::Params& consensus_params,
CBlockTreeDB& blocktree)
{
    if (!blocktree.LoadBlockIndexGuts(consensus_params, [this](const
uint256& hash){ return this->InsertBlockIndex(hash); }))
        return false;

    boost::this_thread::interruption_point();

    // Calculate nChainWork
    std::vector<std::pair<int, CBlockIndex*> > vSortedByHeight;
    vSortedByHeight.reserve(mapBlockIndex.size());
    for (const std::pair<uint256, CBlockIndex*>& item : mapBlockIndex)
    {
        CBlockIndex* pindex = item.second;
        vSortedByHeight.push_back(std::make_pair(pindex->nHeight, pindex));
    }
    sort(vSortedByHeight.begin(), vSortedByHeight.end());
    for (const std::pair<int, CBlockIndex*>& item : vSortedByHeight)
    {
        CBlockIndex* pindex = item.second;
        pindex->nChainWork = (pindex->pprev ? pindex->pprev->nChainWork :
0) + GetBlockProof(*pindex);
        pindex->nTimeMax = (pindex->pprev ?
std::max(pindex->pprev->nTimeMax, pindex->nTime) : pindex->nTime);
        // We can link the chain of blocks for which we've received
transactions at some point.
        // Pruned nodes may have deleted the block.
        if (pindex->nTx > 0) {
            if (pindex->pprev) {
                if (pindex->pprev->nChainTx) {
                    pindex->nChainTx = pindex->pprev->nChainTx +
pindex->nTx;
                } else {
                    pindex->nChainTx = 0;
                    mapBlocksUnlinked.insert(std::make_pair(pindex->pprev,
pindex));
                }
            } else {
                pindex->nChainTx = pindex->nTx;
            }
        }
        if (!(pindex->nStatus & BLOCK_FAILED_MASK) && pindex->pprev &&
(pindex->pprev->nStatus & BLOCK_FAILED_MASK)) {
            pindex->nStatus |= BLOCK_FAILED_CHILD;
            setDirtyBlockIndex.insert(pindex);
        }
        if (pindex->IsValid(BLOCK_VALID_TRANSACTIONS) && (pindex->nChainTx
|| pindex->pprev == nullptr))
            setBlockIndexCandidates.insert(pindex);
        if (pindex->nStatus & BLOCK_FAILED_MASK && (!pindexBestInvalid ||
pindex->nChainWork > pindexBestInvalid->nChainWork))
            pindexBestInvalid = pindex;
        if (pindex->pprev)
            pindex->BuildSkip();
        if (pindex->IsValid(BLOCK_VALID_TREE) && (pindexBestHeader ==
nullptr || CBlockIndexWorkComparator()(pindexBestHeader, pindex)))
            pindexBestHeader = pindex;
    }

    return true;
}
```

Further, in chainparams.cpp, the paraphrase is to be changed to the requisite parameter of choice. In Litecoin, the following parameter is used:

```
    const char* pszTimestamp = "NY Times 05/Oct/2011 Steve Jobs, Apple's
Visionary, Dies at 56";
```

Wallet address

The wallet address is an important part of the blockchain, as without having correct private and public keys it is not possible to perform the relevant changes on the source of Litecoin. The starting letter can be set in the base58.cpp file.

Checkpoints

Checkpoints are hardcoded into the Litecoin Core client. With checkpoints set, all transactions are valid up to the checkpoint condition valid. This is set in case anyone wants to fork the blockchain and start form the very same block; the checkpoint will render false and won't accept any further transactions. The `checkpoints.cpp` file helps in managing the checkpoints in a blockchain source code:

```
namespace Checkpoints {

    CBlockIndex* GetLastCheckpoint(const CCheckpointData& data)
    {
        const MapCheckpoints& checkpoints = data.mapCheckpoints;

        for (const MapCheckpoints::value_type& i :
reverse_iterate(checkpoints))
        {
            const uint256& hash = i.second;
            BlockMap::const_iterator t = mapBlockIndex.find(hash);
            if (t != mapBlockIndex.end())
                return t->second;
        }
        return nullptr;
    }
```

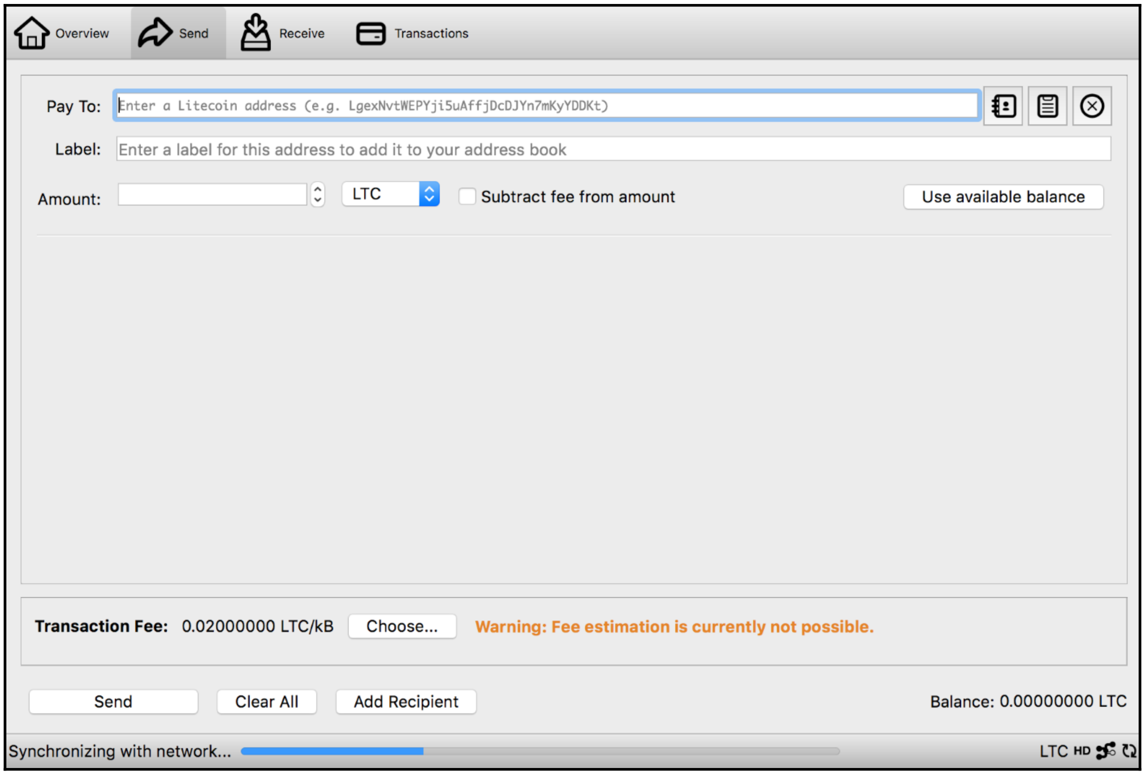
Creatives and graphics

Icons and other graphics can be set and replaced from the `src/qt/res/icons` directory, which contains all the images and the main logo of the coin.

The files `bitcoin.png` and `about.png` contain the logo for the specific coin.

Summing up

By following the preceding points, the coin can be created and made to work by using the Litecoin source; once the coin is created and made to work, the next steps will be to test for actual production level usage. Further steps include compiling the coin source and using the QT-based wallet:



Pay To: Enter a Litecoin address (e.g. LgexNvtWEPYji5uAffjDcDJYn7mKyYDDKt)

Label: Enter a label for this address to add it to your address book

Amount: LTC Subtract fee from amount Use available balance

Transaction Fee: 0.02000000 LTC/kB Choose... Warning: Fee estimation is currently not possible.

Send Clear All Add Recipient Balance: 0.00000000 LTC

Synchronizing with network... LTC HD

Summary

In this chapter, we discussed the creation of our own cryptocurrency, since we had already discussed the creation of tokens based on the Ethereum platform in `Chapter 15`, *Ethereum Development*. In this chapter, we worked on the creation of our own blockchain, by taking a fork of Litecoin, and then making the required changes to create the desired coin with the relevant parameters.

In the following chapters, we will address both sides of the coin in terms of blockchain's future, that is, scalability and other challenges that blockchain is facing. Furthermore, we will discuss the future of blockchain, and how it is going to shape the future of the internet and technology, not just in terms of currency-based industries, but also in terms of other industries in which blockchain acts as a game changer.

21
Scalability and Other Challenges

While the blockchain technology has been hailed as a major breakthrough, it is not without its own issues. Blockchain technology is still in the early stages of its development, and a number of problems have cropped up that are still awaiting some level of resolution. Decentralized networks such as blockchains have their own unique challenges.

In this chapter, we'll look at the following key issues in blockchain:

- Scalability and decentralization
- Scalability and costs
- Usability
- 51% attacks
- Network forks
- Catastrophic bugs
- Lack of interoperability
- Low availability of blockchain skills

Scalability and decentralization

One of the key advantages of blockchain is decentralization, which is the removal of any single authority to control the network. Unfortunately, this has a downside, which is its effect on the performance of a system. Blockchain systems work by keeping all the nodes of the network in sync by trying to achieve consensus so that every computer running a blockchain sees the same system state. More nodes on the network typically results in less centralization. This also means that more work must be done to ensure that all the network participants are in agreement with each other, which limits performance and scalability.

There are a few reasons why a larger number of nodes hinders performance:

- Each node typically must process all transactions. The higher the number of transactions to be processed, the more processing power and network throughput that each node requires. As requirements go up, the network becomes less decentralized as fewer and fewer groups can afford to put the necessary machines on the network. In some areas, bandwidth may be limited. Once this limited bandwidth is exceeded, people in that area cannot participate in the network.
- More nodes also leads to more latency, as the number of peer-to-peer communications increases as each peer propagates messages across the network. While peer-to-peer communication is very efficient for this purpose, it can still lead to lagging, and limit maximum throughput.
- Many blockchains require that participating nodes store the entire blockchain on the local hard drive. Both the storage requirement and the extremely long amount of time it takes to download the chain from the network act as barriers to participation, especially in areas with less consistent networks. For instance, the Ethereum blockchain is now over 400 GB in size. Without compression, this would be too big for the majority of personal laptops and many older desktop computers to store without requiring a dedicated machine to be used just for operating the blockchain.

There have been a few attempts at increasing the performance of blockchain systems. Most of these involve some level of reduced decentralization. For instance, the Bitshares-derived set of systems uses a restricted set of full nodes called **witnesses**, of which there are only 21. Only these computers are used to process and approve transactions; other machines on the network merely submit transactions to the network or observe. As well as many other performance optimizations made by the Bitshares team, they claim a theoretical throughput of up to 100,000 transactions a second. This leads to a second issue with blockchains, which is the cost compared to traditional approaches.

Blockchains in business

While the Bitshares team suggests a theoretical limit of 100,000 transactions a second, they are using technologies and lessons from the LMAX exchange, which claims to be able to process over 6 million transactions a second. Most blockchains do not achieve a performance that is anywhere near the theoretical performance, with most blockchains achieving well under 1,000 transactions a second in practice. For instance, Bitcoin achieves approximately seven transactions a second, and Ethereum around 14. A decent server running MySQL can process 10,000–20,000 transactions a second of similar complexity. Thus, traditional approaches are much easier to scale to larger transaction volumes than blockchain systems.

The cost per performance of traditional database systems, including distributed database systems, is vastly cheaper on every level than blockchain systems. For any business, hiring experienced personnel for these systems is cheaper, easier, and less risky because they are so much better known and documented. Businesses considering blockchain must ask if a blockchain-based system provides a must-have feature in some way, because if not, other approaches are cheaper, easier, faster, or all of the preceding.

A business that is evaluating blockchain should pay close attention to their scaling requirements and cost models when using blockchain. In time, costs and performance may improve, but there are no guarantees that they will improve fast enough if current approaches aren't sufficient.

Usability

Current blockchain systems are relatively difficult to use in comparison to other systems. For instance, the use of any Ethereum-based DApp requires either the installation of a special browser or the MetaMask plugin for Chrome, or the purchase and transfer of Ethereum on one of the public markets and then learning the interface of the desired application. Each action taken requires the expenditure of Ethereum, the exact amount not necessarily known in advance and varying depending on network load.

Once set up, sending a value via a blockchain-based system is relatively easy, but prone to mistakes in the addresses. It is not easy for a human to know whether they are sending the value to `0x36F9050bb22d0D0d1BE34df787D476577563C4fC` or `0xF973EE1Bcc92d924Af3Fc4f2ce4616C73b58e5Cc`. Indeed, ICOs have been hacked by attackers gaining access to the ICO main website and simply changing the destination address, thereby siphoning away millions.

Some blockchains, such as Bitshares and Stellar, have provisions for named accounts that are intuitive and human readable. Hopefully, this trend will continue and the usability will improve.

Lack of protection

When blockchain networks get hacked, users typically have no recourse. This is true whether or not the user is at fault in any way. When centralized exchanges have been hacked, one of the responses is *th*e users should not have trusted a centralized authority. However, with events such as the parity wallet hack and the DAO, users could lose access to their funds even when they did not trust a centralized authority. While one response could be they should have chosen a better wallet or project, it is unclear how users can realistically be expected to do that. For any Parity wallet hack, the Parity team involves some of the most renowned developers of the Ethereum world. The more likely response to such continued issues is to not use blockchain. To achieve mainstream acceptance, blockchain systems will need to be easier to use and provide some sort of advanced protection in the case of attacks, hacks, and other losses.

51% attacks

All blockchains can suffer from consensus attacks, often referred to as 51% attacks because of the original consensus attack possible in Bitcoin. Every blockchain relies on the majority of its users or stakeholders being good actors, or at least not coordinating against the rest of the network. If the majority (or even a large minority) of the powerful network actors in a blockchain system coordinate against the rest, they will be able to launch double-spend attacks and extract large amounts of value from the network against its will.

While once theoretical, there have recently been a number of successful 51% attacks against different blockchains, such as Verge (find the link in the references). In systems that are more centralized, such as proof-of-stake systems where there may be a small number of extremely large stakeholders, it is entirely possible that similar coordination's could occur with something as simple as a few phone calls if concerned stakeholders have sufficient incentives. Such incentives do not necessarily need to be economic or on-chain at all: Politics or international relations could cause a set of patriotic stakeholders to collude for or against other users. For instance, a large amount of mining hardware and network gear originates in China, as does a lot of blockchain development. If the Chinese stakeholders discovered that they held a majority stake during an international fight with a regional rival, these stakeholders might be able to use their network power to punish network users that belong to the rival region, unilaterally.

Network forks

Another issue for public blockchains is the existence of network forks. Bitcoin has been forked twice, splitting into Bitcoin and Bitcoin cash, then again into Bitcoin Gold. There are now three independent networks claiming the Bitcoin mantle. While the original network is still by far the most dominant, all the forks were a result of a breakdown in agreement among the key network participants about the future of the technology.

Ethereum suffered a similar fate, splitting into Ethereum, and Ethereum Classic over how to respond to the DAO hack. The majority felt that the DAO should have the hacked funds restored, but a strong minority disagreed, asking why should this hack get special treatment over others?. The result was two networks, one with the hacked funds restored (Ethereum) and one where the network protocol was the rule no matter what (Ethereum Classic).

Catastrophic bugs

The upside of an immutable ledger is that nothing can be hidden or altered. The downside of an immutable ledger is that nothing can be hidden or altered—including bugs. In the case where networks such as Ethereum write the smart contract code into the chain itself, this means that code bugs cannot be fixed easily; the original code will remain on the blockchain forever. The only workaround is the modular code where each section references some other section, and these pointers have programmatic ways of being updated. This allows the authors of a DApp to upload a new piece of code and adjust the pointers appropriately. However, this method too has issues. Making these updates requires a specific authority to update the code.

Having a central authority that is necessary for updates just creates new problems. Either that authority has centralized control over the decentralized app (which means it is no longer decentralized) or a governing system must be written onto the blockchain as well. Both these options have security trade-offs. A centralized authority might be hacked or have their private keys stolen, resulting in a catastrophic loss to the system. A decentralized governing system requires substantially more code, and is itself at risk of a hack, with the DAO hack being exactly such an event.

Lack of interoperability

Current blockchain technologies do not easily interoperate. While it is possible to write an application that can communicate with multiple blockchains, those blockchains do not have the natural capability to communicate with each other. In many cases, the fundamental approach to transactions and governance may not be compatible. For instance, in the Ethereum network, any user can send any token to any other user, no permission required by the recipient. The recipient is free to ignore the new tokens if they wish, but they still have them (which results in some interesting tax questions). In the Stellar network, however, a user must issue a trustline to another in order to receive custom tokens issued by that user.

Similarly, many networks offer multisignature and multiuser wallets on-chain. However, without a centralized application sitting outside the blockchains themselves, there is no way for users to easily manage all of these networks in one place. This is part of the appeal of centralized exchanges to begin with—no matter what the underlying protocol is, users get a predictable, universal interface that is able to send and receive tokens without regard for the underlying technology.

Low availability of blockchain skills

As with any new technology, the number of skilled personnel will be limited. In the case of blockchain, this natural order is made far worse because of the large and growing number of systems. Looking only at the major chains, systems are written in C, C++, Java, Scala, Golang, and Python. All of these systems have different architectures and protocols. The blockchains that have smart contracts have different contract models, contract APIs, and even completely different programming languages, such as Solidity and Serpent.

On the security front, each blockchain system has subtly different security models and challenges. The simpler the system, the easier those requirements are, but the less the blockchain will be able to do. Moreover, because of the recent surge of interest, blockchain skills are very expensive on the open market. In particular, top talent with a proven track record is very hard to find.

Privacy

One of the supposed advantages of blockchains is that they can be run anonymously. However, if this anonymity is ever broken, the immutable ledger means that every transaction throughout time can now be traced perfectly. Maintaining perfect anonymity is extremely hard, and even if one person is successful, if the people they do business with are not also anonymous as well, then statistical techniques can be used to narrow down the possibilities of their identity considerably.

While many people think about anonymity in terms of avoiding law enforcement or taxation, it is not only governments that might be interested in this information. For instance, plenty of criminal organizations would love to be able to identify wealthy but unassuming people. An organization could trace large holdings or transactions of cryptoassets to a specific person and use that as a targeting method for kidnapping or extortion. Add the immutability of cryptotransactions and this could make such attacks very attractive.

There are some projects that attempt to address this, such as Zcash and Dash, which hide transaction sources and endpoints. Hopefully, more systems will add well thought out protocols to address safety and privacy.

Energy consumption

Some of the largest blockchains, such as Bitcoin and Ethereum, still work on a proof-of-work model. The proof-of-work approach is extremely power hungry and inefficient. One news report suggested that the Bitcoin network alone already consumes more power than the nation of Ireland. Other sources considered this an exaggeration, but even so, it illustrates the tremendous cost of running these systems.

However, more and more systems are moving from proof-of-work systems to other systems for this exact reason. New consensus algorithms, such as proof-of-stake and delegated-proof-of-stake, make such extreme energy costs unnecessary.

Summary

In this chapter, we have covered the current major challenges of using and implementing blockchain systems. Businesses looking to use blockchains should be aware of the current state of the technology, the costs in comparison to other technological approaches, and the security and staffing issues that exist in the blockchain world.

The good news is that none of theses issues are insurmountable, and many of them will certainly improve with time. Numerous projects are under way to improve the usability and security of different blockchain networks. Additional tooling and training resources reduce the difficulty of programming new blockchain applications.

References

1. `https://bitshares.org/technology/industrial-performance-and-scalability/`
2. `https://news.bitcoin.com/proof-of-work-coins-on-high-alert-following-spate-of-51-attacks/`
3. `https://www.theguardian.com/technology/2017/nov/27/bitcoin-mining-consumes-electricity-ireland`

22
Future of Blockchain

In this chapter, we are going to discuss the future of the blockchain technology. There are a number of trends and developments that are likely to drive the development of the blockchain technology and its adoption in the coming years. As with any sort of prognostication, none of the things that we will discuss are set in stone.

In particular, blockchain will be driven by a few key themes:

- Ongoing fragmentation and specialization
- Legal and regulatory evolution
- Technological stabilization
- Intersection with AI and IoT

Ongoing fragmentation and specialization

Blockchain began with only a single implementation: Bitcoin. Now there are thousands of blockchains and hundreds of blockchain technologies. In such a crowded marketplace, it is only natural that the technology will begin to have more purpose-driven implementations. Many of the drawbacks of blockchain systems, such as the speed, ease of use, and so on, are easier to solve with a purpose-specific blockchain system.

We've already seen the beginnings of this. Steemit is a blockchain-based social network built from bitshares/graphene technology. It has a consensus algorithm based on witnesses and voting, and the whole operation is designed around hosting social and blogging content. While still having many of the features of a public blockchain, by being purpose specific, the Steem blockchain is better able to function in its desired ecosystem.

We expect more purpose-specific chain technologies to emerge. Forks of blockchain specifically aimed at logistics, tracking the providence of artistic works, legal work, and so on are all fertile ground for a good team to provide a market-specific blockchain product that is easy to use and integrate into existing systems.

The companies that successfully provide a cost-saving, effective solution for traditional businesses will find the easiest success. In this section, we will look at some specific use cases that are already evolving.

Video games

Some of the most popular games in the world are online role-playing games, such as World of Warcraft. In many of these games, players can accumulate and trade equipment. A common issue for game creators has been the in-game economy: How do you enforce the scarcity of rare items online? Many games have been subject to duping bugs, where a player can duplicate a rare item over and over again and then sell it, thereby making a substantial profit. Blockchain technology solves the issue of digital scarcity.

There are already a few video games experimenting with blockchain technology. However, while there are a few Ethereum-based tokens for gaming, there are no blockchains just for video games yet. We expect this to change, as some of the most lucrative games around make their money through the sale of in-game items.

Real estate

Real estate is another domain where the blockchain technology makes absolute sense. There are already a number of real-estate-based projects under way. However, the world of real estate is complex, and the laws concerning title ownership and transfer vary widely from jurisdiction to jurisdiction. It is likely that a chain built for managing the private/public interaction of land transference will be able to successfully supplant the existing legacy systems. Such a system would greatly increase the efficiency of property markets. Current markets rely on a large number of distinct factors with incomplete information government agencies holding land title records, insurance companies checking flood zones and other issues, issuing prices, and real-estate sales companies carrying out price discovery and advertising. The need for multiple disconnected parties to share in a trusted system is a perfect fit for blockchain technology. Improving real estate would be difficult, but possibly very lucrative for the company that is successful.

Logistics

There are already a large number of well-funded logistics projects in the blockchain field. Most of these are running on a public network such as Ethereum or NEO. However, it is likely that industry-specific blockchains will evolve, as the legal needs and requirements of a regional trucking company are quite different than those of international cross-border shipping. The system requirements for each space may be very different, yet these systems would need to communicate and coordinate. Here is a space where federated, multichain networks might be a good fit. The technology to handle cross-chain transfers is still developing. However, despite the difficulty of tracking tokens across multiple blockchains, teams such as the Tendermint team (the developers of Cosmos) are hard at work enabling precisely that.

Licensing

Currently, software licenses and other rights to intellectual property are managed and transferred using a large number of proprietary systems, or even old-fashioned paperwork. It can be difficult or even impossible to ascertain the ownership of certain intellectual property. In the US, you automatically own any IP you create; however, this ownership is not recorded anywhere. It is possible to register your ownership, but it's far from effortless. A blockchain-powered IP system would enable sites such as YouTube, DeviantArt, and ModelMayhem to create the ability to automatically register works on behalf of their users. Once registered, those same sites could assist users in seamlessly issuing and revoking licenses. Such a scheme would require the cooperation of these large internet sites, but this would not be the first time such a thing has happened; many of the technical underpinnings of the web were created through industry consortiums looking to increase efficiency and interoperability throughout the industry.

Industry consortiums

While the blockchain technology has become increasingly fragmented, there will come an inevitable consolidation. One of the ways that blockchain will consolidate is within different industries. As with other standard processes, it is likely that each industry in which finds a profitable use for blockchain will begin to standardize on one or two key technologies or implementations that will then become the default. For instance, in the logistics industry, there are many competing projects and standards for the tracking and delivery of packages. Some of these projects are being built on Ethereum, others on NEO, and yet more on their own private chains. One of these approaches, or a new one not yet seen, will hit an industry sweet spot and become a standard. This is because blockchain technology is ideally suited for uniting different factors that may not trust each other. This sweet spot technology will almost certainly be the result of a collection of companies, not a start up. A consortium has a much easier time in not feeling such as a threatening competitor, and will instead have an opportunity to pool resources and be part of a winning team.

Likely consortium's to use blockchain include the banking, insurance, package delivery, and tracking industries. Industries such as document and legal tracking, on the other hand, are more likely to have multiple private competitors because of the different market forces. Medical tracking would greatly benefit from a standardized approach, but market forces and regulatory issues make this more difficult.

A large number of total-loss projects

The current state of the blockchain ecosystem has been compared to the dotcom boom in the 90's. The result will likely be similar: A few key projects survive and thrive, whereas the others fail and become liabilities for investors. However, unlike the dotcom boom, blockchain projects have taken vastly more investment from the general public. This could lead to interesting results. One possibility is an event similar to the dotcom crash: funding dries up for quite some time, regardless of project quality. With retail investors taking some of the damage, as well as having the opportunity to continue to invest anyway, the fallout from a blockchain crash may not be as bad. However, a harsh crash that may severely harm retail investors is more likely to draw a regulatory reaction.

Legal and regulatory evolution

A certain near-term change in the blockchain technology will come from regulation. Nearly every major country is looking at the ICO landscape and working out how best to regulate or capitalize on blockchain. In the USA, the SEC seems to be looking at ICOs more and more as a securities offering that will necessitate companies to carry out an ICO to provide financial and other documentation more typically related to stock offerings.

Other countries, such as Belarus, are taking a very hands-on approach in the hopes of luring blockchain companies and investment to their countries.

Security token offerings

In response to the continued SEC action and the failure of a large number of blockchain projects, an increasing number of projects that are seeking funding will issue security tokens. Investors will inevitably start seeking and demanding higher-quality projects, as the low hanging fruit will have been taken and barriers to entry will increase. In addition, security offerings tend to come with financial disclosure rules and other requirements that will act as a gatekeeper to projects that are simply not ready to be funded. The downside, of course, is that it will be harder for unknown players without independent access to funds to even attempt entry into the market.

Currently, there are no exchanges dedicated to security tokens, and, in fact, many existing exchanges do not wish to trade in such tokens in order to avoid security laws themselves. Once the security tokens become a larger part of the market, STO trading platforms will emerge or potentially be added to the existing stock-trading platforms of major exchanges and banks.

Aggregate and insurance products

One of the common criticisms of blockchain tokens is the high volatility of individual assets. To counter this, there have already been a few attempts at creating token pools, or tokens that represent a basket of other tokens in order to reduce volatility and risk, much like ETFs and mutual funds do for stocks.

We expect further attempts at building more stable and better-hedged token products. One possibility would be a new insurance product, protecting against negative volatility in return for a fee. While in practical terms a similar effect can be achieved with futures and options, insurance is a concept more familiar to ordinary consumers and retail investors who have taken an interest in the token markets. We expect that there will be attempts to bring all of these to the market.

Technological stabilization

Technologies tend to go in waves. There is usually a period of growth and experimentation followed by consolidation and standardization before the cycle begins again. Currently, there are a few technologies that are clearly in the lead for being standardization targets.

Ethereum and Hyperledger

Ethereum, Neo, and Hyperledger are all technologies that have already attracted substantial traction from different projects. Any newcomers will have to not only offer a superior technology at the blockchain level, but will also contend with the host of tools and existing libraries and tutorials that are already being developed. In particular, the Ethereum and Hyperledger ecosystems have a tremendous amount of investment dedicated to them, as well as the substantial corporate support. Both projects are the only ones to be offered by Microsoft and Amazon as blockchain-as-a-service offerings.

Service consolidation and product offerings

As the blockchain technology has grown, so have the number of companies offering blockchain services. Just like with individual blockchain projects, we can expect that many of these companies will go away and the more successful ones will become dominant inside of their ecosystems. In some cases, this was the original economic play: IBM wrote much of the Hyperledger fabric and provided it as open source, and now provides an extensive consulting service in the space.

As with Amazon and Microsoft offering blockchain services, there will be an increase in the number of off-the-shelf products for businesses. Currently, custom blockchain development is prohibitively expensive for many businesses, as well as risky. Once standard and dominant players become more clear, it will be easier for startups to provide targeted blockchain-as-a-service and blockchain integration to existing enterprises. It will be at this stage that blockchain adoption genuinely becomes mainstream.

Cross-chain communication

One of the big challenges in the blockchain world is communicating across blockchains. Some projects, such as Cosmos, have this problem at the center of their approach. Efficient and accurate cross-chain communication is a critical capability to enable blockchains to scale, as well as provide the customized services that different ecosystems will demand. While a few approaches are under development, none of them have really been proven out in the real world under typical business conditions. Once a good solution is found, that blockchain ecosystem will have a huge advantage over others.

Intersecting with AI and IoT

The two other major technological buzz-phrases of the moment are AI and IoT. There are a number of ways that these technologies could overlap and intersect.

Blockchain-intersecting AI

Artificial intelligence requires a vast amount of data to be effective. This is part of why so many big companies are pursuing AI projects with such fervor. Larger companies have access to more data, and are therefore able to produce superior AI results than companies that have less data. Thus, larger companies have a vast competitive advantage over small ones if they are able to leverage AI expertise effectively. Public blockchains remove this advantage because the same data is available to all. In this realm, smaller startups that are more nimble and have nothing to lose could leverage this public data to provide new services and offerings through proprietary or highly targeted offerings.

Similarly, it is expected that consortium blockchains will further strengthen incumbents who can share data among one another to keep out new competitors. The existing cartel would have vast data troves larger than any of the individual actors, creating a defensive perimeter around their industry. While the consortium members might still compete among themselves, their data advantage would help shield them from new market entrants.

Blockchain-intersecting IoT

One of the ongoing threats to the internet of things is security. Millions of network devices result in millions of potentially vulnerable items that could be coopted by attackers. Using a blockchain to deliver software updates, along with systems such as trusted computing modules, IoT devices could be at least partially shielded from attacks by requiring an attacker to disrupt the entire chain, not just a specific device. However, this would take careful engineering, and the first few attempts at this form of security will likely fail due to implementation bugs.

Other interactions with blockchain include logistics, where companies tie the movements of their delivery vehicles to a consortium network that broadcasts each package movement across the entire network, improving delivery and identifying issues. For instance, a truck hitting a roadblock and updating its travel across a consortium of logistics firms would allow other trucks to route around, thereby improving deliverability.

Summary

In this chapter, we've discussed the near-term likely future of blockchain technology. It is impossible to know the future, but from what can be gauged, these are the industry trends and forces shaping the technology as it exists today. For those businesses investigating blockchain, each of these three major trends will act to either encourage blockchain adoption or cause avoidance. We hope that the governments of the world issue clear and well thought out regulation as soon as possible, as well as put in place safeguards to prevent fraud. With this in place, this technological evolution will have more guidance in terms of how to bring blockchain products to market safely and with less risk.

Blockchains, in general, will continue to be applied to different industries. It is unknown what new networks such as EOS and Cosmos will bring to the table, as no major chain has yet had the extensive governance models that these systems seek to provide. If large-scale, decentralized governance is able to perform well, it would be an encouraging development for many systems. For instance, the government of Estonia is already doing trials with blockchain for some of their government functions.

Most likely, the current blockchain systems will not be the dominant ones in the end. As with most technologies, the dominant systems will be the ones built on the lessons learned from the current crop of failures.

Index