Introduction to Cryptocurrencies and Blockchain

A High Level Overview

Rustie Lin and Nadir Akhtar

Draft — May 2017

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 0

Introduction

After taking Blockchain at Berkeley's Spring 2017 Introduction to Cryptocurrencies DeCal, I decided to make a set of notes to reinforce my learning and also to compile the core ideas of blockchain onto an easily readable format. Big thanks to the wonderful lecturers, officers, and other folks at Blockchain at Berkeley for getting me hyped. I have structured this set of notes after their lecture slides. This set of notes started off as a personal passion project of mine but has since garnered the interest of Blockchain at Berkeley executive staff, namely DeCal lecturer Nadir Akhtar. This project has been a collaborative effort ever since, and will be provided as complementary material to future DeCal offerings.

There will be bugs, shaky facts, and some hand-wavy stuff. Be warned! In any case, if you do find yourself reading this, I hope you enjoy! – *Rustie Lin*

"how do i get bigger than Bigg but not as big as huge" – *Nadir Akhtar*

Guidelines for use:

These notes are based on the Spring 2017 edition of Blockchain at Berkeley's "Introduction to Cryptocurrencies and Blockchain" student-run class. (For any non-Berkeley students: these courses are referred to as DeCals, short for "Democracy at Cal," exclusively student-run courses.) Any updates to the course, including but not limited to material or structure, are not necessarily be reflected within these notes.

- Each of these notes corresponds to a specific lecture. It may help to read the note before going through the lecture, or go through the lecture before going through the notes, or use one of the two exclusively. We have no clue! You let us know what works best for you, we're curious ourselves!
- All of the crypto/blockchain material comes from Blockchain at Berkeley. Notes will contain about just as much information as lectures, profoundly overlapping in information. Again, these notes are merely a different way to present the information generated by Blockchain at

Berkeley. A few tweaks in wording may have been made based on our judgment, details may have been added or omitted, but all main ideas remain.

- Every note will end with a page on "Key Terms," composed of any bolded terms within the given note. These Key Terms will either be important crypto/blockchain terms or terms perhaps unfamiliar to the average reader. Terms will be organized alphabetically. While reading the Key Terms alone may give you a quick *sense* of the content, they will hardly give you an understanding.
- References to various outside sources are included in a separate file from all notes. Any uncited information (almost everything in these notes) can be assumed to come from Blockchain at Berkeley. We are not validating all information in these notes; responsibility for any improper or false information belongs to the source of info. Don't shoot the messenger. One note, definitions for Key Terms are rarely word-for-word from Blockchain at Berkeley. (In other words, they're mostly paraphrased or even made from scratch based on personal knowledge.)

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 0

Key Terms

Below is a working list of commonly used terms and their definitions in the Blockchain space.

1. **Bitcoin** — A popular cryptocurrency existing purely as software. “Bitcoin” (uppercase) refers to the protocol, software, and community; whereas, “bitcoins” (lowercase) refers to the unit of currency. It is not backed by any asset or regulated by any central entity. (See: *Decentralized, Distributed*)

2. **Blockchain** — An immutable and distributed database that maintains a growing list of records. Blockchains are used to safely and securely record transactions between parties. Originally designed to make Bitcoin possible.

3. **Cryptocurrency** — A digital currency in which encryption techniques are used to regulate the generation of units and currency and verify the transfer of funds, operating independently of a central bank.

4. **Decentralized** — A characteristic that ensures that no single entity has control over all the processing in a system.

5. **Distributed** — A characteristic that ensures that not all the processing in a system is done in the same location.

6. **Trustless** — A characteristic that ensures that a system or protocol can function without trusting a single entity. (e.g. Everyone has a copy of the Bitcoin ledger/blockchain, so users do not have to trust a single entity, organization, or third-party.)

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 1

Bitcoin History: From the Cypherpunk Movement to JPMorgan Chase

Over the years, Bitcoin demonstrated its unique prowess as a currency, **store of value**, and more. As an open financial network secured with **public/private key encryption**, Bitcoin allows for **pseudonymous** financial transactions. Thanks to the Internet, Bitcoin is easily accessible and adoptable, allowing for censorship-resistant payments across borders.

In the modern day, Bitcoin can be used to send money cheaply and efficiently across political and geographic borders. As a "borderless" currency, Bitcoin enables users to send money that retains its value in every jurisdiction, particularly useful in developing economies and as an overseas payment mechanism. Bitcoin also demonstrates immense potential for digital trading, machine to machine payments (especially regarding the **Internet of Things**), autonomous networks, micropayments, and as an alternative store of value. In this note, we will explore the history of Bitcoin and the sequence of events that brought its existence. The goal is to provide insight into the backgrounds and motivations behind the present-day leaders of the Blockchain industry.

Pre Bitcoin-2009: Libertarian Dreams and Ideals

Rapid advancements in computing technology in the 1980s and '90s brought along debates of computing rights and privacy. Rooted in libertarianism and cryptography, a community of dissenters who called themselves the "**Cypherpunks**" began an anti-government movement to counter increasing surveillance enabled by evolving technologies. Eric Hughe's "A Cypherpunk's Manifesto" states: "Privacy is necessary for an open society and in the electronic age."

Cypherpunks speculated about technology's role in shaping interactions between the individual and the state. They believed that individuals deserved access to these new tools, had the right to protect their personal information, and argued against government attempts to deteriorate any individual's privacy.

An early attempt at establishing a currency separated from the state went by the name of **DigiCash**, an early cryptocurrency. Seeing the existing financial system as one of the greatest threats to individual privacy, cryptographer **David Chaum**, inventor of Blind Signature Technology, implemented DigiCash using the latest advancements in public and private key cryptography. Digicash promised complete privacy for users conducting online transactions and included a system of cryptographic protocols, preventing banks and governments from tracing personal online payments. However, a fatal flaw rested at the heart of Digicash's design: centralization. Chaum's company bore the overwhelming burden of validating every digital signature in the system, leading to bankruptcy in 1998. For the community, DigiCash represented not so much a failure as an important learning opportunity. *Moral of the story: For a cryptocurrency to be stable, it must be decentralized.*

2009-2010: Early Development of Bitcoin

In October 2008, a **whitepaper** titled *Bitcoin: A Peer-to-Peer Electronic Cash System* appeared online, published by an unknown person (or persons) by the pseudonym **Satoshi Nakamoto**. His nine-page white paper seemed to combine all previous efforts to create a self-sustaining digital currency. Although some were disheartened by historical attempts (such as DigiCash) to implement a cryptocurrency, a few early pioneers quickly recognized Bitcoin's potential and began supporting the project.

Within a year, Satoshi and other developers released the first Bitcoin implementation. The **Genesis block** was mined in January 3, 2009. The **coinbase** of this block references a story in the Times of London newspaper involving the Chancellor bailing out banks. The first bitcoin transaction occurred on January 12, 2009. A year later on May 21, 2010, an individual first accepted Bitcoin in exchance for a tangible asset. Laszlo Hanyecz purchased $25 worth of pizza for 10,000 BTC. Little did he know that 10,000 BTC would be worth about $20,169,600 (as of May 20, 2017).

2010-2012: Scandals, Hacks, Illegal Activity

In the early stages of Bitcoin, there existed no exchange for trading between bitcoin and regular currencies. In 2010, the creator of online service **mtgox.com** (*Magic: The Gathering Online* eXchange, where *Magic: The Gathering Online* players traded cards like stocks) decided to convert the domain into a bitcoin exchange website. Mt. Gox consolidated itself as the biggest bitcoin exchange during the beginning stages of Bitcoin. Since then, Mt. Gox suffered various security breaches and thefts. On June 19, 2011, a hacker allegedly used a Mt. Gox auditor's computer to transfer an abundance of bitcoins to himself. Mt. Gox was subsequently shut down for seven days to investigate the situation and received several lawsuits, some of which remain unresolved.

"**Silk Road**," another Bitcoin-centric site, also developed during this time. An illicit marketplace for drug deals nicknamed "The eBay for drugs," Silk Road generated a significant impact in the Bitcoin ecosystem, constituting a huge portion of daily transactions, as most illegal merchants only accepted bitcoins due to Bitcoin's anonymous and untraceable features. After launching in February 2011, Silk Road was shut down in October 2013 by the FBI, who seized 3.6 milllion USD worth of bitcoin. Ross Ulbricht, the founder of Silk Road, was issued a life sentence without possibility of parole. The highly publicized story of Silk Road's take-down also changed general sentiment towards Bitcoin.

Association of Bitcoin with illegal online activity, scandals, and hacks swayed public opinion against the cryptocurrency. Despite negative views, speculation about Bitcoin's future popularity also drove the price of one bitcoin to over one thousand USD. For better or for worse, the flurry surrounding Bitcoin drew an unprecedented amount of attention.

2013-2014: Bitcoin Attracts Attention

Just four months after the FBI revealed and shut down revealing the illegal bitcoin marketplace Silk Road, Mt. Gox declared for bankruptcy in February 2014. Mt. Gox reported losing 744,408

BTC in a theft that went unnoticed for years.

In March of 2014, CoinDesk, a popular online Bitcoin news platform, published an article claiming that Bitcoin inventor Satoshi Nakamoto had been “found” in California. The man in question denied his association with the name.

As Bitcoin entered the mainstream market, renowned venture capitalist Tim Draper announced his purchase of nearly 30,000 BTC. In an interview, Draper was optimistic that the price of bitcoin would rise to 10,000 USD; as he sees it, “an entire economy is being rebuilt.” In an interview, Draper described bitcoin as **bullish** in emerging markets, and encouraged investors to buy bitcoin.

2014 also marked the year when major retailers and corporations began accepting bitcoin. In January, Overstock.com and Porn.com became the first companies to accept bitcoin. In April, new marijuana vending machines in Colorado intended on accepting bitcoin. In September, PayPal partnered with Coinbase and Bitpay, two large exchanges. In December, Microsoft also began accepting bitcoin payments.

2013-2014: Venture Funded Bitcoin Startups

Hype for Bitcoin fueled a rise in venture funded Bitcoin startups. One of the most notable was Coinbase, a digital asset exchange company based in San Francisco. Coinbase served mainly as a centralized, online bitcoin wallet, also serving as an important tool for investors to gauge interest in Bitcoin. Founded in June 2012, Coinbase witnessed drastic increases in funding in response to Bitcoin hype. In May 2013, Coinbase received a 5 million USD **Series A**; December 2013, 25 million USD **Series B**; July 2015, 75 million USD **Series C**.

Other early venture funded Bitcoin startups included BitFinex, 21 Inc, BitPay, and Blockstream. Around this time, Andreessen Horowitz (a16z) also got involved in the Bitcoin space. Andreessen Horowitz has been directly involved in developing the Bitcoin industry, as they began investing in Bitcoin Core (the group that maintains the core Bitcoin software), sidechains, and various research firms in the space.

Hype for Bitcoin seemed to die down in 2014, demonstrated by a plummet in bitcoin price. Clearly, Bitcoin was not in its best state.

2015-Present: Ethereum Blows Up, Bitcoin Bounces Back, “Enterprise Blockchain”

Between 2015 and the present (May 2017), a great number of initiatives developed, ranging from other cryptocurrencies to non-crypto blockchain use cases and startups. *(Disclaimer: Not an exhaustive list.)*

A whitepaper released in late 2013 by Russian programmer and cofounder of Bitcoin Magazine described the premise of a Turing-complete protocol running on a blockchain. (Bitcoin has a scripting language with basic functionality but nothing powerful enough to handle complex logic.) In July 2015, this idea came to fruition in form of the Ethereum blockchain, which serves as a platform for people to build and deploy decentralized applications. Applications on Ethereum require "fuel" in the form of Ether tokens to run (more on this in a later note.) Less than a year later, Ether tokens were valued at a total of one billion USD. Ethereum represented a new future for decentralized technologies, and inspired such ideas as new governance models.

Political events in 2016 and early 2017 also had a profound impact on the growing popularity of Bitcoin and blockchain technologies. Most notably, Brexit, India's war on cash, and the inauguration of President Donald Trump in the United States were political events that caused a number of individuals to lose trust and turn away from their governments in favor of decentralized and trustless alternative: Bitcoin. Additionally, economic events in the media highlighting the circumvention of capital controls, as well as general instability in the market caused the price of bitcoin to increase dramatically.

There has also a growing interest in blockchain technologies from banks. Although big banks previously disregarded Bitcoin due to a lack of understanding and resentment towards decentralized control, several banks recently demonstrated interest in integrating blockchain technology into existing services. As Chairman, President, and CEO of JPMorgan Chase, Jamie Dimon stated in February 2016: "The Blockchain is a technology, which we've been studying . . . and yes it's real. It could probably reduce the cost of real application in certain things . . . If it proves to be cheap and secure it will be adopted for a whole bunch of stuff." The separation of Bitcoin and blockchain in the discourse of even big banks reflects the importance of these technologies in recent past.

From the underground Cypherpunk movement to one of the largest and most powerful centralized banking institutions in the United States, Bitcoin leaves permanent marks on industries and continues to rise into mainstream popularity. The culmination of the cutting edge in computer science, cryptography, and the passionate support by a dedicated community has elevated Bitcoin and blockchain technologies to a spotlight that is on its way to profoundly reinvent the future.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 1

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **Bullish** — In reference to stocks, having the expectation to rise in value.

2. **Coinbase** — (Not the exchange!) The "coinbase" (conventionally lowercase, not a proper noun) is a part of the Bitcoin blockchain where a "miner" (not yet formally introduced term) sends a block reward to themselves as a pat of the back for securing the network. In the context of this note: the coinbase has a section in which a miner can leave a small amount of arbitrary information which the rest of the network will save but ignore, similar to comments in code. Most people choose to leave this field blank, as they have no need for it. Satoshi decided to use it to store a link that is now included in the first block of every (valid) version of the Bitcoin blockchain.

3. **Cypherpunks** — A group of 1980s and '90s anti-establishment, libertarian activists whose loyalty lies primarily in code and cryptography. Huge inspiration for privacy-centric cryptocurrencies.

4. **David Chaum** — Considered the "Father of Anonymity" by some, he received a doctorate in computer science (at Berkeley!), going on to develop DigiCash in an attempt to create an electronic, anonymous currency separate from the state.

5. **DigiCash** — A landmark attempt at creating an electronic, anonymous currency. Failed due to lack of scalability, as one entity alone could not monitor and verify the volume of digital signatures.

6. **Genesis block** — The first block of a blockchain. A term originally coined for Bitcoin, the first block mined of any blockchain is now referred to as the "genesis block" for that particular blockchain.

7. **Internet of Things (IoT)** — Connecting various electronic devices that were not originally intended for interoperability.

8. **Mt. Gox (mtgox.com)** — A classic Japanese *Magic: The Gathering* card trading site turned bitcoin exchange story, this exchange incepted in 2011 became infamous for lost funds, hacks, poor management, and dozens of lawsuits. While skepticism about Bitcoin increased, the name spread as well. (A similar gist to Jack Sparrow's retort to being the worst pirate ever heard of: "But you have *heard* of me." – *Nadir*)

9. **Pseudonymous** — While accounts are allegedly anonymous, experts can trace these random online accounts to real people with enough information, meaning that the accounts can be deanonymized with thorough investigation. Difficult, but not impossible.

10. **Public/private key encryption** — In the context of cryptocurrencies, a method by which users certify ownership of a particular account or quantity of coins. Every user has at least one public/private key pair. The public key is given out to others for receiving money, the private key is used to "unlock" those funds.

11. **Satoshi Nakamoto** — The pseudonym for the creator of the Bitcoin whitepaper. Many suspects put forward, but no conclusion on the true identity. (I think it's some British person, from Satoshi using "bloody" in GitHub commits to putting punctuation outside of quotation marks – *Nadir*)

12. **Series A/B/C** — Rounds of funding for startups. Each one demonstrates potential and viability for the company's success, and at each state certain indicators have various weights.

13. **Silk Road** — An illegal drug marketplace run by Ross Ulbricht, who called himself "Dread Pirate Roberts," transactions were primarily conducted in Bitcoin. Started in 2011, shut down by FBI in 2013. While previous Silk Road employees attempted to start Silk Road 2.0, FBI once again shut down the site.

14. **Store of value** — An asset presumed to maintain its value or appreciate (increase in value) over time. Gold, a classic example.

15. **Whitepaper** — A formal academic proposal, defended with research and reasoning.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 2

Bitcoin Protocol and Consensus: A High Level Overview

Beyond the excitement of Bitcoin, cryptocurrencies, and blockchain hides a confusion about what *exactly* Bitcoin is and how it works at a technical level. This note is dedicated to explaining how Bitcoin works at a high level to build a mental model to prepare us for a more specific breakdown of Bitcoin's mechanics.

First, we will build Bitcoin from the ground up, explaining each of the system's key features and their necessity for Bitcoin to be distributed, decentralized, and trustless nature. Then, we will move on to discuss finer details in Bitcoin's implementation, such as a brief overview of mining and the anatomy of key data structures.

Bitcoin from the Ground Up

To fully understand how Bitcoin works, we start by attempting to build Bitcoin from scratch. Each time a major flaw is detected, a new feature must be built in to resolve the problem. By the end of this section, we will have built Bitcoin from the ground up. Keep in mind that our goal is to create a currency that requires no central entity. Let us start off by considering two users of this currency, Alice and Bob.

First step: Establishing a system that allows users to write and sign messages describing transactions, the primary value of currency. Each transaction would be broadcast to the world (more specifically, every other user of Bitcoin). Reasoning: If there exists no central third party to verify history, then every other member of the community becomes the third party. For Alice to send Bob one bitcoin, she would announce to the world, "I, Alice, am giving Bob one bitcoin," and the world would take note. A problem arises if Alice, intentionally or not, sends multiple identical messages to Bob. If she says five times in a row that she wants to give one bitcoin to Bob, did she accidentally repeat herself? Did she mean to send five bitcoins, or only one? The ability to send the same money multiple times would even give Alice the opportunity to spend more money than she has.

Second step: To prevent ambiguity, we introduce uniquely identifiable serial numbers for each transaction conducted on the system. If Alice were to send one bitcoin to Bob now, she would announce "I, Alice, am giving Bob one bitcoin with serial number ####." Modern banks use similar serial numbers to keep track of their transactions. If our version of Bitcoin were to implement serial numbers, it must need a centralized bank to manage serial numbers, along with transactions and balances. However, this defeats the purpose of a decentralized currency and puts us right back where we started.

Third step: Make everyone the bank. Have every user of Bitcoin store a complete record and serve as a witness and third party to all transactions. We will refer to this shared ledger as the "blockchain." (We will elaborate on blockchains later.) In this new scheme, Alice may send Bob a transaction, and Bob will announce the transaction to the world. Other users update their records accordingly, and everyone keeps track of the amount of bitcoins each individual owns. What if, however, Alice double spends on Bob and Charlie? In other words, if Alice sends the same bitcoin to Bob and Charlie, how does the system know which transaction is valid, if any? Simply rejecting the later transaction is unreliable, as in the real world of dropped packets (lost information) and unreliable connections, network speed varies too much. Some users may even see one transaction before the other.

Fourth step: To address this problem, we ensure that the network confirms the validity of the transaction before allowing a recipient of funds to assume a successful transfer. edit the transaction sequence as follows: Alice sends a transaction to Bob, who then announces the transaction to the world. Everyone then verifies the transaction for Bob's protection. If Alice were to send the same funds to Charlie, those verifying would inform Bob and Charlie of Alice's deceit. This prevents Alice temporarily, but the possibility remains for Alice to double spend on Bob and Charlie given persistence. If she goes so far as to fabricate several fake online identities, she can then masquerade as several unbiased third parties while selfishly and maliciously verifying her own transaction, as both Bob and Charlie are unaware that all these other people are actually Alice and acting in Alice's best interests rather than the interests of the whole group. This attack pattern, of creating many fake identities to subvert a system, is well-known as a **Sybil attack**.

Fifth step: To solve the Sybil attack vulnerability of double spending, the system can make it expensive to verify transactions. By requiring users to submit proof of having spent some resource to confirm a pending transaction, they will think twice before spending time, money, and energy on this verification process. Here, we shall implement a **Proof-of-Work** protocol. In a Proof-of-Work consensus system, when Alice wishes to make a transaction, she announces this to the entire network, as before. To correctly verify a pending transaction, an user must do three things. (Let us call this user "David.") First, David must check his records to validate the transaction (i.e. to check if the bitcoins involved have not been spent before, if the person sending bitcoins is the owner/allowed to spend those bitcoins). Second, to prove his commitment to honesty, he must do some work—in Bitcoin's case, compute the solution to a difficult, arbitrarily and systematically chosen mathematical puzzle. Third, he must announce the validity of the transaction back to the network, along with the solution to the puzzle. Other users in the system would then verify the solution of the puzzle and the transaction before appending the transaction to their records. *(Brief comment about this series of notes: "Proof-of-Work," the capital version, refers to the consensus algorithm, the broader scheme characterized by this specific mining challenge; whereas, "proof-of-work," the lowercase version, refers to the solution to the mining puzzle.)*

The most important step in this process is the second, in which David must solve the math puzzle. By requiring users to solve this math puzzle, which takes a great deal of computational power, we deter users from spawning endless identities and double spending, as they are now solely limited by the amount of computational power they have and are willing to spend. To further analyze how this solves double spending, consider the case after this fifth step when Alice attempts to double spend with multiple identities. We easily see the difficulty of identifying dishonest and malicious actors before they display malicious behavior. By requiring a mathematical puzzle to be solved, the system makes it costly for any user in the network to verify transactions. A good way to understand Proof-of-Work is to view it as a competition. Users on the network are constantly in competition to verify their blocks of transactions. Malicious actors such as Alice are in constant competition with other users to verify transactions. As long as the majority of computational power is owned by honest users, then bad actors such as Alice will have difficulty double spending.

Conclusion: In building the system from the ground up, we have highlighted the major features of Bitcoin. In the beginning, signed messages that are announced to the entire network formed the basis of the entire system. After adding serial numbers, the system was able to uniquely identify transactions. By making everyone in the network the "bank," we implemented a blockchain, allowing for a shared record of transactions. Having everyone in the network verify transactions provided a much-needed increase in security. Finally, Proof-of-Work deters malicious actors in the network from double-spending, actualizing a secure and trustless system.

Basic Concepts – Identity in Bitcoin

As described earlier, one of the most fundamental concepts in Bitcoin is sending money between **pseudonymns**. In Bitcoin, a pseudonym refers to a user's Bitcoin **address**, otherwise known as the hash of their public key.

Bitcoin also buids on some cryptographic primitives. Most notably, Bitcoin uses **ECDSA (Elliptic Curve Digital Signature Algorithm)** to produce cryptographically secure public and private key pair functionality similar to username and password schemes. Additionally, Bitcoin uses a **cryptographic hash function** called **SHA-256** to implement its Proof-of-Work scheme. Hash functions take in an input and **deterministically** create an output of fixed size. (While SHA-256 technically can't take in input of *any* size, the limitations are beyond the scope of data used in Bitcoin: one must exceed $2^{64} - 1$ bits to exceed the valid input size, making the limit almost entirely theoretical.) Cryptographic hash functions, specialized hash functions designed for security, are one-way: for a given output, it is very difficult to find its corresponding input, and a small change in the input data completely changes the output. These properties of SHA-256 are effective in maintaining security and consistency across the Bitcoin network, including the Proof-of-Work scheme and distribution of Bitcoin addresses.

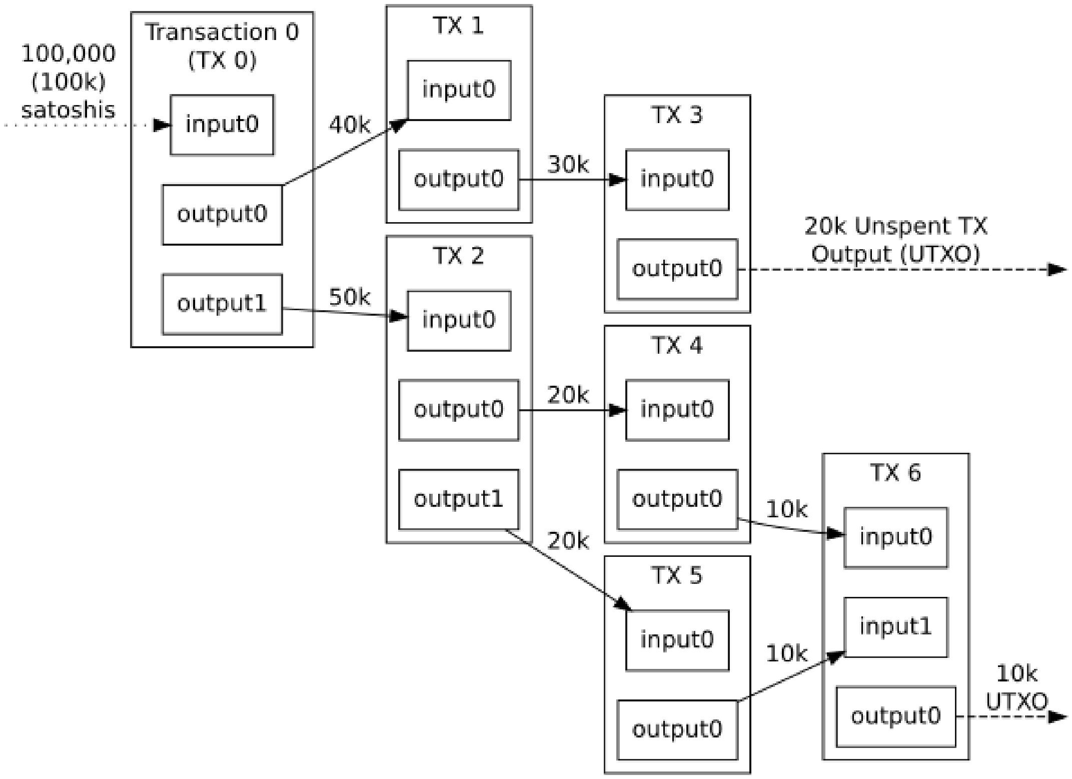
Another key concept in Bitcoin is that bitcoin is untouchable by anyone aside from the rightful owner, for two reasons: 1) Discovering the correct private key for a public key requires a quantum computer to expect a solution within your lifetime, and 2) Two randomly chosen private keys are *incredibly* unlikely (still an understatement) to correspond to the same public key. There are 2^{160} possible addresses. To put this massive number into perspective, consider the following. There are roughly 2^{63} grains of sand on the Earth. Doubling the exponent, 2^{126} is actually only 0.0000000058% of 2^{160}. We only reach 2^{160} if we imagine that for each of the 2^{63} grains of sand on Earth, there is another Earth, each with its own set of grains of sand. The number of bitcoin addresses is so large that the possibility of colliding with someone else's public key whenever a user generates a public/private key pair is negligible. (For those curious, 2^{160} is exactly 1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976 – *Rustie*)

Transactions – A Basic Version

Transactions are usually conducted through wallet software, although it is possible to do this manually. Wallet software automatically generates Bitcoin addresses for its users. Users can then receive money by sharing their addresses, or send money specifying an address and amount. In many online wallets, transactions can even be conducted easily via a series of QR codes (e.g. in the Coinbase interface.) Whenever a transaction is made, it is broadcast to the entire Bitcoin network, where the network verifies the transaction and adds it to the transaction history.

Introduction to Transactions

When a user spends bitcoins, they are always spending bitcoins received in previous transactions. Transactions are mappings from inputs to outputs, where inputs and outputs each contain an address and an amount. For example, let's say Bob spends 0.05 BTC at a coffee shop. If the transaction input was 1.00 BTC, then the outputs would be 0.05 BTC to the coffee shop, and 0.95 BTC back to Bob. The reason for two outputs is because as per the Bitcoin system specification, inputs can only be spent once, creating the need for a change output (the 0.95 BTC) back to the sender. The Bitcoin network also imposes transaction fees: users are expected to construct transactions such that the input amount exceeds the output amount, and their difference equals the transaction fees collected by the network. In the previous example, if Bob sends 0.05 BTC to the coffee shop and only 0.94 BTC back to himself, then the difference in inputs and outputs (1.00 BTC - 0.05 BTC - 0.94 BTC = 0.01 BTC) equals the transaction fee. Keep in mind that transactions with lower transaction fees offered up to the network are less likely to get confirmed—if ever! (You will soon see that "the network" is an abstraction referring to miners, which we will talk about later in this note.)



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

Blocks and Blockchain

A **block** serves the purpose of containing an ordered bunch of transactions. Key characteristics of a block include time-stamps and (with the exception of the genesis block) references to the most recent previous block. Changing a transaction, the block's time-stamp, or the block's reference invalidates a block. This property ensures that blocks are immutable. A **blockchain**, as the word implies, is a series of blocks that are "chained" together. Later parts of this note will discuss further a block's components.

Basic Concepts – UTXO Analogy

In Bitcoin, users do not have accounts from which they add or subtract value to determine the amout of bitcoin they own. Instead, users spend from unspent transaction outputs, or **UTXOs**. UTXOs can be thought of as the global set of unspent bitcoins. Instead of "spending A bitcoin," it is more useful to think of a user as "spending THIS bitcoin," as bitcoins are uniquely identified by their transaction outputs. (Think back to our reasoning for serialized currency in "Bitcoin from the Ground Up.")

Bitcoin is analogous to the Rai Stones of the Yap Islands. Rai Stones are large circular stone disks carved out of limestone by the Yapese, a Micronesian people, and used as currency. These Rai Stones were upwards of 3.6 meters in diameter and 4 metric tons heavy, too much for locals to move around regularly. Instead, to conduct transactions and exchange them for goods, the Yapese agreed on a system that revolved on trust and changing ownership of the Rai Stones. In a transaction, both parties would announce to the rest of the island that the transfer of ownership over stones was taking place. This is how Bitcoin works, as ownership of specific bitcoins are transferred between users during transactions. (There is a slight difference because bitcoins can be divided and renamed, but the concept remains true.)

Mining Sketch

Before exploring mining, we must first understand Proof-of-Work. Proof-of-Work solves the double spending problem and is a **Byzantine consensus algorithm**, an algorithm which tolerates failures involving both faulty and malicious nodes. The former will behave in aimlessly incorrect ways, such as by sending repeat or corrupt messages. The latter intends to subvert the system, either by propagating false information or by preventing the system from coming to consensus. Proof-of-Work is but one of many consensus algorithms, however, so keep in mind that it is not the be-all and end-all to decentralized consensus. Private and hybrid/consortium blockchains use alternative consensus algorithms, but because they are not public (and, by implication, not anonymized), the system depends on trusting a good majority of approved entities, opening up opportunities to use multitudes of various other consensus algorithms given certain assumptions of trust.

In Bitcoin, **mining** serves various essential functions that uphold the entire Bitcion system. Primarily, it serves as a minting mechanism, ensuring fair distributions of coins via the Proof-of-Work challenge. (Some could argue against this, considering the presence of mining pools. More on this in a later note.) Additionally, the treasured block reward incentivizes **miners** to support the greater goal of securing the network. Countless miners maintaining network health and protecting distributed consensus ultimately determine Bitcoin's security and viability.

Mining Sketch – Finding Blocks

Every miner includes a special transaction called a **coinbase transaction** in a block's list of transactions. This transaction allows miners to receive a mining reward $\big($currently at 12.5 BTC (6/20/17)$\big)$ if they find a valid proof-of-work before every other miner. Miners who have found a valid proof-of-work have, in Bitcoin vernacular, "found the block." The miner saves the block to his own blockchain, then broadcasts the block to the rest of the Bitcoin network. Other miners in the network would verify the block and then add it to their own copy of the blockchain.

On average, a block is found every 10 minutes. Not by chance, but by design: because the average level of computational power (which we shall also refer to as "**hash rate**") within the network constantly varies, the difficulty of the problem which miners solve (the Proof-of-Work puzzle) must adjust accordingly. Consider a scenario in which the difficulty remained constant: if the hash rate rises significantly (the expected long-term trend), then the puzzle becomes too easy to solve, and the proof-of-work no longer requires any work to find; and if the hash rate decreases significantly, then the network gets backed up because valid blocks are too difficult and time-consuming to generate. Hence, the puzzle's difficulty must adapt to remain relatively as difficult as before with the network's hash rate fluctuations. (We shall discuss further in Note 5, "Mining.")

The block reward halves every 4 years; at the time of writing, the nearest instance of the block reward halving will be on July 9th, 2017. A halving block reward implies that bitcoins are in limited supply, i.e. deflationary, as the block reward will approach 0 over time and no more bitcoins will be minted. Around Year 2140, the maximum of 21 million bitcoins will be in circulation. There are approximately 15.2 million bitcoins in circulation today (6/19/17).

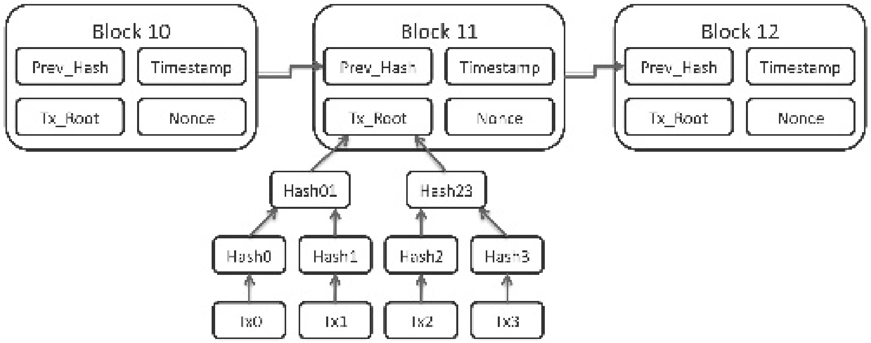
Briefly, the process of mining can be described as follows: A miner attempts to generate a "valid" block header. Before we discuss the definition of "valid," let us look at what defines a potential block header:

$$\textit{Output} = \text{SHA-256}\big(\textit{Merkle Root} + \text{SHA-256}(\textit{Previous Block}) + \textit{Nonce}\big) = \textit{Block Header}$$

The output of the hash of three components, 1) the **Merkle root**, 2) a hash of a previous block header, and 3) a **nonce**, generates a block header. A Merkle root preserves a summary of transactions by using some useful cryptographic properties which we will discuss later. A nonce is an arbitrary number, the piece of the math puzzle miners search for, the coveted proof-of-work, and the only part of this equation that changes. A miner uses SHA-256 to hash together the Merkle root, a nonce, and a hash of the previous block.

A "valid" block header must contain a prerequisite number of leading zeroes agreed upon by the network per the "difficulty." The higher the difficulty, the more leading zeroes required for a valid block header, and vice versa. Validity is defined as leading zeroes because a) any arbitrary pattern could have been selected (leading number of sixes, sequential order of digits), but leading zeroes happened to be selected, and b) the required amount of leading zeroes is easy to adjust, fulfilling our desire from before for an adjustable difficulty. The solution (proof-of-work) that miners vie for is defined as an output that generates a valid block header. The difficulty of the puzzle is constantly adjusted every 2016 blocks (every two weeks at 10 minutes/block).

Blockchain Diagram



By Matthäus Wander (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

http://commons.wikimedia.org/wiki/File%3ABitcoin_Block_Data.svg

Transaction Flow

To more clearly illustrate the process of making a transaction, consider the following example:

Imagine Alice wants to send money to Bob. Alice would first sign a transaction and broadcast this to the network. Miners would then receive and verify the transaction by checking for a) a valid signature and b) sufficient funds. When a miner finds the proof-of-work, thereby solving the block, they would then broadcast the block to the network. As the block propagates, other miners would verify the block themselves. At any given point in time, the longest blockchain is assumed to be the most valid, and since miners are constantly competing for the block reward, they immediately start work on the next block's problem.

Keep in mind that almost all blocks contain hundreds of transactions, so as to increase a miner's transaction fees per block, and that miners prioritize transactions offering the highest fees.

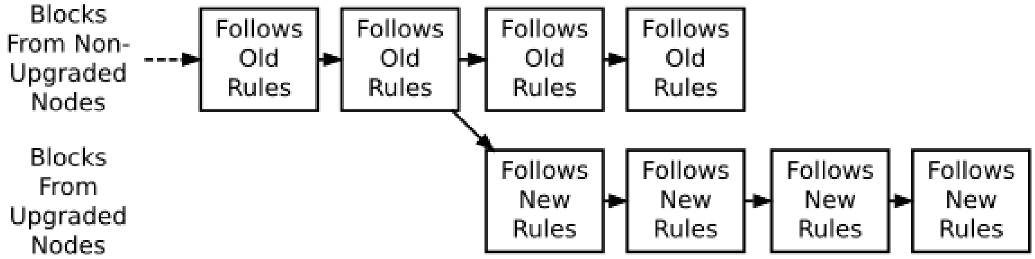
51% Attacks

Bitcoin assumes that a majority of the network is honest. By implication, an honest majority will form the longest Proof-of-Work chain because the majority of the network's hash rate (owned by the honest) would create blocks faster than malicious entities would. Since miners have no incentive to work on a short, malicious chain that will not be recognized as truth in the future, thus offering no block reward or longevity of transactions, miners will work on the longest chain. However, this assumption puts Bitcoin at risk of a **51% attack**, an attempt to overwhelm the honest mining power of a network. While it is possible, amassing 51% of the network's hash power is highly unlikely. (We will also discuss this further in "Game Theory.")

Forking and Consensus Updates

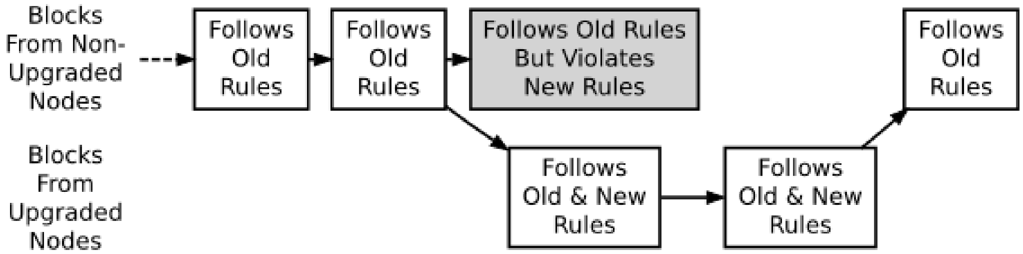
With time, Bitcoin's protocol and software must update to remain efficient and secure. However, nodes may choose to opt out of updating their protocol, causing a split known as a **fork**. There are two categories of forks, each with drastically different implications.

In a **hard fork**, the new version of the consensus protocols is incompatible with the previous because of loosened requirements upon blocks. In other words, hard forks are not forwards compatible. A hard fork potentially causes a permanent divergence in the blockchain. Nodes running an old version of the Bitcoin software would see the new transactions as invalid. An issue can arise when upgraded nodes mine on the new forked blockchain, and non-upgraded nodes continue mining on the old chain. With enough community support, both chains could survive, and we can see this in Ethereum and Ethereum Classic.



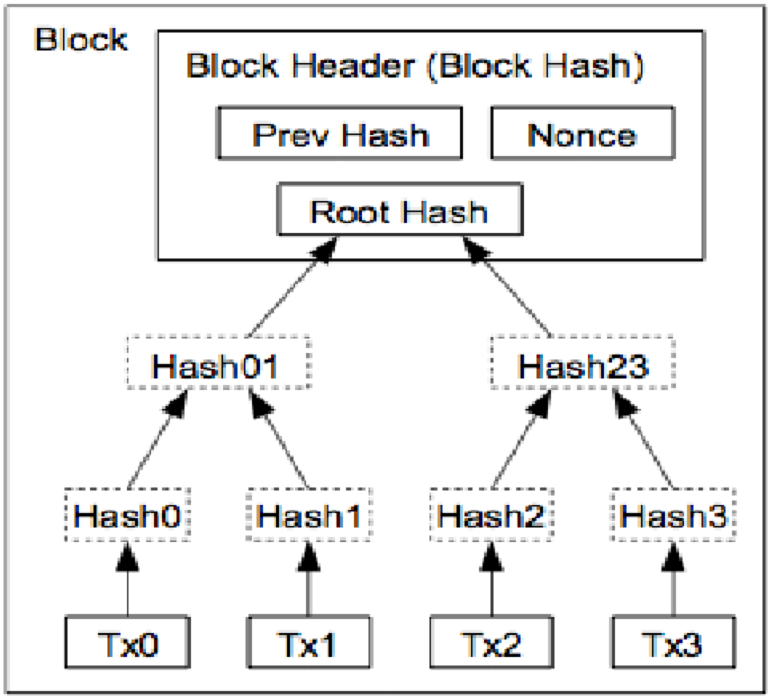
A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain

A **soft fork** tightens the protocol, meaning that old nodes will always accept blocks from new nodes, and the new chain remains valid under old consensus rules. Changes brought about by soft forks are forwards compatible but not backwards compatible, as only a subset of previously types of blocks are acceptable. Although non-upgraded nodes will still be able to see new transactions as valid, they will not be able to mine, as upgraded nodes will reject their blocks. If a majority of the network adopts the new protocol of the soft fork, the network ultimately upgrades to the newer consensus rules.



A Soft Fork: Blocks Violating New Rules Are Made Stale By The Upgraded Mining Majority

Merkle Trees



As seen in the diagram, a **Merkle tree** is a fundamental data structure, "gluing" transactions together and immensely facilitating error detection. It is a **binary tree** in which every non-leaf node equals the hash of the values of its child nodes. To construct a Merkle tree, a miner begins by lining up all transactions, hashing them all once, then hashing them up in pairs up to the final hash. In this diagram, "Tx#" generates "Hash#" in the first hashing step. On the right, hashing

"Hash2" and "Hash3" together yields "Hash23," and same on the left. When all transactions are hashed together, the resulting structure is the **Merkle root**, which is then stored in the block. The rest of the Merkle tree is then discarded to save space in the block.

Let's finally understand part of the blockchain's immutability: imagine that a malicious user tampered with a transaction. (Remember, even a minuscule change in a hash function's input drastically changes the output.) With the change of that transaction follows a change in that transaction's hash value within the Merkle tree, bubbling up to the Merkle root. Due to the properties of SHA-256 and the number of times hashing is required in the process of constructing a Merkle tree, the block containing the corrupted transaction has a drastically different block header than before. The hash of the nonce, Merkle root hash, and previous block hash would then not remain valid per our prior definition, invalidating the block and informing the network of attempted fraud.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 2

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **Address** — In the context of Bitcoin (and other pseudonymous/public blockchains), an arbitrary number representing a user's identity. Others can send money to this address, and a user can send money from this address.
2. **Binary tree** — In computer science, a binary tree is a data structure where a central node, otherwise known as a "root," has two children nodes. These children nodes may also have their own children nodes and represent a subbinary tree.
3. **Block** — Fundamental unit of a blockchain. In Bitcoin, it contains a block header and a list of transactions.
4. **Blockchain** — *From Note 0:* An immutable and distributed database that maintains a growing list of records. Blockchains are used to safely and securely record transactions between parties. Originally designed to make Bitcoin possible.
5. **Byzantine consensus algorithm** — A consensus algorithm is a way for a system of distributed machines to agree on a number of states, or some "truth." A Byzantine consensus algorithm assumes that nodes may be malicious or faulty and still brings the network to consensus if more than two-thirds of the nodes are functioning correctly.
6. **Coinbase transaction** — How the miner earns a block reward. In the coinbase transaction, the miner sends themselves bitcoins equal to the current block reward amount. If this block is solved, propagated, and successfully included in the longest chain, the miner will get to keep those rewards.
7. **Cryptographic hash function** — A hash function takes a certain input and produces a specific output. Every input to a hash function will always produce the same output (deterministic).
8. **Deterministic** — In the context of functions, a deterministic function will always give the same output for a given input. Non-deterministic functions may give different outputs for the same input. One example is the classic 'time()' function. The input is always the same—nothing—but the output will differ depending on the time, naturally.
9. **ECDSA** — Short for Elliptic Curve Digital Signature Algorithm. In Bitcoin, ECDSA is used to generate public and private key pairs, which in turn allows users to sign transactions such that third parties can verify the authenticity of the signature while retaining exclusive ability to create the signature.

10. **Fork** — An update to previous software. Not all forks require disagreement in nodes, but any nodes that choose not to upgrade may run into conflicts with the rest of the network.

11. **Hard fork** — A fork in which the previous software is not forwards compatible with the update. (Backwards compatibility is possible but unlikely.)

12. **Hash rate** — The amount of hashes performed by the network or a specific entity in a given period of time.

13. **Merkle root** — The topmost and final hash value of a Merkle tree, stored as part of the block header.

14. **Merkle tree** — A specific binary tree composed of original data and their hashes. Each datum is hashed once in the first round, then those hashes are concatenated with one another in pairs and rehashed, halving the number of pieces of information in the next round. This process continues until there is only one hash value left, the Merkle root.

15. **Proof-of-work** — A piece of data which is costly to produce but easy for others to verify and which satisfies certain requirements.

16. **Nonce** — An arbitrary value used to produce a valid block header. Nonces are tested in the "mining" process, and a valid nonce is equivalent to the proof-of-work. It takes a massive amount of computing power to find a valid nonce.

17. **Pseudonym** — A false name. (In Bitcoin context, see: *Address*)

18. **SHA-256** — A cryptographic hash function used in various parts of the Bitcoin network such as the proof-of-work challenge.

19. **Soft fork** — A fork in which the previous software is forwards compatible with the update, but not backwards compatible.

20. **Sybil attack** — A strategy by which a single entity generates additional identities, often a great many, in order to gain substantial undue influence in a peer-to-peer network. The rest of the network is not aware that all these identities belong to one person in a successful attack.

21. **51% attack** — An attack in which an entity or group holds a majority of the mining power. With this majority, they can always produce the longest chain and outdo the rest of the network. Any transactions, valid or not, will be confirmed and stored in the longest chain, hence considered validated regardless.

Wallets and Cryptography

The average Bitcoin user often first downloads a **wallet** to manage funds. Analogous to storing dollars in a physical wallet for convenience, wallet software holds Bitcoin or any other cryptocurrency for users, depending on the wallet. This note is dedicated to various uses of Bitcoin wallets and how wallet cryptography keeps everyone safe.

Bitcoin vs Gmail Address Creation

To review public and private keys in Bitcoin, let's compare them with Gmail accounts. When generating a Bitcoin public/private key pair, the user comes up with a private key which generates a corresponding public key via ECDSA as mentioned in the previous note. The set of possible private keys is so large that it is cryptographically safe to just generate a key. (In other words, the chances of anyone guessing or using the same private key as you are very low, so don't fear.) In contrast, the average user registering for an email account with Gmail would choose an address (username) and a password. (Username and password are independent of each other in this case and are not mathematically connected as with Bitcoin.) The main difference in the generation of public and private keys in both examples is centralization. In Bitcoin, users trust the underlying security of cryptography and the negligible probability of colliding public keys. Meanwhile, Gmail users trust Google with private information and regulation of public email addresses.

Base 58

Bitcoin adopts the **base 58** convention to improve the readability of addresses. Most systems use 26 uppercase letters, 26 lowercase letters, and 10 digits, totaling to 62 unique characters. To avoid confusion between visually similar characters, base 58 drops the following characters: uppercase 'I,' uppercase 'O,' lowercase 'l,' and the digit '0,' reducing our options to 24 uppercase letters, 25 lowercase letters, and 9 digits, a total of—you guessed it—58 characters. Although visually similar characters do not fool wallet software, a few points of human failure remain, enough to warrant the use of base 58, such as reading and writing or typing out one's public key for any manual task.

Bitcoin Wallets

A Bitcoin wallet's fundamental use: storing a user's private key. While some wallets allow users to transfer bitcoin as well as store a list of UTXOs, some offline wallets do not provide this functionality but are still defined as wallets.

Bitcoin wallets come in two main flavors—hot and cold—based on their relationship with the Internet. **Hot wallets** are online and connected to the Internet, as demonstrated by apps and online web-wallets. Smartphone and PC app wallets, such as Mycelium and AirBitz, store the user's private key locally. On the other hand, online services such as Coinbase and Blockchain.info store the user's private key in the cloud, giving users access their wallet from any associated device. Hot wallets excel at enhancing user experience at the risk of many security threats due to a continuous Internet connection. What happens if a computer virus infects your computer and steals the private key from your PC wallet app, or if a trusted third party managing your private information gets compromised?

Cold wallets, on the other hand, only exist in the physical world and are never connected to the Internet, sacrificing ease of use for security. Paper wallets, hardware wallets, and brain wallets all fall under this category. Bitaddress.org is a popular website that uses the user's pseudorandom mouse movements to generate a private key, a public key, and a pair of QR codes. By either printing out or writing down the generated data, the user successfully creates a paper wallet. Hardware wallets externally sign transactions via a trusted execution environment. Users connect their hardware wallet to a PC (usually by USB) to validate unsigned and queued transactions. The hardware wallet remains offline, signs the unsigned transactions, and sends the signed transaction back to the PC to broadcast to the network. Finally, brain wallets rely on the user memorizing their own private key, typically by memorizing a series of unrelated words which hash to the private key.

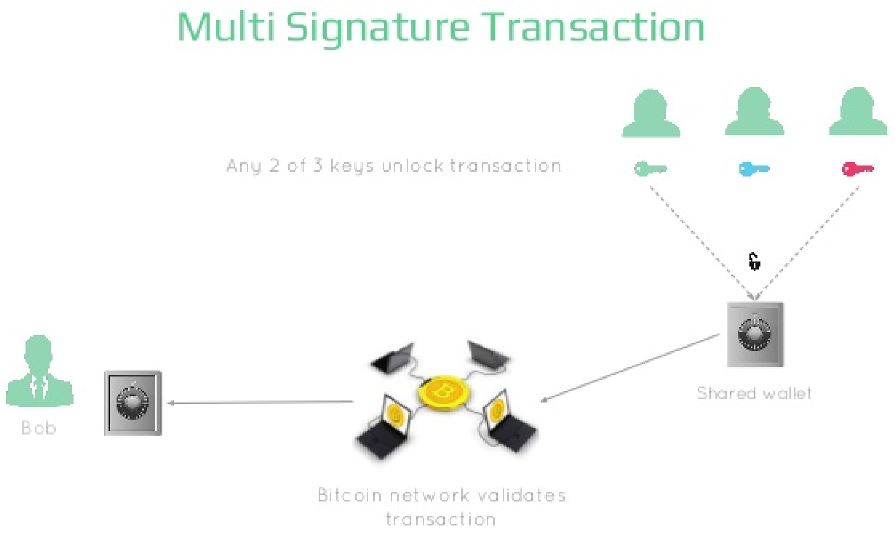
Key Stretching

Regarding the aforementioned brain wallets, consider the human mind's characteristic ineptitude to generate randomness. Users attempt to maximize 'randomness' between words but will never surpass a computer's efforts. We also lose some randomness by only considering words and not random characters within the base 58 character-set, exposing brain wallet to brute force attacks.

Rather than abandoning brain wallets, consider **key stretching**. Hash the hidden sequence of words repeatedly rather than just once. For example, if Alice picks a sequence of words and hashes it 2^{20} times, making her resulting private key exponentially harder to brute force. Although private key regeneration now takes much more work, a malicious user must guess both a sequence of English words and how many times Alice's key was stretched. A small cost for a huge gain in security.

Choosing a Wallet – Multisignature Addresses

Consider a wallet owned by a company. Should one person hold the precious private key? What happens if they steal everyone else's money? Should multiple people each hold a key? **Multisignature addresses** solve this issue by requiring multiple signatures (m of n total signatures) to sign a transaction.


Multi Signature Transaction
Any 2 of 3 keys unlock transaction
Shared wallet
Bob
Bitcoin network validates transaction

In the image above, two of three keys of the shared wallet are necessary in order to sign a transaction. The Bitcoin network validates the transaction by checking the signature produced by the two user keys.

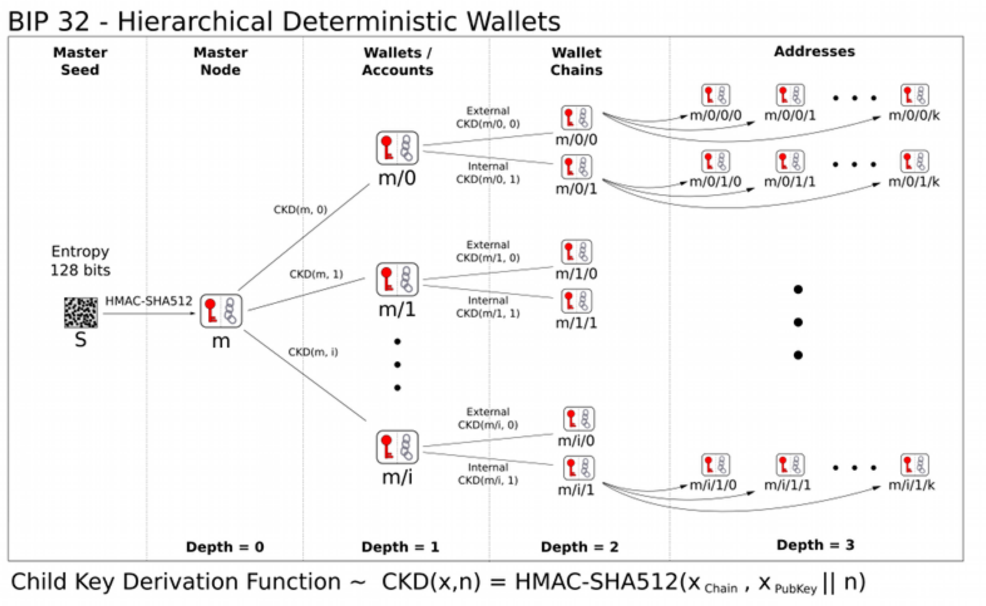
Choosing a Wallet – Generating New Keys

Users commonly generate new keys for every transaction to maintain privacy. As Bitcoin block explorers identify public keys associated with any given transaction, generating new keys for every transaction deters others from tracking your activity to determine how much bitcoin you own or follow your transaction history. Wallet software facilitates generation and storage of different keys by combining funds from different keys so that the user can see exactly how much bitcoin they have while enjoying reinforced security as a perk.

Choosing a Wallet – Wallet Backups

Wallet backups protect against loss, as hardware failures potentially lead to losing one's entire supply of bitcoin. However, generating new keys for every transaction poses a problem: if a user generates a new address with each transaction, then the safest backup scheme requires a new backup for every new address. This backup scheme describes **JBOK (Just a Bunch of Keys) wallets**.

In **BIP (Bitcoin Improvement Proposal)** 32, the Bitcoin protocol accepted **HD (hierarchical deterministic) wallets**, where hierarchal refers to various "ancestries" and deterministic refers to an unchanging hierarchy system, characterized by a process whereby a 'child' private key comes from a 'parent' private key via secure mathematical relationships, removing the burden of backing up new keys upon every transaction as with JBOK wallets. Instead, one saves a set of parent keys and later calculates child keys as needed. Each parent private key can have multiple child private keys, and each child private key can in turn be a parent private key and have children as well, defining the hierarchical property of HD wallets.



Consider HD wallets in the context of complex organizations. A master seed generates and stores a secret. Financial officers or the CEO would control the master node (private key), later distributing wallets/accounts to every department of the organization, which then distributes wallets to each employee, for a manageable hierarchical structure.

Choosing a Wallet – Control & Responsibility

We previously mentioned that Coinbase store private keys in the cloud, allowing Coinbase to spend or move funds as long as your balance stays the same whenever you use your wallet, similar to centralized bank accounts (which some Bitcoin enthusiasts do not appreciate). Centralized organizations retain your private keys, letting you use an email or phone call to retrieve your funds should you forget your Coinbase password. Otherwise, you must maintain your own backups and security.

On the other hand, Mycelium and Electrum do not hold anyone's private keys. Users hold their own private keys and are responsible for their own funds. There is no recourse from these services in the case of a forgotten key or lost funds. The benefit, of course, is security since no one else controls your private keys. However, the lack of a safety net in the case of the loss of a key may scare users away.

As a compromise, hardware wallets such as Case Wallet utilize a multi-signature scheme to sign a transaction. Three signatures are delegated, to the the device, Case service, and a separate backup company. When a user wants to conduct a transaction, two signatures are provided by the user's device as well as by the Case service. In the case of a lost device (i.e. a lost signature), a user can call Case and the separate backup company to get the remaining two signatures and retrieve their funds.

Cryptographic Hash Functions

A **cryptographic hash function**, as discussed in the previous note, maps from some arbitrarily-sized bit string to some fixed-size bit string. In mathematical terms: $H : \{0,1\}^* \mapsto \{0,1\}^k$

The Bitcoin protocol uses the cryptographic hash function SHA-256 extensively. A cryptographic hash function only evaluates in one direction, designed to prevent anyone from discovering a cryptographic hash function's input given an output. They are also deterministic, meaning that the same input always yields the same output.

Due to the **avalanche effect** of cryptographic hash functions, a small change in the input drastically changes the output. The avalance effect distributes outputs thoroughly. In other words, similar inputs will not produce similar outputs. Otherwise, given an output, playing a game of "hot-or-cold" will allow malicious actors to discover the corresponding input. By putting in various inputs and comparing the output similarity, they could modify small parts of the input until the input produces the desired output. The avalance effect removes this threat.

Preimage resistance is also a desired characteristic of good cryptographic hash functions, and describes that it should be extremely difficult to find the preimage of a cryptographic hash function output. Formally:

$$\begin{aligned}
&x = \{0,1\}^* = \text{message (preimage)}\\
&y = H(x) = \text{hash (output)}\\
&x = H^{-1}(y) \rightarrow \text{finding the message should be computationally difficult}
\end{aligned}$$

Second preimage resistance, also desired in a good cryptographic hash function, refers to immense difficulty in finding any value that maps to a specific input. Formally, for any given message x, finding any x' such that $H(x') = H(x)$ should be computationally difficult.

Finally, **collision resistance** in hash functions tells us that the probability of finding two inputs that map to the same output is very small. There are two equivalent ways of formalizing this statement. Suppose we have two messages x_1 and x_2;

1. The probability that their hashes are equal, $P\big(H(x_1) = H(x_2)\big)$, is very small, or
2. finding *any* $x_1,\ x_2$ such that $H(x_1) = H(x_2)$ is computationally difficult.

Use of Cryptographic Hash Functions

In the context of Bitcoin, we have seen cryptographic hash functions used in a number of places: miners use SHA-256 to prove the existence of relevant transactions inside of a Merkle tree; SHA-256 is used to implement Proof-of-Work; and transactions, blocks, and addresses all correspond to hash values. Outside of Bitcoin and cryptocurrencies, cryptographic hash functions find use in hash-based message authentication codes (HMACs), password verification, commitment schemes, pseudo-random number generators (PRNGs), and as the groundwork for cryptographic protocols.

Simple Hash Commitment Scheme

Consider a scenario in which Alice and Bob each bet \$100 on a coin flip. Alice calls the outcome of the coin flip, and Bob flips the coin. If Alice guesses correctly, she wins \$200. Else, the money goes to Bob. In real life, this bet is easily executed. However, what if Alice and Bob are separated and do not trust one another? Since Alice and Bob are separated, it is very easy for Bob to cheat. Alice would send Bob her guess, and Bob would simply send back the opposite of what Alice guessed (to double his money unfairly).

To solve this problem, let's use hash functions to "bind" Alice's guess with a **commitment**:

1. Alice guesses the outcome of the coin flip, B
2. Alice first chooses a large random number, R, to prevent Bob from guessing B (as you will shortly see)
3. Alice generates a commitment to the coin flip by concatenating her guess with the random number, and hashing the result: $C = H(B||R)$
4. Alice sends this commitment to Bob
5. Bob flips the coin and sends the result back to Alice
6. Alice sends Bob the random number and her guess: (R, B)
7. Bob then checks that $C = H(B||R)$ to verify Alice did not change her guess
8. Both Alice and Bob can now agree on who won the \$200

Let's examine the random number's purpose. Because of the limited options for the guess B, Bob could compute himself the hashes of all possible guesses. Upon receiving Alice's commitment, he would only need to compare the output to the possible hashes and work backwards to figure out Alice's guess, breaking the security. By adding the random number R, Bob has no way of determining which guess produced the given output so long as H is a cryptographic hash function. After all, the scheme developed above depends on the effectiveness of our function H. Any well-designed cryptographic hash function would suffice, as illustrated by the following examples:

When Bob receives the commitment $C = H(B||R)$ from Alice, if he can compute the inverse of the commitment's hash function $H^{-1}(C)$ to retrieve the input $B||R$, Bob can recover Alice's guess and play dishonestly by lying to Alice when appropriate. However, if our hash function H is preimage resistant as earlier defined, finding the input for a given output is computationally difficult, keeping both parties safe from cheating.

Alice could cheat as well by sending Bob her commitment $C = H(B||R)$ as before but revealing the opposite guess $(!B, R')$ when advantageous. Alice wins the bet if she can pick R' such that $C' = H(!B||R') = C$. However, if our hash function H is second preimage resistant, this scenario is deterred as well.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 3

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **Avalance effect** — In the context of hash functions: small changes to the input result in large changes to the output.
2. **Base 58** — Base 62 (all lowercase and uppercase letters, along with all ten digits) minus 4 particular characters to prevent mixups between similar characters.
3. **BIP (Bitcoin Improvement Proposal)** — Used to signal approval for a change in protocol. To gauge how much of the network supports a particular Bitcoin initiative, miners can store a piece of code in the coinbase transaction as a signal to tell the rest of the world their opinion.
4. **Cold wallets** — Offline wallets, typically used for storing bitcoins long-term. By never putting the private key online, the likelihood of one's bitcoins being taken from this wallet decrease substantially.
5. **Collision resistance** — In the context of hash functions: the prevention of two inputs generating the same output.
6. **Commitment** — A hash that represents the selection of some decision while obfuscating the decision. Hashing a selection with a random number produces the commitment. With both preimage and second preimage resistance, one can commit to a particular submission or claim without revealing the claim.
7. **Cryptographic hash function** — See Note 1.
8. **Hot wallets** — Online wallets, typically holding frequently exchanged and traded bitcoins.
9. **Key stretching** — Repeated hashing of a private key in the public key generation process, as opposed to only using the hash function once.
10. **Multisignature addresses** — Addresses associated with multiple keys, as opposed to a single private key, allowing multiple users to share an account or to produce backups. Most multisignature addresses do not only require a fraction of keys to validate a transaction, often 2-of-3 wallets.
11. **Preimage resistance** — A property of cryptographic hash functions. In layman's terms, it is excruciatingly difficult to create an inverse hash function that turns outputs to inputs.
12. **Second preimage resistance** — Another property of cryptographic hash functions. In layman's terms, it is excruciatingly difficult to find two inputs to a hash function which produce the same output.

13. **Wallet** — A piece of software which, at a minimum, stores one's private key. For user convenience, hot wallets also store information about past available UTXOs and protocols for creating transactions.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 4

Bitcoin Mechanics and Optimizations

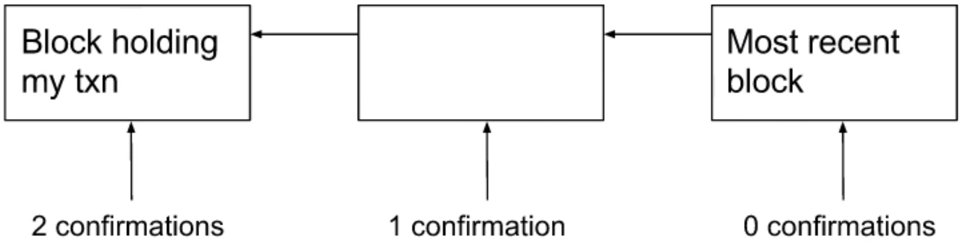
After abstracting away Bitcoin features to focus on fundamental principles and design, it's time to dive into how Bitcoin works on a technical level. In this note, we will cover topics such as advanced Bitcoin mechanics, scripting, and network optimizations techniques. Through these topics, we intend on demonstrating that Bitcoin's design philosophy is as simple, robust, and conservative as possible.

Double Spend — Example

To gain further intuition on Bitcoin consensus, let's understand once more the problem it solves: double spending. Previously, we defined a "double spend" as a circumstance in which a party successfully spends the same funds more than once. In real life, it's as if you used a single dollar bill in two separate purchases. Physically, it's impossible to double spend, but the virtual world creates issues. For example, if Alice wants to purchase something from Bob for 1.00 BTC, she can double spend by creating two transactions: send 1.00 BTC to Bob in one transaction, and 1.00 BTC to herself in another. The double spend is successful if the transaction to herself is the one accepted by the network, which will allow her to keep the money and fool Bob. Double spends are possible when a party gains more than 50% of the network's hash power. Under the hood, there is a fairly involved process that results in the double spend.

Double Spend — Confirmations

A transaction's **confirmations** are defined as the blocks built off of the block containing the transaction. All transactions in the block at the tip of the blockchain have 0 confirmations, as there are no newer blocks. A transaction's confirmations will be one less than the block depth: the most recent block is at depth 1 and has 0 confirmations, and every next block will increase both the depth and confirmations by one.

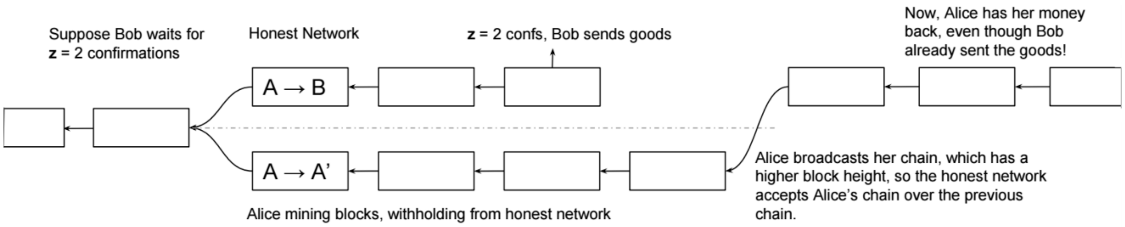


Because of the difficulty of finding the proof-of-work for each block, confirmations represent security in acceptance. Most businesses wait 6 confirmations before considering a transaction to be finalized.

In order to double spend with this confirmation system, one must procure two transactions using the same source of funds, conventionally 1) to the targeted individual and 2) to themselves. Let's call these TX_A and $TX_{A'}$ respectively. The attacker wants the other party to see TX_A but wants $TX_{A'}$ to be the one included in the longest chain, tricking the other party into believing that they will receive funds. The attacker must propagate TX_A to the network, assuming it will inevitably be included in a block, and locally generate the dubious $TX_{A'}$. The attacker will not send this second transaction to the network's **mempool** as it's being used by the attacker alone and would cause confusion should it enter the network and conflict with TX_A. They will include this second transaction in their own block which will be used to make a fork of the blockchain. From there, the attacker must find the proof-of-work *for each block until their fork of the blockchain is the longest*. In the diagram above, an attacker would have to find 3 different valid blocks and fork from this entire chain to reach the same height as the most recent block and invalidate the block holding TX_A. This assumes has a significantly higher hash rate than that of the rest of the network, which is working on the original chain. (There are tricks to conduct such an attack without a majority of the hash rate which will be introduced in the "Game Theory" lecture and note.)

Why do confirmations matter? Suppose Bob does not wait for any confirmations on Alice's transaction. He checks that the transaction from Alice is valid and *immediately* sends Alice the product he is selling, such as if Alice were buying coffee at a cafe where waiting up to an hour for confirmations would inconvenience everyone involved. Since there are no confirmations on Bob's transaction, Bob is vulnerable to a **race attack**. Alice can send a competing transaction to herself at the same time she sends Bob the legitimate transaction, as previously mentioned. Alice can then broadcast the transaction to the entire network. The false, conflicting transaction Alice sends is propagated much faster than the single transaction she sends to Bob, so there is a high likelihood that one of the Alice's conflicting transactions will be mined into a block and accepted by the network as genuine. Alice can also increase the block fee for the transaction to provide more incentive for miners to include the illegitimate transaction into their blocks. In this case, merchant Bob's waiting for 0 transactions makes him vulnerable to anyone that wants to steal goods.

On the other hand, in the case that Bob waits for more than 1 confirmation (z confirmations), Alice must go through a much more involved process to double spend. Say Alice sends Bob a transaction, and Bob waits z confirmations before he sends the goods to Alice. In order to double spend on Bob, Alice must mine on her own private chain, one that does not have the transaction that sent bitcoin from Alice to Bob; Alice can send the same transaction to herself instead. She builds up an illegitimate transaction history such that she sends money to herself rather than to Bob. After mining z blocks, and making sure that her chain is longer than any other chain, Alice can broadcast her chain (preferably after she receives the goods from Bob). The network will adopt Alice's chain over the previous chain because it is longer in length. In fact, it is proven that if Alice has greater than 50% of the total hash power, then she can always generate a longer, illegitimate chain.


Suppose Bob waits for
z = 2 confirmations
Honest Network
z = 2 confs, Bob sends goods
Now, Alice has her money back, even though Bob already sent the goods!
A → B
A → A'
Alice broadcasts her chain, which has a higher block height, so the honest network accepts Alice's chain over the previous chain.
Alice mining blocks, withholding from honest network

Double Spend — Security

Let's model our assertions about double spending and hash power with mathematics:

Suppose Bob waits for z confirmations before sending his goods over to Alice. Alice has h_A hash power, and the honest network has h_H hash power. The total network hash power is then $h_A + h_H$. The probability, p, of the honest network finding the next block is simply the proportion of hash power the honest network controls:

$$p = \frac{h_H}{h_A + h_H}$$

The probability, q, of Alice finding the next block is the complement of p. We calculate it as follows:

$$\begin{aligned} q &= 1 - p \\ &= 1 - \frac{h_H}{h_A + h_H} \\ &= \frac{h_A}{h_A + h_H} \end{aligned}$$

Consider the scenario in which the honest network mines z blocks on top of Alice's transaction to him. Bob sees that there are z confirmations, and then he sends the goods over to Alice. In the meantime, Alice has been hard at work mining on her own chain ever since sending her transaction over, in hopes of double spending on Bob. If λ equals the expected number of blocks mined by Alice in the time that z blocks are mined by the honest network, then we see that λ depends on the ratio of the mining power of each entity (Alice vs the honest network):

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

(<u>Sanity check</u>: if $p = q = 0.5$, then $\lambda = z$, meaning that we expect Alice to mine mine the exact same number of blocks as the honest network. This makes sense, that two entities with equal power would produce an equal number of valid blocks.)

If we model this as a **Poisson distribution**, because we have discrete events each corresponding to a probability, we can show that the probability that Alice generates k blocks is:

$$p_A(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Now the question is: What is the probability Alice can mine enough blocks in secret to successfully broadcast her chain with the double spend? In other words, what is the probability that Alice can produce a longer chain than the honest network?

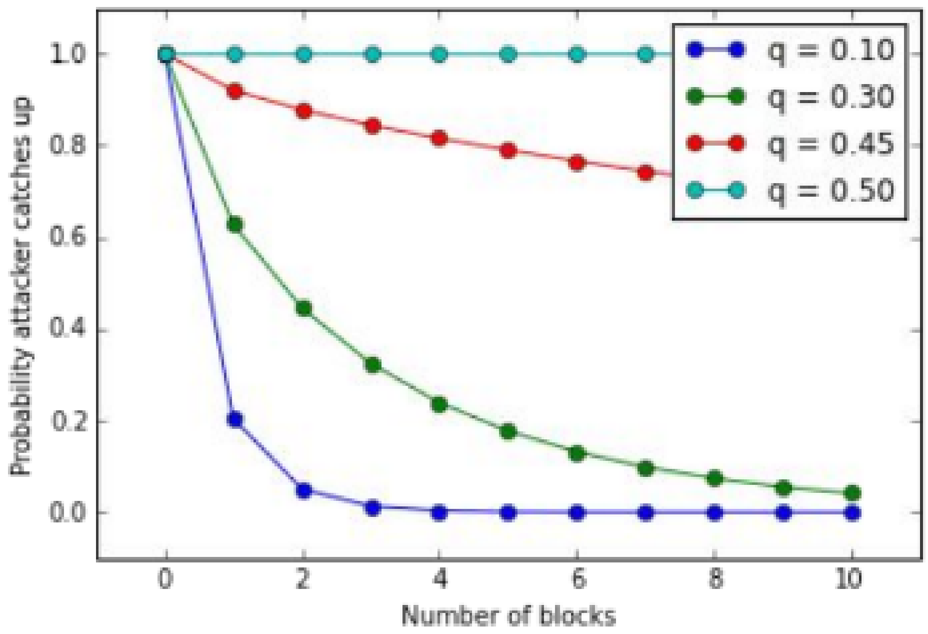
To solve this, first consider the related problem of Alice trying to catch up to a chain that is j blocks ahead of Alice's own chain. The corresponding question is: What is the probability that Alice will ever catch up given an unlimited number of trials? It is also important to consider that the honest network is continuously mining their blocks on their chain, while Alice works maliciously on the side. This problem is actually an instance of the **Gambler's Ruin** problem, which in this case means that if Alice has more than 50% hash power, she can always generate a longer chain, given infinite time. The following is the probability of Alice catching up if she is j blocks behind:

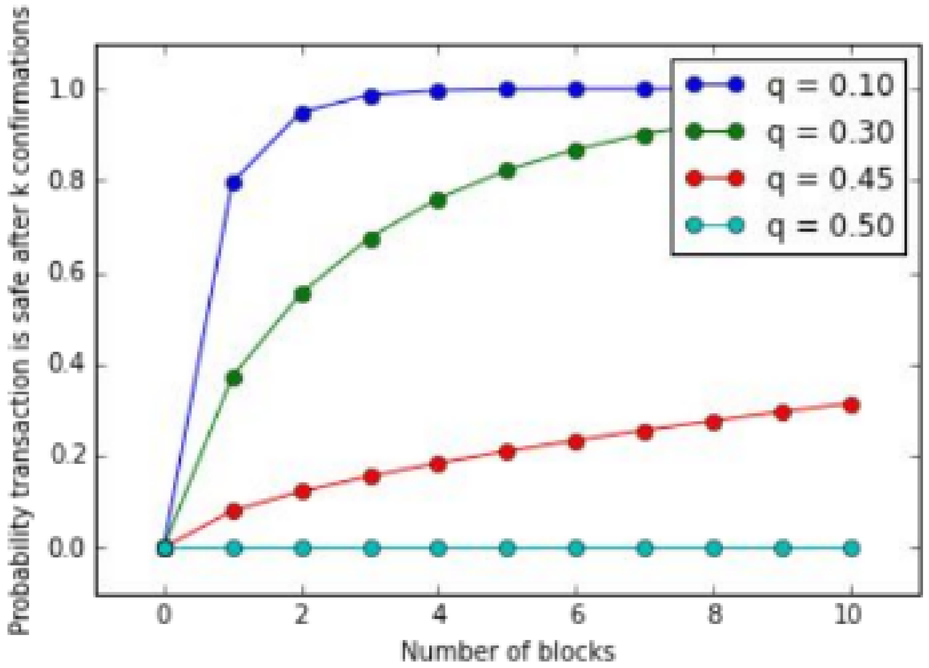
$$p_c(j) = \begin{cases} 1 & q \geq p \text{ OR } j < 0 \\ \left(\frac{q}{p}\right)^j & q < p \end{cases}$$

In other words, Alice can succeed in eventually producing a longer chain if she either has at least as much hash power than the rest of the network or if she is already ahead of the rest of the network. On the other hand, if she has less hash power than the rest of the network, she may still be able to catch up based on the ratio of her own hash power fraction to the rest of the network. The more blocks she is behind, the smaller her chance of producing the longest chain and "winning."

Combining these two probabilities, we can compute the probability that Alice can catch up after z blocks mined on the honest chain. Consider the case when Alice mines k blocks. This means that the honest chain is $z - k$ blocks ahead of Alice. We sum over all possible values of k:

$$\begin{aligned} &\Sigma_{k=0}^{\infty} p_A(k) \cdot p_C(z-k) \\ =&\Sigma_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} 1 & k > z \\ \left(\frac{q}{p}\right)^{z-k} & k \leq z \end{cases} \end{aligned}$$





The above shows Alice's probability of catching up to the honest chain in arbitrary x number of blocks. If Alice has 50% hash power or more, then she can catch up with 100% probability. Probability of catching up to the honest chain decreases as Alice's probability q of getting the next block decreases.

To avoid **infinite tail discrete summation**, we can look at the inverse probability that Alice *cannot* catch up to the chain that is j blocks ahead:

$$
\begin{aligned}
&1 - \Sigma_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} 1 & k > z \\ \left(\frac{q}{p}\right)^{z-k} & k \leq z \end{cases} \\
&= 1 - \Sigma_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \left(1 - \left(\frac{q}{p}\right)^{z-k}\right)
\end{aligned}
$$

The number of confirmations Bob should wait before sending Alice the goods depends on how much hash power Bob assumes Alice to control. After 6 confirmations, the likelihood of a double spend attack drops to zero assuming any malicious party controls 10% or less of the network hash power.

Double Spend — Bribing Miners

We have shown that if Alice controls more than 50% of the total network hash power, she will always be able to double spend. Looking back at our calculations, whenever Alice is some j blocks behind the honest network's chain, she will *always* (given enough time) be able to catch up and out-produce the honest miners. Hence, the probability that Alice can successfully double spend with greater than 50% network hash power is 1.

In reality, it is extremely costly and implausible for Alice to own so much network hash power, especially given the Bitcoin network's current vastness. While Alice might not physically control mining hardware for a double spend attack, she can bribe other willing miners or even entire mining pools to mine on her chain. In the long run, Alice and her miners will profit after managing to overcome the extremely high initial cost.

Double Spend — "Goldfinger" Attack

"Your scientists were so preoccupied with whether or not they could, they didn't stop to think if they should." — Ian Malcolm, *Jurassic Park*

Is double spending a good idea, even with the proper resources? If Alice does in fact succeed, confidence in Bitcoin's value would plummet after the rest of the network detects the security breach. Alice's bitcoins would also tank in value, assuming Alice is staked in Bitcoin when she performs the attack.

However, if Alice wants to use this price drop to her own advantage, she can instead **short** Bitcoin in order to profit after her double spend. Alice can short the exchange rate and use that as collateral to buy, rent, or bribe miners, cashing out on the short.

Consider Alice as a hostile government, adversarial altcoin, or large financial institution with a large amount of **capital**. Alice would then be able to acquire enough mining equipment or bribe enough miners or pools to achieve greater than 50% hash power. Alice can then perform a so-called **Goldfinger attack**, with the objective of destroying the target cryptocurrency with a double spend or spamming the network with empty blocks.

(Note: The Goldfinger attack references the famous villain Auric Goldfinger from the third James Bond movie, who poisons the US gold supply in Ft. Knox to reduce the value of that US gold and to increase the value of his own gold holdings.)

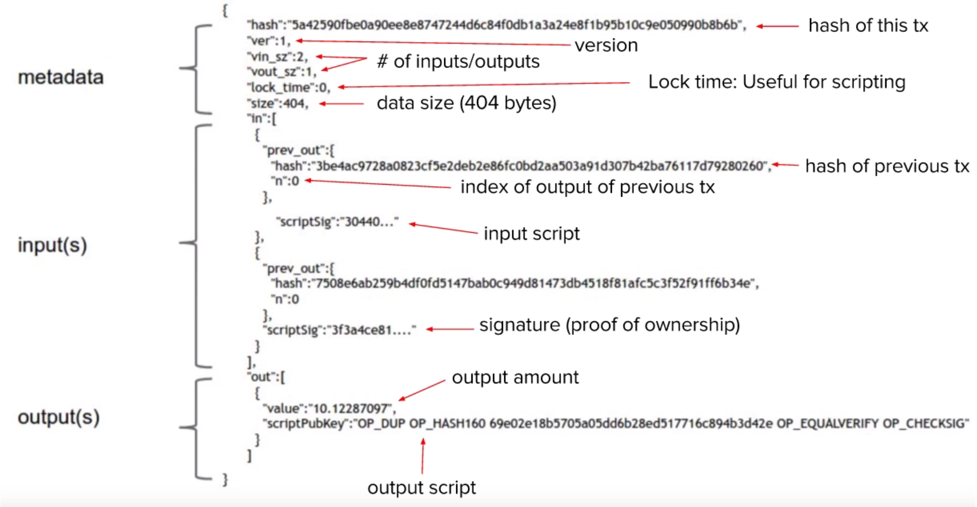
Account vs Transaction Based Ledgers

Consider the notion of **account-based ledgers**. Cryptocurrencies such as Ethereum maintain account-based ledgers. Each address in Ethereum represents an account, and for a user to check how many tokens they have, they simply add the inputs and subtract the outputs. The downside to this method is that the system must track every single transaction that has ever affected that one account, a potential performance issue when scaling to thousands of users. A possible fix? 'Pruning' away old transaction history so that the user does not have to deal with old transaction data. (Currently error-prone due to additional required block maintenance.)

Bitcoin is a **transaction-based ledger** (also known as triple-entry accounting). Users are always spending from previous outputs. As we mentioned in Note 2, UTXOs (unspent transaction outputs) can only be spent once. When someone wants to spend a portion of a UTXO, they send part to the receiving party and the remaining to a "change address" controlled by themselves, for later use. Restricting UTXO usage and introducing the concept of change addresses produces an efficient verification system: to determine how many tokens a user has, sum their valid unspent UTXOs. (Note: Bitcoin also supports joint payments, so that a transaction can have multiple inputs – for example if Alice and Bob both want to pitch in to spend a total of 1.00 BTC. In practice, this feature is rarely used but is supported nonetheless.)

Contents of a Transaction

The contents of a Bitcoin transaction fall into three categories: metadata, inputs, and outputs.



```
{
  "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver":1,
  "vin_sz":2,
  "vout_sz":1,
  "lock_time":0,
  "size":404,
  "in":[
   {
    "prev_out":{
     "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
     "n":0
    },

      "scriptSig":"30440..."
   },
   {
    "prev_out":{
     "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
     "n":0
    },
    "scriptSig":"3f3a4ce81...."
   }
  ],
  "out":[
   {
    "value":"10.12287097",
    "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
   }
  ]
}
```

As seen above, a transaction follows **JSON** structure. Metadata contains information such as the hash of the current transaction (aka the transaction ID), the **version number** (currently 1), the number of inputs and outputs, **lock time**, and **data size**.

Inputs to the transaction include a hash of a previous unspent transaction and an index, which specify the output of the previous transaction from which the user is spending. Transaction outputs are ordered, so an index is required to identify specific outputs in a transaction. Inputs also need a cryptographic signature as a proof of ownership.

The outputs of a transaction include an output amount in BTC as well as an **output script**. The output script is what enables the parties associated with the transaction to claim their bitcoins later on, using a system called the Pay-to-PubkeyHash, which we will discuss later. Shallowly, the script says, “Whoever owns the private key to this public key can redeem these bitcoins.”

Bitcoin Scripting

Previously we stated that transactions map input addresses to output addresses. In practice, these “addresses” are scripts. By defining inputs and outputs through scripting, we can allow for the future extensibility of Bitcoin because scripting is such a low level feature that could potentially support many other future operations if needed. Recall that a signature proves the ownership of a public key, and that the Bitcoin address is a hash of the public key.

$$Hash(PubKey)\ ==\ Address\ ==\ ''PubKeyHash''$$

Inputs and outputs are scripts that specify these addressses, and sending money through transactions involves redeeming previously unspent transaction outputs.

So outputs in a transaction must be constructed in a way that tells the network:

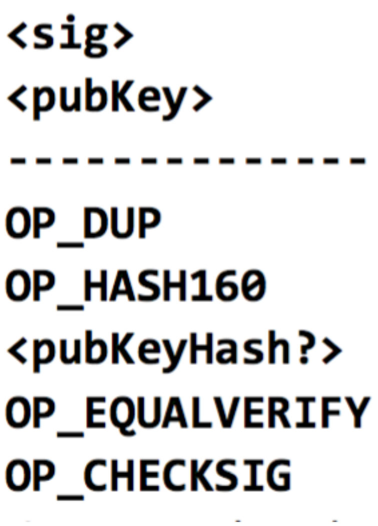
”This amount can be redeemed by a signature from the owner of address X.”

However, by the cryptographic property of preimage resistance, we know that we cannot find a public key given an address. What the transaction should really be saying is:

“This amount can be redeemed by the public key that hashes to address X, plus a signature from the owner of that public key.”

Only the owner of the public key would be able to produce the valid signature necessary to redeem the bitcoin from the output script.

To make input scripts compatible with output scripts, we concatenate the input script to the output script, in that order. Input scripts are called **scriptSigs** and output scripts are called **scriptPubKeys**. This is because output scripts are specified by the senders of the transaction. Outputs need to know a provider – a public key that hashes to the associated address, and makes sure that the signature matches.

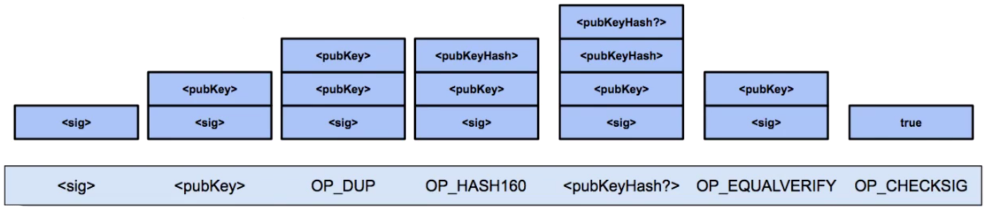


```
<sig>
<pubKey>
--------------
OP_DUP
OP_HASH160
<pubKeyHash?>
OP_EQUALVERIFY
OP_CHECKSIG
```

In terms of code execution, scripts are run line by line, top to bottom. Notice in the diagram how the scriptSig is followed by the scriptPubKey because of the concatenation convention, and also because outputs need to know where the funds are coming from.

The Bitcoin scripting language, called **Script**, (the former is used frequently too) is a simple stack based language. There exists no support for loops, so the language is not Turing complete, but there exists native support for cryptography, making the language very specialized for its own needs. In fact, the entire signature verification process can be written in code as one instruction.

Next, we present an example execution of the previously shown script.



The first two steps are <sig> and <pubKey>, so we push those to the stack in that order. Next, OP_DUP simply duplicates the most recent instruction, so by the end of executing the third instruction, there are two <pubKey>'s. OP_HASH160 hashes the topmost item, <pubKey>, into <pubKeyHash>, as the name implies. <pubKeyHash?> is the address that redeemers of the output must hash their public key to. Next, in the instruction OP_EQUALVERIFY, the script checks to see if <pubKey> truly hashes to the address <pubKeyHash?>. The script then cross checks this with the signature, <sig>, in OP_CHECKSIG, which returns true if the redeemer of this output is truly verified to spend from this output. In a nutshell, the script returns either true or false, depending on the legitimacy of the <sig> and <pubKey> that are passed in.

The output is saying: ”This amount can be redeemed by

1. the <pubKey> that hashes to the address <pubKeyHash?>
2. plus a <sig> from the owner of that <pubKey>, which will make the output script evaluate

to true."

Proof-of-Burn

One application of Bitcoin scripting is to implement something known as a **proof-of-burn**, which allows a user to prove the existence of some data in exchange for destroying bitcoin. There exists an instruction named OP_RETURN that throws an error when reached, stopping code execution. By placing this instruction anywhere in an output script before the script returns either true or false, we can effectively make it so that no one can redeem that output. We have *burned* that coin. Anything that exists after the OP_RETURN will never be executed, but this allows us to prove its existence cryptographically.

As seen in the sample script above, as long as we're willing to burn coin, we can prove the existence of anything at a particular point in time. This would be a word you coined, a hash of a document, music, your creative works, etc. If Alice stores a hash of a word she coined in the blockchain via proof-of-burn, she could then show everyone that "Alice coined word asdf at time 12345." Any data stored in this way in the blockchain is valid because blocks are timestamped. This is especially true when considering that the blockchain is provably immutable with honest actors. There have also been instances of burning bitcoin to bootstrap the value of other cryptocurrencies. To bootstrap an imaginary SuperAwesomeCoin, Super Awesome Bob could require users to burn bitcoin in order to get superAwesomeCoin.

Pay-to-PubKey-Hash vs Pay-to-Script-Hash

The previous example of requiring a public key and a signature in order to spend from a transaction output script is a use case of **P2PKH (pay-to-pubkey-hash)**, in which the vendor says "send your coins to the hash of this public key." This scheme represents the simplest and by far the most common case of transaction.

However, for more complicated scripts, such as those which require multiple signature verification, P2PKH no longer works. For instance, if a merchant wants Alice to send coin payment to an output that allows the merchant to spend using multiple signatures, how would Alice know how to specify such a complicated script?

The solution is to use a **pay-to-script-hash (P2SH)**. To clarify, consider following general cases:

- P2PKH: "Send your coins to the hash of this public key."
- P2SH: "Send your coins to the hash of this script. To redeem those coins, you must reveal the script that has the given hash and provide data that will make the script evaluate to true."

One of the most important improvements to the Bitcoin protocol since its inception, P2SH offloads the burden of complicated script writing to recipients of a transaction. When a merchant wants to receive payments from a customer, they do not want to burden the customer with writing a complicated script that could potentially differ between merchants. Instead, the merchant alone is responsible for writing a correct and secure script for the transaction. Likewise, the optimal customer experience is one in which the customer does not have to care about what the script actually is. They should not have to know anything about the company stores funds; customers should just have to create a transaction, pay, and leave.

Merkle Trees

Let's take a deeper dive into the specifics of Merkle trees, which we briefly touched upon in note 2. As review, merkle trees are inary trees of hash pointers. Blobs of transaction data are hashed together, then their hashes are hashed together, until one element remains—the Merkle root. (Note: Merkle trees are always full. If there are gaps, duplicate the last transaction to fill in the gaps.)

Outputs of cryptographic hash functions are unique by second preimage resistance, making Merkle trees efficient. The Merkle root serves as a summary of the Merkle tree and as a mechanism to maintain history of included information. To prove inclusion of data, one must provide root data and intermediate hashes. After hashing everything together in order, one compares the final hash to the merkle root. To fake this proof, one would need to find hash preimages that hash to values in the merkle tree. As discussed earlier, second preimage resistance makes faking proofs in merkle trees extremely difficult and practically impossible.

The figure above illustrates how one might prove the inclusion of a substring *data*, the circled leaf node. Provided the circled *data*, the proof proceeds as follows.

1. Hash *data* and call it H_{data}
2. Hash H_{data} with the next intermediate hash (height 2)
3. Continue hashing until we reach height 0
4. Hashing the two hashes at height 0 results in H_{root}
   (a) If H_{root} is the same as the merkle root, then we have proven the existence of *data* within the merkle tree
   (b) Else, the proof failed and the merkle tree does not contain *data*

Merkle Trees — Bitcoin Construction

There are two main hash structures in Bitcoin. The blockchain is a hash chain of blocks; the blocks are linked together and based off of each other. They are tamper evident because changing one block changes its hash, which mismatches with the next block's hash of the previous block. Merkle trees exist within blocks and are a way of storing transactions. Changing data within a merkle tree changes its hashes, ultimately bubbling up and changing the merkle root. Changing the merkle root in turn changes the hash of the block it's contained in, invalidating the block.

Merkle Trees — Mining, In More Detail

Previously, we explained for simplicity that for every block, miners hashed together the merkle root, the previous block's hash, and a nonce (varied value) to find a number that is below a certain target value. There are actually two nonces: one in the block header as mentioned, and one in the coinbase transaction, the transaction that is created by and paid out to the miner. Changing the nonce in the coinbase transaction changes the hash of the coinbase transaction, ultimately changing the merkle root.

The reason why there are two nonces is to increase difficulty for the miner. The block header nonce is 32 bits by itself. A modern ASIC such as the Antminer S9 can hash at 14 TH/s. A simple calculation shows that it takes just 0.00031 seconds to compute all the possible combinations in the block header nonce. $2^{32}/14,000,000,000,000 = 0.00031$ seconds. A miner with the right hardware can exhaust all nonce combinations 3260 times per second. Therefore, it is imperative to change the merkle root by including the coinbase transaction nonce.

A common strategy is to increment the coinbase nonce, and then run through all block header nonce combinations. A less efficient strategy is to increment the block header nonce, and then run through all combinations for the coinbase nonce. This is because changing the coinbase nonce changes the merkle root, and the change must propagate up the tree, wasting precious hash time. (Propagating up the merkle tree takes $\theta(\log N)$ time, whereas calculating a hash takes $\theta(1)$ time.) We want to minimize the time spent calculating new merkle tree hashes, so change the coinbase transaction as little times as possible.

SPV — Simplified Payment Verification

The current size of Bitcoin's blockchain is 122.7 gigabytes and growing. Miners, or "full nodes", are required to save the entire blockchain, but for the average user, such a requirement is not feasible if the aim is mass adoption. Enter **SPV (Simplified Payment Verification) nodes**, or "thin" clients. These nodes are designed to be lightweight, as their name implies. They only store the pieces of data needed to verify transactions that concern them, thus relieving the necessity to store the entire blockchain. Nearly all nodes in the Bitcoin network are SPV nodes because users are discouraged by the enormous download size. Those that do run full nodes exchange large amounts of storage and a high bandwidth for a shot at the block reward.

SPV nodes only keep the block headers of the blockchain. This is done by querying different full nodes until the SPV node has the longest chain. To validate an incoming transaction, an SPV node queries full nodes to get the Merkle branch for that transaction. Then, the node hashes the transaction together with intermediate hashes to obtain a Merkle root, which is cross checked with the Merkle root in the corresponding block header that the SPV node has locally. The only thing left to do after this point is to wait for the transaction to have enough confirmations (six) before delivering any goods.

SPV — Security and Cost Analysis

By definition, SPV nodes do not have a full transaction history, and do not know the UTXO set. Therefore, SPV nodes do not have the same level of security as full nodes. The reason is that SPV nodes cannot check if every transaction included in a block is actually valid.

One major assumption SPV nodes make in exchange for their light weight is that they assume incoming block headers are not a false chain. This is a fair assumption to make because block headers include the proof-of-work for each block, and it is very expensive for attackers to create blocks. Over the long term, as long as the majority of the network is honest, SPV nodes can safely assume that the longest chain is honest because malicious behavior is not sustainable. Another assumption is that there are other full nodes out in the network validating all transactions. It is often inefficient for big merchants to query remote full nodes to verify transactions. Instead, it makes sense to keep a full blockchain to run fast local checks. Lastly, SPV nodes also assume that miners ensure that the transactions they include in their bocks are valid. If a miner were to include an invalid transaction, after they find the proof-of-work for that block and propagate it, other full nodes would reject their block because of the invalid transaction. Thus, miners are well incentivized to make sure all transactions in their blocks are valid.

The storage tradeoff for running an SPV is huge. Bock headers are only around $\frac{1}{1000}$ the size of the full blockchain. This translates to 123 MB vs. 123 GB. SPV nodes capitalize on obtaining data for verifying transactions lazily, relying on data from full nodes rather than keeping it locally. For most consumers and users of Bitcoin, SPV is a decent tradeoff.

Flooding Algorithm

We mentioned that SPV nodes query full nodes to get data on block headers. The mechanism through which an SPV node establishes connection to full nodes is worthy of analysis too, especially since we have been fairly hand-wavy when refering to the “Bitcoin network.”

The Bitcoin network ensures that all nodes are equal; there exist no hierarchy of nodes or special nodes that have priority in any process: consensus, transactions, etc. The network is thus fully decentralized. The Bitcoin network is also defined by the property of random topology, meaning that each node in the network pairs with random nodes. Messages between nodes are generally taken as true, and the default behavior for nodes is to accept whichever message they hear about first, but this is not a strictly enforced rule in the network.

The process of joining the Bitcoin network is aided by the presence of hard-coded **seed nodes** in the Bitcoin software. The algorithm is as follows:

1. Pick a seed node and ask for its peers
2. Ask those peers for their peers, etc.
3. Eventually, you pick a random set of nodes to pair with. These nodes become your peers.

In general, this type of message-sending schema is characterized by its pairwise connections, and is called a **flooding algorithm** or **gossip protocol**. The end goal is for the entire network to hear about a transaction. Consider a simple example with our favorite characters Alice and Bob:

1. Alice wants to pay Bob, so Alice first constructs a transaction and tells all of her peers
2. Her peers conduct checks on the transaction and if it passes, they relay the transaction to their peers
3. Each peer conducts checks such as the following:
   - Does the script for each previous output being redeemed return true?
   - Have all redeemed outputs not been spent?
   - Have I already seen this transaction? Do not relay it if so.
   - Only accept and relay ”standard scripts“: based off a small whitelist of scripts.
4. Eventually, the transaction makes it to Bob

Bitcoin Relay Network

One problem Bitcoin is currently facing is **mining pool centralization**. Miners are incentives to join large mining pools because smaller mining pools generally experience higher orphaning rates due to lesser hash power and network connectivity. Large pool on the other hand have lower block orphan rates becauses of the effects of **block propagation delays**.

One solution to this issue is the adoption of the **Bitcoin Relay Network**, also known as the Fast Relay Network. The idea is to serve a high speed “railroad for block data.” The design involves installing nodes in China/Asia, Europe, North America, etc., prioritizing first where the majority of miners are geographically located. Each of the nodes would have a fast internet connection, and would be able to relay data via compression and transmission via TCP to nearby miners. This would rapidly improve bock propagation speed, and thus reduce the number of orphaned blocks.

The problem with the Bitcoin Relay Network is that it relies on TCP as its connectivity protocol. TCP is susceptible to data loss. A sender might send x number packets of data to a receiver, but not all x may arrive. In the case where not all packets arrive, the receiver is notified, and can then ask to sender to resend the missing packets. This causes a round trip delay for the time-sensitive data.

FIBRE

After realizing the problems with the Bitcoin Relay Network, developers conceptualized **FIBRE (Fast Internet Bitcoin Relay Engine)**. Problems with the Bitcoin Relay Network boiled down to its reliance on TCP, which only implements error correction at the IP leve. It uses ARQ (Automatically Repeat Request) to fix errors, which requires another round trip from receiver to sender, back to the receiver. FIBRE fixes this by opting for UDP (user Datagram Protocol), which allows for FEC (Forward Error Correction). This means that if there are errors in FIBRE, the receiver is ale to reconstruct the correct data block without having to contact the sender to send it again. The sender simply includes extra packets to account for potential packet loss or corruption.

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **Account-based ledger** — An account-based ledger associates
2. **Bitcoin Relay Network** — Definition.
3. **Block propagation delays** — Definition.
4. **Confirmations** — Definition.
5. **Fast Internet Bitcoin Relay Engine (FIBRE)** — Definition.
6. **Flooding algorithm** — Definition.
7. **Gambler's Ruin** — Definition.
8. **Goldfinger attack** — Definition.
9. **Gossip protocol** — Definition.
10. **Mining pool centralization** — Definition.
11. **Pay to Public Key Hash (P2PKH)** — Definition.
12. **Pay to Public Script (P2PS)** — Definition.
13. **Proof-of-burn** — Definition.
14. **Race attack** — Definition.
15. **Seed node** — Definition.
16. **Simplified Payment Verification (SPV)** — Definition.
17. **Script** — Definition.
18. **ScriptPubKeys** — Definition.
19. **ScriptSigs** — Definition.
20. **Transaction-based ledger** — Definition.

Cryptocurrency Mining: Proof-of-Work Consensus

Bitcoin mining is the process through which new bitcoins enter the network and is also what keeps the network healthy and resistant to malicious parties—an arms race that rewards those who invest the most into their hardware. But why do miners mine? What incentives do they have, and most importantly, is it profitable? In this section, we will explain in depth the design and mechanics behind Bitcoin mining, taking into account various mining incentives.

Recap: What a Miner Does

In a nutshell, a Bitcoin miner accomplishes the following six tasks:

1. *Download* the entire Bitcoin blockchain to store the entire transaction history.
2. *Verify* incoming transactions by checking signatures and confirming the existence of valid bitcoins.
3. *Create* a block using collected valid transactions.
4. *Find* a valid nonce to create a valid block header (this is the "mining" part).
5. *Hope* that your block is accepted by other nodes and not defeated by a competitor block.
6. *Profit!* Coinbase transacation rewards miners for their work.

The Bitcoin miner maintains the Bitcoin network's health by creating valid blocks and participating honestly in the Proof-of-Work protocol. They ensure that (hopefully) only valid blocks make their way into the blockchain, receiving a bitcoin reward in exchange (also functioning as a mechanism through which new coin created in the Bitcoin network).

Block Difficulty — Analogy

Imagine a game in which you are blindfolded, throwing darts randomly at a dart board. (Say for simplicity that all your darts are guaranteed to land on the dartboard.) There is an equal likelihood of hitting any given point on the dartboard. If one of your darts lands within a certain radius d from the center, then you get a reward. Throwing faster allows for more hits per second, granting a higher likelihood of any of your darts land within the target distance and a higher chance of winning the reward.

Now imagine competing with many other blindfolded individuals to hit the target first. The difficulty of hitting the target is inversely proportional to d. In other words, it's harder to hit a smaller target and easier to hit a larger one. To control how many rewards are distributed within a given period of time, d changes depending on the average time taken to hit the target. This way, if people get better at throwing darts, then d gets smaller, and vice versa.

Block Difficulty — Puzzle Prerequisites

Instead of blindly throwing darts at a dartboard, mining in Bitcoin involves solving a **hash puzzle** that results in a valid block. Miners race each other to find a nonce that makes the following inequality true:

$$H\big(nonce \mid\mid H(previous\ block\ header) \mid\mid Merkle\ root\big)\ <\ target$$

Hash puzzles must meet a few significant requirements:

- Computational difficulty. If finding the proof-of-work required little work, then reaching consensus would be difficult (See Note 4). This is why in the analogy, we blindfolded the dart-throwers.
- Variable cost. This allows for adjustments as the global hash power changes, to let the puzzle difficulty change alongside it. In the analogy, the target size changes with d.
- Easily verifiable. There should not be a need for a central authority to verify nonce validity. Every miner simply rehashes the nonce and other relevant information to verify validity. In the dart analogy, darts stick to the dartboard, so to verify, others simply have to take their blindfolds off to verify.

Block Difficulty — Adjustment

The following is the equation to adjust the **difficulty** of the hash puzzle used in Bitcoin:

$$\textit{difficulty}\ =\ \textit{difficulty}\ *\ \textit{two\_weeks}\ /\ \textit{time\_to\_mine\_prev\_2016\_blocks}$$

We use the ratio between two weeks and the time to mine 2016 blocks because the target block time is 10 minutes:

$$\textit{2016 blocks}\ *\ \textit{10}\ \frac{\textit{min}}{\textit{block}} = \textit{20160 min} = \textit{2 weeks}$$

If the time to mine 2016 blocks is greater than two weeks, then the hash puzzle is too hard (ratio less than 1), so the difficulty scales down. If the time to mine 2016 blocks is less than two weeks, then the hash puzzle is too easy (ratio greater than one), so the difficulty scales up. The block difficulty is inversely proportional to the time required to mine the previous 2016 blocks.

How to Profit From Mining

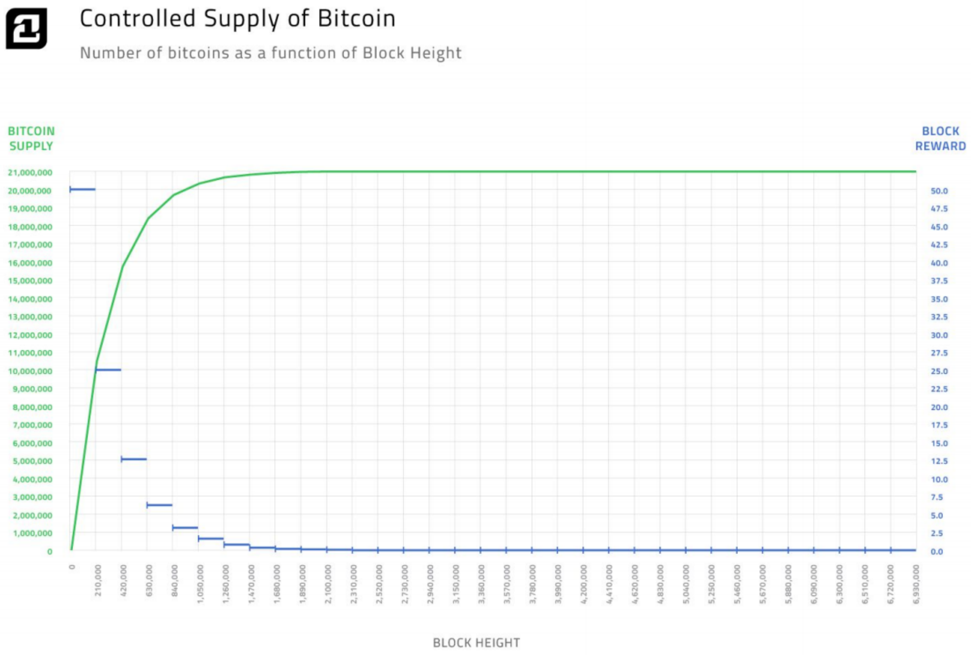
For the Bitcoin miner, the following equations are true.

$$\begin{aligned} MINING\_REWARD &= BLOCK\_REWARD + TX\_FEES \\ MINING\_COST &= HARDWARE\_COST + OPERATING\_COSTS \end{aligned}$$

If mining rewards exceed mining costs, then a Bitcoin miner will profit. In the next sections, we will analyze each of the components in the above equations.

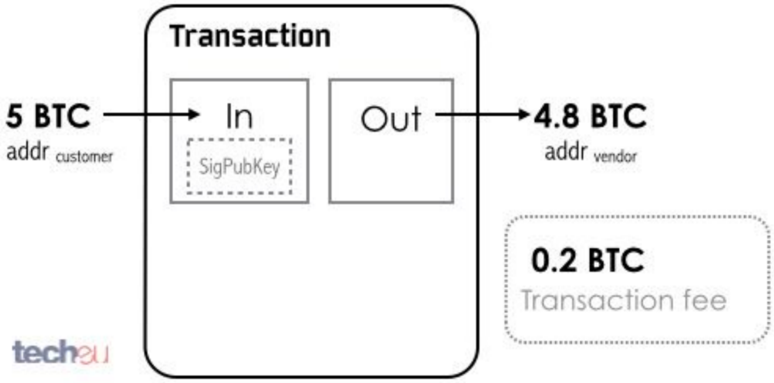
Block Reward

A Bitcoin miner receives bitcoins for producing valid blocks. As of July 23^{rd} 2017, the block reward is 12.5 BTC per block. Potential profit incentivizes honest behavior. The Bitcoin protocol requires finding the proof-of-work first, broadcasting the associated block, and awaiting confirmation to receive a **block reward**. In a block, the miner must include a special transaction to themselves, known as the **coinbase transaction**. The block reward halves every 210,000 blocks to cap the total number of bitcoins that will ever be produced. In other words, Bitcoin is a **deflationary** currency, topping off in 2140 at 21,000,000 BTC (the supply cap).



Transaction Fees

The creator of a transaction sets the transaction fee, a voluntary but practically necessary incentivize miners into prioritizing your transaction. In practice, a higher transaction fee leads to a faster confirmation time. Transaction fees provide extra income to miners on top of the block reward, increasingly important as block rewards diminish. When the block reward becomes 0, transaction fees will become the sole source of revenue for miners. To calculate transaction fees, simply subtract the output from the input amounts for the transaction. The difference implicitly becomes the transaction fee.



In the figure above, the input is 5.0 BTC and the output is 4.8 BTC. The creator of this bitcoin transaction has included an implicit transaction fee of 0.2 BTC that is to be collected by the miner who includes this transaction in the block that is confirmed by the rest of the network.

Hardware Cost

Bitcoin mining began on CPUs. As the block difficulty increased, mining efforts shifted dramatically: first to GPUs, then to FPGAs, and finally to ASICs. (Keep in mind that hardware costs are a one-time expense, unlike everything else we have mentioned.)

As commonly used CPUs began to get too slow for the block difficulty, GPU mining began to gain popularity. GPUs are an order of magnitude more effective for hashing than CPUs. However, with more power comes more expense: a larger consumption of energy and higher production of heat. GPU mining was most common around 2012, but as of now is not viable for mining Bitcoin. (GPU mining is viable for Zcash, Ethereum, and some other cryptocurrencies on the other hand.) GPU mining brought two main disadvantages:

1. GPUs have many components (like floating point units) not applicable to Bitcoin mining.
2. GPUs are not meant to be run in “farms” side by side.

The disadvantages of GPU mining made clear the need to switch to mining specific hardware.

FPGAs (Field Programmable Gate Arrays) were the result of early efforts to develop Bitcoin-specific hardware without losing all hardware customizability, a trade-off between dedicated, specialized hardware and general purpose functionality. If Bitcoin failed, SHA-256 specific hardware would be worthless, but general purpose hardware would still be useful. However, if Bitcoin thrives, specialized hardware for churning out SHA-256 calculations would generate higher profit than generalized hardware. Hence, compromise.

As Bitcoin rose in popularity, general purpose hardware was dropped altogether by many to create the first **ASICs (Application-Specific Integrated Circuits)**. ASICs for Bitcoin do nothing but SHA-256, but they do it better than any other hardware. ASICs, a diverse breed, come with various tradeoffs, one of which is between base cost and electricity usage, depending on how efficiently it uses energy. Another is device size and hashrate, as larger devices take up more space and use more electricity but have a higher overall hashrate. Also, manufacturing and purchasing ASICs takes large upfront capital, which induces centralization—a tradeoff with Bitcoin's core decentralized philosophy. Currently, the Antminer 29 holds the title of "Most Powerful ASIC," maxing out at 14 TH/s, for sale at the steep price of 3000 USD.

Operating Costs

Bitcoin mining operating costs fall into into three primary categories:

- **Embodied energy:** to produce mining hardware.
- **Electricity:** to power the hardware.
- **Cooling:** to maintain hardware health.

All energy turns into heat, dissipating into the air without serving any further purpose. Over the years, initiatives arose to reuse heat generated by operating mining hardware. The "data furnace" idea involves using mining hardware as a heater. An average household could mine bitcoins during the winter to heat up their home while making money on the side.

ASIC-Resistance — Pros and Cons

Calls for **ASIC-resistance** developed based on arguments that mining should not be facilitated with specialized hardware, that there should be no significant benefit when running a mining algorithm in an ASIC as compared a CPU. ASICs are currently the only viable option for mining bitcoins, dominating the Bitcoin network and suppressing regular people. An ASIC-resistant Bitcoin means increased democracy and decentralization, as ASIC farms would have less power over the average CPU miner.

However, there is also a strong case against ASIC-resistance. ASICs can only solve the Bitcoin hash puzzle, rendered to useless electricity-gobbling hardware in the event of a Bitcoin crash. (Attackers could rent general computing resources, however, resulting in no wasted investment—what then? Exercise for the reader.) Investment into expensive ASIC hardware becomes worthless without a puzzle to solve, discouraging attacks on the Bitcoin network. By that rationale, ASICs promote network security.

ASIC-Resistance — Memory-Hard Puzzle

Movements to make Bitcoin ASIC-resistant stirred consideration of **memory-hard puzzles**, requiring large amounts of memory rather than computational power to solve. In other words, these puzzles are **memory-bound**, meaning that memory bottlenecks computation time. Memory-hard puzzles would viably deter ASICs and effectively make Bitcoin ASIC-resistant. ASICs are optimized to perform a specific algorithm while regular CPUs are not, and since optimization of this algorithm (in our case, a hash function) is not the limiting agent, ASICs would not give some miners more power if memory is the limiting agent, not computational optimization. Memory performance increases more slowly than computational power. The cost of solving a puzzle also decreases more slowly, all valid considerations in the ASIC-resistance debate among Bitcoin users.

Scrypt

Scrypt (pronounced “ess crypt” and lowercase by default) is an algorithm used by Litecoin and Dogecoin for enforcing a memory-hard Proof-of-Work puzzle. Designed for hashing passwords and preventing brute force attacks, scrypt demands high memory space, making the size and cost of potential specialized hardware much more expensive than those for other algorithms.

Scrypt’s two main steps:

1. A user must first fill a buffer with interdependent (pseudorandom) data.
2. The user accesses the memory buffer in a pseudorandom way until a correct hash is found.

As a drawback, scrypt requires equal memory to verify, contrary to hash puzzle requirements. In addition, against scrypt’s entire meaning for existing as a cryptocurrency hash puzzle, scrypt ASICs have been developed, killing the ASIC-resistance claim.

ASIC-Resistance — Other Approaches

Memory-hard algorithms such as scrypt do not provide real-world ASIC-resistance. Initiatives using technologies such as **x11** and **x13** hash function chaining have been proposed and implemented. Dash uses x11, which chains together 11 different hash functions. Designing ASICs for x11 and x13 systems are significantly harder than designing ASICs for SHA-256 but but are not impossible, it has been done.

There is also the idea of periodically switching a mining puzzle. For example, a cryptocurrency system could go from SHA-1 to SHA-3 to Scrypt every 6 months. However, this solution is easy to work around for anyone who knows the rotation schedule. Naturally, this solution has also not been implemented. In short, ASIC-resistance seems like an impossible goal. As Mike Hearn, Bitcoin Core developer, famously stated: “There’s really no such thing as an ASIC-resistant algorithm.”

Proof-of-Useful-Work

Instead of having the world's most powerful network of assigned computers incrementing and recalculating one math problem, why not have a massive collective of computational power do some useful work? **Proof-of-useful-work** aims to repurpose the computational power of a Proof-of-Work cryptocurrency network to solve important problems. Historically, long-term public research projects produced great findings, encouraged by Proof-of-useful-work's attempts to repurpose mining compute power to aid through:

- Searching for large primes — Great Internet Mersenne Prime Search, which found the "largest prime number" to date ($2^{57885161} - 1$) twelve straight times.
- Finding aliens — SETI@home, which is the largest project to date, with over 5 million participants.
- Simulating proteins at the atomic level — Folding@home, which has the greatest computing capacity of any volunteer computing project, and has produced more than 118 scientific papers.
- Securing cryptography — distributed.net, which was the first successful public brute-force of a 64-bit cryptographic key.
- Generating predictive climate models
- Producing solar power (Solarcoin distributes coin to those who can produce solar power).

As good as Proof-of-useful-work sounds, design restrictions prohibit its consideration for cryptocurrencies and blockchain. Many distributed computing problems are unsuitable for Proof-of-Work, as a fixed amount of data will lead to trouble. For example, contributers to SETI@Home could run out of raw radio telescope data, leaving no problem to solve. Distributed computing problems lack an inexhaustible puzzle space, rendering them unfeasible. Second, potential solutions to these computing problems are not equally likely, so there is a lack of equiprobably solution space. Third, we must not rely on a central entity to delegate tasks, otherwise we fail to reach the requirement of decentralized and algorithmically generated problems. Thus, Proof-of-useful-work is unviable.

Proof-of-Storage — Permacoin

Instead of distributing work across a network, one idea involves distributing large files. In **Proof-of-Storage**, specifically the Permacoin implementation, a large public file in need of replication is split to be contained by several anonymous peers. An example of such a file would be experimental data from the Large Hadron Collider, several hundred petabytes of data. Such a file is stored in blocks, in a Merkle tree upon which the network agrees, then distributed to miners. Each stores a subset of these blocks, T, based off their public key to ensure that everyone agrees on who is responsible for what. They then continuously hash consensus information with a nonce to pick blocks in their stored subset and find a valid nonce such that its hash with all block headers is less than some target. Each server can query a given index, which returns a block header for the public data. One drawback to Proof-of-storage is that it is hard to find a large enough file to subsection to the network's nodes. If such a file is found, it would be difficult to change the block difficulty or to modify the file.

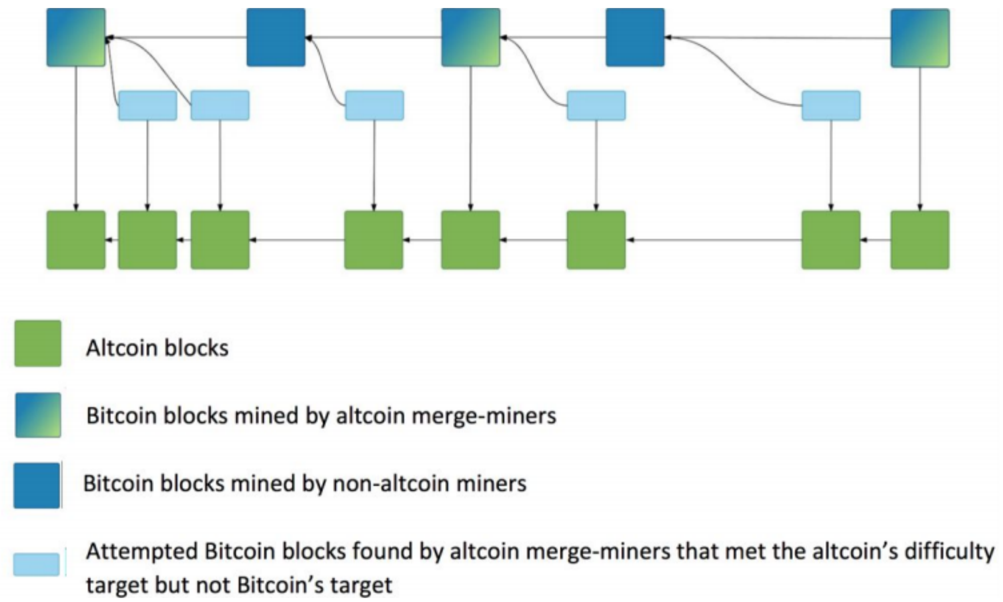
Merge Mining

Consider the potential vulnerabilities of a new altcoin in the current market. Lack of hash power in the altcoin's network means lack of security. An individual can easily amass enough hash power to attack the small network, since not many people are securing the network. A problem altcoins have to face is how to attract miners onto their network. Mining is exclusive by default, meaning that if a miner mines an altcoin, they cannot mine Bitcoin. This is a problem because there is more incentive to mine on Bitcoin and receive profit, rather than mine on a random altcoin that may fail. Miners have to trust the altcoin, and be willing to lose profit mining Bitcoin in order to mine the altcoin.

The problem of securing a new altcoin's network is especially difficult when considering phenomena such as "altcoin infanticide." Altcoins with their minimally secure networks are vulnerable to attacks from miners from larger coins (Bitcoin) with much more hash power than that of the entire altcoin's network. In 2012, Eligius mining pool attacked CoiledCoin, reversing multiple days' worth of transactions and mining a long chain of empty blocks, effectively rendering CoiledCoin useless. Mining pool operator Luke Jr. believed that CoiledCoin was a scam, and while some peers agreed, most did not agree to the altcoin infanticide.

To save altcoins from malicious attacks such as altcoin infanticide, **merge mining** was implemented. Merge mining involves creating blocks with transactions from both Bitcoin and an altcoin. Although merge mining poses no additional costs to miners, additional effort must be taken by the creators of the altcoin in order to design the altcoin in such a way as to allow merge mining on their blockchain. The blocks in the altcoin blockchain must contain information from Bitcoin block headers. The Bitcoin blockchain on the other hand stays the same.

If Bitcoin information is stored in the altcoin blockchain, there must also be a way to store altcoin information in the Bitcoin blockchain. The solution lies within Bitcoin's coinbase transaction, the transaction that awards the block reward. The coinbase transaction's *scriptSig* field is traditionally left empty, and it is here that merge miners can stick a Merkle root of the altcoin transactions. Other Bitcoin clients do not care about the scriptSig field, and will ignore it. Bitcoin miners could then reap both Bitcoin and altcoin rewards. This would result in no additional cost, since the miner could set up the mining puzzles such that the same nonce works for both the Bitcoin and altcoin block.


Altcoin blocks
Bitcoin blocks mined by altcoin merge-miners
Bitcoin blocks mined by non-altcoin miners
Attempted Bitcoin blocks found by altcoin merge-miners that met the altcoin's difficulty target but not Bitcoin's target

In the diagram we can observe some of the useful properties of merge mining. The blockchain on the top represents the Bitcoin blockchain, and the one on the bottom is that of the altcoin. Not all blocks in the Bitcoin blockchain are mined by merge miners, but the ones that are contain a Merkle root of the altcoin transactions (pointer to the altcoin blocks). Also, altcoins generally have a lower difficulty than that of Bitcoin, because they want to make coins more accessible in order to incentivize more miners to secure their network. The long blocks between the Bitcoin and altcoin blockchains represent blocks that meet the altcoin's difficulty target, but not Bitcoin's difficulty target. These contain information on both Bitcion and altcoin blockchains, but are only accepted on the altcoin blockchain.

Bonus: SHA-256

SHA-256 is Bitcion's most fundamental cryptographic hash function, and finds use in mining and also the creation of new addresses. The design for SHA-256 originates from the NSA (National Security Agency), and as one of the successors to SHA-1. Probably due to its connection with the NSA, SHA-256 has been the subject of a number of conspiracy theories which claim the existence of back doors that would potentially allow the NSA to break Bitcoin at will. There has been no proof thus far. Another concern with SHA-256 is Bitcoin's reliance and association with the cryptographic hash function. In the future if SHA-256 no longer provides the necessary security to uphold the Bitcoin network, die-hard Bitcoin developers would then have to explore alternatives.

SHA-256 works by maintaining 256 bits of state. These 256 bits of state are divided into 8 words of 32-bit state. Because the input for SHA-256 can be of any size, the first step in execution would be to pad the input with enough arbitrary data until it is a multiple of 512 bits long. The padded input is then split into messages 512 bits in length, which in turn split into even smaller messages which undergo a chain of secure bitwise tweaks and addition modulo 2^{32}. This is what gives the hash function the properties of first and second preimage resistance. SHA-256 performs these operations a total of 64 times in order to produce the final hash.

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **VOCAB WORD** — Definition.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 6

Game Theory & Network Attacks: How to Destroy Bitcoin

As with any other distributed network, the Bitcoin network is subject to a variety of attacks. In this note, we will take a closer look at potential attacks that could be used to take down the Bitcoin network. As we will find, destroying Bitcoin is actually fairly simple.

Mining Pools

Recall the concept of mining pools, which allow individual miners to combine, or 'pool,' their computational power. Mining pools are run by **pool managers** or **pool operators**. The pool manager conventionally takes a cut of mining rewards first, and the rest is distributed to miners, depending on offered hash power. Thus, instead of mining until finding a single block and profiting immensely, a miner can achieve a relatively stable profit by submitting **shares** to the pool.

In a large mining pool, an individual miner expects consistent profit in proportion to their own hash power, akin to a 100% commission-based salary. In contrast, mining alone implies a higher variance in mining rewards: you expect not to win the block reward the majority of the time, but celebrate the huge payout when you do.

Mining pools lower the barrier of entry for new miners, democratizing the mining scene. Instead of having to purchase an array of ASICs, individuals can opt to participate in a mining pool. Additionally, mining pools are easily upgraded during times of protocol change, since only the pool manager has to upgrade on behalf of the pool instead of each participating node needing to upgrade. Everyone else continues contributing hash power as before.

Mining pools, although enticing to the average miner, detriment Bitcoin's network health. Mining pools are examples of centralization, around one pool manager, and bring with them the security flaws of centralized systems. For example, the pool manager must be trusted to distribute block reward in a fair manner. Especially as mining pools grow larger, and represent the majority of total hash power, the Bitcoin network may not be as safe as we like to think.

Mining Pools — Example

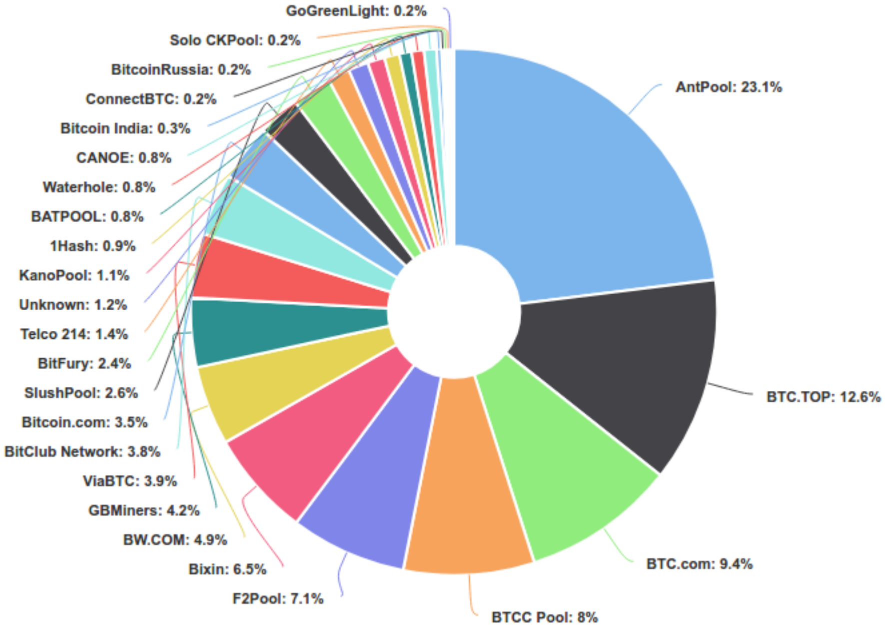
We will now compare estimated profit for a solo miner versus a miner in a mining pool. Suppose you want to start mining today. The best ASIC on the market, the Antminer S9, costs $2400, and has an average hash rate of 14 TH/s. Today's (July 21, 2017) total network hashrate is 6,478,893 TH/s. Buying an Antminer S9 would get you $\frac{14\ TH/s}{6{,}478{,}893\ TH/s} = 0.000216086\%$ of the network hashrate. The total amount of mining reward awarded every year is $1\ yr * \frac{12.5\ BTC}{10\ min} = 657,000\ BTC/yr$.

According to the percentage of network hashrate you own, you would expect to get an annual reward of: $0.000216086\% * 657,000\ BTC/yr = 1.42 BTC/yr$.

However as a solo miner, you can't assume that you will win block reward consistently, with low variance. A more accurate calculation would be to consider the expected number of blocks you will mine, not block reward, proportionate to your hash power. Given the percentage of total hash power you own, you can expect to mine one block every 462,779 blocks. This translates to *one block, 12.5 BTC, every 3214 days (8.8 years)*. This payout is way too infrequent. Now consider the case for mining pools. Assume the mining pool has 1/6th of the network hash rate. The pool would find every 6th block. We simply divide our expected annual reward, based on the proportion of hash power we own, with 8760 hours in a year to get the hourly rate. $1.42\ BTC/yr\ /\ (8760\ hr/yr) = 0.000162\ BTC/hr$.

At today's exchange rate, 1 BTC = 2771.92 USD. Solo mining would yield $34,649\ USD/8.8\ yr$, whereas mining with a large mining pool (assuming the pool has 1/6 of the network hashrate) yields a much more frequent payout of $0.45\ USD/hr$. For most, the frequent payout of large mining pools ensures them the maximum profit. This is especially true when one considers the fact that current-day mining hardware will probably be obsolete in 8.8 years anyways, due to increasing network hash power and overall network security. Paradoxically, as Bitcoin gets more secure, the more we need mining pools, whose dangerous centralization dangerously contrasts with Bitcoin's underlying concepts of decentralization.

Mining Pools — Hash Rate Distribution



Above is a pie diagram of the mining pool hash rate distribution from blockchain.info on July 25, 2017. In general, the Bitcoin community exhibits backlah against large mining pools that gain too much power. In 2014, a large mining pool called GHash.io actually exceeded the 51% threshold, causing controversy among the Bitcoin community. Miners recognized the economic incentives of joining mining pools, and GHash.io provided the most potentiall for profot. Fortunately, miners voluntarily left the dangerously powerful GHash.io to uphold the health of the network.

One of the most frightening aspects of mining pools is a phenomenon called **laundering hashes**, a process by which a single entity participates in multiple pools. Laundering hashes are very difficult to detect because it is difficult to detect where new blocks are coming from, and whether they are associated with mining pools. By extension, this raises the concern that the actual concentration of control over mining hardware is *unknown.*

Mining Pools — Shares

For mining pools to distribute the block reward, they must keep track of which miners are in their pool, as well as the proportion of computational power they contribute to the pool. To prove that they actually contribute to the pool, miners submit **shares**, or “near-valid” blocks, to mining pools. Producing shares implies that computational power has been expended to help the pool find the block reward. After a block reward is found, the mining pool operator distributes the reward proportionally to the number of shares submitted, which approximates the proportion of computational power expended.

Valid blocks are also considered shares. A miner who finds a valid block is not rewarded any extra coins from the block reward. This is because the valid block is based on the Merkle root given by the pool operator. The coinbase transaction goes to the pool operator, who redistributes the profit to the pool. In this way, a miner cannot just submit shares in a pool and keep the reward of the valid block for themselves.

Mining Pools — Basic Reward Schemes

There are two basic reward schemes that mining pools employ to reward miners. **Pay-per-share** is a reward scheme that pays out at every share submitted. Every time a miner submits a share, the pool will pay the miner for their share. Be default, the pay is proportional to the work done. Pay-per-share is beneficial for miners rather than the pool, since individual miners face no risk from reward variance. Instead, it is the pool that has to deal with reward variance, since they only profit when a block reward is found. One major problem of pay-per-share is that there is no incentive for miners to actually submit valid blocks, since miners are paid regardless.

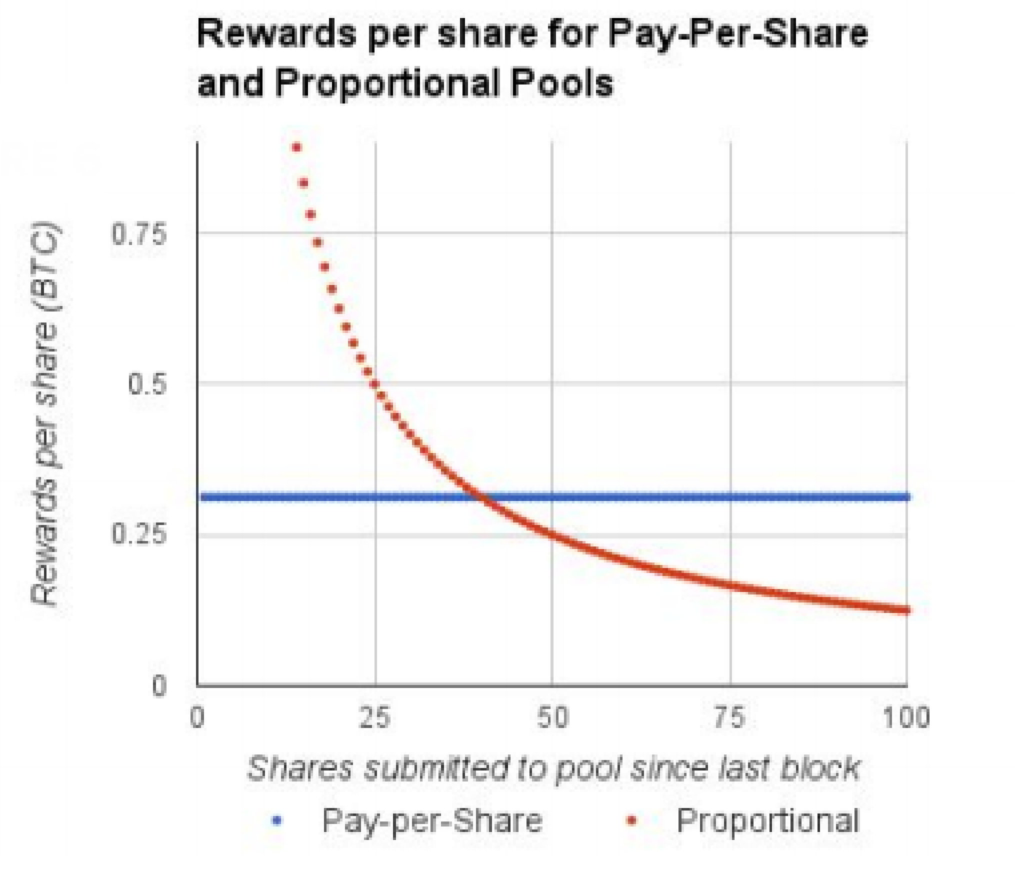
Proportional reward schemes are those that pay out to miners whenever a block is found, proportional to the work that individuals have submitted for the current block. Because individual miners only profit when the entire pool finds a valid block, proportional reward schemes are more beneficial to the pool. Individual miners bear some risk in variance proportional to the size of the pool. They are not paid if a valid block is not found. This is generally not a problem if the mining pool is sufficiently large, and block rewards are fairly consistent. For mining pool operators, proportional reward schemes provide lower risk, since they only have to pay individual miners when a reward is found. In this way, individual miners are incentivized to submit valid blocks, solving the fundamental problem of the pay-per-share reward scheme.

Pool Hopping

Because miners seek profit, the existence of multiple mining pool reward schemes gives rise to questionable behavior aimed to maximize profit. **Pool hopping** is a practice among some miners that involves switching between different mining pools to increase their total profit. This is done in observance of the facts that proportional mining pools pay larger amounts per share if a block is found quickly, and that pay-per-share mining pools pay out regardless, but often times not in as high of quantities as in proportional pools.

An example of a clever strategy that a miner might employ to maximize profits is:

- Mine in a proportional pool shortly after a block was found, while rewards are high since there have not been many shares submitted yet.
- Switch to a pay-per-share pool once the proportional pool is less profitable, since they always pay out.



In the chart above, we compare the rewards per share for pay-per-share and proportional pools. We take, for sake of an example, that the mining pool has 10% of the network hash rate, and that we can expect about 4 shares per miner for each valid block.

The practice of pool hopping is detrimental to the mining pool scene. Honest miners who stay loyal to one mining pool are cheated out of their money, since miners who pool hop submit shares only when it is profitable. Therefore, proportional pools are not feasible in practice. Designing a mining pool reward scheme with aligned incentives , for miners and the pool, that is not vulnerable to pool hopping remains an open problem.

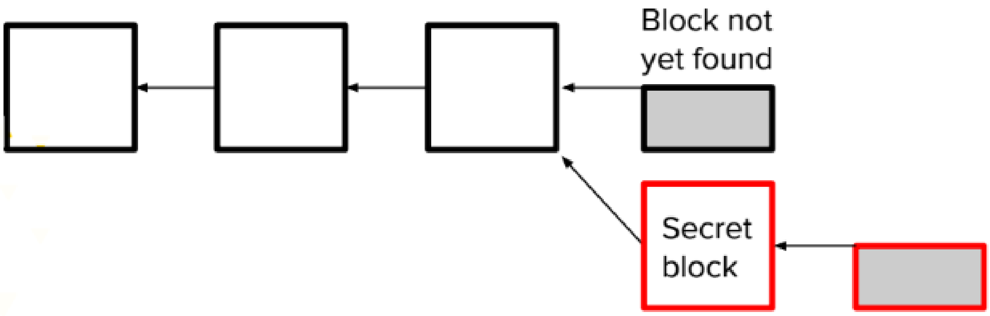
Pool Wars

Let's create a theoretical scenario to simplify some of the calculations we'll be doing in this section. Assume that you have 30% of the network hash rate, and that the block reward is 1 BTC. Owning 30% of the hash rate equates to winning 30% of the mining reward, so on average you would expect to get 0.3 BTC per block. Now, suppose you buy more mining equipment, worth 1% of the current network hash rate. With a standard mining strategy, adding 1% network hash rate would mean you now own 30.69% network hash rate, since $\frac{31}{101} = 30.69\%$. Your revenue gain from 1% hash rate added would be 0.0069 BTC.

Instead of deploying the standard mining strategy, say that you decide to **cannibalize pools**. You distribute your 1% equally among all other pools, and withhold valid blocks. You would still receive mining pool rewards from the shares you submit. The other pool owns 70% of the network hash rate still because you are not contributing in its efforts to find a valid block, you are simply diluting it. 70% network hash rate expects to earn 0.7 BTC per block. You own $\frac{1}{71}$ of the other pool, so your expected value from mining and submitting shares in this other pool would be $\frac{1}{71} * 0.7 = 0.0098\ BTC$. Therefore, we can conclude that since 0.0098 BTC ¿ 0.0069 BTC, it is more profitable to cannibalize pools than to mine honestly.

Selfish Mining (Block-Withholding)

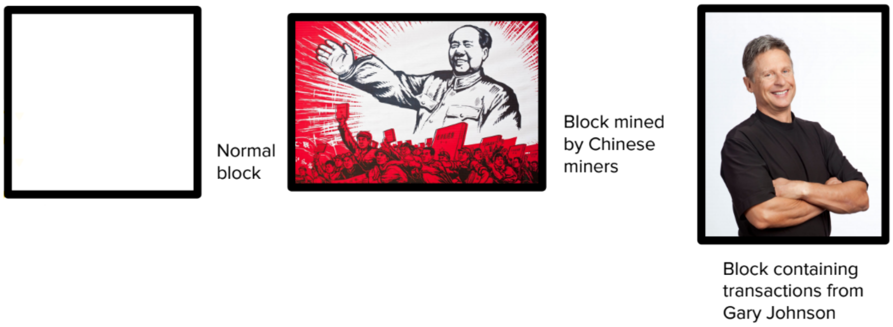
Instead of withholding a block from a mining pool, what happens when we withhold a block from the entire network? This is malicious mining strategy called **selfish mining**, or **block-withholding**. (The terminology can be a bit confusing, since pool cannibalizing is also a form of withholding a block, but in this scenario block-withholding is in the context of the entire network.) Suppose you are a miner and you have just found a block. Instead of announcing the block to the network and receiving a reward, you keep it a secret. You try to find two blocks in a row before the network finds the next one. If you succeed, once you announce this block, you can receive block rewards for both blocks, since you have the longer chain with two blocks that the rest of the network does not have. The image below shows a selfish miner (bottom) two blocks ahead of the rest of the network (top).



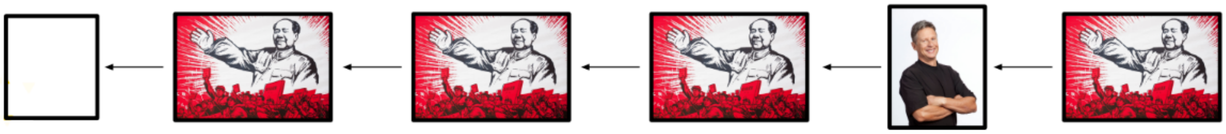
If however your plan fails, and the network finds their new block before you can find a second one, there begins a **race to propagate**. You and the rest of the network have competing blocks. It is mathematically proven that assuming you have 50% chance of winning the race and having your block accepted by the network, this malicious strategy of selfish mining is only profitable if you have more than 25% of the network hash rate. It is also true that if you have more than 33% of the network hash rate, you can lose the race to propagate every time, and your malicious strategy would still be more profitable. While no one individual probably owns this much of the network hash rate on their own, if a large mining pool (or several) banded together in hopes of increasing their own profits, they could potentially deploy the strategy of selfish mining and profit every time.

Blacklisting via Punitive Forking

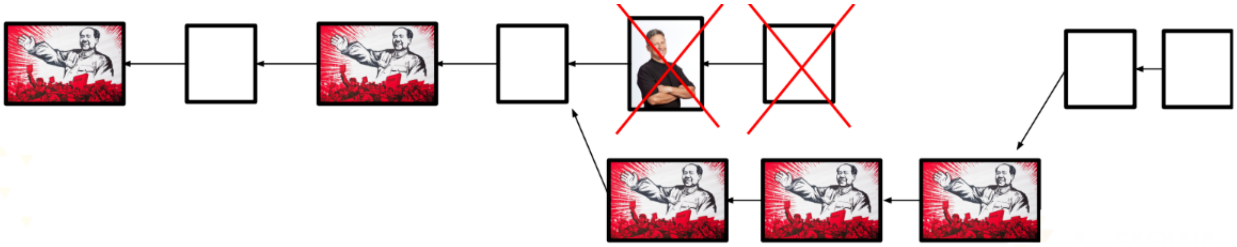
Say you are a government that has jurisdiction over many mining pools, say China. Your objective is to censor the Bitcoin addresses owned by certain people, say Gary Johnson, and to prevent them from spending any of their bitcoin. How do we censor Gary Johnson from the Bitcoin network? In the following sections, we will analyze some strategies that could be used to blacklist someone from the Bitcoin blockchain. We will use the following types of blocks in our illustrations.


Normal block
Block mined by Chinese miners
Block containing transactions from Gary Johnson

Your first strategy might be to tell your country's mining pools to not include any of Johnson's transactions, in a process known as **blacklisting**. This strategy will not work unless you are 100% of the network, and you mine all of the blocks, all the time. If you do not have complete control over the network, it is possible that other miners will eventually include Gary Johnson's transactions into a valid block, and propagate that before your country can. This strategy is not a practical way to censor Johnson from the network, and realistically can only cause delays and inconveniences.



You decide to change up your strategy. You are China after all, and you have more than 51% of the network hash rate, so you mandate that Chinese pools refuse to work on a chain containing transactions spending from Gary Johnson's Bitcoin address. You then announce your blocks to the world as usual. Since you have the majority of the network hash rate, you will always have the longer chain in the long run, and thus you can always invalidate Gary Johnson's transactions whenever you overtake the rest of the network's chain. In other words, if non-Chinese miners include a transaction from Johnson in a block, China will fork and create a longer proof-of-work chain. The non-Chinese block containing Johnson's transactions will be invalidated.



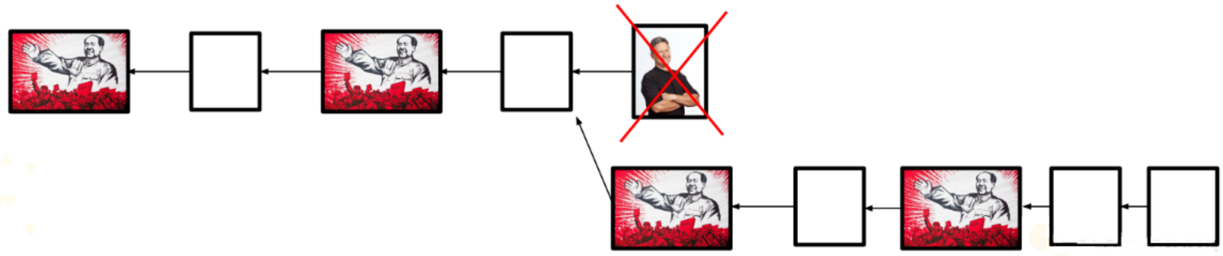
As a result of having their blocks constantly being invalidated, non-Chinese miners would eventually stop trying to include Johnson's transactions when mining blocks. Thus, we have shown how a 51% majority can prevent anyone from accessing their funds (blacklisting). This particular method of blacklisting is called **punitive forking**, since you can make a fork in the blockchain, and since you have majority hash rate, everyone else would have no choice but to join your chain, else waste their computing power on a chain that will never win.

Blacklisting via Feather Forking

One weakness of punitive forking is that it only works if you have a majority of the network hash power. There is a way around this with a strategy called **feather forking**. Following with our previous example scenario, you now change your strategy such that you announce that you will attempt to fork if you see a block from Gary Johnson, but will give up after a while. Assume you no longer have more than 51% of the network hash rate. For example, you might give up forking after a block with Johnson's transactions contains k transactions. You do not want to be attempting to fork forever, since you no longer have more than 51% hash rate.

If you have q proportion of mining power, with $0 < 1 < 1$, and you decide to give up forking after 1 confirmation on Johnson's block, your chances of successfully orphaning (invalidating) Johnson's block is q^2. This is because you would have to mine starting from one block before Johnson's block, and then overtake it by mining 2 additional blocks. If $q = 0.2$, then $q^2 = 4\%$ chance of orphaning a block. Our chances are not looking so great.

However, since we announce to the network that we will attempt to fork on Johnson's blocks, other miners would be aware that their Johnson blocks have q^2 chance of being orphaned. They must now decide whether or not to include Johnson's transactions in their blocks. Since miners are after profit, we look at their potential earnings: Including Johnson's transactions would yield profits of $(1-q^2)*BlockReward + Johnson's\_tx\_fee$. Meanwhile, not including Johnson's transaction at all would yield profit of just equal to the $BlockReward$. Looking at the scenario from a purely monetary point of view, unless Gary Johnson pays $q^2*BlockReward$ in fees for his transactions, other miners would seek higher profit and mine on your malicious chain. At current BTC value, Gary Johnson would have to pay $q^2*BlockReward = 4\%*12.5\ BTC = 0.5\ BTC = 1397\ USD\ minimum/transaction$. Yikes.



Network Attacks — Timejacking

Whereas blacklising and malicious mining pool schemes only affect mining profit schemes, it is also possible to create attacks that endanger the entire underlying Bitcoin network. To recap, Bitcoin nodes exchange monetary value through transactions over vast distances and time zones. It becomes a challenge to keep track of time, since it is necessary for each node to in sync with all the other nodes in the network. To address this problem, nodes maintain an **internal clock time**, which is the median clock time of all its peers. If a node's internal time differs by more than 70 minutes from the local system time, the node reverts its internal clock time to the system time. Additionally, as a precaution, nodes reject new blocks with timestamps that are more than 2 hours ahead of its internal clock.

As an attacker, Alice wants to put a Bob's node out of sync with the rest of the network in order to give herself more time to mine a secret chain and double spend on a target node. To do this, Alice can launch a sybil attack against Bob and every other node. Alice aims to set Bob's internal clock time to be 70 minutes behind the system clock time. For every other node, Alice wants to set the internal clocks 70 minutes ahead of the system clock time.

With these prerequisites fulfilled, Alice can now launch a **timejacking attack**. This involves mining a new block with a timestamp set 190 minutes ahead of the real time. It is important to set the timestamp to be 190 minutes ahead of the real time because blocks have a 120 minute timeframe during which they are considered valid. Every other node would accept the block, since it is within 120 minutes of their time $(190 - 70 = 120\ min.)$ Meanwhile, Bob rejects the block, since it is past the 120 minute validation bound. The result is that Alice has effectively partitioned the network, since Bob thinks every new block is invalid, while the rest of the network continues on.

Assuming that Alice can keep timejacking Bob, Alice has an indefinite amount of time to secretly mine blocks for a double spend attack. Alice can then broadcast these blocks to Bob, who thinks it is an alternate chain of history, and then accept them, not knowing that they are in fact invalid blocks. Bob would then send goods over while Alice would not have to spend any real bitcoin. However, for Alice to keep timejacking Bob in order to double spend, she would require a non-trivial amount of hash power to maintain the timejack. Even with a restricted time window however, Alice would still get 70 minutes, or 7 confirmations worth of time, to mine on her double spend chain — plenty of time for Bob to get tricked and accidentally send goods over in exchange for Alice's fake money.

DoS Attack — Malicious Miners

It turns out that malicious miners can also use timejacking to take down competing miners. They could effectively take the competing miners out of the network and thus increase their own effective hash power. This strategy also works for any type of **denial of service attack (DoS attack)**. Malicious miners that shut down competing miner's connection to the network can gain effective hash power and thus expect more profit. Taking this one step further, miners with access to a distributed Botnet have a competitive advantage over other normal miners.

Transaction Malleability

A consequence of using the cryptographic signature schemes that it does, Bitcoin is also vulnerable to attacks involving **transaction malleability**. Nodes relaying a fresh transaction can actually tweak certain fields in the transaction to make a version of the transaction with a different hash image, but the digital signature still verifies. For example, in ECDSA, the following signature pairs are equivalent:

$$(r, \mathbf{s}\ (modN))\ and\ (r, \mathbf{-s}\ (modN))$$

Both validate the same transaction data, but because they are different values, they have different hash images. Additionally, the scriptSig field can (sometimes) have extraneous script operations tacked on the end. These do not change the functionality of a transaction, but change their hash value.

Transaction malleability is a huge issue because in certain situations, transactions rely on a chain of previous transactions. These types of transactions are common in micro payments, and in the Lightning Network, which we will describe in a later note on scaling the Bitcoin network. Changing the hash image of a prior transactions in a chain of transactions invalidates every subsequent transaction.

Historically, there have been several incidents involving transaction malleability. The most famous is one attack on Mt. Gox: an attacker made withdrawals and Mt. Gox saw what was going on. The attacker used transaction malleability to change the transaction hash during the withdrawal, causing Mt. Gox to think that the transaction did not go through. Meanwhile, the bitcoin was actually sent to the attacker. In the end, Mt. Gox did not deduct the stolen amount from the attacker's account because they could not prove that a transaction had taken place without the right transaction hash. In the end, the attacker basically got free bitcoin from Mt. Gox.

SegWit (Segregated Witness) is a fix for Bitcoin that solves the problem of transaction malleability. It stores transaction signatures in a separate merkle tree. We will discuss more about SegWit in the scalability note.

Conclusion

In this section, we will make some broad generalizations about the Bitcoin network and attempt to arrive at a profound conclusion. First, we have to accept the notion that computational power requires electricity, which requires money. Computational power will reach equilibrium if miners break even or are making profit, since mining is such a competitive process. Thus, we can establish that:

- Lemma 1: Mining Reward = Mining Cost

If you are roughly breaking even with the capital you invest, there is little to no marginal cost to getting more hash rate. If you have really efficient hardware, there is negative marginal cost to getting more hash rate. All it takes to obtain 51% of the network hash rate is to get a lot of capital. This would mean that the ccost of acquiring 51% network hash power is zero or negative, which is less than the mining cost:

- Lemma 2: Cost of acquiring 51% ¡ Mining cost

After owning more than 51% of the network hash rate, you have a lot of power. You could crash the currency by doing a bunch of double spends, and then regain this value (and then some) by launching a gold finger attack. You also effective own 100% of the mining reward. You could just mine only on your own blocks, and since you will always produce the longest proof-of-work chain, all the block reward belongs to you. You can also prevent anyone else from mining using blacklisting via punitive forking. Owning a significant amount of hash rate would also affect the price of bitcoin. If you own 51% of the network hash rate, then that means 49% of blocks are orphaned. If you own 80% of the network hash rate on the other hand, only 20% of blocks would be orphaned. To the average Bitcoin user, this would not really be an issue, and they would be able to make transactions as usual. Because you get so much power from owning 51% of the network hash rate, then we can establish:

- Lemma 3: Value of 51% attack ¿ Mining Reward

Combining all three of these lemmas we have established, we can come to the conclusion that the value of a 51% attack is greater than the cost of acquiring the 51% hash rate in the first place. Game theory says that 51% attacking Bitcoin is profitable. Bitcoin's underlying assumption that no one will get 51% of the network hash rate is actually profitable, so what is stopping anyone from doing this?

Generalization of Vulnerabilities

Currently, there are a lot of orphaned blocks due to pool wars and other malicious mining strategies. A possible way to offset the costs caused by mining pools is to create insurance contracts for bitcoin stakeholders based on the number of orphaned blocks that are detected. Orphaned blocks let us know that pool wars are probably happening somewhere, and that a miner is being excluded. The concept of exclusion derives from the concept that in general, Bitcoin mining is zero sum. To increase your earnings past your fair share, another miner needs to be excluded, wasting their computational power.

Suppose miners join a collusion until 80% of the network hash rate is included, and then exclude the rest of the network, meaning that they do not mine on any blocks that are not created by the collusion. There would be no incentive not to join. If the attack succeeds, then everyone involved gets an increased reward. Additionally, the attack would not fail: the collusion could conduct the attack in such a way that it would not start excluding other blocks until the threshold of 80% is reached. With these points in mind, Game Theory dictates that there would be no incentive not to join. As a naive example, consider the current day. All it takes to conduct such an attack would be for three or more large mining pools to collude; they would then own more than 51% of the network hash rate. They could potentially ignore every 10th block of another pool in attempts to increase their own power in the network. Such an attack would be practically undetectable, since blocks are found every 10 minutes, and because it is hard to make any statistically significant, detectable changes. Such a strategy would be more profitable than honest mining strategies. Miners are primarily incentivized by profit, so the question is not whether or not this sort of attack has happened before, but: how many of these secret attacks are going on today?

Post-Block Reward Bitcoin

As a recap, the Bitcoin block reward halves every 4 years, and by the year 2140, all bitcoin to ever exist will have already been mined. To further comment on the viability of Bitcoin post-block reward, let us analyze the case of an average future Bitcoin user, who holds $100,000 in Bitcoin, and is willing to pay $1,000 in fees. To determine the Bitcoin network security, we have to look at mining incentives, since it is mining competition that creates new valid blocks. The amount of hash power that goes into Bitcoin is dependent on mining reward, and as the block reward diminishes to zero, money must move. Miners must be paid in transaction fees so that they can collect is as mining reward — incentive to uphold the health of the network. If mining reward is based solely on transaction fees in the absence of block reward, is mining still sustainable?

$$\frac{average\ fees\ paid}{average\ holdings} = \frac{cost\ of\ attacking}{market\ cap}$$

Average fees paid and average holdings are our previously assumed values of $1,000 and $100,000, respectively. The cost of attacking is the amount of security and mining power that is being invested in the network. Market cap is the market cap of Bitcoin when the maximum of 21 million BTC has been mined. From this proportion, we can deduce that an attacker only needs to pay 1% of the market cap of Bitcoin in order to attack. One potential solution to this frightening result would be to increase the velocity of money in Bitcoin. By increasing the speed at which transactions are confirmed and included, miners can add more transactions to their blocks to earn more of the transaction fees. To achieve this, changes to the Bitcoin protocol, such as adoption of the Lightning Network, to boost transaction velocity must be made.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 6

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **VOCAB WORD** — Definition.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 7

Ethereum and Smart Contracts: Enabling a Decentralized Future

In this note, we will go in depth into what Ethereum is, some technical details on its architecture, how the network and blockchain works, and how the EVM executes Ethereum smart contacts. We will then explore some use cases for Ethereum and see how the technology applies in the real world.

Looking at articles on Ethereum as well as Ethereum's own website landing page, we often come across a plethora of "buzzwords" primarily used in publicity campaigns. Decentralized apps, smart contracts, DAOs, Bitcoin 2.0, etc. are all confusing buzzwords used to hype up Ethereum. Our first task will be to demystify Ethereum and to analyze how it works at a high level. Then we will move on to discuss some of the various applications that have been built on Ethereum, and how the emergence of decentralized apps on Ethereum are shaking up markets everywhere.

Note: Unless otherwise specified, use of the word "blockchain" or "network" refers to the Ethereum blockchain or the Ethereum network.

What is Ethereum?

Looking at it from a very high level, **Ethereum** is a *decentralized* platform designed to run smart contracts. Ethereum is decentralized in the same way Bitcoin is decentralized: there is no single point of control or failure, and the network is essentially censorship resistant since the resources required to do so would be immense. Additionally, Ethereum is an **account-based blockchain**, which differs from Bitcoin's UTXO model. It is also useful to think of Ethereum as a **distributed state machine** in which blocks of transactions are equivalent to state transition functions, which contain information on how to transition between blocks/states.

Ethereum has a native asset called **ether**, which represents the basis of value in the Ethereum ecosystem. It is used to align the incentives of the various different types of nodes in the system: miners, full nodes, etc. Miners are rewarded in Ether for helping to secure the system once they find the proof-of-work and propagate the first valid block.

Ethereum vs. Bitcoin

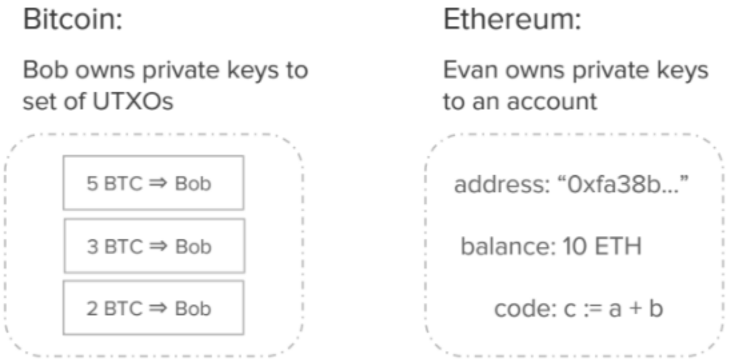
Since we already know quite a lot about how Bitcoin works from previous notes, it is useful to draw comparisons between Bitcoin and Ethereum. Ethereum markets itself primarily as a smart contract platform. It is complex and feature-rich (we'll discuss this in later sections) and most importantly, features a Turing complete scripting language. Bitcoin on the other hand is a decentralized asset system primarily used to trade value between users, It is simple and robust. This extends even to its underlying scripting language (Script, or often times just the Bitcoin Scripting Language), which is simple and stack-based, and not at all Turing complete. Why having a Turing complete scripting language is important is because it allows Ethereum developers to define arbitrary computations and programs, all which run on the blockchain. What is important to note is that both Bitcoin and Ethereum have both decentralized assets and scripting languages. They are just designed differently to serve differnt use cases.

The existence of ether is not actually a primary goal of Ethereum. It is used to align incentive within the network. From a game theroretical standpoint, it is important for users to have more incentive to act honestly than to cheat the system. In this way, because the ceators of Ethereum wanted miners to help secure the network via a Proof-of-Work consensus algorithm, they needed to incentivize miners with a block reward of ether. Ether is primarily traded between smart contracts. Another distinguishing factor between Ethereum and Bitcoin is that Ethereum plans to swap out its Proof-of-Work model for an alternative consensus algorithm called Proof-of-Stake, which rewards users for owning a particular percentage/stake of the entire network's capital. We'll go over alternative consensus protocols in a later note.

In terms of implementation details, Ethereum has a target block creation time of about 12 seconds, compared with Bitcoin's 10 minute block creation time. For its version of the Proof-of-Work protocol, it uses Ethash over SHA-256 for its primary cryptographic hash function. Ethash is currently ASIC resistant, but whether or not it stays so in the future is up to debbate. At the time of writing (July 31, 2017), 1 ETH → 200 USD, while 1 BTC → 2847 USD.

Accounts vs. UTXOs

Recall that a Bitcoin user's available balance is the sum of the unspent transaction outputs (UTXOs) for which they own the private keys to the output addresses. Instead of following Bitcoin's UTXO model, Ethereum keeps track of users' available balances by using **accounts**, each of which consist of an address, a balance, and some optional code. A user's available balance in Ethereum is simply the balance of all the accounts the user has private keys to.



The distinction between UTXO and account based models might be subtle for the common user, but it significantly affects the way developers write smart contracts.

Ethereum Account Types

There are two types of accounts in Ethereum. Both are fundamentally similar, except one contains code while the other generally does not. **Externally owned accounts (EOAs)** are usually owned by some external entity, for example a person or corporation. As do all accounts, they have an address and an ether balance. They can also send transactions to other accounts, whether they be other externally owned accounts or contract accounts.

Contract Accounts (Contracts) on the other hand not only have addresses and ether balances, but also have associated contract code. Code execution for contract code is triggered by transactions or messages (which work as function calls) received from other contracts or externally owned accounts. Contracts have persistent storage for their contract code, as their code, referred to as smart contracts, live and execute distributedly on the blockchain.

All Accounts == Network State

In Bitcoin, the state of the network at any point in time is defined by the entire set of UTXOs. In contrast, the state of the entire Ethereum network is defined by the state of all accounts. The entire Ethereum network agrees on the current balance, storage state, and contract code of every single account, and this is the network state that changes for every block in the Ethereum blockchain. It is useful to think of blocks as a state transition function. A block takes the previous network state and produces a new state based on the changes and transactions that have taken pace since the previous block. Every node processes blocks and then agrees upon a new network state based on the changes posed by the blocks. Accounts interact with the network, other accounts, other contracts, and contract state through the transactions that are contained within each block.

Accounts Rationale

One of the reasons Ethereum chose to use the account model rather than Bitcoin's UTXO model is that they wanted to save space. Instead of having to save every UTXO to determine one's balance, Ethereum just has to update each account's balance after every transaction. The account model is also more intuitive when writing a smart contract. It's much easier to make a program that transfers value between accounts with a balance, rather than having to constantly update a UTXO set to compute a user's available balance.

Smart Contracts

Ethereum is smart contract platform, but what exactly is a smart contract? Google says that a contract is "a written or spoken agreement...that is intended to be enforceable by law." For example, a futures contract is an agreement to sell a particular asset at a predetermined price at a specified time in the future. If a contract goes well, assets exchange hands as specified, and all goes well. If not, parties associated with the contract could potentially sue and go to court, which would decide the lawful outcome of the situation. Similarly then, we can define smart contracts as "code that facilitates, verifies, or enforces the negotiation or execution of a digital contract." In order for smart contracts to function properly, they must be executed by a trusted entity. Instead of a trusted third party, trust can be reached by using a distributed and decentralized network. Previously we mentioned that the blockchain is a structure that establishes trust, so the logical conclusion is to take smart contracts and encode them onto the blockchain – and that's exactly what Ethereum does. Smart contracts on Ethereum run in the context of the global state of the network, established by blocks, and everyone on the Ethereum network agrees on what these smart contracts do.

What is a Smart Contract?

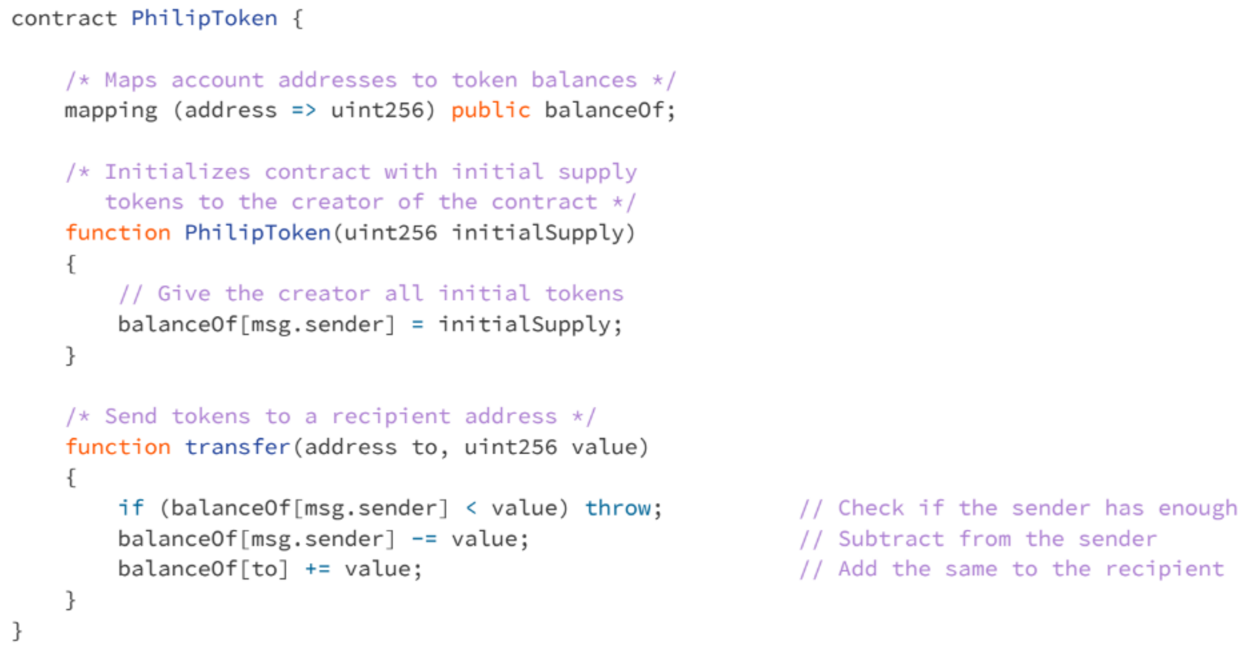
In Ethereum, smart contracts are executed by the network itself. Every single node in the network (full nodes, miners, etc.) execute the code in each contract when a function is run. This happens in lockstep across all nodes at the same time, meaning that every node is executing the same step at the same time, maintaining a parallel redundancy. This ensures that no one can violate the contract, because everyone in the network agrees on what the contract should do. Network consensus removes the need for a trusted third party, and this removes any worries about maliciously altered contract execution. Like in the case of Bitcoin, we can assume that Proof-of-Work solves all these problems, and for someone to successfully violate a contract, they must first subvert the entire network.

One way to think of smart contracts is to imagine them as autonomous agents that live inside of the Ethereum network. The way we trigger them and to have them react to the external world is to "poke" them with transactions, which call certain functions defined in their smart contract code. Smart contracts have direct control over their own internal ether balance, internal contract state, and also permanent storage, so these could be potentially changed based on how a smart contract is designed to react to transactions.

Ethereum smart contracts generally serve four purposes. Firstly, contracts can be used to *store and maintain data*. This could be useful in representing something useful to users or other contracts, and could for example be used to build another token currency within Ethereum, keep track of an organization's membership, or maintain a domain name registrar. Other contracts *manage contract or relationship between untrusting users*, and these are commonly used when parties do not necessarily trust each other, but want the same code to be executed by all parties. Such contracts find use in financial contracts, escrow, and insurance, where it is easy for a party to attempt to cheat. In which case, a single, common source of truth – the Ethereum network – executes the code and helps maintain integrity. Some contracts exist to *provide functions to other contracts*, and do not talk directly with normal users. Examples of such contracts are those that serve as software libraries. Lastly, some contracts offer *complex authentication* functionality. Contracts could implement M-of-N multisignature access, where M number of nodes of a set of N are needed to make decisions – think multisig wallets and board members coming to consensus. While we identified only four general purposes, there are many contracts in the Ethereum network that are a combination of the four classes mentioned.

Minimum Viable Token

Next, let's look at a basic sketch of a smart contract that defines the functionality of a new token currency, called PhilipToken. As with most other smart contracts, PhilipToken is written in an Ethereum specific language called Solidity, and compiles down to EVM (Ethereum Virtual Machine) bytecode.



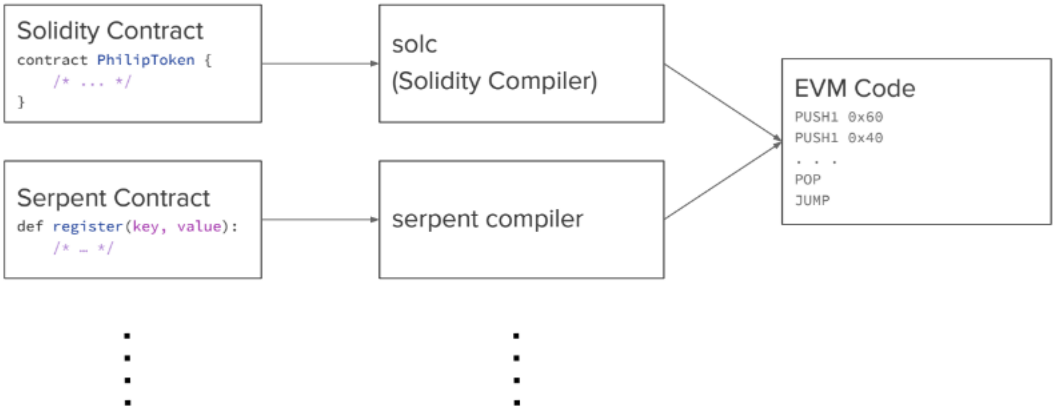
```
contract PhilipToken {

    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply
       tokens to the creator of the contract */
    function PhilipToken(uint256 initialSupply)
    {
        // Give the creator all initial tokens
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send tokens to a recipient address */
    function transfer(address to, uint256 value)
    {
        if (balanceOf[msg.sender] < value) throw;          // Check if the sender has enough
        balanceOf[msg.sender] -= value;                    // Subtract from the sender
        balanceOf[to] += value;                            // Add the same to the recipient
    }
}
```

In PhilipToken, we can instantiate the contract in such a way as to give all initial tokens to the creator. This seeds PhilipToken and defines the total amount of PhilipToken that is available. PhilipToken contains a mapping from address to token amount, and this is useful when keeping track of who owns how much token. After initializing a new PhilipToken instance, the creator can then transfer PhilipToken to other addresses. We can see in the bottommost function that if the sender has enough token, the sender's balance decreases by the transfer amount, and the recipient's balance increases by the same number. If the sender does not have sufficient funds, then an error is thrown.

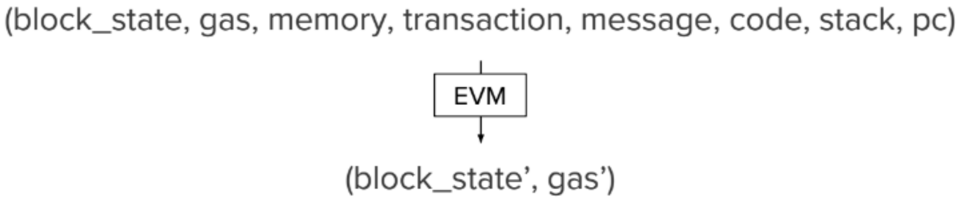
While this example is very basic, it outlines the potential of Ethereum as a platform. With a few lines of code, we have defined a new token currency that can be traded between any Ethereum users. Because the network is decentralized and distributed, the currency is secure, and thus immune to censorship or other attacks by governments.

Ethereum Virtual Machine

Contract code on Ethereum is run on the **EVM (Ethereum Virtual Machine)**. The contract code we saw in the previous section was written in a language called Solidity, one of many languages developers can choose to write Ethereum smart contracts in (other languages are Serpent, LLL, Viper, etc). Contract code written in these higher level programming languages compiles down to EVM bytecode that is executed on every node. EVM bytecode is a low-level, stack-based bytecode language similar to that of the JVM (Java Virtual Machine.) Every Ethereum node runs the EVM on all of the transactions in a block as part of its verification procedure, to update their internal state to match the new network state. The diagram below illustrates how contract code compiles down to EVM bytecode that is executed on all nodes in the network.



The EVM is essentially the underlying state transition mechanism for each of the nodes in the Ethereum network. We start off with the current block state, the gas required for code execution, the memory of the contract, the transaction that's calling the contract, message metadata, the code of the contract, stack of the contract, and the program counter. All this data is fed to the EVM, which calculates a new block state with all the updated account info on balances and long-term storage, and however much gas remains from the execution.


(block_state, gas, memory, transaction, message, code, stack, pc)
EVM
(block_state', gas')

EVM Design Goals

Some of the funamental design goals of the EVM are simplicity, space efficiency, determinism, specialization, and security. The EVM code was designed to be as simple as possible, to minimize the number of op-codes available by making them as low-level as possible. By making the set of op-codes smaller, we can minimize the amount of space required to represent the op-codes in bytecode, so that gives us a bonus in space efficiency. Determinism ensures that every node executes EVM code in the same exact way, regardless of the underlying architecture of the node. The same input state should always yield the same output state. The EVM should also be specialized, meaning that it should be ale to easily handle common operations such as those used in the underlying cryptography of Ethereum. The EVM must handle 20-byte addresses and custom cryptography with 32-byte values, modular arithmetic, and block and transaction reading with speed and accuracy. Security ensures that there is no way to overflow the EVM and take over a node, for example. Gas cost makes the EVM realistically non-exploitable. We'll talk about how this happens in the next section.

EVM Gas and Fees

One immediate concern with the design of Ethereum smart contracts is that if they are executed on all nodes, what if someone writes a contract that executes an infinite loop? Assuming someone writes a transaction to this contract, all nodes would essentially be stuck executing the same loop forever, halting the network. This would essentially be a denial of service attack, and would immediately cripple the network. By the *halting problem* of compatibility theory, it is impossible to determine ahead of time whether a written contract will ever terminate. How do we solve this network-breaking problem?

Ethereum's solution is to implement the concept of **gas**, which every contract requires in order to "fuel" contract execution. Every EVM op-code requires some gas in order to execute. In building a new transaction to a contract, one must specify in the transaction the *startgas*, or the maximum quantity of gas they are willing to consume in the transaction. *Gas price* specifies the fee in ether the transaction is willing to pay per unit gas – in other words, the gas to ether exchange rate.

At the start of a new transaction, the user calculates $startgas * gasprice$, and this amount in ether is subtracted from the user's account balance. If the contract successfully executes, any remaining gas is refunded to the user's balance. If the contract execution runs out of gas before it finishes, then the execution reverts. Also, the initial amount of ether the user put in, $startgas * gasprice$, is not refunded. Therefore, it is important to always overestimate the amount of gas required to execute a contract.

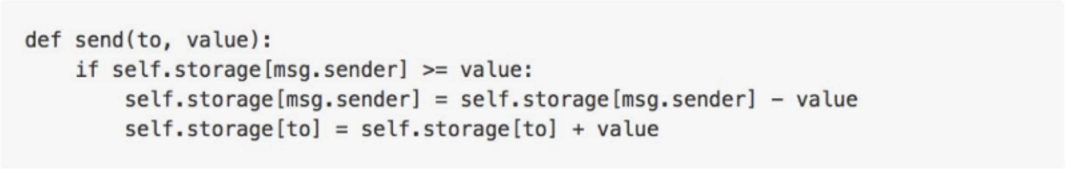
Looking back at the infinite loop problem, we can see that it is technically possible to write an infinite loop and launch a denial of service attack. The attacker would just have to pay enough ether to fund the attack, and this amount is so large that it realistically is not a threat to the Ethereum network. You can think of purchasing gas as purchasing distriuted, trustless computational power. Like how attackers in Bitcoin would have to use a lot of capital to get enough hardware to subvert the network, attackers in Ethereum must obtain enough Ether to use as gas to fuel their infinite loop denial of service attack.

Ethereum Conclusions

One important idea to grasp about Ethereum is that it is not designed to prioritize efficiency of computation. In fact, when compared with the average computer, the Ethereum virtual machine is terribly slow. Ethereum is optimized for distributed and trustless execution. All transactions are run across all nodes in the network, making Ethereum redundantly parallel. This in turn is an efficient way to reach consensus on the system state without needing to trust a third party. Furthermore, because contract executions are redundantly replicated across nodes, computation is expensive. The cost associated with executing a contract creates an incentive not to use the Ethereum blockchain for computation that can be done off chain.

Smart Assets

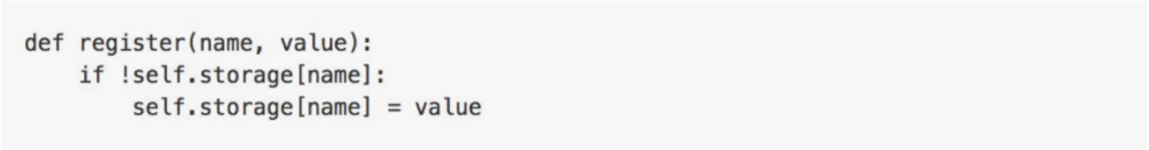
Now we'll dive into some of the use cases of smart contracts in Ethereum. With our PhilipToken, we have seen how easy it is to implement a new token system in Ethereum. It is essentially modeled after a database with a single operation: to ensure that Alice has enough money to pay Bob. If so, subtract that amount of token from Alice and give it to Bob. All this can be done in a few lines of code, as can be seen in PhilipToken, and also in the Ethereum whitepaper:



```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

Public Registry/Pubic Database

Smart assets contain a database to keep track of how much token each address has. We can use the same idea to build a public database that can be used as a domain name registry for example. We start to build this DNS system by mapping domain names to their corresponding IP addresses, through which they are hosted. Having this database on the Ethereum blockchain makes the data stored in it immutable, since everyone has to come to consensus about it, and could easy cross check the mappings. Such a system would be easy to implement on Ethereum too, as shown in the Ethereum whitepaper:



```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

Crowdfunding and Incentivization

Another use case for Ethereum is for crowdfunding. "Ether-on-a-stick" is a smart contract that allows Ethereum users to put a bounty on the completion of arbitrary tasks. Contributors pool money into a smart contract that pays out to a specified recipient if and only if contributors vote that the task was indeed complete. An example use case could be if a company is polluting a local river, and nearby residents are bearing a negative externality. If the local government is slow and unresponsive, and the residents are willing to pay, the residents could pool money together to incentivize the company to clean up the river. If the company cleans up the river, the contributors could come to consensus over whether the river was cleaned up, and the smart contract would then pay out to the company. "Ether-on-a-Stick" implements a dominant assurant contract and solves the free rider problem.

(Note: "Ether-on-a-Stick" was a hackathon project of Blockchain at Berkeley members Max and Philip)

Smart Energy Grids

Imagine that a neighborhood has a smart energy grid. There might be some imbalances in energy (heat, electricity, etc.) across the community. House A might have excess heat, while House C may be too cold. House A could sell its excess energy to House C, cancelling out the energy difference. In other words, there would have to be an energy market. The energy market would save both parties, House A and House C in this case, money, and would make the entire community as a whole more power efficient.

The problem with such a system is that there is no existing infrastructure that supports the direction of electricity sales. Energy infrastructure currently flows unidirectionally, from the utility companies to individual houses. Utility companies generally do not want to build smart grid infrastructures since this would entail less revenue for them, since energy sales between households would cut into their profits. The government also has little to no incentive to help build these smart grids. Only the household peers would benefit from the smart grid infrastructure.

The solution would be to set up an Ethereum smart contract for financial commitments. The smart contract could help coordinate the development of the smart grid in a neighborhood without individual houses having to trust one another. A contract would read something like this: "I, House A, commit $10,000 to building this infrastructure. If House B and House C also commit the same amount, the total amount will go to this contractor to begin construction on our smart grid, and the total amount can only go to this specific contractor." The Ethereum blockchain thus serves as a coordination layer for the community's decision. Households fall back onto the blockchain when coordination cannot be found via a central entity or government. What was originally a social problem can now be solved technologically with the Ethereum blockchain.

Decentralized Prediction Markets

The idea of decentralized prediction markets draws on the wisdom of crowd-sourced knowledge in an attempt to predict or forecast the future. Essentially, these would work by first allowing market makers to create an event. For example, they could make one for "Who will win the 2020 US Presidential election?" Events for decentralized prediction markets must be public and easily verifiable, with a set due date. This is because anyone must be able to verify the result, and also we need to know when to stop taking predictions. Participants in the decentralized prediction market would then buy shares of Trump or Zuckerberg, and pay a small fee. On election day, random oracles on the network would vote on who won, essentially bringing in external data from the real world into the blockchain. Oracles who voted with the majority collect the fee that was paid by the participants. Otherwise, the oracles are penalized, and could potentially lose their status as an oracle. Shareholders who voted correctly would be able to cash out on their bet after election day.

The share price for each market accurately represents the best predicted probability of an event occurring. Say that the shares for Trump and Zuckerberg are currently trading at 0.50 USD. This roughly equates to Trump and Zuckerberg having an equal probability of winning the 2020 election. However, if you know somehow through secret insider information that Zuckerberg's chance of winning is actually 70%, you could buy Zuckerberg shares until the share price goes up to 0.70 USD. Therefore, you have acquired shares that were worth 0.70 USD at the price of 0.50 USD. Based on the expected profit then, you have made money.

The powerful potential behind decentralized prediction markets is that it allows people to bet on not only political events, but practically anything else as well. It's a cost efficient way to buy information on a future event. Instead of hiring pundits and experts, you could create a market for your event and let the wisdom of the crowd generate a prediction for you. For example, you could set up a decentralized prediction market for determining if a new movie will flop or not. Users betting for and against this event create liquidity for that particular prediction. Hollywood insiders who know whether the movie will flop or not would then be incentivized to vote on whichever side of the prediction is beneficial to them. Then, you have essentially bought this insider information from them.

Decentralized prediction markets also find use cases in hedging and insurance. For example, whenever you buy fire insurance, you are betting that your house will burn down, otherwise you lose the funds you invested into the fire insurance. You could create a market asking the network to predict whether your house will burn down, and vote yes, because you're betting that your house will burn down. Insurance companies could then take sides on this prediction, depending on their expert analysis and judgment. If they predict that your house will not burn down, and it does, then you receive all the money that was pooled in. Of course there would be moral concerns with such a prediction market, but as an example, this shows the power of decentralized prediction markets. Taking the example a step further, it is even possible to implement an entire insurance liquidity pool onto a smart contract, avoiding the costs of offices, employees, etc.

Augur is one of the biggest decentralized prediction markets currently in operation. The way they secure their codebase uniquely leverages their product to detect bugs. First, they set up a security bug bounty on their decentralized prediction market. "Will someone be able to steal the money in this prediction market?" Augur then bets heavily against this prediction to create a financial incentive for someone to find a vulnerability. If someone successfully finds a vulnerability, they would buy affirmative shares to reflect their belief, and then perform their hack. Once the hack is detected, the individual would profit.

Another way to use a decentralized prediction market is to signal – essentially the economic term for "put your money where your mouth is." You can demonstrate your commitment to something by showing you will take a large financial loss if you miss your commitment. For example, say you are the leader of a huge Kickstarter campaign, and investors are worried that you will delay the launch date. You could then create a vote for "Will my Kickstarter campaign launch on time?" and bet heavily that you will in fact launch your product on time. If you lose, then you take a huge financial loss. However, this shows that you are confident that you will carry through with your commitment, since you are betting so much money and willingly exposing yourself to so much financial risk.

There are many benefits of creating decentralized prediction markets – the first being that there are no restrictions on market creation. We previously mentioned that prediction markets could be created for not only politics, but for a variety of other subjects as well, opening the possibility of hosting such functionality as the advanced examples we discussed. Arbitrary markets allow for much wider range of applicability. The caveat however is that we need a defense mechanism against unethical or illegal markets. Augur solves this problem by having oracles vote on whether prediction markets are unethical or undecidable. This prevents assassination markets or other markets with moral danger from popping up. Another benefit of decentralized prediction markets is that they create a shared liquidity pool. There's no reason why the same market should exit in multiple countries. Decentralized prediction markets could be used as a back-end service to take votes for variety of different applications. Other benefits are that decentralized prediction markets are censorship-resistant, because of properties of the Ethereum's distributed network, and also allow for automatic, trustless payments through the smart contract code.

Decentralized IOT

Ethereum's support for general smart contracts lends itself nicely to IoT (Internet of Things) because now we can automate the communication of various IoT devices on a distributed network. Filament is a company at the forefront of designing the "Blockchain-based decentralized Internet of Things." They build IoT sensors intended for industrial applications that are connected to one another via an "ad hoc mesh network." Sensor units with a 10 mile range, and include a battery that lasts for years. No internet connection is required, as sensors are connected using a mesh network. One of the use cases of Filament's solution is for the agriculture industry. Units could be deployed across vast farmlands and could be used to transfer important information in a secure way.

An interesting note is that blockchain is only one aspect in Filament's entire technologica stack. Blockchain is only used to keep track of name-address bindings for each individual sensor. This is then used to discover new devices on the network and to verify the authenticity of certain bindings. The name-address bindings are incorruptible because of the blockchain's property of immutability. Contrast this with a central database that could be maliciously altered with a single point of failure. Thus, Filament is a great application of decentralized tech especially because of its emphasis on resilience and dependability. Data for the bindings is duplicated across on all connected sensor units, increasing data redundancy, which entails increased durability.

After Hours Trading

24 hour stock trading would be convenient, and given that our world is highly digitized, why does it not exist? After hours trading has not taken off because it faces a problem in liquidity. There is not enough liquidity, so prices are more volatile and destabilizing. The government sees this as dangerous, and passes regulation that restricts trading to only using limit order. This is only sustainable at larger brokerages where there is enough stock to sustain any kind of market. It's nice to have convenience, but it's more important to have a better price discovery in the market.

A solution to enable after hours trading might be to create a blockchain-based asset exchange with legal bindings. Brokerages can let users "withdraw" their stock into stock tokens on a blockchain. The stock is then subtracted from the user's account, and the brokerage holds it in the meantime.

The brokerage then writes a legal contract saying that: “Anyone who proves that they own a stock token is eligible to redeem it for a stock I am currently holding.” These legal contracts make it such that tokens on the blockchain are backed by real stocks. Now you can create a trustless blockchain-based stock exchange that allows users to trade stock tokens, which can be redeemed for actual stock via the legal contracts written by the brokerages.

The result is that after hourus trading is enabled with very little cost. Anyone on the Ethereum network can now use their ether to buy stock tokens and trade after hours: 24/7/365. Multiple brokerages can participate and create a shared liquidity pool. In addition, these brokerages don’t need to trust each other, as the blockchain ensures that all the rules of asset management, deduplication, and stock trading in general been encoded in a smart contract, are being carefully followed. Anti money laundering is enforced when traders redeem their tokens. All trading history is transparently logged on the blockchain, so malicious traders can be caught with evidence.

Decentralized Autonomous Organizations

The last Ethereum use-case we are going to cover is something called a **decentralized autonomous organization (DAO)**, which is an organization governed entirely by the code within smart contracts. DAOs create and vote on proposals based on their design, and could even theoretically replicate entire businesses on the blockchain. After all, functionality such as decision making and making payments is easy to create on Ethereum. Such businesses would have uncertain legal status however. The general idea of DAOs can be summarized by the phrase “code is law” – that is, trust in math and code rather than in governments and institutions.

Some issues with DAOs are that they are hard to edit once they are deployed. Smart contracts are immutable after all. A possible solution would be to implement a Child DAO, but historically this has not been secure, especially when considering this was the cause of the crash of TheDAO. TheDAO was the largest crowdfunded project in history, which at the time raised more than $150 million in Ether. It was hacked in June 2016, when more than $60 million worth of Ether was stolen (10% of the Ethereum market cap.)

The use of a DAO can be seen in Dash, a privacy-centric cryptocurrency, where a certain percentage of the mining reward goes to a set of “master nodes” which decide on changes to be made. Dash thus creates a self-sustaining, decentralized ecosystem. Some other cryptocurrencies have something called a “founder’s reward” which allocates some award to a specific group of people, presumably the founders and creators of the currency, making the entire system centralized. Dash refuses this temptation in an attempt to make a more pure decentralized, privacy-centric cryptocurrency solution.

Limitations on Blockchain Tech

One of the main limitations of blockchain technology is that there is *no trustless way to access outside data*, creating the effect of blockchains existing within silos, or having to take the risk of trusting others. Blockchains often times must rely on **oracles** to provide information from outside the blockchain. We saw an example of this in the decentralized prediction markets section.

The problem is that we must find a way to trust these oracles. A potential solution is **proven execution**, which involves an oracle proving that an HTTP request for useful data existed at a certain period of time, and that they did not manipulate it before sharing it with the blockchain. Tools like Oraclize offer proven execution implementations for communication between Ethereum smart contracts and a variety of Web APIs. Oraclize uses TLSnotary, which is a modification of the TLS (transport layer security) protocol to provide cryptographic proof of receiving an HTTPS page. Another potential solution to the problem of getting data off-chain onto the blockchain could be to create an **oracle network**, which is what Augur does in its decentralized prediction markets. A drawback however is that you would have to build a consensus protocol into the oracle network in order to ensure security, and this would essentially mean you would have a consensus protocol (for the oracle network) built on top of another consensus protocol (that of Ethereum). There are a number of edge cases that arise from consensus protocol layering that are difficult to test.

Another limitation of blockchain technology is that there is *no way to enforce on-chain payments*. It's difficult to impllement financial products such as loans and bonds because they would require money be held on the blockchain to ensure payment. For example, you cannot issue loans via contract and trustlessly have the sum returned to you, since trust is fundamental to borrowing money. Intuitively, with loans, we usually pay an interest because there default risk, or a risk that someone will not be able to pay back their loans. A compromise involving the blockchain that we can implement is a decentralized reputation system, or a credit rating system. This would give some level of trust to the nodes with higher reputation/credit.

The final limitation of blockchain technology we will mention is that *contracts cannot manipulate confidential data*. If you create a smart contract to handle confidential data, having it on the blockchain means that everyone else on the network potentially has access to your data. You could encrypt the data before putting it onto the blockchain, but you would have to decrypt the data locally. Manipulating data directly on the blockchain after encryption would be incredibly difficult, as the only solution as of now would be to use **homomorphic encryption**, but such technology is still being designed.

Essential Properties of Killer Blockchain Apps

With so many use cases of Ethereum and blockchain technology, it is often difficult to identify well deigned applications from ones that are not, especially with all the hype going around. For every application, it is helpful to ask yourself: Why is using a blockchain better than using a centralized database? Understanding why blockchain is so good at what it does, and how sticking a blockchain into any ordinary application could make them worse, is a good exercise in identifying killer blockchain applications. The following is a list of some essential features and properties where blockchains excel over the standard centralized database.

Blockchains are really good at creating trustless environments that need consensus or coordination. It's by the design of blockchains and their underlying consensus algorithms that enable them to be decentralized, distributed, and trustless. Think of Ether-on-a-Stick, in which community members can come to consensus over whether a company has cleaned up a nearby polluted river, or smart grids, which have to come to consensus over the state of the current energy market.

Blockchains also allow for disintermediation and censorship-resistance. Cryptocurrencies allow for truly peer-to-peer payments through the blockchain, that bypass the need for a central authority managing and tracking where all the money flows. Blockchains' property of immutability means that it's costly for anyone to try to rewrite the block history, and this property naturally extends to encompass censorship-resistance, as we saw in the last note in the section on censorship attacks.

Data collected and managed by people can often be faulty, and it might be that markets can handle data with a higher degree of accuracy. Writing a smart contract with a set of rules on how to handle data means that the data is "correct by design," because the blockchain guarantees that any rules defined by the smart contract will be enforced. Data generated can thus be traceable, and also have a high degree of authenticity because it was agreed upon by the network. The only caveat is that data entering the blockchain must also be correct – a problem for oracle networks or similar solutions to handle, and not necessarily a shortcoming of the blockchain infrastructure.

The combination of money and code gives rise to the concept of easily programmable money. (How surprising!) Since blockchains are traditionally good at handling cryptocurrencies, and that smart contracts allow us to programatically deal with money, it is easy to see how automated payments could be built. This application of blockchain technology would find use in machine to machine payments on the Internet of Things. IBM ADEPT is a project by IBM and Samsung that attempts to create a blockchain-powered Internet of Things. Programmable money is also particularly useful for micropayments. For example, when the Lightning Network – or another equally fast extension to the blockchain – is in place, automated micropayments with tiny transaction fees could be implemented. Currently, micropayments are not realistic because miners prioritize transactions with larger transaction fees.

As we saw with Augur and also Ether-on-a-Stick, blockchains also give us ways of creating incentives. People and organizations are after mainly profit as one of their primary incentive, and blockchains conveniently feature value in the form of cryptocurrencies. Cryptocurrencies could be used to incentivize others to provide information or truth, or to take a particular action. Additionally, smart contracts give us away of creating robust and secure management systems, as we see with a majority of smart contracts that have internal rules and code that the network confirms. This however often breaks down at the interface with the real world, as that is when you have to rely on trust and not the guarantees of the blockchain. Building on smart contract rules and management, blockchains can also create new governance models. Although this is a relatively new field, examples in decentralized autonomous organizations (DAOs) have proven that this is effective, at least on-chain.

Conclusion

Contrast all the perks of blockchain technology with the perks of having a centralized database and you realize that there are use cases for both of them. It's not realistic to slap a blockchain onto an existing application and instantly have that app upgraded. Centralized databases are still better than blockchains at deep integration and creating a great and cohesive user experience. Facebook owns all of our data, and because they have all this data on their own servers, they can run machine learning tools on our data to tailor what we see on our Facebook news-feeds to what we usually generally interact with. This isn't practical on a blockchain solution because blockchains aren't efficient enough. (In fact, blockchains are probably the world's worst database

– it's only selling point is that it is decentralized and distributed, and allows for this handy thing called consensus without the need for trust.) Centralized control also allows for clearly defined read/write permissions over data. Blockchains being distributed means that everyone has access to all the data. Finally, not everything can be done on a blockchain (yet), and often times problems that were unaccounted for during project design need human level interaction to solve. One of the reasons why we're a long way from a "decentralized Airbnb" is that potential disputes between users are hard to predict and write into logic onto smart contracts. Instead, it is more efficient to have actual people manage this unpredictable human level complexity on a case-by-case basis.

The conclusion is that while blockchain technology and distributed tech as a whole is poised to lead to a new world of decentralization, it is important to consider the properties that make distributed solutions so great in the first place. Blockchains are great, but not applicable to everything at their current state.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 7

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **VOCAB WORD** — Definition.

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 8

Alternative Consensus and Enterprise Blockchain

At the most fundamental level, all public blockchains rely on efficient and secure consensus algorithms. In our analysis of Bitcoin and Ethereum, we learned how the Proof-of-Work algorithm allows both networks to come to consensus on blocks. Consensus algorithms in general ensure that the network on which it is deployed agrees on a singular blockchain as the only truth, and that malicious users cannot successfully break or fork the blockchain. In study, we find that Proof-of-Work wastes enormous amounts of energy for computation, may lead to centralization where electricity (for mining) is cheap, and in general does not scale well. This note will discuss some alternative consensus protocols, and how they are used both in public and private blockchains.

Properties of a Distributed System

Of course, in order to define some essential properties of distributed systems – of which Bitcoin is an example – we need to first define what a distributed system is. While the definition of a **distributed system** might vary from person to person, depending on their field of study, we are primarily interested in categorizing distributed systems as a network of independent nodes (computers), communicating via messages, and all working concurrently to reach a common goal. It is difficult to track time precisely in a distributed system, as every node could have its own slightly-off internal clock, so there is no global clock. Indeed, it would be impractical for every node in a network to share the same synchronized clock, and such a system may not be considered distributed at all. Instead, time is relative to a previous state of the distributed system (e.g. in blockchain systems, referring to a block number.) Components within a distributed system also have the potential to fail.

Liveness properties guarantee that a system must continue to make progress no matter the current state. In other words, they guarantee that a distributed system will always return a value and never reach a stuck state that cannot progress. In the context of consensus algorithms, liveness guarantees that all nodes will eventually decide on a value, and reach a new state. (Note the use of the word *decide* and *agree*, because a node may have to conform to what is agreed upon by the rest of the network.)

Safety properties guarantee that the network will always return a correct value. In other words, safety properties guarantee that something bad will never happen to a system. In regards to consensus algorithms, safety guarantees that no two nodes will decide on different values.

FLP Impossibility Theorem

In the **Fischer Lynch Paterson (FLP) Impossibility Theorem**, there are a total of three fundamental properties of distributed systems. We introduced liveness and safety already. The third property is fault tolerance, which guarantees that if one node in a system fails, the entire node will not fail. FLP Impossibility states that in any consensus system, only two of the three properties can be satisfied at once. Although fault tolerance is important, we will focus mainly on how to ensure safety and liveness in this note.

Distributed Properties in Consensus

To ensure the proper execution of a distributed system, we must also define **correctness** as the property that ensures that a distributed system will achieve its intended goal. Correctness can be expressed using our previously defined liveness and safety properties. **Consensus algorithms** are used to achieve three properties of correct distributed systems. **Validity** states that any value decided upon by the network must have been proposed by one of the processes or nodes. **Agreement**, a safety property, states that all non-faulty nodes must agree on the same value and will never decide on different values. **Termination**, a liveness property, states that all non-faulty nodes eventually decide on some value and will never be stuck.

CAP Theorem

Key Consensus Protocol Problems

Two key problems that consensus protocols must tackle are Sybil attacks and the Byzantine Generals Problem. Recall that Sybil attacks involve flooding a network with false identities and transferring malicious information to other nodes. Bitcoin makes Sybil attacks difficult because of its underlying Proof-of-Work algorithm and also some design considerations, including restrictions on connecting with geographically close IP addresses.

The **Byzantine Generals Problem** is a classic agreement problem that revolves around the concept of communication over an unreliable medium. Imagine a group of Byzantine generals surrounding a city, each in a different location. They must all commonly agree on a plan of action – either to attack or retreat. To communicate with one another, they must send messengers. The catch is that there may or may not be traitors among the generals, and also among the messengers. The question is: how do we ensure that all generals agree on the same plan of action? The positioning of the generals represents a distributed network. The possibility of there being traitor generals represents the possibility of faulty/malicious nodes, and the possibility of traitor messengers represents the unreliable network. The Bitcoin's Proof-of-Work algorithm is one possible solution to the Byzantine Generals Problem.

Bitcoin solves the Byzantine Generals Problem by assuming that the longest chain is always "correct" and that the majority of the network is not malicious. By this assumption,

Introduction to Cryptocurrencies and Blockchain
Lin, Akhtar

Note 8

Key Terms

A collection of terms mentioned in the note which may or may not have been described. Look to external sources for deeper understanding of any non-crypto/blockchain terms.

1. **VOCAB WORD** — Definition.

References

- http://www.coindesk.com/bitcoin-venture-capital/
- https://letstalkpayments.com/high-profile-investments-bitcoin-2013/
- http://www.coindesk.com/venture-capital-funding-bitcoin-startups-triples-2014/
- http://www.coindesk.com/6-weird-wonderful-bitcoin-events-2014/
- http://www.coindesk.com/7-biggest-crypto-scandals-2014/
- http://www.coindesk.com/year-headlines-coindesks-top-news-stories-2014/
- https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274
- http://www.coindesk.com/math-behind-bitcoin/
- https://en.bitcoin.it/wiki/Proof_of_work
- http://www.coindesk.com/short-guide-bitcoin-forks-explained/
- https://themerkle.com/bitcoin-hot-wallet-vs-cold-wallet/
- http://www.econinfosec.org/archive/weis2013/papers/KrollDaveyFeltenWEIS2013.pdf
- Narayanan, Arvind, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. N.p.: n.p., n.d. Print.
- http://www.coindesk.com/bitcoins-nervous-system-getting-upgrade
- Note 7
- http://www.oraclize.it/
- Note 8
- https://www.coindesk.com/short-guide-blockchain-consensus-protocols/
- http://the-paper-trail.org/blog/flp-and-cap-arent-the-same-thing/
- http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/
- http://henryr.github.io/cap-faq/