Blockchain Basics.
What It Is, How It Works, and Why it Matters.

Preston Byrne (Moderator)
Partner, Byrne & Storm, P.C.

Jakki Kerubo
Founder, Novum Communications Consulting

Greg Piccolo
Lead Engineer, Jigsaw XYZ

BLOCKCHAIN BASICS FOR THE LEGAL PRACTITIONER

By: Preston J. Byrne, Partner, Byrne & Storm, P.C.

1. BLOCKCHAIN 101

1.1. Bitcoin (proposal published 2008, paper released 2009) was the first true "blockchain" system. A blockchain is a *database.* Furthermore, it is a *peer-to-peer*, *distributed* database.

1.2. A blockchain database differs from other types of distributed databases because it has certain properties which make it *fault-tolerant*, i.e., even if one computer operating a piece of the blockchain, known as a *node*, fail, the rest of the network can continue to operate.

1.3. Blockchain fault-tolerance is also sometimes described as being *Byzantine* fault tolerant, a term of art which means that a blockchain should run even if the failure of a particular component, or a significant number of components, is caused not by an innocent failure but rather by a malicious attacker gaining control of a large number of nodes.

1.4. Blockchain nodes communicate with one another through peer-to-peer networking protocols over TCP-IP, much like the rest of the Internet. However, with blockchains, nodes communicate with each other directly (e.g. as BitTorrent does with the BitTorrent Mainline DHT[1]) rather than through a central server found at a particular address (e.g. https://facebook.com). These communications may or may not be encrypted.

1.5. Blockchain systems are, generally speaking, designed to communicate *economically significant transactional data.* Blockchains use cryptography to verify whether transactional communications are *valid.* If a blockchain node receives a valid communication, it will propagate that valid communication to other network nodes, with the consequence that the copy of the valid communication should eventually be written to the local copy of the receiver's blockchain and every other copy of the chain.

1.6. If a blockchain receives an invalid communication, the data will be rejected by an honest node and that node will not propagate the transaction to other nodes on the network. receiver and will not appear on the blockchain. Due to this, much like in commercial transactions transactional lawyers work with on a daily basis, a transaction on a blockchain "either completes or doesn't occur at all, and can't be left in an intermediate state."[2]

1.7. Valid transactions are batched together by individual nodes as they are received. Both the procedure, and end-state, whereby every node on a blockchain comes to an agreement that all of the transactions recorded on a blockchain are valid, are each referred to in the industry by the term "consensus." Consensus ensures that every blockchain node sees the same data as every other blockchain node.

1.8. This batching procedure, and resultant end-state, whereby every node on a blockchain arrives at an agreement that all of the transactions recorded on a blockchain are valid, are each referred to in the industry by the term "consensus." Consensus ensures that every blockchain node sees the same data as every other blockchain node.

[1] "DHT" means "Distributed Hash Table."
[2] Buchman, Ethan. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains,* p. 5. https://atrium.lib.uoguelph.ca/xmlui/handle/10214/9769

1.9. A node with permission to validate transactions, sometimes known as a *validator node*, has the power to decide that a number of transactions it has seen is valid.

1.10. In most blockchain systems, a validator node will batch transactions in a data structure which is known as a *block* and propose the inclusion of the block of transactions to the other validator nodes. Put another way, it asks the other validator nodes for their approval and consent to the block's inclusion in the blockchain database.

1.11. If a requisite majority of the other validator nodes agree that the block is also valid, then that block will be published to or adopted by the network, sometimes together with separate cryptographic proof of the validator nodes' collective consent. As part of this process, the validator nodes will also embed cryptographic proofs in each new block which are linked to certain cryptographic proofs in the previous block. This ensures that any tampering with any data in any prior block in the chain of blocks, or *blockchain*, will be immediately apparent to an observer looking at the present state of the data.

1.12. Once a block is agreed and published, it is appended to the end of the blockchain and the process begins anew, with new transactions being received by validator nodes and, in due time, being proposed to the network to form new blocks. How frequently this procedure repeats itself is dependent on the underlying architecture used by the blockchain in question. Currently there is a broad range, with systems like Bitcoin publishing new blocks that the network agrees on, i.e. *confirm* a block, on average every six minutes, and other systems, e.g. Tendermint, being able to confirm blocks once per second.

2. CRYPTOCURRENCIES: THE FIRST BLOCKCHAIN APPLICATION

2.1. ***Examples of virtual currencies***

2.1.1. **Bitcoin.** The first cryptocurrency, based on proof of work consensus.

2.1.2. **Ripple or Stellar.** Other early cryptocurrencies, based on proof of stake consensus.

2.1.3. **Dogecoin**. A so-called "altcoin," one of thousands of altcoins, with few technical differences between itself and Bitcoin. Altcoins' focus can be entirely arbitrary; in Dogecoin's case, the coin is themed around a cute Shiba Inu dog.

2.1.4. **Ethereum**. An altcoin that allows coin holders to upload scripts, known as "smart contracts," on the blockchain. "Smart contracts" are not smart and are not contracts, but allow users of cryptocurrency systems to model financial contracts (such as escrow, collateralized lending, or prediction markets) in blockchain code which settle entirely on-chain. Systems like Tezos or Eos are also smart contract systems.

2.2. ***Cryptocurrencies are not all the same***

2.2.1. Most major cryptocurrencies use the same elliptic curve cryptography for signing transactions, but do not encrypt transactions or data. Exceptions to the rule include ZCash and Monero, which do encrypt transactions and data.

2.2.2. Most major cryptocurrency protocols are licensed for free public use under open-source licenses. MIT, Apache 2.0, and GPL 3.0 are popular licensing schemes.

2.2.3. Cryptocurrencies differ from each other mostly on (a) how they achieve consensus, (b) what type of data they allow their users to communicate, and (c) performance characteristics.

2.2.4. **Proof of work** cryptocurrencies use a competitive game that requires the consumption of vast quantities of electricity to determine block-by-block consensus through a process known as **mining.** Bitcoin and Ethereum are examples of proof of work cryptocurrencies.

2.2.5. **Proof of stake** cryptocurrencies allow holders of existing quantities of the native cryptocurrency to vote on which transaction should be included in the next block through a process known as **staking.** Tendermint and NXT are examples of proof of stake cryptocurrencies. There are variations on proof of stake, such as "delegated proof of stake," utilized by cryptocurrencies such as Eos.

2.2.6. **Developers are constantly looking for new and better ways to achieve consensus on blockchain networks,** such as the "proof of space and time" method being used by Chia (founded by Bram Cohen, founder of BitTorrent).

2.2.7. **Transaction speeds vary widely.** Because cryptocurrencies are stateful systems, i.e. blockchain networks store all data that has ever been sent to them, developers of these systems need to make tradeoffs between performance and scalability. Where a system is designed to handle high transaction throughputs and data-heavy transactions, such as Ethereum (14 transactions per second/TPS plus smart contract code), the blockchain will swell in size, or *bloat*, making it difficult for ordinary users to run blockchain nodes.

2.2.8. **"How decentralized is it?"** …is never an easy question to answer. When this term, "decentralization," comes up, it can mean one of a number of things, including (a) how large and distributed the network of nodes is for a given network, (b) the chosen consensus mechanism being used by a given network, (c) the distribution model for coins issued on/by a given network, or (d) a range of other factors the market latches onto from time to time. The term lacks a concrete legal or technical definition, although senior officials of the U.S. Securities and Exchange Commission have tried to use the concept to determine the applicability of U.S. securities laws to cryptocurrency systems.[3]

2.3. ***Major legal issues arising from the cryptocurrency context***

2.3.1. **Anti money laundering/money transmitter licensing is** central to any cryptocurrency business.

2.3.1.1. Small-time Bitcoin exchangers have been charged with operating unlicensed money transmission businesses under 18 U.S.C. § 1960.

[3] Hinman, William. *Digital Asset Transactions: When Howey met Gary (Plastic.)* 14 June 2018, *https://www.sec.gov/news/speech/speech-hinman-061418*

2.3.1.2. **Required reading**: FinCEN 2013 Guidance defining "users," "administrators" and "exchangers"[4] of cryptocurrency systems

2.3.1.3. **Note "Layer 2" solutions** like Lightning Network do not eliminate money transmission concerns and require a standalone analysis depending on the design of the proposed application.

2.3.2. **Securities regulation:** a prevailing view among venture capitalists and others in the 2015-17 period was that securities regulation would be swept aside like municipal taxi regulations were swept aside by Uber. This view proved incorrect; practitioners should assume that, at point of issuance, all tokens are potentially securities and that if issued by companies should either be registered or benefit from an exemption to the registration requirement.

2.3.3. **Required reading:**

2.3.3.1. DAO Report of Investigation (SEC)[5]

2.3.3.2. Paragon/AirFox SEC Orders (non-registration of securities by issuer)[6]

2.3.3.3. EtherDelta SEC Order (non-registration of securities exchange)[7]

2.3.3.4. Crypto Asset Management, LP SEC Order (non-registration as investment adviser)[8]

2.3.3.5. U.S. v. Ignatov et al. indictments[9]

2.3.4. **New York Virtual Currency Business License ("Bitlicense")**

Required for, per 23 NYCRR 200.3(a):

- Receiving cryptocurrency for transmission *unless* for nonfinancial purpose and for a nominal amount
- Storing, holding, maintaining custody of cryptocurrency
- Buying and selling cryptocurrency as a customer business
- Performing exchange services as a customer business
- Controlling, administering or issuing cryptocurrency.

3. ENTERPRISE BLOCKCHAINS

3.1. ***What is an enterprise blockchain?***

3.1.1. An enterprise blockchain is a blockchain database that is not used in a "decentralized" manner.

[4] https://www.fincen.gov/sites/default/files/shared/FIN-2013-G001.pdf
[5] https://www.sec.gov/litigation/investreport/34-81207.pdf
[6] https://www.sec.gov/litigation/admin/2018/33-10574.pdf
[7] https://www.sec.gov/litigation/admin/2018/34-84553.pdf
[8] https://www.sec.gov/litigation/admin/2018/33-10544.pdf
[9] https://www.justice.gov/usao-sdny/pr/manhattan-us-attorney-announces-charges-against-leaders-onecoin-multibillion-dollar

3.1.2. Generally this means that *validator nodes* are identified in advance and controlled by known persons against whom legal recourse can be sought.

3.1.3. Possible range of applications as broad as software itself, generally focused on transaction and event control:

3.1.3.1. Payments and remittances (Ripple, R3)

3.1.3.2. Securities lifecycle automation (R3 Corda, JP Morgan's Quorum)

3.1.3.3. Hardware security and device authentication[10]

3.1.3.4. Supply chain automation and verification (IBM/Hyperledger Fabric frequently encountered)

3.1.3.5. Stock exchange infrastructure (Digital Asset Holdings, R3 Corda)

3.2. *What characteristics distinguish enterprise chains from cryptocurrencies?*

3.2.1. Cryptocurrencies express data mainly in the form as cryptocurrency token balances; enterprise blockchains do not need tokens to operate, and can therefore use more expressive smart contract scripts to describe and manage economically relevant events.

3.2.2. Validator/consensus arrangements usually follow what is required by the contractual terms of the transaction. So, e.g., a security might have consensus dictated by the note trustee working in concert with a platform provider.

3.2.3. Lawyers can be useful here in providing critical input to software design. Startup entrepreneurs often require a critical eye to ensure that their on-chain proposals and designs accurately reflect the commercial realities of the transactions they're trying to automate.

3.3. *Popular implementations*

3.3.1. R3 Interbank Consortium

3.3.1.1. Corda

3.3.2. The Hyperledger Consortium (under the auspices of the Linux Foundation)

3.3.2.1. Hyperledger Fabric (IBM blockchain protocol)

3.3.2.2. Hyperledger Sawtooth (Intel blockchain protocol)

3.3.2.3. Hyperledger Burrow (Monax blockchain protocol, Hyperledger's Ethereum Virtual Machine)

[10] See e.g. https://medium.com/blockchain-blog/blockchain-based-authentication-of-devices-and-people-c7efcfcf0b32
or
https://csrc.nist.gov/CSRC/media/Presentations/Leveraging-Blockchain-based-Protocols-in-IoT-Syste/images-media/1_iot_stavrous.pdf

3.3.2.4. Hyperledger Iroha (Soramitsu blockchain policy)

3.3.3. Ripple Labs

3.3.3.1. Ripple XRP (quasi-cryptocurrency) and Interledger (permissioned blockchain)

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

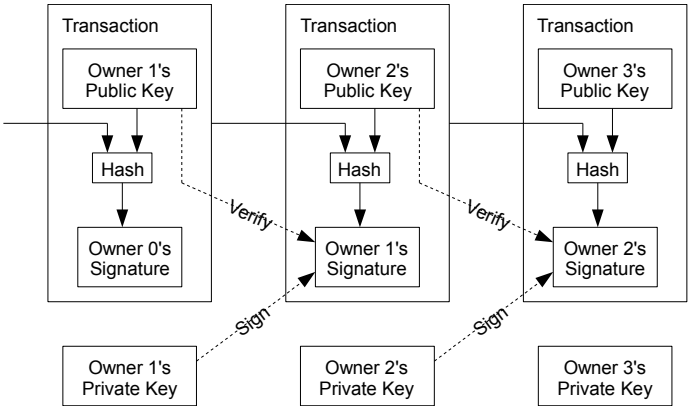
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

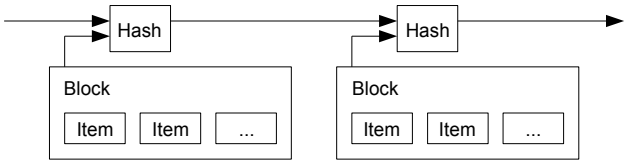


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

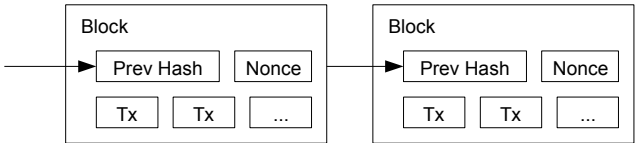
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

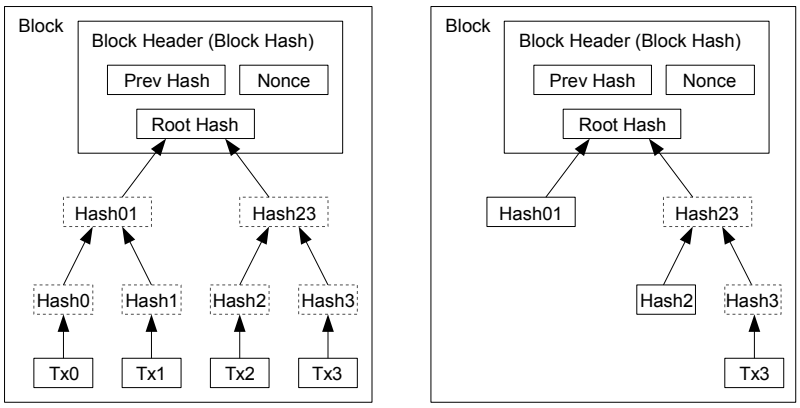
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



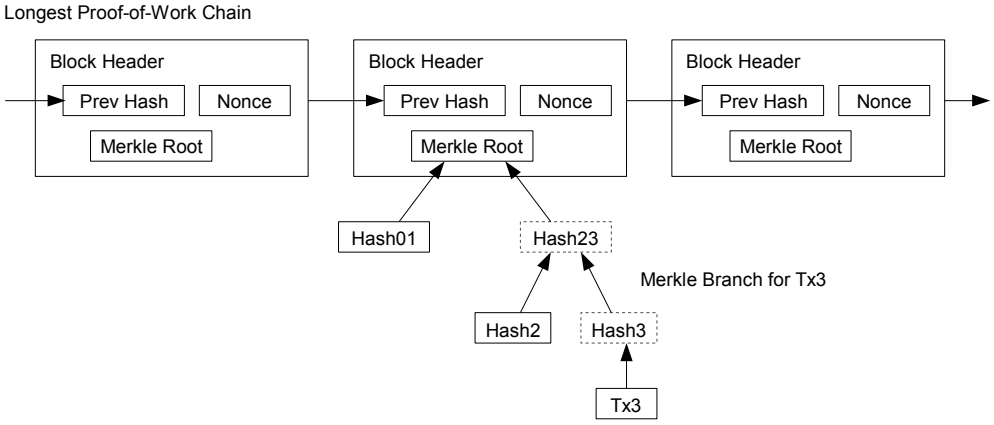
Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

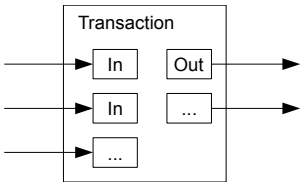
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

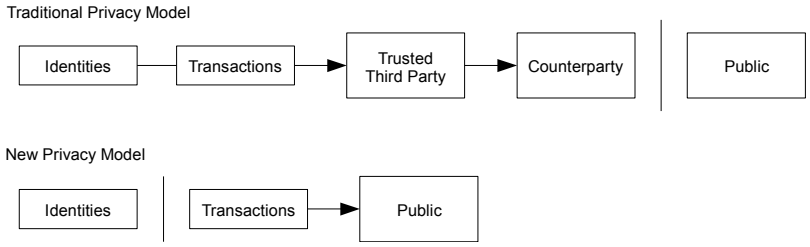
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

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

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.