UFO (Uniform Fiscal Object):
A Peer-to-Peer Electronic Cash System
based on Bitcoin codebase

www.ufobject.com

Based on works by Satoshi Nakamoto and
John Doering <ghostlander@phoenixcoin.org> and
Peter Bushnell <peter.bushnell@feathercoin.com>

Abstract

The original idea of a purely peer-to-peer version of electronic cash that would allow online payments to be sent directly from one party to another without going through a financial institution was introduced by Bitcoin's author Satoshi Nakamoto in his Bitcoin white-paper [9].

UFO implements the idea of being a younger brother of Bitcoin:

- decentralized,
- PoW mined,
- up to date codebase,
- ASIC resistant,
- well-suited for learning the blockchain – it is much faster, learning costs are barely affected by the Bitcoin price movements.

After a few years of Bitcoin blockchain, UFO blockchain was created in 2014 as an alternative solution to some issues that growing Bitcoin popularity has discovered. While based completely on Bitcoin's code base, it introduced an ASIC-resistant Neoscrypt algorithm, faster transactions, Automatic Checkpoint Protection (ACP) to protect the history of the chain and native Segwit support to keep the UFO blockchain up to date with the Bitcoin's latest features.

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. Satoshi's original solution to the double-spending problem using a peer-to-peer network is kept intact. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work blocks, 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 processing power. As long as a majority of processing 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 chain with the most proof-of-work as proof of what happened while they were gone.

In 2014 John Doering presented a new password based memory intensive cryptographic solution called NeoScrypt designed for general purpose computer hardware [10].

Since Bitcoin block time is averaged to 10 minutes, which is considered to be relatively slow in everyday use, UFO aimed to speed up the block time by decreasing the interval and adding extra protection with ACP.

In 2014 Peter Bushnell updated UFO with Neoscrypt hashing and ACP, which helps to mitigate the risks of chain reordering to the last 5 blocks while keeping all the previous chain unchanged.

UFO specifications

Proof of Work (PoW) NeoScrypt hashing algorithm generates blocks each 90 seconds on average. Subsidy is halving every 400k blocks. Max supply is around 4 billion coins. Genesis block was generated on the 2nd of January 2014.

Recommended confirmation time is 6 confirmations per transaction.

Important Bitcoin Improvement Proposals (BIP) supported include BIP141, BIP199.

Introduction

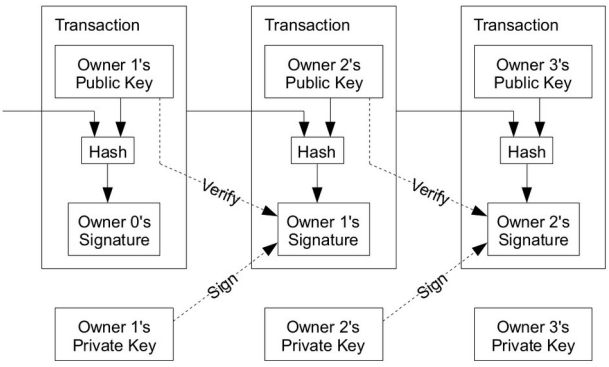
Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible 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.

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.

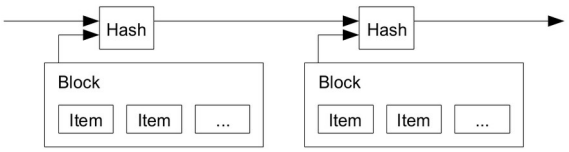


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new 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.

Timestamp Server

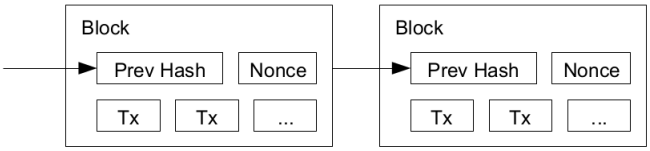
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to 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.



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. 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.

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.

Advanced Checkpoint Protection

Automatic Checkpointing protects the history of the blockchain against 51% attacks where malicious parties try to alter the history of the blockchain with the significant extra hashing power. Often this is to remove a transaction from the blockchain and return those funds to the sender leaving the recipient without the funds they previously held.

Automatic Checkpointing uses the peer-to-peer system along with signed checkpoint messages to make other nodes on the network aware of blocks on the checkpointed chain, nodes will not switch from a checkpointed chain to a chain that does not contain that checkpoint.

An automatic checkpointing masternode holds the private key for the checkpoint system, every time a new block is mined, a block at a fixed but definable depth in the chain is then used in a checkpoint message. The message itself consists of the checkpoint version, blockhash and a signature. Recipients of this message have a public key defined locally that relates to the central node's private key, the signature is validated against the signature and if correct the checkpoint is then set as the current checkpoint, but only if it is newer than the previous checkpoint and belongs to the same chain as the previous checkpoint.

The current automatic checkpointing system implementation is a trade off of centralisation for security and is proof-of-concept. The target architecture of the system is a decentralised solution where multiple automatic checkpointing masternodes will be defined, all nodes will send checkpointed messages as before. Recipient nodes will not accept a specific block as checkpoint, until messages are received from enough masternodes to meet a quorum of over 50% of the total number of masternodes.

Incentive

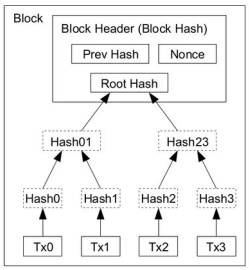
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant 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.

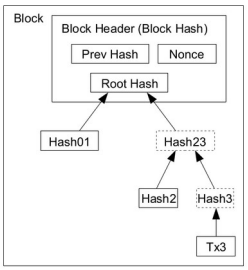
The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to 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.

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

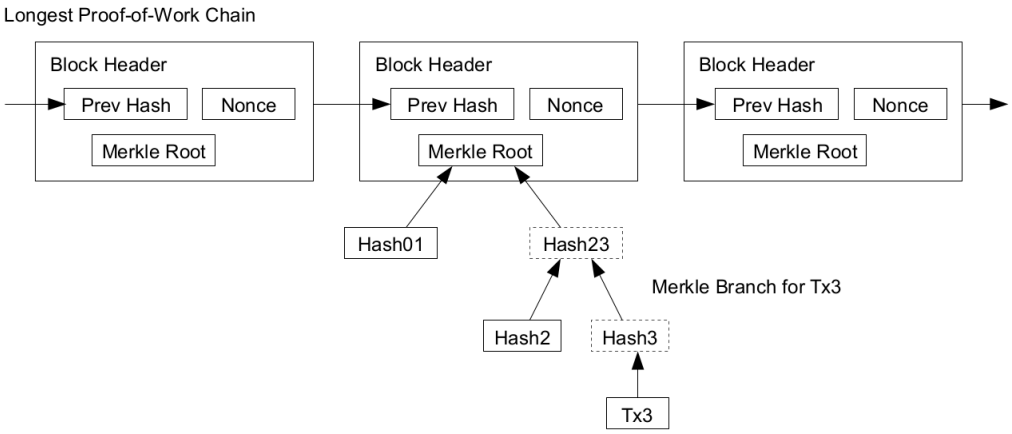


After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 90 seconds (40 blocks per hour), 80 bytes * 40 * 24 * 365 = 26MB per year. With Moore's Law predicting hardware capabilities growth, storage should not be a problem even if the block headers must be kept in memory.

Simplified Payment Verification

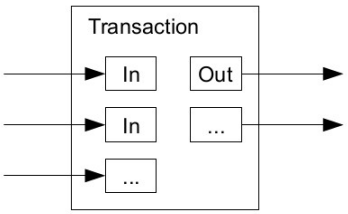
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction 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.

Combining and Splitting Value

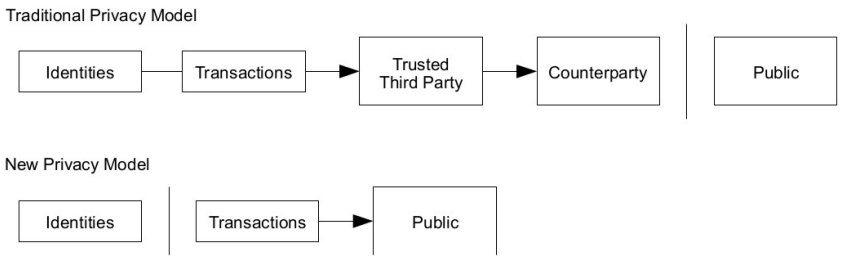
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous 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.

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.

Probability of altering the chain

Even if the scenario of an attacker trying to generate an alternate chain faster than the honest chain 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 probability of an attacker catching up is covered and calculated in the original Bitcoin whitepaper [9].

UFO Extended features

Segwit

Segregated Witness is activated in UFO blockchain, as described in BIP141. It is a structure called a "witness" that is committed to blocks separately from the transaction merkle tree. This structure contains data required to check transaction validity but not required to determine transaction effects. In particular, scripts and signatures are moved into this new structure.

Segwit implements protocol upgrade intended to provide protection from transaction malleability and increase block capacity. SegWit separates the *witness* from the list of inputs. The witness contains data required to check transaction validity but is not required to determine transaction effects.

Hash Time-Locked Contract (HTLC)

UFO blockchain has Hashed Time-Locked Contracts (HTLC) active for generalized off-chain contract negotiation, as described in BIP199.

HTLC is a script that permits a designated party (the "seller") to spend funds by disclosing the preimage of a hash. It also permits a second party (the "buyer") to spend the funds after a timeout is reached, in a refund situation.

HTLC enables support for Atomic Swaps and Lightning channels built over UFO blockchain.

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.

[9] Satoshi Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System», 2008.

[10] John Doering <ghostlander@phoenixcoin.org>, “NeoScrypt, a Strong Memory Intensive Key Derivation Function”, 2014.