3

PROOF OF WORK CONSENSUS

PoW Precursor - Hashcash Hashcash was proposed by Adam Back in 1997, for anti-sapm, anti-DoS detection [1].

How it worked? The sender of an email provided proof that he expended a certain amount of CPU work before sending the email. Sending large number of emails couldn't be profitable for someone who has to consume some energy for each message. On the contrary, the receiver or any other auditor could effortlessly verify the proof and easily detect spam emails.

Technically, Hashcash algorithm is based on the pre-image resistance of hash function, i.e. given a hash function $hash$, and a hash value h, it should be difficult to find any message m such that $h = hash(m)$. In particular it should be difficult to find a value m such that $0 = hash(m)$. Meanwhile, it's easy to calculate $hash(m)$, for any message m.

For Hashcash, proof of work is send in the form of a counter (or *nonce*, value used only once), attached in the email's header, such that the first 20 digits (5 significant hex digits) of the hash (originally SHA-1) value of the email are zeros. Finding a valid value for the counter is done in a brute force manner, by randomly choosing a value for the counter and then calculating the hash of the email. The search for a valid counter ends when the calculated hash

values is less then the the target value $2^{len(hash(m)-20)}$, i.e. having the first 20 digits zeros. Once a valid nonce value is found, anyone could verify it by calculating the hash value of the email. Finding the nonce is very much like completing a puzzle, nonce being the missing piece of the message to be sent.

The time needed to complete the header with a valid nonce is exponential with the number of required zeros, whereas time to confirm a valid header is constant and much faster.

Reusable tokens Suppose a spammer tries to send the same text message to more then one recipient. Is it possible for him to reuse the same generated hash (hash-token), minimizing computational effort to the cost of sending only one email? To answer this question, notice that the hash function is applied to a string formed by the email's header and email's body. The email header contains not only the counter, but also the recipient email address (or other resource string) and a timestamp denoting the exact moment the email was sent. Hence, the counter must be adjusted to match the target hash value, for each recipient email address, each time the message is sent.

But if hashcash tokens are not reusable isn't the energy spent for their creation wasted? What if receivers of an email would be able to use to POW token in outgoing emails, assuming that spammers don't receive any emails. As Nick Szabo pointed out in [11], hashcash tokens resemble precious metals, being unforgeable scarce and difficult to mine. There is though, a difference worth to be noticed. Paying with metals it's very weighty, but digital assets are easy to copy and transfer, hence it is mandatory to have a mechanism of preventing double-spending.

Hal Finney proposed a system [7, 10], *Reusable Proofs of Works*, that allows proof of work tokens to be reused. Actually it allows token exchange. A secured RPOW server generates RPOW tokens that replace incoming POW tokens, so that a proof of work token, rather than being discarded after use, can be replaced by a newly created token, then the new token can substituted by another one, etc in a kind of sequential reuse. A POW token can be replaced only once by a RPOW token, the RPOW server keeps track of all incoming tokens in a tree like database (a Markle-tree structure). More or Merkle trees will be covered in the following chapters.

RPOW is one of the ideas behind UTXO transaction model presented in the previous chapter. It inspired also the POW consensus protocol used in Bitcoin and Ethereum. Consensus protocols are a fundamental part of the blockchain technologies, more generally they play an essential role in distributed systems, where all nodes must agree on some data value. For instance in distributed databases the consensus protocol decides what transactions are processed and in which order; in state machine replication consensus protocol choose the order in which inputs/operations are processed. Consensus protocols should tolerate the presence of faulty or malicious nodes without imperil network security.

Consensus protocols for Blockchain ensures that all nodes participating in the P2P network maintain the same distributed ledger. Regardless of the transaction model (UTXO or account model), in a blockchain, valid transactions are gather into blocks, linked in a chain of blocks via block hashes. So maintaining the same distributed ledger actually means maintaining the same chain of blocks in all nodes.

Bitcoin network is advanced and secured by miners. Any node in the network can act as a miner, but special hardware resources are needed as mining is a costly process (CPU

mining, GPU mining, FPGA mining, ASIC mining etc.) both in terms of storage capacity and in computing power needed to solve the PoW puzzle. A new block is mined and added to the network each 10 minutes.

The steps for adding a new block are as follows:

1. **Nodes receives and validate transactions**, according to some rules imposed on transactions (valid signatures, output amount equals input amount etc.) as presented in previous chapter). Transactions are added in a transaction pool. *Transaction pool* stores a set of transactions that have been validated but not yet mined. Nodes synchronize their transaction pools by sending P2P network messages. Thus each new transaction is broadcast and validated by all nodes.

2. **Each node includes a set of new valid transactions from transaction pool into a new block**. The header of the new block points to the last accepted block in the blockchain. As depicted in figure 7 the block header include *previous block hash*, a timestamp and the root of the B-tree structure (Merkle tree) storing transaction hashes.

3. **Each node works to find the nonce in the proof of work puzzle** as in the hashcash algorithm. Nonce is the field in the block header to be set by the miner so that the block header's hash is less then block target, also stored in the block header, in field difficulty_target, see figure 7.

4. **After a node solves the proof-of-work puzzle, it broadcasts the block to all nodes.** After finding the appropriate nonce, the node advertise in the P2P network the newly mined block.

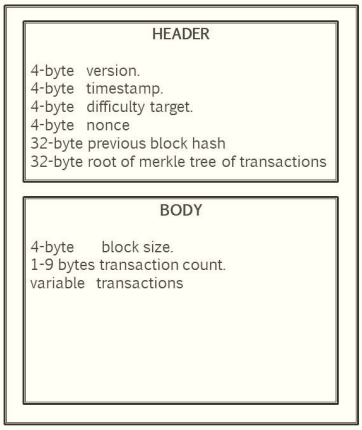

Figure 7: Bitcoin block structure

5. **All nodes accept the block**. Block is accepted if all transactions in it are valid. Also nodes verify that the hash of the block does not exceed target. Nodes express their acceptance of the block by adding it to their accepted version of the chain, i.e using the hash of the accepted block as the previous hash in the block they continue to work on.

Block target and difficulty The target is a 256-bit number [4] shared by all Bitcoin clients. The SHA-256 hash of a block's header must be lower than or equal to the current target for the block to be accepted by the network. The algorithm that finds the nonce satisfying the target (algorithm 1) resembles a lottery. If header block hash is below the target, then the miner wins. If not, miner increments the nonce and try again. The lower the target, the more difficult it is to generate a block.

The network has a global block difficulty [3] which measures how difficult it is to find a hash below a given target.

Algorithm 1 PoW finding block header nonce

```
nonce ← 10
while nonce < 2^32 do
    digest ← SHA_256(SHA_256(bl_header))
    if digest < target then
        return nonce
    else
        nonce ← nonce + 1
    end if
end while
```

The correlation between difficulty and target is given by the formula

$$difficulty = difficulty\_1\_target / current\_target$$

where $difficulty\_1\_target$ is the difficulty of the first block, namely a a hash where the leading 32 bits are zero and the rest are one.

The target is recalculated each 2016 blocks. If previous 2016 blocks were found in more than two weeks (expected rate is 10 minutes per block, for 2016 block this means two weeks) then the mining difficulty will be lowered, and if they were mined faster then that it will be raised. The more (or less) time was spent on finding the previous 2016 blocks the more will difficulty be lowered (raised). The idea is to keep he average time of finding a single block to a constant 10 minutes interval. Thus the formula calculating a new difficulty given the difficulty used by the last 2016 block is:

$$difficulty\_new = difficulty\_old * 20160 / difficulty\_epoch\_time$$

, where $difficulty\_epoch\_time$ is the total time measured in minutes, spent by miners to mine the last 2,016 blocks.

Coincidence or not, United States' Executive Order 6102 (6102 being the reverse of 2016) required confiscation of privately owned gold in 1933.

Longest chain rule. [6] Block difficulty is not only used in making the mining process harder or easier, but also in deciding the order of transactions. Blocks are chronologically ordered, beginning with the first block (*genesis block*). Since each block contains a hash of the previous block, it is guaranteed that each block is added in the chain after the previous block, thus the chain adopted by the majority of nodes in the network gives the correct transaction order.

As mentioned in [2], if two nodes broadcast different versions of the next block simultaneously or within a very short timeframe of each other, some nodes may receive one or the other first. In that case, there is a temporary fork in the network and nodes work on the first one they received, but save the other branch in case it becomes longer.

Such a fork caused by miners simultaneously adding a block will be solved 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. Honest miners will always build onto a block if it is the latest block in the longest valid chain. The length of the chain is measured not in terms of block height but in terms of accumulated block difficulty. Thus, the majority decision is represented by the chain which has the greatest proof-of-work effort invested in it.

Blocks in shorter chains (or invalid chains) are not used and are often called "orphan" blocks. When a node adopts the longer chain, all valid transactions of the blocks inside the shorter chain are re-added to the transaction pool and will be included in another block. The reward generated for the blocks on the shorter chain will not be present in the

longest chain, so they will be practically lost. A network-enforced 100-block maturation time prevents spending the rewards generated in orphaned blocks before they are invalidated. The miner of a block is allowed to spend the reward only after 100 blocks have passed after the block is processed. The 100-block maturation rule gives time for chain reorganization.

Block reward. For each new block added in the blockchain miners receive a reward. The first transaction in a block is a coinbaise transaction, issuing new BTCs in the miner's wallet. For the first four years of Bitcoin's existence, the the block reward was 50. Every 10 minutes 50 new bitcoins were issued. Since then, every four years, the reward is reduced by 50%. In 2022 for each new block, the miner receives 6.25 BTC. The halving process is one of the primary means to keep Bitcoin cryptocurrency deflationary as halving reduce the available amount of new supply, as demand increases. The price of bitcoin has increased significantly after each halving. For instance, price on 2020 halving day was $8821.42 and after 150 day it jumped to $10,943.00. Over time, each halving will diminish the block reward until it approaches zero. Hence, before the final halving that will take place in 2140, a total 21 million coins will be issued. After that, there will be no increase of the money supply. When the block reward will reach 0, miners will receive only transaction fees.

Block size limit. In Bitcoin network, Block size is another deciding factor for block's validation. [5]. In 2010, an explicit block size limit of 1 MB was introduced by Satoshi Nakamoto. Several **forks** modified this limit over time. A fork in a blockchain network (here with a different sense then that of a temporary split of the main chain due to mining two blocks in the same time as an accidental fork) is a

change of the rules to which network nodes adhere, or in other words, a change in the underlying protocol. There are two kinds of forks, **hard forks** and **soft forks**, introducing either radical or smother changes.

A hard fork occurs when security risks must be mitigated, new functionalities are added or when some transactions must be reverted to correct the effects of an attack.

The upgrades made by a hard fork are incompatible with the previous rules adopted by the nodes (it is not backward-compatible) and can see previous transactions as invalid. As a result, all nodes must upgrade to the newer version of the protocol. If some nodes switch to the new version of the block chain and some keep the old rules this leads to a permanent separation of the block chain. For example, a new cryptocurrency (Bitcoin Cash) was created after the **Hash Cash** fork, spliting the Bitcoin network. Until 2017, due to the 1MB block size limitation, number of transactions waiting for confirmation was constantly increasing. As a result users experienced longer transaction time and higher fees. In August 2017 Bitcoin Cash proposed to decrease fees and transaction validation times by increasing the size of a block between 8 MB and 32 MB.

A soft fork introduces new features and functionalities that do not thoroughly change the protocol rules, allowing both previous and new blocks to be accepted, keeping two versions of the block chain until all nodes reach consensus. This means a soft fork is backward-compatible.

Segregated Witness *SegWit* is a soft fork introduced in Bitcoin with the purpose of scaling the number of transactions included in a block, competing Hash Cash solution. If was activated on bitcoin's mainnet on August 1st, 2017. The term witness designates a cryptographic condition placed on an unspent transaction output (UTXO), for bitcoin it represents

the signature in the unlocking script. Segregated Witness moves the witness data from the unlocking script (ScriptSig) field of a transaction into a separate witness data structure that accompanies a transaction, in other words it separates the signatures from the transaction.

SegWit has two advantages:

- it increases transaction throughput;
- eliminates third-party transaction malleability.

SegWit is achieving a greater number of transactions (the main purpose of increasing the block size) by simply reducing transaction size, by separating the signatures from the transactions. Actually, nodes running SegWit accept a blocksize limit of 4MB but when calculating the block size, the non-witness data is multiplied by 4, while witness data count as 1/4 of a regular byte.

The nodes running the old protocol don't see any signatures in the newly created blocks, so the signatures don't count when the 1M size limit is checked by legacy nodes.

Transaction malleability means that a malevolent third-party can alter an unconfirmed transaction before it is validated in the network. After the transaction it's altered it will never be confirmed. Every transaction has a TX ID (transaction id) which is calculated by double hashing the transaction data. Recall that transaction ids are used to link transaction inputs to transaction outputs.

$$TX\_ID = SHA\_256(SHA\_256(transaction\ data))$$

Before SegWit fork, every input in a transaction was tailed by the signature that unlocked it. Thus, transaction data included signatures, and if signatures could be modified by an attacker, the id of a transaction could be modified.

Suppose Alice sends Bob 1BTC with transaction id TX_ID1 and then Bob forges a new transaction TX_ID2, copying the first transaction but changing only the signature. If Bob inform Alice that he never get the money, Alice will seek for transaction TX_ID1 which will never get confirmed, and will eventually decide to send Bob another 1 BTC. Using the SegWit rules, this scenario would not be possible, since transactions id are calculated by hashing the transaction data without signatures. The only part of the transaction that may be forged is actually omitted from the transaction data.

To better understand hard forks and soft fork, see figure 8 and 9.

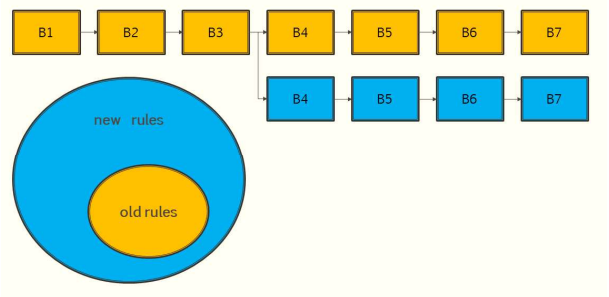

Figure 8: Hard fork

In generally, consensus rules become less restrictive after a hard fork and more restrictive after a soft fork, meaning that legacy blocks (blocks validated before the fork) are compatible with hard fork rules but not compatible with soft fork rules. Meanwhile blocks generated after a hard fork are incompatible with rules applied before the fork (legacy rules) and blocks generated after a soft fork are compatible with legacy rules.

For instance, after a hard fork that increases the limit of the block size, blocks generated before the fork, would pass the new requirement since they fit in a lower size limit. But new blocks, with size grater than the previous limit would not be accepted by nodes running the older version of the protocol.

As for the example of SegWit soft fork, all clients running the old or the new protocol accept new transaction data, with or without the accompanying witness data. Old clients interpret empty signatures as indicating outputs that may be spent by everyone. Meanwhile new nodes accept only blocks with segregated signatures.

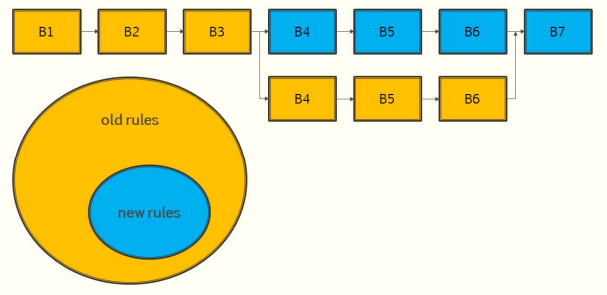

Figure 9: Soft fork

Block validation rules In the previous chapter we presented rules for validating transaction. Similarly there are rules for validating blocks. When a node receives a new block, it checks the following:

- The block data structure is syntactically valid
- The block header hash is less than the target difficulty
- The block timestamp is less than two hours in the future

- The block size is within acceptable limits.
- The first transaction is a coinbase generation transaction
- All transactions within the block are valid.

To sum up the last to two chapters, Bitcoin's consensus protocol encompasses four processes that run independently on every node:

- verification of each transaction, based on a comprehensive set of rules (as presented in previous chapter);
- aggregation of transactions into new blocks and demonstrating the computation through a proof of work algorithm;
- verification of new blocks (rules presented above);
- choosing the chain with the most cumulative effort invested in it (longest chain rule).