# Hash

Like a human fingerprint, a hash function is a very efficient means of comparing data. For this purpose, the hash function is applied to the data to be compared, i.e. its hash value is calculated and the hash values are compared with each other. These are of a limited length and can be compared much more efficiently than the original data.

There exist many different hash functions. How long the generated string is depended solely on the hash function. The hash functions most commonly used in Blockchain technology are the so-called *SHA-1* and *SHA-256* functions. SHA stands for **Secure Hash Algorithm**. They were developed by the *National Institutes of Standards and Technology* and meet specific security standards, such as collision-free operation. The *SHA-1* maps each character string to a fixed length of **160 bits**, the *SHA-256* generates a character string with a length of **256 bits**.

Hash functions have the following characteristic properties:

**Collision-free operation**

Hash functions are practically collision-free. This means that two different input values are almost always mapped to different output values. The fact that each element of the target set is accepted as a function value only once is called injectivity.

The following observation shows that hash functions are not strictly mathematically injective: Any character string can be an input value for hash functions, so there is an infinite number of input values. However, the number of output values is limited because the length of the hash value is limited. The number of possible input values is, therefore, greater than the number of output values. It is therefore impossible to guarantee 100% collision freedom. However, since the number of possible output values is large, you can ensure that the collision probability is close to zero.

**One-way property**

Hash functions ensure that it is not possible to deduce the input value from the output value. Functions that have these properties are called one-way functions. As a result, it is not possible to deduce the original input string from the created hash value.

**Determinacy**

After the previous properties, the temptation is obvious to think that the hash function assigns an entirely random value to the input. However, the hash functions are not random functions, but deterministic functions. This means that the same input value is always mapped to the same output value. Hash functions also ensure the one-way property. A hash value of the word "Blockchain" is therefore always the same, no matter who executes the hash function, at what time or place.

**Pseudorandom function**

In addition, the function is designed in such a way that an entirely new output value is created with the smallest change of the input value. If only one character is changed in an input value, a completely new character string is created as the output value. A differently set point or space can result in a completely new hash value.

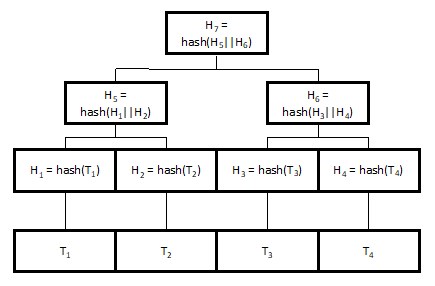
The word **Blockchain**, for example, is mapped by the SHA-256 to the hash value 625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1, while the only slightly changed word **Blockchain?** is mapped to the hash value 4a48bc5b69ec7587c4672030953978ab2956e32de5fe1ff65a373b5460413c67.

It also becomes clear why the hash value is suitable for comparing two information stands with each other. Every slight change of one information level leads to a completely different hash value. Due to the collision-free design, it is practically impossible for two different information states to have the same hash value. On the other hand, two identical information stands always have the same hash value, since the function is deterministic.

# Merkle Tree

A Merkle tree is a data structure. In this data structure, it is easy to determine the validity of an individual piece of data. Imagine there are four transactions. All these transactions (in reality, far more than four) are contained in one block. To see whether a specific transaction is contained in a block or if a transaction was unaltered, it would be necessary to search among all transactions of a block and then hash the entire block with all transactions formed. This would mean a substantial effort.

Verifying the validity of a transaction can be simplified considerably. Imagine Alice sends a known amount of a cryptocurrency to Bob. Alice now wants to know whether the transaction was integrated into a specific block. How is it possible for her to not have to download the entire block with all the transactions? The Merkle tree is the solution.



Here we see an example of a Merkle Tree. Below are the four transactions that are typically stored in a block. The data is hashed, and the four hash values H1 to H4 are created in the second line. Pairs are formed from these four hash values. Here H1 and H2 form a pair, and H3 and H4 form a pair. The hash values of the pairs are concatenated (represented by the symbol ||). In concrete terms, chaining means that the values are concatenated and then hashed again. Thus the two new values, H5 and H6, are created in the third line. These two values are then attached and hashed again and result in our final hash value H7. This is then called the Merkle Root. Only this Merkle root is integrated into the block header. The block header with the Merkle root is public and requires only **80 bytes** of memory.

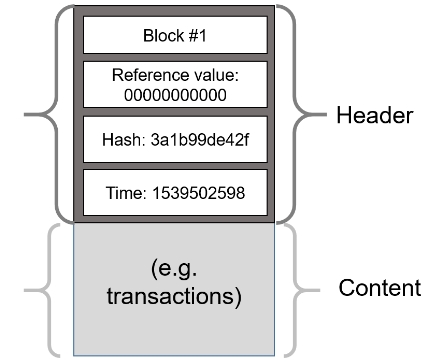
If someone wants to know whether a transaction is really contained in a block at a particular place, only a part of the tree must be known to him. This part of the tree can also be acquired from dishonest sources because the Merkle root is unchangeably stored in the block header. If the dishonest source provides false information regarding the requested part(s) of the tree, this can be determined immediately. Then a different value is obtained when calculating the Merkle root. Because of the collision-free nature of the hash functions, it is nearly impossible to find another modified Merkle tree that has the same Merkle root. A source for the missing parts would be the miner of the specific block in which the transaction should be contained.

Let's imagine that we want to make sure that the first transaction, T1, actually says that Bob sent us 5 bitcoins and that this is contained in a block. To verify this, only H2 and H6 are required in addition to the transaction information. You can use precisely these values to calculate the Merkle Root and compare it with the published Merkle Root. Therefore, you don't have to have the whole tree, but only one value per line and the original transaction you want to check. This means that the number of files to be sent is log2(4) + 1, because the logarithm to base 2 is the number of rows. There must be one value per line. The added one describes the transaction to be checked. This makes verification much easier because much fewer files are required. In our example, the difference is not striking; in real blockchains there can be thousands of transactions per block, which certainly matters then. The block headers therefore, only contain the Merkle roots. The individual parts of the trees are only contained in certain computers, for example in those which carry out the transaction.

# Block

As the name suggests, the blockchain consists of many blocks. The blocks are numbered (starting with block one - the Genesis block) in their sequence. Each block contains a list of transactions as described in the previous chapter. Furthermore, each block contains a timestamp, i.e., the exact time and date of the point in time at which the transaction is added. The set of all blocks is called the Distributed Ledger, or blockchain.

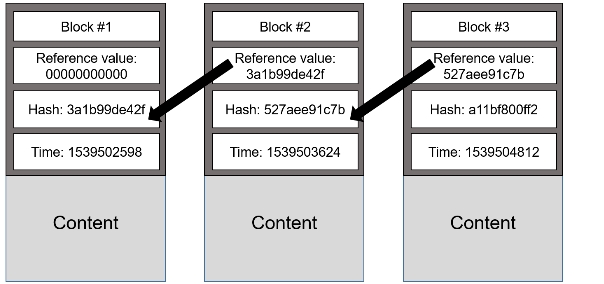
The data in the blockchain is stored in a public peer-to-peer network. To protect the blockchain contents, a concept must be developed so that no malicious entity can tamper with the recorded data. For this, the hash value of each block is calculated and stored in the successor block. The hash value is obtained by forming the SHA-256 value from all contents (i.e., the list of transactions, the time stamp and the block number).



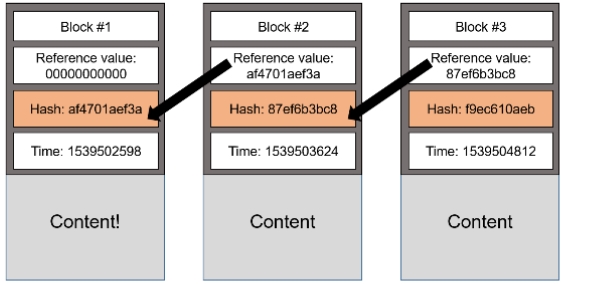
If something in the block is changed, the hash would change as well, because hash functions are pseudorandom functions, and therefore every change in the input value leads to a changed output value.

## Chain of blocks

The reference value in a block is the hash value of the previous block, which is also included in the calculation of the respective hash value. Note that the very first, the genesis block, has no previous block.



If a block is changed, the hash value of this block changes. This changes the reference value of the next block. Since the reference value is included in the formation of the hash value of a block, the hash value of the next block also changes. There implies a change in all subsequent blocks.



In the figure above the contents of the first block has been changed to **Content!**. As a result, the hash value of block one is changed, and the reference value of blocks two and three is changed. Since the reference value is included in the formation of the hash value of a block, the hash value of block two and three also changes. The change takes place in this way in all subsequent blocks.

Now the name blockchain finally becomes clear: the many blocks are connected by reference values, just like a chain.

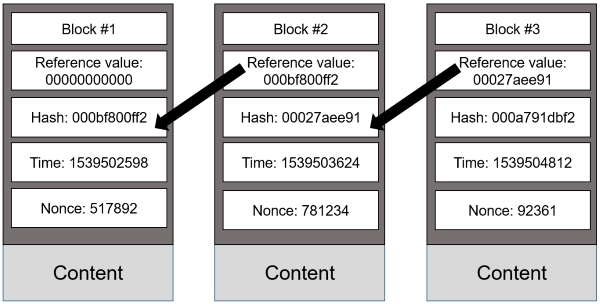
# Proof of Work

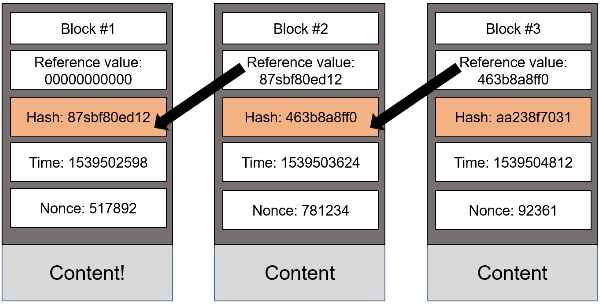
Currently, a change in the content of one block leads to changes in all other blocks. Further consequences do not exist yet. To render a block tamper-proof, a challenge or puzzle is added to the blockchain. To add a new block to the blockchain, this puzzle has to be solved. Solving the puzzle takes processing power. If an old block is changed, then the solutions of all puzzles of the block and the following blocks change. To construct a valid blockchain, all puzzles have to be solved again, which takes a lot of computational power.

The puzzle to be solved can be compared well with a bicycle lock. There is a bicycle lock on each block. This must be opened by finding the right combination of numbers. If something is changed in a block, then all bicycle locks of this block close and the following blocks and new number combinations must be found.

But what does this puzzle look like with the blockchain? In each block, an additional value, the **nonce**, is recorded. A nonce is a number that is part of the calculation of the hash value of a block. The nonce must now be changed until a certain condition is fulfilled, for example until the first n digits of the respective block's hash value are zero. The number n of zeros varies. The more zeros are necessary, the lower the probability to fulfill the condition, the more trying is required to find the matching nonce. This process is called **Proof of Work**. Each time a new block is appended to the chain, this proof of work must be performed, which is consuming processing power. Since the matching nonce can only be found by trial and error, the nonce's presence proves that a computer has done a time-consuming job of appending a new block.

The Proof of Work of Bitcoin's blockchain is designed such that it takes about ten minutes on average to be solved.





Brute force is the best-known approach to solving proof of work, why is that? The reason for this is the puzzle friendliness property of hash functions ([see this lecture](https://www.youtube.com/watch?v=igEeRf3h4aw&list=PLOa3v6xgsJullbz4uD13nm-U5D_cw0xLh&index=3)). This means that it is not possible to deduce the original value from a hash value. Instead, original values must be tried out until the desired hash value is found.

We found out earlier that a change in the content of a block results in a change in the hash values of that block and the following blocks. This has the immediate effect that the zeros in the first few digits are lost. The Proof of Work has to be done for this block as well as for all other blocks. This again is connected with a huge expenditure of computation. Other computers in the network have already attached a large number of new blocks in this period. So the modified ledger may be much shorter than the unmodified one, and hence is unlikely to become the main chain. The changed blockchain is recognized immediately and distinguished from the unchanged one. In summary, the longest blockchain, with the most proof of work is the valid one.

Now that we have understood how the blockchain works, it is important to understand how new blocks are appended.

# Gossip

The principle of a ledger distribution can be well explained by a comparison with the human phenomenon of social interaction and communication. When information is shared among friends or colleagues, it is highly likely that it will be shared with other friends. In most cases, the probability is even higher if the person is asked not to pass on the information. This can be explained by the fact that sharing information about others is usually a relatively efficient way to strengthen social contacts or make new friends. People are also interested in information about the people around them. That is why there is a large exchange of information. The information is disseminated by individuals.

The distribution of the current versions of the Distributed Ledger can be explained very well using the above example of information distribution within a circle of friends. There is no central computer in the peer-to-peer network. The information cannot be sent from one element to all other computers. The basic idea behind the dissemination of information in the peer-to-peer network is the same as in the human communication example. The computers in the peer-to-peer network each have "friends". We call "friends" those computers with which the individual computer exchanges information. Between computers, there are also the three types of conversations that we have categorized regarding the exchange of information between people.

Each computer has its own list of computers ("friends") with which it communicates. Therefore, information is first sent to the own "friends", who then send the message to a number of computers that have nothing to do with the original computer. When a computer receives a message, it forwards it to all peers on its "friends" list. These computers then do the same with their list of "friends". The information is then securely distributed to each individual computer. This principle is called the Gossip Principle because it is comparable to the gossip of a group of people.

Each message sent between computers has its own hash value. This makes it easy to identify and ignore duplicate messages. Since each message is timestamped, the problem of time delay is elegantly solved. When a computer receives a message a second time, it does not resend it because all its peers have already received it. So a message is not forwarded in the network forever.

This concept makes it possible to disseminate information in the peer-to-peer network and to ensure that no important information is lost. Note that gossip, or broadcasting information isn't necessarily very efficient.

# Signature

In a network with unknown participants, each time data is exchanged, it must be proven that the person claiming to be the sender account owner is actually the owner of that account. This is implemented on the Blockchain with the help of the Digital Signature Algorithm. Each transaction is signed with this algorithm. This makes it possible to prove that the transaction was signed by the account holder. In addition, this signature is depend on the transaction data and can therefore not be reused. Furthermore, the transaction must not be changed after it has been signed, since the signature becomes invalid after an unauthorized change was made to the transaction.

Each account on a Blockchain has a private key, i.e., a kind of password, which is required for each transaction from an account. This password is top secret and should not be disclosed to anyone. If the secret key is lost, the corresponding cryptocurrency is lost as well. The account name is public and consists of a sequence of letters and numbers. This is a public key, visible to everyone. The public key is generated with the private key's help, but it is not possible to deduce the private key from the public key.

If a node of the peer-to-peer network wants to check whether a transaction is valid, it can verify the transaction signature with the account's public key.

