# BLOCKCHAIN: WHAT IS IN A BLOCK?

Damien Cosset

## INTRODUCTION

In my previous article, I tried to described the concept of a blockchain with code. This time, I'll try to describe the structure of a single block. I will use the Bitcoin blockchain to explain blocks, but keep in mind that the concepts will remain more or less the same. It could be useful to read my last article to understand a few things first.

## STRUCTURE OF A BLOCK

A block is a container data structure. In the Bitcoin world, a block contains more than 500 transactions on average. The average size of a block seems to be 1MB (source). In Bitcoin Cash ( a hard fork from the Bitcoin blockchain ), the size of a block can go up to 8MB. This enables more transactions to be processed per second.

Anyway, a block is composed of a header and a long list of transactions. Let's start with the header.

## BLOCK HEADER

The header contains metadata about a block. There are three different sets of metadata:

- The previous block hash. Remember that in a blockchain, every block is inherits from the previous block because we use the previous block's hash to create the new block's hash. For every block N, we feed it the hash of the block N-1.
- Mining competition. For a block to be part of the blockchain, it needs to be given a valid hash. This contains the timestamp, the nonce and the difficulty. Mining is another crucial part of the blockchain technology, but it is outside the scope of this article.
- The third part is a merkle tree root. This is a data structure to summarize the transactions in the block. And we will leave it at that for now. More on this later.

## BLOCK IDENTIFIERS

To identify a block, you have a cryptographic hash, a digital signature if you will. This is created by hashing the block header twice with the SHA256 algorithm. For example, this is a block. I will refer to this block as an example for this article.

The block header hash for this particular block is (right column): *000000000000000000301fcfeb141088a93b77dc0d52571a1185b425256ae2fb*

We also can see the previous block's hash (right column): *0000000000000000004b1ef0105dc1275b3adfd067aed63a43324929bed64fd7*

Remember that we used the second hash to create the first. Every block uses the previous block's hash to construct its own hash. The block hash is a unique identifier. You won't find two blocks with the same hash.

The other way to identify a specific block is the block height. The is the position of the block in the blockchain. Our example's block is in the 500312 position. This means that there are 500311 blocks before this one. Since the creation of the Bitcoin blockchain in 2009, 500312 blocks have been created ( at the time of writing obviously ).

A block height is not unique. Several blocks can compete for the same position in the case of a fork, like Bitcoin Cash for example.

## MERKLE TREES

The transactions in a block are contained in a structure called a merkle tree or binary hash tree.

I feel that topics like that are easier to understand with actual examples. So we'll go coding for this. A merkle tree is constructed by recursively hashing pairs of nodes ( in this case, transactions ), until there is only one hash, called the root or merkle root. If we stay in the Bitcoin world, the cryptographic hash algorithm used is SHA256. This is applied twice each time.

An example: We have a block with 4 transactions. For the sake of simplicity, each transaction is a string:

```
const tA = 'Hello'
const tB = 'How are you?'
const tC = 'This is Thursday'
const tD = 'Happy new Year'
```

To construct our merkle tree, we start from the bottom. We take each transaction and double-hash them. I'll use the js-sha256 package here.

```
const sha256 = require('js-sha256').sha256

// Double-hashing here
const hA = sha256(sha256(tA))
const hB = sha256(sha256(tB))
const hC = sha256(sha256(tC))
const hD = sha256(sha256(tD))

//Results
52c87cd40ccfbd7873af4180fced6d38803d4c3684ed60f6513e8d16077e5b8e //hA
426436adcaca92d2f41d221e0dd48d1518b524c56e4e93fd324d10cb4ff8bfb9 //hB
6eeb307fb7fbc0b0fdb8bcfdcd2d455e4f6f347ff8007ed47475481a462e1aeb //hC
fd0df328a806a6517e2eafeaacea72964f689d29560185294b4e99ca16c63f8f //hD
```

Ok, great. Now remember that I wrote a merkle tree is constructed hashing *pairs* of nodes. So, we will pair our transactions and concatenate their hashes. Then, we will double hash them too. We will create a hash using the hashes *hA* and *hB*, and another for *hC* and *hD*. Then, we repeat that process until we have only one hash left and no more pairs to work with. The last hash will be our merkle root.

With only four transactions, this will be rather quick:

```
//Pairing hA and hB

const hAB = sha256(sha256(hA + hB))
//5dc23d1a2151665e2ac258340aa9a11ed227a4cc235e142a3e1738333575590b

//Pairing hC and hD

const hCD = sha256(sha256(hC + hD))
//ff220daefda29821435691a9aa07dd2c47ca1d2574b8b77344aa783142bae330

// We do it again. We pair hAB and hCD
// This is our root!
const hABCD = sha256(sha256(hAB + hCD))
//301faf21178234b04e1746ee43e126e7b2ecd2188e3fe6986356cc1dd7aa1377
```

The node at the top of the merkle tree is called the root. This is the information that is stored in the block header in each block on the blockchain. This is how transactions are summarized in each block. In our example block given earlier, the merkle root can be found in the right column: *a89769d0487a29c73057e14d89afafa0c01e02782cba6c89b7018e5129d475cc*

It doesn't matter how many transactions are contained in a block, they always will be summarized by a 32 bytes hash.

*Note:* The merkle tree is a binary tree. If there is an odd number of transactions, the last one will be duplicated so we can construct our tree.

Because all the leaves in our tree depends on other leaves, it is impossible to alter one leaf without altering others. If you change only one leaf ( one transaction ), the hash changes, therefore the hash you constructed by pairing it with another leaf changes, therefore the merkle root will be different.

You can prove that any transaction is included in a block by creating a *authentification path* or *merkle path*. You only need to know *log base 2(N)* 32-byte hashes. For example:

-For my 4 transactions merkle tree:

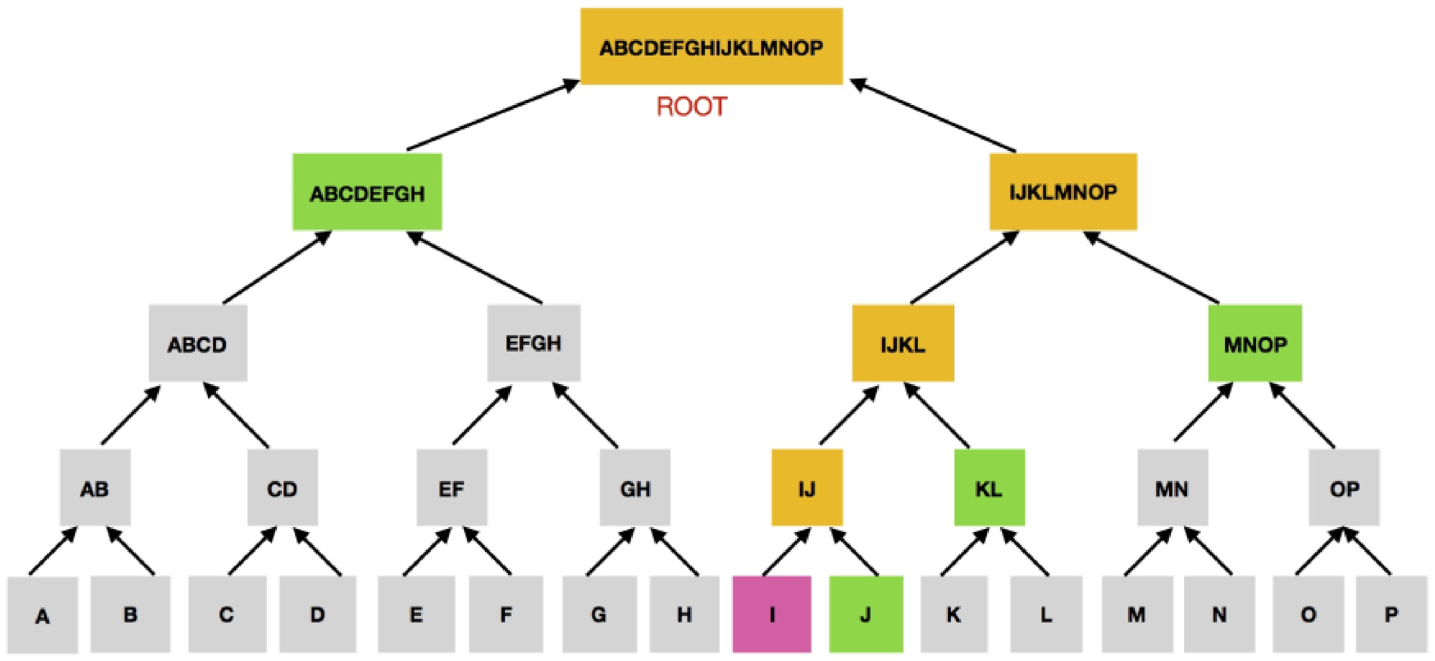
log base 2( 4 ) = 2 => If I have a path of 2 hashes for a tree of 4 transactions, I can manage to prove if a transaction belongs to this merkle tree.

For a 16 transactions merkle tree:

log base 2( 16 ) = 4 => If I have a path of 4 hashes for a tree of 16 transactions, I can manage to prove if a transaction belongs to this merkle tree.

log base 2( 1500 ) = 10.55 => If I have a path of 11 hashes for a tree of 1500 transactions, I can manage to prove if a transaction belongs to this merkle tree.

Perhaps a little diagram will help.



There are 16 leaves in this tree. We construct our tree from the bottom up by pairing each leaf. Now, anybody can prove that the leaf *I* ( in orange ) is part of this block by having the path given in green. We have only 4 hashes, but that is enough to know if the leaf *I* belongs here. That is because with those informations, we are able to construct every single leaf we need( in yellow ). We can create *IJ*, *IJKL*, *IJKLMNOP* and the root and check if those hashes correspond. This is why it is very complicated to cheat a blockchain. To change one thing means you must change everything.

Well, that's pretty much what a block contains in the Bitcoin blockchain. Hope it helped!