Building a Biodiversity Blockchain from Scratch

Paul Oldham

29/12/2019

"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." (Nakamoto 2008)

This paper provides a crash course in writing a biodiversity blockchain from scratch. Blockchain is not intuitive since it involves unfamiliar cryptographic ideas and tools. In response to this data scientists have written guides to creating blockchains in a number of programming languages such as R and Python. Data scientist and artist Massimo Franceschet from the Maths Department at the University of Udine in Italy has written a step by step guide to the process in R (Franceschet, n.d.). We will use Massimo Franceschets code as a worked example of block chain creation. Credit for the code belongs to Massimo Franceschet with annotations and minor changes added by the author.

As a starting point it is important to bear in mind the order in which blockchain transactions are addressed. As Massimo Franceschets explains the order is:

1. new transactions are broadcast to all nodes (computer nodes in the network);
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;[1]
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.
7. In addition, nodes always take the longest chain as the correct chain, search for and then work on that chain.

In this paper for ease of explanation we will not strictly follow this order but suggest it is borne in mind.

[1] In the case of cryptographies such as Bitcoin as transaction is regarded as complete when six blocks have been added after the block containing the transaction. This appears to reflect the lag between the broadcast of new blocks and nodes in the network validating the blocks and the longest chain to continue work.

Creating a Chain of Blocks

As a first step we will start by creating a simple set of blocks. A block normally contains a set of transactions with an identifying number, a time stamp, some data and a reference to the previous block. To illustrate the creation of a blockchain we will use three blocks with a single entry. A chain of blocks is created by adding the hash of the previous block in the next block. The first block in a blockchain is known as the "genesis" block and its parent hash is zero because it has no parent.

As the idea of hashing may be unfamiliar we illustrate with one simple block. A hash or digest is created using cryptographic hash functions. The most popular of these functions come from the Secure Hash Algorithm 2 family created by the United States National Security Agency. One of the most widely used of the hash algorithms is SHA-256. Hashes are easy to create in programming languages, in this case we use the digest package in R.

```
library(digest)

block1 <- list(number = 1,
               timestamp = "2018-10-01 17:24:00 CEST",
               data = "Lepidium meyenii genome data or link",
               parent_hash = "0") # genesis block

block1$hash = digest(block1, "sha256") # hash block 1

block1$hash

## [1] "027dca8f7f8e6585606774c66aaec07515d61b61d19528a5400c6d3ac1eabb6f"
```

The cryptographic hash that identifies block 1 is printed above. As we create a chain the hash of the previous block is added to the block and chains the two together. This process continues in a sequence as illustrated below.

```
library(digest)

block1 <- list(number = 1,
               timestamp = "2019-10-01 17:24:00 GMT",
               data = "Lepidium meyenii genome data or link",
               parent_hash = "0") # genesis block

block1$hash = digest(block1, "sha256") # hash block 1

block2 <- list(number = 2,
               timestamp = "2019-10-01 17:24:15 GMT",
               data = "Banisteriopsis caapi genome data or link",
               parent_hash = block1$hash) # add block 1 hash

block2$hash = digest(block2, "sha256") # hash block 2
```

```
block3 <- list(number = 3,
               timestamp = "2019-10-01 17:24:30 GMT",
               data = "Hoodia gordonii genome data or link",
               parent_hash = block2$hash) # add block 2 hash

block3$hash = digest(block3, "sha256") # hash block 3

# create the chain

blockchain = list(block1, block2, block3)

# print the last block in the chain

blockchain[[3]]
```

```
## $number
## [1] 3
##
## $timestamp
## [1] "2019-10-01 17:24:30 GMT"
##
## $data
## [1] "Hoodia gordonii genome data or link"
##
## $parent_hash
## [1] "fbd176ec4060624faba873b9836f97f8d7e6b2c20d7b79b9b6438aa3396488d9"
##
## $hash
## [1] "248f26c58a7d07c01ba7787fa63d3430bf367244a444befde6ea7da8cbde683e"
```

As discussed above, a key feature of a blockchain is that it should be unchangeable or immutable. We can illustrate this by testing if we have a valid blockchain and see if we can alter it. For the sake of brevity we will not show the validation function which tests that the parent hash matches the hash of the previous block.

Is the blockchain valid?

```
validate(blockchain)
```

```
## [1] TRUE
```

Is the chain still valid if we attempt to change the data in block 1?

```
blockchain[[1]]$data = "Human genome"

validate(blockchain)
```

```
## [1] FALSE
```

This example illustrates that the chain is immutable. Any effort to change an element of the chain will render the chain invalid and will not be accepted as valid in the network of nodes involved in maintaining the blockchain ledger.

Proof of Work

As discussed earlier some kind of `Proof` is required in blockchains as a basis for adding new blocks to the chain. In Bitcoin and other blockchain this involves a miner solving a hard cryptographic puzzle in order to create a new coin. The Bitcoin proof of wok consists of requiring a miner to find a number (known as a nonce) that when added to the front of a cryptographic hash will create a specific number of leading zeros. For illustration, in the example below the miner must find a nonce of 252 that creates the hash with the three leading zeros.

```
## $hash
## [1] "00017734f1bfea30824c555c1ecae98caddd1c315ae82622d91cbdf90c854e07"
##
## $nonce
## [1] 252
```

It is important to emphasise that some form of Proof such a `Proof of Work` is required for the creation of new blocks to avoid corruption of the chain with spam or fraudulent blocks. For that reason the problem to be solved must be difficult while the incentive to try and solve the problem, typically a coin or fee, must be sufficiently attractive to promote participation. At the same time, while the problem must be difficult the means of verification should be easy.

Digital Signatures

Having generated a block using a `Proof` as a container for the transactions we are presented with a set of transactions inside the block that need to be verified.

The importance of digital signatures in blockchains becomes clearer if we return to the original Nakomoto Bitcoin paper. In this paper Nakomoto 2008 states that:

"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." (Nakamoto 2008)

In the case of a blockchain involving access and benefit sharing we are not dealing with coins but with recording something such as:

"provider A agreed to give user B access to genetic data X on condition Y in return for Z"

However, in common with the Bitcoin example what needs to be validated are the digital signatures involved for each transaction in a block.

As a block is actually a record of transactions we adjust our example blocks to contain transactions between the provider of genome data and the user of genome data. The id of the genome is shown for illustration.

In this example, `benefit` refers to some form of benefit value for accessing the genome data. The `fee` in this case goes to the miner who creates the block to validate the transaction. The fee provides the incentive for someone in the network to validate the transaction so that the provider can receive the benefit value. In this case the `benefit` value has no definition other than to suggest that the provider might receive some unit of benefit in return for providing access to the genome data (e.g it could be zero). For the purpose of illustration the fee is set as a percentage of the benefit value.

Based on the earlier discussion of the BioSample database and data use categories for human genome material we will include these elements in the transaction ledger data.

provider	user	genome_id	accession_id	data_use	benefit	fee
R	V	Lepidium meyenii	1	commercial	82	0.41
J	J	Banisteriopsis caapi	2	non-commercial	11	0.61
T	X	Hoodia gordonii	3	specific	91	0.46
O	K	Oryza sativa	4	commercial	73	0.12
D	D	Zea mays	5	non-commercial	23	0.33

We now create two blocks that contain the transactions.

```
blockchained(transactions, nblocks = 2, block_size = 3)
```

```
## [[1]]
## [[1]]$number
## [1] 1
##
## [[1]]$timestamp
## [1] "2020-01-16 11:56:33 GMT"
##
## [[1]]$data
## # A tibble: 3 x 7
##   provider user  genome_id            accession_id data_use       benefit
fee
##   <chr>    <chr> <chr>                <chr>        <chr>            <dbl>
<dbl>
## 1 J        J     Banisteriopsis caapi 2            non-commercial      11
0.61
## 2 T        X     Hoodia gordonii      3            specific            91
0.46
## 3 R        V     Lepidium meyenii     1            commercial          82
0.41
##
## [[1]]$parent_hash
## [1] "0"
##
## [[1]]$nonce
## [1] 4267
##
```

```
## [[1]]$hash
## [1] "000b6560c233eeecc2437362d993763100ba5701a921a6326347034e9ac70f7f"
##
##
## [[2]]
## [[2]]$number
## [1] 2
##
## [[2]]$timestamp
## [1] "2020-01-16 11:56:34 GMT"
##
## [[2]]$data
## # A tibble: 3 x 7
##   provider user  genome_id        accession_id data_use       benefit
fee
##   <chr>    <chr> <chr>            <chr>        <chr>            <dbl>
<dbl>
## 1 D        D     Zea mays         5            non-commercial   23
0.33
## 2 O        K     Oryza sativa     4            commercial       73
0.12
## 3 R        Z     Lepidium meyenii 1            commercial        1.48
0.01
##
## [[2]]$parent_hash
## [1] "000b6560c233eeecc2437362d993763100ba5701a921a6326347034e9ac70f7f"
##
## [[2]]$nonce
## [1] 4495
##
## [[2]]$hash
## [1] "000601258afe407924ccc8405867091ab4a01b716a4b5f497cbc3fadf244044b"
```

We now confront the problem that we do not know whether the transactions are actually valid or not. That is, how do we know that the provider agreed to provide the user with access to the genome data for the specified benefit and that the user agreed to pay the benefit?

Block chain systems use public key cryptography for digital signatures. Public key cryptography involves a pair of keys.

a) a public key. This key is shared publicly.
b) a private key. This key is only held by the owner. However, it can be used to create a digital signature that can be authenticated by anyone with the public key.

In messaging systems rather than adding a signature to the message itself, such as a genome sequence, the message is hashed. The hash is then signed by the sender with their private key. The sender then encrypts the message with the public key of the person they are sending it to. The receiver then uses the senders public key to validate the source of the message and decrypts the message sent to them with their own private key. This is simpler

in reality than it sounds. The central point of public key cryptology is that only the person holding the private key can decode a message encoded with the corresponding public key.

For the purpose of the blockchain as a ledger, what is visible is that A is entering into a transaction with B for the benefit X. The transaction can be authenticated using digital signatures.

We will illustrate this for a transaction involving a provider who gives a user access to a genome for the benefit of 100.

In the following code we:

a) create a public and private key using the common RSA algorithm
b) create a transaction involving a provider and user (an agreement that A will provide B with access to a marine genome for the benefit value of 100)
c) convert the transaction to bytes for easier encryption, storage and transmission (serialize)
d) create a hash of the bites in the transaction that is signed with the PROVIDERS private key
e) encrypt the message with the USERS public key that can be decrypted by the USERS private key

```
library(openssl)
# generate a private key (key) and a public key (pubkey)
key <- rsa_keygen()
pubkey <- key$pubkey
# create a transaction
trans = list(provider = "A", user = "B", genome_id = "marine metagenome",
benefit = "100")
# serialize data
data <- serialize(trans, NULL)
# sign (a hash of) the transaction with private key
sig <- signature_create(data, sha256, key = key)
# cipher the transaction with public key
cipher_message <- rsa_encrypt(data, pubkey)
```

If we view the signed transaction contained in the cipher message we just see a string of bytes like this.

```
cipher_message
```

```
##   [1] 36 6a 6b aa ad d9 ac 55 b5 7a 3f 7f 12 fb 98 28 4e 06 36 b4 ad 90 78
c7 70
##  [26] b0 dd b7 b3 a6 4b b9 e5 62 7f f9 ab fc af 80 04 2e 98 62 0a 42 17 e2
08 75
##  [51] ef dc e4 8a 6b 33 ad 90 4f 37 23 61 bb 58 18 a8 dd 71 58 b5 73 d5 36
f2 42
##  [76] 41 35 8e 76 18 76 0b 45 b1 e3 5c 23 9f bf 82 97 3c 00 db d6 32 14 7c
ba 8b
```

We now use the public key (pubkey) to check that the message is from who it claims to be from.

```
signature_verify(data, sig, sha256, pubkey = pubkey)
```

```
## [1] TRUE
```

We now decrypt the message and convert it from bytes back to something readable using the `private` key. Note that this example does not include the accession or data use categories used earlier.[2]

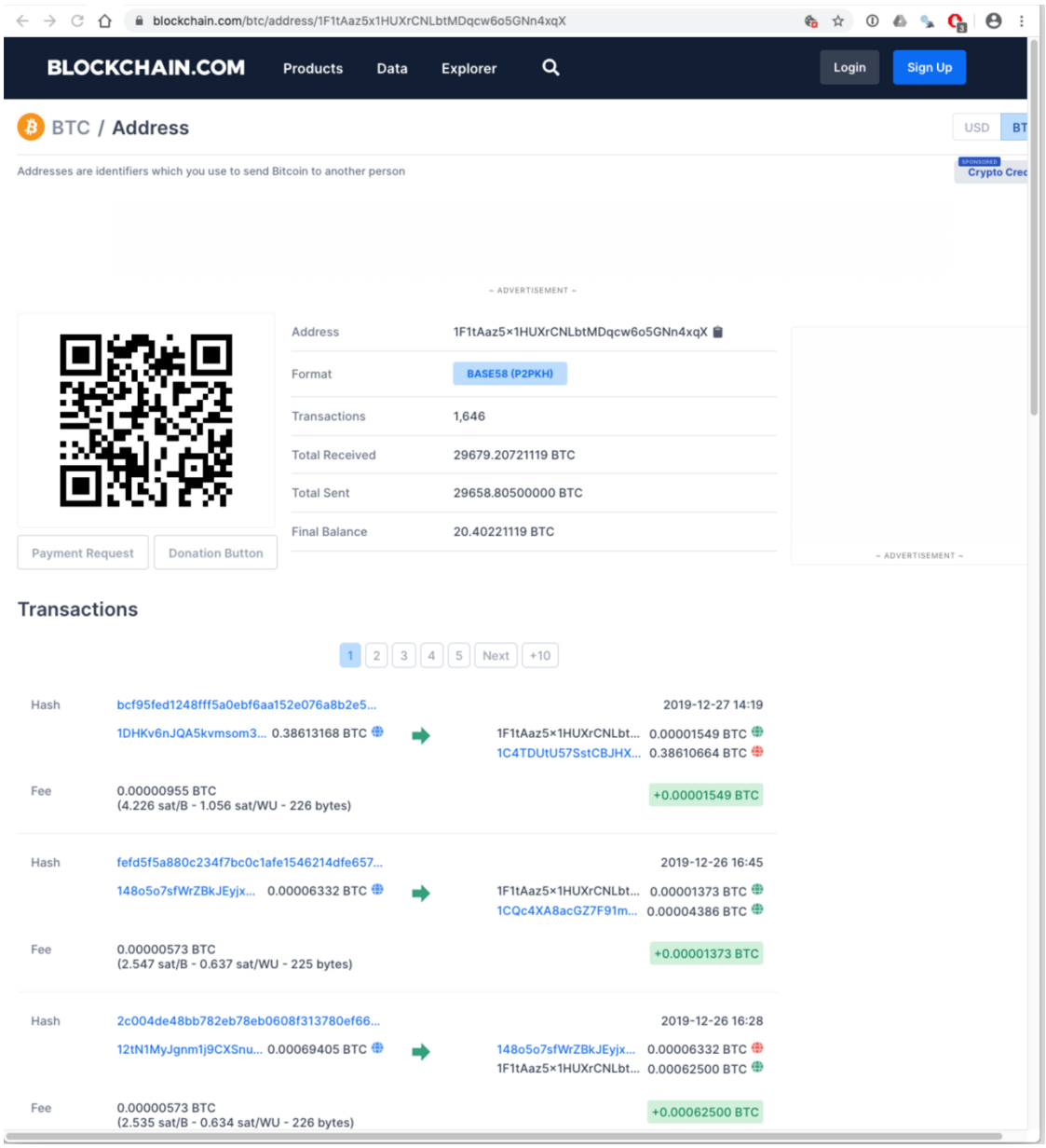
```
unserialize(rsa_decrypt(cipher_message, key))
```

```
## $provider
## [1] "A"
##
## $user
## [1] "B"
##
## $genome_id
## [1] "marine metagenome"
##
## $benefit
## [1] "100"
```

What is important here is that from the perspective of nodes in the blockchain is that what is visible in a block is a set of transactions that contain digital signatures that can be validated.

To finish off we are now in a position to make sense of a real world example using Bitcoin. Figure 1 shows transactions from a single bitcoin address.

[2] open ssl is not intended for long data fields but is convenient for use in creating examples.



In Figure 1 we can see that Bitcoin transactions are made up of sets of hashes that record transactions. In this case we see records for a specific address, details of the transfer and fees paid in the public ledger. However, an important innovation in the Bitcoin system is that users are represented by hashes. As Nakomoto 2008 explains:

"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. (Nakamoto 2008)

The privacy provisions of the Bitcoin blockchain are also what makes it so attractive to those operating outside the law. In the case of Bitcoin the public address (public key) of a

user is commonly regenerated for each transaction (meaning that a user cannot be tracked over time). However, these measures are not a necessary precondition for the use of blockchain technology. That is, those developing blockchain applications can decide on public and private aspects of a system such as the use of permanent ids that are linked to named institutions or persons and so on.

Conclusion

This paper has demonstrated how to build a biodiversity blockchain from scratch using R with code written by Massimo Franceschet. In practice an actual blockchain would be written in a language such as C++ but R helps us get to grips with the concepts involved in a step by step way.

References

Franceschet, Massimo. n.d. “Building a Blockchain in R.” http://users.dimi.uniud.it/~massimo.franceschet/HEX0x6C/blockchain/blockchainR.html.

Nakamoto, Satoshi. 2008. “Bitcoin: A Peer-to-Peer Electronic Cash System.” https://bitcoin.org/bitcoin.pdf.