Manual verification of Brabocoin hashes and signatures

Benne de Weger (TU/e)
b.m.m.d.weger@tue.nl

v0.3, November 3, 2021

1 Introduction

Brabocoin (https://brabocoin.org) is an educational cryptocurrency developed at TU/e by Dennis van den Berg, David Dekker, Sophie van den Eerenbeemt and Sten Wessel as their Bachelor Final Project [BDEW1]. One of the educational goals of Brabocoin is to enable Brabocoin users to verify for themselves the cryptographic aspects, so that they can build confidence in blockchain concepts at a technical level. That is why the Brabocoin GUI (Graphical User Interface) has the ability to show the user the raw data of transactions and blocks. To ease cryptographic verification and inspection of the contents of the Brabocoin blockchain, the *Brabocoin Viewer* and *Brabocoin Calculator* have been developed, but it still is not trivial how to use the three pieces of software together to do the verification succesfully. This document fills this gap.

2 Software Installation

The Brabocoin software (currently `brabocoin-0.4.1.exe` and `brabocoin-0.4.1.jar`) can be downloaded from `https://brabocoin.org/download`. It comes with its own user manual [BDEW2]. Note that for 'ordinary' users it only makes sense to use the GUI version, not the 'headless' version. On a Windows system the software should run without problems. It has been tested successfully on Linux and MAC but we've also encountered cases (of Linux) where we couldn't get it running (in particular on a Raspberry Pi), so you're on your own there: no support is provided beyond what's already in the manual. We strongly recommend that you just install `brabocoin-0.4.1.exe` on Windows.

Note that the first time you start up Brabocoin a lot of data has to be fetched, and this will take some time. Make a note of the location of the Brabocoin database, which on a Windows system probably is something like
`C:\Users\USERNAME\AppData\Local\Brabocoin\app\data\1\blocks\blk0.dat`.

The Brabocoin Viewer software (currently `BCVv1.0.exe` and `BCVv1.0.jar`) and the Brabocoin Calculator software (currently `BCCv1.0.exe` and `BCCv1.0.jar`) can be downloaded from `https://brabocoin.org/bcc`. They should run on any system that has a not too old java version installed.

The Brabocoin Viewer works best if two other files are present in the directory where the `BCCv1.0` program resides:

- `BrabocoinViewer.cfg`, in which the location of the Brabocoin database should be given

in the following format:
`[filename] C:\Users\USERNAME\AppData\Local\Brabocoin\app\data\1\blocks\blk0.dat`

- `BrabocoinViewerOwnerAddressList.txt`, in which a list of Brabocoin addresses and owner names can be provided, in the following format:
  `ADDRESS1;NAME1`
  `ADDRESS2;NAME2`
  etc.

Examples of these two files can be downloaded from `https://brabocoin.org/bcc`. You can edit both files in a simple ASCII text editor such as Notepad. In the file `BrabocoinViewer.cfg` replace the file name by the location of the actual Brabocoin database on your system; probably you only have to replace `USERNAME` by your own Windows username.

3 Block 1947

As an example throughout this document we will be looking at block 1947 in the current Brabocoin blockchain.

In the Brabocoin GUI, go to “Current state”, tab “Blockchain”, scroll down to “Height” 1947, and click on the line to select it. In the Brabocoin Viewer, manually enter “1947” in the block height field of the Block View. See Figures 1 and 2.

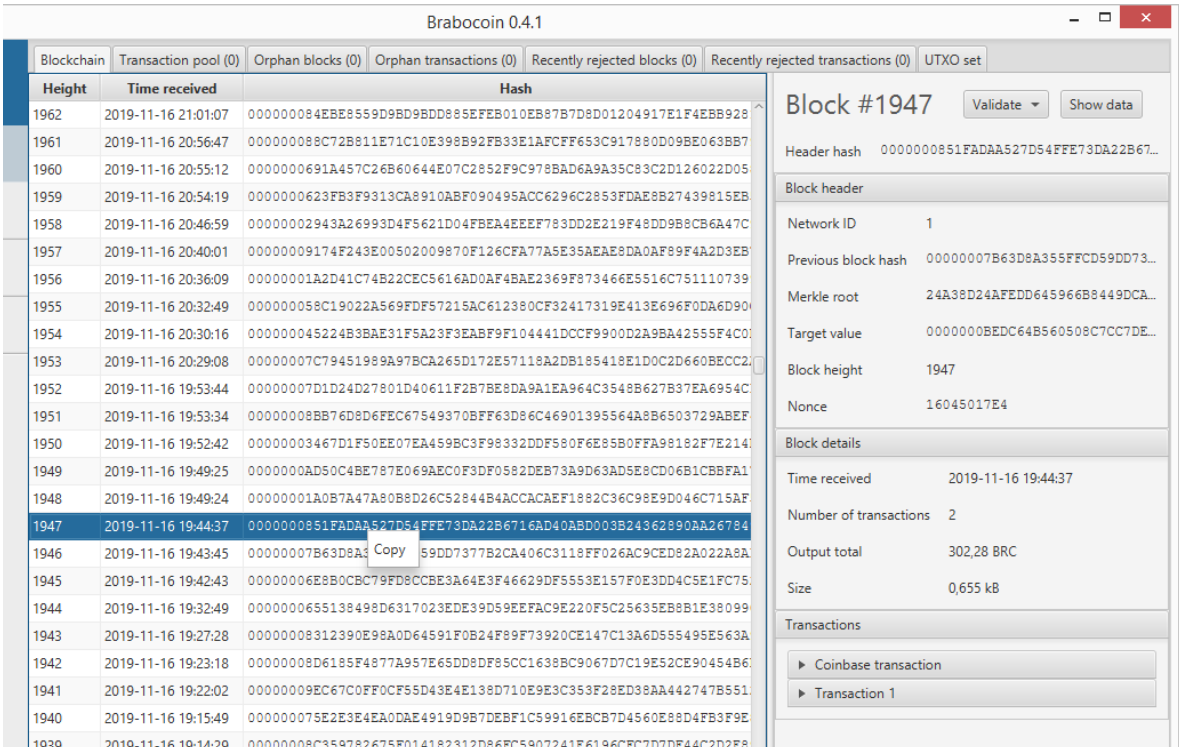

Figure 1: Block 1947 as shown in the Brabocoin GUI.

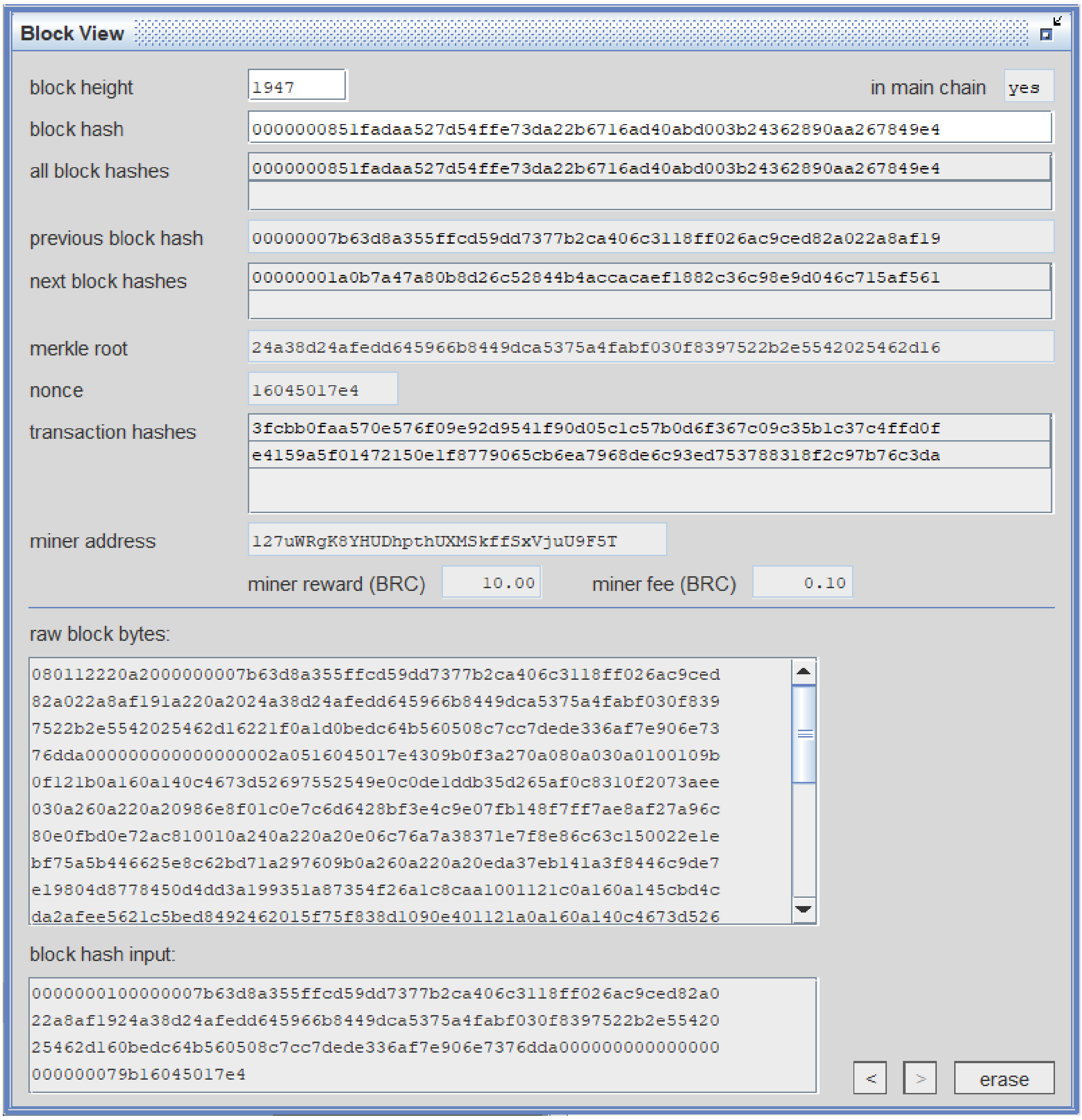

Figure 2: Block 1947 as shown in the Brabocoin Viewer.

You should note the following:

- All "Hash" values start with 7 zeroes. More accurately, they are all smaller than the "Target value" starting `0000000BEDC64B....` This is the result of the mining procedure.
- The "Hash" of the block (also called "Header hash" or "Block header hash") is `0000000851FADAA527D54FFE73DA22B6716AD40ABD003B24362890AA267849E4`.
  You can copy it from the Brabocoin GUI to your computer's clipboard by right-clicking the selected "Hash" field and then clicking "Copy" (as shown in Figure 1). Also you can copy it from the Brabocoin Viewer to your computer's clipboard by right-clicking

the “block hash” field and then clicking “Copy” (as not shown in Figure 2). This copy feature actually is implemented for many fields in the GUI windows of the two programs.

- The “Previous block hash” starting `00000007B63D8A3....` is the same as the “Hash” of block 1946. This is the chaining idea of the blockchain.
- There is a “Merkle root” starting `24A38D24....`
- There is a “Nonce” with value `16045017E4`.
- There are two transactions: a “Coinbase transaction” and a normal “Transaction 1”. The Brabocoin Viewer shows the transaction hashes, the coinbase transaction is always the top one.

Next click “Show data”. This will bring up a window, see Figure 3, that contains the actual raw block data in two formats: JSON and raw hex. JSON is a more human readable format of the raw data, which shows the structure in there. It is clear that there are the following fields: `networkId`, `previousBlockHash`, `merkleRoot`, `targetValue`, `nonce`, `blockHeight`, `transactions`.

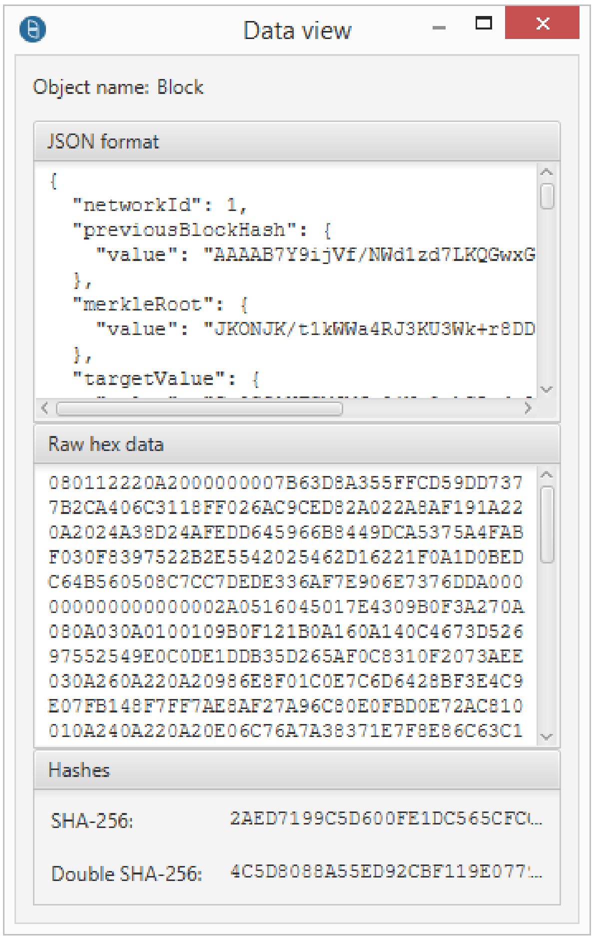

Figure 3: Block 1947 data.

```
080112220A2000000007B63D8A355FFCD59DD7377B2CA406C3118FF026AC9CED
82A022A8AF191A220A2024A38D24AFEDD645966B8449DCA5375A4FABF030F839
7522B2E5542025462D16221F0A1D0BEDC64B560508C7CC7DEDE336AF7E906E73
76DDA000000000000000002A0516045017E4309B0F3A270A080A030A0100109B
0F121B0A160A140C4673D52697552549E0C0DE1DDB35D265AF0C8310F2073AEE
030A260A220A20986E8F01C0E7C6D6428BF3E4C9E07FB148F7FF7AE8AF27A96C
80E0FBD0E72AC810010A240A220A20E06C76A7A38371E7F8E86C63C150022E1E
BF75A5B446625E8C62BD71A297609B0A260A220A20EDA37EB141A3F8446C9DE7
E19804D8778450D4DD3A199351A87354F26A1C8CAA1001121C0A160A145CBD4C
DA2AFEE5621C5BED8492462015F75F838D1090E401121A0A160A140C4673D526
97552549E0C0DE1DDB35D265AF0C8310121A670A2001FA536CD6E3114C157E94
154DCE3511CC243E4D8AC154DA81595BB48C6EE92912204A4C574A86167D9281
C4EEFE1DBA822E924F81BE11582EDEB6F2B0C3F544C93D1A2102714FB63AC085
407F64740CF01B3DFEC171A23E86971C840512FC6BF8341D06521A680A21008B
6F457EBD9B5ABFD5D00ABD73EEA1F2BD71B4E977C624FA5DD403811A523E0F12
2000AF4D6BF956B5E896FA67A3375657FA4D996374183171B5EBEEA4E702AD06
C11A2102714FB63AC085407F64740CF01B3DFEC171A23E86971C840512FC6BF8
341D06521A690A2100DAD20F9897078815D2DFFED5ECE139C14DC404BE66C20F
87E8A8EC07F0BF1FE0122100E09A38912620430BBC4DFF7D89CB6CB325D74600
5157B5454EE8EDBCA8DCA3651A2102714FB63AC085407F64740CF01B3DFEC171
A23E86971C840512FC6BF8341D0652
```


Table 1: Raw hex data of block 1947 as shown in the Brabocoin GUI; blue is, sort of, the Block header, red is the Coinbase transaction, and green is Transaction 1.

We say in the capture of Table 1 that the blue data is "sort of" the block header. The true block header that serves as input to the hash operation producing the block hash is slightly different. See Section 6.1 for details.

Many values are given in Base64 encoding, we will not bother you with that[1] or with details of JSON. It is more important that we look at the raw hex data structure, as that is where the input for hashes and digital signatures comes from. This raw hex data is presented in Table 1.

For now it is important to see that the block is split into the "Block header" part which consists of the first 6 fields, and the "Transactions" part.

4 Block header data

The Block header of block 1947 consists of the blue bytes in Table 1, we will explain this in detail. Raw hex means that every byte (8 bits) is denoted by one *octet*[2], consisting of two "hex" characters from Table 2.

hex	bits	hex	bits	hex	bits	hex	bits
0	0000	4	0100	8	1000	C, c	1100
1	0001	5	0101	9	1001	D, d	1101
2	0010	6	0110	A, a	1010	E, e	1110
3	0011	7	0111	B, b	1011	F, f	1111

Table 2: Hexadecimal characters; the Brabocoin GUI uses uppercase, the Brabocoin Calculator uses lowercase.

[1] The Brabocoin Calculator has a built-in Base64 encoder and decoder.

[2] Actually, *octet* can be taken as a synonym for *byte*; we will use the word "byte" further on.

For example, the last two bytes of the header: `9B0F`, consists of two octets: `9B`, `0F`, and should be read as the bits `1001 1011 0000 1111`.

We split up the header into the 6 fields, and highlight more about the structure in Table 3.

field	key	length	data
`networkId`	08		01
`previousBlockHash`	12	22	0A2000000007B63D8A355FFCD59DD7377B 2CA406C3118FF026AC9CED82A022A8AF19
`merkleRoot`	1A	22	0A2024A38D24AFEDD645966B8449DCA537 5A4FABF030F8397522B2E5542025462D16
`targetValue`	22	1F	0A1D0BEDC64B560508C7CC7DEDE336AF7E 906E7376DDA0000000000000000
`nonce`	2A	05	16045017E4
`blockHeight`	30		9B0F

Table 3: The Block header of block 1947 dissected, blue is a hash value (though the `targetValue` here misses the leftmost 6 zero values).

The structure in this data is according to Google's "proto3" encoding. We only give relevant details here and will not describe the full proto3 standard[3]. See Section 10 for a few more relevant details of proto3 you might find interesting if you're a real nerd[4].

The first byte of each field is a so called *key*[5]. It can be seen as an indication for the *meaning* of the data following it, namely networkID, etc. It also indicates the *type* of the data that follows. Two types are present here: the keys `08` and `30` are followed by data of the type "varint", while the keys `12`, `1A`, `22` and `2A` are followed by data of the type "length-delimited".

A *varint* is an encoding of an integer. For example, `01` stands for the integer 1, and `9B0F` stands for the integer 1947. See Section 10.2 if you want to really understand this.

A *length-delimited* structure starts with a varint that represents the bytelength of the following object. For example: the `nonce` field in block 1947 is `2A0516045017E4`, here the first byte is the key, telling that this is the nonce, and that the following structure is length-delimited, so we expect next a varint, and its value is 5. So the next 5 bytes form the nonce.

The `previousBlockHash` field has a length of 34 bytes. Those 34 bytes again have a proto3 structure, so the first byte `0A` is a key, here meaning "byte array", length-delimited. So the next byte `20` is a varint with value 32, and the following 32 bytes are the actual value of the hash of the previous block. This you can check in Figure 1 at block 1946.

The `merkleRoot` field has the same structure. See Section 6.3 for how to verify it.

The `targetValue` field contains the upper bound for the Brabocoin hash puzzle. Here the 3 leading zero bytes have been left out, because this hash value is treated as an integer in comparing it with actual block hashes. The target is the same in all blocks, its value is 3216×10^{65}, see [BDEW1].

[3]See `https://developers.google.com/protocol-buffers/docs/overview`.

[4]There's nothing wrong with being a nerd. I'm one myself.

[5]Not to be confused with a cryptographic key...

5 Transaction data

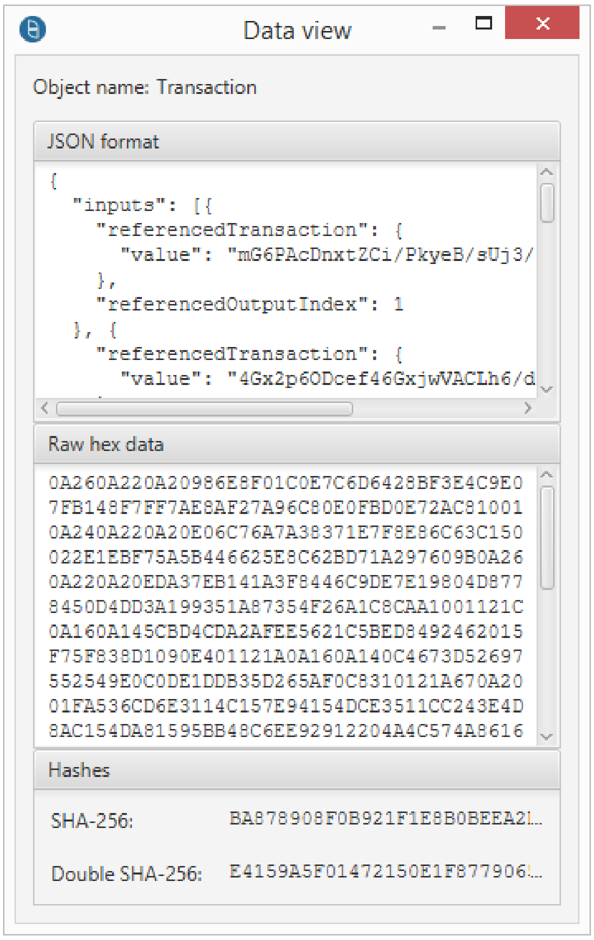

Figure 4: Block 1947 Transaction 1 data as shown in the Brabocoin GUI.

In the Brabocoin GUI, both on the "Blockchain" tab in the "Current state" page, and on the "Transaction history" tab on the "Wallet" page you can access transactions. There is a button "Show data", that lets you choose between unsigned and signed (the difference is whether the signatures are included or not). See Figure 4 for a signed transaction. You can also find transactions in the blocks. In the Brabocoin Viewer, use the "Transaction View", see Figure 5 for the same transaction.

Note that the transaction raw data do not include the proto3 key and length fields.

In block 1947 two transactions are present, filling up the entire remainder of the block after the header. A transaction is a length-delimited structure beginning with the key `3A`. In the block's raw data, after the block header ending in `9B0F`, you see `3A27`, indicating the start of a transaction (key `3A`) of byte length 39 (varint `27`). This is the coinbase transaction. Immediately after these 39 bytes you see `3AEE03`, that's where the second transaction starts. The first bytes are `3AEE03`, here `3A` is the key, and `EE03` is a varint, representing the number 494. Indeed the remainder of the block has 494 bytes.

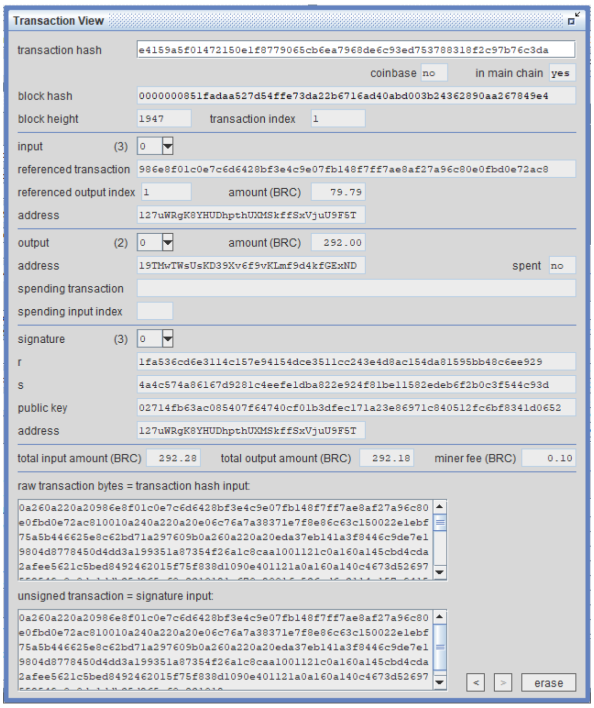

Figure 5: Block 1947 Transaction 1 data as shown in the Brabocoin Viewer.

5.1 Normal transaction

We first look at the second, "normal" transaction, i.e. block 1947's Transaction 1, see Figures 4 and 5. A transaction consists of a number of inputs, a number of outputs, and for each input a signature. So it should not come as a surprise to observe in the raw data what is shown in Table 4.

field	key	length	data
`input`	0A	26	0A220A20...2AC81001
`input`	0A	24	0A220A20...A297609B
`input`	0A	26	0A220A20...8CAA1001
`output`	12	1C	0A160A14...1090E401
`output`	12	1A	0A160A14...0C831012
`signature`	1A	67	0A2001FA...341D0652
`signature`	1A	68	0A21008B...341D0652
`signature`	1A	69	0A2100DA...341D0652

Table 4: The structure of Block 1947 Transaction 1.

Apparently here the keys are `0A`, `12`, `1A`, standing for `input`, `output` and `signature` respectively. So meanings of keys depend on the context, see Section 10.1.

The structure of a `input` field, here shown for the first input in the example, is in Table 5. The hash value is a transaction hash, used as an identifier. The index (a varint) is the output index in the referred transaction. If there is no index field (as in the second input), the index value should be taken 0.

field	key	length	data
transaction	0A	22	0A20986E8F01C0E7C6D6428BF3E4C9E07F B148F7FF7AE8AF27A96C80E0FBD0E72AC8
index	10		01

Table 5: An input in Block 1947 Transaction 1; blue is a hash.

The structure of an `output` field, here shown for the first output in the example, is in Table 6. The address is given in Base58 encoding. The amount is in Brabocoin cents, here `90E401` encodes the number 29200, see Section 10.2.

field	key	length	data
address	0A	16	0A145CBD4CDA2AFEE5621C5BED8492462015F75F838D
amount	10		90E401

Table 6: An output in Block 1947 Transaction 1; blue is a Brabocoin address.

The structure of an `signature` field, here shown for the first signature in the example, is in Table 7. The first two values are r and s, the third is the public key, in compressed form. Here the integers are not in varint format but in raw two's complement byte format[6].

field	key	length	data
r	0A	20	01FA536CD6E3114C157E94154DCE3511 CC243E4D8AC154DA81595BB48C6EE929
s	12	20	4A4C574A86167D9281C4EEFE1DBA822E 924F81BE11582EDEB6F2B0C3F544C93D
pubKey	1A	21	02714FB63AC085407F64740CF01B3DFEC1 71A23E86971C840512FC6BF8341D0652

Table 7: A signature in Block 1947 Transaction 1.

Note the meaning of the keys here. Also note that the three signatures were created by the same private key, as the three signatures have identical public keys (it is possible that signatures in one transaction have different public keys).

FInally note that the Brabocoin Viewer may omit leading zeroes in the r and s fields, as it sees those values as numbers. For numerical verification this does not matter.

5.2 Coinbase transaction

A Coinbase transaction is a lot simpler than a normal transaction, as there are no inputs and thus also no signatures, and only one output. Actually there is an input field there, but its hash field just consists of one zero byte (signalling that this is a coinbase transaction), and for the index the block height is given. The output address is the address of the miner, and the amount (in the example BRC 10.10) collects both the mining fee and the transaction fee.

[6]For positive numbers this means that the first bit of the first byte should be a `0`, if necessary a complete zero byte is added at the front.

6 Verifying hashes

6.1 Block hash

The way the Block hash is computed in Brabocoin does not use the proto3 structure but concatenates the values of the fields as follows:

- the networkID value, in a 4 byte format, in the presently running Brabocoin system this is always `00000001`;
- the previousBlockHash, the first blue value in Table 3;
- the merkleRoot value, the second blue value in Table 3;
- the targetValue, the third blue value in Table 3;
- the blockHeight value in a 4 byte format, for 1947 this is `0000079B`;
- the nonce value, in two's complement format, see Table 3.

Note that blockHeight and nonce have been swapped. The resulting input for the Double SHA256 Block hash for block 1947 is given in Table 8. Note that it is identical to the contents of the field "block hash input" in the "Block View" of the Brabocoin Viewer. For this input the SHA256 is
`7115a11a3fe28c913cb7e62a0933fb3b2a9db67b1d275b8c11973aad1d9da21f`,
and the SHA256 of this SHA256 value is
`0000000851fadaa527d54ffe73da22b6716ad40abd003b24362890aa267849e4`.

```
0000000100000007B63D8A355FFCD59DD7377B2CA406C3118FF026AC9CED82A0
22A8AF1924A38D24AFEDD645966B8449DCA5375A4FABF030F8397522B2E55420
25462D160BEDC64B560508C7CC7DEDE336AF7E906E7376DDA000000000000000
000000079B16045017E4
```

Table 8: Input for the Block hash computation of block 1947.

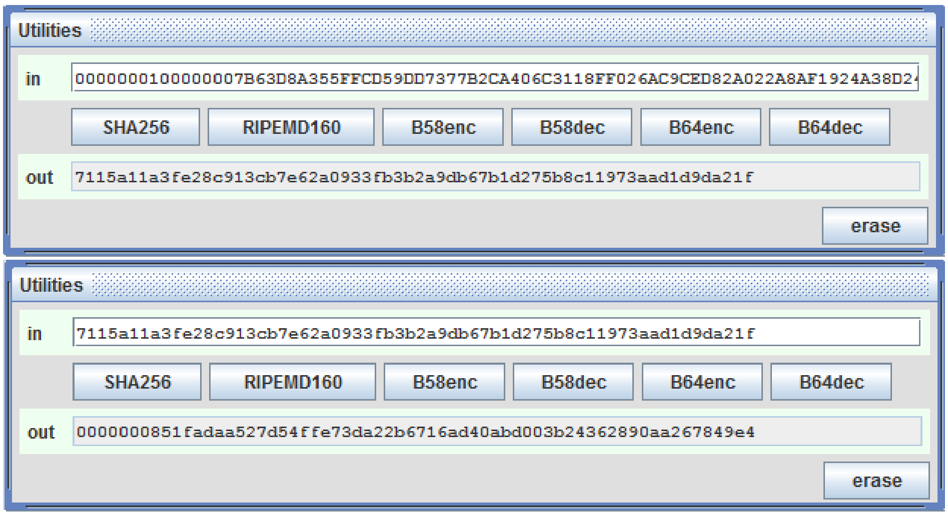

Figure 6: The hash puzzle solved for block 1947.

Here the miracle has happened: this is a hash that, seen as an integer, is indeed below the target value. See Figure 6.

6.2 Transaction hash

A Transaction hash simply is a double SHA256 hash over the signed transaction data, i.e. the transaction raw bytes, including the signatures. For Transaction 1 of block 1947 the input to this computation thus is given as the green data in Table 1, but without the key and length fields, so starting with `0A260A22...`, and then all the way to the final `...341D0652`. Note that this is identical to the contents of the field "transaction hash input" in the "Transaction View" of the Brabocoin Viewer. Feed this data as one long hex-string into the Brabocoin Calculator's "in" field in the "Utilities" window, and click "SHA256", this gives
`ba878908f0b921f1e8b0beea2d5a4c5ff80213d69fe448e6493dcaba16191f35`.
Then feed this value again to the "in" field and again click "SHA256", this gives
`e4159a5f01472150e1f8779065cb6ea7968de6c93ed753788318f2c97b76c3da`.
This is indeed the Transaction hash as shown in the Brabocoin GUI, and also in Figure 4.

6.3 Merkle root

The Merkle root computation follows a Merkle tree structure, see Figure 7 below for an example with 11 transactions. The transaction hashes are at the bottom row, and hashes then are combined two at a time onto a higher level row in the tree, where the last element of a row with an odd number of elements simply is copied to the row above (in contrast to Bitcoin, where in such a case the last element is repeated). This is repeated until only one hash is left: the Merkle root at the top of the tree.

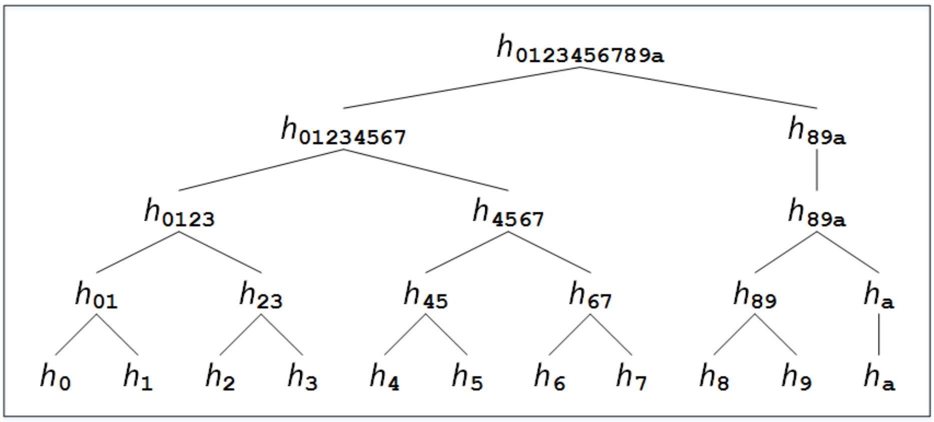

Figure 7: The Merkle tree for a block with 11 transactions, $h_0, \ldots, h_{\mathtt{a}}$ on the bottom row are the 11 transaction hashes, and going up one row is done by the rule $h_{\mathtt{xy}} =$ SHA256(SHA256($h_{\mathtt{x}} \| h_{\mathtt{y}}$)) for any strings $\mathtt{x}, \mathtt{y}$, where $\|$ denotes concatenation of hashes.

In the case of block 1947 there are only two transaction hashes, for the Coinbase transaction: `3fcbb0faa570e576f09e92d9541f90d05c1c57b0d6f367c09c35b1c37c4ffd0f`, and for Transaction 1: `e4159a5f01472150e1f8779065cb6ea7968de6c93ed753788318f2c97b76c3da`. The Merkle root can easily be verified using the Brabocoin Calculator: the SHA256 of the SHA256

of `3fcbb0fa...7c4ffd0fe4159a5f...7b76c3da` is
`24a38d24afedd645966b8449dca5375a4fabf030f8397522b2e5542025462d16`,
which is exactly the value in the `merkleRoot` field.

7 Verifying signatures

The input for computing the signature on an input actually is the complete unsigned transaction, i.e. the transaction without the signatures. For block 1947 this input data is given in Table 9, see also the "signature input" field in the "Transaction View" of the Brabocoin Viewer, and this can be easily verified with Tables 4 and 1.

```
0A260A220A20986E8F01C0E7C6D6428BF3E4C9E07FB148F7FF7AE8AF27A96C80
E0FBD0E72AC810010A240A220A20E06C76A7A38371E7F8E86C63C150022E1EBF
75A5B446625E8C62BD71A297609B0A260A220A20EDA37EB141A3F8446C9DE7E1
9804D8778450D4DD3A199351A87354F26A1C8CAA1001121C0A160A145CBD4CDA
2AFEE5621C5BED8492462015F75F838D1090E401121A0A160A140C4673D52697
552549E0C0DE1DDB35D265AF0C831012
```

Table 9: Input for the double hash computation of Block 1947 Transaction 1.

Feed this data as one long hex-string into the Brabocoin Calculator's "in" field in the "Utilities" window, and click "SHA256", this gives
`78943053a4cc71b52ede329f06efe2db7c4f80fd8f92c10fd90f1d8c605eeb33`.
Then feed this value again to the "in" field and again click "SHA256", this gives
`f43ab8e4ed4143af1b3cbd2e11b34b98a15aac9ff37aaf0ddb9e1f19c991eefd`.
This is the Double SHA256 hash value that is used as input for the signature generation and verification. Note that for all signatures the same data is signed. It is however possible that for different inputs different keys are used, so indeed for each input there should be a separate signature.
For the verification of the first signature of block 1947 we now have data as in Table 10, these data can also be seen in the signature block of the "Transaction View" of the Brabocoin Viewer (except for the hash).

h	`f43ab8e4ed4143af1b3cbd2e11b34b98a15aac9ff37aaf0ddb9e1f19c991eefd`
r	`01FA536CD6E3114C157E94154DCE3511CC243E4D8AC154DA81595BB48C6EE929`
s	`4A4C574A86167D9281C4EEFE1DBA822E924F81BE11582EDEB6F2B0C3F544C93D`
Q	`02714FB63AC085407F64740CF01B3DFEC171A23E86971C840512FC6BF8341D0652`

Table 10: ECDSA verification data for the input in Table 9; Q is the public key.

The values of h, r and s are just the hexadecimal representations of the numbers. The public key is in *compressed form*, this means the following: if the point on the Brabocoin curve has coordinates (x, y), then for given x there are only 2 possibilities for y (the so called *square roots* of $x^3 + 7$), if the one is y_1 then the other is $p - y_1$. One of these is even, the other one is odd. The compressed form of the point (x, y) then consists of the bytes of x, preceded with `02` if y is even, and with `03` if y is odd. The Brabocoin Calculator can switch between compressed and uncompressed form (but only when hexadecimal representation is chosen). There exists an efficient algorithm for computing square roots modulo p.

Further the Brabocoin parameters are needed, they are p, n and the generator G, these parameters are available in the Brabocoin Calculator.
Verification of the signature now is the following (see [BDEW1], or [dW]):

- compute $u_1 = hs^{-1} \pmod{n}$,
- compute $u_2 = rs^{-1} \pmod{n}$,
- compute the point $R = u_1G + u_2Q$,
- verify that its x-coordinate is equal to r.

We show how to do this with the Brabocoin Calculator. Make sure it is in hexadecimal and compressed mode.

- computation of u_1:
  - enter s in the field **b** of the "Modular Calculator",
  - enter h in the field **a**,
  - copy n to the field **m**,
  - click "**a/b**", then the field **c** contains u_1, move it to the field **u** on the "Clipboard",
- computation of u_2:
  - leave s in the field **b**, and leave n in the field **m**,
  - enter r in the field **a**,
  - click "**a/b**", then the field **c** contains u_2, move it to the field **v** on the "Clipboard",
- computation of u_1G:
  - copy G to the field **A** in the "Elliptic Curve Calculator",
  - copy u_1 from the field **u** to the field **k**,
  - click "**kA**", then the field **C** contains u_1G, move it to the field **P** on the "Clipboard",
- computation of u_2Q:
  - enter the public key Q in the field **A** in the "Brabocoin Elliptic Curve Calculator",
  - copy u_2 from the field **v** to the field **k**,
  - click "**kA**", then the field **C** contains u_2Q, move it to the field **Q** on the "Clipboard",
- computation of $R = u_1G + u_2Q$:
  - copy u_1G from the field **P** to the field **A**,
  - copy u_2Q from the field **Q** to the field **B**,
  - click "**A+B**", then the field **C** contains R.

In the field **C** there now is a compressed point. Disregard its first byte, the remaining bytes form the x-coordinate, that should be equal to r (which still might be visible in the field **a**).
In Table 11 we show the values found for the first signature in block 1947.

u_1	cec1a76523eb8ff4e56ab2e992c1dd873e38b52989f237894f0922c4c24751ee
u_2	66b68c471c6b963469db3a436057f2136b52ecce3ff81d4829544fd459bf4d73
u_1G	0310b65f3aa45c8659faa8ba4740b9264b6c550096e6c422a4e17a86c47488d765
u_2Q	02d872fe0cbff1ef0627194980876465 2cd0f4eb070487ac738d297b36948d6e81
R	021fa536cd6e3114c157e94154dce3511cc243e4d8ac154da81595bb48c6ee929

Table 11: Verification results for the signature in Table 10.

Indeed this signature verifies correctly.

8 Verifying public keys

The previous section only treated the cryptographic verification of one signature, but assumed that the proper public key was used. For real-life verification this assumption has to be verified as well. This means that one should check that the public key used in the signature verification is the same as the public key that is behind the Brabocoin address that was used in the inputs.

The public key coming with the signatures is the Q given in Table 10 as
02714FB63AC085407F64740CF01B3DFEC171A23E86971C840512FC6BF8341D0652.
In the transaction inputs however the Bitcoin address receiving the money is not directly given. Only the transaction hash and an index are given there. So we should look up this transaction and look at its details.

The transaction hash in the first input is shown in Table 5:
986E8F01C0E7C6D6428BF3E4C9E07FB148F7FF7AE8AF27A96C80E0FBD0E72AC8.
Note that its transaction index is also shown, it is 1. In the Brabocoin GUI it is not clear what the amount of this input is. It is in general cumbersome to find a transaction somewhere in the blockchain. This is where the Brabocoin Viewer comes in very handy, because when it reads the database at startup, it already does a lot of additional bookkeeping that now turns out to be very useful. In the input block it already shows the amount of BRC 79.79. Now, double clicking on the "referenced transaction hash" immediately shows in the "Transaction View" this transaction, which turns out to be in block 1946. Select the output with index 1, which is indeed for the amount of BRC 79.79, and it is to address 127uWRgK8YHUDhpthUXMSkffSxVjuU9F5T. The Brabpcoin Viewer even shows in which transaction it has been spent, which the Brabocoin GUI would not be able to show easily.

Brabocoin addresses are computed as Base58(RIPEMD160(SHA256(pubKey))), where the public key is in compressed form. This is easily done with the Brabocoin Calculator, The results for the public key Q of Table 10:
SHA256: 600fa825dc12f6d063444536336529b4fdef528f0c01709efc542bb53c4fa0c5.
RIPEMD: 0c4673d52697552549e0c0de1ddb35d265af0c83.
Base58: 127uWRgK8YHUDhpthUXMSkffSxVjuU9F5T.
So this indeed confirms that the spender of this transaction is entitled to the claimed inputs.

Let's also check the output addresses. The first, Coinbase transaction, has as output address
0c4673d52697552549e0c0de1ddb35d265af0c83.
Note that this is exactly the RIPEMD(SHA256(Q)) as found above. So this is the address of the miner of block 1947, as the mining and transaction fees go to the miner.

The second transaction, Transaction 1, has two outputs, the first one having address
5CBD4CDA2AFEE5621C5BED8492462015F75F838D,

and the second one having address
`0c4673d52697552549e0c0de1ddb35d265af0c83`.
This second output address is where the change money goes to; this happens to be the miner's address, so this transaction originates from the miner himself. To find the Brabocoin address of the recipient of the transaction money, compute the Base58 of the output address:
Base58(`5CBD...838D`) = `19TMwTWsUsKD39Xv6f9vKLmf9d4kfGExND`.

9 Verifying amounts

Let us compare inputs and outputs in the Transaction 1 of block 1947 (the Brabocoin Viewer gives this information immediately, but it's useful to ckeck it by hand). There are 3 inputs[7].

Input 0 has transaction hash `986E....` This transaction can be found in Block 1946 as Transaction 2. We need the output with index 1. This has address `127u....` and an amount of BRC 79.79.

Input 1 has transaction hash `E06C....` This transaction can be found in Block 1945 as Transaction 7. We need the output with index 0. This has address `127u....` and an amount of BRC 200.00.

Input 2 has transaction hash `EDA3....` This transaction can be found in Block 1946 as Transaction 1. We need the output with index 1. This has address `127u....` and an amount of BRC 12.49.

So the total input amount is BRC 292.28.

Transaction 1 in block 1947 has two outputs.

Output 0 is to address `5CBD...` for an amount of BRC 292.00.

Output 1 is to address `0c46...` for an amount of BRC 0.18.

So the total output amount is BRC 292.18.

The difference between input and output amounts is exactly the transaction fee of BRC 0.10 that can be found in the Coinbase transaction and that in this way is collected by the miner.

So this is what happened: the owner of address `0c46...` (a.k.a. `127u....`) wanted to pay BRC 292.00 to address `5CBD...` (a.k.a. `19TM....`), and he allowed a transaction fee of BRC 0.10. He collected from his unspent coins enough to get an amount at least as big as BRC 292.10; the three inputs he found happened to add up to 292.28. So this is why the transaction has a second output of 0.18 to his own address, i.e. the change money.

Note that blockchains work fundamentally different from bank accounts. A user receiving some coins in a transaction knows this because his address is mentioned in an output. But there is no direct correspondence with one (or a few) input(s): the transaction may have a number of inputs signed by different private keys, and the money from those inputs may be divided somehow over a number of outputs to different addresses. In block 1947 the inputs all came from the same address, so there the origin of the money is at least somewhat clear.

[7] Note that computer scientists start counting at 0, which is a horrible habit.

10 Proto3 details

10.1 key

We found as relevant keys: `08`, `0A`, `10`, `12`, `1A`, `22`, `2A`, `30`, `3A`, and their meaning depends on the context. Here is how this works: the first 5 bits of a key give an index, the last three bits give a type. Type 0 means varint, type 2 means length-delimited. See Tables 12 and 13. The index simply refers to the fields in the data structure that is being used at that moment.

key	index	type
08	1	0
0A	1	2
10	2	0

key	index	type
12	2	2
1A	3	2
22	4	2

key	index	type
2A	5	2
30	6	0
3A	7	2

Table 12: Proto3 keys.

object type: Block		
index	object name	object type
1	`networkID`	varint
2	`previousBlockHash`	Hash
3	`merkleRoot`	Hash
4	`targetValue`	Hash
5	`nonce`	bytes
6	`blockHeight`	varint
7	`transactions` (repeated)	Transaction

object type: Transaction		
index	object name	object type
1	`input` (repeated)	Input
2	`output` (repeated)	Output
3	`signature` (repeated)	Signature

object type: Hash		
index	object name	object type
1	`value`	bytes

object type: Input		
index	object name	object type
1	`referencedTransaction`	Hash
2	`referencedOutputIndex`	varint

object type: Output		
index	object name	object type
1	`address`	Hash
2	`amount`	varint

object type: Signature		
index	object name	object type
1	`r`	bytes
2	`s`	bytes
3	`publicKey`	bytes

Table 13: Proto3 data structures in Brabocoin.

10.2 varint

The proto3 encoding of nonnegative integers works as follows (we do not treat negative integers here as it's not relevant for Brabocoin). First write the integer out in binary, i.e. in bits. Add as many zero bits, but at least one[8], to the front as needed to make the total number of bits a multiple of 7. Group the bits into 7-tuples, add a `0` to the front of the first 7-tuple, add a `1` to the front of every other 7-tuple, so that you get full bytes. Then reverse the list of bytes[9].

Here's an example: 1947 is in binary `11110011011`, so we add three `0`'s to the front: `00011110011011`, split into 7-tuples: `0001111 0011011`, add a `0` and a `1` to get bytes: `00001111 10011011`, reverse the order: `10011011 00001111`, and translate to raw hex: `9B0F`. We've seen that one before.

[8]This is important, as the *two's complement* representation is used.

[9]Software engineers say: convert from *big endian* to *little endian*. Endianness problems are responsible for many disasters that have plagued (the computing part of) mankind in many ways over the last 60 years.

Here's an other example, where we do the conversion backwards, from proto3 varint to normal integer. In block 1947, there is (the underlined bytes in Table 1) the bytes `...1090E40112....` Here `10` is a key indicating that a varint follows, and as this is not a length-delimited type, we do not know a priori how many bytes this varint structure has. Indeed, that is exactly the reason for having added the 8th front bit to every 7-tuple: scan the following bytes for their first bit only; as soon as a first bit `0` is encountered, that must be the final byte of the varint. In this case the bytes `90` and `E4` both start with a bit `1`, and the next byte `01` starts with a bit `0`, so the varint is actually `90E401`, and the next bytes `12...` belong to the next structure in the block. We write `90E401` in bits: `10010000 11100100 00000001`, reverse the order: `00000001 11100100 10010000`, remove the first bit in every byte: `0000001 1100100 0010000`, remove the leading zeroes: `111001000010000`, and that is the binary representation of our positive integer, which is 29200.

11 Conclusion

This document shows that it still quite a hassle to understand, at byte level, how hashes and digital signatures are used in practice. As both cryptographic tools are extremely sensitive (and should be!) for changes in the inputs, an important part in building trust in cryptographic software is the absolutely correct handling of input data. Preferably cryptographic verification should be done using software that has been developed completely independently from the software under verification[10]. This is what we wanted to show in this document.

We need your feedback

We very much appreciate any questions or feedback you might have, on the Brabocoin software, the Brabocoin Calculator, and on this document.

Acknowledgements

I am grateful to Dennis, David, Sophie and Sten for, in the first place, having developed Brabocoin, and also for their technical help in getting Brabocoin running on Linux, and in writing this document.

References

[BDEW1] D.P. van den Berg, D.J.C. Dekker, S. van den Eerenbeemt and S. Wessel, "Brabocoin, an educational cryptocurrency based on Bitcoin", Bachelor Thesis, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, May 13, 2019. Online: `https://brabocoin.org/doc/res/report.pdf`.

[BDEW2] D.P. van den Berg, D.J.C. Dekker, S. van den Eerenbeemt and S. Wessel, "User manual for Brabocoin, an educational cryptocurrency based on Bitcoin", Faculty of Mathematics and Computer Science, Eindhoven University of Technology, April 9, 2019, version 0.4. Online: `https://brabocoin.org/doc/res/manual.pdf`.

[10] The Brabocoin Calculator has indeed been developed completely independent of the Brabocoin software.

[dW] Benne de Weger, “ISTS 3USU0 Cryptography”, Lecture Notes, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, November 2019, version 0.6.