

1/5

Each cell represents a block or a transaction.  
Letters A to G represents metadata.

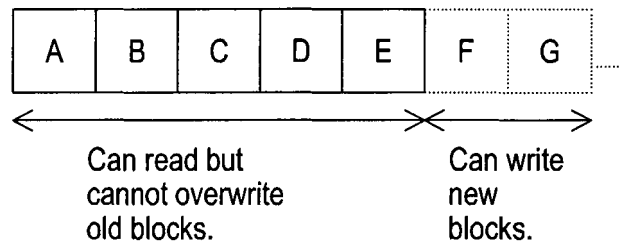


Fig. 1

```
i = 0
repeat {
  # If Condition Then Action
    i = i + 1;
} until (i = J)

Get cryptographic hash ( $H_1$ ) of the code above
```

Fig. 2

*# If Condition Then Action code block example*

```
IF trigger_detected [1] THEN
    Perform Action [2]
    Increment index and update control data [3]
END-IF
```

[1] the "*trigger*" may be a particular state of the blockchain, or an event detected off-block (e.g. a date or temperature reading, etc.) or a combination of both

[2] *Action* may include sending a signal to cause an event off clock, or broadcasting a new transaction, or a combination of both

[3] The index may be maintained (i) off block within the Manager or may be (ii) a value stored within a transaction that is then broadcast.

(i) and (ii) represent two alternative ways to maintain the control data

Fig. 3

3/5

| Word           | Opcode | Hex  | Input   | Output  | Description                                                                   |
|----------------|--------|------|---------|---------|-------------------------------------------------------------------------------|
| OP_TOALSTACK   | 107    | 0x6b | x1      | (alt)x1 | Puts the input onto the top of the alt stack. Removes it from the main stack. |
| OP_FROMALSTACK | 108    | 0x6c | (alt)x1 | x1      | Puts the input onto the top of the main stack. Removes it from the alt stack. |

Fig. 4

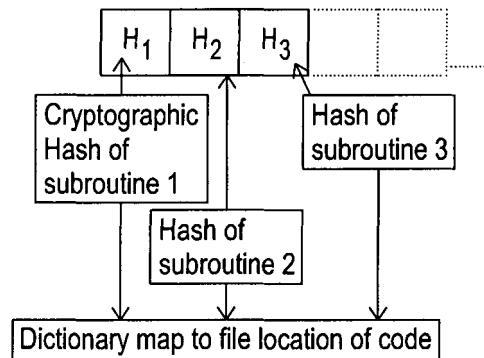


Fig. 5

| Field               | Subfield    | Bytes | Value      | Description                                                         |
|---------------------|-------------|-------|------------|---------------------------------------------------------------------|
| Metadata-CodeHash   | CodeHash    | 20    |            | RIPEMD-160 (SHA256(the actual code file addressed by CodePointer)). |
|                     | Index       | 12    |            | Integer index / denoting which iteration of the loop is at.         |
| Metadata-CodPointer | CodeType    | 4     | 0x00000001 | Indicates it is a Manager code that performs loops.                 |
|                     | CodePointer | 16    |            | IPv6 address of the actual code file location.                      |
|                     | Padding     | 12    | 0x00000... | Spare bytes.                                                        |

Fig. 6

4/5

| Field                  | Subfield      | Bytes | Value      | Description                                                               |
|------------------------|---------------|-------|------------|---------------------------------------------------------------------------|
| Metadata-OutputHash    | OutputHash    | 20    |            | RIPEMD-160 (SHA256(the output metadata file addressed by OutputPointer)). |
|                        | Padding       | 12    | 0x00000... | Spare bytes.                                                              |
| Metadata-OutputPointer | OutputPointer | 16    |            | IPv6 address of the metadata relating to the output of the code.          |
|                        | Padding       | 16    | 0x00000... | Spare bytes.                                                              |

Fig. 7

|                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Redeem Script: 4 Metadata-OldCodeHash<br>Metadata-NewCodeHash Metadata-<br>OldCodePointer Metadata-NewCodePointer<br>PK-User PK-OS PK-Developer PK-Vendor 8<br>OP-CHECKMULTSIG |
| Locking Script: OP_HASH160 < 20-byte hash<br>of redeem script > OP_EQUAL                                                                                                       |
| Unlocking Script: Sig-User Sig-OS Sig-<br>Developer Sig-Vendor Redeem Script                                                                                                   |

*Legend:**Hash: Hash of the code block**Metadata: Metadata associated with the transaction**OS: Operating System**PK: Public Key**Sig: Signature*

Fig. 8

|                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Redeem Script: 4 Metadata-OldCodeHash<br>Metadata-NewCodeHash Metadata-<br>OldCodePointer Metadata-NewCodePointer<br>PK-User PK-OS PK-Developer PK-Vendor 8<br>OP-CHECKMULTSIG |
| Locking Script: OP_HASH160 < 20-byte hash<br>of redeem script > OP_EQUAL                                                                                                       |
| Unlocking Script: Sig-User Sig-OS Sig-<br>Developer Sig-Vendor Redeem Script                                                                                                   |

*Legend:**Hash: Hash of the code block**Metadata: Metadata associated with the transaction**OS: Operating System**PK: Public Key**Sig: Signature*

Fig. 9

5/5

# Voting Counting Bot Example Repeat Loop

# The conditions of the vote

We have 100 people and if we have 57 yes votes by 1<sup>st</sup> January 2016, we will release payments to Jason.

# Part1. Run the send vote tokens bot

- i. Generate 100 Bitcoin public keys and fill each with a small amount of Bitcoins.
- ii. Each address represents an authorised token which must be used to cast a vote.
- iii. Securely send the token and the corresponding private key to each person entitled to vote.
- iv. The bot will keep the list public key associated with each authorized token, and make this list public before the vote.
- v. To ensure voting cannot be rigged, destroy the list of private keys.
- vi. To ensure voting is anonymous, destroy any association of token with voter's identity to anonymise them.

# Part2. Run the vote count bot

Array\_of\_tokens = [list of 100 authorized tokens];

Yes\_count = 0;

repeat {

    this\_token = pop from Array\_of\_tokens;

    if ( signature received from this\_token and  
        current date < 1st January 2016 )

        Increment Yes\_count;

    }

} until (Array\_of\_tokens is empty)

if ( Yes\_count >= 57 ) {

    Pay to Jason's Account.

}

Fig. 10