Lesson 16

BITCOIN: ADDRESSES, SCRIPTS, MERKLE TREES

Laura Ricci
03/05/2017
BITCOIN ADDRESS

• destination of a Bitcoin payment:
  • example: 1JArS6jzE3AJ9sZ3aFij1BmTcpFGgN86hA
  • like an email address

• derived from the public key
  • take the private key and generate public key by elliptic Curve Digital Signature Algorithm (ECDSA)
  • DSA variant based on elliptic curves
  • generate address by applying several hashes
RIPMED reduces the size of the output from 256 to 160.

The translation is more complex (the details not shown).
### BASE 58 CODING

<table>
<thead>
<tr>
<th>valore</th>
<th>base58</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>C</td>
</tr>
<tr>
<td>12</td>
<td>D</td>
</tr>
<tr>
<td>13</td>
<td>E</td>
</tr>
<tr>
<td>14</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>G</td>
</tr>
<tr>
<td>16</td>
<td>H</td>
</tr>
<tr>
<td>17</td>
<td>J</td>
</tr>
<tr>
<td>18</td>
<td>K</td>
</tr>
<tr>
<td>19</td>
<td>L</td>
</tr>
<tr>
<td>20</td>
<td>M</td>
</tr>
<tr>
<td>21</td>
<td>N</td>
</tr>
<tr>
<td>22</td>
<td>P</td>
</tr>
<tr>
<td>23</td>
<td>Q</td>
</tr>
<tr>
<td>24</td>
<td>R</td>
</tr>
<tr>
<td>25</td>
<td>S</td>
</tr>
<tr>
<td>26</td>
<td>T</td>
</tr>
<tr>
<td>27</td>
<td>U</td>
</tr>
<tr>
<td>28</td>
<td>V</td>
</tr>
<tr>
<td>29</td>
<td>W</td>
</tr>
<tr>
<td>30</td>
<td>X</td>
</tr>
<tr>
<td>31</td>
<td>Y</td>
</tr>
<tr>
<td>32</td>
<td>Z</td>
</tr>
<tr>
<td>33</td>
<td>a</td>
</tr>
<tr>
<td>34</td>
<td>b</td>
</tr>
<tr>
<td>35</td>
<td>c</td>
</tr>
<tr>
<td>36</td>
<td>d</td>
</tr>
<tr>
<td>37</td>
<td>e</td>
</tr>
<tr>
<td>38</td>
<td>f</td>
</tr>
<tr>
<td>39</td>
<td>g</td>
</tr>
<tr>
<td>40</td>
<td>h</td>
</tr>
<tr>
<td>41</td>
<td>i</td>
</tr>
<tr>
<td>42</td>
<td>j</td>
</tr>
<tr>
<td>43</td>
<td>k</td>
</tr>
<tr>
<td>44</td>
<td>m</td>
</tr>
<tr>
<td>45</td>
<td>n</td>
</tr>
<tr>
<td>46</td>
<td>o</td>
</tr>
<tr>
<td>47</td>
<td>p</td>
</tr>
<tr>
<td>48</td>
<td>q</td>
</tr>
<tr>
<td>49</td>
<td>r</td>
</tr>
<tr>
<td>50</td>
<td>s</td>
</tr>
<tr>
<td>51</td>
<td>t</td>
</tr>
<tr>
<td>52</td>
<td>u</td>
</tr>
<tr>
<td>53</td>
<td>v</td>
</tr>
<tr>
<td>54</td>
<td>w</td>
</tr>
<tr>
<td>55</td>
<td>x</td>
</tr>
<tr>
<td>56</td>
<td>y</td>
</tr>
<tr>
<td>57</td>
<td>z</td>
</tr>
</tbody>
</table>
BITCOIN ADDRESSES

- can be created by everybody
  https://www.bitaddress.org

- enough to receive bitcoins without knowing anything about the protocol or having to install a client
**TRANSACTIONS AND BLOCKS: RECAP**

**Transaction** (tx)

- can be created by everyone who owns currency units
- transfer currency units

- encapsulates number of transactions
- created by *miners*, include *proof-of-work*
- chained together to synchronized time line

both are disseminated to all Bitcoin users bin the P2P network via a gossiping protocol
A "REAL" BITCOIN TRANSACTION

encoded in JSON
among other housekeeping information:

- hash of the entire transaction, its unique identifier
- lock_time: the transaction may only be accepted by miners until the blockheight indicated in this field has been exceeded (it may be specified also a time).
a JSON array

- each element contains a pointer to a previous transaction (its hash), the index of the previous transaction's output, and a script
a JSON array

- each element contains the value to be transferred and a script, which includes the hash address of the receiver
BITCOIN SCRIPTS

- a script is a piece of code that verifies a set of arbitrary conditions that must be met in order to spend coins

- most common type of script: redeem a previous transaction by signing it with the correct key
  - 99% are simple signatures checks
  - 0.01% are MULTISIG
  - 0.01% are Pay-to-Script-Hash
  - remainder proof-of-burn

- scripts have been introduced to specify also more complex spending conditions
  - escrow transactions
  - green addresses
  - micro payments
WHAT IS THE TASK OF A SCRIPT?

- Proving that someone has the right to spend the bitcoins
WHAT IS THE TASK OF A SCRIPT?

- Proving that someone has the right to spend the bitcoins
WHAT IS THE TASK OF A SCRIPT?

- Proving that someone has the right to spend the bitcoins
WHAT IS THE TASK OF A SCRIPT?

- Proving that someone has the right to spend the bitcoins
BITCOIN SCRIPTS

- a transaction output doesn’t just specify a public key but a **script**, a piece of code that
  - defines the condition under which that output may be spent
  - **ScriptPubKey**: “to get this output, check that the public key of the signer hashed to the address included in the transaction, then check the signature through the public key of the signer”

- the transaction input needs to specify a script that:
  - gives the proof that the user performing the transaction has the right to perform it
  - **ScriptSig**: “here is the public key and the signature you are looking for, I include this information in the transaction input”

- both scripts are concatenated
  - the resulting script must run successfully for the transaction to be valid
  - only if the script executes successfully, the transaction will be included in the blockchain
BITCOIN SCRIPTING LANGUAGE: DESIGN GOALS

• Bitcoin has its own scripting language for transactions
  • inputs and outputs are scripts written in a stack based, non-Turing complete language
  • built for Bitcoin, but inspired by FORTH

• simple, compact: one byte OPCODE, 256 instructions
  • basic arithmetic, basic logic (IF...THEN...ELSE), special purpose instructions to support cryptography
    • hashes
    • signature verification
    • multisignature verification

• limits on time/memory
  • no looping, not Turing complete: no power to compute arbitrary functions
  • a design requirement: miners have to run these scripts and they do not have to fall in a loop
OUTPUT ADDRESSES ARE SCRIPTS

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
INUT ADDRESSES ARE ALSO SCRIPTS

30440220...
0467d2c9...

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

TO VERIFY: Concatenated script must execute completely with no errors
scriptPubKey (locks output):

\[
\text{OP\_DUP} \text{ OP\_HASH160} \text{ <pubKeyHash>} \text{ OP\_EQUALVERIFY} \text{ OP\_CHECKSIG}
\]

scriptSig (unlocks output i.e. in input):

\[
<\text{sig}> \text{ } <\text{pubKey}>
\]

Concatenation of the scripts

\[
<\text{sig}> \text{ } <\text{pubKey>} \text{ OP\_DUP} \text{ OP\_HASH160} \text{ <pubKeyHash>} \text{ OP\_EQUALVERIFY} \text{ OP\_CHECKSIG}
\]
Pay-To-PublicKeyHash: SCRIPT EVALUATION

\[\text{scriptPubKey} \text{ (locks output):} \]
\[
\text{OP\_DUP \ OP\_HASH160 <pubKeyHash> OP\_EQUALVERIFY \ OP\_CHECKSIG}
\]

\[\text{scriptSig} \text{ (unlocks output i.e. in input):} \]
\[
\text{<sig> \ <pubKey>}
\text{\textbf{concatenation of the scripts}}
\]
\[
\text{<sig>} \text{ \ <pubKey> \ OP\_DUP \ OP\_HASH160 <pubKeyHash>}
\text{\ OP\_EQUALVERIFY \ OP\_CHECKSIG}
\]

\[\text{State of the stack} \]

\[
<\text{sig}> 
\]

Dipartimento di Informatica
Università degli Studi di Pisa

Laura Ricci
Bitcoin: addresses, scripts, Merkle trees
Pay-To-PublicKeyHash: SCRIPT EVALUATION

scriptPubKey (locks output):

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

**concatenation of the scripts**

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

State of the stack
Pay-To-PublicKeyHash: SCRIPT EVALUATION

scriptPubKey (locks output):
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):
<sig> <pubKey>
concatenation of the scripts
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

State of the stack

- <pubKey>
- <pubKey>
- <sig>
Pay-To-PublicKeyHash: SCRIPT EVALUATION

scriptPubKey (locks output):

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

State of the stack

```
<sig> <pubKey>  OP_DUP  OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
```
Pay-To-PublicKeyHash: SCRIPT EVALUATION

scriptPubKey (locks output):

\[
\text{OP\_DUP OP\_HASH160 \text{<pubKeyHash>\ OP\_EQUALVERIFY\ OP\_CHECKSIG}}
\]

scriptSig (unlocks output i.e. in input):

\[
\text{<sig> <pubKey>}
\]

\textit{concatenation of the scripts}

\[
\text{<sig> <pubKey> OP\_DUP OP\_HASH160 \text{<pubKeyHash>\ OP\_EQUALVERIFY\ OP\_CHECKSIG}}
\]

State of the stack:

\[
\begin{array}{c}
\text{<pubKeyHash>} \\
\text{<pubKeyHash>} \\
\text{<pubKey>} \\
\text{<sig>}
\end{array}
\]
Pay-To-PublicKeyHash: SCRIPT EVALUATION

scriptPubKey (locks output):

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig>  <pubKey>
```

concatenation of the scripts

```
<sig>  <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
```

State of the stack
Pay-To-PublicKeyHash: SCRIPT EVALUATION

**scriptPubKey** (locks output):

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

**scriptSig** (unlocks output i.e. in input):

```
<sig> <pubKey>  
concatenation of the scripts
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>  
OP_EQUALVERIFY OP_CHECKSIG
```

State of the stack:
```
true
```
Laura Ricci
Bitcoin: addresses, scripts, Merkle trees
ZOOMING IN THE BLOCK CHAIN

- Coinbase transaction: first transaction in the block, include the block reward
  - single, dummy input: no linked to any output, it has no purpose
    - Satoshi included a message in the coinbase of the genesis block
      “The Times 03/Jan/2009: Chancellor on brink of second bailout for banks.”
  - multiple outputs: the sum of outputs equal the block reward + the transaction fees.

- The block preceding a given block is called its parent block.
  - blocks reference their parent block in the blockchain by including its hash
  - first block: genesis block
  - block height: order of a block in the blockchain, starting from the genesis block is called the block height.
  - blockchain head last block added to the blockchain is called the
A TAMPER FREE BLOCKCHAIN

If an attacker changes the content of a block, the hash of the block changes:

- this requires to recompute the proof of work
- the hash pointer in the successor block has to be changed, and then the pointer of the successor os the successor and so on
- an attacker would have to recompute the PoW for the entire chain
- this would require an enormous computing power.
BLOCK AND MERKLE TREE

• the structure of a block is slightly more complicated

• transactions are grouped into blocks

• if miners compute consensus on each transaction individually
  • the rate at which new transactions are accepted by the system would be much lower
  • a hash chain of blocks is much shorter than a hash chain of transactions: the block chain data structure can be verified faster.

• Miners compute the hash of the block
  • hash of the entire group of transactions?

• to optimize: compute a digital fingerprint of the entire set of transactions by exploiting Merkle trees
Merkle Root
\[ H_{ABCC} = \text{Hash}(H_{AB} + H_{CC}) \]

\[ H_{AB} = \text{Hash}(H_A + H_B) \]
\[ H_{BC} = \text{Hash}(H_C + H_C) \]
\[ H_A = \text{Hash}(\text{Tx A}) \]
\[ H_B = \text{Hash}(\text{Tx B}) \]
\[ H_C = \text{Hash}(\text{Tx C}) \]

+ concatenation operator
MERKLE TREES

• a binary balanced tree of hashes

• constructed by recursively hashing pairs of nodes until there is only one hash, called the root, or merkle root.

• cryptographic hash algorithm used in is SHA256 applied twice, also known as double-SHA256

• it needs an even number of leaf nodes
  • if there is an odd number of transactions, last transaction hash will be duplicated

• why Merkle trees?
  • provide a very efficient process to verify whether a transaction is included in a block.
divide each block into
  - header
  - transactions
  - store the root of the Merkle tree in the block header, while the transactions are stored in the block body
  - Merkle tree internal nodes nodes computed “on demand”
only keep the root hash in the block header
- delete the interior hash value to save space
- block header is about 80 bytes
- 80 byte * 6 per/h * 24 hrs * 365 = 4.2 MB per year
WHY MERKLE TREES?

- provide a very efficient process to verify whether a transaction is included in a block

- consider simple payment verification nodes (SPV node)
  - lightweight clients that do not validate and not maintain the blockchain transaction history
  - only download the block headers, not the whole block

- contrast with full nodes which receive, validate and propagate every transaction and every block throughout the network

- Merkle trees used extensively by SPV nodes to verify transactions.
  - SPV does not store transactions
  - ask to a full node if a transaction is included in a block
  - the full node has to prove to the SPV that the transaction is in that block
WHY MERKLE TREES?

- a SPV node receives a payment to one of its addresses.
  - contact a full node N for the transaction (for instance TX3)
  - the full node N sends the transaction + a verification path computed on demand
  - a verification path
    - Hash2, Hash01 for TX3

- the SPV node
  - combines the hash of the transaction with the hashes of the verification path
  - compares the result with the root hash in its block chain
  - can verify that the full node is not fraudulent
CONCLUSION: RECAP OF THE PROTOCOL

Block header

version

time
CONCLUSION: RECAP OF THE PROTOCOL

Block header

- version
- time
- mhash

mtree(tx1, ...)

- tx 1
- tx 2
- tx 3
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version
hashprev
time
mhash

mtree(tx1,...)

Block n

tx 1

tx 2

tx 3
**CONCLUSION: RECAP OF THE PROTOCOL**

Block n - 1

Block header

- version
- hashprev
- time
- target
- mhash
- nonce

mtree(tx1,...)

Block n

- tx 1
- tx 2
- tx 3

---

Defines the required number of zeros at the beginning of the PoW hash*

32 bit value can be changed to find PoW via brute-force search
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version hashprev

time target

mhash nonce

mtree(tx1,...)

Block n

tx 1

tx 2

tx 3

hash(b) starts with 000...

? yes valid

? no invalid

Defines the required number of zeros at the beginning of the PoW hash*

32 bit value can be Changed to find PoW via brute-force search
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version | hashprev
-----|-----
time | target
mhash | 0

mtree(tx1,...)

Block n

hash(b)

starts with
000...

yes
valid

no
invalid

32 bit value can be Changed to find PoW via brute-force search

Defines the required number of zeros at the beginning of the PoW hash*
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version
hashprev

time
target

mhash

1

mtree(tx1,...)

Block n

tx 1

tx 2

tx 3

hash(b)

starts with 000...

**

101...

yes → valid

no → invalid

Defines the required number of zeros at the beginning of the PoW hash*

32 bit value can be changed to find PoW via brute-force search

Laura Ricci
Bitcoin: addresses, scripts, Merkle trees
CONCLUSION: RECAP OF THE PROTOCOL

Defines the required number of zeros at the beginning of the PoW hash*

32 bit value can be changed to find PoW via brute-force search

* 000...

valid

invalid

starts with

yes

no
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version
hashprev

time
target

mhash
2

mtx(tx1,...)

Block n

tx 1

tx 2

tx 3

Block n + 1

version
hashprev

time
target

mhash
?

tx 4

tx 5
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version hashprev

time target

mhash 2

mtree(tx1,...)

Block n

tax 1

tax 2

tax 3

version

time

mhash

Block n + 1

version

time

mhash

? 000...

tax 4

tax 5
CONCLUSION: RECAP OF THE PROTOCOL

Block n - 1

Block header

version  hashprev

time         target

mhash       2

mtree(tx1,...)

Block n

confirmed transactions
1 confirmation

Block n + 1

version  000...

time         target

mhash

Block n + 1

unconfirmed transactions
0 confirmation

tx 1

tx 2

tx 3

tx 4

tx 5