The Bitcoin Network

Bitcoin is a peer-to-peer electronic cash system without any trusted third party and was described by Satoshi Nakamoto in Oktober 2008[14]. The name Bitcoin refers to both the operating network of nodes and the system's native currency. The network and protocol is usually refereed to with a large B, Bitcoin, and the currency with a small b, bitcoin.[1]

1 Digital Signatures

Part of the solution is provided by one way asymmetric encryption utilizing the `secp256k1` elliptic curve. Asymmetric encryption or public-key encryption allows the generation of key pair with the ability to derive a public key from a private key without the ability to derive the private key from the public key and the ability to prove ownership of the private key through signatures without disclosing the private key. Ownership and expenditure of bitcoin is proved by signing a previous transaction output payable to the public key derived bitcoin address[2] with the accompanying private key.

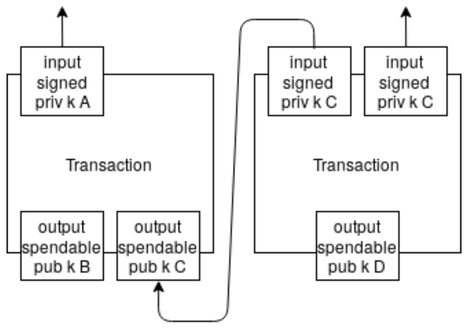

Figure 1: *A simplified illustration of two transactions.*

Bitcoin is essentially a long ledger of inputs and outputs. Note that the whole output must be spent in the same transaction and usually there is an output leading back to an address the spender controls with the 'change'. The sum of all the inputs and the sum of all the outputs usually does not line up, the differential is the implicit miners fee the miner receives if the transaction is included in a block.

[1]While it's convention to not capitalize currencies in the English language, it's not been widely followed in mainstream or semi-mainstream outlets. Also the plural form of currencies is equivalent to the singular. The convention will be followed in this report.

[2]A bitcoin address is derived from the public key by hashing the public key with both `SHA256` and `RIPEMD160`

2 The Double-Spending problem

While digital signatures are able prove ownership of bitcoin. There is nothing to stop the owner of an amount of bitcoin to sign two different transactions spending the same transaction output and broadcasting them to different parts of the network. This is called the double-spending problem which is a variation of the more general Byzantine's General Problem. The network reaches a coherent global view over the transaction history by a method known as *Proof-of-work*. The Bitcoin *Proof-of-work* algorithm is very similar to that of hashcash as it uses a hash-based cost function which requires a selected amount of work and the proof of the work easy to verify[9][3]..

The *Proof-of-work* algoithm is directly tied into the bitcoin transaction history data structure composite of a chain of blocks as seen in figure 2.

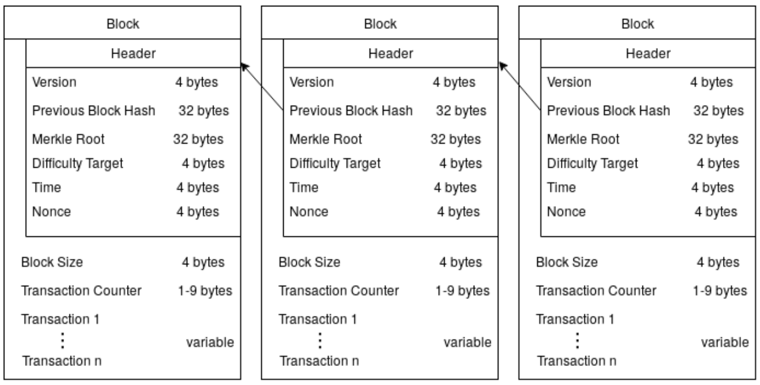

Figure 2: *Three blocks linked by the hash of the previous block's header. Only the content in the header is immutable without redoing the Proof-of-work.*

The 'work' part of the algorithm is to find a sufficiently small hash of the previous block's header. A hash is valid if it's smaller than a number imposed by a difficulty target. There is no way to reverse engineer a sufficient hash and the algorithm repeatedly hashes the block header, changing the nonce, until a hash is found. Consensus is reached by regarding the longest chain of work to be the valid chain. The blocks are in a sense chained through these hashed headers and each additional block adds cumulative work on the blocks below reducing the chance of ever being reverted. The

[3]It's similar to hashcash in both being hash-based with a cost function that is non-interactive, publicly auditable, trapdoor-free and have an unbounded probabilistic cost[9]. Bitcoin uses the double `SHA256` hash function.

difficulty target adjusts every 2015th[4] block[4] as part of the consensus algorithm and forces the average time between blocks to always be ten minutes. Running the bitcoin *Proof-of-work* algorithm is known as *mining* as per the similarities with the finding and minting of gold coins.

2.1 Merkle Tree

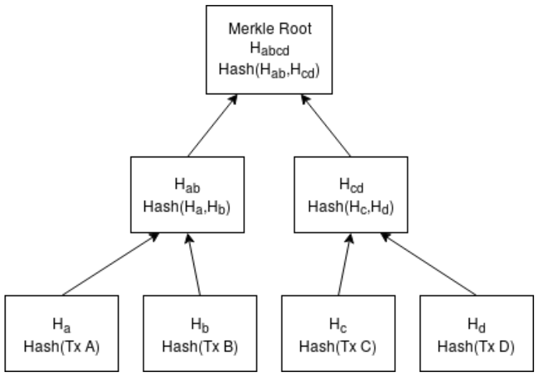

Figure 3: *A merkle tree resulting from four transactions A, B, C, D.*

The content in the block header is practically immutable since a change would make the hash invalid. The transactions are not in the header but are still immutable by the use of a Merkle tree(see fig 3). A Merkle tree is a binary tree in which the parent is the hash of its two children. The root hash of the tree is included in the header and if any transaction were to change that would cause the root hash to change rendering each transaction immutable by implication. Bitcoin uses the double `SHA256` as the merkle tree hash function.

2.2 Incentive

The miners are incentivized by receiving the transaction fees from the transactions included in mined block as well as a subsidy. The subsidy is on the form a coinbase transacion which has only outputs, this is how new bitcoin is minted. Every fourth year the subsidy halves and by 2140 will be zero. The miner must follow the consensus rules and mined blocks not following the rules will not be propagated through the network causing the miner to loose out on the block reward.

A miner, or a miner cartel, controlling more than 50% of the hashing power could technically double-spend transactions by mining blocks on a secret chain **A**, instead of the public chain **B**, without broadcasting them to the greater network and simultaneously spending outputs **OB** on chain **B**. When the chain **A** is broadcast and if it more work than **B** the outputs **OB** would be invalidated and never have transpired per the consensus rules. A huge problem with such an attack is that the confidence in the network would decrease causing the value of the 'stolen' bitcoin outputs **OA** to be close the worthless. The attacker would still own a lot of specialized hardware, only capable of running the `SHA256` hashing algorithm, that is worthless without utility outside the network. In a sense such an attack would cause mutual destruction for both the network and the attacker.

[4]This is off by one bug, 2016 blocks is two weeks

3 Script

Transactions utilize a forth-like polish-notation stack based execution scripting language[7]. Each transaction output consist of a locking script[5]. In order to spend an output a valid unlocking script must be provided in the input to the spending transaction that solves the locking script[6]. Validation is performed by executing the unlocking script and locking script sequentially as seen in figure 4.

```
 OP_1    OP_2 OP_ADD OP_3 OP_EQUAL
unlock             lock
```

Figure 4: *A simple predicate. Evaluated from left to right (1 2 +) -> 3 and (3 3 =) -> true. This lock script can obviously be solved by anyone and should not be used with real bitcoin.*

The two scripts added together forms a predicate. If the predicate is true the transaction is valid, Nakamoto considered naming the language predicate but went with script to be more inclusive to a broader audience[15].

3.1 Pay to Public Key Hash

The Pay-to-Public-key-hash or P2PKH for short was the standard script for a long time. It allows only the owner of the private key to the public key to spend the output.

The P2PKH pushed the signature produced by the private key and the public key to the stack. The public key gets duplicated and the top one gets

[5]Also refereed to as scriptPubKey. Locking script is a broader term and technically scriptPubKey scripts $\subseteq$ locking scripts.

[6]scriptSig, witness $\subseteq$ unlocking scripts.

```
<Sig><Pub_Key>
   unlock

OP_DUP OP_HASH160 <Pub_Key_Hash>
OP_EQUALVERIFY OP_CHECKSIG
   lock
```

Figure 5: *The P2PKH locking and unlocking scripts.*

hashed. The `OP_HASH160` is a double hash and first hashes the key with `SHA256` and then `RIPEMD160`. This is done to hide the public key until expenditure, which is useful if, for example, the ECDSA would break and private keys could be reversed. In the early days of the network this obfuscation technique wasn't used, for example the first transaction between Nakamoto and Finney did not hide the public key[2]. The `OP_EQUALVERIFY` pops the two public key hashes and terminates the script with false if they are not equal. The `OP_CHECKSIG` verifies that the signature is indeed a match with the public key and returns true.

3.2 Pay to Script Hash

The Pay-to-Script-Hash or P2SH is a more flexible transaction than the P2PKH and was introduced with BIP016[6]. It allows the locking script to only be the hash of the redeemable script. If a long script with many public keys is considered as seen in figure 6.

```
0 <Sig A> <Sig B>
   unlock

2 <Pk_A><Pk_B><Pk_C> 3 OP_CHECKMULTISIG
   lock
```

Figure 6: *Showing a multisig transaction which require to two signatures out of three to be spent. Note the 0 in the unlock script, it's there due to a bug with `OP_CHECKMULTISIG` which pops an extra item from the stack. If not for the 0 it would try to pop an empty stack. The dummy value must be a 0 with BIP 147 compliant node implementations[13].*

This multisig transaction(in figure 6) could be rewritten as a P2SH by hashing the locking script with `SHA256` and `RIPEMD160` and utilizing it as seen in figure 7. Note that 'redeem script' corresponds to the lock script in figure6.

```
0 <Sig A> <Sig B> <redeem script>
   unlock

OP_HASH160 <redeem script hash>
OP_CHECKVERIFY
   lock
```

Figure 7: *The P2SH transaction which is essentially equivalent to the multisig in figure 6*

Converting to a P2SH transaction does two things:

- It shifts the fee burden from the sender to the recipient. The locking script is shorter, the unlocking script is longer.
- All unspent UTXOs must be kept in the RAM of the node, shifting the burden does reduces the node RAM load.

P2SH also allows the script to be used as an address(see BIP013[5]). By simply sending to the script address could fund any type of complex transaction with no added complexity for the sender.

Even though script is not Turing complete allowing many quite advanced systems on top of it, transactions or output scripts is sometimes referred to as 'smart contracts'[7].

```
IF
  IF
    2
  ELSE
    <30 days> CHECKSEQUENCEVERIFY DROP
    <Pubkey D> CHECKSIGVERIFY
    1
  ENDIF
  <Pk A><Pk B><Pk C> 3 CHECKMULTISIG
ELSE
  <90 days> CHECKSEQUENCEVERIFY DROP
  <Pubkey D> CHECKSIG
ENDIF
```

Figure 8: *A more complex multisig locking script involving timelocks.*

One example of a more advanced script is this multisignature locking script with timelocks by Antonopoulos seen in figure 8[8]. The script consists of three clauses, the clause executed is left to the unlocking script since the condition to `IF` is the top item on the stack, not the following item as in most languages. The output could be spent by either clause being triggered:

- 2 out 3 of the A, B, C signatures.
- 30 days passed, D signature and 1 out of A, B, C signatures.
- 90 days passed and D signature.

3.3 Relative lock time

The script(figure 8) uses a construction known as a relative time lock and uses the `OP_CHECKSEQUENCEVERIFY` code in the bitcoin scripting language[11]. The relative lock time will only become true after the defined time since the parent transaction were included in a block has passed. To avoid the block miners to become oracles on time, the median time of the eleven last blocks are used as the lock time(BIP113[12]). The `OP_CHECKSEQUENCEVERIFY` was activated by BIP112[10]. The block time is therefor approximately an hour behind real time.

The Lightning Network utilize many complex transactions and more will be seen in chapter **??**.

[7]First formally defined by Nick Szabo in 1994[16]

4 Network

The Bitcoin network consists of nodes complying with the consensus protocol. In the beginning only the reference implementation available but with the growth of popularity others implemented independent implementations[4][1][3].

The mining component of nodes has mostly been decoupled from ordinary nodes with the advent of ASIC. Mining nodes still do everything as an ordinary node does as it needs the transaction history to construct valid block. Non mining nodes validates and propagates blocks and transactions across the network. It allows some benefits in constructing transactions but mainly allows the operator to validate that the network operates as expected instead of trusting that it does.

Construction of transactions can be possible without running a node and many bitcoin wallets does not.

References

[1] btcd: a full node bitcoin implementation. https://github.com/btcsuite/btcd.

[2] The first transaction, satoshi nakamoto -> hal finney in block 170. https://www.blockchain.com/btc/tx/f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16. (Accessed on 2019-02-06).

[3] Light weight bitcoin node implementation. https://github.com/lightninglabs/neutrino.

[4] Bitcoin reference implementation, 2009.

[5] Gavin Andersen. Bip 0013 address format for pay-to-script-hash. https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki, January 2012. (Accessed on 2019-02-07).

[6] Gavin Andersen. Bip 0016 pay to script hash. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki, January 2012. (Accessed on 2019-02-07).

[7] Andreas Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, first edition, July 2014.

[8] Andreas Antonopoulos. Sf bitcoin devs seminar: Advanced bitcoin scripting. https://www.youtube.com/watch?v=yU3Sr07Qnxg, April 2017.

[9] Adam Back. Hashcash - a denial of service counter-measure. 2002.

[10] BtcDrak, Mark Freienbach, and Eric Lombrozo. Bip 0112 checksequenceverify.

[11] Mark Freienbach, BtcDrak, Nicolas Dorier, and kinoshitajona. Bip 0068 relative lock-time using consensus-enforced sequence numbers. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki, May 2015. (Accessed on 2019-02-13).

[12] Thomas Kerin and Mark Freienbach. Bip 0113 median time-past as endpoint for lock-time calculations.

[13] Johnson Lau. Bip 0147 dealing with dummy stack element malleability. https://github.com/bitcoin/bips/blob/master/bip-0147.mediawiki. (Accessed on 2019-02-07).

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[15] Satoshi Nakamoto. Bitcoin talk forum post. https://bitcointalk.org/index.php?topic=195.msg1611#msg1611, 2010. (Accessed on 2019-02-11).

[16] Nick Szabo. Smart contracts. 1994.