1 Bitcoin, a deeper dive

In order to construct a voting system on top of Bitcoin there are several areas which we need to understand in detail. The below sections describe these areas in greater depth.

1.1 Transactions

A transaction is a transfer of Bitcoin value that is broadcast to the network and collected into blocks. This transaction typically references previous transaction outputs as new transaction inputs and dedicates all input Bitcoin values to new outputs [1], however, these transactions do not simply consist of a from and to address. Transactions that can be recorded in the bitcoin ledger have a lot of expressive power built into them in the form of a scripting language which allows for complex conditions to be included allowing for contracts to be easily built on top of Bitcoin [2].

This script is essentially a list of instructions recorded with each transaction that describe how the next person wanting to spend the Bitcoins being transferred can gain access to them. Scripting provides the flexibility to change the requirements of what's needed to spend a bitcoin, for example, you could require two private keys to unlock a transaction [3]. You can invisage every bitcoin transaction as a small program which describes under which conditions the transaction is valid, a transaction is considered valid if the combination of the input and output scripts return a boolean, True. The scripting language is stack-based, processed from left to right and is purposefully not turing complete, with no loops (ensuring these scripts will halt). Each transaction consists of two parts, an Input and Output section.

An input contains a reference to an output from a previous transaction (often multiple transactions are listed so that that input values add to the required output value) along with a scriptSig which is the first half of the transactions script. This scriptSig contains two components, a signature and public key, where the public key matches the hash given in the script of the redeemed output and the signature (combined with the public key) proves the transaction was created by the real owner [1].

An output contains the required instructions to send bitcoins. This contains a Value (in Satoshis) which will be the transactions worth when claimed and must be fully spent (as each output transaction can only ever be referenced once to redeem). This means it's common to send 'change' back to an address you own as part of this transaction, though any bitcoins not redeemed are considered as a transaction fee which can be claimed by whoever validates the block in the Blockchain. The output also contains a scriptPubKey, which is the second half of the transactions script.

The scripts included in the input and output sections combine to form the validation script and are evaluated in the order of scriptSig, scriptPubKey, with scriptPubKey using the values left on the stack from scriptSig. Virtually any script can be included in a transaction but miners will only accept standard scripts for security reasons. Bitcoin currently uses five (as of Bitcoin Core 0.10) different scriptSig/scriptPubKey pairs, these are the Pay-to-PubKeyHash (P2PKH), Pay-to-Script-Hash (P2SH), Pay-to-Public-Key (P2PK), Multisignature and `OP_RETURN` transaction types, of which the first one is the most commonly used [4][5].

A typical P2PKH Bitcoin address looks like 15Cytz9sHqeqtKCw2vnpEyNQ8teKtrTPjp, and by having this address we can write an output script which will send coins to it. Conversely by having the private key for this address we can write an input script that is able to spend these coins. A P2PKH input script (the scriptSig) contains only the signature and public key (no other OPCODES) which will then be pushed onto the stack. An example P2PKH scriptSig looks as follows: `<signature> <pubKey>`. A P2PKH output script (the scriptPubKey) contains a pubKeyHash (the hash of the bitcoin address you wish to pay to) along with a set of OPCODES to verify the transaction. An example P2PKH scriptPubKey looks as follows: `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG` [5][1].

When combined in a transaction they form the validation script which must evaluate to true for the transaction to be valid. The checking process for the above scriptSig and scriptPubKey examples is as follows [4][1]

Table 1: The script checking process

Stack	**Script**	**Description**
empty	`<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`	Validation script is the combined scriptSig and script-PubKey
`<pubKey>` `<sig>`	`OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`	First two values are pushed onto the stack
`<pubKey>` `<pubKey>` `<sig>`	`OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`	Top of the stack is duplicated
`<pubKeyHashAnswer>` `<pubKey>` `<sig>`	`<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`	Top of the stack is hashed, result pushed onto stack
`<pubKeyHash>` `<pubKeyHashAnswer>` `<pubKey>` `<sig>`	`OP_EQUALVERIFY OP_CHECKSIG`	Public key value is pushed onto the stack
`<pubKey>` `<sig>`	`OP_CHECKSIG`	Checks equality between the top two stack items.
True		Signature is checked between the top two stack items

1.2 Validated transactions

Due to the distributed nature of the Bitcoin Network, the prefered method of transmitting new data across the network is peer-to-peer. This has the drawbacks of less efficiency, as data is replicated many times and updates need to be sent to each node in the network, but results in each peer being more independent and thus the network as a whole being more robust (as there is no central server which can be controlled or closed down) [6].

In peer-to-peer models, even if all peers can be 'trusted', there can be the problem of consensus of what is the 'real' state of the data' say if each peer is operating at different speeds. In the Bitcoin network we cannot afford to trust any of the other peers as a malicious peer could seek to gain the system to their own ends. How, therefore, can all nodes agree which transaction blocks are genuine and should be accepted to build future transactions on.

In short, it is the voting system (briefly outlined earlier) that shows the majoritys support and fixes the transaction order in the Blockchain. This is done via the 'proof-of-work' voting system which ensures voting has a cost, computing resources and time, which makes it infeasible for a single anonymous voter to sway the vote. This vote therefore acts as a form of 'proof' to the rest of the world of the majorities will, because the solution could not be generated without the majorities computing resources [7]. The 'proof-of-work' challenge itself comes from the world of cryptography, specifically, the goal is to find a hash input thats smaller than a given target. The smaller the target, the fewer number of outputs that fulfil the equation and the harder it is to find a successful input.

In order for transactions to become validated, new transactions are forwarded around the network and placed into a pool of unconfirmed transactions. These are not considered 'accepted' yet but are available for all to see almost instantaneously. Miners draw from this pool to create a candidate next set of transactions to be officially accepted which will form the next block. The full text of all of these candidate transactions, along with the hash of the previous block and a nonce, are input into the the hash function (SHA256) and miners will try different values for the nonce until the resulting hash is below a certain value. Because its a cryptographic hash, theres no way to find a nonce that satisfies the output hashs condition other than attempting to guess [8]. At this point, all of the miners are in a competition to find the hash first, each with a potentially different set of transactions to confirm. Once a miner succeeds they announce their solution to the rest of the network, their block becomes the next block in the Blockchain, and the transactions therein become confirmed. This strategy means that one miner will choose the next set of confirmed transactions, but the hash function effectively makes the miner a random one. All other mines then validate this new block, and the transactions held within, and can choose to accept it and start work on the next block. It's important to note the properties of the hash function being used which, although requiring many billions of hashes to compute (the network hash rate was around 2,400,000 trillion hashes per second in January 2015 [9]), allowing trivial verification of the hash. As the new block contains the hash of the previous block, this forms a chain of confirmed blocks securing the order of the transactions held within.

Occasionally, two miners may find a solution to the problem at the same time creating two potential next blocks in the chain. When miners produce simultaneous blocks at the end of the block chain, each node individually chooses which block to accept, this is usually the first block they see. Eventually another miner finds the solution to another block which attaches to only one of the competing blocks. This makes that side of the fork stronger and, as the general consensus is to use the strongest chain, other nodes will switch to this longer Blockchain [8]. While this is statistically unlikely to happen, it is even more unlikely for the subsequent blocks to be solved at the same time, meaning that one fork will grow quicker than the other and the fork will resolve itself quickly. Transactions that were in the fork that wasn't chosen are not lost and are placed back into the unconfirmed transactions pool [10]. The fact that the end of the chain can be forked and rearranged means you shouldn't trust transactions at the end of the chain as much as ones further back. In Bitcoin, a transaction is not considered confirmed until it is part of a block in the longest fork, and at least five blocks follow it in the longest fork. In this case we say that the transaction has "6 confirmations". This gives the network time to come to an agreed-upon the ordering of the blocks [11].

1.3 Multisignature transactions

Multisignature (also called multisig) refers to the requirement of more than one key being present to authorise a transaction. Bitcoin has had an alternative to single-key addresses since early 2012 [12], when a new type of address called Pay-to-Script-Hash (P2SH) was defined and standardized. A typical P2SH address looks like 347N1Thc213QqfYCz3PZkjoJpNv5b14kBd. Note that P2SH addresses always begins with a '3', (instead of a '1' as in P2PKH addresses), because P2SH addresses have a version byte prefix of 0x05, which resolves as a '3' after base58check encoding [13]. A P2SH address can support arbitrary sets of N keys, any M of which are required to transact. This is commonly referred to as `'M-of-N'` [14].

There are many advantages to using multisignature transactions. Firstly, we could eliminate single points of failure by ensuring that keys are stored on completely separate devices in an `2-of-2` address. This effectively creates 2 factor authentication for your funds. For example, if one wallet is on your primary computer and the other on your smartphone. The funds cannot be spent without a signature from both devices. Thus, an attacker must gain access to both devices in order to steal your funds which is much more difficult than one device [12]. If a third key was added to the this scenario in a `2-of-3` address then the user could lose either device and still be able to access their funds using the remaining device along with the third offline key.

We can also address the problem of access control, for example, to funds of an organization. Trusting all of the funds to one key (perhaps in control of the CEO) is both dangerous, as anyone who knows the key could potentially steal the money with no way to trace it; and cumbersome, as the individual holding the key would be required to approve each transaction. Alternatively you could give more people access to this key, though this then increases the risk of the first scenario or of one of the parties being compromised and the key stolen. You can imagine an n-of-m address being very useful in this situation, as you could give one key to a department and one to a central policy controller. If a department then wishes to spend funds they must seek the approval of the central controller who can either approve or deny in accordance with company policy.

This also opens the possibilities of new transactions which can be unlocked using multisig technology. For example, we could perform a trustless escrow transaction using a `2-of-3` multi signature address, between two parties (Alice and Bob) and a trusted third party who could adjudicate a dispute (Charlie). Bob creates a P2PH address using the public keys of the three parties which Alice then pays the money to. After confirming payment has been sent, Bob sends the item and, if the transaction goes smoothly, when Alice receives the item Bob can construct an Output transaction which he will then send to Alice to release the funds. Note that there was no need to involve the arbitrator, Charlie, during the entirety of the undisputed transaction. At any point during the transactions process either party can enlist the help of Charlie to resolve a dispute. After evaluating evidence from both sides Charlie can then release the funds as he sees fit using his key along with either party [10][14].

P2PH was designed so that, rather than having senders enter long scripts into their scriptPubKey, they would allow a hash of their spending conditions instead. These spending conditions are known as a 'redeem script' and allows the sender to use a concise 34-character address (the hash of the redeem script) rather than a long unwieldy one containing full details of the spending conditions [13]. The redeem script is only revealed during the spending transaction, when it is checked against the redeem script hash which puts the responsibility for providing the redeem script on the recipient of the funds. This has the added benefit of the sender being able to fund any arbitrary transaction without knowing what the spending conditions are, or indeed, any of the public keys involved [13].

1.4 Security justification

With ever increasing quantities of money being transferred over the Bitcoin network, and the scope of this project aiming to design a secure voting system built on top of it; it's essential to confirm the level of security and to understand the risks involved.

Bitcoin is often called a cryptocurrency and, as the name suggests, cryptography is a central part of the protocol. Bitcoin makes use of two hashing functions, SHA-256 and RIPEMD-160, but it also uses Elliptic Curve DSA on the curve secp256k1 to perform signatures [15].

Signatures in bitcoin are performed exactly the same as in conventional public-key cryptography. We can verify that Alice is the owner of a transaction because she has signed it with her private key, which can then be verified with her public key. Bitcoin uses the secp256k1 elliptic curve for generating the key pairs which appears to have been chosen for possible speed optimizations in the future [15]. Attacks here could involve breaking the underlying elliptic curve cryptography either by solving the discrete logarithm problem (which could be possible with quantum computers, but there is currently no efficient non-quantum algorithm), or by finding vulnerabilities in the specific curve chosen.

As with most public-key cryptography, Bitcoin does not sign the entirety of the message as this would be too expensive. Instead, it signs a hash which can then be checked against the message to verify its integrity. If we could break the hash function, it could be possible to generate a secondary input transaction to hash to the same value as an original. This could then be used to perform a signature replay attack to forge a message. Note: this would not be possible if a unique address is used for every transaction (as suggested in the Bitcoin specification) as there would be signiture to reply [15].

The likelihood of either of these two scenarios being immediate threats is however, relatively low. Bitcoin uses industry standard technology and a breach in any of the underlying cryptography would undermine a large proportion of the security we enjoy on the internet.

Despite the underlying cryptography being secure, there are still a number of possible attacks which could be utilised against the network. While a dishonest miner cannot make bitcoins (illegitimately), steal bitcoins or make payments on your behalf (pretend to be you), they could delay or refuse the relaying of valid transactions to other nodes, attempt to create blocks which exclude specific transactions of their choosing or attempt to create a longer blockchain that would render previously accepted blocks invalid [6]. However all of the above require the attacker to have sufficient block creation power, effectively a 51% attack on the network. When an individual or group owns more than half of the network they could produce enough "computational work" to convince others that their blockchain is the best choice [15]. The bitcoin network was nearly exposed to such an attack in January of 2014 as the mining group Gash.io started to approach 50% of the mining power of the entire network. At one point the group had collectively solved 42% of the blocks in a 24 hour period [16]. The situation was resolved without incident, due to miners leaving Ghash.io for smaller pools, as well as the pools own decision to stop accepting new miners [15].

References

[1] 2016. URL: https://en.bitcoin.it/wiki/Transaction (visited on 05/12/2016).

[2] Albert Wenger. *Bitcoin As Protocol — Union Square Ventures*. 2013. URL: `https://www.usv.com/blog/bitcoin-as-protocol` (visited on 05/12/2016).

[3] 2016. URL: `https://en.bitcoin.it/wiki/Script` (visited on 05/12/2016).

[4] Davide De Rosa. *Standard scripts*. 2015. URL: `http://davidederosa.com/basic-blockchain-programming/standard-scripts/` (visited on 05/12/2016).

[5] Roland Kofler. *Thinking in Transactions*. 2014. URL: `https://bitcoinmagazine.com/articles/thinking-transactions-1401650873` (visited on 05/12/2016).

[6] Brave New Coin. *A gentle introduction to blockchain*. 2016. URL: `http://www.the-blockchain.com/docs/A-gentle-introduction-to-blockchain-technology-web.pdf` (visited on 03/12/2016).

[7] 2016. URL: `http://www.blockchaintechnologies.com/blockchain-definition` (visited on 03/12/2016).

[8] 2016. URL: `https://bitcoin.org/en/developer-guide#proof-of-work` (visited on 03/12/2016).

[9] 2016. URL: `https://en.bitcoin.it/wiki/Hash_per_second` (visited on 06/12/2016).

[10] Scott Driscoll. *Introduction to Bitcoin and Decentralized Technology — Pluralsight*. 2016. URL: `https://app.pluralsight.com/library/courses/bitcoin-decentralized-technology/table-of-contents` (visited on 30/11/2016).

[11] Michael Nielsen. *How the Bitcoin protocol actually works*. 2013. URL: `http://www.michaelnielsen.org/ddi/how-the-bitcoin-protocol-actually-works/` (visited on 06/12/2016).

[12] 2016. URL: `https://en.bitcoin.it/wiki/Multisignature` (visited on 07/12/2016).

[13] Soroush Pour. *Bitcoin multisig the hard way: Understanding raw P2SH multisig transactions*. 2014. URL: `http://www.soroushjp.com/2014/12/20/bitcoin-multisig-the-hard-way-understanding-raw-multisignature-bitcoin-transactions/` (visited on 07/12/2016).

[14] Ben Davenport. *What is Multi-Sig, and What Can It Do? — Coin Center*. 2015. URL: `https://coincenter.org/entry/what-is-multi-sig-and-what-can-it-do` (visited on 07/12/2016).

[15] Edward Z Yang. *The Cryptography of Bitcoin : Inside 736-131*. 2011. URL: `http://blog.ezyang.com/2011/06/the-cryptography-of-bitcoin/` (visited on 07/12/2016).

[16] Alec Liu. *Bitcoin's Fatal Flaw Was Nearly Exposed*. 2014. URL: `http://motherboard.vice.com/blog/bitcoins-fatal-flaw-was-nearly-exposed` (visited on 07/12/2016).