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 [**30˙transactions˙-˙bitcoin˙wiki˙2016**], 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 [**29˙wenger˙2013**].

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 [**31˙script˙-˙bitcoin˙wiki˙2016**]. 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 [**30˙transactions˙-˙bitcoin˙wiki˙2016**].

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 [**32˙de˙rosa˙2015**][**33˙kofler˙2014**].

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` [**33˙kofler˙2014**][**30˙transactions˙-˙bitcoin˙wiki˙2016**].

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 [**32˙de˙rosa˙2015**][**30˙transactions˙-˙bitcoin˙wiki˙2016**]: