Distributed Trust and Blockchains	Date:	*31 August 2018*
Instructor: *Sujit Prakash Gujar*	Scribes:	Gunda Kaushik, Anish Gulati, Rahul Bishnoi

Lecture 9: Bitcoin Scripts and its Applications

1 Recap

In the last lecture, everything discussed in the first seven lectures was revised and then Bitcoin Ledger Architecture, Bitcoin transactions and Bitcoin scripts were discussed.

1.1 Bitcoin Ledger Architecture and Bitcoin Transactions

- Bitcoin uses a ledger that keeps track of transactions similar to that of ScroogeCoin
- Transactions specify a number of inputs and outputs where inputs can be thought of as coins being consumed and output as coins being generated.
- Checking of the validity of new transaction is effecient as we just need to verify the transaction referenced in the input.
- Checking for double spending is also effecient as we just have to search from the transaction specified in the input upto the current transaction.

1.2 Bitcoin Scripts

The Bitcoin scripting language is built specifically for Bitcoin. Some properties about Bitcoin scripts are

- It is a stack-based programming language
- It has inbuilt support for cryptographic applications
- It does not have looping statements
- It has only two types of instructions - data instructions and OP_CODE.
  - OP_CODE acts on the value on the top of the stack and put their result also on the top of the stack.
- Some common OP_CODE instructions are :
  - OP_DUP: duplicates the data on the top of the stack
  - OP_HASH160: computes the hash of the data on the top of the stack and replace it with the result
  - OP_EQUALVERIFY: verifies if the first two data on the top of the stack are equals. If so, these two data are removed
  - OP_CHECKSGN: controls the digital signature using the public key and returns true or false depending on whether the signature verifies or not
  - OP_CHECKMULTISIG: will execute without errors if there are at least t valid signatures belonging to the n public keys.
- *scriptPubKey* is a locking script placed on the output of a Bitcoin transaction that requires certain conditions to be met in order for a recipient to spend his/her bitcoins.
- *scriptSig* is the unlocking script that satisfies the conditions placed on the output by the *scriptPubKey*, and is what allows it to be spent.

2 Overview for this lecture

- Bitcoin scripts execution
- Bitcoin payment methods
  - Pay-to-Public-Key-Hash
  - Pay-to-Public-Key
  - Pay-to-ScriptHash
- Applications of Bitcoin scripts
  - Escrow transactions
  - Green addresses
  - Efficient Micro-payments
  - Lock Time

3 Bitcoin Scripts Execution

The execution of Bitcoin scripts was then discussed using the example of a script used to redeem a transaction. This script is formed concatenating the *scriptSig* in the current transaction input with the *scriptPubKey* in the referred previous output. Here this script is demonstrated using the following example: In transaction-1 Amit transfers 1 BTC to Mohit, in transaction-2 Mohit transfer 1 BTC to Rohit. Both the transactions are shown in Figure 1 below.

Example: Mohit redeems 1 BTC he got from Amit by executing script:

$< Sign_{Mohit} >< PubKey_{Mohit} >$ DUP HASH160 $BitcoinAddress_{Mohit}$ EQUALVERIFY CHECKSIG
This script is the combination of output of Transaction-1 and input of Transaction-2.
The generalised script is as follows
$< sig >< pubKey >$ DUP HASH160 $< pubKeyHash? >$ EQUALVERIFY CHECKSIG

The execution of this script is presented in Figure 2. In this, for each operation the change in stack is displayed.

4 Bitcoin payment methods

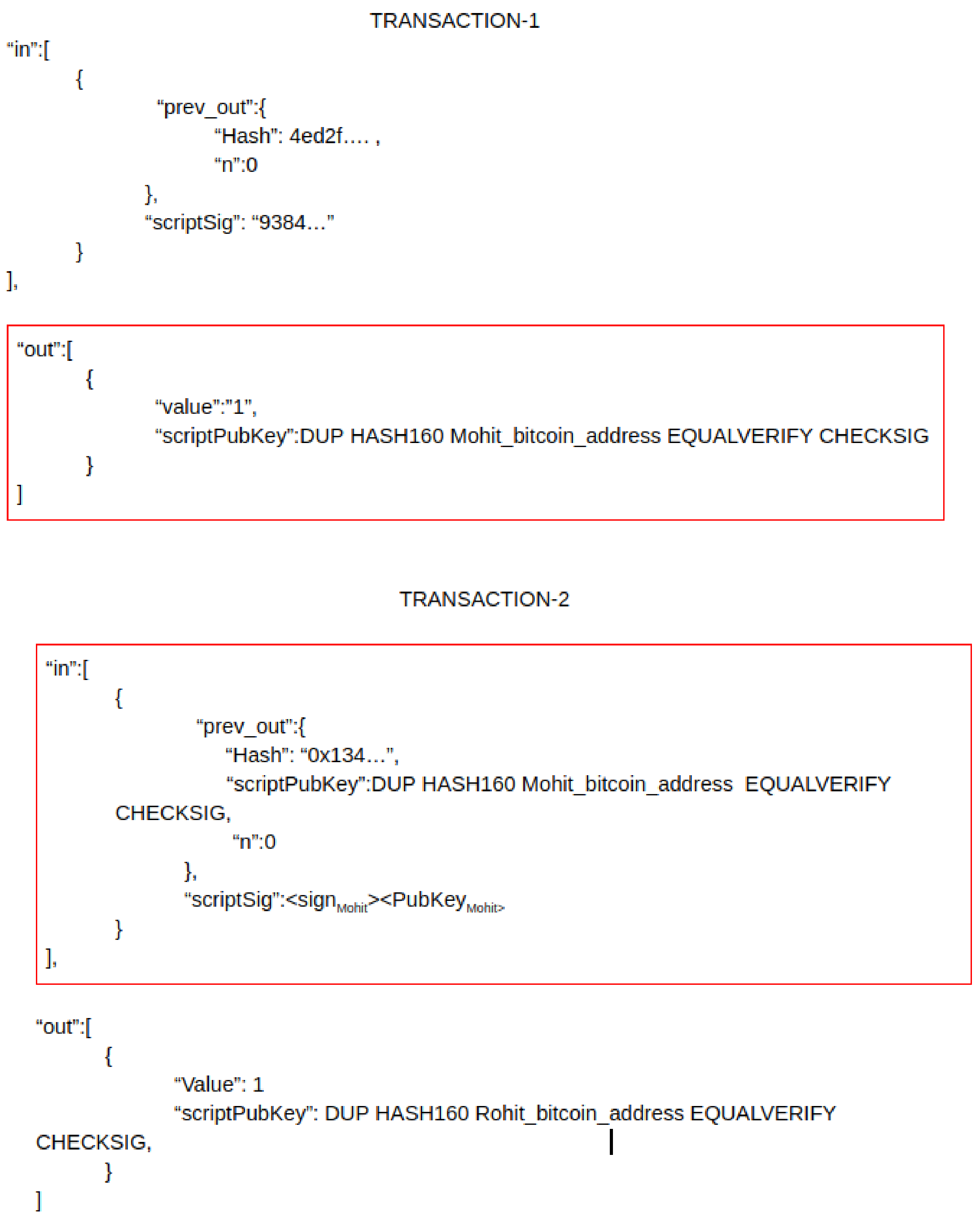
There are three different types of payment methods for bitcoin or more specifically three types of different methods in which we can specify the recipient of bitcoin payment as described in the following subsections:

4.1 Pay-to-Public-Key-Hash

The pay-to-public-key-Hash is the basic form of making a transaction and is the most common form of transaction on the Bitcoin network. Here the transaction specifies an address to which the payment is to be sent in the form of the hash of the public key. An example of a transaction using this method is described below:

$<ScriptPubKey =$ OP_DUP OP_HASH160$<PublicKeyHash>$ OP_EQUAL OP_CHECKSIG

$$ScriptSig =< Signature >< PublicKey >$$



```
                              TRANSACTION-1
"in":[
        {
                "prev_out":{
                        "Hash": 4ed2f…. ,
                        "n":0
                },
                "scriptSig": "9384…"
        }
],

"out":[
        {
                "value":"1",
                "scriptPubKey":DUP HASH160 Mohit_bitcoin_address EQUALVERIFY CHECKSIG
        }
]

                              TRANSACTION-2

"in":[
        {
                "prev_out":{
                    "Hash": "0x134…",
                    "scriptPubKey":DUP HASH160 Mohit_bitcoin_address  EQUALVERIFY
        CHECKSIG,
                     "n":0
                },
                "scriptSig":<sign_Mohit><PubKey_Mohit>
        }
],

"out":[
        {
                "Value": 1
                "scriptPubKey": DUP HASH160 Rohit_bitcoin_address EQUALVERIFY
CHECKSIG,
        }
]
```

Figure 1: Transaction 1 - between Amit and Mohit and Transaction 2 - between Mohit and Rohit

Only two outputs are possible for validation i.e. *True* or *False*. Validation of the script is done as shown below.

Validation = $<Signature><PublicKey>$OP_DUP OP_HASH160$<PublicKeyHash>$ OP_EQUAL OP_CHECKSIG

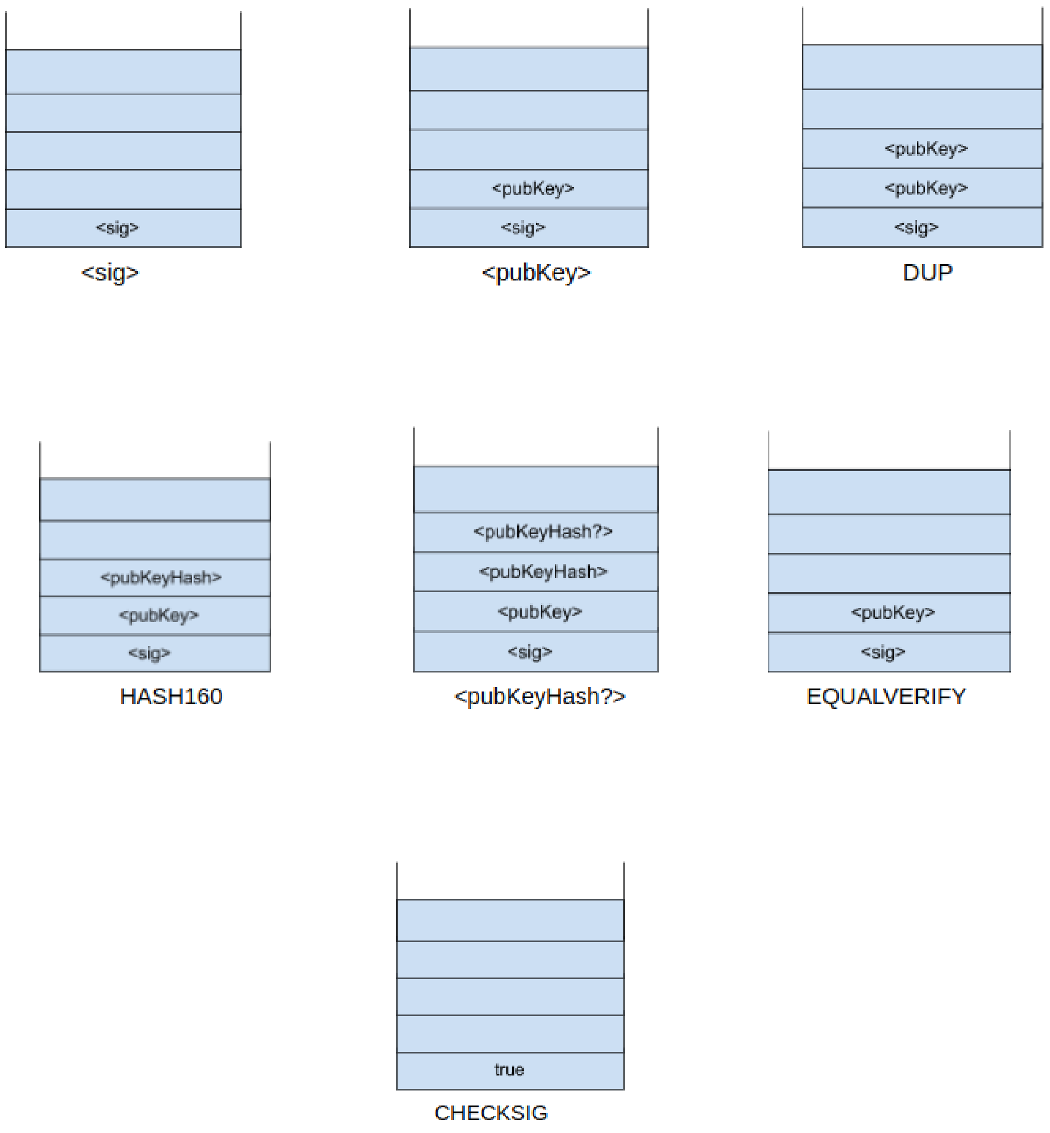


Figure 2: Execution of the script

Figure 4 shows that the process operates from left to right and the operators add to the stack in LIFO process(Last in First out). First the signature is moved to the stack, then the public key is moved to the top of the stack. The OP_DUP duplicated the top item on the stack and then the Hash160 function hashes the top item on the stack turning the public key into its hash. The OP_EQUAL checks whether the two values of the public key are the same and then the OP_CHECKSIG verifies whether the signature is true.

4.2 Pay-to-Public-Key

The pay to public key is a very similar process as the pay to public key hash. The only difference is that the presented data is slightly different as the public key is presented as the elliptic curve point and so the hashing and duplication operators are not needed.

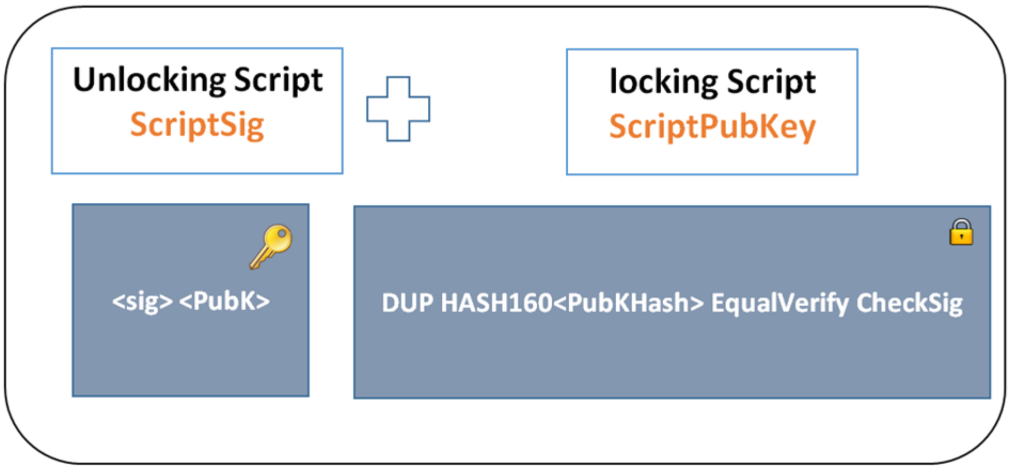


Figure 3: The ScriptPubKey and ScriptSig are joined together and executed in the above format [3].

An example of a transaction using this method is described below:

$$ScriptPubKey =< PublicKey > \text{OP\_CHECKSIG}$$

$$ScripSig =< Signature >$$

This transaction is a very simple use of the OP_CHECKSIG operator and validates the signature as being generated by the required private key and therefore having permission to unlock the specific transaction.

4.3 Pay-to-ScriptHash

Pay-to-Script-hash, or P2SH, is an extension of the multisignature idea but reducing the burden on the Bitcoin infrastructure in terms of storage required. A common 2-of-3 multisignature transaction can take up to five times as much space as a simple pay to public key transaction.
In P2SH we remove the complexity from the sender, so the recipient can just specify a hash of some *script* that the sender sends money to. As miners have to keep track of the output scripts that haven't been redeemed yet, and with P2SH outputs, the output scripts are now much smaller as they only specify a hash.

One *consequence* of the way that Bitcoin scripts work is that the sender of coins has to specify the scripts exactly. Figure 5 an example of multisignature transaction using scripts. Here the script addresses are not made through the usual process of *elliptic curve cryptography* but are instead the hash of the $<ScriptPubKey>$ of a multi signature transaction. In P2SH, the input is paid to the hash of the script and the transaction can be redeemed by the person having the script which matches the same hash.

Another important point is that the details of the transaction script are not stored in the blockchain, only the hash, and therefore the creator of the transaction must keep a copy of the script or remember the route to generate the specific hash.

5 Applications of Bitcoin scripts

It turns out that we can do many neat things that will justify the complexity of having the scripting language instead of just specifying public keys.

5.1 Escrow transactions

The escrow application means that there is a trusted intermediate party during a transaction, who verifies the whole transaction and then confirms it. Consider two parties want to do a transaction,

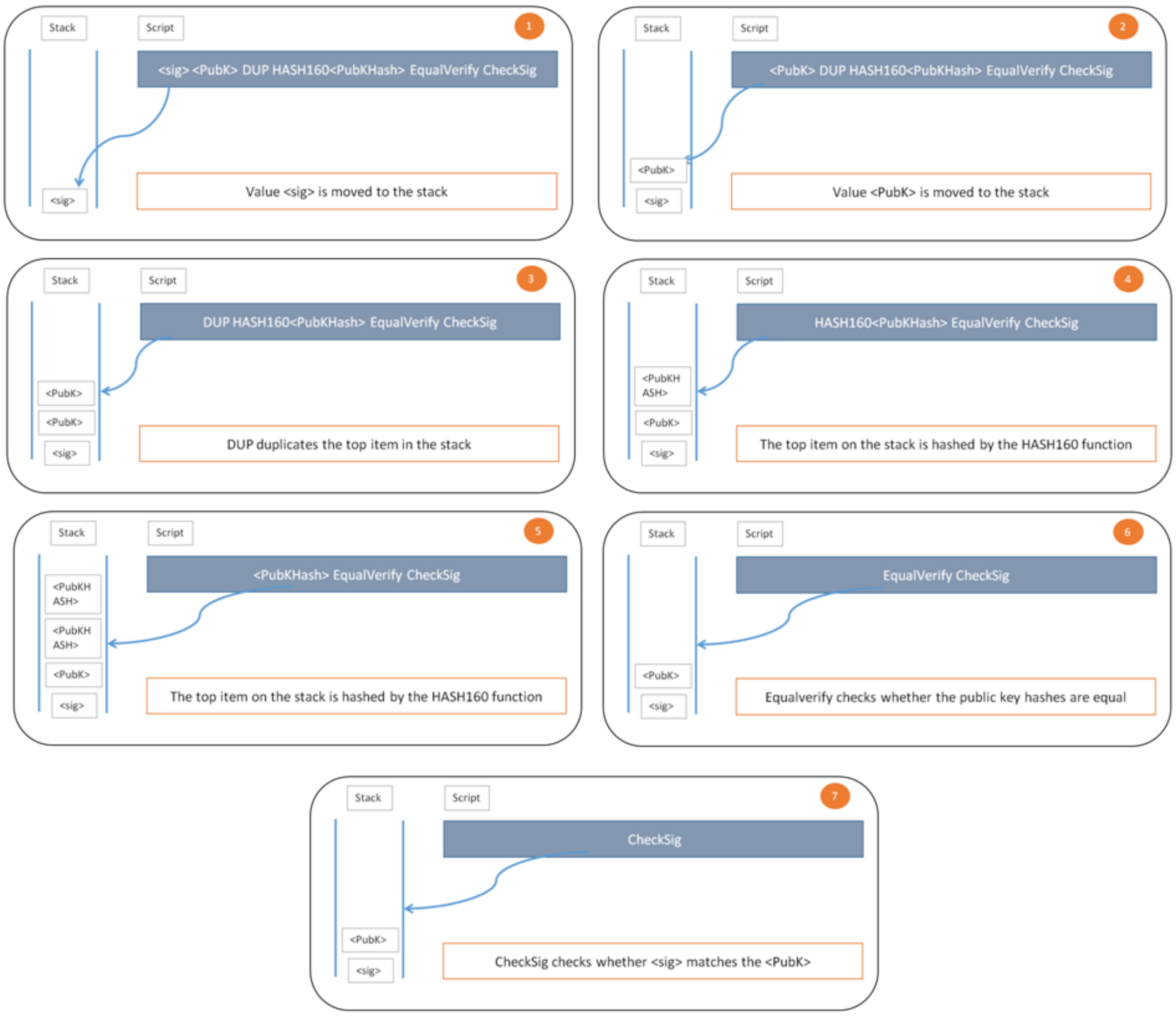

1
Stack
Script
<sig> <PubK> DUP HASH160<PubKHash> EqualVerify CheckSig
<sig>
Value <sig> is moved to the stack
2
Stack
Script
<PubK> DUP HASH160<PubKHash> EqualVerify CheckSig
<PubK>
<sig>
Value <PubK> is moved to the stack
3
Stack
Script
DUP HASH160<PubKHash> EqualVerify CheckSig
<PubK>
<PubK>
<sig>
DUP duplicates the top item in the stack
4
Stack
Script
HASH160<PubKHash> EqualVerify CheckSig
<PubKH ASH>
<PubK>
<sig>
The top item on the stack is hashed by the HASH160 function
5
Stack
Script
<PubKHash> EqualVerify CheckSig
<PubKH ASH>
<PubKH ASH>
<PubK>
<sig>
The top item on the stack is hashed by the HASH160 function
6
Stack
Script
EqualVerify CheckSig
<PubK>
<sig>
Equalverify checks whether the public key hashes are equal
7
Stack
Script
CheckSig
<PubK>
<sig>
CheckSig checks whether <sig> matches the <PubK>

Figure 4: A run through of exactly how the validation works [3]

but they do not trust each other. Then to do that transaction, they will initiate an escrow transaction. Escrow transactions can be implemented quite simply using MULTISIG data instruction. M-of-N represents the number of keys required to verify a transaction i.e. at least M signatures are needed out of N signatures for that transaction to be valid. 2-of-3 is the most common safety measure for multi-signature transactions.

Continuing, the payee will initiate a 2-of-3 MULTISIG transaction that sends some coins she owns and specifies that they can be spent if any 2 of payee(customer), *Judge*, or the merchant signs. This transaction is included in the block chain, and at this point, these coins are held in escrow between payee, *Judge* and the merchant. Now the merchant will feel safe to send goods to the customer because is neutral third party which is trusted by both the customer and the merchant.

In normal case, customer and merchant both are honest. So when the customer receives the goods, both payee and merchant sign a transaction redeeming the funds from escrow, and sending them to merchant. In this case, the *Judge* never had to get involved at all.

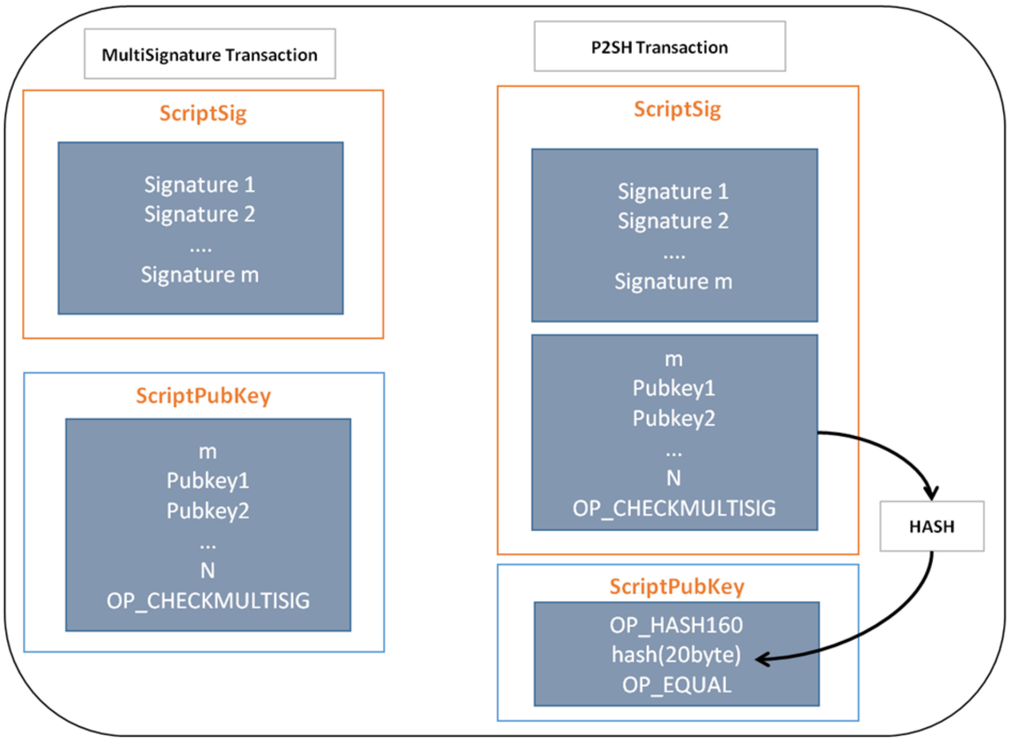


Figure 5: On the left, the multisignature transaction using pay-to-public-key. On the right, the multisignature transaction using P2SH transaction [4].

But in case, where the customer was fraud, or the merchant was fraud and never released the goods to customer, or the goods never reached to customer, here the *Judge* comes into the picture. Judge will have to decide which of the two people deserves the money. Hence, it depends on the signature of Judge to whom the money will finally go from the escrow.

Here the nice thing is that *Judge* won't have to get involved unless there is a dispute.

5.2 Green addresses

The green address is a third party trust trick and can help resolve most problems related to the need of wait for confirmations (which are 6 in Bitcoin). It also solves the problem, that one can not go and check the block chain (for any reason, example: he does not have an internet connection) for whether that transaction is added or not. To solve this problem of being able to send money using Bitcoin without recipient being able to access the block chain, green address were introduced. Green address is also just a Bitcoin address. Essentially, instead of directly paying to merchant, we ask the green address to pay to merchant. Here green address is trusted by both merchant and customer. It guarantees that green address will not double spend this money.
This is not a Bitcoin-enforced guarantee. This is a real world guarantee.
Possible drawbacks of green addresses:

1. Less anonymity, or even no anonymity at all.
2. Blockchain spam(more transactions for the bitcoin network).
3. Its not enough to create a green address, you still need to be trusted so that your green address is useful.

Now a days green addresses aren't used very much, as people have become quite nervous about the idea and are worried that it puts too much trust in the bank(green address).

5.3 Efficient Micro-payments

Consider a customer who wants to pay the continually pay the merchant small amounts of money for some of the merchant's service.

One way is to create a Bitcoin transaction every time the customer pays that small amount. But this does not seems feasible because he might end up paying a huge amount of fraction as the transaction fees to the miner.

Another way to do this is, to combine all these small payments into one big payment and pay at the end. We can do this efficiently by using micro-payments. Customer will start a MULTISIG transaction that pay the maximum amount he would ever need to spend to an output requiring both customer and merchant to sign to release the coins. After every service(that merchant provides), customer will sign the transaction paying the amount of service used till now from the start. Note that the input script used in all these transactions will all point towards the same hash i.e. same previous transaction is used as input to pay every time. Even though all these transactions are technically a double spend attack, only one of these transaction will be valid after signing. Hence when the customer stops using the service, then the merchant will sign the last transaction(it will have largest amount) sent by the customer and publish it to the block chain.

5.4 Lock Time

Consider that in the previous example of customer and merchant, the merchant never signs any of the transaction. Merchant might loose money, but even the customer won't be able to claim the outstanding amount from the coin(input transaction). This means even though merchant is loosing some money, but customer can encounter huge looses because of this. To tackle this, *lock* time was introduced.

Lock time is the time at which a particular transaction can be added to the blockchain. This is the earliest time that miners can include the transaction in their hashing of the Merkle root to attach it in the latest block to the blockchain.

Hence in the above example, the customer will create a new MULTISIG transaction containing the sign of customer and the merchant, which refund's all of the customer's money back to him. But this refund is locked till some time in the future. This transaction will be invalid before either a specific block number, or a specific point in time, based on the timestamps that are put into blocks.

References

[1] *Bitcoin and Cryptocurrency Technologies*, Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

[2] http://www.cryptocompare.com/wallets/guides/

[3] http://www.cryptocompare.com/wallets/guides/bitcoin-transactions-pay-to-address-pay-to-public-key-hash/

[4] https://www.cryptocompare.com/coins/guides/bitcoin-transactions-pay-to-script-hash/