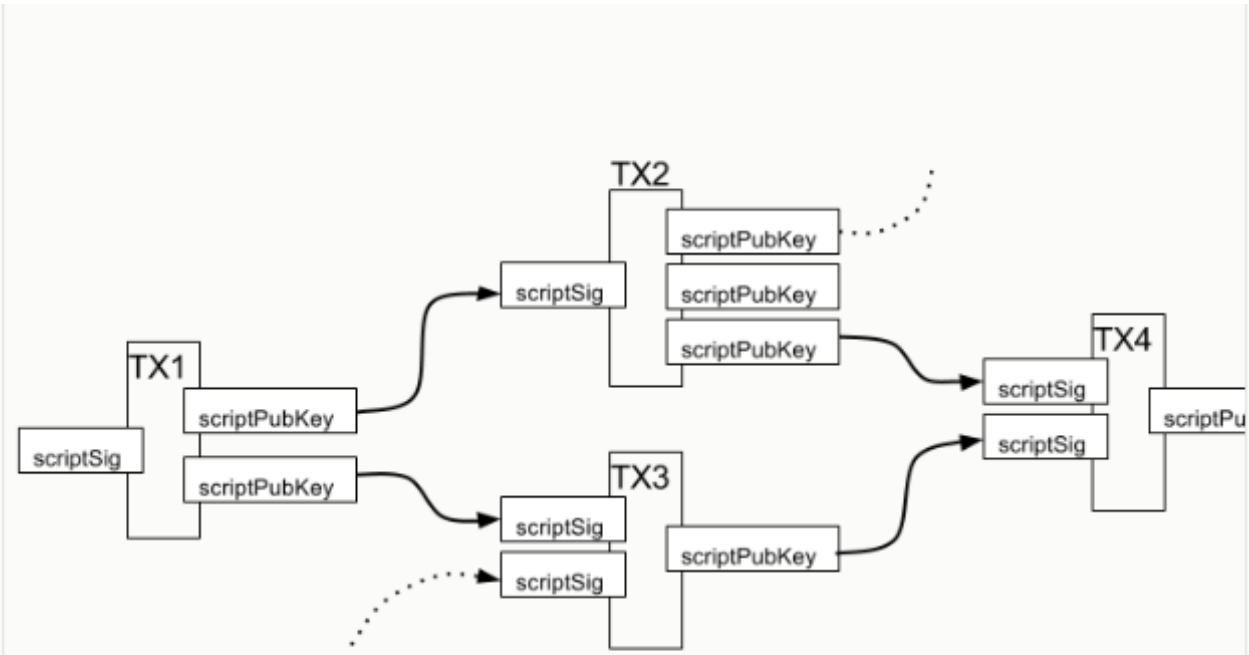
BITCOIN MAGAZINE



Thinking in Transactions

by ROLAND KOFLER on JUNE 1, 2014

0

This post is also in issue 18.

Every Bitcoin transaction runs a small program that describes under which conditions the transaction is valid. This surprising behavior can be exploited to construct a variety of contracts on top of bitcoin. In this article you will learn how basic transaction scripts work in principle.

A new transaction is valid if the transaction scripts of its own input field and the transaction script of its predecessing transaction validates to true.

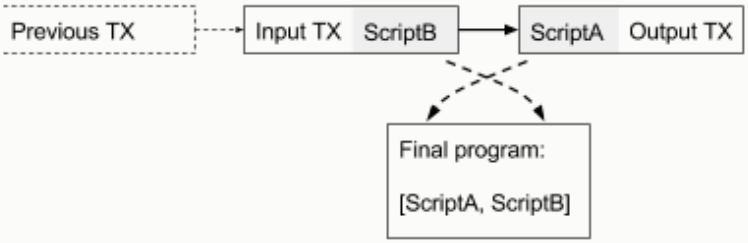


Figure: An output tx is valid only if a script program results in a boolean TRUE. The script is composed of two scripts blocks; the execution order of the scripts is: first ScriptA from OutputTX is executed, then ScriptB from InputTX.

The scripting language is stack-based, this means that each data, input or output is put on a stack of other data. The script of OutputTX is executed first: i.e. the redeemer's code. But it is not possible to stop the script before it is executed entirely and marking the transaction as valid. Finally the InputTX scriptB is executed and when the program terminates its return value determines if the OutputTX is considered valid.

Provably Unspendable Transaction

The null transaction, probably the simplest one.

This transaction will always be invalid. Whatever code is in ScriptA, when ScriptB is executed, the OP_RETURN op-code stops the execution of the transaction script and validates to FALSE. This pattern is often used to encode data in the blockchain. After the OP_RETURN you can insert arbitrary data. The advantage is that the simple bitcoin nodes can prune the transaction, saving memory, while full nodes will hold it. This is considered good behavior when 'misusing' the blockchain for storing data.

Anyone-Can-Spend Transaction

If the outputscript of the first transaction is empty, a redeeming second transaction can simply put TRUE on the stack so that the transaction is valid. Arguably anyone can do this if he is lucky to spot such an 'empty' transaction. Anyone-Can-Spend are currently non-standard, and not broadcasted in the network. But they can play an important role in future, for example with Fidelity Bonds.

Generation Transaction

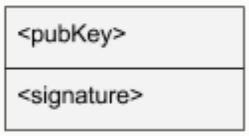
Normally a Bitcoin transaction is validated against the previous transaction (the input transaction). When a miner wins in the hashing competition and redeems his prize, there is no previous transaction. He then simply creates an Input TX with the publicly known mining fee. Redeeming such a transaction is allowed to anyone who can provide a valid signature of the public key in the Input TX, i.e. the miner himself.

Now it is time to execute the script step by step. Remember that a transaction script is executed on a stack. At the beginning the stack is empty and the program is:

<signature> <pubKey> OP_CHECKSIG

1. First the program reads the first token <signature> and since it is data, it puts it on the stack. <signature> should be a piece of data encrypted with the private key of the authorized redeemer.

2. Now the second token is also data, so we put <pubKey> on the stack. <pubKey> is the public key (the unhashed bitcoin address) of the redeemer. At the end of these two operations the stack looks like this:

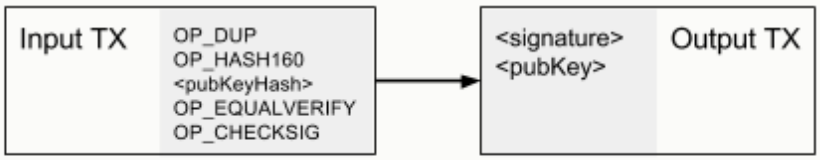


3. The next token is an operation. OP_CHECKSIG takes the first argument <pubKey> from the stack and validates the second argument <signature>. Basically it tries to open <signature> with the public key <pubKey>. If it succeeds it returns true, thus making the transaction Output TX valid.

We have seen: only the owner of the private key can redeem the Generation Transaction, he is the lucky miner.

Simple Transaction

The standard transaction script looks like this:



Therefore the full script looks like this:

<signature> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

Let's step through the execution of the program:

1. The two data tokens are put on the stack in the first two steps of the program

Stack

<pubKey>
<signature>

2. The operation OP_DUP duplicates the first element on the stack, so that we get:

<pubKey>
<pubKey>
<signature>

step executed in script:

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

3. The operation OP_HASH160 hashes the first element on the stack, so that we get:

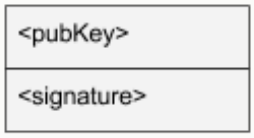
<pubKeyHash2>
<pubKey>
<signature>

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

4. <pubKeyHash> is put on the stack:


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

5. The operation OP_EQUALVERIFY compares the first two elements of the stack, in reality this is a composed operation: OP_EQUAL and OP_VERIFY are executed. OP_EQUAL puts TRUE on the stack if the two elements are the same. OP_VERIFY marks a transaction valid, if the top stack element is true, and removes the top stack element if its TRUE, if it's false it leaves it there. Generally a command executed on the stack takes its parameters from the stack, and puts its result at the stack. So OP_VERIFY behaves normally if it's FALSE, because it leaves the result on the stack, but it behaves abnormally when it's TRUE. It removes the result TRUE from the stack to continue with the next parameters.
   The result of a positive validation is therefore:



6. Finally the signature on the stack is verified with OP_CHECKSIG in the same way as in the Generation Transaction. TRUE is returned if the check succeeds.


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

The simple transaction therefore is not so simple at all. First we prove that the public key that the redeemer states is the same as we had in the Input TX, than we verify if the redeemer has the right secret key by verifying the signature of the transaction.

Conclusions

In this article we have learned that a transaction script is composed of the payer's input script and the output script of the predecessing transaction. The script needs to validate to TRUE for the transaction being considered valid. We have seen how a script is executed step by step.

With this knowledge at hand we can learn about more advanced contracts like multisig, escrow and smart property.

Reference

Description of opcodes and details of the scripts: https://en.bitcoin.it/wiki/Script

Some advanced transaction scripts: https://en.bitcoin.it/wiki/Contracts

Technical Introduction to Bitcoin: youtube.com/watch?v=Lx9zgZCMqXE

How the bitcoin protocol actually works: michaelnielsen.org/ddi/how-the-bitcoin-protocol-actually-works

Vitalik Buterin's bitcoin programming intro: http://bitcoinmagazine.com/9249/developers-introduction-bitcoin

Tip BTC: 14UKFUGd5C7hdzAQ6TYuFwVy6BnKS6mhUp

Tip LTC: LXXSuq8bUPsUuK4kC7ESMMooKTwrE4p4G1

By Roland Kofler

Roland Kofler is a software architect at TIS innovation park, South Tyrol, Italy and writes code 'for food' since 1998. He studied computer science and economics at Vienna University of Technology. Currently he is researching Bitcoin transaction scripting.