This doc was written by andytoshi on Feb 5 2013. It is a self-contained description of the work by Mike Hearn(?) on the Bitcoin wiki Contracts page. The idea was developed by Mike Caldwell (of Casascius coins) on bitcointalk.org. So this is certainly not my original work ☺.

1 Bitcoin Transactions

Often bitcoin is described in terms of "coins" being spent between "addresses". This is a good analogy to traditional currency, and describes the way we actually use bitcoin on a high level, but it bears little resemblance to the way that bitcoin actually works. A better analogy might use “spending authority” rather than “coin” and “signing key” rather than “address”[1].

So, rather than me saying “I send you 1BTC”, I might say “I have the authority to spend 1BTC, and I’m signing a message which gives this authority to you”.

How is this implemented? Built into the bitcoin protocol is a simple scripting language. Coins are represented by script outputs; to spend a coin, you write a new script (whose inputs are references to earlier scripts, creating a contiguous chain: the coin’s history). A typical script associated to a bitcoin address X is:

0. **Input:** a public key and signature of the entire script.
1. Check the hash of the key against X.
2. If this matches, check that the signature is valid.
3. If it’s valid, return success.

This script is how the bitcoin network sees addresses. In effect, it says “return success for the person who owns address X”.

If I own address X, and it has some balance, it means that in the past there was some script of this form associating the balance to the address.

To send the coin to address Y, I write a new script. I bundle this script into a transaction $\hat{Y}$, calling my script an *output* and the old script an *input*. The network checks this transaction to ensure that the balances add up and the input script returns success.

Notice that these scripts don’t *require* I use addresses in any way. I could easily write a script which always returns success — this would be a coin that anybody could spend.

2 Room for Chance

Okay, let me be a bit more specific: when I say that some script is an *input*, what I actually put on the wire is a hash of that script’s transaction (plus an index, since transactions can have multiple scripts). This is hash is a cryptographic hash, so it can’t be predicted, reversed, or forced to be some value.

[1] A bitcoin address is the RIPEMD-160 hash of the SHA256 hash of the public half of a signing key.

Since you need this hash to list a script as an input, if you don't know the hash, you can't spend the coin, *even if you have the necessary signing key(s).* If I create a bitcoin transaction to send you some money, and I run the input scripts and everything, but don't publish it, the coin is still mine.

So, suppose I do this. I have a valid transaction, giving you 1BTC, and only I know its hash. Call the hash H. (Note that H is just some number.) Then if I tell you "H is between 50000 and 60000", and I'm not lying, you've got a 1/10000 chance of guessing H.

So, you can create a new transaction giving me the 1BTC back, plus 1BTC from some other transactions, and send this transaction to me. You don't know H, so you'll have to guess. There's a 1/10000 chance that you guess right — then I can publish my transaction along with yours, and I'll be paid 1BTC.

But if you guess wrong, your transaction will have a bad input, so it's invalid and I get nothing.

Basically, your expected cost is 0.0001BTC. Over time, you could spend a thousand such transactions, but the network will only see (on average) 1. No spam, no fees.

Pretty cool, huh. But maybe there's room for one of us to cheat…

- *What if I just wait for your guess, then make a transaction which matches that guess?*

  No. I cannot force a cryptographic hash to have some target value.

- *What if I'm lying about the range that my hash lies in?*

  Then your chances of paying me drop from 1/10000 to 0. Clearly, I gain nothing.

- *What if you send me a guess outside of the range?*

  Same as if you mailed me monopoly money. I won't give you anything.

- *What if you double-spend while I'm holding the transactions secret?*

  Same as always. If I'm smart, I'll wait for confirmations before giving you anything.

- *What if you double-spend* to another probabilistic-transaction recipient*?*

  Ah, this is a real problem. Me and some other guy are both holding transactions secret, so we wouldn't notice the conflict. And chances are, you'll guess wrong for at least one of us, so we might not ever notice the problem.

  There are two strategies to mitigate this:

  – I could record all the transactions you send me, even the invalid ones. Then I could check for double-spends after the fact, and if I saw one, I'd stop doing business with you.

    But of course, you might be anonymous so this might be impossible.

  – I could share all my transactions with the other guy (and vice-versa), and we could verify that no monkey business was going on. More likely, some intermediary would be called in to do the verification for us.

Anyway, I hope this helps explain how probabilistic transactions work.