Subject: **Holding coins in an unspendable state for a rolling time window**

From: **Mike Hearn** <mike@plan99.net>
Date: Mon, Apr 18, 2011 at 11:14 PM
To: Satoshi Nakamoto <satoshin@gmx.com>

Hello Satoshi,

I hope this mail finds you well. Recently I've been thinking about how BitCoin can help handle internet abuse and would appreciate your thoughts.

My "day job" is on the Google abuse team. We make extensive use of phone verification to control outbound spam from our network. Facebook and Craigslist do the same. Phone verification works well because phone numbers are something most people have access to at least one or two of, but rarely more. Yet it has significant downsides - it's expensive (for us), flaky, some people don't like the privacy implications, and some spam is profitable enough that buying lots of SIM cards is worth it.

It would be ideal if BitCoins could be put up as collateral against an account. The amount put up would help determine the limits the system placed on your behavior (eg how much mail you can send), in an anonymous and private way. But how to implement this?

Burning coins forever is easy, just set the only output to be <pubkey> OP_FALSE. Now you can sign some server-provided challenge with that key and prove you did indeed burn those coins. A key would only be usable with one account so spammers cannot simply put up a huge collateral and then resell signatures generated with that key. If the account was found to be abused it'd be terminated like today, and the coins would be "gone".

But people do come and go from these big networks and the thought of losing the coins if you quit Google to run your own mail is unappealing. It would be ideal if coins could be locked up for a period of time such that they cannot be spent until time X, where X can be constantly pushed into the future if the owner desires it but otherwise the coins eventually become spendable again. To verify your Google account, you would take some amount of coins (say 10) and set it up so you cannot spend them for 6 months.

The script language has no concept of time. OP_BLOCKNUMBER was ruled out because re-orgs could potentially invalidate entire chains of transactions. But is an OP_DAY feasible? I'm thinking of an opcode that returns the timestamp from the block header, but rounded to the nearest day to handle the natural clock drift seen in the block chain. If it could work then a TX that ties up coins until past a certain day is easy to construct. Updating it so the deadline is constantly moving is harder. A simple brute force solution is to require the user to put

up 2x the coins such that at the point the first tx is about to expire
and become spendable again, the second tx is created. In this way you
always have at least one tx of sufficiently distant deadline to act as
collateral. But this is inelegant. A better way would be to introduce
a new rule allowing a tx to connect to such an output before the
deadline has passed, as long as the output of that tx is once again a
deadlined output of the same form. However this is less general than
the scripting language so is also somewhat inelegant.

What do you think?

From: **Satoshi Nakamoto** <satoshin@gmx.com>
Date: Wed, Apr 20, 2011 at 11:39 AM
To: Mike Hearn <mike@plan99.net>

If the script language is not stateless, if it has access to any outside information that changes or varies between nodes, attackers can use it to fork the chain. The only exception is if it is always false before a certain time and permanently true after, which is implemented with nLockTime.

Since Google is trusted, couldn't users pay a token deposit to Google and Google pays them back when they close the account?

To answer your question though, yes it can be done without using trust:

Tx 1 from User pays to a script that requires the signature of both Google and User to spend.

Tx 2 (the contract) spends Tx 1 and pays it to User. nLockTime is the time to release the money.

Steps:
1) Google gives User a pubkey to use in creating Tx 1.
2) User privately creates Tx 1, does not broadcast it yet.
3) User gives the hash of Tx 1 to Google.
4) Google signs its part of Tx 2, with nLockTime set, and gives it to User.
5) User broadcasts Tx 1.
6) User signs his half of Tx 2 and broadcasts it.

With these steps, the user already has Google's signed half of Tx 2 in hand before he broadcasts Tx 1, so he is assured of what bargain he is signing the money to.

This is the general pattern for safely signing contracts. Tx 2 is prepared first so the parties know what they're paying into. Tx 1 is broadcast to lock up the money and assign it to Tx 2. In other words, all parties assign their money to a pool that is controlled by the unanimous agreement of the group, but first the group has already signed agreement for the default action to take with the money, or partially signed multiple available options that a party can complete by adding the last signature.

By mutual agreement, the parties can always write another version of Tx 2 that releases the money immediately.

From: **Mike Hearn** <mike@plan99.net>
Date: Wed, Apr 20, 2011 at 1:55 PM

To: Satoshi Nakamoto <satoshin@gmx.com>

Thanks, that's helpful. I'm understanding contracts better now.

So the issue with having an OP_TIME/OP_BLOCKNUMBER opcode is not only that the results can change after a re-org (you said that previously), but also that people could use it to produce transactions that cease to be valid entirely after a certain time and cause a fork. Kind of obvious in hindsight.

Since Google is trusted, couldn't users pay a token deposit to Google and Google pays them back when they close the account?

Google and trust is a complicated issue. Lots of people use our services despite having little trust in us. Some people start out trusting us but then read (often sensationalist or wrong) stories in the media that change their minds, and so on. This is one of the problems we have with phone verification ... a few people don't want to give us their phone number.

For this case it'd probably be OK because trust around data privacy is different to trust around obeying contracts. I'm sure nobody would doubt that Google will pay them back - I bet we'd have an even better credit rating than the US Government in that sense :-) But we have quite a high rate of false positives with our verifications and some people would suspect we were accidentally verifying users in order to accumulate big piles of coins with which to earn interest. I've seen much sillier conspiracy theories gain traction.

Besides, avoiding the need to trust big, complex institutions is much more BitCoin-ish. And I correctly suspected there was a way to do it I didn't understand yet so it's a good chance to learn more about BitCoin.

To answer your question though, yes it can be done without using trust

If I wrote a wiki page on how to build contracts with BitCoin, would you mind reviewing it?

I'm thinking it might be a good idea to re-enable transaction replacement soon because as the network grows, it will become harder and harder to upgrade. In one sense this is good as it makes it hard to change the fundamental rules of the system. On the other hand, we risk having a protocol which has many unused features because they aren't widely supported enough. HTTP suffered this fate with many of its verbs as well as features like pipelining.

Did you have any list of tasks for re-activation, some kind of audit or finishing off some code?

I had a few other things on my mind (as always). One is, are you planning on rejoining the community at some point (eg for code reviews), or is your plan to permanently step back from the limelight? One reason I'm peppering you with questions is I worry that much of BitCoins potential lies in careful use of currently inactive features, but there's little guidance on how to do it. And frankly, I don't think I'm smart enough to figure it all out on my own. Maybe theymos is, he seems to understand it well. But if one day you leave entirely, parts of the protocol might fall into disuse, which would be a shame.

Another is the economics of mining after the transition to a fully fee based system. Right now difficulty is roughly a function of the USD/BTC exchange rate and per-block inflation. When mining reward is set by the market, it might be possible for a "Tragedy of the commons" to occur in which everyone benefits from a high difficulty, but nobody specifically wants to pay fees to get it. Besides, valuing difficulty is quite hard as you never know what the capabilities of attackers are until it's too late. Would it be possible for fees to trend towards zero over time as some miner is always willing to accept cheaper transactions and as miners drop out, the difficulty adjusts so the delays never get too bad to tolerate?

As always thanks for your insights.

From: **Satoshi Nakamoto** <satoshin@gmx.com>
Date: Sat, Apr 23, 2011 at 3:40 PM
To: Mike Hearn <mike@plan99.net>

I had a few other things on my mind (as always). One is, are you planning on rejoining the community at some point (eg for code reviews), or is your plan to permanently step back from the limelight?

I've moved on to other things. It's in good hands with Gavin and everyone.

I do hope your BitcoinJ continues to be developed into an alternative client. It gives Java devs something to work on, and it's easier with a simpler foundation that doesn't have to do everything. It'll get critical mass when impatient new users can get started using it while the other one is still downloading the block chain.