How the Bitcoin protocol actually works

by Michael Nielsen on December 6, 2013

Many thousands of articles have been written purporting to explain Bitcoin, the online, peer-to-peer currency. Most of those articles give a hand-wavy account of the underlying cryptographic protocol, omitting many details. Even those articles which delve deeper often gloss over crucial points. My aim in this post is to explain the major ideas behind the Bitcoin protocol in a clear, easily comprehensible way. We'll start from first principles, build up to a broad theoretical understanding of how the protocol works, and then dig down into the nitty-gritty, examining the raw data in a Bitcoin transaction.

Understanding the protocol in this detailed way is hard work. It is tempting instead to take Bitcoin as given, and to engage in speculation about how to get rich with Bitcoin, whether Bitcoin is a bubble, whether Bitcoin might one day mean the end of taxation, and so on. That's fun, but severely limits your understanding. Understanding the details of the Bitcoin protocol opens up otherwise inaccessible vistas. In particular, it's the basis for understanding Bitcoin's built-in scripting language, which makes it possible to use Bitcoin to create new types of financial instruments, such as **smart contracts**. New financial instruments can, in turn, be used to create new markets and to enable new forms of collective human behaviour. Talk about fun!

I'll describe Bitcoin scripting and concepts such as smart contracts in future posts. This post concentrates on explaining the nuts-and-bolts of the Bitcoin protocol. To understand the post, you need to be comfortable with **public key cryptography**, and with the closely related idea of **digital signatures**. I'll also assume you're familiar with **cryptographic hashing**. None of this is especially difficult. The basic ideas can be taught in freshman university mathematics or computer science classes. The ideas are beautiful, so if you're not familiar with them, I recommend taking a few hours to get familiar.

It may seem surprising that Bitcoin's basis is cryptography. Isn't Bitcoin a currency, not a way of sending secret messages? In fact, the problems Bitcoin needs to solve are largely about securing transactions — making sure people can't steal from one another, or impersonate one another, and so on. In the world of atoms we achieve security with devices such as locks, safes, signatures, and bank vaults. In the world of bits we achieve this kind of security with cryptography. And that's why Bitcoin is at heart a cryptographic protocol.

My strategy in the post is to build Bitcoin up in stages. I'll begin by explaining a very simple digital currency, based on ideas that are almost obvious. We'll call that currency *Infocoin*, to distinguish it from Bitcoin. Of course, our first version of Infocoin will have many deficiencies, and so we'll go through several iterations of Infocoin, with each iteration introducing just one or two simple new ideas. After several such iterations, we'll arrive at the full Bitcoin protocol. We will have reinvented Bitcoin!

This strategy is slower than if I explained the entire Bitcoin protocol in one shot. But while you can understand the mechanics of Bitcoin through such a one-shot explanation, it would be difficult to understand *why* Bitcoin is designed the way it is. The advantage of the slower iterative explanation is that it gives us a much sharper understanding of each element of Bitcoin.

Finally, I should mention that I'm a relative newcomer to Bitcoin. I've been following it loosely since 2011 (and cryptocurrencies since the late 1990s), but only got seriously into the details of the Bitcoin

protocol earlier this year. So I'd certainly appreciate corrections of any misapprehensions on my part. Also in the post I've included a number of "problems for the author" – notes to myself about questions that came up during the writing. You may find these interesting, but you can also skip them entirely without losing track of the main text.

First steps: a signed letter of intent

So how can we design a digital currency?

On the face of it, a digital currency sounds impossible. Suppose some person – let's call her Alice – has some digital money which she wants to spend. If Alice can use a string of bits as money, how can we prevent her from using the same bit string over and over, thus minting an infinite supply of money? Or, if we can somehow solve that problem, how can we prevent someone else forging such a string of bits, and using that to steal from Alice?

These are just two of the many problems that must be overcome in order to use information as money.

As a first version of Infocoin, let's find a way that Alice can use a string of bits as a (very primitive and incomplete) form of money, in a way that gives her at least some protection against forgery. Suppose Alice wants to give another person, Bob, an infocoin. To do this, Alice writes down the message "I, Alice, am giving Bob one infocoin". She then digitally signs the message using a private cryptographic key, and announces the signed string of bits to the entire world.

(By the way, I'm using capitalized "Infocoin" to refer to the protocol and general concept, and lowercase "infocoin" to refer to specific denominations of the currency. A similar useage is common, though not universal, in the Bitcoin world.)

This isn't terribly impressive as a prototype digital currency! But it does have some virtues. Anyone in the world (including Bob) can use Alice's public key to verify that Alice really was the person who signed the message "I, Alice, am giving Bob one infocoin". No-one else could have created that bit string, and so Alice can't turn around and say "No, I didn't mean to give Bob an infocoin". So the protocol establishes that Alice truly intends to give Bob one infocoin. The same fact – no-one else could compose such a signed message – also gives Alice some limited protection from forgery. Of course, *after* Alice has published her message it's possible for other people to duplicate the message, so in that sense forgery is possible. But it's not possible from scratch. These two properties – establishment of intent on Alice's part, and the limited protection from forgery – are genuinely notable features of this protocol.

I haven't (quite) said exactly what digital money *is* in this protocol. To make this explicit: it's just the message itself, i.e., the string of bits representing the digitally signed message "I, Alice, am giving Bob one infocoin". Later protocols will be similar, in that all our forms of digital money will be just more and more elaborate messages [1].

Using serial numbers to make coins uniquely identifiable

A problem with the first version of Infocoin is that Alice could keep sending Bob the same signed message over and over. Suppose Bob receives ten copies of the signed message "I, Alice, am giving Bob one infocoin". Does that mean Alice sent Bob ten *different* infocoins? Was her message

accidentally duplicated? Perhaps she was trying to trick Bob into believing that she had given him ten different infocoins, when the message only proves to the world that she intends to transfer one infocoin.
What we'd like is a way of making infocoins unique. They need a label or serial number. Alice would sign the message "I, Alice, am giving Bob one infocoin, with serial number 8740348″. Then, later, Alice could sign the message "I, Alice, am giving Bob one infocoin, with serial number 8770431″, and Bob (and everyone else) would know that a different infocoin was being transferred.

To make this scheme work we need a trusted source of serial numbers for the infocoins. One way to create such a source is to introduce a *bank*. This bank would provide serial numbers for infocoins, keep track of who has which infocoins, and verify that transactions really are legitimate,
In more detail, let's suppose Alice goes into the bank, and says "I want to withdraw one infocoin from my account". The bank reduces her account balance by one infocoin, and assigns her a new, never-before used serial number, let's say 1234567. Then, when Alice wants to transfer her infocoin to Bob, she signs the message "I, Alice, am giving Bob one infocoin, with serial number 1234567″. But Bob doesn't just accept the infocoin. Instead, he contacts the bank, and verifies that: (a) the infocoin with that serial number belongs to Alice; and (b) Alice hasn't already spent the infocoin. If both those things are true, then Bob tells the bank he wants to accept the infocoin, and the bank updates their records to show that the infocoin with that serial number is now in Bob's possession, and no longer belongs to Alice.

Making everyone collectively the bank

This last solution looks pretty promising. However, it turns out that we can do something much more ambitious. We can eliminate the bank entirely from the protocol. This changes the nature of the currency considerably. It means that there is no longer any single organization in charge of the currency. And when you think about the enormous power a central bank has – control over the money supply – that's a pretty huge change.

The idea is to make it so *everyone* (collectively) is the bank. In particular, we'll assume that everyone using Infocoin keeps a complete record of which infocoins belong to which person. You can think of this as a shared public ledger showing all Infocoin transactions. We'll call this ledger the *block chain*, since that's what the complete record will be called in Bitcoin, once we get to it.
Now, suppose Alice wants to transfer an infocoin to Bob. She signs the message "I, Alice, am giving Bob one infocoin, with serial number 1234567″, and gives the signed message to Bob. Bob can use his copy of the block chain to check that, indeed, the infocoin is Alice's to give. If that checks out then he broadcasts both Alice's message and his acceptance of the transaction to the entire network, and everyone updates their copy of the block chain.

We still have the "where do serial number come from" problem, but that turns out to be pretty easy to solve, and so I will defer it to later, in the discussion of Bitcoin. A more challenging problem is that this protocol allows Alice to cheat by double spending her infocoin. She sends the signed message "I, Alice, am giving Bob one infocoin, with serial number 1234567″ to Bob, and the message"I, Alice, am giving Charlie one infocoin, with [the same] serial number 1234567″ to Charlie. Both Bob and Charlie use their copy of the block chain to verify that the infocoin is Alice's to spend. Provided they do this

verification at nearly the same time (before they've had a chance to hear from one another), both will find that, yes, the block chain shows the coin belongs to Alice. And so they will both accept the transaction, and also broadcast their acceptance of the transaction. Now there's a problem. How should other people update their block chains? There may be no easy way to achieve a consistent shared ledger of transactions. And even if everyone can agree on a consistent way to update their block chains, there is still the problem that either Bob or Charlie will be cheated.

At first glance double spending seems difficult for Alice to pull off. After all, if Alice sends the message first to Bob, then Bob can verify the message, and tell everyone else in the network (including Charlie) to update their block chain. Once that has happened, Charlie would no longer be fooled by Alice. So there is most likely only a brief period of time in which Alice can double spend. However, it's obviously undesirable to have any such a period of time. Worse, there are techniques Alice could use to make that period longer. She could, for example, use network traffic analysis to find times when Bob and Charlie are likely to have a lot of latency in communication. Or perhaps she could do something to deliberately disrupt their communications. If she can slow communication even a little that makes her task of double spending much easier.

How can we address the problem of double spending? The obvious solution is that when Alice sends Bob an infocoin, Bob shouldn't try to verify the transcation alone. Rather, he should broadcast the possible transaction to the entire network of Infocoin users, and ask them to help determine whether the transaction is legitimate. If they collectively decide that the transaction is okay, then Bob can accept the infocoin, and everyone will update their block chain. This type of protocol can help prevent double spending, since if Alice tries to spend her infocoin with both Bob and Charlie, other people on the network will notice, and network users will tell both Bob and Charlie that there is a problem with the transaction, and the transaction shouldn't go through.

In more detail, let's suppose Alice wants to give Bob an infocoin. As before, she signs the message "I, Alice, am giving Bob one infocoin, with serial number 1234567″, and gives the signed message to Bob. Also as before, Bob does a sanity check, using his copy of the block chain to check that, indeed, the coin currently belongs to Alice. But at that point the protocol is modified. Bob doesn't just go ahead and accept the transaction. Instead, he broadcast Alice's message to the entire network. Other members of the network check to see whether Alice owns that infocoin. If so, they broadcast the message "Yes, Alice owns infocoin 1234567, it can now be transferred to Bob." Once enough people have broadcast that message, everyone updates their block chain to show that infocoin 1234567 now belongs to Bob, and the transaction is complete.

This protocol has many imprecise elements at present. For instance, what does it mean to say "once enough people have broadcast that message"? What exactly does "enough" mean here? It can't mean everyone in the network, since we don't *a priori* know who is on the Infocoin network. For the same reason, it can't mean some fixed fraction of users in the network. We won't try to make these ideas precise right now. Instead, in the next section I'll point out a serious problem with the approach as described. Fixing that problem will at the same time have the pleasant side effect of making the ideas above much more precise.

Proof-of-work

Suppose Alice wants to double spend in the network-based protocol I just described. She could do this by taking over the Infocoin network. Let's suppose she uses an automated system to set up a large number of separate identities, let's say a billion, on the Infocoin network. As before, she tries to double spend the same infocoin with both Bob and Charlie. But when Bob and Charlie ask the network to validate their respective transactions, Alice's sock puppet identities swamp the network, announcing to Bob that they've validated his transaction, and to Charlie that they've validated his transaction, possibly fooling one or both into accepting the transaction.

There's a clever way of avoiding this problem, using an idea known as *proof-of-work*. The idea is counterintuitive and involves a combination of two ideas: (1) to (artificially) make it *computationally costly* for network users to validate transactions; and (2) to*reward* them for trying to help validate transactions. The reward is used so that people on the network will try to help validate transactions, even though that's now been made a computationally costly process. The benefit of making it costly to validate transactions is that validation can no longer be influenced by the number of network identities someone controls, but only by the total computational power they can bring to bear on validation. As we'll see, with some clever design we can make it so a cheater would need enormous computational resources to cheat, making it impractical.
That's the gist of proof-of-work. But to really understand proof-of-work, we need to go through the details.

Suppose Alice broadcasts to the network the news that "I, Alice, am giving Bob one infocoin, with serial number 1234567″.

As other people on the network hear that message, each adds it to a queue of pending transactions that they've been told about, but which haven't yet been approved by the network. For instance, another network user named David might have the following queue of pending transactions:

I, Tom, am giving Sue one infocoin, with serial number 1201174.

I, Sydney, am giving Cynthia one infocoin, with serial number 1295618.

I, Alice, am giving Bob one infocoin, with serial number 1234567.

David checks his copy of the block chain, and can see that each transaction is valid. He would like to help out by broadcasting news of that validity to the entire network.

However, before doing that, as part of the validation protocol David is required to solve a hard computational puzzle – the proof-of-work. Without the solution to that puzzle, the rest of the network won't accept his validation of the transaction.

What puzzle does David need to solve? To explain that, let h be a fixed hash function known by everyone in the network – it's built into the protocol. Bitcoin uses the well-known **SHA-256** hash function, but any crypographically secure hash function will do. Let's give David's queue of pending transactions a label, l, just so it's got a name we can refer to. Suppose David appends a number x (called the *nonce*) to l and hashes the combination. For example, if we use $l =$ "Hello, world!" (obviously this is not a list of transactions, just a string used for illustrative purposes) and the nonce $x = 0$**then** (output is in hexadecimal)

```
h("Hello, world!0") =

  1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
```

The puzzle David has to solve – the proof-of-work – is to find a nonce x such that when we append x to l and hash the combination the output hash begins with a long run of zeroes. The puzzle can be made more or less difficult by varying the number of zeroes required to solve the puzzle. A relatively simple proof-of-work puzzle might require just three or four zeroes at the start of the hash, while a more difficult proof-of-work puzzle might require a much longer run of zeros, say 15 consecutive zeroes. In either case, the above attempt to find a suitable nonce, with $x = 0$, is a failure, since the output doesn't begin with any zeroes at all. Trying $x = 1$ doesn't work either:

```
h("Hello, world!1") =

  e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8
```

We can keep trying different values for the nonce, $x = 2, 3, \ldots$. Finally, at $x = 4250$ we obtain:

```
h("Hello, world!4250") =

  0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9
```

This nonce gives us a string of four zeroes at the beginning of the output of the hash. This will be enough to solve a simple proof-of-work puzzle, but not enough to solve a more difficult proof-of-work puzzle.

What makes this puzzle hard to solve is the fact that the output from a cryptographic hash function behaves like a random number: change the input even a tiny bit and the output from the hash function changes completely, in a way that's hard to predict. So if we want the output hash value to begin with 10 zeroes, say, then David will need, on average, to try $16^{10} \approx 10^{12}$ different values for x before he finds a suitable nonce. That's a pretty challenging task, requiring lots of computational power.

Obviously, it's possible to make this puzzle more or less difficult to solve by requring more or fewer zeroes in the output from the hash function. In fact, the Bitcoin protocol gets quite a fine level of control over the difficulty of the puzzle, by using a slight variation on the proof-of-work puzzle described above. Instead of requiring leading zeroes, the Bitcoin proof-of-work puzzle requires the hash of a block's header to be lower than or equal to a number known as the **target**. This target is automatically adjusted to ensure that a Bitcoin block takes, on average, about ten minutes to validate. (In practice there is a sizeable randomness in how long it takes to validate a block – sometimes a new block is validated in just a minute or two, other times it may take 20 minutes or even longer. It's straightforward to modify the Bitcoin protocol so that the time to validation is much more sharply peaked around ten minutes. Instead of solving a single puzzle, we can require that multiple puzzles be solved; with some careful design it is possible to considerably reduce the variance in the time to mine a block of transactions.)

Alright, let's suppose David is lucky and finds a suitable nonce, x. Celebration! (He'll be rewarded for finding the nonce, as described below). He broadcasts the block of transactions he's approving to the network, together with the value for x. Other participants in the Infocoin network can verify that x is a

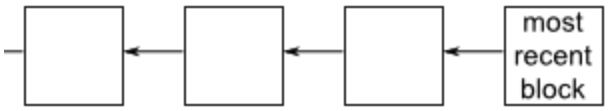
valid solution to the proof-of-work puzzle. And they then update their block chains to include the new block of transactions.

For the proof-of-work idea to have any chance of succeeding, network users need an incentive to help validate transactions. Without such an incentive, they have no reason to expend valuable computational power, merely to help validate other people's transactions. And if network users are not willing to expend that power, then the whole system won't work. The solution to this problem is to reward people who help validate transactions. In particular, suppose we reward whoever successfully validates a block of transactions by crediting them with some infocoins. Provided the infocoin reward is large enough that will give them an incentive to participate in validation.

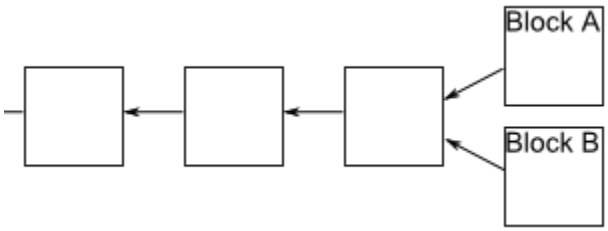
In the Bitcoin protocol, this validation process is called *mining*. For each block of transactions validated, the successful miner receives a bitcoin reward. Initially, this was set to be a 50 bitcoin reward. But for every 210,000 validated blocks (roughly, once every four years) the reward halves. This has happened just once, to date, and so the current reward for mining a block is 25 bitcoins. This halving in the rate will continue every four years until the year 2140 CE. At that point, the reward for mining will drop below 10^{-8} bitcoins per block. 10^{-8} bitcoins is actually the minimal unit of Bitcoin, and is known as a *satoshi*. So in 2140 CE the total supply of bitcoins will cease to increase. However, that won't eliminate the incentive to help validate transactions. Bitcoin also makes it possible to set aside some currency in a transaction as a *transaction fee*, which goes to the miner who helps validate it. In the early days of Bitcoin transaction fees were mostly set to zero, but as Bitcoin has gained in popularity, transaction fees have gradually risen, and are now a substantial additional incentive on top of the 25 bitcoin reward for mining a block.
You can think of proof-of-work as a competition to approve transactions. Each entry in the competition costs a little bit of computing power. A miner's chance of winning the competition is (roughly, and with some caveats) equal to the proportion of the total computing power that they control. So, for instance, if a miner controls one percent of the computing power being used to validate Bitcoin transactions, then they have roughly a one percent chance of winning the competition. So provided a lot of computing power is being brought to bear on the competition, a dishonest miner is likely to have only a relatively small chance to corrupt the validation process, unless they expend a huge amount of computing resources.

Of course, while it's encouraging that a dishonest party has only a relatively small chance to corrupt the block chain, that's not enough to give us confidence in the currency. In particular, we haven't yet conclusively addressed the issue of double spending.

I'll analyse double spending shortly. Before doing that, I want to fill in an important detail in the description of Infocoin. We'd ideally like the Infocoin network to agree upon the *order* in which transactions have occurred. If we don't have such an ordering then at any given moment it may not be clear who owns which infocoins. To help do this we'll require that new blocks always include a pointer to the last block validated in the chain, in addition to the list of transactions in the block. (The pointer is actually just a hash of the previous block). So typically the block chain is just a linear chain of blocks of transactions, one after the other, with later blocks each containing a pointer to the immediately prior block:

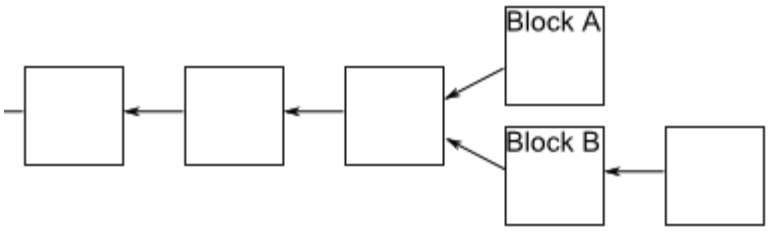


Occasionally, a fork will appear in the block chain. This can happen, for instance, if by chance two miners happen to validate a block of transactions near-simultaneously – both broadcast their newly-validated block out to the network, and some people update their block chain one way, and others update their block chain the other way:



This causes exactly the problem we're trying to avoid – it's no longer clear in what order transactions have occurred, and it may not be clear who owns which infocoins. Fortunately, there's a simple idea that can be used to remove any forks. The rule is this: if a fork occurs, people on the network keep track of both forks. But at any given time, miners only work to extend whichever fork is longest in their copy of the block chain.

Suppose, for example, that we have a fork in which some miners receive block A first, and some miners receive block B first. Those miners who receive block A first will continue mining along that fork, while the others will mine along fork B. Let's suppose that the miners working on fork B are the next to successfully mine a block:



After they receive news that this has happened, the miners working on fork A will notice that fork B is now longer, and will switch to working on that fork. Presto, in short order work on fork A will cease, and everyone will be working on the same linear chain, and block A can be ignored. Of course, any still-pending transactions in A will still be pending in the queues of the miners working on fork B, and so all transactions will eventually be validated.

Likewise, it may be that the miners working on fork A are the first to extend their fork. In that case work on fork B will quickly cease, and again we have a single linear chain.

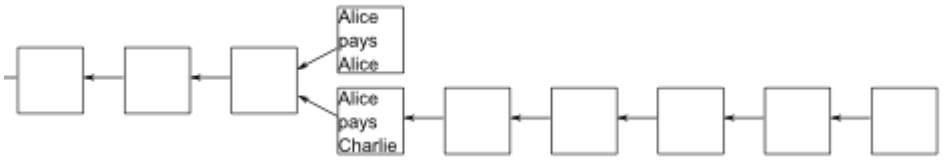
No matter what the outcome, this process ensures that the block chain has an agreed-upon time ordering of the blocks. In Bitcoin proper, a transaction is not considered confirmed until: (1) it is part of a block in the longest fork, and (2) at least 5 blocks follow it in the longest fork. In this case we say

that the transaction has “6 confirmations”. This gives the network time to come to an agreed-upon the ordering of the blocks. We’ll also use this strategy for Infocoin.

With the time-ordering now understood, let’s return to think about what happens if a dishonest party tries to double spend. Suppose Alice tries to double spend with Bob and Charlie. One possible approach is for her to try to validate a block that includes both transactions. Assuming she has one percent of the computing power, she will occasionally get lucky and validate the block by solving the proof-of-work. Unfortunately for Alice, the double spending will be immediately spotted by other people in the Infocoin network and rejected, despite solving the proof-of-work problem. So that’s not something we need to worry about.

A more serious problem occurs if she broadcasts two separate transactions in which she spends the same infocoin with Bob and Charlie, respectively. She might, for example, broadcast one transaction to a subset of the miners, and the other transaction to another set of miners, hoping to get both transactions validated in this way. Fortunately, in this case, as we’ve seen, the network will eventually confirm one of these transactions, but not both. So, for instance, Bob’s transaction might ultimately be confirmed, in which case Bob can go ahead confidently. Meanwhile, Charlie will see that his transaction has not been confirmed, and so will decline Alice’s offer. So this isn’t a problem either. In fact, knowing that this will be the case, there is little reason for Alice to try this in the first place.

An important variant on double spending is if Alice = Bob, i.e., Alice tries to spend a coin with Charlie which she is also “spending” with herself (i.e., giving back to herself). This sounds like it ought to be easy to detect and deal with, but, of course, it’s easy on a network to set up multiple identities associated with the same person or organization, so this possibility needs to be considered. In this case, Alice’s strategy is to wait until Charlie accepts the infocoin, which happens after the transaction has been confirmed 6 times in the longest chain. She will then attempt to fork the chain before the transaction with Charlie, adding a block which includes a transaction in which she pays herself:



Unfortunately for Alice, it’s now very difficult for her to catch up with the longer fork. Other miners won’t want to help her out, since they’ll be working on the longer fork. And unless Alice is able to solve the proof-of-work at least as fast as everyone else in the network combined – roughly, that means controlling more than fifty percent of the computing power – then she will just keep falling further and further behind. Of course, she might get lucky. We can, for example, imagine a scenario in which Alice controls one percent of the computing power, but happens to get lucky and finds six extra blocks in a row, before the rest of the network has found any extra blocks. In this case, she might be able to get ahead, and get control of the block chain. But this particular event will occur with probability $1/100^6 = 10^{-12}$. A more general analysis along these lines shows that Alice’s probability of ever catching up is infinitesimal, unless she is able to solve proof-of-work puzzles at a rate approaching all other miners combined.

Of course, this is not a rigorous security analysis showing that Alice cannot double spend. It's merely an informal plausibility argument. The **original paper** introducing Bitcoin did not, in fact, contain a rigorous security analysis, only informal arguments along the lines I've presented here. The security community is still analysing Bitcoin, and trying to understand possibility vulnerabilities. You can see some of this research**listed here**, and I mention a few related problems in the "Problems for the author" below. At this point I think it's fair to say that the jury is still out on how secure Bitcoin is.
The proof-of-work and mining ideas give rise to many questions. How much reward is enough to persuade people to mine? How does the change in supply of infocoins affect the Infocoin economy? Will Infocoin mining end up in concentrated in the hands of a few, or many? If it's just a few, doesn't that endanger the security of the system? Presumably transaction fees will eventually equilibriate – won't this introduce an unwanted source of friction, and make small transactions less desirable? These are all great questions, but beyond the scope of this post. I may come back to the questions (in the context of Bitcoin) in a future post. For now, we'll stick to our focus on understanding how the Bitcoin protocol works.

Problems for the author

- I don't understand why double spending can't be prevented in a simpler manner using **two-phase commit**. Suppose Alice tries to double spend an infocoin with both Bob and Charlie. The idea is that Bob and Charlie would each broadcast their respective messages to the Infocoin network, along with a request: "Should I accept this?" They'd then wait some period – perhaps ten minutes – to hear any naysayers who could prove that Alice was trying to double spend. If no such nays are heard (and provided there are no signs of attempts to disrupt the network), they'd then accept the transaction. This protocol needs to be hardened against network attacks, but it seems to me to be the core of a good alternate idea. How well does this work? What drawbacks and advantages does it have compared to the full Bitcoin protocol?
- Early in the section I mentioned that there is a natural way of reducing the variance in time required to validate a block of transactions. If that variance is reduced too much, then it creates an interesting attack possibility. Suppose Alice tries to fork the chain in such a way that: (a) one fork starts with a block in which Alice pays herself, while the other fork starts with a block in which Alice pays Bob; (b) both blocks are announced nearly simultaneously, so roughly half the miners will attempt to mine each fork; (c) Alice uses her mining power to try to keep the forks of roughly equal length, mining whichever fork is shorter – this is ordinarily hard to pull off, but becomes significantly easier if the standard deviation of the time-to-validation is much shorter than the network latency; (d) after 5 blocks have been mined on both forks, Alice throws her mining power into making it more likely that Charles's transaction is confirmed; and (e) after confirmation of Charles's transaction, she then throws her computational power into the other fork, and attempts to

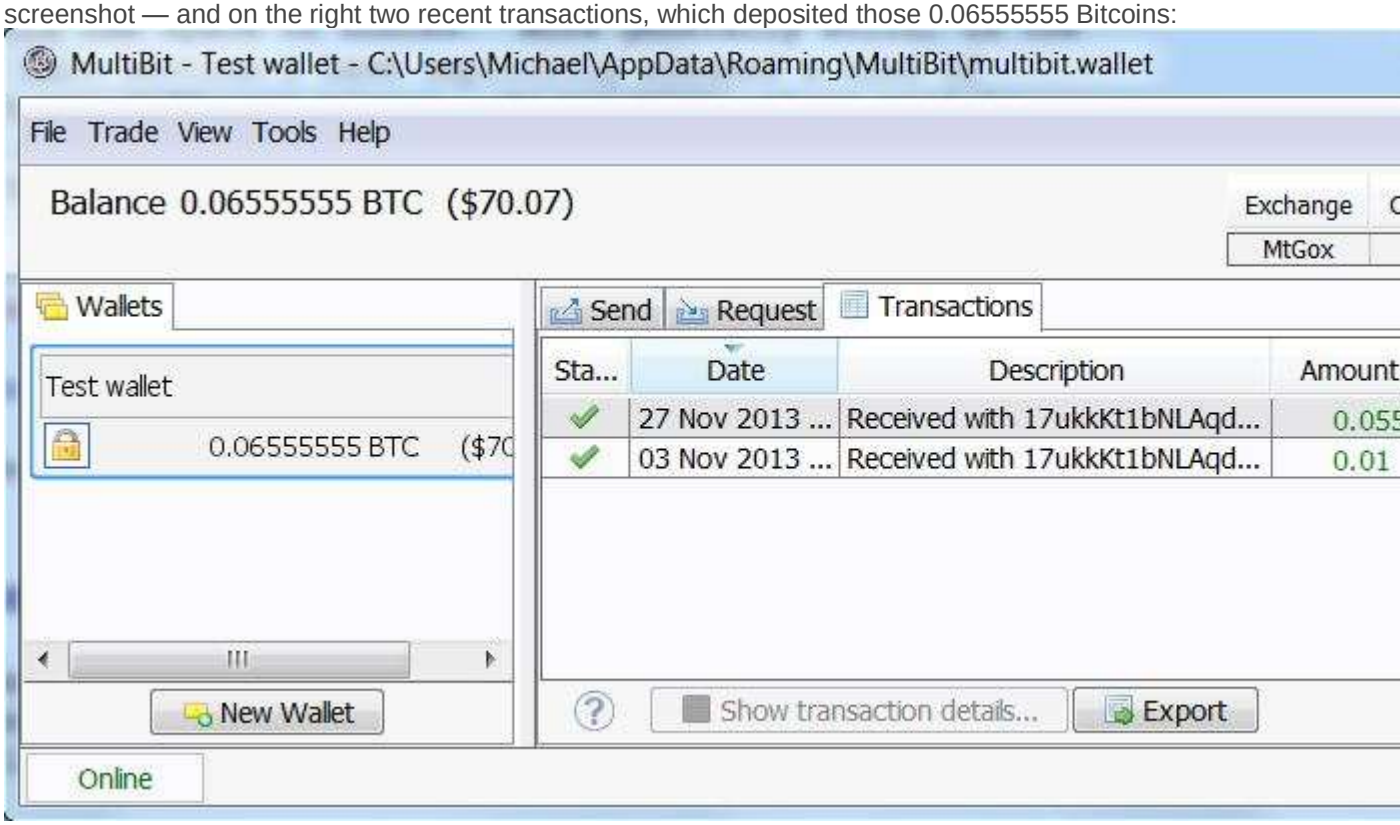
regain the lead. This balancing strategy will have only a small chance of success. But while the probability is small, it will certainly be much larger than in the standard protocol, with high variance in the time to validate a block. Is there a way of avoiding this problem?

- Suppose Bitcoin mining software always explored nonces starting with $x = 0$, then $x = 1, x = 2, \ldots$. If this is done by all (or even just a substantial fraction) of Bitcoin miners then it creates a vulnerability. Namely, it's possible for someone to improve their odds of solving the proof-of-work merely by starting with some other (much larger) nonce. More generally, it may be possible for attackers to exploit any systematic patterns in the way miners explore the space of nonces. More generally still, in the analysis of this section I have implicitly assumed a kind of symmetry between different miners. In practice, there will be asymmetries and a thorough security analysis will need to take account of those asymmetries.

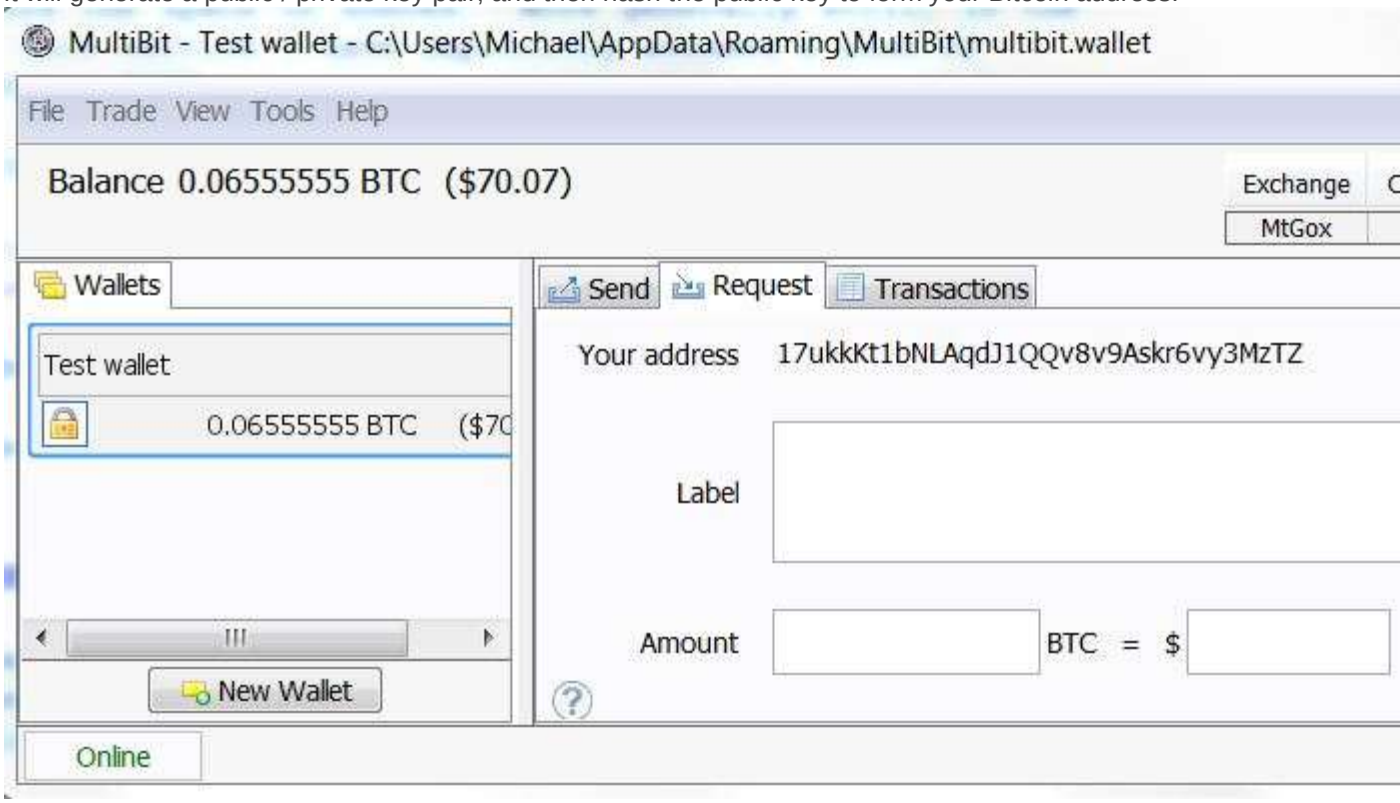
Bitcoin

Let's move away from Infocoin, and describe the actual Bitcoin protocol. There are a few new ideas here, but with one exception (discussed below) they're mostly obvious modifications to Infocoin.

To use Bitcoin in practice, you first install a **wallet** program on your computer. To give you a sense of what that means, here's a screenshot of a wallet called **Multbit**. You can see the Bitcoin balance on the left — 0.06555555 Bitcoins, or about 70 dollars at the exchange rate on the day I took this screenshot — and on the right two recent transactions, which deposited those 0.06555555 Bitcoins:



Suppose you're a merchant who has set up an online store, and you've decided to allow people to pay using Bitcoin. What you do is tell your wallet program to generate a *Bitcoin address*. In response, it will generate a public / private key pair, and then hash the public key to form your Bitcoin address:



You then send your Bitcoin address to the person who wants to buy from you. You could do this in email, or even put the address up publicly on a webpage. This is safe, since the address is merely a hash of your public key, which can safely be known by the world anyway. (I'll return later to the question of why the Bitcoin address is a hash, and not just the public key.)

The person who is going to pay you then generates a *transaction*. Let's take a look at the data from an **actual transaction** transferring 0.31900000 bitcoins. What's shown below is very nearly the raw data. It's changed in three ways: (1) the data has been deserialized; (2) line numbers have been added, for ease of reference; and (3) I've abbreviated various hashes and public keys, just putting in the first six hexadecimal digits of each, when in reality they are much longer. Here's the data:

```
1.  {"hash":"7c4025...",
2.  "ver":1,
3.  "vin_sz":1,
4.  "vout_sz":1,
5.  "lock_time":0,
6.  "size":224,
7.  "in":[
```

```
8.    {"prev_out":

9.      {"hash":"2007ae...",

10.      "n":0},

11.    "scriptSig":"304502... 042b2d..."}],

12. "out":[

13.   {"value":"0.31900000",

14.    "scriptPubKey":"OP_DUP OP_HASH160 a7db6f OP_EQUALVERIFY OP_CHECKSIG"}]}
```

Let's go through this, line by line.

Line 1 contains the hash of the remainder of the transaction, `7c4025...`, expressed in hexadecimal. This is used as an identifier for the transaction.
Line 2 tells us that this is a transaction in version 1 of the Bitcoin protocol.

Lines 3 and 4 tell us that the transaction has one input and one output, respectively. I'll talk below about transactions with more inputs and outputs, and why that's useful.

Line 5 contains the value for `lock_time`, which can be used to control when a transaction is finalized. For most Bitcoin transactions being carried out today the`lock_time` is set to 0, which means the transaction is finalized immediately.
Line 6 tells us the size (in bytes) of the transaction. Note that it's not the monetary amount being transferred! That comes later.

Lines 7 through 11 define the input to the transaction. In particular, lines 8 through 10 tell us that the input is to be taken from the ouput from an earlier transaction, with the given `hash`, which is expressed in hexadecimal as `2007ae...`. The `n=0` tells us it's to be the first output from that transaction; we'll see soon how multiple outputs (and inputs) from a transaction work, so don't worry too much about this for now. Line 11 contains the signature of the person sending the money, `304502...`, followed by a space, and then the corresponding public key, `04b2d...`. Again, these are both in hexadecimal.
One thing to note about the input is that there's nothing explicitly specifying how many bitcoins from the previous transaction should be spent in this transaction. In fact, *all* the bitcoins from the `n=0`th output of the previous transaction are spent. So, for example, if the `n=0`th output of the earlier transaction was 2 bitcoins, then 2 bitcoins will be spent in this transaction. This seems like an inconvenient restriction – like trying to buy bread with a 20 dollar note, and not being able to break the note down. The solution, of course, is to have a mechanism for providing change. This can be done using transactions with multiple inputs and outputs, which we'll discuss in the next section.
Lines 12 through 14 define the output from the transaction. In particular, line 13 tells us the value of the output, 0.39 bitcoins. Line 14 is somewhat complicated. The main thing to note is that the string `a7db6f...` is the Bitcoin address of the intended recipient of the funds (written in hexadecimal). In fact, Line 14 is actually an expression in Bitcoin's scripting language. I'm not going to describe that language in detail in this post, the important thing to take away now is just that `a7db6f...` is the Bitcoin address.

You can now see, by the way, how Bitcoin addresses the question I swept under the rug in the last section: where do Bitcoin serial numbers come from? In fact, the role of the serial number is played by transaction hashes. In the transaction above, for example, the recipient is receiving 0.39 Bitcoins, which come out of the first output of an earlier transaction with hash `2007ae...` (line 9). If you go and look in the block chain for that transaction, you'd see that its output comes from a still earlier transaction. And so on.

There are two clever things about using transaction hashes instead of serial numbers. First, in Bitcoin there's not really any separate, persistent "coins" at all, just a long series of transactions in the block chain. It's a clever idea to realize that you don't need persistent coins, and can just get by with a ledger of transactions. Second, by operating in this way we remove the need for any central authority issuing serial numbers. Instead, the serial numbers can be self-generated, merely by hashing the transaction.

In fact, it's possible to keep following the chain of transactions further back in history. Ultimately, this process must terminate. This can happen in one of two ways. The first possibilitity is that you'll arrive at the very first Bitcoin transaction, contained in the so-called **Genesis block**. This is a special transaction, having no inputs, but a 50 Bitcoin output. In other words, this transaction establishes an initial money supply. The Genesis block is treated separately by Bitcoin clients, and I won't get into the details here, although it's along similar lines to the transaction above. You can see the deserialized raw data **here**, and read about the Genesis block **here**.

The second possibility when you follow a chain of transactions back in time is that eventually you'll arrive at a so-called *coinbase transaction*. With the exception of the Genesis block, every block of transactions in the block chain starts with a special coinbase transaction. This is the transaction rewarding the miner who validated that block of transactions. It uses a similar but not identical format to the transaction above. I won't go through the format in detail, but if you want to see an example, see**here**. You can read a little more about coinbase transactions **here**.

Something I haven't been precise about above is what exactly is being signed by the digital signature in line 11. The obvious thing to do is for the payer to sign the whole transaction (apart from the transaction hash, which, of course, must be generated later). Currently, this is *not* what is done – some pieces of the transaction are omitted. This makes some pieces of the transaction **malleable**, i.e., they can be changed later. However, this malleability does not include the amounts being paid out, senders and recipients, which can't be changed later. I must admit I haven't dug down into the details here. I gather that this malleability is under discussion in the Bitcoin developer community, and there are efforts afoot to reduce or eliminate this malleability.

Transactions with multiple inputs and outputs

In the last section I described how a transaction with a single input and a single output works. In practice, it's often extremely convenient to create Bitcoin transactions with multiple inputs or multiple outputs. I'll talk below about why this can be useful. But first let's take a look at the data from an **actual transaction**:

```
1. {"hash":"993830...",

2. "ver":1,
```

```
3. "vin_sz":3,
4.  "vout_sz":2,
5.  "lock_time":0,
6.  "size":552,
7.  "in":[
8.    {"prev_out":{
9.      "hash":"3beabc...",
10.        "n":0},
11.     "scriptSig":"304402... 04c7d2..."},
12.    {"prev_out":{
13.        "hash":"fdae9b...",
14.        "n":0},
15.      "scriptSig":"304502... 026e15..."},
16.    {"prev_out":{
17.        "hash":"20c86b...",
18.        "n":1},
19.      "scriptSig":"304402... 038a52..."}],
20.  "out":[
21.    {"value":"0.01068000",
22.      "scriptPubKey":"OP_DUP OP_HASH160 e8c306... OP_EQUALVERIFY OP_CHECKSIG"},
23.    {"value":"4.00000000",
24.      "scriptPubKey":"OP_DUP OP_HASH160 d644e3... OP_EQUALVERIFY OP_CHECKSIG"}]}
```

Let's go through the data, line by line. It's very similar to the single-input-single-output transaction, so I'll do this pretty quickly.

Line 1 contains the hash of the remainder of the transaction. This is used as an identifier for the transaction.

Line 2 tells us that this is a transaction in version 1 of the Bitcoin protocol.

Lines 3 and 4 tell us that the transaction has three inputs and two outputs, respectively.

Line 5 contains the `lock_time`. As in the single-input-single-output case this is set to 0, which means the transaction is finalized immediately.

Line 6 tells us the size of the transaction in bytes.

Lines 7 through 19 define a list of the inputs to the transaction. Each corresponds to an output from a previous Bitcoin transaction.

The first input is defined in lines 8 through 11.

In particular, lines 8 through 10 tell us that the input is to be taken from the `n=0`th output from the transaction with `hash 3beabc...`. Line 11 contains the signature, followed by a space, and then the public key of the person sending the bitcoins.
Lines 12 through 15 define the second input, with a similar format to lines 8 through 11. And lines 16 through 19 define the third input.

Lines 20 through 24 define a list containing the two outputs from the transaction.

The first output is defined in lines 21 and 22. Line 21 tells us the value of the output, 0.01068000 bitcoins. As before, line 22 is an expression in Bitcoin's scripting language. The main thing to take away here is that the string `e8c30622...` is the Bitcoin address of the intended recipient of the funds.
The second output is defined lines 23 and 24, with a similar format to the first output.

One apparent oddity in this description is that although each output has a Bitcoin value associated to it, the inputs do not. Of course, the values of the respective inputs can be found by consulting the corresponding outputs in earlier transactions. In a standard Bitcoin transaction, the sum of all the inputs in the transaction must be at least as much as the sum of all the outputs. (The only exception to this principle is the Genesis block, and in coinbase transactions, both of which add to the overall Bitcoin supply.) If the inputs sum up to more than the outputs, then the excess is used as a *transaction fee*. This is paid to whichever miner successfully validates the block which the current transaction is a part of.
That's all there is to multiple-input-multiple-output transactions! They're a pretty simple variation on single-input-single-output-transactions.

One nice application of multiple-input-multiple-output transactions is the idea of*change*. Suppose, for example, that I want to send you 0.15 bitcoins. I can do so by spending spending money from a previous transaction in which I received 0.2 bitcoins. Of course, I don't want to send you the entire 0.2 bitcoins. The solution is to send you 0.15 bitcoins, and to send 0.05 bitcoins to a Bitcoin address which I own. Those 0.05 bitcoins are the change. Of course, it differs a little from the change you might receive in a store, since change in this case is what you pay yourself. But the broad idea is similar.

Conclusion

That completes a basic description of the main ideas behind Bitcoin. Of course, I've omitted many details – this isn't a formal specification. But I have described the main ideas behind the most common use cases for Bitcoin.

While the rules of Bitcoin are simple and easy to understand, that doesn't mean that it's easy to understand all the consequences of the rules. There is vastly more that could be said about Bitcoin, and I'll investigate some of these issues in future posts.

For now, though, I'll wrap up by addressing a few loose ends.

How anonymous is Bitcoin? Many people claim that Bitcoin can be used anonymously. This claim has led to the formation of marketplaces such as **Silk Road**(and various successors), which specialize in illegal goods. However, the claim that Bitcoin is anonymous is a myth. The block chain is public, meaning that it's possible for anyone to see every Bitcoin transaction ever. Although Bitcoin addresses aren't immediately associated to real-world identities, computer scientists have done a**great deal of work** figuring out how to de-anonymize "anonymous" social networks. The block chain is a marvellous target for these techniques. I will be extremely surprised if the great majority of Bitcoin users are not identified with relatively high confidence and ease in the near future. The confidence won't be high enough to achieve convictions, but will be high enough to identify likely targets. Furthermore, identification will be retrospective, meaning that someone who bought drugs on Silk Road in 2011 will still be identifiable on the basis of the block chain in, say, 2020. These de-anonymization techniques are well known to computer scientists, and, one presumes, therefore to the NSA. I would not be at all surprised if the NSA and other agencies have already de-anonymized many users. It is, in fact, ironic that Bitcoin is often touted as anonymous. It's not. Bitcoin is, instead, perhaps the most open and transparent financial instrument the world has ever seen.

Can you get rich with Bitcoin? Well, maybe. Tim O'Reilly **once said**: "Money is like gas in the car – you need to pay attention or you'll end up on the side of the road – but a well-lived life is not a tour of gas stations!" Much of the interest in Bitcoin comes from people whose life mission seems to be to find a *really big* gas station. I must admit I find this perplexing. What is, I believe, much more interesting and enjoyable is to think of Bitcoin and other cryptocurrencies as a way of enabling new forms of collective behaviour. That's intellectually fascinating, offers marvellous creative possibilities, is socially valuable, and may just also put some money in the bank. But if money in the bank is your primary concern, then I believe that other strategies are much more likely to succeed.

Details I've omitted: Although this post has described the main ideas behind Bitcoin, there are many details I haven't mentioned. One is a nice space-saving trick used by the protocol, based on a data structure known as a **Merkle tree**. It's a detail, but a splendid detail, and worth checking out if fun data structures are your thing. You can get an overview in the **original Bitcoin paper**. Second, I've said little about the**Bitcoin network** – questions like how the network deals with denial of service attacks, how nodes **join and leave the network**, and so on. This is a fascinating topic, but it's also something of a mess of details, and so I've omitted it. You can read more about it at some of the links above.

Bitcoin scripting: In this post I've explained Bitcoin as a form of digital, online money. But this is only a small part of a much bigger and more interesting story. As we've seen, every Bitcoin transaction is associated to a script in the Bitcoin programming language. The scripts we've seen in this post describe simple transactions like "Alice gave Bob 10 bitcoins". But the scripting language can also be used to express far more complicated transactions. To put it another way, Bitcoin is*programmable money*. In later posts I will explain the scripting system, and how it is possible to use Bitcoin scripting as a platform to experiment with all sorts of amazing financial instruments.

Footnote

[1] In the United States the question "Is money a form of speech?" is an important legal question, because of the protection afforded speech under the US Constitution. In my (legally uninformed)

opinion digital money may make this issue more complicated. As we'll see, the Bitcoin protocol is really a way of standing up before the rest of the world (or at least the rest of the Bitcoin network) and avowing "I'm going to give such-and-such a number of bitcoins to so-and-so a person" in a way that's extremely difficult to repudiate. At least naively, it looks more like speech than exchanging copper coins, say.