Zero-Knowledge Proofs

The key of zero-knowledge proofs is that you have a prover who wants to prove that they know something without actually playing their hand and a verifier who's keeping them accountable, but should not learn anything about the actual answer.

Graph Isomorphisms

Let's say you have two graphs, each with the same total number of edges and vertices, and you have a hunch that they're the same graph drawn in different ways. How can you verify that in an efficient manner?

We don't know a poly-time algorithm for this problem. But if I do come up with an isomorphism between two graphs (i.e., a permutation of the vertices), how can I prove this claim with a zero-knowledge proof?

To do so, I come up with a new graph that's an isomorphism of both input graphs. The verifier requests that I prove that this isomorphism is valid for one of the input graphs; to do so, I provide a permutation of the vertices between their selected graph and the new graph I created. The key is: providing this isomorphism tells the verifier nothing about the isomorphism between the two input graphs. If I lie, I'll be caught with a 50% probability; so we can run this protocol multiple times to come up with a probabilistic assurance of correctness.

Discrete Log Problem

We can come up with a similar protocol for the discrete log problem, i.e., the problem of finding x such that $a^x \equiv b \bmod p$.

Say we, as the prover, can come up with such an x, but we don't want to reveal the answer to the verifier. To do so, we come up with $x_1 + x_2 \equiv x \bmod (p-1)$ and send the verifier $a^{x_1} = c$ and $a^{x_2} = d$. The verifier then checks that $cd = b$ and asks the prover to reveal either x_1 or x_2 to show that they didn't just make up c and d. (Zero-knowledge comes from the fact that, even if the verifier knows one of x_1 or x_2, they still cannot solve the discrete log problem and deduce the other.)

Again, this is a probabilistic proof: there's some chance that we merely guessed correctly, or even that the prover just chose x_1 and didn't actually know x_2, but lucked out in that the verifier requested that they reveal x_1. So the verifier can run this protocol multiple times to get a probabilistic guarantee that we're not lying to them.

Non-Interactive Proofs

In many cases, we won't be able to challenge a prover, as in the schemes above. To run this in a non-interactive manner, require the prover to pick 160 different pairs of (x_1, x_2), which we'll label $(x_1^1, x_2^2), ..., (x_1^{160}, x_2^{160})$. They then compute $H(a^{y_1}||...||a^{y_{160}})$ and publish y_i if $h_i = 0$ (i.e., the ith bit of h) and z_i otherwise. In this way, we can avoid interactivity by performing several "rounds" of the protocol at once.

Zerocoin

Zerocoin is a protocol-level mixing; that is, it has mixing capability built into its protocol, giving you a cryptographic guarantee of mixing. Unfortunately, Zerocoin is not currently compatible with Bitcoin.

How would it work?

At a high level, Zerocoin would consist of two different currencies interoperating with each other. For sake of argument, we'll call them *Basecoin* (a Bitcoin-lke Altcoin) and *Zerocoin*, an extension of Basecoin with the property that it can be converted into Basecoin and back again. Converting between currencies breaks the link between the original and new Basecoin; that is, these links cannot be inferred, even by miners.

A Zerocoin will be a cryptographic proof that you owned a Basecoin and made it unspendable. Miners can verify these proofs and their existence gives you the right to redeem a new Basecoin.

Cryptographic challenges

- We need to come up with some sort of zero-knowledge proof. The types of proofs we'll want to provide will be of the form: "I know an input that hashes to da39a3ee5b", or "I now an input that hashes to some hash in the following set."
- We need to make sure that each of these proofs can only be "spent" once.

Commitment

To mint a Zerocoin, we'll need to use commitment. The procedure is as follows:

1. Generate a serial number S, which will eventually be made public.
2. Generate a random secret r, which will *never* be public (to provide unlinkability).
3. Compute $H(S, r)$ (this is a simplification: in practice, use the Pedersen commitment scheme).

We put $H(S, r)$–but neither S nor r–on the blockchain by creating a special mint transaction. This transaction contains a hash pointer to the previous transaction it's spending; this latter transaction will be a Basecoin transaction, while the new transaction will be a Zerocoin transaction.

To later spend the coin, you need to convince the miners that you know the serial number corresponding to *any one* of the Zerocoins on the chain. The steps are as follows:

1. Reveal the serial number S. This will *not* reveal which coin you're trying to spend, but *will* protect against double-spends, as the miners will store the spent S values on the blockchain.

2. Create a zero-knowledge proof such that: I know a number r such that $H(S, r)$ is one of the Zerocoins in the blockchain (see the Zerocoin paper for details).

3. Pick an arbitrary Zerocoin in the blockchain and use it as input to your new transaction.

Anonymity

Since r is secret, no one can figure out which Zerocoin corresponds to the serial number S. Thus, while you can redeem a coin, no one can figure out which coin it was.

Zerocoin is also "efficient": the proof is a giant disjunction over all Zerocoins, but the proof itself is relatively small due to some clever tricks and optimizations in the Zerocoin paper.

Zerocash

Zerocash is similar to Zerocoin, except it eliminates the need for Basecoin. Essentially, the differences are as follows:

- It uses different cryptography for its proofs to make it more efficient.

- It's able to run the system without Basecoin.

All transactions are Zerocoins, with merging and splitting supported.

The implication is that Zerocash is **untraceable**. You can put transaction values inside the envelope, with the ledger merely recording the *existence* of transactions. Only the owner of an address knows about which transactions have been sent to an address, their denominations, etc.

Initialization

The problem with Zerocash is that, to kick-off the system, a party is required to produce some random, secret inputs, which will be used to generate public parameters. These secret

inputs must then be securely destroyed. **No one** can know the secret inputs–they can be used to break the entire system and mint coins at-will.

There's been some discussion of having a known party generate these parameters on film and then destroy the computer they'd used, but this becomes a social trust problem more-so than a cryptography problem.

Levels of Anonymity

We've seen five levels of anonymity:

1. **Bitcoin**: subject to transaction graph analysis.
2. **Single mix**: transaction graph analysis, bad mixing.
3. **Multi mix**: side channels, bad mixes and peers.
4. **Zerocoin**: side channels (possibly).
5. **Zerocash**: none.

Zerocash does not even depend on the existence of other users in the system, unlike in the previous models we've seen. It's a truly untraceable system.