Anonymous e-cash via blind signatures

In 1982, David Chaum came up with a cryptocurrency involving a blind signature: *two-party protocol to create digital signature without signer knowing the input.* The protocol worked as follows:

1. User A withdraws an anonymous coin by sending the bank a random ID.
2. Bank signs the random ID, but **the bank knows neither the plaintext ID nor the output signature**.
3. User A can then spend the coin by showing others that the signature came from the bank.
4. User A sends coin to User B.
5. User B goes and updates with the bank by depositing the signed token and the ID that was inside the token (the bank did not know this ID).
6. Bank can verify that signing the plaintext ID leads to the signed token.
7. Puts the signature in the list of spent coins.

With this system, you have defense against double-spending but you have to trust the bank. The cryptography is only used for anonymization (i.e., the bank has no idea that User A and User B completed a transaction because they don't know which ID they gave User A).

At the time, people thought that anonymity was the most important property of a cryptocurrency system; as it turns out, decentralization is more desirable.

Blind signatures

Let's see the math behind this protocol.

In RSA, you have a public key (n, e) and a private key d. To sign a message m, you output $s = m^d \mod n$. To verify, you check that $m == s^e \mod n$.

For this two-party protocol, we'll use a variant (RSA Blind Signature). Everyone knows (n, e), and the bank knows the private key d. In this variant, Alice picks a random number r. For a message m, sends $(m)(r^e) \mod n$. The bank signs this entire message, returning:

$$\begin{aligned}(mr^e)^d \mod n &= m^d r^{ed} \mod n \\ &= m^d r \mod n\end{aligned}$$

Alice then divides by r, obtaining $m^d \mod n$, which is exactly the signature on the message. Note that the signer cannot decide the original message m, nor the signature m^r.

If Alice gets unlucky, she could withdraw a coin that's already been withdrawn, which would woe out poorly for her (as the bank will refuse when someone tries to redeem it). To avoid this, you need to make the RSA key space fairly large to provide probabilistic guarantees.

Comparisons to Bitcoin

Here, we have a central bank that protects against double-spending. In Bitcoin, we have this public ledger that lets *anyone* detect double-spending. Almost all of the Altcoins that we'll look at use this "public ledger" approach.

Ripple

Ripple is another Altcoin that takes a different approach.

If you try to do consensus in a traditional model, one of the problems you run into is that adversaries can create tons of copies of the software and construct a majority. To prevent this, we use a proof-of-work model in Bitcoin, where the number of copies is not important, but rather, the amount of computational power.

There are other ways to get around this problem:

- Remove decentralization and centralize node creation.
- Proof-of-stake: your influence stems from the amount of currency that you hold, rather than the number of nodes.

Even if there's no central authority, some nodes will still trust other nodes (i.e., you can vouch for your friends). We can use this *friend-to-friend* network to construct a consensus protocol. Ripple implements such a protocol.

Example protocol

This is not what Ripple uses, but is interesting to think about. Consider the following protocol:

1. At round 0, each node proposes the transactions that it thinks should be in the next block.
2. Each node aggregates the proposals from the nodes it trusts.
3. For round 1, each node outputs the majority set of its friends outputs from round 0 where *majority* is determined by whether or not a majority of your friends proposed the transaction.

4. In general, at round $n+1$, you output the majority-voted set of transactions based on your friends' outputs at round n.

To ensure convergence, you can increase the majority threshold over time.