[2018.03.10: updated with exit procedure]

Special thanks to Karl Floersch for discussion and coming up with much of this, as well as @danrobinson's earlier posts that expressed similar ideas.

Basically, we can design a version of Plasma with the following modifications:

1. Every single deposit corresponds to a unique coin ID; tokens are indivisible and cannot be merged.
2. Instead of storing transactions in a binary Merkle tree in order of txindex, we require them to be stored in either a sparse simple Merkle tree or a patricia tree, with the index being the ID of the coin that is spent.

Note that this now allows a user to have a somewhat compact proof that their coin is valid: all the transactions since the time the coin was deposited that represent that coin's history, plus a proof of non-inclusion for every block that does not contain a transaction spending the coin, to verify that the coin was not double spent. With n

coins and t

blocks, this proof has size t * log(n)

. If a user transfers a coin to another user, he could simply pass along the entire proof to that user.

Hence, a Plasma operator could simply maintain connections with each user, and every time they create a block they would publish to them only the proofs, not any data related to coins that they do not own. It's clearly the case that any data that is not part of these proofs could not be used to fraudulently exit or double-spend the user's coin, so the user is safe. Because coins are non-fungible, successfully defrauding other users cannot allow the Plasma contract to turn into a fractional reserve, as is possible in minimal viable plasma.

The Plasma chain operator could be sharded, so there is virtually no limit to the system's scalability from the point of view of either the chain operator or the users, though the limitation that if Plasma-like systems (and channel systems) start processing a very high transaction load, mass challenge attacks may overflow the blockchain and prevent some users from exiting or responding to challenges still remain. This kind of setup seems ideal for very-high-throughput but low or medium-state applications, like micropayments and exchanges.

Additionally, we can remove the need for confirmations. We do this by having the following exit procedure:

1. Anyone can exit their coin by providing the last two transactions in the coin's ownership history (ie. the coin they are exiting C and its parent P(C)).
2. An exit can be challenged in three ways: (i) provide a proof of a transaction spending C, (ii) provide a proof of a transaction spending P(C) that appears before C, (iii) provide a transaction C* in the coin's history before P(C)
3. A challenge of type (i) and (ii) blocks the exit immediately. A challenge of type (iii) can be responded to by providing the direct child of C*, which must be either equal to or before P(C)

This relies on honest users maintaining the key property that they never spend a coin until they have fully authenticated the entire history up to that coin. It could be the case that a Plasma chain starts including unavailable or invalid data while a transaction is in flight, in which case double spends or invalid spends could appear between P(C) and C; the slightly more complicated exit mechanism takes this into account.