See also: Plasma Cash Defragmentation

Another way to do Plasma Cash defragmentation is to have the operator

publish a permutation that contributes toward defragmenting coins. Consider a design as follows. The operator can, at any point, make a special transaction to commit the Merkle root of a permutation which has the property that consists of a set of swaps between pairs (eg. 1 2 3 4 5 6 -> 4 2 5 1 3 6 is ok, as it swaps (1,4) and (3,5), but 1 2 3 4 5 6 -> 2 5 6 4 1 3 is not ok). The operator commits the Merkle root of a tree where the value at each position is the difference between the position and the value in the permutation (eg. for 1 2 3 4 5 6 -> 4 2 5 1 3 6 this would be +3 0 +2 -3 -2 0).

Any exits, challenges or other on-chain operations involving data published before the permutation transaction is placed related to coin x

must now use branches at position p(x)

in the transaction trees (where p

is the permutation), along with branches of the permutation tree at x

and p(x)

; this de-facto reassigns ownership of coin x

to the owner of p(x)

.

For two weeks after the permutation transaction is placed, a special type of exit can be made which exits a coin using purely pre-permutation data; if some coin x

exits in this way, then the owner of x

that used to be the owner of p(x)

can publish the permutation branches, claiming p(x)

for themselves. This should be used by honest users if the Plasma operator fails to publish permutation branches for their coins. This data can later be pointed to in an adjudication battle to essentially claim that in this permutation round, it’s “really” the case that (if q = p(x)

), that x

does not exist and p(q) = q

.

If yellow pre-exits, then B is left without a coin, so B can use the permutation data to take ownership of green

.

Note that this is why the permutation must be a self-inverse: if there were cycles with arbitrary lengths, there could be arbitrarily long cascades of x

pre-exiting, leading to the new owner of x

claiming p(x)

, but wait p(x)

also pre-exited, so they need to claim p(p(x))

, and so on, potentially requiring one party to publish O(N)

data to chain.

You can show that it’s possible to use log(N)

such permutations to shuffle N coins in an arbitrary way; the simplest algorithm is a greedy algorithm which looks for places where c[i] \ne i

and swaps i

and c[i]

as much as possible; this will reduce the distance between the current permutation and your desired permutation by at least half in each iteration.

This approach does lead to old data needing more and more auxiliary branches over time to exit with; however, an ordinary user would only need to look up old data in one of two cases: (i) they did not make a transaction in a very long time and are exiting, or (ii) they are responding to a challenge made by another malicious user. A user can avoid (i) by sending to themselves every few months, and (ii) is at most 1:1 griefing factor, and can be avoided by making the procedure of sending to oneself every few months mandatory and disallowing challenges using very old data, or alternatively Plasma XT-style state commitments.

Edit: here’s some python code to play with defragmentations:
https://github.com/ethereum/research/blob/master/defrag/permutation2.py