Suppose that you have a Plasma chain that works by committing state roots to the main chain, along with a S*ARK (verified by the contract on the main chain that accepts the state roots) that proves that the state root is a valid state transition from the previous state root. That is, we have a state transition function STF

which processes a block and a state and outputs a new state and a list of exits, and the contract on the main chain maintains a state root R

, and if it receives a tuple (R', \Pi)

where \Pi

is a proof that there exists some block B

, old state S

, new state S'

and list of exits E

such that (i) STF(S, B) = (S', E)

, (ii) root(S) = R

, (iii) root(S') = R'

. If the proof passes, then the root is replaced by R'

and for each exit in E

the contract sends out the required amount of funds to the required recipient. This is essentially the architecture used by https://www.starkdex.io in its current form.

The problem with this architecture is that while an operator cannot steal funds, as only valid user exit operations can create an exit and those operations decrease the user's balance in the state, the operator can still cause all funds in the system to be locked up forever by shutting down.

We solve this problem as follows. First of all, users chain publish requests to exit to the contract on the main chain. For a new state root to be accepted, the proof must in addition to the three claims above prove a fourth claim: the exit list E

must include all exits published by users to the contract since the last root was published. Now, the child chain cannot progress at all without honoring users' exits.

Second, we add a mechanism where if the child chain has not made progress within some period (eg. 1 week), another 1-week period begins during which any user can vouch that they know the full state represented by some historical state root of the child chain, along with a deposit. After this second period ends, we make a list of all historical state roots that have been vouched for, in most-recent-to-oldest order. We add another 1-week period during which anyone can publish the full state corresponding to the most recent root to the chain that was vouched for

. If this full state is published, the chain processes exits from any user directly. If this state is not published for a week, the user that vouched for that state loses their deposit, and another 1-week period starts during which anyone can publish the full state corresponding to the next most recent

state root, and so forth.

Security argument

Suppose that you know a full state S

for which the root has been successfully published to chain. If you want to exit the child chain at any point in the future, you can publish an exit request.

Case 1

: the operator publishes a new root, which includes and successfully processes your exit (success).

Case 2

: the operator never publishes any more roots. After a week, you vouch for S

. After two weeks, either you can publish S

or someone else publishes a state more recent than S

(if someone else wins on vouching priority but fails to publish, eventually the queue will reach you), and at that point you can process your exit on chain (success).