I just ‘came up’ with an interesting design for MVP chains, reducing the withdrawal time while at the same time decreasing maintenance costs for the Plasma Manager (also coming along with some other improvements) and would love to get some feedback from the community on whether you see any security flaws. :)

1. Reducing Operating Cost:

The basic features of this concept have already been mentioned in the past as for instancehere on ethresearch. The idea is, not to store every block header on the root chain but to summarize a set of exactly N

blockheaders into a bigger merkle tree whose root is then stored on the main chain which allows for a much shorter block time on the plasma chain while at the same time reducing the operating cost of the plasma chain. The problem with this is that the less frequent blocks are being validated on the base layer, the more users of the plasma chain have to trust the Plasma Operator will not engage in malicious behavior.

1. Eliminating Need for Trust into a Plasma Manager:

As an extension to PoA, in this design, the Plasma Manager is required to stake R

coins of the base currency into the smart contract on the root chain. This stake functions like a key to be able to operate the plasma chain. If the stake does not match a certain amount, the Plasma Manager will not be able to store new block headers into the chain which will in turn alert users to exit the plasma chain.

After the stake has been deposited, the Plasma Manager can start creating blocks and add these in the form of a merkle root to the main chain, whenever N + N0

new blocks have been created and some predefined time T (in seconds)

has passed (it will become clear later why additional N0

blocks need to be produced before and which role T

plays in this concept). To ensure that no faulty blocks are being attached to the plasma chain, every node has the opportunity, should the Plasma Manager actually produce a faulty block, to submit the operators signature on this block to the smart contract, thereby, proving the malicious behavior. In this case, the account which reported the block is asked to correct the chain and produce the remaining blocks until another set of N + N0

new blocks can be stored on the base layer and will then be rewarded with the stake, the Plasma Operator locked up inside the smart contract earlier. Should the reporter also engage in malicious behavior, the aforementioned procedure will simply repeat itself until someone produces a valid set of blocks. Also, as stated before, in order to keep the chain running, the Plasma Operator again needs to store R

coins into the smart contract on the base chain. (N0

and T

are to give the reporter enough time to check the submitted blocks and report if necessary)

A pleasant effect of this mechanism is that not just when stored in the main chain but already after being signed by the Operator, a block can be considered valid. This is since for every block, the operator signs with his ECDS, he essentially says: ‘I bet you R

coins that this block is valid!’. Of course, R

could also be a function R(v)

, dependent on the overall amount of value being transferred inside this block in order to make sure the reward for taking the risk of manipulating the chain would not be worth it.

However, since this scenario is ideally never going to happen, there probably has to be an additional incentive for validators to keep checking the chain (e.g. randomly from time to time including a particular kind of malicious transaction (which would not harm the chain if not detected) on purpose into a block which will be detected by the smart contract, not as an attempt to defraud but a simple reward for the sender of the transaction (or the first 10 reporters, …)).

(In a later version, when the chain has caught traction, this could perhaps even be extended to many different block producers. However, this is just a quick thought I had - it is not worked out.)

1. Decreasing Withdrawal Time:

The approach to this design has previously been brought uphere (burn proof as 2nd exit method). Yet, this mention leaves the problem arising in the event of a block-withholding-attack unsolved.

In this design, since the Plasma Manager has to produce another N0

blocks before they can store the merkle root of the previously created N

blocks on the base layer, in this case, either, the checking nodes would notice that they are not getting new blocks and, thereby, assume the operator is performing a block-withholding-attack so that they can alarm the chain and everybody exits in time (Here, the chain will enter a special exit-mode - further details will follow.). Yet, the Plasma Manager could also just continue feeding valid blocks to the chain while simultaneously creating another chain which is not being broadcasted to the network with the intention to include this one instead of the chain, send to the network. The solution to this is fairly simple: Should this happen, its an easy game for the reporters since they can immediately proof that the block creator signed a block which has not been included in the merkle tree, stored on the root chain (blocknumber prevents malicious behavior on the part of the reporters). The reward for the reporter who found the mistake first, again, would be R

.

Admittedly, if the block producer is clever, they will not try to keep up the illusion of still running the chain correctly as this is handing reporters the missing piece to proving the Plasma Manager's wrong-doing. Without submitting faulty blocks, proving malicious behavior is pretty much impossible. Now, as a basis, take the preceding design and add an exit mode to it which will only be entered if a certain condition* is met. Since the Plasma Manager is obliged to produce N + N0

blocks and wait for at least T

seconds, starting after the last submission of blocks to the root chain, the Plasma Manager cannot include a set of blocks into the chain immediately after the (N + N0)

th block has been created and included. This should give the reporters and at least a portion of the rest of the network enough time to exit the chain in the 'conventional' way. Since probably only a rather small number of nodes in the network will be monitoring the two chains, still, not everybody would exit the chain in time. This is where the exit mode steps in*. (btw. the exit-mode is also entered when a reporter successfully reports a malicious block)

When I am making a transaction and nothing happens for 10 or 15 minutes (presuming the block time is somewhere at 1-2 seconds), I am alarmed there might be something shady going on. When now a root is being submitted to the chain but one still has not received any updates/blocks, they will definitely exit the chain. For that reason: should more than a certain percentage of users try to exit the chain within the same timeframe, the chain automatically enters the exit-mode in which only the traditional exit method* for plasma chains, involving a challenging period as security mechanism, is accepted. Furthermore, when a burn proof is submitted, it will only be admissible if it has been included at least in the second last preceding stack of blocks, stored on the main chain so that this is also protected from incorrectly created UTXOs by the Plasma Manager through the reporting mechanism. (Even if the Plasma Manager had created UTXOs out of thin air while withholding blocks, he could only submit them after the next root has been stored which takes at least T

seconds. During this time the users can exit and get the chain to switch into exit-mode.)

Benefits:

- Lower maintenance cost for the Plasma Manager
- Users can quicker withdraw tokens, making depositing coins more attractive
- No need for users to storing the whole chain on their device (better scalability)
- Shorter block time for higher refresh rates of the chain status

Downside:

- Reporting might get dis-incentivized by temporary spikes in transaction fees on the root chain
- Expensive stake for Plasma Manager (one-time investment)

(I hope I did not overlook something majorly obvious

)