TL;DR

We propose an alternative light-client concept for plasma chain clients. This solution requires clients to have even fewer resources (memory, CPU and bandwidth) as compared to plasma cash requirements. The solution is based on zk-snark.

Background

Plasma-cash brought a lot of innovation to light. Plasma cash eliminated confirm signatures, enabled a low resource validation of plasma chains for clients and tackled the mass exit network overload challenge. Just to name the biggest 3 advantages.

But the regular plasma is catching up. The "More Viable Product" is eliminating confirm signatures for plasma as well. The mass exit network overload challenge can be tackled by longer exit periods (more than 2 weeks) and great services.

This post shows how plasma can be implemented, such that the resource requirements for clients are even lower than in plasma cash using zk-snarks.

You see, plasma is not just catching up, plasma is plasma

No more complicated coin splitting and merging or cash exchangers are required.

Zk-snark abilities

Let P be a program which takes two inputs. The first input i is meant to be public. The second input w is a witness which can be kept private. We use o for an output of P.

Given a program P, an input i and an output o, if a prover knows a witness w such that o = P(i, w) then the prover can produce a proof where:

- (space density) The proof is very short (just a few hundred bytes even for a very large program; e.g. Zcash achieves 288 bytes)
- (time density) The proof verifies in time linear to the lengths of i and o (and does not depend on w)
- (zero knowledge) The proof leaks nothing about the execution of P or the contents of w other than the existence of w

Zk-snark for plasma

Let's change the concept from above for our purposes.

We set:

i = (i_1, i_2): I_1 is Merkle root hash of all valid, spendable utxo of our plasma chain and i_2 is the list of new deposits into the plasma chain and exits from it.

w = (w_1, w_2): w_1 is the list of valid spendable utxo, w_2 is the list of transactions of a plasma block.

o = (o_1, o_2): o_1 is the Merkle hash of the new valid, spendable utxo after processing the block's transactions. o_2 is the Merkle hash of the w_2.

And furthermore we assume that P runs the following logic:

- check i == Merkle_root_hash(w_1)
- check that each transaction is signed correctly
- incorporates i_2 to the list of valid, spendable utxos w_1,
- check that for each transaction the inputs are valid
- updates with each transaction the valid, spendable utxos
- build the Merkle hash of the newest list of valid, spendable utxo after processing all transactions
- calculate the Merkle hash of all transaction processed.

With this setup, verification is very easy. For each block, the chain operator publishes o_1 to the root chain and publishes o_2 and the zk-snark proof of o=P(i,w) to all clients. For verifying a correct block transition from one block to another block, the clients only need to verify:

- zk-snark proof and o_2 is available
- o_2 coincides with the published hash to the root chain,

- zk-snark proof for P(i,w) is valid, where i is o_1 from the previous block (the hash of the previous list of valid, spendable utxos)

If the client checks only these 3 things, he knows, that his utxo was not spent and all utxos have a valid origin. Hence, he knows that he will be able to withdraw his utxos later. If any of these 3 checks fails, he is just supposed to withdraw his uxtos from the chain.

Analysis

This described solution keeps everything related to zk-snarks off the chain and hence doesn't cost any gas, in contrast to other proposed solutions. The creation of these proofs is the most resource intense component, but it can be done by the central plasma chain operator. Verifying them can be done very, very efficiently by the clients.

Currently, I am trying to implement this zk-snark protocol with this library. If you see any challenges with this approach, please let me know.

edits:

If there is a new exit on the main chain, clients need to know that it is a legit exit. Hence, the chain operator could provide a Merkle proof that the exit is in current w_1. If the clients receive these Merkle proofs, all is good. But if the chain operator turns malicious, then everyone needs to get the data for the challenging process from somewhere else. So there needs to be at least some trust-worthy data centers, which are storing all transactions and are willing to distribute them later.