Plasma snapp-1-bit - 1 bit on-chain data per transaction

In the previous Plasma snapp spec, we described a plasma chain using snark proofs to ensure correct state transitions. In case of data unavailability, we would roll back the correct chain until the last block for which we have all the data available. These roll-backs can be quite complicated. In this post, we show that we can tackle data availability better with a mix of double-signing and snarks.

Special thanks to @keyvank for giving the inspiration, and @fleupold for helping to think this through.

Idea:

One of the fundamental problems with snark application is that the chain operator could post valid new blocks, but no one has the data needed to calculate the newest state from these blocks. Especially for payment senders, this is very problematic, as they have to convince the payment receiver that the newest block has added the payment to their balance. In this construction, we enforce data-availability in the newest plasma block for all transaction senders for this block. We do this by introducing a double-signature mechanism: All transactions data, which will be allowed to be included in the next block, has to be confirmed by transaction senders by double-signing it.

It would roughly work like this: all transactions of a potential new block are collected and hashed by the chain operator. This hash is sent to all transactions owners, who will sign the hash only if they have seen all transactions belonging to this hash. Now, the snark enforces that the operator includes only transactions for which he also has received the double-sign message. Hence, these included transactions are a subset of all transactions proposed for this block. The trick: Now every participant has seen all potential data from this block and all transaction owners can proof statements about the latest state.

This will be combined with special non-interactive withdrawals, which allows withdrawals to be processed even if there is a data-unavailability on tip of the chain.

Specification:

(This is minimal spec only)

We store the current state of the plasma chain in a Pederson Hash called StateHash

. This Pederson Hash is calculated from the list of items describing the state. Each item of the list stores (public key, token, balance)

, where the public key

is the key of the account, thetoken

is number associated with a token address and the balance

is the balance the accounts hold.

Every state transition based on transfers in the plasma chain comes with a snark ensuring a correct execution. The snarks checks the right execution of the following programm:

P_transfer(StateHash_i, ProposalTransactionBundleHash, [witness data]) = (StateHash_(i+1), DoubleSignBitmapHash, LastAccountTouchHash)

Here, the StateHash_i

is the stateHash from a previous state, which gets transformed based on the transaction to the new StateHash_(i+1)

. Before a block of the plasma chain is generated, the plasma operator collects the transactions. Then, he hashes these transactions to a ProposalTransactionBundleHash

. This hash and all transactions are sent by the operator to everyone who has passed him a transaction. The owner of a transaction needs to verify the ProposalTransactionBundleHash

, signs it and sends back the signature to the operator. The operator checks the double signatures and builds a map DoubleSignBitmap

, where he puts a 1 at the i-th place in the map if he received a double sign message for the i-th transaction. DoubleSignBitmapHash

is just the hash of the bitmap. We also maintain for each account a L-bit number indicating the distance between the current block and the block of the last balance change of this account. This is important for the withdrawals later.

Now, P_transfer

is doing the following:

- Checks that the current state data from the witness gets hashed to StateHash
- Checks that the transaction data from the witness hashes to

ProposalTransactionBundleHash

- Checks that the DoubleSignBitmap

from the bitmap hashes to

DoubleSignBitmapHash

- Checks that the DoubleSignBitmap from the witness was constructed correctly, by checking

the double-sign signatures

- And applies all transactions with a positive 1 in the bitmap to calculate the new state for the output: StateHash_(i+1).
- And increase the number in the LastAccountTouch list by 1 for each account not sending a transaction and set them 0 otherwise.
- Then hash the list LastAccountTouch together to get LastAccountTouchHash.

The operator can publish the snark proof together with the DoubleSignBitmap

on the blockchain. The smart contract would accept the snark if the snark is valid and DoubleSignBitmap

hashes to DoubleSignBitmapHash

. Note that DoubleSignBitmap

only needs to be published as payload and does not need to be stored.

With this construction, we have ensured that anyone included in the latest plasma block is fully aware of all data and can easily calculate the new state of the plasma chain from the transactions data, the previous state, and the DoubleSignBitmap

.

Deposits and exits:

Deposits are handled mostly like in the original plasma snapp. Deposits create new hashes from deposit information. Then a dedicated snark retrieves this information by checking the hashes and then processes the deposits. The only difference is that the LastAccountTouch will be set to the max value: 2^L-1, if a new account is doing its first deposit.

Exits are a little bit more complex, as there are two cases:

Exits with an online operator:

Users intending to withdraw their balance will register this on the smart contract. They will send the information over to the smart contract, which then creates from this information a hash. A dedicated withdraw-snark would get these hashes as public input and process them. Together with the publishment of the withdraw-snark, the operator would publish the current LastAccountTouch list as payload and its correctness will be verified by hashing and comparing it with the snark variable: LastAccountTouchHash. If the operator wants to publish new blocks, he will be forced by the smart contract to process the pending exit-queue by publishing these withdraw-snarks within K blocks.

Notice that these exits are kinda expensive as the operator needs to publish the LastAccountTouch list, which would be (= 25k *L/8 * 68 gas) gas for 25k accounts. Hence, K should be so big that the operator has to do it only once a month. For fast withdrawals, decided services will do a swap with on-chain funds.

Exist with an offline operator:

If the operator has not published new blocks for X days, then the chain will freeze and no transfers will be possible anymore. Now people will be rewarded for publishing the latest LastAccountTouch list from the highest plasma block. If we have 50k accounts, this means people have to upload only 50k * L bits (= 50k * L/8 * 68 gas).

Once this LastAccountTouch list is determined from the highest block h, we allow each account to withdraw by providing a snark proofing their balance for any block between h and h-(L-bit number from the LastAccountTouch). This means everyone will be able to withdraw the balance he/she has at the tip - block h - of the plasma chain.

To ensure that a client is always able to generate this snark proof for the exit, he/she must have the data of one block between h and h-(L-bit number from the LastAccountTouch). Hence, they have to come online and get the newest state each 2^L - K blocks. Notice that the operator can never steal in any case funds, funds could only get stuck.

To ensure that a client will be able to force the LastTouchIndex list to be the latest one after a chain freeze, the clients need to be online every X days.

We think that a good trade-off is:

Enforced withdraw := K blocks == 60 days

Waiting time for LastTouchIndex submission =: X days == 60 days,

Lookback in LastTouchIndex =: 2^L == 182 days

Choosing these numbers, a client only needs to be online once every second month for full security and the operator has do withdrawals only once a month(gas costs for publishing the LastTouchList == 68 *L/8* 25K = 4 mio gas for 25k accounts)

(Note, owners of new deposits, which have a data-unavailability for the tip of the chain, would not prove their balance in a previous block, but that they actually did the deposit during h and h-(L-bit number from the LastAccountTouch))

Concrete examples:

Imagine Bob wants to send Alice some coins in this plasma chain. The workflow would be the following: Bob sends the transaction sending funds from his account to Alice to the operator. Probably, Bob would pay a tiny fee to the operator for considering the transaction via a payment channel as well. He needs to stay online for 15 more seconds as the operator will send all transactions back to Bob. Bob will hash these transactions and sends back the signed hash of the transactions.

Now, the operator will include Bob's transaction into the next snark proof, which will update the global state of the plasma chain StateHash

. This new StateHash

will have the coins subtracted from Bob's account added to Alice account.

Alice does not need to online to receive this payment. But she needs to have the data of the latest block for a withdraw if the operator will be unavailable. Hence, she would only accept the payment, if someone provided the data to her. Here, usually, she would get the data from the operator, but she could also get the data from the payment sender, as he knows the latest state for sure.

If Alice wants to withdraw the coins, she would just send a transaction to the plasma contract and the operator would do the withdrawal for her.

Only if the operator is not available and the chain gets frozen, she would have to calculate a snark proof for her balance in one of the eligible

blocks. Here, a block is eligible if it is in the range indicated by the LastAccountTouch

published on the root-chain.

Summary:

Many current snark sidechain solutions are publishing all transaction data on-chain to ensure data availability. With this solution, we reduce the on-chain publishing to only 1 bit per transaction: the bit per transaction in the DoubleSignBitmap. This reduces costs significantly, however it requires clients to double sign messages and it increases the operation costs slightly because of the publishing of the LastTouchIndex list. I posted this write-up just for usual payment processing. However, this method can be used for all kind of applications. It fits especially well to all kind of multiplayer and auction mechanisms.