Authors: Carl Park, Aiden Park, Kevin Jeong

Plasma EVM updated to new version: link

And please forget about the previous versioning. Further updates will be made in this repository.

This update includes publishing as PDF and rewriting description and diagram. Notable changes and rationales are briefly as follows:

(Skip this if you didn't read previous post)

- Get rid of the term "user-activated fork" to avoid confusion. It is renamed to "rebase".

Actually, fork implies "fork and rebase". Many people wonder why the RootChain contract have to support it because forks can generally occur whenever some miners want them or have network latency in PoW. However "user-activated fork" or "rebase" does not happen simultaneously in Plasma EVM.

- "rebase" occurs periodically, not initiated by user with paying cost.

Previous construction charged high costs to users who cause rebase with some parameters. Rebase was considered as exceptional situation. This naive and financial approach has two drawbacks; 1) The cost model cannot define clearly which parameters should be. # of users who want to exit, total value of the exits, deposits from operator to launch the plasma chain, even what curve the cost function has. 2) Anyone can revert their transactions in plasma chain only if they pay the high cost.

Continuous Rebase, which enforces rebase periodically as the default behavior, must be processed by the operator. So we need halting condition (like slashing condition in plasma snapp) to deal with the case when the operator does not act as expected, such as decline to process the rebase, because RootChain contract initiates rebase. (It was users who initiate rebase in previous construction, so halting condition was not required.)

This approach can eliminate the first problem and make the second one open to all users about how many money they have. Anyone can cancel his transaction only if he can pay fee in Ethereum. But we can say that it is a matter of finality.

Request

Users can enter or exit any state of requestable contract. If request is created in root chain, it is applied in child chain in the form of requeest transaction whose sender is 0x00

, nonce is 0

, and data field is based on the request parameters like this. 2 requestable contracts deployed in both chain are interchangeable in accordance with the rule defined in contract. (RequestableSimpleToken)

3 Types of Block and Epoch

- Non-Request Block (NRB): a regular block that users send transaction. It is same as block of Ethereum, Bitcoin, and other blockchain.
- Request Block (RB) is a block to apply request in child chain. Users create request and operator mines request block to apply the request in child chain. It is enforced what transactions must be included in request block by RootChain contract calculating transactions root.
- Escape Block (EB) is kind of request block to provide user to deal with block withholding attack by operator.

Epoch is a period of blocks. There are Non-Request Epoch (NRE), Request Epoch (RE) and Escape Epoch (EE) for each block type. Operator must mine and commit blocks of same type as epoch. txRoot

of request block and escape block is enforced by RootChain contract, and if non-request block contains request transaction, operator is peneralized.

The length of epoch is the number of blocks in the epoch. NRE has a constant length, but the length of RE and EE depends on the number of requests.

2 Step Commit and Rebase

A block is committed twice in root chain to provide a way out for users who notice block withholding attack.

- In Pre-commit stage, operator mines and pre-commits blocks only with its txRoot

. Users can notice whether data for the txRoot

is available or not.

- In Pre-commit stage or DA check stage, users can make escape requests depending on data availability. DA check stage gives addtional time to check for the last pre-commited block.
- In Commit stage, operator mines escape block and place them before pre-committed blocks. (it is like git's rebase, applying commits to other branch. there can be conflict. in this case, it is transaction canceling). Escape epoch is located after previous cycle's committed block (in Commit stage). Operator commits this rearranged blocks' stateRoot

and receiptsRoot

to root chain.

CommitEpochs[0] = rootchian.EscapeEpochs(cycleNumber) CommitEpochs[i] = PreCommitEpochs[i - 1]

The length of pre-commit and commit stages are the number of epochs in stages, so length of commit stage is length of pre-commit stage plus 1.

- After Commit stage, challenge period for committed blocks starts in Challenge stage. Here users can verify blocks. Because, if he couldn't, he would have escaped the child chain in DA check stage.

If operator failed to pre-commit or commit for specific time(e.g., 1 day), halting conditions are fulfilled and child chain is in shutdown to prevent futher NRB or RB from being (pre-)committed. In this case, only DA check and commit stage are repeated , and anyone can commit escape block.

Cycle

Those 4 stages are included in a single cycle. If there is no successful computation challenge in challenge stage, a cycle and commited blocks are finalized. Plasma EVM client can seamlessly run next pre-commit stage only if current pre-commit stage is completed, regardless of whether commit stage is completed or whether current cycle is finalized.

[

1642×828

](https://i.imgur.com/zZaDUsM.png)

Block difficulty is computed as 1 bit(1 if rebased, otherwise 0) + 127 bits(cycle number) + 128 bits(block number)

to give priority to commit stage over cycle number.

[

1331×316

](https://i.imgur.com/T0ttosz.png)

Computation Challenge

If operator commits block with invalid computation, user can challenge with Truebit-like verification game. This game includes interaction between challenger and operator in two layers.

The first one is finding what transaction is invalid. Challenger queries with txIndex

and operator responds with post transactional state after [0, ..., txIndex]

transactions are computed and numSteps

. If txIndex == len(txs)

, check block sealing (e.g, block reward). If txIndex > len(txs)

, operator responds with [0x00, 0]

to challenge on transactions out of range

. In this case, challenger have to respond with merkle proof of txIndex

transaction.

Next is finding which step of opcodes is invalid. Challenger queries with step

of EVM computation of the transaction and operator responds with state after [0, ..., step]

opcodes are computed. If step == numSteps

, check transaction fee deduction.

This challenge is possible only if data availability is guaranteed. But in the case of block withholding attack, user would have escaped the plasma chain in DA check stage (or Pre-commit stage).

Limitations

Current fork and rebase approach cannot resolve transaction canceling issue. It can be mitigated if all period of cycle is meaningfully short, but period of DA check stage should be enough for all users to be able to determine data availability.