Here is Plasma Prime design proposal from BANKEX Foundation team. At the moment it's at the draft stage

. Waiting for discussions & criticism. Thank you.

## Abstract

Below we are going to describe Plasma design based on plasma cashflow improved by RSA accumulators and range chunking. Range chunking simplifies block verification for block observer by compressing history.

\DeclareMathOperator{\included}{included} \DeclareMathOperator{\parent}{parent} \DeclareMathOperator{\child}{child} \DeclareMathOperator{\inclusionproof}{inclusionproof} \DeclareMathOperator{\exclusionproof}{exclusionproof} \DeclareMathOperator{\True}{True} \DeclareMathOperator{\False}{False}

## Segments

Similar to plasma cashflow this proposal is based on the UTXO model where each unspent output defines ownership of specific segment. Length of the segment is equal to a value of the asset that was deposit to plasma.

Each segment is mapped to a prime number generated by the prime hash function, usage of that prime number will be shown later.

Due to the limitation of prime numbers that we generate with reasonable efforts plasma has a finite set of segments that can belong to different owners during the plasma lifetime.

We can say that we may have up to 2^30 different segments in the demo implementation.

## Segments chunking

Suppose there exists a segment X

. During the timeA

some transaction that affects X

was published, e.g. split or merge transactions.

During the time B

transactions that affect segments different from A

was published.

At the time moment C

transaction that affects X

, was published, at that moment we want to check the history of that transaction is valid.

To do that we need to generate proof that covers only transactions related only to range X, and not the whole transaction history of plasma.

**Example**

Lets split plasma segments space into the four chunks.

The image below illustrates scenario when two sequential deposits transactions (txA, txB) creates two segments:

- txA creates the segment (0 - 1.75) that matches to chunk (1 - 2)
- txB creates the segment (1.75 - 5) that matches to chunks (3 - 4) and (5 - 6)

Note: Deposit transactions are coming from the smart contract, each one located in a separate block, and doesn't have reference to any previous block

Suppose after N blocks we got following layout of segments inside plasma:

To verify that C

exit is valid, block observer only needs to check the history of the segment that matches to chunks (3 - 4) and (5 - 6). Proof of inclusion of range C

to mentioned chunks will include transactions that affect the same two chunks.

## RSA proof of segment inclusion and non-inclusion

The similar prooving schema was proposed by@vbuterin here. We proposed schema with shorter proofs of inclusion (because we need to proof only inclusion of several numbers) and same proof of non-inclusion.

Let's define boolean function \included

for each aligned slice P_i

and introduce following auxiliary variables \alpha

and \beta

, mapping each aligned slice to two unique prime numbers:

$$\alpha(P) = \included(P) \bigwedge_{Q \in \parent(P)}! \included(Q)$$

$$\beta(P) = \exists Q \in \child(P) \cup P: \alpha(Q)$$

Then determine following functions:

$$\inclusionproof(P) = \exists Q \in \parent(P) \cup P: \alpha(Q) \bigwedge_{R\in\parent(Q) \cup Q} \beta(R)$$

$$\exclusionproof(P) = !\beta(P) \bigwedge_{Q\in\parent(P) \cup P} !\alpha(Q)$$

It is obviously to check, that

$$\inclusionproof(P) \Rightarrow \inclusionproof(Q), \forall Q \in \child(P).$$

$$\exclusionproof(P) \Rightarrow !\inclusionproof(Q), \forall Q \in \child(P) \cup P.$$

$$\inclusionproof(P) \Rightarrow !\exclusionproof(Q), \forall Q \in \child(P) \cup P.$$

In plasma we use the following properties to get only necessary parts of plasma history.

## Block structure

We propose the following block structure:

Notes: TX Netto is actually transaction content without signatures. A transaction can include one or two signatures, two - in case of atomic swap and one in all case of a split and merge.

struct Block { BlockHeader header, Transactions[] transactions }

struct BlockHeader { SumMerkleRoot sumMerkleRoot, uint2048, RSAAccumulator, RSAInclusionProof RSAChainProof }

struct RSAInclusionProof { uint2048 b, uint256 r }

struct Transaction { TransactionContent content, Signatures[] signatures }

struct TransactionContent { Input[] inputs, Output[] outputs, uint64 maxBlockIndex }

struct Input { uint160 owner, uint64 blockIndex, uint32 txIndex, uint8 outputIndex, Segment amount }

struct Output { uint160 owner, Segment amount }

struct Segment { uint256 begin uint256 end }

One transaction may have multiple txIndex

values because it can be fragmented to different segments.

**SumMerkleRoot**

We use SumMerkleTree to quickly verify that there's no overlap between different transaction segments in the block and efficient generation of inclusion proof.

We build the tree in the following way:

node.length = left.length + right.lengh node.hash = Hash(left.length, left.hash, right.lengh, right.hash)

The leaf of this tree corresponds to the transaction hash of the transaction or null hash.

Length of the leaf is the length of the corresponding segment in the transaction.

Note: Each transaction in the block can produce more than one leaf of this tree in case transaction contains more than one segment inside.

Similar structure is proposed here. Our approach differs, because we store only one tx hash or zero hash inside each leaf.

## Deposits, withdrawals, and fragmentation

Segment based model may have a problem of excessive fragmentation since withdrawing leave a hole between segments. To solve that plasma operator we can:

- Make deposits to the holes
- Take extra fee for the withdrawal that makes significant fragmentation

Note: Exit possible with part of UTXO (one of the segments)

## Exit game

### Force inclusion of the transaction

This idea is proposed in Plasma Cashflow spec. A transaction may be force included into a block by following game:

1. Somebody can present the unsigned transaction (corresponding TransactionContent

structure) and the bond

1. Anybody can commit signatures (with auto refund the gas)
2. Anybody can challenge the transaction by presenting spend of inputs (including withdrawals) or non-inclusion proof for inputs
3. If the transaction is challenged, the challenger takes the bond. Else if all signatures collected, the transaction is included into a special block with the same exit priority as the oldest input of the transaction. In both cases (signatures collected or not) the transaction is considered to be removed from all other blocks

### Standard exit game

1. Somebody starts the exit process from segment [x_1, x_2]

and attach his output to this process and set the bond.

1. (1) can be challenged by non-inclusion proof or spend (including withdrawals) immediately and get the bond
2. Anybody can start a special challenge request.
3. If the game is challenged by challenge requests, the bond is split to all

### Challange request

1. Anybody present output \in [x_1, x_2]

older than the output of tx exiting

1. Anybody can challenge him presenting the spend. The spend must not be newer than output exiting in standard exit game.
2. Anybody can replace the challenge presented an older challenge
3. Challenger (1) can continue challenge game selecting utxo \in [x_1, x_2]

from tx in remained challenge

1. Same as (2) or (3)
2. The first challenger in (2-3) gets the gas refund. The second challenger in (5) get the bond. Or challenge request is considered to be completed.

[

]
(https://gist.githubusercontent.com/snjax/66147fced3bca090884e6e7c048bc679/raw/fdcf7355aa9b1eca8aacafbf7433e7360af67cd7/exit_game(2).svg?sanitize=true)See the schema with better resolution.

### Merkle proof structure

We use two kinds of sum Merkle proof:

- Standard sum Merkle proof to proof mapping from segment to transaction
- Full sum Merkle proof to prove inclusion of all segments corresponding the transaction

We use standard Merkle proof inside the transaction:

1 bit for tx netto proof (must be zero for outputs, inputs, max_blockid and must be one for signatures) 5 bits for Merkle proof for outputs, inputs, and maxBlockId 3 bits for Merkle proof for signatures.

### Exit game methods

function withdrawal(Input point, RSAInclusionProof proof) external payable returns (bool);

function withdrawalChallangeSpend(Input point,
Transaction tx, FullSumMerkleProof txProof, uint8 spendIndex, RSAInclusionProof spendInclusionProof) external returns (bool);

function withdrawalChallangeBlock( Input point, SumMerkleProof txProof,MerkleProof inputProof, uint64 maxBlockIndex, MerkleProof maxBlockIndexProof ) external returns (bool);

## RSA proof of chain validity

Plasma owner can potentially fork RSA accumulator chain. Then the spend may be included into any subsegment of blocks, but not included into the segment to which the subsegment belongs.

Let's put into each block header special prime number and the proof that this number is included into [A_{n-1}, A_{n}]

.

If the operator forks the chain, he does not present the proof, because we have no division and modulo operations in exponential rate in RSA.

# Bibliography


Plasma call #16

Plasma call #17

Benedikt Bünz, Scaling Bitcoin 2018 “Kaizen” Day 1 Part 3

@karl, Plasma cash spec

Plasma cashflow spec

@vbuterin, RSA Accumulators for Plasma Cash history reduction

@vbuterin, Log(coins)-sized proofs of inclusion and exclusion for RSA accumulators

@kfichter, More Viable Plasma

Benjamin Wesolowski, Efficient verifiable delay functions