Matic Plasma follows a model similar to Plasma MoreVP, but is an account-based implementation compared to other UTXO-based implementations. The sidechain is EVM-compatible. Using the MoreVP construction, we also eliminate the need for confirmation signatures.

PoS layer and Checkpoints

The Matic Network uses a dual strategy of Proof of Stake at the checkpointing layer and Block Producers at the block producer layer to achieve faster blocktimes and achieves finality on the main chain using the checkpoints and fraud proofs.

On Matic Network's checkpointing layer, for every few blocks on the block layer of the Matic Network, a (sufficiently bonded) validator will create a checkpoint on the main chain after validating all the blocks on the block layer and creating the Merkle tree of the block hashes since the last checkpoint.

Apart from providing finality on the mainchain, checkpoints play a role in withdrawals as they contain the proof-of-burn (withdrawal) of tokens in the event of user withdrawal. It enables the users to prove their remaining tokens on root contract using Patricia Merkle proof and header block proof. Note that to prove remaining tokens, the header block must be committed to the Root Chain through PoS (Stakeholders). The withdrawal process will incur Ethereum gas fees as usual. We leverage the checkpoints heavily for the exit games.

UTXO-like event logs

For ERC20/ERC721 transfers, this is achieved by using a UTXO-like event log data structure. Below is a LogTransfer

event for reference.

event LogTransfer(address indexed token, address indexed from, address indexed to, uint256 amountOrTokenId, uint256 input1, // previous account balance of the sender uint256 input2, // previous account balance of the receiver uint256 output1, // new account balance of the sender uint256 output2 // new account balance of the receiver);

So, basically every ERC20/ERC721 transfer emits this event and the previous balances of the sender and receiver (input1

and input2

) become the input (UTXO like) to the tx and the new balances become the outputs (output1

and output2

). The transfers are tracked by way of collating all the related LogTransfer

events.

Exit Games

Since the blocks are produced by a single block producer (or very few), it exposes a surface for fraud. We'll briefly discuss the attack scenarios and then talk about how the plasma guarantees safeguard a user.

Attack Vectors

A. Malicious operator

The following discusses the scenarios where operator could become malicious and try to cheat.

1. Out-of-nowhere tokens / double spends / malformed receipts that fradulently increases (for an operator controlled account) / decreases (for a user) the token balance.

2. Data unavailability

After a user sends a tx, let's say the operator included the tx in the plasma block but made the chain data unavailable to the user. In that case, if a user starts an exit from an older tx, then they could be challenged on-chain by showcasing their most recent tx. It becomes easy to grief the user.

1. Bad checkpoint

In the worst case, an operator could perform A.1 and(or) A.2 and collude with the validators to commit those invalid state transitions to the root chain.

1. Halting the side chain

The operator stops producing blocks and the chain comes to a halt. If a checkpoint has not been submitted for a specified duration, it would be possible to mark the side chain as halted on the root chain. After that no more checkpoints can be submitted.

For reasons listed above or otherwise, if the plasma chain has become rogue, the user's need to start mass exiting and we aspire to provide exit constructions on the root chain that the users can leverage, if and when the time comes.

B. Malicious user

1. User starts exit from a committed tx but continues to spend tokens on the side chain. Similar to double spending but across 2 chains.

We are building upon the ideas ofMoreVp.

In a nutshell, MoreVP introduces a new way to calculate exit priority, called the "youngest-input" priority. Instead of ordering exits by the age of the output, moreVP orders exits by the age of the youngest input. This has the effect that exits of outputs, even if they're included in withheld blocks after "out of nowhere" transactions, will be correctly processed as long as they only stem from valid inputs. We define getAge

which assigns an age to an included tx. This is as defined inminimum viable plasma.

function getAge(receipt) { const { headerNumber, plasmaBlockNum, txindex, oindex } = receipt return f(headerNumber, plasmaBlockNum, txindex, oindex) // multiplied with their respective weights }

Introducing some terminology before we continue discussing the exit scenarios.

Terminology

- Withdrawer

: A user who wants to the exit the plasma chain.

- Committed tx

: A tx that has been included in a matic chain block and has been checkpointed on the root chain.

- Spend tx

: A tx that changes the user's token balance in response to an action signed by the user (does not include incoming token transfers). This maybe a user initiated transfer, burn tx etc

- Reference tx

: Txs just preceding the exit tx for that particular user and token. As defined in our account balance based UTXO scheme, the outputs to the reference tx become the inputs to the tx being exited from.

- MoreVP exit priority

: Age of the youngest input (among the reference txs) to a particular tx. It'll most often be used for calculating the exit priority.

Exit Scenarios

A. Burn tokens

To exit the sidechain, a user would launch a withdraw aka burn tokens

tx on the plasma chain. This tx will emit a Withdraw

event.

event Withdraw(address indexed token, address indexed from, uint256 amountOrTokenId, uint256 input1, uint256 output1);

Here input1

denotes the user's previous balance for the token in question and output1

denotes the number of tokens left on the side chain. This construction is coherent with our account based UTXO

scheme. A user will present the receipt of this withdraw tx to withdraw the tokens on the main chain. While referencing this receipt, the user also has to provide the following:

1. Merkle proof of the inclusion of a receipt in a side chain block (receiptsRoot

)

1. Merkle proof of the inclusion of a transaction in a side chain block (transactionsRoot

)

1. Proof of the inclusion of the side chain block header in the checkpoint on the root chain

startExit(withdrawTx, proofOfInclusion /*of the withdrawTx in the checkpoint*/) { Verify inclusion of withdrawTx in checkpoint using proofOfInclusion Verify msg.sender == ecrecover(withdrawTx)

uint age = getAge(withdrawTx) // add exit to priority Q PlasmaExit exit = ({owner, age, amount, token}) addExitToQueue(exit) }

Whenever a user wishes to exit the plasma chain, they (or abstracted out by their client app i.e. wallet) should burn the tokens on the side chain, wait for it to get checkpointed and then start an exit from the checkpointed withdraw tx.

Challenge period

1. The user burnt all their tokens on the side chain and did not make any more txs, then there is nothing to challenge with and the exit will see it's due course.

2. User could choose to burn their balance partially, or could have received tokens that they can potentially spend on the side chain. The fact that the user is not spending any more balance than left on the side chain is guaranteed by the EVM, as long as the operator is not maliciuos.

3. If the operator himself creates malformed txs that attempt to spend more tokens than available, the only option is to start mass exiting and exits with MoreVP construction will save the users. See the next 2 scenarios for more details.

B. Exit from the last ERC20/721 transfers (MoreVP)

Consider the scenario, user made a ERC20 transfer on the side chain. The operator added a out-of-nowhere tx just before the user's transfer and colluded with the validators to checkpoint this block. In this scenario and more generally, in the attack vectors A1 through A3 discussed above, the user may not have had the opportunity to burn their tokens before a malicious tx is included and hence would need to start an exit from the last checkpointed tx on the root chain - for this reason, in addition to the burn exit, we need to support exits from a variety of txs like ERC20/721 transfers among others. Building upon this attack vector and breaking down the 2 scenarios:

Outgoing transfer

I transferred some tokens to a user, however I noticed that the operator included a malicious tx in the block/checkpoint before including my transfer tx. I need to start exiting the chain. I'll start an exit from the transfer tx. As defined in MoreVP, I'll need to provide a reference tx (input UTXO

) that'll define the exit priority of the exit. So, I'll reference a tx that updated my token balance and just precedes the outgoing transfer tx.

startExit(referenceTx, proofOfInclusion, exitTx) { Verify inclusion of referenceTx in checkpoint using proofOfInclusion Verify token balance for the user after the input tx was executed >= tokens being transferred in the exitTx Verify msg.sender == ecrecover(exitTx)

uint age = getAge(referenceTx) // add exit to priority Q PlasmaExit exit = ({owner, age, amount, token}) addExitToQueue(exit) }

Incoming transfer

I noticed that the operator included a malicious tx in the block/checkpoint before including my incoming transfer tx.

I'll start an exit from the incoming transfer tx while referencing the counterparty's balance - because here the input UTXO

is the counterparty's token balance.

startExit(referenceTx, proofOfInclusion, exitTx) { Verify inclusion of referenceTx in checkpoint using proofOfInclusion Verify token balance for the counterparty after the input tx was executed >= tokens being transferred in the exitTx Verify input.sender == ecrecover(exitTx) && input.receiver == msg.sender

uint age = getAge(referenceTx) // add exit to priority Q PlasmaExit exit = ({owner, age, amount, token}) addExitToQueue(exit) }

Challenge period

If a user started an exit from a particular state but continued to spend tokens on the side chain, they will be challenged. To challenge, a challenger will provide any tx that the user made that appears chronologically after the reference tx.

C. Exit from an in-flight transaction (MoreVP)

This scenario is to combat data unavailability scenario. Let’s say I made a tx but I do not know whether that tx has been included due to data unavailability. I can start an exit from this in-flight tx by referencing the last checkpointed tx. The user should be careful not to make any txs whenever they start a MoreVP style exit, otherwise they will be challenged.

Notes

When exiting from a moreVP style construction, a user can start an exit by providing reference txs, exit tx and placing a small exit bond

. For any exit, if the exit is successfully challenged, the exit will be cancelled and exit bond will be seized.

Limitations

1. Large proof size: Merkle proof of the inclusion of the transaction and merkle proof of the inclusion of block (that contains that transaction) in the checkpoint.

2. Mass exit: If the operator turns malicious, the users need to start mass exiting.

The spec is in a nascent stage and we would appreciate any feedback that helps us improve it or redesign altogether if this construction is hopelessly broken. The implementation is work in progress in our contracts repository.