While some MVP implementations are approaching production, more teams are exploring the realm of general computation on Plasma chains.

Here are some assumptions/expectations about smart contract running on Plasma that I have been hearing:

- Account-like structure with code and persistent state.
- Ability to have custody of funds.
- Exit games under subjective data-availability assumptions.

In this post I want to describe my thoughts on why smart contracts are infeasible on Plasma under those assumptions and give an example.

TL;DR

Once data-withholding starts, all contract states and balances have to be considered corrupted, as state transitions during data withholding can not be verified. Subjective data-availability prevents the construction of an efficient exit game.

Example

We assume a very basic multi-signature wallet contract on the Plasma chain. Alice and Eve have a key in the wallet, the wallet balance is 1 PETH (Plasma ETH), and the contract has only 2 bits of state, signed-by-alice

and signed-by-eve

. To flip one of the bits a transaction needs to be send to the wallet with a valid signature by Alice or Eve. Once both bits are set to true

, 1 PETH is transferred to Eve.

In the scenario of the operator becoming byzantine, we want to be able to construct an exit game for the given contract that allows to exit the state (2 bits) and contract funds (1 PETH) into an identical contract on the main network. Alice and Eve should be able to continue using the multi-sig wallet on Ethereum.

Youngest State Exits

A naive exit game with the following rules is constructed:

- Period 1
- Find the youngest included contract transaction and concluding state. Allow to append further non-included transactions that spend directly from that state, but require a bond for each.
- Period 2
- Allow to challenge any appended transactions using a Truebit-like computation verification game. Successful challenger receives bonds of all following invalidated transactions.

After the second period the latest unchallenged state of the contract is exited.

The table below shows the state of the Plasma chain at different times (t

):

t - 1

- 1 PETH in contract, no signatures to spend from contract.

t + 0

- This block is withheld from Alice by operator.

t + 1

- The new contract state is Alice having approved the withdrawal already. (Yet, she never sent her signature, the operator simply flipped the bit in the unavailable block.)

t + 2

- Eve signs with her key and receives 1 PETH.

t - 1

t + 0

t + 1

t + 2

Alice

b: 0

~

b: 0

b: 0

Eve

b: 0

~

b: 0

b: 1

Contract

b: 1, s: [0, 0]

~

b: 1, s: [1, 0]

b: 0, s: [1, 1]

Obviously Alice will initiate a Plasma exit, as she can not see the data in t + 0

, and try exit state t - 1

. Eve will be able to use youngest transaction

rule from period 1 of the exit game and replace t-1

with t+1

. Eve will then append t+2

during the exit game. Alice will not be able to challenge, as the transition is valid, and the wallet will be exited without funds, even though Alice never signed a withdrawal to Eve.

Oldest State Exits

We construct another naive exit game with rules to catch the previous case:

- Period 1
- Find the oldest

included contract transaction. Allow to append further included or non-included transactions that spend directly from that state, but require a bond for each.

- Period 2
- Same as before, filter out invalid state transitions through crypto-economic computation verification.

The table below shows the state of the Plasma chain at different times (t

):

t - 99

- Contract balance is 1 PETH, signature by Alice present.

t - x

- A lot of interactions with contract that don’t change state.

t - 2

- Contract balance is 1 PETH, signature by Alice present.

t - 1

- Eve signs and receives 1 PETH.

t + 0

- Data withholding starts.

t - 99

t - x

t - 2

t - 1

t + 0

Alice

b: 0

…

b: 0

b: 0

~

Eve

b: 0

…

b: 0

b: 1

~

Contract

b: 1, s: [1, 0]

…

b: 1, s: [1, 0]

b: 0, s: [1, 1]

~

Eve has received 1 PETH on the Plasma chain already, and exits the funds through the priority queue. In addition, Eve decides to propose an old state of the contract at t - 99

according to the exit rules for contracts defined above. Due to a bunch of spam interactions with the contract, other participants are not able to “catch up” the state, as it is too expensive to submit 90+ transactions to the main net.

=> The contract is exited with a balance of 1 ETH and a ready approval by Alice in its state. Eve claims the 1 ETH with an on-chain transaction to the multi-sig and now has 2 ETH total.

It is assumed that the extra ETH comes from an honest participant of the Plasma chain that had a UTXO younger than

Eve's. This participant will not be able to exit.

General Computation with Tradeoffs

The previous example has shown that subjective data availability paired with persistent state prevents the construction of any efficient exit game for smart contracts. How does this change if some of the assumptions made about contracts on Plasma are relaxed?

- Drop State:

Verification of computation can be done during the exit game independent of previous state. While loosing the convenience of stateful programming, Bitcoin script-like applications would still be possible.

- Drop Custody:

Contracts can be executed on Plasma, but can never hold funds or exit their state. This enables routing of funds on Plasma and chains for DEX-like applications. Interesting use-cases that require custody need to stay on main net.

- Notarize youngest available block:

Knowing the youngest available block would make the first exit game in the example feasible. This could be done with a bonded set of notaries that vote on the availability of blocks using crypto-economic aggregate signatures. (Note:

This is out of scope of Plasma, more similar in design to sharding.) The security assumptions of funds held in smart contracts would not be the same as for other UTXO's on the Plasma chain. While funds held directly would be as secure as on the main network, contract funds would only be secured up to the deposit size of the notaries.

- Write all contract as zkSNARK/STARK circuits:

This is an interesting approach that would allow to construct Plasma chains that are correct by construction. It is currently limited by circuit size, developer convenience, and the ability to nest proofs.

Questions

- Do you agree or disagree with the statement that smart contracts are infeasible on Plasma?
- Which one is you favorite tradeoff?