CryptoeconomcisLab team(@sg @yuriko) implemented another simple exit game for Plasma cashflow construction. If you have a similar implementation or idea, I hope discussion.

I think our implementation is different from @danrobinson 's proposal but similar to priority increase game(Plasma Prime design proposal) and MoreVP.

A description and vyper contract are here.

The methods for exit-game are only 3. “exit”, “challenge” and “requestHigherPriorityExit”.

- Description
- Source Code

Overview

This proposal is simple exit-game for Plasma cashflow construction supportingedge case. When a user exit with the transaction and the transaction was used as a challenge in the past, the priority of exit is Plasma block number of the parent transaction. Exiting users can request higher priority exit to the lower exit. (This is similar to invalid history challenge or priority queue) The lower priority exit is never finalized unless the higher priority exit is challenged.

Proposal

exit(segment, exitTx, proof, sigs)

- start exit procedure
- each exit has priority
- Normally, the priority of the exit is it’s block number
- If the tx used for exit was used as a challenge, priority of this exit is previous exit’s block number.
- Register this exit as childExit

of the previous exit

github.com

cryptoeconomicslab/plasma-chamber/blob/d449a1e28a823c91c6011e257dfc9475acee8df9/packages/contracts/contracts/RootChain.vy#L448

1. tokenId: uint256 = self.listed[token]
2. assert passed
3. self.processDeposit(
4. depositer,
5. tokenId,
6. amount / self.listings[tokenId].decimalOffset)
7.
8. **@dev exit**
9. @public
10. @payable
11. def exit(
12. _utxoPos: uint256,
13. _segment: uint256,
14. _txBytes: bytes[496],

```
_proof: bytes[512],
_sig: bytes[260],
_hasSig: uint256
):
assert msg.value == EXIT_BOND
exitableAt: uint256 = as_unitless_number(block.timestamp) + EXIT_PERIOD_SECONDS
blkNum: uint256 = _utxoPos / 100
```

challenge(exitId, segment, challengeTx, proof, sigs)

- Spent challenge and Double spent challenge
- If there is parentExit(in case of that exitId

is childExit of some exit), users can prove spent of parentExit(This is double spent challenge)

- If there is lowerExit

of the exitId

, lowerExit's challenge counter is decremented. * And also exit period will be extended

- And also exit period will be extended

github.com

cryptoeconomicslab/plasma-chamber/blob/d449a1e28a823c91c6011e257dfc9475acee8df9/packages/contracts/contracts/RootChain.vy#L499

```
})
if _hasSig > 0:
self.extendExits[exitId].forceInclude = _hasSig
self.removed[txHash] = True
ERC721(self.exitToken).mint(msg.sender, exitId)
log.ExitStarted(msg.sender, exitId, exitStateHash, exitableAt, _segment, blkNum, _hasSig > 0)

```

@dev challenge

@param _utxoPos is blknum and index of challenge tx

```
@public
def challenge(
_exitId: uint256,
_childExitId: uint256,
_exitStateBytes: bytes[256],
_utxoPos: uint256,
_segment: uint256,
_txBytes: bytes[496],
_proof: bytes[512],
_sig: bytes[260]
):
blkNum: uint256 = _utxoPos / 100
```

requestHigherPriorityExit(higherPriorityExitId, lowerExitId)

- This is an alternative of invalid history challenge and priority queue
- The priority of higherPriorityExit should be higher than exit
- Segments should be overlapped
- lowerExitId’s challenge counter is incremented
- lowerExit can’t be exited until higherPriorityExit is challenged

github.com

cryptoeconomicslab/plasma-chamber/blob/d449a1e28a823c91c6011e257dfc9475acee8df9/packages/contracts/contracts/RootChain.vy#L572

```
clear(self.exits[_exitId])
else:
self.exits[_childExitId].owner = ZERO_ADDRESS
clear(self.exits[_childExitId])
send(msg.sender, EXIT_BOND)
log.Challenged(_exitId)

@dev requestHigherPriorityExit
This is an alternative to invalid history challenge.
@public
def requestHigherPriorityExit(
_higherPriorityExitId: uint256,
_lowerPriorityExitId: uint256
):
higherPriorityExit: Exit = self.exits[_higherPriorityExitId]
exit: Exit = self.exits[_lowerPriorityExitId]
higherPriority: uint256 = higherPriorityExit.blkNum
lowerPriority: uint256 = exit.blkNum
if self.extendExits[_higherPriorityExitId].priority > 0:
higherPriority = self.extendExits[_higherPriorityExitId].priority
if self.extendExits[_lowerPriorityExitId].priority > 0:
```

finalizeExit

- challenge counter should be 0
- after exit period

Other important points

- Challenger should prove spent of exit state
- verify spent is application specific logic in UTXO programming
- verify spent is application specific logic in UTXO programming
- The valid transaction which has multiple inputs require confirmation signature

Attack scenarios

Exit spent segment

Precondition

1. In block 1, Alice sent segment A to Bob.
2. In block 2, Bob sent segment A to Carol
3. Bob attempts exit segment A with the transaction in block 1

How Carol save money

Anyone can challenge Bob's exit by the transaction in block 2.

Exit with double spending segment

Precondition

1. Alice sent segment A to Bob in block 1
2. Alice sent segment A to Carol in block 2(The operator allowed Alice's invalid behavior)

How Bob save money

1. Carol attempts to exit with block 2
2. Bob exit with block 1
3. Bob do requestHigherPriorityExit(his exit in block1, Carol's exit in block 2)
4. The operator can't finalize the exit in block 2
5. Bob can withdraw his money

Exit with invalid history segment

Precondition

1. Alice sent segment A to Bob in block 1
2. The operator puts an invalid state transition in block 2

How Bob save money

1. The operator attempts to exit with block 2
2. Bob attempts exit with block 1
3. Bob do requestHigherPriorityExit(his exit in block1, operator's exit in block 2)
4. The operator can't finalize the exit in block 2
5. Bob can withdraw his money

Edge case: with double spending

Precondition

1. Adam sent segment A to Alice in block 1

2. Alice sent segment A to Bob in block 2
3. Alice sent segment A to Carol in block 3(The operator allowed Alice's invalid behavior)

(Alice and Carol and operator were a malicious party)

How Bob save money

1. Alice attempts to exit with block 1
2. Carol challenge Alice's exit with block 3
3. Carol attempts to exit with block 3(The priority of his exit is “1”. This is an important note.)
4. Bob attempts exit with block 1
5. Bob do requestHigherPriorityExit(his exit in block2, Carol's exit in block 3), but he can't because the priority of Carol's exit is “1”!
6. Bob challenge Carol's exit showing double spending with block 2(block 3 is double spending!)
7. Bob can withdraw his money with block 1

Edge case: exit with invalid history and withholding

Suppose Alice owns a coin at block 1 and submits a spend to Bob, but the operator puts an invalid state transition in block 2 and then includes Alice's transaction in block 3 (without making any of the data available). Alice can use the following protocol to exit her coin (cooperatively with Bob):

In our exit game, Bob can exit his coin cooperatively with Alice. Alice already signed off on transfer transaction.

1. Alice attempts to exit with block 1
2. Within the exit period, operator challenge Alice's exit by block 3
3. Bob attempts to exit with block 3.Priority of this exit is block1 because the tx was used for the challenge.
4. Within the exit period, operator exit with block 2
5. Operator may requestHigherPriorityExit(block 2 exit, Bob's block 3 exit). But the priority of Bob's exit is block1, so this request will be failed.
6. Bob can finalizeExit of his exit with block 3

Source Code

GitHub

cryptoeconomicslab/plasma-chamber

Plasma Chamber is a DApps development framework that guarantees security, scalability, and usability utilizing Plasma technology. - cryptoeconomicslab/plasma-chamber