Reliable Exits of Withheld In-flight Transactions (“Limbo Exits”)

This post is a more long-winded version of ideas that came out of conversations with Piotr Dobaczewski,@danrobinson, @DavidKnott, and @jcp. Credit of the original concept goes to Piotr, who astutely noticed that we can settle withheld in-flight transactions on the root chain. Some modifications are made to Piotr’s original concept.

Edit

Since publishing this post, I’ve realized that the scheme only works if the outputs are cooperating. When we’re talking about a single output, that’s probably reasonable. If we have multiple unrelated outputs, that’s less reasonable. The More Viable Plasma construction doesn’t have this problem (assumes non-cooperation), but I still think limbo cancels are a very important because they make things easier when there is cooperation.

Problem Background & Motivation

Both Plasma Cash and David Knott’s no-conf version of Plasma MVP suffer from the problem of “transactions in limbo.” In a nutshell, this problem arises when a transaction is in-flight but part (or all) of the Plasma chain is unavailable. Neither party to the transaction knows where (or if) the transaction is included in a block, so the sending party must attempt an exit on their possibly spent coin.

If the transaction has already been included in a block, then the operator can submit a challenge and claim the sending party’s bond. In the Plasma Cash case, the operator could even choose to create a new block that includes the transaction after the fact specifically to claim the bond. This action gives the receiving party enough information to exit from the now-included transaction.

I’m of the opinion that this isn’t a particularly satisfying solution. There’s some sense of uncertainty about who will actually receive the exit, and the grieving factor against the sender is possibly non-negligible. This is, of course, up for debate. If users and implementers are generally OK with this attack vector, then the mechanism described in this post is unnecessary.

Limbo Exits

“Limbo Exits” (as coined by Piotr possibly? not sure) are a special type of exit that attempts to solve the above problem. The general idea of the construction is that the sender will allow the receiver to “force” the transaction to complete on the root chain. This effectively becomes a confirmation signature that’s only necessary in the very specific case of withheld in-flight transactions. It’s possible to extend this construction so that the receiver may force the transaction not

to complete, but the first use case seems generally more useful.

We’ll detail how this scheme would work in the context of Plasma Cash. First, let’s set up a scenario where the limbo exit will become necessary:

1. A

broadcasts a transaction sending a coin to B

while the Plasma chain height is at N

.

1. The operator begins withholding blocks at Plasma block N + 1

.

1. Neither party knows if the transaction from A

to B

is included in block N + 1

.

Without limbo exits, A

is forced to attempt an exit first. If the operator has included the transaction from A

to B

, then the operator will challenge the exit started by A

, giving B

the required information to exit.

Limbo exits instead allow the two parties to agree to complete the transaction on the root chain and exit simultaneously. Limbo exits are started as follows:

1. A

sends B

a signature on the hash of the in-flight transaction plus some special constant. This minimizes the size of the signature and makes the signature is different than the signature A

has already provided on the transaction hash. The special constant can probably just be a single bit.

1. B

starts a limbo exit by submitting all data A

would submit if A

were exiting from the coin, as well as the signature received in (1) and the in-flight transaction. “All data A

would submit” is effectively the entire tx giving A

control of the coin, the block in which that tx was included, and an inclusion proof. Note that no Merkle proof-of-inclusion is required for the in-flight transaction because this transaction may or may not actually be included. B

places a bond on the exit as normal.

The limbo exit can then be blocked if the challenger provides either:

1. Any signed transaction from B

spending the coin that appears in a block after the one given in (2) or

1. Any signed transaction from A

spending the coin (except the one given in (2)!

) that appears in a block after the one given in (2)

These challenge conditions guarantee (I think?) that the exit can only be challenged if A

or B

genuinely double spent the coin.

Caveats

Note that this mechanism only works when A

and B

are cooperating. We assume that if the two are cooperating, they will want to complete the transaction as planned. As stated before, it’s possible to adapt this design to allow B

to “reject” the transaction by effectively running the signature scheme in reverse.

The reliance on cooperation is probably not an issue. If A

and B

are not cooperating, then the transaction may or may not complete depending on if the operator has included it or not. This situation would probably require extra-protocol resolution anyway and is similar to the case where a customer pays for a product but the merchant fails to send it.

Optimizations

- We can reduce the amount of data B

is required to submit by allowing B

to simply assert that the transaction giving A

control was included in the given block. This means we need to allow another challenge type where the challenger proves the transaction for that coin in the stated block did not give A

control or some equivalent interactive game.

Notes/Changelog/Etc.

So I think

this works. I may be entirely wrong, we may need more challenge conditions, it may not work at all. Feedback/criticism/review is always more than welcome!

Again, a special thanks to Piotr Dobaczewski, Dan Robinson, David Knott, and Joseph Poon for useful conversations about this idea.