thanks to @gakonst for review

Many non-channel L2 systems tend to suffer from the following problem. We wish to make it not costless to make claims about the current off-chain state that are incorrect (e.g., exit a spent coin). To do so, we introduce exit bonds that are in some circumstances slashed. From the point of view of making incorrect exits costly, we would like to make the slashing occur whenever the exit fails (i.e., whenever the exitor has been shown to make a false claim). The problem arises when there is a possibility of an in-flight transaction (i.e., a transaction which has been submitted to the operator, which may or may not have been included in an committed but unavailable block) with multiple beneficiaries.

Let P

= {\mathrm{Alice}, \mathrm{Bob}, \mathrm{Carol}, \mathrm{Dave}}

. Suppose that the in-flight transaction T

consumes UTXOs owned by Alice and Bob [1] and produces UTXOs owned by Carol and Dave, and T is withheld. We do not assume that any member of P

can cooperate with any other. It follows that an exit must be able to be initiated by some single member of $P4. There are two cases: we can disallow Alice and Bob from being beneficiaries by allowing Carol or Dave to exit as if T

were always in the chain (“forced include exits”), or we can allow any of them to be beneficiaries by allowing exitors to make a claim that (i.e., an exit that succeeds when) the chain either contains T

or not (“limbo exit”). In either case, an individual exitor must make a claim that can could be false by no fault of their own.

There are a few existing solutions to this problem:

1. Restrict some set to be a singleton, e.g., provide only single-input-single-output transactions.
2. Accept that some claims will be proven wrong without a corresponding bond slashing. Example: NOCUST data availability challenges.
3. Accept that an individual exitor may lose an exit bond by no fault of their own. Example: [2]
4. Rely on exit order manipulation. Example: MoreVP
5. Confirmation signatures. Example: original plasma and MVP
6. Multi-stage staggered protocols, e.g. something like [3], such that every individual step has a singleton beneficiary. One can view confirmation signatures as a special case of this.

This note proposes a yet-unexplored solution similar to confirmation signatures but that is hopefully more composable. The idea is to note that confirmation signatures use the existence of the same set of signatures for two distinct purposes: the existence of the full set of confirmation signatures determines both:

1. If a given transaction is valid
2. If a given exit bond should be slashed

We can use different signatures for these two things. We give an example of adding this to a UTXO-based plasma cashflow system with limbo exits [4] [6]. We start with any design that we know is secure without exit bonds, with the restriction that all exits must be initiated by a potential beneficiary of the exit (hence a limbo exit of a transaction T

must have msg.sender

equal to an address that’s either an input or an output of T

). All exits put up an exit bond that is slashed if a signature is provided signed by msg.sender of the digest hash(X, "slashme")

where X

is the TXO owned by msg.sender

that is either an input or an output of T

.

The procedure for making a transaction T

is as follows:

1. Each owner of an input TXO I

to T

sign a message allowing their TXO to be spent to T

and send this signature to the owners of the output TXOs of T

, as well as the operator. For concreteness we can say that the signature is of the digest hash(I, T, "spend")

, and call this a “type V signature”.

1. The operator produces and commits to a block B

with T

at each range in the input of T

.

3a. An owner of an output of T

considers the payment received when they verify that the chain up to B

is valid and available.

3b. After an owner of an input verifies that the chain up to B

is valid and available, they send the second signature of hash(I, "slashme")

to every output owner and to the operator. Call this a “type S signature”.

1. Coin history consists of both type V and type S signatures, if they exist

Note that steps 3a and 3b happen in parallel.

In the case that step 2 and 3a are completed but not step 3b for an input I

, the owner of I

can start a fraudulent exit costlessly. This is a way to grief an output owner by making their coin history contain an TXO which is spent but type-S-unsigned. However, a participant is indifferent to completing step 3b (they have no in-protocol incentive one way or the other), and any small incentive that pushes in the direction of 3b (e.g. the default client software could always produce a type S signature when safe to do so) makes the expected state of the chain that all spent TXOs are eventually type-S-signed. [5]

In the context of offline defragmentation (a transaction T

that leaves total ownership unchanged but tends to arrange TXOs with the same owner to be adjacent, and where we do not require all inputs to T to be online at the same time), a client logging out can complete step 2 for T

and then, much later, complete step 3b.

Notes / references

[1] In an account-based system: supposed it reduces the balances of Alice and Bob

[2] Plasma Cash Minimal Atomic Swap

[3] https://en.bitcoin.it/wiki/Atomic_swap

[4] Since forced-include exits are “better” than limbo exits, one could see this as evidence that this design is more composable.

[5] Here are some other ideas. In the case when a participant sends part but not all of their funds to someone else, the operator can create an incentive by censoring the participant’s other funds in the chain until that participant publishes a type S signature. Alternatively, when a participant joins the plasma chain, the operator could open a channel-on-plasma with the participant that is used to both pay fees to the operator and to trustlessly buy type S signatures. Finally, if we are desperate, we can argue that users are all ETH holders and the incentive is that they wish to see layer 2 systems succeed.

[6] If we layer this on top of a scheme where the last step in making a transaction valid is a “revocation” or “deprecation” signature (e.g. as supported in https://medium.com/plasma-group/plapps-and-predicates-understanding-the-generalized-plasma-architecture-fc171b25741), we have two distinct concepts of revocation: one of the right to successfully withdraw a

certain state, and one of the right to fail to withdraw a certain state without being slashed.