Design out of conversation with vi, David Knott, Ben Jones, and Eva Beylin.

Note

We're making heavy modification to this document. This is out of date and broken.

TL;DR

This post introduces an alternative to confirmation signatures in MVP that effectively allows any user to set the exit priority of a transaction (position in the priority queue when exiting).

Background

Plasma MVP relies on the concept of confirmation signatures to ensure that both parties to a transaction have data availability before it's considered complete. However, it appears that all proposed versions of the confirmation signature scheme are vulnerable to griefing and introduce annoying user experience artifacts. Generally, confirmation signatures operate on the assumption that all transactions are invalid until they are explicitly valid.

This has motivated the construction of alternative mechanisms which provide the same guarantees. We introduce a design whereby any party may attempt to set the exit priority of a transaction. This means that the transaction would now be processed earlier in the exit priority queue. This attempt is challengeable if the new priority would be greater than the priority of any of the transaction's inputs, if any of the inputs have been double-spent, or if the transaction's new priority would be lower than it's current priority.

The effect of this design is that an in-flight transaction that may or may not be included in the Plasma chain (a transaction "in limbo") can now be guaranteed to be included. Furthermore, if all of a transactions inputs were created at a time when the chain was valid, then the transaction will be successfully processed before any invalid transactions are processed.

Construction

A user may attempt to set the priority of a transaction by submitting a signed transaction and a block number to the root chain, subject to a challenge period. This action is bonded, and the priority of deposit ("out of nowhere") transactions cannot be modified. If no successful challenge occurs during the challenge period, then the transaction will have a priority as if it were the added to the end of the given block.

A priority modification can be challenged in four ways (higher priority = would be exited sooner, lower priority = would be exited later):

1. Someone shows that the specified transaction was included in a block, and that its original priority is higher than the new priority. This prevents someone from decreasing a transaction's priority.
2. Someone shows that any of the transaction's inputs would have a lower priority than the transaction. This prevents someone from moving a transaction "earlier" than any of the transaction's outputs.
3. Someone requests that someone prove some input to the transaction has a higher priority than new priority. This is necessary if blocks are being withheld and a challenge of type (2) is not directly possible.
4. Someone shows that there exists another spend from any of the transaction's inputs. This spend must not be the same as the original transaction.

A challenge of type (1) or (2) blocks the modification and slashes the submitter's bond.

A challenge of type (3) or (4) must be bonded, and are subject to a response.

Type (3) Response

Anyone may respond to a type (3) challenge by revealing that the requested input has been included in the chain such that it has a higher priority than the new priority. A valid response would slash the challenger's bond. A failure to respond would slash the original bond and block the modification.

Type (4) Response

Anyone may respond to a type (4) challenge by revealing another included transaction that spends from the original transaction's inputs. This revealed transaction may or may not be the same as the original transaction. A successful response slashes the challenger's bond. A failure to respond slashes the original bond, blocks the modification, and burns any inputs that have now been provably double spent. This is accomplished by adding the input to the priority queue and burning it when processing instead of sending it to its owner.

We burn the double-spent inputs to directly punish a user for creating the second transaction that's used to challenge the modification. This is necessary because the user requesting modification may not be either of the inputs, and may not know that there exists a double spend if blocks are being withheld. We want to disincentivize users from being punished (losing a bond) if they're acting honestly. We can't prevent this type of griefing, but we can make it very expensive to do so.

A failure to respond also entirely invalidates the original transaction, meaning it can't be spent, exited, or used to challenge any exit from the inputs. This ensures that at the very least, any users who have not double spent can exit without being challenged.

Exit Piggybacking

At any point in time, any of the owners of the outputs to the submitted transaction may "piggyback" onto the modification. For each output, the would-be owner can place a bond, just as if they were exiting from the outputs. If the modification is successful, then these owners will automatically have exits added to the priority queue, at the new priority. If the modification is unsuccessful, then these owners have their bonds refunded (they're not at fault).

This option is necessary because of the exit time restriction in MVP (everything must happen within 7 days of things going bad). If someone is using this modification functionality when the chain is non-byzantine, then this piggybacking won't be necessary.

Data requirements

Each priority modification requires the user publish the full transaction to the root chain, although we only need to store a 32 byte (1 EVM word) hash, so the gas cost isn't ridiculous (~100k gas at a max). I think we can reasonably process a few hundred thousand of these actions within the required 1 week period.

Notes

As always, feedback and comments are more than welcome. Please feel free to challenge any part of this, there very well may be issues.