Suppose that we have two rollups, A and B, and Alice wishes to exchange some quantity of coins on rollup A for the same coins on rollup B. There are already proposals for how to do this in a decentralized way if A and B both have full smart contract support. This document proposes how to do it in the case where only rollup B has full smart contract support (and rollup A can only process simple transactions).

We assume that transactions on rollup A have some kind of “memo field”; if they do not, we can use lower-order digits of the value sent as a memo.

Proposal

Suppose there is an exchange intermediary, Ivan (in a real implementation there would be many intermediaries to choose from). Ivan has an account IVAN_A

on rollup A (that he fully controls). Ivan also has some funds deposited in a smart contract IVAN_B

on rollup B.

The smart contract IVAN_B

has the following rules:

- If anyone sends a transaction sending TRADE_VALUE

coins to IVAN_A

, containing an address DESTINATION

as a memo, then after MIN_REDEMPTION_DELAY

blocks they can send a transaction to IVAN_B

containing a proof of the transfer, which queues a withdrawal of TRADE_VALUE

coins to address DESTINATION

.

- Withdrawals are processed after some delay (eg. 1 day) in order of the batch and index the transfers were included in on rollup A.
- When Ivan sees that he received funds at IVAN_A

, he has the ability to personally send TRADE_VALUE * (1 - fee)

coins to DESTINATION

. He can do this by sending the transaction through a method in IVAN_B

, which saves a record that prevents the automated-send clause in the contract from triggering for that trade.

The expected behavior is simple:

1. Alice sends a transaction to IVAN_A

with N coins and a memo ALICE_B

1. Ivan sends a transaction sending TRADE_VALUE * (1 - fee)

coins through IVAN_B

to ALICE_B

The second step can happen immediately after the first step. The contract can even have rules that allow the fee

to be greater if Ivan shows proof that the timestamp difference between the second transaction and the first is very low.

The “worst-case” behavior is if Ivan does not

send coins to ALICE_B

as he is expected to. In this case, Alice can wait until the transaction on rollup A confirms, find some alternate route to getting coins on rollup B to pay fees, and then simply claim the funds herself.

Capital costs

The main limitation of the scheme is that IVAN_B

needs to hold a large amount of capital to ensure that all senders will be paid. Particularly, suppose that:

- We place a trade size limit of TRADE_LIMIT

coins (so transactions going to IVAN_A

with value > TRADE_LIMIT

are not valid trades)

- Each rollup batch can contain a maximum of TXS_PER_BATCH

transactions

Alice can check herself how many unprocessed trades there are before the upcoming batch on rollup A, subtract this value from the capital she sees in the IVAN_B

contract, and check if the remaining amount is enough. Because withdrawals are processed sequentially (this is the goal of the queue mechanism above), Alice need not concern herself with the possibility of future withdrawals being processed before her own.

The maximum amount that could be traded in one batch is TRADE_LIMIT * TXS_PER_BATCH

, and so the IVAN_B

contract needs to hold at least this amount of ETH, plus enough to cover unprocessed trades. As an example, suppose TRADE_LIMIT = 0.1 ETH

(low limits are okay because a larger trade can be done with multiple transactions) and TXS_PER_BATCH = 1000

. Then, IVAN_B

would need to hold 100 ETH.

Note that there is an extra implicit fee in this design, because anyone trading more than 0.1 ETH would need to waste block space. This is traded off against capital requirements: if you halve the block waste, you double the capital requirements, and vice versa. It seems likely that the correct balance would be the point where the implicit fee is a few times smaller than the explicit fee that emerges in the market.

If we want to reduce or remove this waste, rollup A could be designed to do so, for example by having the sequencer send a signed message attesting to Alice all messages approved in the batch so far. Alice would then know that there are no trades before hers (though a malicious sequencer could, at high cost to themselves, trick Alice).

Memos

The design above assumes that transactions on rollup A have a memo field that Alice can use to specify ALICE_B

as her destination. If rollups do not

have this feature, then we can use the following workaround. Alice can register ALICE_B

on rollup B in a sequential registry contract, and get a sequentially assigned ID (so Alice's ID equals the number of users who registered before her). Let MAX_USER_COUNT

be a maximum on the user count; if necessary, this value could adjust upwards over time. Alice can simply ensure that TRADE_VALUE % MAX_USER_COUNT

equals (Alice's ID), using the low-order digits of TRADE_VALUE

, which represent an inconsequential amount of value, to represent the amount she wants to trade.

B to A trades

If Alice starts with coins on rollup B and moves them to rollup A, a similar mechanism can be used, except with reversed roles:

- Alice sends coins to IVAN_B

- After some delay, she gets the right to take the coins back
- She loses that right if Ivan can prove to IVAN_B

that he sent Alice coins on rollup A