The Sequencer and Censorship Resistance

The Sequencer is a specially designated Arbitrum full node which, under normal conditions, is responsible for submitting users' transactions onto L1. In principle, a chain's Sequencer can take different forms; asArbitrum One currently stands, the Sequencer is a single, centralized entity; eventually, sequencing affordances could be given to a distributed committee of sequencers which come to consensus on ordering. However, regardless of its form, the Sequencer has a fundamental limitation that doesn't apply to any other part of the system: it must operate under its own security assumptions; i.e., it can't, in principle, derive security directly from layer 1. This brings up the question of how Arbitrum Rollup maintains its claim to censorship resistance when-and-if the Sequencer misbehaves.

Here we will describe the mechanics of how the Sequencer typically operates, and how any user can bypass the Sequencer entirely to submit any Arbitrum transaction (including one that, say, initiates an L2 to L1 message to withdraw funds) directly from layer 1. Thus mechanism thereby preserves censorship resistance even if the Sequencer is being completely unresponsive or even malicious.

The Core Inbox

When we talk about "submitting a transaction into an Arbitrum chain," we're talking about getting it included into the chain's core Inbox, represented by thesequencerInboxAccs byte array inBridge . Once transactions are included in the core Inbox, their ordering is fixed, execution is fully deterministic, and we can trustlessly treat the resultant state as having L1-level finality (see "Transaction Lifecycle"). The Sequencer's role (or lack thereof) concerns strictly what happens prior; i.e., how a transaction makes its way into the core Inbox. We'll break down the possible routes a transaction can take into two scenarios: a well-behaved Sequencer, and a faulty Sequencer.

Happy/Common Case: Sequencer Is Live and Well-behaved

Here, we start by assuming that the Sequencer is fully operational, and is running with the intent of processing users' transactions in as safe and timely a manner as possible. The Sequencer can receive a user's transaction two ways — either directly via an RPC request, or via the underlying L1.

If a user is posting a "standard" Arbitrum transaction (i.e., interacting with an L2 native dapp), the user will submit the signed transaction directly to the Sequencer, much like how a user submits a transaction to an Ethereum node when interacting with L1. Upon receiving it, the Sequencer will execute it and nearly instantaneously deliver the user a receipt. Some short time later —usually no more than a few minutes— the Sequencer will include the user's transaction in a batch and post it on L1 by calling one of theSequencerInbox 'saddSequencerL2Batch methods. Note that only the Sequencer has the authority to call these methods; this assurance that no other party can include a message directly is, in fact, the very thing that gives the Sequencer the unique ability to provide instant, "soft-confirmation" receipts. Once posted in a batch, the transactions have L1-level finality.

Alternatively, a user can submit their L2 message to the Sequencer by posting it on the underlying L1. This path is necessary if the user wishes to perform someL1 operation along with the L2 message and to preserve atomicity between the two — the textbook example here being a token deposit via abridge (escrow on L1, mint on L2). The user does this by publishing an L1 transaction (i.e., sending a normal transaction to an L1 node) that calls one of the relevant methods on theInbox contract; i.e.,sendUnsignedTransaction . This adds a message onto what we'll call "the delayed Inbox", (represented by thedelayedInboxAccs in theBridge contract), which is effectively a queue that messages wait in before being moved over to the coreInbox . The Sequencer will emit an L2 receipt about ~10 minutes after the transaction has been included in the delayed Inbox (the reason for this delay is to minimize the risk of short term L1 reorgs which could in turn cause an L2 reorg and invalidate the Sequencer's L2 receipts.) Again, the last step is for the Sequencer to include the L2 message in a batch — when calling the batch submission methods, the Sequencer specifies how many messages in the delayed inbox to include — finalizing the transaction.

In sum — in either happy case, the user first delivers their message to the Sequencer, who in turn ensures that it arrives in the core Inbox.

Unhappy/Uncommon Case: Sequencer Isn't Doing Its Job

Now let's suppose the Sequencer, for whatever reason, is entirely failing to carry out its task of submitting messages. A user can still get their transaction included in two steps:

First, they submit their L2 message via L1 into the delayed Inbox as described above: note that although atomic cross-chain messages are the common case for using the delayed Inbox, it can in principle be used to submitany L2 message.

Once in the delayed Inbox, we obviously can't rely on the Sequencer to include the transaction in a batch. Instead, we can useSequencerInbox 'sforceInclusion method. Once a message has been in the delayed Inbox for a sufficient amount of time,forceInclusion can be called to move it from the delayed Inbox into the core Inbox, at which point it's finalized. Crucially, any account can callforceInclusion .

Currently, on Arbitrum One, this delay time between submission and force inclusion is roughly 24 hours, as specified bymaxTimeVariation.delayBlocks /maxTimeVariation.delaySeconds . A force inclusion from L1 would directly affect the state for any unconfirmed L2 transactions; keeping conservatively high delay value ensures it should only be used under extraordinary circumstances.

On top of the delay itself, theforceInclusion path has the downside of uncertainty around transaction ordering; i.e., while waiting for a message's max delay to pass, a malicious Sequencer could, in principle, directly post messages in front of it. However, there's ultimately nothing the Sequencer can do to stop it from being included in the core Inbox, at which point its ordering is finalized.

While the slow, "unhappy" path isn't optimal, and should rarely, if ever, be necessary, its availability as an option ensures Arbitrum Rollup always preserves its trustless security model, even if the permissioned parts of the system act faulty. Edit this page Last updatedonApr 29, 2024Previous Overview: The Lifecycle of an Arbitrum TransactionNext Inside AnyTrust