L2 decentralisation has been a hot topic for over a year now, but we have yet to see many major moves across rollups. Recently, Vitalik provided a useful framework for understanding the landscape of options available to rollup teams. This post builds off of that framework to provide some more detailed designs and analysis. In particular, we will be focusing on non-"based" solutions, as making any of these changes for based rollups would require changes to the base layer.

Thanks to @sarah.allen , @Hasu and Cooper Kunz for feedback and thoughts!

Blockchain architectures, in my view, need to provide at least these four security properties by order of priority:

1. Correctness of state transitions
2. this property needs little elaboration, we should have no state transitions that are not both signed off by the correct key and allowed by the state transition function.
3. Safety
4. honest nodes tracking the chain state can at some point agree on some state transition and know that it cannot be undone. This property mostly comes up in the context of avoiding double spends or reorgs. Recently, "preconfirmations"

have been a hot topic. Safety of a confirmations would mean that a node can treat a confirmation as ~final, having high confidence that it won't be reorged.

1. Liveness

(CR) - we sometimes take this property to mean that the system as a whole is still running and accessible to most users. For this article, we are also including censorship (i.e., loss of liveness for a specific user or type of operation). In particular, we have strong censorship

which means the targeted transaction(s) will never land and weak censorship

which says that targeted transactions will eventually be processed, but only after an abnormal delay.

1. User MEV protection
2. Here, I am referring specifically to the property that information about a user's transaction will not be taken advantage of and used to their detriment, ignoring other related topics like welfare maximising ordering rules and infrastructure operator incentives.

What about decentralisation?

I will invoke decentralisation in two ways.

This article works with the framing that all non-trivial architectures do provide the properties above, but these properties can only be provided given some assumptions, such as an actor following certain behaviours or not. We often use decentralisation

to refer to the strength of these assumptions. Assuming that a single operator is honest with no way of verifying this is a very strong assumption and therefore not decentralised, assuming <1/3 of stake in a large network behaves honestly is an improvement, assuming nodes are rational is better and the security of basic cryptographic primitives is probably the weakest assumption we could want to make.

We also sometimes use decentralisation to mean neutrality

, which is the meta-property that the desired properties of a system apply equally to all participants. The properties I mentioned above probably are not sufficient to call a system neutral (for example, what about preferred transaction ordering or access to data feeds?), but we at least want neutrality for those properties.

(There are other notions of decentralisation like logical decentralisation which I don't mention. In particular, I don't cover governance attacks.)

With this definition of decentralisation in mind, this post sets out to lay out some "sub-protocols" for making rollups more decentralised.

Rollups

Looking at the decentralisation of current single-sequencer rollup architectures:

Correctness of state transitions

is enforced by replicating full nodes and validating bridges, which rely only on cryptographic assumptions.

Liveness

is partly derived from the L1 which provides an avenue for transaction inclusion even if the sequencer is down or censoring. However, weak censorship/liveness failure in which the sequencer delays transactions until they get posted to L1 is still possible.

Safety

is inherited by the L1 when L1 blocks containing batches become (probabilistically at first) safe. However, this is slow so most users rely on earlier confirmations from the sequencer - i.e., slots instead of epochs in Vitalik's framing. Given that a sequencer can easily go back on such a confirmation and has incentive to do so, the assumption that they won't is quite strong.

User

MEV protection

also relies on an actor that does have incentive to misbehave not to. Since users are forwarding transactions directly to the sequencer, they are protected as long as the sequencer can be trusted. However, should this honesty assumption in a single entity fail, the user has effectively no protections while detection of misbehaviour is far from guaranteed.

Why do we need to improve over this?

Perhaps some use cases work perfectly well under the status quo, however others are very sensitive to all of these failure modes. For DeFi, for instance:

- Safety failures mean that sequencers can tell traders that one outcome has happened while this outcome is never realised on L1 (a reorg basically). For example, a sequencer could falsely report a successful trade or large price movement, causing traders to trade on this new information, revealing their hand, sending more transactions or repositioning on a different venue. The sequencer could position themselves to benefit both from the anticipated reactions from traders and the resultant information asymmetry.
- Weak censorship (i.e., forced delays) means that sequencers can delay trades during a market crash, delay margin posts to force liquidations, conversely delay liquidations to force bad debt or delay bids in an auction. This means that the financial protocols are limited in efficiency (e.g. running auctions over a longer time period or requiring higher collateralisation ratios). It also means that the market neutrality can be lost because actors favoured by the sequencer (i.e., not delayed) have a latency advantage and can thus trade with more information.
- User MEV protection relates to the above but also encompasses attacks in which the sequencer abuses their view of unexecuted transactions to frontrun trades. Again, undermining market neutrality.

Protocols

Protocols 1: TEE-hardened Single-Sequencer Architecture [Safety, MEV]

[

TEE hardened SS

1200×543 38.6 KB

](https://collective.flashbots.net/uploads/default/original/2X/a/ae917981b9a0bb1747f8ae630badd9442dc52158.png)

In Vitalik's post, this is the "server with blockchain scaffolding" route. We can address two of the identified shortcomings by requiring the sequencer to run "in" a TEE - in two importantly different ways.

- Safety:

a very simple use of TEEs can prevent a private key from being used to sign multiple blocks at the same height, preventing reorgs of "preconfirmed" blocks. Anyone should be able to post blocks (signed by the sequencer) to L1 to prevent the sequencer from signing but not posting blocks.

We can layer on additional security by allowing users to post proofs of safety faults in the form of a signed "confirmed" transaction which conflicts with what was posted to L1. These proofs can be used to slash the sequencer.

- MEV protection:

a more complex use of TEEs can keep transaction information private until committed to in a signed block and enforce a specific publicly-known algorithm for block construction. This should mean that even a malicious sequencer cannot frontrun users. Due to the complexity of the use case, this requires a much more careful implementation to avoid information leaks

through side channel attacks.

- Liveness

concerns remain largely unchanged. Weak censorship is lightly ameliorated by an architecture which keeps transactions unknown to the operator until they are committed to. Unfortunately the sequencer can still pull off attacks like delaying all transactions which come from unknown sources, prioritising whitelisted entities. The biggest improvement is that censorship should now be much more easily detectable.

The case in which the sequencer stops processing any transactions is not changed by the use of TEEs. This is covered in the next section.

All of the above relies on TEE security models - that hardware manufacturers produced hardware as they claimed, that cloud providers are not physically tampering with your server and that there is no critical security failure found in whatever TEE model(s) are being employed. While legitimate threats, if the hardware manufacturer or the cloud provider did decide to start performing attacks, we would have bigger issues on our hands.

For an attack to happen the sequencer would need to be corrupted AND there would need to be a TEE security failure. Because of relative simplicity (and additional slashing penalties), this is less likely for the anti-equivocation use case that can build the block normally and only have signing constrained by a TEE (a long-known approach).

To avoid a long aside in this article, let us be satisfied with the claim that adding TEEs improves security assumptions over the single unrestrained sequencer case and leave the degree to which this is the case for elsewhere (click here for fighting).

Protocol 1.a: Single-Sequencer + rotation [liveness]

[

Rotating SS

872×421 52.3 KB

](https://collective.flashbots.net/uploads/default/original/2X/6/63d9e74a8d510df3e589eea130d4b2f4abb80f76.png)

Protocol 1 doesn't address the case of sequencers going down, but can be extended to do so. Currently, rollups like Optimism and Arbitrum leave the users with only the L1 (with a 24h delay) as a means to have transactions processed if the sequencer goes down. This could be made more graceful, either by removing delays and falling back to being based or by rotating the sequencer if it has been inactive for a certain period of time. With this kind of rotation technology available, one could go further and rotate the sequencer at some set frequency to increase neutrality and improve on censorship delays by increasing the diversity of nodes operating the system.

However, any kind of rotation of sequencer needs to have a sufficient delay period if we are to avoid reorgs. Consider a scenario in which the current proposer has offchain confirmed up to block n+k and posted up to block n on L1. If we were to execute a rotation too fast we wouldn't give the current sequencer enough time to overcome realistic censorship and gas market issues, and the next sequencer would start building on block n, creating a conflicting chain.

[

Choosing head

852×239 17.4 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/266a3770bb48c862753a44dc4b5bb5a4eacd84c5.png)

Of course, the next sequencer could try to post n+1,…,n+k themselves, but this opens up the attack in which the current sequencer intentionally waits to reveal these blocks to cause a reorg. Hence, a large delay is required before a new sequencer can take over.

(An interesting, but unexplored way around this delay for scheduled rotations is to require blocks leading up to a rotation to be signed by both the current and next sequencer.)

Apart from the downtime itself, this delay points to a larger challenge. In this kind of architecture, there is no (known) way to be sure that a sequencer is producing blocks much faster than the delay period

and sequencers do have incentives to introduce these kinds of delays for greater MEV capture (1, 2). Large artificial delays are fortunately detectable, but we would rather not rely on social mechanisms for dealing with this issue.

As far as I know, Aztec has published the most ontheir thinking around sequencer rotation, albeit with a more complex setup than most rollups as they support private state.

Protocol 2: Censorship-Resistance Committee [Liveness]

[

CR committee

1216×543 42.4 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/27723041845ed328d94fb26219b7811a91b34fcd.png)

We can address (weak) censorship

by using a committee. There are a few options for how to do this, but, at a high level, the implementation looks like this:

- A user transaction is sent to a committee of CR nodes
- At a certain interval, each CR node signs a list of transactions it has seen in that round - we can call these “inclusion lists”
- The block produced by the sequencer must include transaction from at least k/m of these lists, with different rules to handle the special case of full blocks.

Since we don’t require CR nodes to exchange messages, this should run quite fast.

Of course, there are many unspecified implementation details around things like gas fees, but these all seem solvable. The harder tradeoffs come in with:

- DA/CR tradeoff: Rollups using succinct proofs or building blocks in TEEs (not just using TEEs for anti-equivocation) should be able to enforce this CR rule without too much additional cost. A real-time proving scheme - which we don’t have but maybe one day could with faster proving schemes - can add a check of the lists as an additional validity condition. Similarly, TEEs could be configured not to release any blocks without being fed a sufficient number of these lists. However, without a using a TEE or massive improvements in validity proofs, we would need to post the lists on chain (and propagate to full nodes), incurring additional DA costs.
- System liveness/CR tradeoff: if we require m/m lists to be considered for a block, then censorship resistance is maximally high, but any node going offline halts the system. We can reduce the fraction of lists needed, but as we do this we trade off censorship resistance for better overall liveness.

The impact on system liveness can be softened by having a good fallback mechanism, although this is an open area of research. Given that many architecture assign a single node to produce the next block, requiring a few nodes in a large committee to be live may not be much of an additional liveness risk.

Technique 3: Fast Finality Committee (“finality gadget”) [safety, liveness]

This is Vitalik’s “compromise approach”.

[

Finality committee

1209×474 34.8 KB

](https://collective.flashbots.net/uploads/default/original/2X/f/f0fa3653805bf91edbb69f62f6f61d20785a0822.png)

We can bolster the safety

property by introducing a committee that signs only one block per height. Nodes tracking the rollup and bridge contracts would expect blocks to be signed by some number of nodes from this committee and the committee would only ever sign one block per height. Thus, for a reorg to happen, both the sequencer and more than 1/3 of the committee must equivocate - an attack that can be further defended against by requiring committee members to run TEEs for anti-equivocation as well. Note that the committee signatures are an additional requirement for a block to be valid.

It’s fine if the committee signs an invalid block because the bridge and full nodes are still checking the state transitions - the sequencer can be slashed in this case.

Note we don’t necessarily need to run a consensus protocol since we don’t need committee members to agree on what block was canonical at height h to sign a block at h+1. This means that we don’t even require committee members to talk to each other in the hot path, keeping latency low.

Aside from improved safety guarantees this technique also introduces:

- Additional liveness tradeoff, similar to the CR committee. Since the protocol must be safety favouring, more than 1/3 of the committee going offline will halt the finality mechanism. A fallback to a single sequencer model could be useful here, but more thought needs to go into how this is triggered.

- Easy sequencer rotation. As it stands, there are a variety of reasons (censorship, fee market spikes) that an L2 batch may not land on L1 despite the sequencer (or whatever party) trying to post the batch honestly, requiring long delays when changing the sequencer (see above). Conversely, the sequencer should have no difficulty posting blocks to the committee who are listening only to the sequencer. Thus, it is easier to change the elected sequencer either when the sequencer is down or at a predetermined frequency. This fixes the lack of timeliness incentives

in non-committee-based mechanisms, provided the committee remains timely. Regular sequencer rotation also increases network neutrality through increasing operator and geographic diversity.

This approach is compatible with both the protocols mentioned above:

- The same finality committee can also perform the censorship resistance

duties of building inclusion lists. This is reminiscent of existing consensus protocols (e.g. see here). Combining the two provides a way around the DA/CR tradeoff mentioned for protocol 2 since the finality committee could check this condition while the bridge contracts don't - at the cost of some security assumptions.

- The sequencer can also be required to run a TEE, providing user MEV protection

.

- One can also use TEEs as an additional safety device so that users can get confirmations faster than the "block time" of the consensus committee. If the TEE security model were to fail, the sequencer could only execute a reorg up to the last block posted to the consensus committee, which should not be long at all.

This would impact the leader rotation scheme employed since there must be enough time for ecosystem participants to post the blocks which a sequencer may have signed and shared, but not propagated to the consensus committee (I believe this lower bounds the number of missed slots before rotation at 2). Its unclear whether the complexity is worth the latency gain.

How does this relate to interop?

The protocols above all consider rollups in isolation, which is not representative of reality. In this section, we briefly touch on how the protocols above relate to concepts in interoperability (message passing) between rollups, avoiding questions like "if two rollups share have dependent validating bridges, are they different rollups"?

Native bridge

- if you want to natively bridge things like assets (native=without additional trust assumptions) without going through L1 then you need a shared validating bridge like what Optimism's superchain and Polygon's Agglayer are supposed to allow. This is unrelated to the above.

Bridge speed

- Bridge operators only take an action on the destination chain if the triggering action happened on a source chain. Thus, providing faster confirmations that an action did irrevocably (or with high probability) happen on the source chain allows for faster and low-risk bridging.

Atomicity

- Cross-chain atomicity is the guarantee that a set of transactions targeting different chains will only be included if all of their execution paths succeed and they are included at the ~same timestamp. We consider motivations for cross-chain atomicity as outside the scope of this article. To enable cross-chain atomicity, sequencers for different chains (at the same height/timestamp) would need to trust each other and have confidence that they will be able to land their own block. The most practical approach to this is to have one entity constructing blocks for multiple chains. Done naively, this introduces correlated liveness risk with a single point of failure which one may not want. To navigate this tradeoff we can do the following:

Technique 4: Happy Path Atomicity

[

x-domain builder

1110×744 70.7 KB

](https://collective.flashbots.net/uploads/default/original/2X/f/f92e05d3989f14f57d7131b789a24829619230b7.png)

We can allow cross-domain atomicity for a construction employing any of the techniques mentioned above by applying the following protocol:

- the sequencer maintains the same network responsibilities, but receives blocks from an atomic builder
- the builder constructs blocks such that if all blocks are accepted, transactions can be executed atomically
- if the sequencer does not receive the block in time, it defaults to local block construction. Due to this, and potential sequencer failures, the builder can never provide full atomicity guarantees.

This transformation provides cross-chain atomicity guarantees assuming all actors follow the protocol and falls back to previous protocols if this is not the case. If a sequencer is running a TEE for user MEV protection, then so must the atomic builder. This construction also requires the sequencers to publish blocks at the same time - something which can be synchronised by the builder sending blocks.

The protocol, as described above, introduces erodes the neutrality of the overall system as a single actor is given greater influence. Ideally, inclusion list committees and TEEs can reduce much of the freedom of these central actors. We can also introduce rotation schemes for these x-domain builders, but that’s out of scope for this post.

Final thoughts

This post mentions several desirable properties that rollup architectures should seek to provide with strong assurances and proposed several complementary protocols for achieving this. Notably, this post didn’t touch on incentives as much as it should have. For example, sequencers collecting MEV could be incentivised to wait longer than desirable before committing to their next block. Additionally, protocols 4 does not consider all the complexities of distributing revenues from shared operations like cross-chain arbitrage, nor do any of the committee based approaches discuss the challenge of incentivising and selecting a committee.