Resistance is ~not~ futile; CR in mev-boost

[

upload_dfc0918679b14108d6784e891e12c7cf

1236×1357 102 KB

](https://ethresear.ch/uploads/default/original/2X/c/c30027d4d740584304bd949a6aedcd5b91274fb9.jpeg)

\cdot

by mike

sept 27, 2023

\cdot

tl;dr;

Multiple proposals to incorporate stronger censorship resistance properties into mev-boost

have been presented. We analyze mev-boost with inclusion lists

, an idea voiced by Bert and Quintus of Flashbots, and mev-boost+

, a design put forth by Sreeram and Kydo of EigenLayer. With this context, we present a third option, referred to as relay-constructed inclusion lists

, which makes a slightly different set of tradeoffs.

There is no silver bullet for out-of-protocol censorship resistance; as a core desideratum of Ethereum, an in-protocol mechanism is much preferred. However, given inclusion lists will be enshrined at the earliest in Electra, experimenting with ideas that we can start implementing today is extremely valuable because it both provides an avenue to prototype similar constructions and immediately tries to improve the censorship resistance of the protocol. These three proposals seem worth considering as potential interim solutions in light of the recent rise in builder censorship observed on the network (see Toni's thread and website). Readers familiar with the existing proposals may choose to jump to the "Relay-constructed ILs" section to see the new design and its analysis.

\cdot

Related work

title

description

Censorship Resistance: crlists in mev-boost

Quintus' proposal

Comment #12 on Vitalik's post

Bert's proposal

The Cost of Resilience

Min-bid post

Preserving Block Proposer Agency with mev-boost using Eigenlayer

Sreeram's proposal

The litList (crList) Builder

apriori's post

MEV-boost+/++: Liveness-first relay design

Kydo's proposal

Agency & MEV-boost++

apriori's summary

Censorship resistance via restaking

Sreeram's SBC talk

\cdot

Acronyms

source

expansion

IL

inclusion list

CR

censorship resistance

\cdot

Thanks

Many thanks to Kydo, Quintus, Bert, Barnabé, Tim, Toni, mteam, Justin, Jon, stokes, Vitalik, Terence, & Francesco for discussions and comments.

Existing proposals

As seen in the "Related work" table above, there are several proposals for out-of-protocol censorship resistance implementations – let's start there.

mev-boost with inclusion lists

This proposal, from Quintus and Bert, strongly mirrors the ideas for enshrined inclusion lists. Proposers construct a list of transactions that they believe are being censored and communicate that list to the relay. The relay enforces the inclusion list along with the other checks it performs on the builder block submissions. The figure below captures the flow.

[

upload_df8b522c68f24839e1515640a11fe500

1211×616 83.5 KB

](https://ethresear.ch/uploads/default/original/2X/6/6d40894c790830b33826f0c75b31599fecf4ba1a.png)

Steps

1. The proposer sends an inclusion list to the relay.
2. The relay forwards the list to the builder.
3. The builder submits (or abstains from submitting) conformant blocks.
4. The relay checks the builder submissions to assert their conformance.
5. The proposer calls getPayload()

with their signed header.

1. The relay publishes the full block to the network.

The validator is given some agency over the set of transactions that get included and the builder is forced to either build a conformant block or abstain from participation in that slot. This mirrors the proposer-builder relationship described in ["No free lunch – a new inclusion list design

"](https://ethresear.ch/t/no-free-lunch-a-new-inclusion-list-design/16389) (the key difference being that in the protocol, the proposer constructs the list for the subsequent slot for incentive compatibility reasons – see "Forward Inclusion Lists").

In this design, relay trust is extended: the proposer trusts the relay to enforce their inclusion list.

Since the proposer is already trusting the relay for payment verification, payload availability, and block validity, including IL enforcement doesn't change the overall trust assumption (since it's a repeated game, a misbehaving relay will quickly become irrelevant). We could

use a Merkle proof to ensure that a specific transaction is included in the block, but this would not be sufficient to ensure that the block is valid, which would require a full zkEVM.

Downsides of mev-boost with ILs

– There are a few issues that arise when considering this design.

- Proposer incentive compatibility

–

Proposers are tasked with creating the inclusion list for their slot. By constraining the builders, proposers run the risk of earning less for their slot if builders abstain from participating (in the extreme case, if no builder submits a block, the proposer will revert to local block-building mode). This is what prompted the "forward" version of the enshrined inclusion list designs. Forward inclusion lists in mev-boost

are less viable because the next proposer may use multiple relays and there is no agreed-upon enforcement of the IL. Even if the relays communicated the status of their IL for the subsequent block, the mev-boost

protocol itself could easily be forked to circumvent this restriction. Out-of-protocol solutions require some amount of proposer altruism.

- Additional half-round of communication between proposer and relay

–

The proposer has to communicate their inclusion list to the relay. This adds another round of communication that must take place near the beginning of the proposer's slot (because it relies on the validator's current mempool view).

- Additional half-round of communication between relay and builder

–

The builder has to fetch the inclusion list from the relay. This also takes place at the beginning of the slot, but is a one-time action and not in the fast path of the builder submissions.

- Validators have discretion over censorship resistance

–

With an out-of-protocol and "opt-in" approach, large validators may shy away from touching regulated transactions. The validators may prefer to have *less* control over the contents of their blocks in the same way that ISPs have no visibility into the contents of encrypted data that flows over their networks. It is probably still better to have the CR properties enforced by validators rather than builders or relays, but it is still worth acknowledging the risk, especially in out-of-protocol solutions (though in-protocol solutions may suffer from the same problem). We do benefit from "inclusion" being optional and not enforcing any form of "exclusion", which would pose a different question to the validators.

- Relay trust

–

The additional trust assumption doesn't differ significantly from the existing checks executed by the relay. This is less of a downside and more of a note; we will circle back to this point.

mev-boost+

EigenLayer's design uses a slashing condition on restaked ETH

to allow the proposer to credibly commit to selling the top of their block to a builder. We focus on mev-boost+

because it seems more relevant in the short term; mev-boost++

is more intricate and involves a high-speed, external DA layer as part of the block production flow. The design of mev-boost+

is represented in the figure below.

[

upload_ea14c8aa06a07ed8e9aaeabc05907874

1290×546 99.5 KB

](https://ethresear.ch/uploads/default/original/2X/5/546dee70aacc7b3e6e176f885d2a02c74cc19fd1.png)

Steps

1. The builder sends blocks to the relay.
2. The proposer signs a commitment to use the builder's transactions as the prefix to their block and sends the commitment to the relay.
3. The relay stores the commitment.
4. The relay responds by sending the builder block to the proposer.
5. The proposer constructs and publishes the full block, appending any transactions as a suffix to the builder block.
6. The relay checks the published block for a commitment violation (changing of the block prefix). If such a violation is found, the proof is submitted to EigenLayer, and the restaked ETH

is slashed.

This design leverages restaking and the proposer's signature to enforce that the builder block is not modified. It gives a large amount of agency back to the proposer because they are responsible for constructing the full block and getting it published. They can add any arbitrary transactions (so long as they are valid) in their suffix.

In this design, relay trust is modified: the builder trusts the relay to store the proposer's commitment and enforce the slashing in case of a violation.

This replaces the existing trust of the relay verifying the proposer correctly signs the header before the block is published. You could add additional complexity (e.g., the relay publishes the proposer commitment or sends it directly to the builder), but since the builder is already trusting the relay to not steal their MEV in the first place, it seems logical to rely on the existing trust.

Downsides of mev-boost+

– There are a few issues that arise when considering this design.

- EigenLayer/other restaking provider dependency

–

By explicitly depending on a non-native piece of software, we take on additional risk. Especially considering this software has not been tested in highly adversarial economic environments.

- Builder incentive compatibility

–

The builder takes on risk by sending blocks to a mev-boost+

relay because they no longer have atomicity guarantees of their payload. The proposer has access to their transactions in the clear and has the power to reorder and unbundle (with the consequence of a restaked slashing). The Low-Carb Crusador proved that unbundling builder blocks can be extremely lucrative. Additionally, under mev-boost+

the proposer doesn't need to equivocate to unbundle, so the attesting committee cannot enforce the commitment; the EigenLayer slashing condition is the only enforcement mechanism in this case. It's not clear why a builder would choose to take on this risk when they can instead keep the same guarantees by sending to a full-block relay exclusively (Kydo points this out in his post).

- (same as above) Proposer incentive compatability

–

The proposer wants the highest-paying block. If they connect to a full-block relay and a partial-block relay, the bids from the full-block relay may be higher (it is strictly better for the best builder to only submit blocks to the full-block relay – see ["The centralizing effects of private order flow on Proposer-Builder Separation"

](https://arxiv.org/pdf/2305.19150.pdf)). This forces the proposer to put a "price on their agency", where they decide how much of a haircut they are willing to take on their MEV reward to retain the right to add a suffix. Out-of-protocol solutions require some amount of proposer altruism.

- Solo-stakers may be less likely to restake

–

We rely heavily on the long tail of solo-stakers for censorship resistance in the protocol (e.g., those who choose to self-build blocks today despite the existing mev-boost

infrastructure – ty Vitalik :-). This set of validators is probably not as likely to chase yield by restaking their ETH

(though EigenLayer may find a way to directly incentivize specifically solo-stakers). Thus it is not clear who would sign up for mev-boost+

and if they would meaningfully contribute to the CR of the protocol.

- (same as above) Validators have discretion over censorship resistance

–

With an out-of-protocol and “opt-in” approach, large validators may shy away from touching regulated transactions. In some ways, the validators may prefer to have *less* control over the contents of their blocks.

- (same as above) Relay trust

–

The additional trust assumption doesn’t differ significantly from the existing checks executed by the relay. This is less of a downside and more of a note; we will circle back to this point.

Relay-constructed ILs

While the above designs are viable and make different trade-offs, we believe there is an even simpler solution. We already have the relay serving as a trusted third party, so why not leverage that by asking the relay to construct the inclusion list on behalf of the proposer? The proposer can sign up for the relay-constructed inclusion list in their validator registration, which triggers the relay enforcement of the list. The figure below captures this.

[

upload_085e70f28e11cbc3855990c78c4ddddf

1186×585 94 KB

](https://ethresear.ch/uploads/default/original/2X/2/2351b69f0470f7b606958b5c55cf2532a0ff73c5.png)

Steps

1. The validator registration contains a field indicating if the proposer would like to make use of the inclusion list feature on the relay.
2. The relay listens to the P2P network and constructs an inclusion list based on transactions it labels as censored.
3. The relay forwards the list to the builder.
4. The builder submits (or abstains from submitting) conformant blocks.
5. The relay checks the builder blocks to assert their conformance.
6. The proposer calls getPayload()

with their signed header.

1. The relay publishes the full block to the network.

This design relies on the fact that the relay is neutral and can enforce the IL on behalf of the proposer. Further, we could incorporate a feature flag similar to min-bid, where relays only enforce the inclusion list if the difference between the highest-paying conformant block and the highest bid is kept in a reasonable range (say 0.01 ETH

). We present validators with an additional degree of freedom; they can choose if they want to contribute to the censorship resistance of mev-boost

without explicitly identifying transactions and constructing an inclusion list.

In this design, relay trust is extended: the network trusts the relay to construct and enforce inclusion lists.

This would require relays to implement this behavior, but relays have already been partitioned into the censoring & non-censoring sets. Additionally, it would allow validators who connect to non-censoring relays to express an even stronger preference for censorship resistance by asking the relay to enforce ILs on their slot.

An alternative mechanism could allow the relay to construct block “suffixes” and modify the payload to include transactions in its IL. This adds latency to the block submission pipeline and makes the submissions incompatible with optimistic relaying (because the full contents need to be downloaded and simulated before the suffix is added), but it could offer a better alternative compared to builders simply not submitting to that relay for a slot. It also changes the relay to be more active in the block builder process; modifying the contents of the payload changes the contract with builders and could open the opportunity for MEV to be captured in the suffix. Nevertheless, it is an interesting thought experiment in terms of allowing censoring builders to still compete for the slot and giving them plausible deniability for having touched specific transactions (because they only sign over their prefix which excludes them).

Pros

- Minimal changes to the API.

If the relay constructs the list locally (rather than sourcing it from a proposer), then most of the logic is self-contained. The validator registration would need an additional boolean field to provide a way for validators to express their preference, but the default could be false to allow for backward compatibility. Critically, validator consensus client software would not require any changes.

- Relay trust already exists.

As mentioned in the analysis of all three proposals, builders and proposers already trust the relay for numerous tasks (e.g., payment verification, payload DA, protection from MEV stealing). This trust is powerful because of the repeated nature of the game; any misbehaving relay will be ignored immediately. If none of the proposals remove the trusted relay, why not rely on that trust to improve the overall censorship resistance of the protocol?

- Forward compatible.

If and when future CR proposals are implemented (either in-protocol or out-of-protocol), the relay behavior can easily be modified to accommodate these changes. Indeed the in-protocol IL schemes would easily fit into the relay enforcement bucket given it becomes part of the block validity condition.

- No validator latency issues.

Since the relay constructs and serves the IL to the builder, there is not an additional round of communication between the proposer and the relay before the block building begins.

- Compatible with optimistic relaying.

If we consider the IL enforcement as an extra-protocol block validity condition, then the optimistic relay can simply demote a builder that misrepresents the conformance status of a specific submission. Thus latency-optimized relays can still make use of the relay-constructed ILs

(note that the other two proposals listed above are also compatible with optimistic relays).

Cons

- (same as above) Proposer incentive compatibility

–

Proposers need to opt-in and may earn less rewards. Out-of-protocol solutions require some amount of proposer altruism.

- Though if the builders submit compliant blocks, then the validator rewards could increase slightly because of additional gas-paying transactions being included.
- Though if the builders submit compliant blocks, then the validator rewards could increase slightly because of additional gas-paying transactions being included.
- (same as above) Additional half-round of communication between builder and relay.
- Not a major concern because it is out of the fast path of builder submissions.
- Not a major concern because it is out of the fast path of builder submissions.
- Requires relays to do more work.

Relays are already un-funded public goods. This proposal requires a relay update and some additional development work to

support the IL logic. * This argument doesn't carry much weight because the amount of additional work is quite scoped and relays already run execution layer nodes that have mempool visibility. The software to identify censored transactions could be of general-purpose use and developed by non-relay teams. Additionally, some of this software may be reused in the enshrined IL implementations.

- This argument doesn't carry much weight because the amount of additional work is quite scoped and relays already run execution layer nodes that have mempool visibility. The software to identify censored transactions could be of general-purpose use and developed by non-relay teams. Additionally, some of this software may be reused in the enshrined IL implementations.

- Increases rather than decreases relay dependency.

Intuitively, this may feel like we are going in the wrong direction. * While this is true, we are not adding a strong relay dependency. We are using the existing infrastructure to improve the current status of censorship resistance, which is already a rather subjective metric. This is an immediate-term fix while enshrining a censorship resistance mechanism in protocol remains the medium-term goal.

- While this is true, we are not adding a strong relay dependency. We are using the existing infrastructure to improve the current status of censorship resistance, which is already a rather subjective metric. This is an immediate-term fix while enshrining a censorship resistance mechanism in protocol remains the medium-term goal.

- Some relays will not construct ILs.

Relays that censor presently are certainly going to avoid constructing ILs. * This is also true, but it doesn't feel like a deal breaker. No CR scheme is going to get 100% adoption, and we already know that many relays are committed to credible neutrality. Those are the relays we would expect to adopt inclusion list construction. Additionally, relays could identify the types of transactions they include in their ILs.

- This is also true, but it doesn't feel like a deal breaker. No CR scheme is going to get 100% adoption, and we already know that many relays are committed to credible neutrality. Those are the relays we would expect to adopt inclusion list construction. Additionally, relays could identify the types of transactions they include in their ILs.

Relay-constructed ILs

is a simplification of mev-boost with ILs

that reduces the proposer-side changes to a minimum. Additionally, it gives proposers an additional way to contribute to the censorship resistance of the protocol, without explicitly constructing the IL themselves; the validator still signs the header blind and has no idea of the contents of their block until it is revealed.