Many thanks to @fradamt @casparschwa @vbuterin @ralexstokes @nikete for discussions and comments related to the following post. Personal opinions are expressed in the first-person-singular.

Protecting the proposer and ensuring liveness of the chain are a big part of why PBS is considered to be moved into the Ethereum protocol. Ideally, when the proposer utilises the services of a builder, there is a contract between parties for the delivery of some goods (valuable blockspace), and the contract is honoured atomically:

- Either the contract fails to be made and the goods are not delivered/block content is not published, or
- The contract is successfully made and payment always succeeds, no matter what the party committed to supply the goods does.

This stands in contrast to MEV-boost, where a proposer could enter into a commitment with a relay, by signing a block header, after which the relay could fail to publish the block in time, and the proposer is not trustlessly compensated while missing the opportunity to make a block.

But with our version of in-protocol PBS (IP-PBS), we bind ourselves to a very specific mechanism for making these contracts, where there is trustless infrastructure for the proposer to sell off entirely

their right of making the block. Amendments exist, such as inclusion lists, or increasing proposer agency by letting them build part of the block. Still, few results exist showing that a proposer can be fairly unsophisticated and achieve most of the value their position confers upon them.

As an example, what if there is economic value for the proposer in selling the rights to make their block in advance, say 10 slots before? Under IP-PBS, a cartel of builders must honour an out-of-protocol market, where the winner of the blockspace future (perhaps auctioned at slot n-10

) trusts the winner of the slot n

IP-PBS auction to let them make the block. Yet the notion of an IP-PBS “winner” is semantically violated, and the value cannot be achieved by an untrusted proposer. Builder colocation with trusted proposers could also increase the delta between what IP-PBS returns to an unsophisticated proposer and what trusted proposers can achieve, beyond simple latency improvements.

In such cases, the incentive to use IP-PBS is cosmetic, as builders can make arrangements out-of-band with proposers, who then ignore bids received via the IP-PBS facility. The incentive to “go around” IP-PBS is reduced with mechanisms such as MEV-smoothing (much in the same way EIP-1559 makes off-chain agreements moot), but they entrench further a specific allocation mechanism of blockspace, which if suboptimal, doesn’t allow the Ethereum protocol to realise the highest possible social welfare.

It is hard to foresee what the future will ask of the protocol, or how economic value will be realised by agents interacting with the protocol. The current version of IP-PBS feels strongly opinionated with respect to the organisation of a market around blockspace, while what we seem to be trying to solve for is trustless infrastructure for commitments to be honoured, e.g., the commitment to provide (possibly partial) block contents or be penalised up to the promise that was made. Meanwhile IP-PBS attempts to set a “good default” for unsophisticated proposers, yet appears to require proposer intervention to ensure censorship-resistance.

In other words, we may need a mechanism for credible signalling in general

, not a mechanism for credibly realising one specific signal

(”I can make the whole block for you and return you x

amount of ETH for it”). With such a mechanism, we can look towards future protocol duties which proposers may wish to outsource, as outlined in Vitalik’s recent ethresear.ch post, ideally without requiring changes to the protocol and without needing to design and enshrine a specific mechanism for the contracting of proposers and third-parties.

This post explores this alternative and attempts to draw its consequences, in the spirit of exhaustive search of the design space. Many open questions remain to be answered to consider this direction a feasible alternative to existing designs.

TL;DR

- In the following we build towards a trustless, permissionless scheme for validators to enter into commitments with third parties. We start by protocolizing Eigenlayer, to ensure that out-of-protocol commitments entered into by validators are reflected into the protocol, e.g., tracking their effective balance if they are slashed by Eigenlayer middleware.
- We recognise that this is not enough, since it allows for attacks up to some “maliciousness upper bound”: the protocol cannot enforce the validator won’t deviate when the profit is greater than the slashed amount. So we need to go further, and not simply move to the protocol the outcome of commitments (if the validator is slashed or not) but also whether the commitment was satisfied or not

, and base our protocol validity on it.

- As a stepping stone, I propose an optimistic block validity

notion, where a validator could do something slashable with respect to the commitments they entered into, and such a slashable behaviour could be made canonical by attesters, but everyone involved eventually gets slashed.

- To return to pessimistic block validity

(validator behaviour must be proven correct (no slashing) for block validity), we allow proposers to submit commitments expressed as EVM execution

in their block. Attesters can then simply check validity of the block they receive with respect to the commitments that were made.

- We then need to deal with data availability, to ensure that neither proposer nor committed third-party is able to grief one another. Here we observe a fundamental difference between “selling auctions”, where the proposer auctions something valuable to a third-party, e.g., the right to make a block, and “procurement auctions”, where the proposer attempts to obtain something valuable from the third-party, e.g., a validity proof of the execution payload.
- Finally, assuming the existence of such a generalised market for commitments, we revisit the idea of “proposer dumbness”, as expressed by the addition of protocol features aimed at levelling the playing field between dumb and smart proposers.

In-protocol Eigenlayer (IP-Eigenlayer)

Eigenlayer is a good starting point to build towards our mechanism, as it allows, in its own words “permissionless feature addition to the consensus layer”. However, relying only on Eigenlayer-provided guarantees is weaker than what the protocol may enforce, even if it comes as a handy tool to augment current out-of-protocol mechanisms such as builder markets.

One issue with Eigenlayer, seen from the PoS protocol’s perspective, is the principal agent problem (PAP). The protocol outsources its security to a set of validators, which are staked. When validators are slashed out-of-band by Eigenlayer middleware, the protocol does not realise that the agents to which it has delegated security may have weaker incentives to participate in the protocol than the protocol is led to believe via its own state. One way to make the protocol aware of such discrepancies is to allow Eigenlayer or any other system to update a validator’s in-protocol balance, a protocol we call IP-Eigenlayer

.

We let validators allow external addresses to slash them, and the protocol is able to see the amount slashed. For instance, a validator is allowed to sign a message saying “Address 0xabc

is allowed to slash me”, and this message is included on-(beacon-)chain, after which 0xabc

can submit a slashing message to the protocol, for some amount of stake.

[

eigslash

976×592 38.7 KB

](https://ethresear.ch/uploads/default/original/2X/0/06ee025547d2f7530c5116960a5776056a3e5c5c.jpeg)

Allowing external constructs to influence the state of the protocol may sound unwise. But with the existence of Eigenlayer, validators will

enter into such schemes, so protocolizing it doesn't reduce the potential for protocol misalignment coming from validator restaking and getting slashed out-of-protocol, either due to operational errors, not performing their duties correctly, middleware smart contract risk, bribing or anything else. Yet protocolizing removes part of the principal agent problem coming from restaking.

With IP-Eigenlayer, the protocol at least has a correct view of the amount currently guaranteeing safety of the system. There are multiple issues to think through, e.g., does an Eigenlayer slashing exit the validator from the active PoS set or simply diminishes their balance, should there be a fee market for restaking/slashing beacon chain messages etc, but we only offer here this construction as a thought experiment to build towards a more general mechanism.

Indeed, while the PAP problem is removed, this does not solve the commitment-based problem of maliciousness upper bound

: a restaked validator could decide to get slashed if they expect their malicious action to net them a greater payoff than the slashed amount.

Protocol-enforced proposer commitments (PEPC, "pepsi")

If we want to generalise beyond the whole block auction proposed by IP-PBS, we need the validator to be able to enter into any commitment, while being secured by the protocol. We should first recognise that there are many protocol-related commitments which are verifiable on-chain, via the BEACONROOT

opcode, for instance, "the exec-block made at slot x

was signed by builder y

, as committed by proposer z

". There is a space of such "verifiable protocol-related commitments", indeed smaller than the larger space of "all possible commitments", which could include exotic things such as "I will slash proposer z

unless they show up at my doorstep dressed as a clown" (word to the wise: don't enter into such commitments!)

To determine whether a commitment was entered into and whether it was appropriately fulfilled, the protocol needs to distinguish between three potential outcomes:

1. The validator entered into a commitment with a third-party and either:

i. The third-party delivered their part of the commitment, the commitment payout is processed.

ii. The third-party did not deliver their part of the commitment, the commitment payout is processed.

1. The validator never entered into a commitment.
2. The validator entered into a commitment with a third-party, then did something violating that commitment, e.g., stole the goods from the third-party for their own benefit.

We want the protocol to make safe only

one of the first two alternatives, (1)

or (2)

. Importantly, it should not be possible for the validator to enter into a commitment with a third-party and then

finalize a version of history where they did not enter into a commitment with a third-party. The current two-slot IP-PBS design satisfies this property, if we replace “finalize” with “make safe up to the builder’s risk tolerance”, since we do not (yet

) have single-slot finality (SSF).

In the following, we have the[two-slot IP-PBS pattern] in mind, where the proposer first enters into and records commitments, to be made safe/finalized in a first round, after which committed third-parties are expected to deliver on the commitments in a second round. Even though we use this pattern as a template, there could be important deviations to consider and possibilities to generalise beyond it, e.g., schemes where the proposer is able to enter into commitments well before their own slot. We believe this does not undermine the core idea expressed in the following.

Besides ensuring that commitments cannot be reverted, the other ingredient needed is for the protocol to determine whether the commitment was fulfilled, i.e., discriminate between outcomes (1.i.)

and (1.ii.)

We build towards this using the IP-Eigenlayer mechanism described above, adding attesters as validity checkers

.

Committee-based optimistic block validity

In the IP-Eigenlayer construction, external commitments are entered into by the validator via a smart contract deployed on the execution layer. An instance of such a commitment is “I promise to let builder y

build my exec-block”.

To see why attesters need to check validity of the commitment fulfillment, consider the following attack from the proposer:

- Suppose the proposer commitment is finalized or made safe enough for the committed third-party to be willing to release their goods.
- The committed third-party releases their contribution.
- The proposer “steals” the goods (think bundle theft in a blockspace market), by e.g., releasing a block violating the commitments they set, in time for attesters to vote on it.
- The proposer is slashed by the protocolized Eigenlayer, as the violation of their commitment can be proven on the execution layer.
- Still, the proposer makes off with the goods’ value because attesters make canonical the proposer’s theft. This value may be far greater than the stake they committed.

In other words, we run into this issue with non-protocol-enforced commitment-based schemes, because it is possible for the proposer to make canonical a history which violates their commitments.

[

youshouldleave

1654×924 79.8 KB

](https://ethresear.ch/uploads/default/original/2X/0/05f7869c2005f7fd654fa068f7976b4fc95afc7a.jpeg)

The trick is to realise attesters are also able to determine whether the proposer fulfilled their end of the bargain

, or whether they deviated. With IP-PBS, attesters won’t vote on a builder block made by the proposer, since it would violate the validity of the beacon chain state transition function (BC-STF) where the proposer gave rights to the builder to make the block. In this PEPC design, the BC-STF doesn’t have a specific validity condition for each commitment entered into by the proposer.

But attesters are still able to determine out-of-band whether the content they are voting on satisfies the validator commitments that were made. Again, this is a thought experiment, where we assume the presence of IP-Eigenlayer and commitments are enforced via the on-(exec-)chain restaking smart contracts. The protocol could further enforce that an attester voting on content violating the commitments stands to be slashed by the protocolized Eigenlayer mechanism. This optimistic block validity

condition allows the protocol to differentiate ex post

between outcomes (1)

and (3)

.

(For data availability/fulfilment of commitments, attesters do as they normally do, for instance, they vote “empty” with a (block, slot)

fork choice rule, allowing the protocol to differentiate between outcomes (1.i.)

and (1.ii.)

. We give more explicit details later in the post.)

We may be uncomfortable with slashing attesters based on a validity condition that is not executed “pessimistically in-protocol”, for at least two reasons:

- Since we have this Turing-complete space of commitments, someone could grief the system by entering into a very complex commitment, making a lot of attesters somehow vote on a block that did not satisfy the commitment, and force the chain to process a lot of expensive slashings for the attesters. This could be remedied by not including one proof per attester, but one proof per invalid block, and slashing all attesters who voted on such a block. Broadly, we have to think carefully about the computational metering of such operations, as mixing consensus-critical messages (slashings) with regular execution could lead to bad outcomes.
- We might also be concerned that all this business of slashing attesters mitigates the commitment-based failure but does not eliminate it. Whenever cost of corruption < profit

, there is a strict incentive to do the malicious thing, as it is not protocol-enforced, especially if there is enough money to go around to bribe attesters into getting themselves slashed (see also this great thread by Sreeram Kannan on all the open questions around slashings). For instance, the proposer who observes that their committed builder just revealed a 1000 ETH block could make their own block and bribe attesters into making the commitment-invalid block valid from the fork choice’s point of view.

Committee-based pessimistic block validity

So how could we make validation pessimistic, to enforce strictly that a safe/finalised commitment is fully satisfied by the proposer?

We do so by providing the proposer with a limited amount of EVM execution, by which they could specify the validity conditions of their commitments, e.g., a simple piece of code stating “the builder of the exec-block is y

”. Attesters execute the code, which is considered valid unless it reverts or returns False

or any other way to determine that the proposer commitments were not satisfied.

Third parties who enter commitments with the proposer are responsible for doing due diligence with respect to the proposer commitment, checking that the piece of code represents the commitment they expect to enter into. As knowledge about commitments mature, a standard library could exist, for instance containing IP-PBS and its variants as primitives.

One idea is to gas-meter the code contained in the proposer’s validity conditions, so there is no free lunch for the proposer. The more gas used by the proposer’s conditions, the less available to external builders they may want to call upon, or for the

block they intend to make. While this encourages parcimony regarding proposer commitments, it is possible for trusted proposer-builder coalitions to coordinate out-of-band such that the proposer does not expend the gas to make their commitment but the builder still provides them with a block, putting untrusted proposers at a financial disadvantage. It is perhaps possible to design a more appropriate scheme, e.g., a gas amnesty up to some limit, or disabling certain features for this specific execution (e.g., giving a gas amnesty provides “free data availability” that can be filled up opportunistically by the proposer, but storage operations could be disabled, such that validity conditions are read-only). In any case, the scope of validity conditions is an important point which should be considered more thoughtfully.

A SNARK-based scheme would be much nicer of course, where the proposer submits the validity conditions their block will satisfy (e.g., “my exec-block body will be built by y

”) and a validity-proof is submitted along with the block. Now the attesters have a simple validity check, which can also be encoded into the BC-STF: ignore a block that doesn’t have a validity proof stating the block satisfies all the commitments it entered into! Once there is a protocol-supported zkEVM, we can easily upgrade to such a scheme.

Properties of PEPC

To summarise the properties we are considering and how they are satisfied:

Property 1:

A proposer can’t revert a commitment they made.

- IP-PBS + Gasper:

Proposer can bribe attesters to revert original commitment

- IP-PBS + SSF:

Proposer must produce a safety fault to revert the commitment

- PEPC + Gasper:

Proposer can bribe attesters to revert original commitment

- PEPC + SSF:

Proposer must produce a safety fault to revert the commitment

Property 2:

A commitment-invalid block cannot be made canonical.

- IP-PBS:

Satisfied via BC-STF

- PEPC:

Attesters may be slashed for voting on a commitment-invalid block (optimistic) or

validity of the commitments satisfaction is part of block validity (pessimistic).

Property 3:

Outcome (3)

can never be enforced by a proposer if and only if

Property 1 and Property 2 hold.

Data availability in PEPC

Regarding data availability, we still need to generalise IP-PBS to the delivery of any

good. How do we make that happen? If the proposer enters into a commitment with a third party for them to deliver the SNARK proof of the exec-block's validity, how do we organise the delivery of that proof such that attesters recognise the third-party has done what was expected of them? Let's try to answer this by distinguishing between two types of contracts:

Contracts where the third-party pays

This looks more like the block auction. The proposer is selling away a profitable opportunity they alone possess, e.g., the right to make an execution block and reap the profits from such a block. An off-chain auction takes place, where buyers submit bids to the proposer, until the proposer eventually settles on a bid and commits to the builder who made the bid.

The difficulty is ensuring Bob cannot grief Alice, by making a high bid which they don't end up paying, regardless of their delivery of the good. How can we make Bob's bid binding, without including the notion of a "block bid" that the protocol understands? (remember, we want to have a more general system)

This can be done by making Bob's bid itself an EVM transaction that only executes if Alice made a commitment in her block to let Bob be the block builder.

- If Bob delivers his payload, the condition is trivially satisfied: a block where Bob is the builder must

be a block for which Alice entered into the commitment to let Bob build the block, since otherwise the block wouldn't be valid.

- If Bob doesn't deliver his payload, Alice must still be able to earn Bob's promise to pay. The construction could be similar to those of a payment channel, where a party goes offline yet the honest counter-party is able to exit with their legitimate funds.
- If Alice never commits, Bob's transaction cannot be executed.

To process Bob's bid, Alice could instead sell off the right to make a partial block: she reserves to herself the right to place Bob's bid at the end of the block, but lets Bob make the block prefix (unless Alice can trust Bob to insert his own bid in the block he is supplying to Alice, but obviously this is not in Bob's interest…)

Seeing Bob's message containing the (partial) exec-block, attesters of the "reveal" slot attest that they have seen Bob's message. Here we need a feature to allow proposers to build "template" blocks, where parts of the block can be retroactively applied once they are accepted by the attesters, e.g., the block made by Alice has partial content provided by Bob. For partial building, it should even be possible to template the execution payload itself, e.g., dividing it between "this part built by Alice" and "this part built by Bob". Let's assume this exists

this post is long enough and is primarily concerned with the outline of the PEPC mechanism.

Contracts where the proposer pays

In such contracts, the proposer is missing a part of their block, which they need to ensure block validity, e.g., we require blocks to provide a SNARK proof of their validity, otherwise they are dismissed as invalid. The proposer is willing to pay a third-party to do something on their behalf. An off-chain "procurement" auction could be run, where third-parties submit bids such that the proposer eventually picks the lowest bid, and commits themselves to paying the third-party against delivery of the good. Let's say that exec-block validity proofs are recorded on-chain as SignedValidityProof

messages. The proposer's commitment contract states "Validity proof of my block is signed by y

".

- Either the proposer does not select a bid, the "no-commitment" move is recorded (Property 1) and the onus is on them to provide the validity proof.
- Or the proposer selects a bid and the commitment to a third-party is recorded (Property 1).
- If the third-party delivers the proof in time, attesters vote on the message, which is retroactively integrated into the

block via block templating.

- What if the third-party fails to deliver the proof in time? There is an obvious griefing vector here: a malicious adversary could consistently bid a very low amount to underbid every honest supplier. There is no easy solution to this: one could require a “deposit” from the supplier, to be slashed if the supplier fails to deliver the goods. But even requiring a deposit seems to only add more weight to the property underlying the proposer’s business with the supplier: trust. The proposer has a very explicit incentive to pick a reliable supplier, since without the validity proof, the proposer’s block will be invalid.

- If the third-party delivers the proof in time, attesters vote on the message, which is retroactively integrated into the block via block templating.

- What if the third-party fails to deliver the proof in time? There is an obvious griefing vector here: a malicious adversary could consistently bid a very low amount to underbid every honest supplier. There is no easy solution to this: one could require a “deposit” from the supplier, to be slashed if the supplier fails to deliver the goods. But even requiring a deposit seems to only add more weight to the property underlying the proposer’s business with the supplier: trust. The proposer has a very explicit incentive to pick a reliable supplier, since without the validity proof, the proposer’s block will be invalid.

Does the nature of the trade mean that there is no trustless market where such griefing is impossible? Perhaps not. Maybe suppliers are required to supply along with their bid something like a zero-knowledge proof (”I can’t tell you what the proof is, but I can prove to you that I have a validity proof for your block”). But even then, after having convinced the proposer that they do know such a proof, the supplier could refuse to supply the proof itself. The supplier could otherwise encrypt the validity proof to some public key, to be decrypted by a committee of attesters via threshold decryption.

Revisiting “proposer dumbness”

In the scheme above, a proposer is considered “dumb” when they do not enter into any commitment: they make their own beacon block and execution-payload and whatever else is expected of them.

In IP-PBS, the “dumb” proposer is different. There, a “dumb” proposer would enroll themselves into listening to bids received for the right to make an execution payload, and select the highest bid by default—unless they are actually forced to, as in MEV-smoothing. “Dumbness” is subsidised by saying implicitly “the best possible outcome for you as a proposer is to passively listen to bids selling off your whole right to make a block and choose the highest one”.

My argument here is that restricting or narrowing the possibilities for proposers to organise their blockspace’s allocation does not make non-dumb proposers worse (since they can go around any mechanism deployed in protocol), but they could

make dumb proposers worse, e.g., by making it unable for them to organise in a way that is preferable to them. In this case, I would rather proposers be given template commitments they can decide to use or forego, than imposing on them a specific form of commitment in the shape of IP-PBS. I note as well that inclusion lists are a form of commitment that we now feel like proposers should be able to ape into.

In other words, we have a philosophy of promoting proposer dumbness, yet in many places problems are solved by giving proposers more agency. Good defaults certainly help set a baseline revenue to ensure validators remain decentralised. Still, by kneecapping proposers, we may also lose the hard-earned strengths of a massively decentralised set.

We may now revisit the conclusions drawn from theEndgame post. It was right to recognise that as the protocol gets more complex, it will become impossible for all proposers to “do it all” without increasing centralisation towards sophisticated proposers, unless certain protocol-required functions are outsourced to agents outside the boundaries of the protocol. But from this observation, three approaches may be considered:

- Do nothing, and let proposers figure out the best way to outsource their core functions, away from the protocol’s eyes. This is essentially what MEV-boost embodies today.

- Custom-build a market for every possible function that the proposer is expected to satisfy, and hope that the resulting market maximises proposer welfare as a default, while instantiating specific mechanisms to backstop the mechanism (e.g., censorship resistance schemes). This is essentially embodied by IP-PBS and its variants.

- Recognise that as the protocol ossifies, and given the huge complexity of figuring out each market structure independently (if it is at all possible to figure them out in a timeless manner, markets do change after all!), it may be better to provide a trustless infrastructure for proposers to enter into commitments with third-parties, even though there is no “near-optimal” default strategy encoded via the protocol and markets are structured via proposer commitments instead. Should there truly be a bullet-proof, “near-optimal” strategy for the proposer, it may be suggested as part of a client software default package. Meanwhile, the community figures out which commitments work best in various situations, and proposers are free to choose which commitments they are willing to enter into. At most, the protocol may provide more commitment-legos for proposers to choose from, such as whitelisting particular block templates. This feels like an appropriate scope for the protocol in general, which does not overdetermine the economic organisation of its actors.

In this post, we attempt to sketch this third way, while recognising that more work is necessary to gauge the costs, benefits and implementability of such an approach.