TL;DR:

We introduce an exploratory implementation of a based preconfirmations protocol for sub-second transaction confirmations on Ethereum, inspired on Justin Drake's proposal.

Thanks to @diego, William X, @ballsyalchemist, @ed, Tim Becker, @justindrake, Cooper Kunz, and others for the feedback and discussions.

[

preconfirmations

1024×1024 345 KB

](https://ethresear.ch/uploads/default/original/2X/e/e720183c77fcce7e0cfe98a3ca7f78c4a648666e.jpeg)

Background

Ethereum preconfirmations, or "preconfs" for short, are a proposed mechanism to enable faster transaction confirmation and improve the user experience on Ethereum, and subsequently, on layer 2 rollups and validiums. The key idea is to have Ethereum block proposers issue signed promises to users guaranteeing that their transactions will be included in a certain block. Users pay a tip to block proposers for this service. By acquiring preconfirmation promises from upcoming block proposers, users can get assurance of speedy inclusion, with latencies as low as 100ms. Preconfirmations can be applied for a broad list of use cases, like timely blob transactions for L2s with regular cadence, markets of future block space, and expressivity for inclusion constraints.

Implementation

Check out the implementation on GitHub

This implementation of preconfirmations leverages EigenLayer's AVS (Actively Validated Services) system to power the slashing mechanism, ensuring the security of preconfirmation guarantees. Ethereum validators who wish to participate in the preconfirmation service must opt-in via EigenLayer and run the AVS software, known as preconf-operator

, which enables them to receive and confirm requests with preconfirmation promises for blocks that they will propose. In the event that a preconfirmation is violated (safety fault) or not included in the established block (liveness fault), a fault proof can be posted to the service smart contracts, resulting in the retroactive slashing of the validator's stake.

To simplify the user experience, we introduce preconf-share

, a fork of MEV-Share that serves as a trusted matchmaker between users and block proposers. Users submit preconfirmation requests along with hints to the preconf-share

node, which then gossips the requests to validators that are the next block proposers. Based on the request hints and the associated tip, block proposers decide whether to promise a preconfirmation. The preconf-share

node receives and ranks the preconfirmation promises according to the user's preferences, such as the desired and maximum block. Once a preconfirmation promise is selected, the block proposer receives the raw transaction that should be included in the promised block. The user receives a receipt of the preconfirmation, and the signed promise of the block proposer is made public, allowing it to be used as evidence if the promise is violated (alongside the proof).

To further enhance the user experience and enable the use of preconfirmations in wallets without requiring changes, we introduce a trusted JSON-RPC wrapper for preconf-share

. This wrapper is forked from Flashbot's RPC endpoint and allows users to define their default preconfirmation preferences and hint criteria.

Preconf-Operator

Validators who wish to receive and promise preconfirmations must register in the EigenLayer service and run the preconf-operator

AVS client. Upon initialization, this client registers the given validator to the service smart contract and begins listening to the event stream of the preconf-share

node, which is run by a trusted third-party. When a preconfirmation request is received, the client evaluates its available parameters to determine whether to send a promise. For example, block proposers may be more receptive to requests that include logs or calldata information, even if they have lower tips, as it opens the door for value extraction or OFAC

limitations. Conversely, requests with little information can be accompanied by a high tip to balance the lack of information and have a higher inclusion rate. Block proposers must also consider other preconfirmations they have made for future blocks to make sure that new preconfirmations don’t collide with transactions obtained from the global public mempool.

If a block proposer sends a preconfirmation promise back to the preconf-share

node, the preconf-operator

client will wait for the defined window time in case they provided the best promise and consequently receive the complete preconfirmation information, including the raw transaction, to fulfill the promise. If a promise was made for a specific future block, the preconf-operator

client will hold the transactions until its turn to propose that block. It is important to note that although the preconf-share

node enforces a series of checks on both user requests and preconfirmation promises, block proposers can still be slashed if they provide an out-of-protocol promise for a block that they are not building/proposing (or if they delegated the work to another party that didn’t fulfill it).

Request Event Scheme

{ hash: string, inclusion: RequestInclusion, logs?: LogParams[], txs: Array<{ to?: string, functionSelector?: string, callData?: string, }>, }

- hash

: Hex-string of the transaction or request hash.

- inclusion

: Inclusion parameters, containing block target, max block, and tip.

- logs

: Array of JSON-encoded event logs emitted by executing the transaction using the last state.

- txs

: Transactions in the request, for each one: * to

: Transaction recipient address

- functionSelector

: 4-byte function selector.

- callData

: Calldata of the transaction.

- to

: Transaction recipient address

- functionSelector

: 4-byte function selector.

- callData

: Calldata of the transaction.

Promise Callback Scheme

{ hash: string, txs: Array<{ tx: string, }>, }

- hash

: Hex-string of the transaction or request hash.

- inclusion

: Inclusion parameters, containing block target, max block, and tip.

- logs

: Array of JSON-encoded event logs emitted by executing the transaction using the last state.

- txs

: Transactions in the request, for each one: * tx

: The raw transaction bytes.

- tx

: The raw transaction bytes.

Proofs & Slashing

The preconf-share

node publishes signed preconfirmation promises, enabling anyone to raise a dispute by posting a bond as soon as the target block has passed. The slashing smart contract verifies the authenticity of the preconfirmation promise by checking if it was correctly signed by a valid block proposer. If the signature is valid, the contract opens a challenge window, during which the block proposer has the opportunity to win the dispute by submitting a transaction inclusion proof.

The proof is constructed off-chain by the block proposer and passed to the slashing smart contract, which then submits it to Relic Protocol's Transaction Prover. The prover verifies whether the given transaction hash was indeed included in the specified block. If the block proposer successfully proves the transaction's inclusion, the disputer loses their bond, which is then transferred to the block proposer as a reward for their honesty.

However, if the block prposer fails to respond within the designated challenge window, the disputer automatically wins the challenge. In this case, the block proposer is considered to have broken their promise and is subsequently slashed, with a portion of their stake potentially being redistributed as a reward to the disputer.

Looking ahead, Tim proposed the possibility of adding a a non-existence prover to streamline the dispute process. This approach would trade off the current design's round of interactivity and communication for a higher cost. By utilizing a non-existence prover, the disputer could directly prove that a block proposer has broken their promise by demonstrating that the transaction hash is absent from the targeted block. This optimization would reduce the reliance on block proposer to actively participate in the dispute resolution process, thereby enhancing the efficiency and trustlessness of the slashing mechanism. Similar to the inclusion proof, the non-existence proof would also be generated off-chain by the disputer and submitted to the slashing smart contract for verification.

Preconf-Share

The preconf-share

component serves as an API endpoint for users to send and receive requests and for block proposers to send promises with extremely low latency and cost. Additionally, preconf-share

acts as an intermediary between users and block proposers, shielding IP addresses on both sides. The features inherited from MEV-Share, such as hints via privacy settings and inclusion conditions, help control the principal-agent problem (PAP), where a block proposer could unbundle the request to include only the tip transaction while omitting the other transactions in the request. This is achieved through the following measures:

1. When relaying user requests, only the transaction hash, hints, tip, and preferences are shared with the block proposers.
2. The signed raw transactions are shared only with the block proposer that provided the best preconfirmation.

preconf_sendRequest

preconf_sendRequest

allows users to send a preconfirmation request for signed transactions with privacy settings and inclusion conditions.

{ inclusion: { desiredBlock: string, // hex-encoded number maxBlock?: string, // hex-encoded number tip: string, // hex-encoded number (wei) }, body: Array< { tx: string } >, privacy?: { hints?: Array< "calldata" | "contract_address" | "logs" | "function_selector" | "hash" | "tx_hash" >, operators?: Array, } }

- inclusion

: Parameters used by preconf-share

node to rank and filter preconfirmation promises. * desiredBlock

: The best block to include this request.

- maxBlock

: The maximum block height in which this request may

be included.

- tip

: The tip to pay the block proposer, denominated in wei, which must match the tip transaction in the request.

- desiredBlock

: The best block to include this request.

- maxBlock

: The maximum block height in which this request may

be included.

- tip

: The tip to pay the block proposer, denominated in wei, which must match the tip transaction in the request.

- body

: An array of signed transactions. It must contain the tip transaction and match the tip

amount in the inclusion

parameters.

- privacy

: Preferences on what data should be shared to block proposers about the request. * hints

: Each item additively specifies which data about all transactions in the request to share. If no hints are specified, no transaction data is shared.

- operators

: Addresses of block proposers that have permission to receive this request and provide a promise.

- hints

: Each item additively specifies which data about all transactions in the request to share. If no hints are specified, no transaction data is shared.

- operators

: Addresses of block proposers that have permission to receive this request and provide a promise.

preconf_confirmRequest

preconf_confirmRequest

allows block proposers to send a signed preconfirmation promise.

{ preconf: { request: PreconfSendRequestParams, block: string }, signature: string }

- preconf

: The content of the preconfirmation, which is composed of the user request received from the stream and the block number where the block proposer will include the request transactions. * request

: The request received from the preconf-share

stream.

- maxBlock

: The block where the block proposer will include the request transactions. It is verified by preconf-share

that the block proposer is both an operator of the AVS and the proposer of that block.

- request

: The request received from the preconf-share

stream.

- maxBlock

: The block where the block proposer will include the request transactions. It is verified by preconf-share

that the block proposer is both an operator of the AVS and the proposer of that block.

- signature

: The preconf

parameter signed with the block proposer's (operator) private key.

Transaction flow

[

swimlanes-f8dc5a95d3307a60e4fff4dc4dfb1bbf

1300×1011 71.3 KB

](https://ethresear.ch/uploads/default/original/2X/2/2311d83e60a2dce3bc197a7fe47e2680ae7167ea.png)

[

preconf-post-final

4249×1908 588 KB

](https://ethresear.ch/uploads/default/original/2X/a/a00cc30ec982528b24e8cab451abde471e7f6b36.jpeg)

Security

The economic security of this protocol relies on the AVS implementation using EigenLayer. For the guarantees to be considered secure, the cost of corruption must exceed the profit from corruption, creating a deterrent against malicious behavior. Under the current Ethereum design, stakers can have a maximum of 50% (16 ETH) of their total staked ETH slashed. Consequently, block proposers are incentivized to violate preconfirmation promises if:

- The sum of tips in the request is more than 16 ETH. In this case, the block proposer can unbundle the request and only include the tip transactions in a block, profiting from the tips while avoiding the execution of the other transactions.
- The sum of profits from extracting value using the request transactions is more than 16 ETH. For example, a block proposer may choose to unbundle a request containing a white-hat funds rescue and tip, and only include the tip while replacing the white-hat rescue with one done by themselves (essentially keeping both the rescue funds and tip), leading to partial inclusion.

It's important to note that slashed ETH can potentially be used by the AVS to compensate parties that were wronged, such as the affected users in these scenarios. However, external factors like gas costs, complexity, and opportunity costs also influence the cost of corruption.

In the context of MEV, the 16 ETH slashing penalty may not always be sufficient to economically secure transactions, a problem similar to that which occurs under PBS implementations. According to data from libMEV for the period from February 23 to March 24, five bundles yielded profits exceeding 16 ETH for searchers: 22.33 ETH, 32.53 ETH, 59.19 ETH, and 124.36 ETH. Nevertheless, it's worth noting that out of the 183,440 bundles that landed on-chain during this period, the average profit was $12.53, out of the total value. This suggests that the vast majority of transactions can be economically secured by the 16 ETH slashing penalty.

To enhance the security of individual preconfirmation promises, three potential approaches can be explored:

1. Implementing an attestation system that provides additional assurances beyond the slashing mechanism.

2. An idea from @diego is to extend the AVS or EigenLayer smart contracts to allow block proposers to stake additional ETH, increasing the potential cost of violating a preconfirmation promise.

3. As noted by @ed, implementing preconfirmations using enshrined execution tickets, which aims to solve the economic security problem as well.

Looking Ahead

Having developed and tested the protocol on local networks, the natural next step is to deploy it on a testnet environment. This will provide valuable insights into how the various components and dynamics of the system play out in a more realistic setting.

One key area to explore is the interaction between preconfirmations and mev-boost. In the current implementation, block proposers act as both block builders and proposers. However, integrating preconfirmations with mev-boost could potentially expand the pool of block proposers and improve the efficiency of the system. To achieve compatibility with mev-boost, a communication channel would need to be established, enabling block proposers to share their preconfirmation promises with builders, who can then include them in the blocks they are constructing.

It's important to note that this approach introduces a trust assumption, as block proposers would need to rely on builders to include their promises in the constructed blocks, whereas inclusion lists offer a trustless alternative. Another potential solution to mitigate this trust assumption is to leverage builders running under Trusted Execution Environments (TEEs), such as those in SUAVE. preconf-share

nodes are also trusted to not publish signed preconfirmations that were not chosen, so it should eventually be decentralized (akin to mev-boost relays).

An additional aspect to consider is the complexity of being a block proposer. For many EigenLayer AVS users, the sophistication required to effectively operate as a block proposer may be quite high. Builders, on the other hand, are well-positioned to take on this role, as they already possess the necessary expertise and infrastructure for block building and have developed in-house algorithms for optimizing transaction inclusion. As a result, less sophisticated block proposers may choose to offload the preconfirmation responsibilities to specialized parties.

To further enhance the protocol and explore its potential, several areas of research and development can be pursued:

1. Investigating the economic incentives and game-theoretic aspects of the AVS to ensure a stable and secure system.

2. Developing more advanced algorithms for block proposers to optimize transaction inclusion and maximize their rewards.

3. Exploring potential interactions with mev-boost, MEV-Share and in-block conditions (like preconfirmations for top-of-block).

References

- Based preconfirmations by Justin Drake
- Analyzing BFT & Proposer-Promised Preconfirmations by Ellie Davidson, et al.
- MEV-Share by Flashbots

Check out the implementation on GitHub