Purpose

This document presents the integration of the Credible Layer[1]

, a novel crypto-economic protocol which enables dapps to enforce their custom invariants on every transaction that interacts with them. The discussion around the transaction supply chain and relevant guarantees from parties have been so far restricted to MEV and related economic subjects. We aim to kickstart the discussion around how the same parties can also provide security guarantees, in a way that is both powerful, as also sovereign and credibly neutral.

Currently, we only have released our whitepaper, but we plan to make the totality of our code and specs open-source as we move closer to an initial integration with a few networks.

Mechanism Overview

Credible Blocks introduces a novel security approach based on Assertions

- computations about on-chain state expressed in EVM bytecode and executed off-chain. Think of assertions as “security intents” - they define what states MUST NOT be reached for a particular dApp. We posit that this is explicitly easier for developers to do, as they don’t have to know how

a vulnerability is exploited (e.g bug in the code, leaked private key, etc.), but only the what

(e.g that a contract should not be upgraded in the middle of the night).

The system operates through two key markets:

1. State Oracle Market
2. DApps submit assertions which are associated with their contracts
3. Anyone can provide proofs about invalidation based on current or historical state of the chain (Proof of Realization), or possible invalidations (Proof of Possibility)
4. Dapps can define EVM calldata to be executed upon the submission of a Proof of Possibility – trust-minimized mitigation
5. Assertion Enforcement Market
6. Block builders/sequencers enforce assertions
7. DApps pay fees for protection
8. Enforcers face slashing for violations

Key Participants

- Assertion Adopters

: Smart contracts using assertions

- Assertion Submitters

: Security researchers/protocol engineers

- Assertion Enforcers

: Block builders/proposers

- Assertion Consumers

: Any entity verifying assertions

[

image

2237×1583 171 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/25a14b0cc25f08526bad814a19039557be24ae17.png)

This mechanism creates a robust security layer where incentives align between those needing protection and those

providing it, enabling proactive vulnerability detection and automated response mechanisms.

Assertion Enforcers vouch that the transactions which they are responsible for adding to the network will not invalidate any assertions. If a transaction lands and puts a dApp into a state where any of it's assertions are invalidated, anyone (Assertion Consumer) can submit a Proof of Realization and slash the Assertion Enforcer.

Assertion submission can be from the dApp itself, or 3rd parties (Assertion Submitters) who are given incentives to provide assertions and thus contribute to the security of dApps.

Assertion Execution

Assertions are expressed as EVM bytecode, which is evaluated off chain in PhEVM and can leverage precompiles (cheatcodes) to switch between forks of the state, in a devex similar to that of Foundry.

An Assertion Enforcer has, essentially, a new STF when deciding if a transaction should be added to the network.

[

image

3403×2023 209 KB

](https://collective.flashbots.net/uploads/default/original/2X/d/d3dc7ffa4e5503457fbef1f0aa92b52de392f78e.png)

In a way, every transaction now has new validity rules which change according to what contracts the transaction interacts with (or to be more precise, what contracts it touches).

This is crucial for two reasons:

1. Every Dapp remains sovereign, as they define what is acceptable and what is not. The system makes no assumptions on it's own.
2. It's extremely performant and has minimum overhead in the block building process, as the execution of assertions is bounded by which contracts the transaction interacts with and whether they are enrolled in the protocol.

[

image

5994×1305 400 KB

](https://collective.flashbots.net/uploads/default/original/2X/0/095e29643e0fde7b2924526853f1ac2a9971c43b.png)

OP-Stack Integration

The OP-Stack integration makes use of the push by Flashbots to move towards a future where block proposal and building are separated in this majestic monolith we call the Sequencer. This enables networks to make use of the protocol and integrate, without changing the Sequencer code

. We believe that the Sequencer is the most critical infrastructure of every network, and thus any change to that was considerably risks. With PBS, the block builder can play the role of the Assertion Enforcer, while containing risk considerably.

Sequencer Flow as an Assertion Enforcer

[

image

1920×1890 102 KB

](https://collective.flashbots.net/uploads/default/original/2X/1/123c8e481628400badf3ec12e23525f3c355f8a5.jpeg)

- The Block Builder keeps track of the active assertions by communicating with the assertion-loader

. Apart from the communication with the assertion-loader

, it otherwise behaves identically to the block builder described in the specs.

- The Block Builder, after building a block, will execute all related assertions for every transaction that is included in the block in the PhEVM

. If a transaction is found to violate an assertion, it is removed and all the transactions after that are re-applied to the state and re-executed in the PhEVM

.

- A number of optimizations have been made so that the assertion execution adds minimal latency to the system. After the initial implementation, numerous additional improvements are slated to be made, such as compiling assertions to machine code (AOE compilation)

Architecture Overview

Originally, we built the Assertion Enforcer integration leveraging the Builder API spec and building our own rbuilder-based builder. But we plan to converting it to Rollup-Boost, as the community seems to be aligning on that design path for now.

Essentially, an OP-Stack rollup that implements the builder or uses Rollup-Boost can run op-talos, which is a builder identical to the default implementation rbuilder [2]

but with the following differences:

1. It uses reth as the execution client
2. It uses an ETL pipeline to consume MXDB directly and transform it into a data structure that we have in-memory
3. It uses sled as a backup database to checkpoint the in-memory database

Initial benchmarks have shown that the execution of assertions with a naive sequential execution approach adds an overhead of about 50ms to the block building process. The benchmark has the following parameters:

- 1900 txs of about 70M total gas (2x average block size for Base)
- 900 assertions
- 599 malicious transactions (transactions which invalidate assertions)

We expect this to be substantially lower as we implement aggressive parallelization where assertions are executed in parallel as the block is built (optimistic block building).

[

Untitled diagram-2024-11-13-051607

3840×3740 423 KB

](https://collective.flashbots.net/uploads/default/original/2X/d/d7bced2b6b2fab2df8a8f99c71fc83dbe106b109.png)

Future Work

1. Explore TEE integration and how that can affect the mechanism outlined here and in the whitepaper. Essentially, how does the mechanism changes if we can verify the honest operation of the Assertion Enforcer as a stronger guarantee.
2. Open-Source the specs and code outlined here
3. Release OP-Talos, which is a block builder meant to be used with rollup-boost and enable OP-Stack rollups to integrate with the Credible Layer and act as Assertion Enforcers for any Dapp deployed on the rollup
4. Share more information around Fee structure (for assertion execution), slashing, and the rest of the crypto-economic mechanisms we explore in the whitepaper

References

1. Credible Layer Whitepaper ↩
2. GitHub - flashbots/rbuilder: rbuilder is a blazingly fast, state of the art Ethereum MEV-Boost block builder written in Rust ↩