EIPs-6404 and 6466

An Impact Study on the Transition to SSZ for Existing Contracts Relying on Proofs of RLP-Encoded Data

June 14, 2023

Abstract

This study aims to examine the potential impact of Ethereum Improvement Proposals (EIPs) 6404 and 6466, which propose the modification of Merkle-Patricia Trie (MPT) commitments for transactions and receipts, respectively. Importantly, this entails a change in the serialization algorithm, from Recursive Length Prefix (RLP) format to the Simple Serialize (SSZ) format for the Receipts and Transactions containers. In turn, this changes the *Receipts Root* and *Transactions Root* fields in the execution layer headers.

A primary concern is that this transition could disrupt contracts that rely on RLP for proofs on data committed to the Ethereum mainnet. These contracts may include critical parts of decentralized bridges, which generate proofs about some log that was emitted in historical transactions.

This research seeks to quantify and qualify the extent of potential disruption caused by these changes. Identifying the specific on-chain patterns that verify commitments in this manner represents a significant challenge, necessitating a semi-automated examination of all smart contracts deployed on the Ethereum network, together with their recent behavior. The study also attempts to identify which projects these contracts are part of, and whether actions can be taken, on-chain (such as upgrading) or off-chain (such as modifying their respective oracles) to limit the impact of these changes.

Executive Summary

For the proposed EIPs, we were able to measure the extent of the impact of these changes. The effects are observed on a handful of known projects, all of which are cross-chain bridges.

Project Name	**Website**	**Estimated Impact**
zkBridge	https://zkbridge.com	Moderate
LayerZero	https://layerzero.network/	Moderate
Telepathy	https://docs.telepathy.xyz/	Moderate

Notably, many other protocols that do employ RLP functionality are *not* affected. For instance the Optimism and Polygon bridges use RLP operations for inclusion proofs when bridging from L2 networks back to Ethereum, and, thus, are not affected by the Ethereum encoding of transactions.

Finally, an interesting result of our study is that out of the two proposed EIPs, only EIP-6466 (Receipts Root EIP) was observed to have an impact on the inspected protocols. This makes sense as log-inclusion proofs are probably the most common way to conduct cross-chain message passing.

The language used in the rest of the study assumes the reader is already familiar with the EIPs, Merkle-Patricia Trie Commitments, and RLP encoding.

Scope of study

EIP-6404, titled “SSZ Transactions Root,” and EIP-6466, titled “SSZ Receipts Root,” aim to allow the execution layer of the Ethereum protocol to migrate from the existing RLP-encoded Merkle-Patricia Trie (MPT) commitments to SSZ-encoded MPT commitments. Both proposals aim to make data verification easier for light clients by adopting the SSZ format, demonstrating the Ethereum community’s commitment to

ensuring that the chain remains accessible to users without high-end hardware. Since the Ethereum merge, SSZ is already being used in the consensus layer. The changes of EIP-6404 and EIP-6466 will make the computed commitments (for transactions and receipts) consistent with the corresponding fields in the consensus layer.

The assessment presented in this document, although neither fully sound nor complete, largely addresses the questions posed by the Ethereum team, as agreed upon by the respective statement of work. The primary concern underpinning this study is the potential disruption to contracts on Ethereum mainnet that rely on proofs of RLP-committed information to the chain, as a result of the proposed migration to Simple Serialize (SSZ). Examples of such contracts include decentralized bridges that generate proofs about logs that were emitted in historical transactions. This study aims to ascertain the extent of potential breakage that could be caused by this change. The study also seeks to determine the magnitude of these contracts, their upgradeability and their usage in terms of transaction volume and balance.

The study does not consider all non-Ethereum, EVM-compatible chains (e.g., BSC) that read process RLP-encoded data derived from Ethereum mainnet, however a small sample of contracts on external chains (BSC and Fantom) were inspected.

Experimental Findings and Study

This section describes the experiments conducted during this impact study. The methodology utilized static program analysis (looking for specific patterns in the contract code), and, in order to cross validate the approaches, dynamic program analysis on past transactions. In addition, we have inspected smart contracts or debugged past transactions to better understand the impact of the EIPs being studied.

Identifying Candidates via Static Analysis

The goal of this task is to identify protocols that have recently interacted with RLP-like contracts. By RLP-like contracts we mean contracts that appear to have logic for decoding RLP data. In order to identify such contracts, we take advantage of the fact that RLP decoding involves certain constants and comparisons.

More specifically, a typical RLP library implementation contains a set of standard comparisons of program variables with the constants 0x80, 0xb8, 0xc0 and 0xf8. These comparisons are part of the logic for decoding RLP data as explained in the RLP specification. Due to the stack-based nature of the EVM, such a query cannot be performed directly on the bytecode, as the operands of the inequality operations we're looking for haven't been resolved yet. For this reason, we ran this query against the TAC (Three Address Code) representation that our decompiler outputs. In this highly-normalized output, the EVM execution stack has been resolved to instructions of the form `x := y + z`, and, more importantly for our application, has information about constant operands.

As we've decompiled and stored the vast majority of Ethereum contracts in our database, we can detect these with a simple SQL query (with some syntactic abstractions to make it easier to read):

```
select md5_bytecode
from decompiled_code
where
    decompiled_code.tac_level similar to  [variable] = [variable] < 0x80 and
    decompiled_code.tac_level similar to  [variable] = [variable] < 0xb8 and
    decompiled_code.tac_level similar to  [variable] = [variable] < 0xc0 and
    decompiled_code.tac_level similar to  [variable] = [variable] < 0xf8
```

In the actual query we also looked for the "symmetric" comparison (e.g., `[variable] > 0x80`) for completeness, which we didn't include above for brevity. In addition, for each of the 4 constants, we also checked for comparisons of their off-by-one values. (Their negation is used to produce greater-/less-than-or-equal comparisons.)

This yielded a result set of 556 unique contract bytecodes. In the subsequent query, we'll refer to this set as `rlp_comparison_checked`.

With this dataset, we can filter the contracts that have been interacted with in recent transactions (have been transacted with since block 16600000). To assist with our inspections, we also record some sample top-level transaction entry points (aka the first contract that is invoked by an EOA), as well as some sample transactions:

```
with rlp_addrs as (
    select address
    from rlp_comparison_checked join contracts using (md5_bytecode)
)
select
    rlp_tx.to_a,
    count(distinct (tl.block_number, tl.transaction_index)) as tx_cnt,
    (
        array_agg(distinct tx_hash(tl.block_number, tl.transaction_index))
    )[:10] as sample_rlp_txs,
    array_agg(distinct tl.to_a) as sample_rlp_tx_entrypoints
from
    transaction_detail tl join transaction_detail rlp_tx on
        (tl.block_number, tl.transaction_index, tl.vmstep_start) =
        (rlp_tx.block_number, rlp_tx.transaction_index, 0) join
    rlp_addrs ra on rlp_tx.to_a = ra.address
where rlp_tx.block_number >= 16600000
group by rlp_tx.to_a
```

```
order by count(distinct (tl.block_number, tl.transaction_index)) desc
```

This query yielded 156 RLP contract addresses. Out of these we inspected those with at least 100 recent transactions, for a final set of 68 contracts to be inspected.

Identifying Candidates Via Dynamic Analysis

To evaluate the possible breakage caused by the proposed changes in Ethereum's transaction and receipt encoding, a dynamic analysis of recent Ethereum transactions' calldata was performed. The primary objective was to determine if this data contained RLP-encoded information, likely stemming from EVM-compatible chains.

It is first worth noting that RLP-encoded data is usually passed as an argument of type `bytes` in Solidity. Arguments of type `bytes` in Ethereum are ABI-encoded in two parts: (i) a 32-byte length field (see Fig 1. below, word 'x'), which indicates the number of bytes in the data, and, (ii) the actual byte data, which is padded on the right with zeroes to the nearest multiple of 32 bytes (see Fig 1. below, word 'y').

The analysis relied on a heuristic algorithm designed to identify whether calldata is RLP-encoded. This algorithm, implemented in the Rust language as a PostgreSQL stored procedure, operates on the basis of how RLP-encoded data is structured. In RLP encoding, the prefix of the encoded data indicates its type and length. For instance, a prefix greater than or equal to 248 is used for lists longer than 55 bytes.

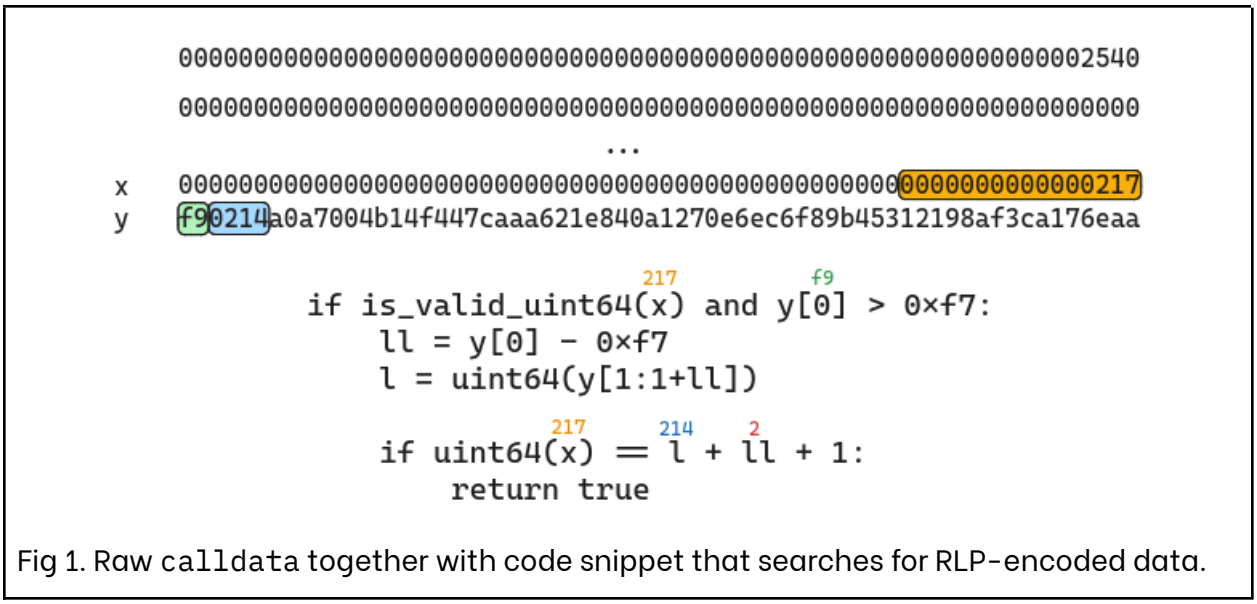

Fig 1. Raw `calldata` together with code snippet that searches for RLP-encoded data.

The algorithm begins by checking if the data length is greater than a fixed number of bytes. If not, it returns false, as it's not possible to encode transactions or receipts in such short data. For longer data, the function iterates over it, inspecting blocks of 32 bytes at a time (to account for Ethereum's 32-byte word structure). At each step it checks using the code in Fig 1. for an RLP list prefix in the first byte of the current block (green portion of y). If found, it calculates the length of the list using the prefix and the following bytes (blue portion of y), as specified by the RLP encoding rules. It then verifies if this calculated length matches the expected length based on the ABI-encoded calldata structure (orange portion of x). If a match is found, the function concludes that the calldata is RLP-encoded and returns true. Otherwise, it continues to the next 32-byte block until it has traversed all the data.

This heuristic algorithm is effective due to the fact that RLP-encoded data is further ABI-encoded according to Ethereum's ABI format. This affords two unambiguous "length" fields, which, after accounting for the RLP prefix and length field itself, *should be exactly equal* if the data is indeed RLP encoded. In other words, the simple algorithm leverages the known structure of RLP-encoded data and Ethereum's ABI-encoded calldata to make an educated guess about the presence of RLP-encoded data. However, as with any heuristic, it may not catch all cases. It will however catch cases in which RLP-encoded information is nested within other structures, as long as this is also ABI-encoded.

Note that the dynamic analysis returned around 1000 possible contracts over a million blocks, however for our inspections we picked the most recent popular 100 of these contracts.

Inspection of Candidate Contracts

During our inspection we found that most **un**affected protocols can be categorized as follows:

1. The protocol doesn't perform MPT proofs over RLP-encoded data (analysis false positive)
2. The protocol performs receipts/transaction root MPT inclusion proofs, but only for other chain data (L2s, side-chains).
3. The protocol doesn't perform receipts/transactions root MPT inclusion proofs, or the proofs are done against MPT commitments that are unrelated to Ethereum block header fields (e.g., custom structures, state root) or has some entirely different proof mechanism over RLP-encoded data.

Our inspection results can be summarized in the following table of "core" protocols that we encountered:

Protocol	**Inspection Notes**
Polygon Bridge	Not affected (2) – RLP operations are required to do inclusion proofs when bridging from Polygon back to Ethereum.
Optimism Bridge	Not affected (2) – similar to Polygon, RLP is used only for the L2-to-L1 direction, so changes to the Ethereum protocol shouldn't affect it.
BitTorrent Chain	Not affected (2) – similar in function to Polygon. RLP only involved for L2-to-L1 proofs.
Socket	Not directly affected – this is a meta-bridge, meaning that it uses other bridge solutions under the hood.

Protocol	Inspection Notes
Boba	Not affected (2) – similar in function to Optimism.
Metis	Not affected (2) – similar in function to Optimism.
1inch	Not affected (1) – dynamic analysis false positive.
Hop Protocol	Not affected (3) – inclusion proofs are done on structures that are unrelated to Ethereum's headers
Connext	Not affected (3) - MPT commitments appear to be on structures that are unrelated to Ethereum's headers
Uniswap	Not affected (1) – dynamic analysis false positive.
Seaport	Not affected (1) – dynamic analysis false positive.
Ankr	Not affected (3) – inclusion proofs are done via a centralized, signing mechanism (Ankr backend signs the receipt).
Relic	Not affected (3) – it performs account state inclusion proofs for the state root, which is unaffected by the EIPs.
Orbiter	Not affected (1) – dynamic analysis false positive.
0x	Not affected (1) – dynamic analysis false positive.
LayerZero	**Affected** – the current default configuration uses MPT inclusion proofs for the receipts root. A sample transaction can be found here.
zkBridge	**Affected** – the protocol is integrated into LayerZero as an Oracle that users can configure in LayerZero's UltraLightNode. The oracle posts block data and receipt roots on-chain that use the current MPT commitment scheme, leading to a similar problem as LayerZero, as applications need to use similar RLP in their inclusion proof.
Telepathy	**Affected** – despite their use of consensus-layers headers, the log-inclusion proof logic utilizes the historical summaries buffer, and specifically the block root which is a function of the receipts root.

Protocol	Inspection Notes
	As receipts roots are currently MPT commitments, the code uses RLP logic during the inclusion proof, similar to the two other affected protocols.

Impact Opinion

As elucidated in previous sections, the impact of the two proposed EIPs is moderate, affecting a handful of "core" bridge protocols and possibly client applications that leverage them for their cross-chain logic. Interestingly, while we found that EIP-6466 (SSZ Receipts Root) will have some impact, no evidence of potential impact was found for the accompanying EIP-6404.

Upgradeability of affected protocols. While the implementation of EIP-6466 will affect some protocols, there are upgradeability paths that can be taken to address the issues:

1. LayerZero: The default inclusion proof library can be updated to reflect the new SSZ commitment for the receipts root.
2. zkBridge: zkBridge can either wait for LayerZero to address this, or implement their own proof validator and prompt client applications to update their LayerZero configuration.
3. Telepathy: Upgrading the inclusion proof logic should suffice. Their router contract is using the UUPS proxy pattern which, together with the fact that they also control the oracle logic, should make the upgrade process relatively painless. It's worth noting that while log proofs are affected by the EIP, Telepathy also supports storage proofs which utilize state roots. This means that the protocol can still theoretically function post EIP, but with much more expensive cross-chain gas fees.

Another upgradeability path could be to modify the oracles so that they continue to publish receiptsHash commitments using the current MPT scheme. However this might be more complicated and will introduce technical debt for the projects.

It should also be noted that the LayerZero case is a bit more complicated—due to its highly-configurable nature (applications can have their own relayer, oracles and validation library), any protocols that have a custom configuration might also need to perform some form of update to their on-chain transaction validation logic and even to their oracles.

Conclusion

We performed a comprehensive study on the impact that the introduction of EIP-6404 and 6466 might have on the smart contract ecosystem on Ethereum. We observe that although there are a few affected protocols, they are moderately affected, mainly due to the fact that upgradeability paths for these protocols exist.

Furthermore, all of the observed impact appears to only be caused by EIP-6466 (SSZ *Receipts Root* EIP). In other words, out of all the contracts, transactions, and protocols we observed, no evidence of reliance on the *Transactions Root* was found.

Consequently, we categorize the (disruption) impact of EIP-6466 as “moderate but manageable” and the impact of EIP-6404 as “insignificant - low”.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent Web3 protocols. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.