OpenZeppelin | security

OpenZeppelin Contracts v5.6 Audit

February 27, 2026

Table of Contents

Table of Contents 2

Summary 4

Scope 5

System Overview 9

Bug Fixes and Standards Compliance 9

Breaking Changes and Deprecations 9

Cross-Chain Additions 10

Cryptography, Signers, and Low-Level Utilities 10

Security Model and Trust Assumptions 11

Medium Severity 12

M-01 Memory Aliasing for Single-Byte Inputs 12

M-02 receiveId Check Causes Accidental DoS/Incompatibility With Empty receiveId Gateways 12

Low Severity 13

L-01 Unexpected Revert in WebAuthn._validateChallenge 13

L-02 Bypassing WebAuthn-Specific Validations in SignerWebAuthn._rawSignatureValidation 13

L-03 Possible Permanent Message Loss in ERC7786Recipient._processMessage 14

L-04 Missing Bounds Check in Accumulators.flatten 14

L-05 Incomplete Documentation 15

L-06 tryParseV1 Does Not Reject Input With Both Empty chainReference and Address, Violating ERC-7930 16

L-07 Incorrect Documentation 17

L-08 Inaccurate Error Message When Traversing Empty Branch Slot 19

L-09 TrieProof Rejects Valid Merkle-Patricia Proofs With Inline Extension Leaf Nodes 19

L-10 Incorrect Lower-bound Validity Check in getValidationData 20

Notes & Additional Information 21

N-01 Missing Bound Check for Long-Form Length-of-Length 21

N-02 Redundant item.length() Call in RLP Encoding 22

N-03 escapeJSON Does Not Escape Raw Control Characters Like NULL Byte 22

N-04 Inconsistent Use of Decimal vs Hexadecimal Notation for Slice Offsets 23

N-05 setFreeMemoryPointer Should Follow the Unsafe Naming Convention to Reflect Potential Memory Corruption 23

Client Reported 24

CR-01 uint8 Wraparound In tryParseV1 And tryParseV1Calldata Causes Incorrect Slice Bounds 24

Conclusion 26

Appendix 27
Issue Classification 27

Summary

Type	Library
Timeline	From 2026-01-26 To 2026-02-05
Languages	Solidity

Total Issues	18 (14 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	10 (7 resolved, 2 partially resolved)
Notes & Additional Information	5 (4 resolved)

Scope

OpenZeppelin performed a diff audit of the OpenZeppelin Contracts library between release v5.4.0 at commit c64a1ed (tag `v5.4.0`) and release candidate v5.6.0-rc.1 at commit 68e4095 (tag `v5.6.0-rc.1`).

In scope were the following files:

```
contracts/
├── access/
│   ├── AccessControl.sol
│   ├── extensions/
│   │   ├── AccessControlDefaultAdminRules.sol
│   │   ├── AccessControlEnumerable.sol
│   │   └── IAccessControlDefaultAdminRules.sol
│   └── manager/
│       ├── AccessManager.sol
│       └── IAccessManager.sol
├── account/
│   ├── Account.sol
│   ├── extensions/
│   │   ├── draft-AccountERC7579.sol
│   │   ├── draft-AccountERC7579Hooked.sol
│   │   └── draft-ERC7821.sol
│   └── utils/
│       ├── EIP7702Utils.sol
│       ├── draft-ERC4337Utils.sol
│       └── draft-ERC7579Utils.sol
├── crosschain/
│   ├── CrosschainLinked.sol
│   ├── ERC7786Recipient.sol
│   └── bridges/
│       ├── BridgeERC20.sol
│       ├── BridgeERC20Core.sol
│       └── BridgeERC7802.sol
├── finance/
│   └── VestingWallet.sol
├── governance/
│   ├── Governor.sol
│   ├── IGovernor.sol
│   ├── TimelockController.sol
│   ├── extensions/
│   │   ├── GovernorProposalGuardian.sol
│   │   ├── GovernorStorage.sol
│   │   ├── GovernorSuperQuorum.sol
│   │   ├── GovernorTimelockCompound.sol
│   │   ├── GovernorTimelockControl.sol
│   │   ├── GovernorVotesQuorumFraction.sol
```

```
│   │   └── GovernorVotesSuperQuorumFraction.sol
│   └── utils/
│       ├── IVotes.sol
│       ├── Votes.sol
│       └── VotesExtended.sol
├── interfaces/
│   ├── IERC3156FlashLender.sol
│   ├── IERC4626.sol
│   ├── IERC6909.sol
│   ├── IERC7751.sol
│   ├── draft-IERC4337.sol
│   ├── draft-IERC6093.sol
│   ├── draft-IERC7579.sol
│   ├── draft-IERC7786.sol
│   └── draft-IERC7802.sol
├── metatx/
│   ├── ERC2771Context.sol
│   └── ERC2771Forwarder.sol
├── proxy/
│   ├── Clones.sol
│   ├── Proxy.sol
│   ├── ERC1967/
│   │   ├── ERC1967Proxy.sol
│   │   └── ERC1967Utils.sol
│   ├── transparent/
│   │   └── TransparentUpgradeableProxy.sol
│   └── utils/
│       └── UUPSUpgradeable.sol
├── token/
│   ├── ERC20/
│   │   ├── ERC20.sol
│   │   ├── extensions/
│   │   │   ├── ERC20Crosschain.sol
│   │   │   ├── ERC20FlashMint.sol
│   │   │   ├── ERC20Permit.sol
│   │   │   ├── ERC20Votes.sol
│   │   │   ├── ERC20Wrapper.sol
│   │   │   ├── ERC4626.sol
│   │   │   └── IERC20Permit.sol
│   │   └── utils/
│   │       └── SafeERC20.sol
│   ├── ERC721/
│   │   ├── ERC721.sol
│   │   ├── extensions/
│   │   │   ├── ERC721Burnable.sol
│   │   │   ├── ERC721Consecutive.sol
│   │   │   ├── ERC721Enumerable.sol
│   │   │   ├── ERC721Pausable.sol
│   │   │   ├── ERC721Royalty.sol
│   │   │   ├── ERC721URIStorage.sol
│   │   │   ├── ERC721Votes.sol
│   │   │   └── ERC721Wrapper.sol
│   │   └── utils/
│   │       ├── ERC721Holder.sol
│   │       └── ERC721Utils.sol
│   ├── ERC1155/
```

```
│   │   ├── ERC1155.sol
│   │   ├── extensions/
│   │   │   ├── ERC1155Burnable.sol
│   │   │   ├── ERC1155Pausable.sol
│   │   │   ├── ERC1155Supply.sol
│   │   │   └── ERC1155URIStorage.sol
│   │   └── utils/
│   │       └── ERC1155Holder.sol
│   └── ERC6909/
│       ├── ERC6909.sol
│       └── extensions/
│           ├── ERC6909ContentURI.sol
│           ├── ERC6909Metadata.sol
│           └── ERC6909TokenSupply.sol
└── utils/
    ├── Address.sol
    ├── Arrays.sol
    ├── Base58.sol
    ├── Base64.sol
    ├── Blockhash.sol
    ├── Bytes.sol
    ├── Create2.sol
    ├── LowLevelCall.sol
    ├── Memory.sol
    ├── Multicall.sol
    ├── RLP.sol
    ├── ReentrancyGuard.sol
    ├── ReentrancyGuardTransient.sol
    ├── RelayedCall.sol
    ├── ShortStrings.sol
    ├── SlotDerivation.sol
    ├── Strings.sol
    ├── draft-InteroperableAddress.sol
    ├── cryptography/
    │   ├── ECDSA.sol
    │   ├── EIP712.sol
    │   ├── MerkleProof.sol
    │   ├── MessageHashUtils.sol
    │   ├── SignatureChecker.sol
    │   ├── TrieProof.sol
    │   ├── WebAuthn.sol
    │   ├── draft-ERC7739Utils.sol
    │   ├── signers/
    │   │   ├── MultiSignerERC7913.sol
    │   │   ├── SignerECDSA.sol
    │   │   ├── SignerEIP7702.sol
    │   │   ├── SignerWebAuthn.sol
    │   │   └── draft-ERC7739.sol
    │   └── verifiers/
    │       ├── ERC7913P256Verifier.sol
    │       ├── ERC7913RSAVerifier.sol
    │       └── ERC7913WebAuthnVerifier.sol
    ├── introspection/
    │   └── ERC165Checker.sol
    ├── math/
    │   ├── Math.sol
```

```
    │   └── SafeCast.sol
    ├── structs/
    │   ├── Accumulators.sol
    │   ├── Checkpoints.sol
    │   ├── CircularBuffer.sol
    │   ├── DoubleEndedQueue.sol
    │   ├── EnumerableMap.sol
    │   ├── EnumerableSet.sol
    │   ├── Heap.sol
    │   └── MerkleTree.sol
    └── types/
        └── Time.sol
```

System Overview

The diff from v5.4.0 to v5.6.0-rc.1 is a wide-ranging update spanning two release cycles (v5.5 and v5.6). It includes standards compliance fixes, several breaking API updates, new cryptographic and cross-chain primitives, and numerous gas optimizations. The changes smooth out rough edges in existing abstractions while adding building blocks for account systems, signature verification, trie proof verification, and message-based interoperability.

The changes can be grouped into the following categories:

Bug Fixes and Standards Compliance

This update includes targeted fixes that reduce unexpected reverts and improve standards-facing behavior. **ERC-7579 module introspection** now avoids reverting on malformed context in fallback modules and returns `false` instead, improving compliance and tooling compatibility. **ERC-165 checking** was corrected so `supportsERC165` returns `false` if a target reverts during the `supportsInterface(0xffffffff)` probe. **SignatureChecker** fixes an invalid length check. **Base64** fixes `InvalidBase64Char` argument encoding at certain positions. **ERC-4626** computes `maxWithdraw` using `maxRedeem` and `previewRedeem` so that changes to the preview functions affect the max functions.

Breaking Changes and Deprecations

Several changes affect overrides, imports, and runtime semantics:

- **ERC-1155** batch transfers with exactly one id/value no longer call `onERC1155Received`. Instead, `onERC1155BatchReceived` is called.
- **ERC-721 and ERC-1155** prevent setting an operator for `address(0)` (in ERC-721, this could lead to obfuscated mint permission).
- **ERC1967Proxy** and **TransparentUpgradeableProxy** mandate initialization during construction, reverting with `ERC1967ProxyUninitialized` otherwise.
- **Account** `_validateUserOp` now takes an explicit `signature` argument.
- **ERC-4337** support is bumped to **Entrypoint v0.9**, with paymaster signature parsing and timestamp/block number-based validity ranges.

- **ERC-7579** fallback module install/uninstall requires at least 4 bytes of (de)init data. Uninstall is forced when `onUninstall` reverts.
- **ECDSA** malleability protection has been partly deprecated.
- `Initializable` and `UUPSUpgradeable` are no longer transpiled.
- Various import path changes (ERC-6909 finalized, `SignerERC7702` renamed to `SignerEIP7702`) and a minimum pragma bump to 0.8.24.

Cross-Chain Additions

New cross-chain building blocks have been introduced: a generic **ERC-7786 recipient** base contract, draft **ERC-7786 interfaces**, and an **ERC-7930 interoperable address** library. On top of these, an ERC-7786-based cross-chain bridging infrastructure has been added:

- **`CrosschainLinked`**: A helper contract that maintains a registry of `Links` specifying the address of an ERC-7786 gateway and a counterpart on the target chain using the `InteroperableAddress` format.
- **`BridgeERC20Core`**: It extends `CrosschainLinked` to facilitate cross-chain ERC-20 transfers, using `virtual` functions `_onSend` and `_onReceive` to handle burning/minting or locking/releasing.
- **`BridgeERC20`**: A lock/release bridge implementation.
- **`BridgeERC7802`**: A cross-chain burn/mint bridge for ERC-7802 tokens.
- **`ERC20Crosschain`**: An ERC-20 extension that embeds an ERC-7786 based cross-chain bridge directly in the token contract.

Cryptography, Signers, and Low-Level Utilities

A substantial expansion has been made to cryptography and performance-oriented utilities. **WebAuthn/P256** support includes a WebAuthn verification library, **`SignerWebAuthn`** (which validates WebAuthn authentication assertions, falling back to P256 signature validation when WebAuthn validation fails), and **`ERC7913WebAuthnVerifier`** (a stateless ERC-7913 signature verifier supporting WebAuthn authentication assertions using P256 public keys). A **TrieProof** library for Ethereum Merkle Patricia Trie proof verification and an **RLP** library for encoding and decoding data in Ethereum's Recursive Length Prefix format are introduced.

Dynamic domain separator hash generation based on ERC-5267 has also been added. Several new low-level utilities have been introduced—**Memory** (slices/pointers), **Accumulators** (linked-list construction in memory with O(1) insertions and O(n) flattening), **LowLevelCall**, and **RelayedCall**—along with extensions to Bytes, Arrays, and existing data structures. Many of

these involve inline assembly and are sensitive to boundary conditions and encoding correctness.

Security Model and Trust Assumptions

Auditing libraries requires a shift in focus due to their composability within blockchain protocols. While the scope of an audit is typically limited to the code itself, the scope expands when it comes to libraries because of their potential internal and external integrations. Libraries act as foundational components for many protocols. This means that their security is influenced not just by their internal robustness, but also by how they are utilized by integrators. As a result, ensuring a library's security involves reviewing the code as well as anticipating its various use cases and integration scenarios.

In addition to the above, the complexity grows because, while a library must accommodate a wide range of potential use cases, the responsibility for secure implementation often falls on developers who integrate it into their projects. These developers must carefully review the security considerations when extending contracts from the library. A library's security risks can multiply depending on how well developers understand and utilize its contracts. Therefore, extra care is necessary to identify and address all potential threats, both direct and indirect, or to document them so that developers are fully aware of the associated security risks.

Medium Severity

M-01 Memory Aliasing for Single-Byte Inputs

The `encode` and `readList` functions return the input directly without copying when the input is a single byte with value less than 128. In this case, the returned bytes memory shares the same memory location as the input. Any subsequent modification to the input will also modify the **encoded** result, and vice versa.

Protocols using this library for security-sensitive operations such as transaction encoding, Merkle proofs, or cross-chain messaging may experience silent data corruption if the input is modified after encoding. The affected edge case covers common values including small nonces, chain IDs under 128, and boolean representations. Since no error is produced, such corruption would be difficult to diagnose in production.

Consider always returning a copy for single-byte inputs to ensure consistency with the rest of the function's behavior and prevent unexpected memory aliasing.

Update: *Resolved in pull request #6342 at commit c2e62b4. The team stated:*

> *The aliasing issue was fixed for `encode` and now single-byte inputs below 128 are no longer returned as-is. Instead, a fresh `bytes memory` is returned so the result is independent of the input.*
>
> *In the case of `readList` we didn't change it because it returns a `Memory.Slice[]` and we consider `Slices` to be read-only objects. Returning newly allocated bytes would change the API and semantics (copies instead of read) and isn't consistent with the current design. Instead, it was noted in both `readList` and `decodeList`. Callers that need independent data can copy from the slice themselves.*

M-02 `receiveId` Check Causes Accidental DoS/ Incompatibility With Empty `receiveId` Gateways

`ERC7786Recipient` implements replay protection keyed by (gateway, `receiveId`) using a bitmap. This means that if an authorized gateway delivers more than one message with

`receiveId == bytes32(0)`, the first message succeeds and all subsequent messages revert, regardless of sender/payload.

However, EIP-7786 allows `receiveId` to be "empty" (in addition to being unique per gateway). This creates an interoperability risk where a compliant gateway that uses "empty" IDs (or uses zero as a sentinel) can unintentionally DoS delivery to an otherwise compliant recipient. A gateway that repeatedly uses `receiveId == 0x0` can cause permanent message delivery failure after the first message.

Consider updating the `receiveId` check to allow the delivery of empty `receiveId` messages.

Update: *Resolved in pull request #6346 at commit 157b001.*

Low Severity

L-01 Unexpected Revert in `WebAuthn._validateChallenge`

`_validateChallenge` is expected to return `true`/`false`. However, it may revert instead if the addition before `slice()` overflows when `challengeIndex` is close to `max`. The noted discrepancy becomes an issue if the caller expects `false` on invalid input instead of a revert. For instance, `ERC7913WebAuthnVerifier.verify` is expected to return `(bytes4)`. However, it may revert if `WebAuthn.verify`, which calls `_validateChallenge`, reverts.

Consider ensuring consistent behavior by `_validateChallenge` and its callers on failure.

Update: *Resolved in pull request #6329.*

L-02 Bypassing `WebAuthn`-Specific Validations in `SignerWebAuthn._rawSignatureValidation`

In `SignerWebAuthn._rawSignatureValidation`, if `WebAuthn` verification returns `false`, the contract falls back to raw P256 signature verification as also mentioned in the comments. This allows for bypassing `WebAuthn`-specific validations such as UV. For instance, an attacker may have a valid P256 signature but no valid `WebAuthn`.

Consider keeping the `WebAuthn` and raw P256 validations independent. Alternatively, consider documenting the justification of this behavior.

Update: *Resolved in pull request #6337.*

L-03 Possible Permanent Message Loss in `ERC7786Recipient._processMessage`

In `ERC7786Recipient.receiveMessage`, the `received` flag is set before `_processMessage`. That is fine if `_processMessage` reverts on failure. However, if `_processMessage` does not revert on failure (e.g., silently exists without actually processing the message), then the message can be permanently lost (i.e., never processed).

Consider explicitly documenting that `_processMessage` should revert on failure.

Update: *Resolved in pull request #6330.*

L-04 Missing Bounds Check in `Accumulators.flatten`

`Accumulators.flatten` transforms all Accumulator entries into a single bytes buffer. However it does not perform any bounds check (i.e. any `data.asSlice` length in `push` and `shift` can be passed). This may cause `flatten` to throw an out of bounds (OoB) revert.

Consider imposing checks on the slices lengths to prevent an OoB revert or at least documenting this behavior.

Update: *Resolved in pull request #6302. The team stated:*

> *We're including a new `isReserved(Slice)` function that checks whether the `Slice` provided as argument starts and ends before the current free memory pointer. This function is now used in `push` and `shift` to validate bounds.*

L-05 Incomplete Documentation

Throughout the codebase, multiple opportunities for improving the documentation were identified.

- In `RelayedCall`, `getRelayer` uses the `push0` instruction, which is a new instruction. Consider documenting that the library should be used on chains that support `push0`.
- In `WebAuthn`, the following three checks are not implemented, since they are responsibility of the authenticator and do not have on-chain use cases: origin validation, RP ID hash validation, and signature counter. Consider stating this explicitly in the documentation.
- The `draft-InteroperableAddress` library's `tryParseV1` (and derived `tryParseEvmV1`) accepts inputs that contain a valid ERC-7930 v1 encoding followed by arbitrary trailing bytes. Consider documenting this behavior explicitly (i.e., that parsing does not enforce canonical length / does not reject extra bytes), since multiple distinct byte strings can decode to the same semantic target and callers may assume the input bytes are a unique/canonical identifier.
- In `RLP`, the `readAddress` function accepts inputs with length 1 or 21 bytes. However, `encode(address)` always produces 21 bytes regardless of the address value. This creates an asymmetry where `encode(readAddress(data))` may produce different bytes than the original data for non-canonical inputs. Consider adding documentation explaining the encoding/decoding asymmetry and the rationale for accepting non-canonical inputs during decoding.
- In `TrieProof`, the use of the word "prefix" here is correct, but it is too easy to be confused with the prefix that is prepended to the nibble sequence and determines even or odd length and extension versus branch node (as in this usage, which occurs 8 lines later).
- In the `Memory` library, the documentation does not explain what a `Slice` is: namely, that its leftmost 128 bits store the length and its rightmost 128 bits stores the offset (address in memory) measured in bytes.
- In `Accumulators`, the documentation does not explicitly mention that the free memory pointer after accumulation and flattening may not be 32-byte aligned. This is not necessarily a problem, but should be documented in case the developer requires 32-byte alignment.
- In `ERC1155`, since the requirements have been updated, the documentation listing them should also include the added requirement that `owner` cannot be the zero address.

Consider addressing all the noted instances of incomplete/improvable documentation.

Update: *Partially Resolved in pull request #6330. The team stated:*

> *We implemented the suggested documentation changes except in these cases:*
>
> *- **RLP:** The possibility that `encode(readAddress(data))` differs from the original bytes for non-canonical inputs is a consequence of accepting multiple encodings for the same value (as already documented for `readAddress` and `readUint256`). We consider round-trip byte equality to be an implicit property of any permissive encoder/decoder and have not added specific documentation for it.*
>
> *- **Memory:** We intentionally do not document how the `Slice` type is represented in memory (e.g. layout of length and offset). The library is designed to be used only via its public API, and we prefer that users do not depend on or manipulate those internals.*
>
> *- **Accumulators:** Solidity does not in general guarantee that the free memory pointer remains 32-byte aligned after operations. We do not treat the post `flatten` FMP as a special case of the Accumulators library and have not documented alignment in this context.*

L-06 `tryParseV1` Does Not Reject Input With Both Empty `chainReference` and Address, Violating ERC-7930

According to ERC-7930, an interoperable address MUST have at least one of `chainReference` or address be non-empty. The specification states: "AddressLength MUST NOT be zero if ChainReferenceLength is also zero."

The `formatV1` function correctly enforces this requirement by reverting with `InteroperableAddressEmptyReferenceAndAddress` when both fields are empty. However, `tryParseV1` and `tryParseV1Calldata` do not validate this constraint. They successfully parse and return `success=true` for input where both `chainReferenceLength` and `addrLength` are zero.

This creates an asymmetry where `formatV1` rejects empty reference combined with empty address, but `tryParseV1` accepts it as valid. A contract relying on `tryParseV1` returning `success=true` as proof of ERC-7930 compliance would incorrectly accept malformed input that violates the standard.

Consider adding validation in `tryParseV1` and `tryParseV1Calldata` after parsing both lengths to return failure when both are zero.

Update: *Resolved in pull request #6331.*

L-07 Incorrect Documentation

Throughout the codebase, multiple instances of incorrect documentation were identified:

- Wrong comment for `_trySupportsInterface`: "*`supported`: true if the call succeeded AND returned data indicating the interface is supported*". The `supported` value is independent of the `success` of the call. They are checked together by the caller.
- In the `Bytes` and `Arrays` libraries:
  - The documentation for `splice` in the `Bytes` and `Arrays` libraries states that it "replicates the behavior of JavaScript's `Array.splice`". However, the function's behavior is fundamentally different from JavaScript's `splice`. The current NatSpec describes the operation as: "*Moves the content of `array`, from `start` (included) to the end of `array` to the start of that array*". This description omits the fact that the array is shrunk. For instance, splicing the array `[A, B, C, D, E, F]` at `[C, D]` yields `[C, D]`, while one may wrongfully expect `[C, D, C, D, E, F]` based on the documentation. Consider updating the documentation to clearly describe that the function keeps a specified range and shrinks the array to that range. The effect can be more succinctly stated as "overwrites `array` with `array[start..end]`" or similar.
  - The comment "allocate and copy" within `Bytes.replace` and `Arrays.replace` is misleading because no memory allocation occurs in this function. Consider revising this comment to reflect that this function overwrites `buffer` using `mcopy`.
- In `TrieProof`:
  - The documentation refers to nodes as either large or small (see 1, 2), while the code refers to nodes as either large or *short*. Ideally, within the code, the nodes should be named either long/short or large/small.
  - This comment is misleading: in trie proof verification, a 32-byte nodeId is assumed to be a hash (nodes exactly 32 bytes in length are also hashed).
  - Lines 25 and 27 use AsciiDoc-style link syntax (`URL[link text]`) which NatSpec does not support. The `[link text]` suffix won't render as a clickable link and will display literally as part of the URL. Consider using plain URLs instead.
- In `LowLevelCall`, this comment states that `staticcallReturn64Bytes` is useful for functions that return a tuple of single-word values. However, the length of this tuple is always 2, so "ordered pair" would be more precise.

- In the `Memory` library, there are discrepancies between names in the documentation versus in the code. On lines 77 and 83, the comment refers to the beginning of the bytes array as `start` while the code refers to the same thing as `offset`. On line 83, the comment refers to the length of the byte array as `length` while the code refers to it as `len`.
- In `ERC2771Forwarder`, since `tryRecover` was replaced with `tryRecoverCalldata` in the code, the documentation should update "ECDSA-tryRecover" to "ECDSA-tryRecoverCalldata".
- The following typographical errors were also identified:
  - In `Accumulators`, the word "not" should be capitalized.
  - In `Arrays`, "i.e." should be "e.g."
  - In `RLP`, "cannot de" should be "cannot be".
  - In `RLP`, the following brace references 1, 2, 3 are malformed and the "-" should be removed.
  - In `RLP`, the second backtick is missing before "length".

Consider addressing all the noted instances of incorrect documentation to improve the clarity and maintainability of the codebase.

Update: *Partially Resolved in pull request #6330. The team stated:*

> *The incorrect documentation instances have been addressed with the following exceptions:*
>
> *- In `TrieProof` :*
>
> *- This comment in the internal `_getNodeId` function is kept. We consider it's worth documenting the behavior of the function when a slice input of length 32 is provided. Especially since there's no dedicated error. In practice, nodes should be <32 bytes or >=33 bytes.*
>
> *- In `LowLevelCall` we're keeping the use of "tuple" to match how Solidity refers to multi-value returns like `(bytes32, bytes32)` . There is no separate "pair" type or term; "tuple" is the standard word for "multiple values with a fixed order."*
>
> *- AsciiDoc syntax is kept to match our documentation engine*

L-08 Inaccurate Error Message When Traversing Empty Branch Slot

When traversing a branch node where the target child slot is empty (indicating the key does not exist in the trie), the function returns `INVALID_SHORT_NODE` or `INVALID_LARGE_NODE` instead of a more descriptive error.

In RLP encoding, empty branch slots are represented as 0x80 (the RLP empty string), which is 1 byte, not 0 bytes. When `_getNodeId` is called is called with this 1-byte slice, it returns `nodeIdLength = 1`. On the next iteration, validation fails due to length mismatch: - If next proof element ≥ 32 bytes: `currentNodeIdLength != 32` (1 ≠ 32) → `INVALID_LARGE_NODE` - If next proof element < 32 bytes: `currentNodeIdLength != encoded.length` (1 ≠ len) → `INVALID_SHORT_NODE`

These errors suggest that the proof encoding or hashing is incorrect, whereas the actual issue is that the key does not exist in the trie.

Consider checking for the RLP empty string `0x80` before calling `_getNodeId` and return a dedicated error such as `KEY_NOT_IN_TRIE` or `EMPTY_BRANCH_SLOT`. This would align with how other "key not found" scenarios are handled (e.g., `MISMATCH_LEAF_PATH_KEY_REMAINDER`, `INVALID_PATH_REMAINDER`).

Update: *Resolved in pull request #6330. The team stated:*

> *We documented the empty-branch-slot case in the same NOTE as the 32-byte case in `_getNodeId` because both lead to INVALID_LARGE_NODE / INVALID_SHORT_NODE for a different underlying reason. We did not add a dedicated error (e.g. KEY_NOT_IN_TRIE) so we avoid an extra check and keep the happy path cheaper; the comment explains how to interpret the existing errors.*

L-09 `TrieProof` Rejects Valid Merkle-Patricia Proofs With Inline Extension Leaf Nodes

`tryTraverse` fails to verify valid Ethereum Merkle-Patricia trie proofs when an extension node references an inline (non-hashed) child node. In these cases, traversal reaches the end of the provided proof without returning a value and incorrectly reports `ProofError.INVALID_PROOF`. This behavior is incompatible with the Ethereum MPT specification and with proofs returned by standard clients.

This causes false negatives during proof verification: valid proofs may be rejected even though they correctly prove inclusion against the provided root. While this does not introduce a security vulnerability (invalid proofs are not accepted), it may lead to:

- incompatibility with standard Ethereum proofs
- failed verification for otherwise correct transactions, receipts, accounts, or storage slots
- operational issues for protocols relying on `TrieProof` for on-chain verification

Consider modifying `TrieProof.tryTraverse` to correctly handle extension nodes that reference inline (non-hashed) child nodes. By doing so, traversal can continue without exhausting the proof.

Update: *Resolved in pull request #6351.*

L-10 Incorrect Lower-bound Validity Check in `getValidationData`

`getValidationData` incorrectly marks a `UserOperation` as out of time range when `current == validAfter`, contradicting the validity semantics defined in EIP-4337, which specify that a `UserOperation` is valid starting at `validAfter`. This results in a one-unit (timestamp or block) false-negative validity window.

The function determines whether the current timestamp or block number is within the validity window encoded in `validationData`. The lower-bound check uses `current <= validAfter` which causes `outOfTimeRange` to be true when `current == validAfter`. However, according to EIP-4337, `validAfter` is inclusive, meaning the operation must be considered valid starting at that exact timestamp or block.

Consider updating the lower-bound check to respect the inclusive semantics defined by EIP-4337.

Update: *Acknowledged, not resolved. The team stated:*

> *Our logic is intentional: it matches the canonical EntryPoint, which uses `block.timestamp <= data.validAfter` and `block.number <= validAfterBlock`. We keep this behavior so our utils stay consistent with the reference implementation. If the ERC or EntryPoint change, we can revisit.*

Notes & Additional Information

N-01 Missing Bound Check for Long-Form Length-of-Length

The RLP encoder implementation does not enforce the RLP specification's implicit bound that the length-of-length (`lenOfLen`) used in long-form encoding must be ≤ 8 bytes (i.e., payload length must be `< 2**64`). For `lengths ≥ 2**64`, the computed prefix byte overflows the valid RLP prefix ranges and can wrap into a different item class (e.g., bytes encoding producing a list prefix), resulting in malformed / type-confused encoding.

RLP long-form encoding reserves a single prefix byte to encode both the item type and the number of bytes used to represent the payload length:

- Long bytes prefix range: `0xB8..0xBF` -> $lenlength \in [1..8]$
- Long list prefix range: `0xF8..0xFF` -> $lenlength \in [1..8]$

If `lenlength > 8`, the computed prefix exits these ranges:

Bytes case (`offset = 0x80`)

- For `length >= 2**64, lenlength = 9` and `prefix = 0xB7 + 9 = 0xC0` 0xC0 is the short-list base prefix, meaning a long-bytes encoding could be misinterpreted as a list.

List case (`offset = 0xC0`)

- For `lenlength = 9`: `prefix = 0xF7 + 9 = 0x100`

Since the implementation writes the prefix with `mstore8`, this value truncates to `0x00`, producing an invalid/nonsensical RLP prefix. This issue is not currently exploitable on the EVM, because constructing/copying a bytes payload of size `≥ 2**64` is infeasible, and the encoder will run out-of-gas before returning any output.

If the encoder were used in an environment where extremely large payload lengths are possible (or if the implementation were refactored to avoid copying full payloads), it could produce malformed encoding with incorrect type prefixes.

Consider checking that the length to encode is bound to `2**64 - 1`.

Update: *Acknowledged, not resolved. The team stated:*

> *On the EVM the scenario is unreachable: either the ABI decoder runs out of gas when copying calldata, or, with an assembly-forged large buffer, execution reverts (e.g. OOG) on the first use of the length before the encoder runs. We are not adding a length bound check.*

N-02 Redundant `item.length()` Call in RLP Encoding

In the `_decodeLength` function, `item.length()` is called again in the short list case despite having already stored this value in `itemLength` at the beginning of the function. The Slice length extraction involves bit-masking operations that are repeated unnecessarily, resulting in a gas inefficiency.

Consider reusing the existing `itemLength` variable instead of calling `item.length()` again.

Update: *Resolved in pull request #6330.*

N-03 `escapeJSON` Does Not Escape Raw Control Characters Like NULL Byte

The `escapeJSON` function escapes backslashes (which covers unicode escape sequences like `\uXXXX`), double quotes, and specific control characters (`\b`, `\t`, `\n`, `\f`, `\r`). However, it does not escape raw control characters in the range U+0000 to U+001F that are not in the explicit list.

For example, the NULL byte (0x00) is not escaped and will be left as-is in the output. According to RFC-4627 Section 2.5, all characters from U+0000 to U+001F must be escaped in JSON strings. Given that `escapeJSON` is designed for NFT metadata that gets rendered by marketplaces in browsers, unescaped control characters could potentially cause parsing issues or unexpected behavior in downstream consumers.

Consider escaping all control characters in the range U+0000 to U+001F, or document that only specific control characters are escaped and that callers should sanitize input if full RFC compliance is required.

Update: *Resolved in pull request #6344.*

N-04 Inconsistent Use of Decimal vs Hexadecimal Notation for Slice Offsets

The account utilities use inconsistent notation for slice offsets when extracting addresses and other fixed-size data. For the same concept (address = 20 bytes), different files use different notation:

- `ERC7579Utils.sol` uses hexadecimal: `executionCalldata[0x00:0x14]`
- `AccountERC7579.sol` uses decimal: `signature[20:]`

In addition, `ERC7579Utils.sol` mixes notation within the same expression, using decimal for the start index and hexadecimal for the end index: `executionCalldata[0:0x14]`.

Consider adopting a consistent notation convention across the codebase.

Update: *Resolved in pull request #6345. The team stated:*

> *In ERC7579Utils we use hex slice bounds (e.g. 0x14, 0x34) so offsets align with EVM word size (0x20) and are easy to read. In AccountERC7579 we use decimal for the signature (e.g. signature[20:]) because "20" clearly means "after the 20-byte address."*
>
> *So 4 and 20 are kept in decimal where they denote selector/address size; hex is used where we're counting memory/word layout.*
>
> *Other numeric literals follow our GUIDELINES.*

N-05 `setFreeMemoryPointer` Should Follow the Unsafe Naming Convention to Reflect Potential Memory Corruption

In the `Memory` library, the `setFreeMemoryPointer` function sets the free memory pointer, located at byte offset `0x40` in memory, to an arbitrary user-supplied `Pointer` (alias for `bytes32`) value. The assembly block is annotated as `memory-safe`, which is a semantic promise that all memory-safety invariants are preserved, including that the free memory pointer

always points to unallocated memory. Since the function accepts an arbitrary value, this invariant is not enforced and depends entirely on the caller:

- Setting FMP to `0x00` would cause subsequent allocations to overwrite scratch space, the FMP itself, and the zero slot.
- Setting FMP backwards would cause subsequent allocations to overwrite previously allocated data.

While the library's documentation states that the user should conform to Solidity's memory-safety guidelines and the `setFreeMemoryPointer` NatSpec warns that everything after the pointer may be overwritten, the function name does not signal the danger level to callers. The function's philosophy is to give users raw access without restrictions, but since there is no runtime enforcement that the supplied value conforms to memory-safety invariants, the `memory-safe` annotation on the inner assembly block becomes a promise the function cannot guarantee on its own.

Consider renaming the function to `unsafeSetFreeMemoryPointer` to align with OpenZeppelin's naming convention for functions that can produce unsafe outcomes if misused (e.g., `unsafeAccess` in `Arrays.sol`). This makes the risk explicit at the call site without adding runtime overhead. Moreover, consider reinforcing the warning at the NatSpec level with explicit constraints (e.g., the pointer must be `>= 0x80` and must not point backwards into already-allocated memory).

Update: *Resolved in pull request #6348. The team stated:*

> *The setter is renamed to `unsafeSetFreeMemoryPointer` to match our convention for unsafe helpers (e.g. `unsafeAccess`), and we've removed `asPointerasBytes32` so callers must use the UDVT directly and can't create pointers from arbitrary values. The NatSpec already states the FMP must not be set below `0x80`.*

Client Reported

CR-01 `uint8` Wraparound In `tryParseV1` And `tryParseV1Calldata` Causes Incorrect Slice Bounds

`tryParseV1`, `parseV1`, `tryParseV1Calldata` and `parseV1Calldata` can return a corrupted empty `addr`. The root cause is that `chainReferenceLength` and `addrLength`

were declared as `uint8` (see `tryParseV` 1 and 2, and `tryParseV1Calldata` 1 and 2). Inside the unchecked block, Solidity infers arithmetic involving these variables as `uint8`-width, so the expression `0x06 + chainReferenceLength + addrLength` wraps modulo 256 when the sum exceeds 255. This caused the `addr` slice bounds to be computed from the wrapped value, returning empty bytes for `addr` with no revert and `success = true`. The issue occurs when any call to `parseV1` / `tryParseV1` (and their `calldata` variants) where `chainReference.length + addr.length > 249`.

Update: *Resolved in pull request #6372.*

Conclusion

Overall, the v5.6 changeset was found to be of high quality, well documented, and consistent with secure development best practices. The findings are mostly of low- or note-severity, and center on clarity and edge-case hardening in newly added utilities, Merkle-Patricia trie, and encoders/decoders—particularly RLP (memory aliasing behavior, non-canonical encodings, missing error parameters, `encode(bytes[])` concatenation semantics), Base64 (documentation mismatch regarding revert behavior), and `InteroperableAddress` (parsing functions accepting inputs that violate ERC-7930 when both `chainReference` and address are empty, trailing bytes acceptance). Additional observations include minor documentation improvements, standardizing hex-style patterns across account utilities, `escapeJSON` not escaping all RFC-required control characters, enforcing non-user-provided keys in Checkpoints, and missing length/bounds checks (e.g., `Accumulators.flatten`, `WebAuthn._validateChallenge`).

The medium-severity issues identified include an interoperability/DoS risk in `ERC7786Recipient` where replay protection keyed by `(gateway, receiveId)` can permanently block message delivery if a compliant gateway represents an "empty" `receiveId` as `bytes32(0)`, and memory aliasing in RLP single-byte encoding where returned bytes share memory with the input, potentially causing silent data corruption in security-sensitive operations. Related low- and note-severity observations highlight that cross-chain and WebAuthn integrations are sensitive to threat-model and integration choices (replay/reentrancy assumptions, potential message loss around `_processMessage`, and validation bypass/fallback behaviors).

The final state of the audited codebase, including all implemented resolutions, is reflected in commit 5fd1781.

Collaborating with the OpenZeppelin Contracts team has been a pleasure, as always. The Contracts team is commended for the good work, and thanks are extended to them for promptly and diligently answering our numerous technical questions in great detail and engaging in follow-up discussions.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.