Scanned Code Report

AUDITAGENT

Powered by NETHERMIND SECURITY

Code Info

Auditor Scan

Scan ID 37	Date February 19, 2026
Organization igor53627	Repository evm-linear-accumulator
Branch main	Commit Hash 10b6ac13...dfec3775

Contracts in scope

src/LibLinearAccumulator.sol

Code Statistics

Findings 4	Contracts Scanned 1	Lines of Code 124

Findings Summary

Total Findings

- High Risk (0)
- Medium Risk (0)
- Low Risk (2)
- Info (2)
- Best Practices (0)

Code Summary

The `LibLinearAccumulator` contract is a gas-optimized library for performing linear hash accumulation over a finite field. It provides a cryptographic primitive for applications requiring incremental hashing and data integrity checks.

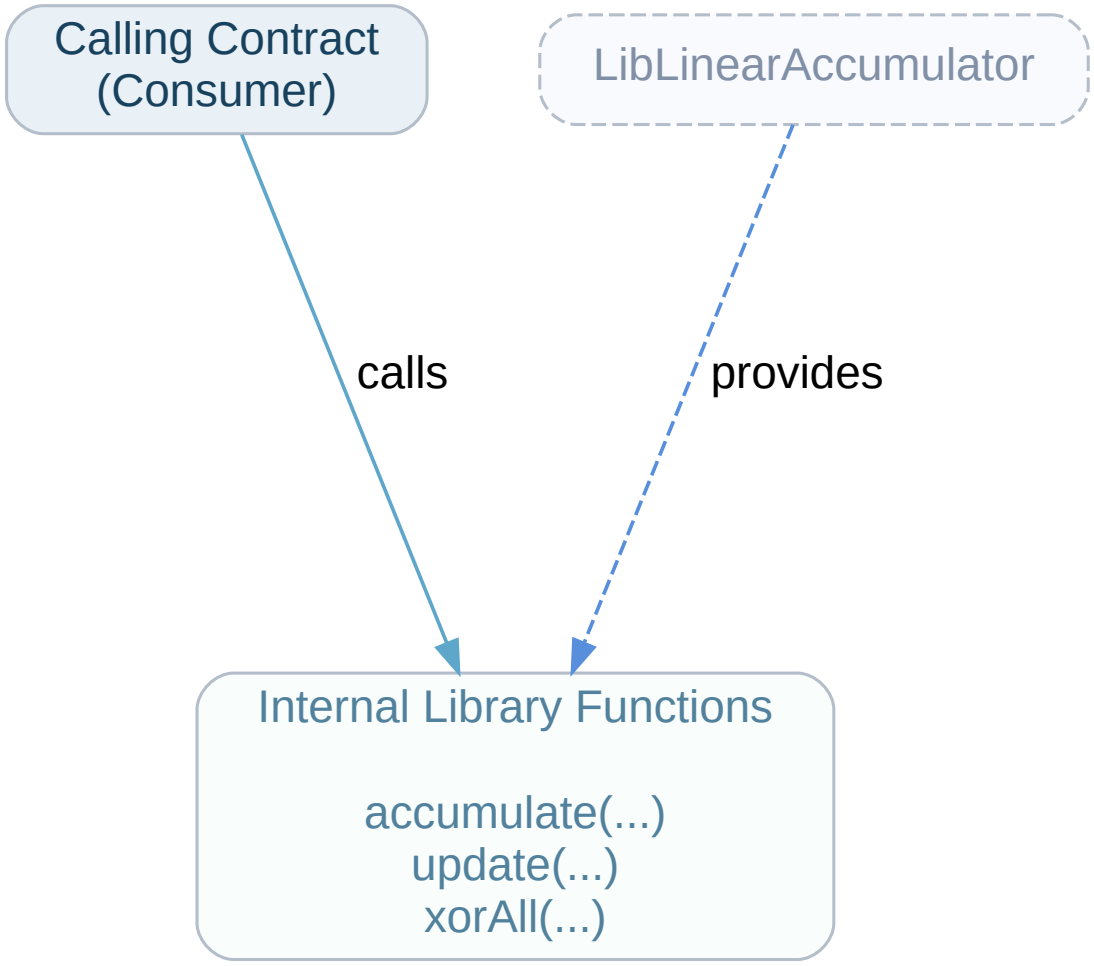
The core functionality revolves around computing `H(x) = A * x mod q`, where `x` is an input bit-vector and `A` is an N-by-N matrix. The matrix `A` is not stored but is derived deterministically on-the-fly from a provided `seed`, a `stepIndex`, and the row index, using the `keccak256` hash function. This approach saves significant storage costs while ensuring the matrix is pseudo-random and reproducible. The calculations are performed modulo `q`, which is constrained to fit within 16-bit lanes for efficiency. The resulting hash is a packed array of 16-bit elements.

The library is implemented using inline assembly for maximum gas efficiency and is intended as a general-purpose tool for more complex protocols. Potential use cases include trace integrity for optimistic rollups, state accumulators, fraud proof integrity checks, and other scenarios involving incremental hashing. It also includes helper functions to update an existing accumulator state with new input.

Entry Points & Actors

This contract is a library and contains only `internal pure` functions. It is designed to be used by other contracts and does not have any external state-modifying entry points that can be directly called by actors.

Code Diagram



1 of 4 Findings

src/LibLinearAccumulator.sol

Magic Numbers Instead Of Constants

Low Risk

The contract uses the magic number `4` multiple times in array return type declarations `uint256[4] memory` across 10 different function signatures. Instead of repeating this literal value, a constant state variable should be created and referenced throughout the contract to improve code maintainability and reduce the risk of inconsistencies if the value needs to be changed in the future.

2 of 4 Findings

src/LibLinearAccumulator.sol

PUSH0 Opcode Compatibility Issue

Low Risk

The contract uses Solidity compiler version 0.8.20 or higher (pragma solidity ^0.8.20), which defaults to Shanghai EVM version. This means the generated bytecode will include PUSH0 opcodes. If deployment is intended for chains other than Ethereum mainnet, such as Layer 2 solutions (Arbitrum, Optimism, Polygon, etc.) or other EVM-compatible chains, ensure the appropriate EVM version is explicitly configured. Many L2 chains and alternative networks do not yet support the PUSH0 opcode, and deployment will fail without proper EVM version configuration.

3 of 4 Findings

src/LibLinearAccumulator.sol

Modulus `q` is treated as a finite-field modulus in documentation, but the library permits composite `q`, which can invalidate “field” assumptions

Info

The project documentation describes the accumulator as operating over a finite field Z_q, but the implementation only enforces a numeric range constraint on `q` and does not ensure that `q` is prime.

Code paths accept any `q` in `[2, 65521]`:

```
require(q >= 2 && q <= 65521, "q must be 2-65521");
```

Internally, coefficients and the accumulated sum are reduced with `mod(..., q)` and conditional subtraction:

```
let aij := mod(and(shr(mul(k, 16), blockDigest), 0xFFFF), q)
...
if bitVal {
  sum := add(sum, aij)
  if iszero(lt(sum, q)) { sum := sub(sum, q) }
}
```

If a caller supplies a composite modulus (e.g., `q=4`, `q=65520`, etc.), the arithmetic is no longer over a field (it is over a ring with zero divisors). Any higher-level protocol logic that relies on field properties (e.g., invertibility conditions, uniqueness arguments, algebraic soundness arguments that assume all non-zero elements are invertible) can silently become invalid even though this library will still return outputs that satisfy the lane bounds `< q`.

This is configuration-dependent: the library itself functions consistently for composite `q`, but it does not enforce the “finite field” precondition implied by the documentation.

4 of 4 Findings

src/LibLinearAccumulator.sol

XOR compression in update function creates collision space undermining incremental accumulation integrity

Info

Root cause: update() reduces the 4-word (1024-bit) accumulator state to a single 256-bit value via xorAll(acc) before mixing with newInput and re-accumulating. This is a many-to-one compression: any acc1 != acc2 with xorAll(acc1) == xorAll(acc2) will always produce identical update outputs for the same (newInput, stepIndex, numRows, seed, q).

Exploit path: Practical exploitation depends on attacker influence over acc, not on finding natural collisions. If a consuming protocol accepts externally supplied accumulator states, or stores/loads acc from attacker-writable locations without provenance validation, an attacker can trivially craft alternate acc values with a chosen xorAll(acc) and thereby cause divergent histories to be indistinguishable under update().

Non-exploit scenario: If acc is strictly protocol-controlled (only derived from prior trusted accumulate/update operations) and never attacker-injectable, the existence of the collision space is generally not practically exploitable, though it remains a sharp integration risk.

Impact: Protocols using update() for binding-critical applications (trace integrity / wire binding, fraud-proof integrity) may lose the ability to distinguish different intermediate state histories if acc can be manipulated or injected. Documentation currently presents update as a generic incremental hashing mechanism for integrity use cases without clearly warning about this lossy compression and the required provenance constraints.

Recommendation: Document update() as a convenience tradeoff (1024→256-bit compression) and recommend single-shot accumulate() for maximal binding. Warn integrators to enforce accumulator provenance and parameter consistency. Consider an alternative update that preserves full state (e.g., mixing over the full 4-word state before re-accumulating, or removing xorAll compression from binding-critical flows).

Disclaimer

Kindly note, no guarantee is being given as to the accuracy and/or completeness of any of the outputs the AuditAgent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The AuditAgent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the AuditAgent.

Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.