OpenZeppelin | security

Stylus Contracts Library v0.3.0 Audit

September 8, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-08-11 To 2025-08-29
Languages	Rust

Total Issues	17 (17 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	4 (4 resolved)
Low Severity Issues	5 (5 resolved)
Notes & Additional Information	7 (7 resolved)

Scope

OpenZeppelin audited the OpenZeppelin/rust-contracts-stylus library at commit 231d4f1 (`v0.3.0-rc.1`). This is a follow-up audit after the v0.2.0 audit, focusing on the refactors and newly added features.

In scope are all the changes made in the following directories:

```
├── contracts/**/*.rs
├── contracts-proc/src/**/*.rs
└── lib/crypto/src/**/*.rs
```

The pull requests containing the in-scope changes are listed here.

System Overview

The changes under review introduce a series of improvements, including breaking changes, new features, and important bug fixes that expand the library's scope and usability.

On the contracts side, the library adds support for `ERC721Holder` and `ERC1155Holder`, enabling safer and more standardized handling of token transfers in Stylus-based applications. The release also introduces native support for the UUPS upgradability pattern, aligning the library with widely adopted upgrade standards and making proxy-based upgrade flows more flexible. Furthermore, the update extends the `EnumerableSet` utility with generic type support, allowing developers to work with sets of more complex data types beyond basic primitives.

On the cryptography side, the library adds a precompile wrapper for `secp256r1` (P-256) signature verification, leveraging Arbitrum's native precompile to enable efficient and gas-optimized verification of modern authentication schemes such as passkeys. Complementing this, the update also introduces support for `Ed25519` (EdDSA) signatures, broadening the available cryptographic primitives for Stylus contracts.

Additionally, the release enhances interoperability between Solidity-style unsigned integers and Rust primitive integers by introducing conversion functions that explicitly handle 64-bit and 128-bit integer sizes, thereby improving low-level arithmetic compatibility. The addition of more documentation and tests further enhances the library's robustness.

Overall, these updates make the OpenZeppelin Stylus library significantly more robust, feature-complete, and aligned with modern Ethereum and Arbitrum standards. The codebase appears to be well-documented, with all the changes organized neatly.

Security Model and Trust Assumptions

This audit assumes that the Stylus SDK and all third-party dependencies used by the audited library are secure and behave as documented. Users of the audited library are assumed to

strictly follow the API descriptions, warnings, and examples to avoid potential errors. Especially, the following assumptions apply:

- Users of the UUPS proxy will adhere to the upgrade guidelines, properly initializing the proxy and incrementing the version number in new implementations.
- The signing functions of `Ed25519` signatures should only be used in an off-chain environment, with the `ExpandedSecretKey` data kept confidential.

High Severity

H-01 Incorrect Bandersnatch and Jubjub Curve Parameters

The `bandersnatch.rs` file defines the Bandersnatch elliptic curve parameters, but several constants do not match the standard specification from the referenced paper.

For example, the generator point coordinates `G_GENERATOR_X` and `G_GENERATOR_Y` do not match the standard subgroup generator. The current values are:

```
x = 19232933407424889111104940496320988988172100217998297030544530116054238325480
y = 17060630597514316424753713474449400526842970574619404045211133826199305117375
```

However, the standard values are:

```
x = 0x29c132cc2c0b34c5743711777bbe42f32b79c022ad998465e1e71866a252ae18
y = 0x2a6c669eda123e0f157d8b50badcd586358cad81eee464605e3167b6cc974166
```

In addition, there are internal inconsistencies among the defined values. For example, the `COFACTOR` is correctly set to 4, but the cofactor inverse `COFACTOR_INV` appears to be incorrect, since:

```
4 * 9820571595336158597396451345284868594481866920080427688839802480047265754601 ≠ 1 mod
r
r = 13108968793781547619861935127046491459309155893440570251786403306729687672801
```

The correct inverse should be `9831726595336160714896451345284868594481866920080427688839802480047265754601`.

A similar issue also exists in the `jubjub.rs` file. For example, the scalar field modulus (`Fr::MODULUS`) is set to `72370055773322622139731865630429942408571163593799076060019509382854542585 7`, which is not the standard prime subgroup order.

Consider revising these parameters to be consistent and meet the standard specification.

Update: *Resolved in pull #809 and merged at commit 7c7fda9. The team corrected the parameters and added more test cases.*

Medium Severity

M-01 Inadequate Byte Padding in `ruint` to `Uint` Conversion

The `From<ruint::Uint<B, L>> for Uint<L>` implementation contains a potential unexpected panic due to insufficient byte padding during type conversion. The conversion process involves two steps:

1. Converting `ruint::Uint<B, L>` to little-endian bytes via `to_le_bytes_vec()`
2. Creating a new `Uint<L>` from those bytes using `from_bytes_le()`

While `ruint::Uint<B, L>` **allows** `B <= L*64` (ensuring all bits can fit within the limbs), the `to_le_bytes_vec()` function returns exactly `ruint::Uint::BYTES` bytes. However, `from_bytes_le()` expects **exactly** `Uint::BYTES` bytes and will panic if the byte count does not match, even with enough space (limbs) to hold the bits. For example, the `to_le_bytes_vec` function produces a 200-byte vector for `ruint::Uint<200, 4>`, but `Uint<4>::from_bytes_le()` expects 256 bytes, causing a panic despite having sufficient limb capacity.

Consider padding the bytes returned from `to_le_bytes_vec` if they are shorter than `L*64`.

Update: *Resolved in pull #808 and merged at commit a53fac4.*

M-02 Incorrect Limb Number Check in `into_u128`

The `into_u128` method requires access to two limbs (`self.limbs[0]` and `self.limbs[1]`) to construct a `u128` value, but only checks that `N >= 1`. This allows the method to be called on `U64` types, which will panic when attempting to access the non-existent second limb, causing an index-out-of-bounds panic. In addition, converting an unsigned integer with 64 bits to a `u128` value should be allowed and should always succeed.

Consider revising the logic in the `into_u128` method to only access `limbs[1]` if the `Uint` variable has more than or equal to 2 limbs. For the `U64` type, the highest 64 bits of `u128` should be set to zero.

Update: *Resolved in pull #815 and merged at commit 7777e3d.*

M-03 Zeroization of `ExpandedSecretKey` Is Incomplete

The `ExpandedSecretKey` struct in `eddsa.rs` holds secret material (the signing scalar and the hash prefix). It is intended to be zeroized on drop via the `zeroize` crate:

> Instances of this secret are automatically overwritten with zeroes when they fall out of scope.

However, the current design is ineffective: the type is marked `Copy`, which is not allowed for types with destructors, so a custom `Drop` (or `Zeroize`/`ZeroizeOnDrop`) cannot be used. Moreover, implementing the `ZeroizeOnDrop` trait alone does nothing without deriving `Zeroize` and enabling the drop hook. As a result, no on-drop zeroization occurs, and secrets can remain in memory. The code also risks the silent duplication of secrets due to `Copy`.

In contrast, `ed25519-dalek` uses a destructor drop to ensure proper zeroization.

Consider removing `Copy` and using `Zeroize` and `ZeroizeOnDrop` with the derive macro.

Update: *Resolved in pull #831 and merged at commit aafb627.*

M-04 Potential Signature Malleability in `p256_verify` Precompile Wrapper

The spec of the `P256VERIFY` EVM precompile requires two checks to be implemented:

> - Verify that the `r` and `s` values are in `(0, n)` (exclusive) where `n` is the order of the subgroup.
> - Verify that the point formed by `(x, y)` is on the curve and that both `x` and `y` are in `[0, p)` (inclusive 0, exclusive p) where `p` is the prime field modulus. Note that many implementations use `(0, 0)` as the reference point at infinity, which is not on the curve and should therefore be rejected.

Notably, the spec does not enforce low-`s` values for ECDSA signatures on secp256r1, allowing malleability where `(r, s)` and `(r, n - s)` are both valid.

The `p256_verify` function is a thin wrapper around the precompile and does not enforce low-`s` normalization either. This can lead to practical issues such as replay or duplicate processing in systems relying on signature uniqueness (e.g., nonce-based flows), potentially enabling unauthorized actions or other unexpected results.

Consider adding a check (`s <= n/2`) before invoking the precompile, as is done in the Solidity library.

Update: *Resolved in pull #825 and merged at commit 7c04df8.*

Low Severity

L-01 Incorrectly Indexed Parameters in `AdminChanged` Event

According to the EIP-1967, changes to the admin slot should be notified by the `AdminChanged` event, which is defined as:

```
event AdminChanged(address previousAdmin, address newAdmin);
```

Both the `previousAdmin` and `newAdmin` parameters are not marked as `indexed`, which means that they will be stored in the log data field and not as separate topics. The `IERC1967` interface from the OpenZeppelin Contracts library follows this standard.

However, in the Stylus ERC-1967 `mod.rs` file, both parameters have been marked as `indexed`, which means that they will be stored as separate topics. While this does not pose a direct security threat, the inconsistent logging behavior may cause issues for users, particularly off-chain entities that parse emitted logs.

Consider removing the `indexed` keyword to ensure consistency with the standard.

Update: *Resolved in pull #794 and merged at commit f3bb0e4.*

L-02 Incorrect Transformation for Projective Points with Zero z-coordinate in `normalize_batch`

The `normalize_batch` function normalizes multiple curve points to their affine representations with the `(x, y, t, z) -> (x/z, y/z, t/z, 1)` conversion. To get inversion `1/z` value, it first calls the `batch_inversion` function, which leaves zero `z` elements unchanged, so any `g.z == 0` in the input remains zero after inversion.

During the affine transformation step later, the code checks `g.is_zero()` to decide whether to return `Affine::zero()`. However, since `g.is_zero()` returns false when `g.z == 0` for the Twisted Edwards curve, the code proceeds to compute `x = g.x * z` and `y = g.y * z`, which results in `(0, 0)` for the affine point. This is not a valid affine representation and may lead to incorrect results in downstream cryptographic operations.

Consider updating the affine transformation logic in `normalize_batch` to check for `g.z.is_zero()` situations.

Update: *Resolved in pull #817 and merged at commit bb3720a.*

L-03 Insufficient Solution for the Immutable Implementation Address in UUPS

The standard UUPS upgradability pattern in Solidity uses an immutable variable to store the implementation's (i.e., the logic contract's) deployed address. In `_checkProxy`, this immutable value is compared with the implementation address stored in the proxy to determine whether the current implementation is correct.

Since Stylus cannot handle immutable variables, there is no way to embed a variable into the implementation's code at deploy time. The current workaround in `only_proxy` is to save the implementation address directly in the Proxy's storage and use it for comparison with the current contract address. However, this solution is insufficient: by storing the address directly in the Proxy's storage and comparing it against the Proxy's own storage, it defeats the purpose of the check. This could lead to issues, such as failing to detect whether the current call is using the correct implementation.

Consider improving the `only_proxy` design by taking the following measures:

- Introduce a variable `logic_flag` in the implementation's storage to indicate that the current storage belongs to the implementation, while leaving the Proxy's storage uninitialized for this variable. This variable is checked to ensure the context is in a delegate call.
- Check that the ERC-1967 slot is not empty to ensure that it is a 1967 proxy.
- Maintain a `version_number` variable both in the Proxy's storage and in the implementation code (`const`, i.e., not in the implementation's storage). Keeping these values in sync allows for comparing the implementation code and the Proxy's storage to verify that the current version is correct.

Update: *Resolved in pull #810 and merged at commit 4c8275a.*

L-04 `allowance` Function Returns Zero for Non-Existent Tokens or Contracts with Fallbacks

The `allowance` function is called by `safe_increase_allowance` and `safe_decrease_allowance` to read the current allowance of the `spender`. In pull request #765, the `Address::has_code(&token)` check in `allowance` was removed with a comment:

> There's no code check in Solidity. In case of no code, a 0 (zero) is expected as the result
>
> For reference, the Solidity code is `token.allowance(address(this), spender)`.

However, the Solidity compiler performs an existence check before making any high-level calls, and reverts if the check fails. Such calls also revert if the return data cannot be decoded as `uint256`. By contrast, the Stylus implementation decodes empty return data as zero via `U256::from_be_slice(&result)`. This creates two inconsistencies with the Solidity library:

- When `token` is set to a non-existent address, the Solidity call reverts, while Stylus returns `0`.
- When `token` is set to a non-ERC-20 contract that has a fallback function, the Solidity call reverts, while Stylus returns `0`.

Consider updating `allowance` to first check whether `token` exists and requires exactly 32 bytes of return data (`U256`).

Update: *Resolved in pull #833 and merged at commit 0015161. The team added a check to ensure the `token` contract exists and replaced the `RawCall` with a high-level call.*

L-05 `*_relaxed` Functions Behave Differently From the Solidity Library

The `transfer_and_call_relaxed`, `transfer_from_and_call_relaxed`, and `approve_and_call_relaxed` functions invoke the `call_optional_return` helper to call the corresponding function on the `token` contract. This helper succeeds if the return data decodes to `true`, or if the return data is empty and the `token` address has code.

However, this differs from the OpenZeppelin Solidity library, where the corresponding functions are invoked via high-level calls (e.g., `token.transferAndCall`). In particular, if `token` is mistakenly set to a non-token contract that has a fallback function, the Solidity version reverts because no boolean return value is provided, while the Stylus `*_relaxed` versions succeed.

Consider aligning the Stylus `*_relaxed` functions with the Solidity behavior by requiring that the `token` contract exists and that the call returns data decoded to `true`.

Update: *Resolved in pull #837 and merged at commit 1389d05. The team adds checks to ensure contracts exist and uses high-level calls.*

Notes & Additional Information

N-01 Panic When Converting Projective with a Zero z-coordinate to Affine

The implementation of `From<Projective<P>> for Affine<P>` in `affine.rs` attempts to convert a projective point to its affine representation by inverting the `z` coordinate. The logic first checks if the point is the identity (`is_zero()`) and then checks if `z` is one (already normalized). If it is not, then the logic proceeds to invert `z`. However, for the twisted Edwards curve, `is_zero()` will return `false` for a point with a z-coordinate that is 0. After this, the code assumes the following:

> Z is nonzero, so it must have inverse in a field

Then, the code attempts to compute `p.z.inverse().unwrap()`. Since the inverse of zero does not exist, this will cause a panic at runtime.

While projective points over Edward's curves cannot receive Z coordinate as zero during computations, consider adding an explicit error panic message when `p.z.is_zero()`, and revise the comment to make it clear about the panic situation.

Update: *Resolved in pull #816 and merged at commit ba3da10.*

N-02 Unimplemented `values` Function with `start` and `end` Parameters

The enumerable set module does not implement the `values` function with `start` and `end` parameters. Such a function could reduce gas consumption or mitigate out-of-gas scenarios when processing large sets.

Consider implementing the `values` function to match the Solidity version of the contract.

Update: *Resolved in pull #827 and merged at commit 9167016.*

N-03 Inconsistent Use of Public Key in EdDSA

In the `compute_R` EdDSA function, the public key is used as `self.point` even though it was stored earlier in `A`.

For clarity, consider consistently using `A` as the public key, which is also in accordance with the formula notation.

Update: *Resolved in pull #830 and merged at commit 09a048c.*

N-04 Missing Checks for Equality of `Scalar` and `WideScalar` in EdDSA

In the EdDSA implementation, it is critical that during the signing process, the `Scalar` and `WideScalar` types both reduce their input (the scalar `r` coming from the hash of the prefix and the message) modulo the scalar field `MODULUS` of the curve. This requirement is currently satisfied: the implementation correctly reduces values by the scalar field order of Curve25519.

However, a future refactor or upgrade could accidentally introduce a mismatch (i.e., a situation in which the reduction modulus of `Scalar` differs from the reduction modulus of `WideScalar`).

Consider adding an explicit compile-time assertion (e.g., `assert!(Curve25519FrParam::MODULUS == Curve25519Fr512Param::MODULUS)`) to guarantee that both parameter sets always share the same modulus.

Update: *Resolved in pull #834 and merged at commit d1f074d. Instead of adding an assertion, the team ensures the `GENERATOR` and `MODULUS` parameters are the same between curve ed25519 with `512-bit` inner integer size and `256-bit` size.*

N-05 Missing Gas Consumption Warning for Enumerable Functions

The `clear`, `values`, and `get_role_members` functions iterate over or return all elements in a set, potentially incurring significant gas costs when the set contains a large number of elements. This could result in unexpectedly high gas consumption or even a denial-of-service situation. For reference, the Solidity version by OpenZeppelin contains relevant comments about the possibly expensive operations.

Consider incorporating warning documentation for these functions to alert developers regarding the potential risks.

Update: *Resolved in pull #832 and merged at commit dc58c6e.*

N-06 Typographical Error in EdDSA Error Message

This error message in the EdDSA implementation uses an incorrect unit `bit`:

> .expect("Y coordinate should be of 32 **bit**").

Since the length of the Y coordinate is 32 **bytes**, consider updating the error message to "Y coordinate should be 32 bytes" in order to avoid confusion and accurately reflect the expected size.

Update: *Resolved in pull #826 and merged at commit b1b0a0c.*

N-07 Missing Validation on MSB of `y` in `CompressedPointY`

In the compressed Edwards-Y encoding, the most significant bit (MSB) is reserved for the x "sign" (parity), while the lower 255 bits hold y. The current code uses XOR to set this bit (`s[31] ^= (is_odd as u8) << 7`), which depends on y's MSB being zero. Given the existing type checks in `Affine`/`Projective` which ensure that points are on-curve and in the prime-order subgroup, y's MSB should always be zero, making the current XOR safe.

To further harden the code and fail fast on invalid inputs, consider adding an `assert!(s[31] >> 7 == 0)` check before the XOR. This preserves correctness for valid inputs and surfaces malformed points early without requiring a behavioral change to the encoding logic.

Update: *Resolved in pull #835 and merged at commit 72f042e. The team added a compile-time check to ensure the* `MODULUS` *has a spare bit, which means the Y coordinate will also have a spare bit.*

Conclusion

The Stylus library has been upgraded with support for `ERC721Holder` and `ERC1155Holder`, improving token-transfer handling, alongside native UUPS upgradability for flexible proxy upgrades and generic type support in the `EnumerableSet` contract for more data structures. The Crypto Library has been expanded with a precompile wrapper for `secp256r1` signature verification and `Ed25519` signature support, enabling efficient and modern authentication. Additional integer conversion functions for Solidity-style and Rust integers enhance interoperability. Supported by improved documentation and tests, the library maintains a clean, well-organized codebase.

The audit identified one high-severity issue, four medium-severity issues, and some low-severity issues. During the audit, the development team was proactive, promptly addressing any questions posed by the audit team. To ensure ongoing security and functionality, regular audits are recommended with future updates, fostering continued growth in the Arbitrum Stylus ecosystem.