AUTHORS
[3]

TRISTAN BRITT
@trbritt

OxALCIBIADES
@0xAlcibiades

GRUG
@CapitalGrug

OCTOBER
[2024]

[TITLE]

SolBLS Remediation Warlock × Zellic

SolBLS Changes

Invalid curve points accepted by `isValidPublicKey`

The initial version of this function did indeed only perform a curve check of a point of $E'(\mathbb{F}_{p^2})$, and crucially did not explicitly perform a subgroup check for membership in the correct torsion, $[r]E'(\mathbb{F}_{p^2})$. The rationale behind this decision was that the key cryptographic functionality, namely the calling of the `ecPairing` precompile at `0x08`, would perform the subgroup membership check and revert for an invalid public key. However, this is a subtle detail that, as correctly identified, leaves ambiguous and potentially insecure usage of this library. Therefore, per the suggestion, we have implemented the subgroup membership check for $\mathbb{G}_2$ using the `ecPairing` precompile. Note that this does indeed also check for curve membership, and therefore suffices as the sole check required for a valid public key.

Remediation

We created the `isElementOfG2` function that is used in `isValidPubKey` to perform both curve membership and subgroup membership checks.

`ModexpInv`, `ModexpSqrt`, and `sqrt`

Listings 3.2, 3.3, and 3.4 relate to the modular exponentiation through which square roots and inverses are computed mod N.

In the original square root implementation that used the addition chain method, the library accepted values larger than the modulus, and reduced modulo N accordingly. Furthermore, the `ModexpSqrt` library would return a value that would either be a square root of x, or $-x$, depending on the relevant Legendre symbol. It was therefore the onus of the user to check if what was returned by `ModexpSqrt` was indeed a valid square root or not. This was accomplished by the `sqrt` function in `BLS.sol`, which performed the relevant `hasRoot = mulmod(x, x, N) == xx` check, and returned this bool accordingly. However, it was noted that because of this ambiguity in the validity of the value returned by `ModexpSqrt`, combined with the fact that the function accepts values larger than the modulus, that the check `hasRoot` would always be false if the base was greater than or equal to the modulus, or if there was no square root of the base modulo N.

Remediation

We implemented the suggested check in `sqrt` of `BLS.sol` that `hasRoot = mulmod(x, x, N) == (xx % N)` to handle the case of the base being larger than the modulus, such that `hasRoot` is indicative only of the presence of a valid square root. This behaviour is now documented in both the `@notice` of `ModExp/ModexpSqrt` and in `BLS/sqrt`.

This being said, as noted in the report, the usage of the addition chain method is inefficient given the cost of the precompiles post EIP-198.

Remediation

The core of the `ModexpSqrt` library (as well as `ModexpInv`) now relies on the precompile at `0x05` to perform the relevant exponentiation.

A key behaviour to note is that by EIP-198, this precompile is required to return 0 for a base of 0, which is compliant with RFC 9380's standard for `inv0` in the modular inverse case, and consistently correct in the sqrt case, meaning that no special handling is needed for base 0 in either of these libraries. Likewise, both reth and geth handle the case of a base larger than the modulus, see the implementation of `modexp` in `reth`, and `geth`.

Inefficient gas usage in `mapToPoint`

The final steps of the SvdW encoding rely on checking the Legendre symbol of various values to determine if additional computation is necessary. However, given the updated usage of the square root functions, this is redundant, as the squareness of a value is already determined in these computations.

Remediation

The SvdW encoding now uses a more efficient implementation of the final steps of the encoding, allowing for the entire deletion of the `legendre` and `modExpLegendre` functions.

Inconsistent returns in `verifySingle`

Currently, the return value of this function does not consoliate the execution success of the precompile and the result of the pairing, which is inconsistent with usages throughout other functions.

Remediation

This function now returns a single boolean indicating both success in the static call of the precompile and of the result of execution of the precompile.

Verily,

WARLOCK

WARLOCK