Three Sigma

Code Audit

keyring

Keyring Zero-knowledge compliance solution

Disclaimer

Code Audit

Keyring Zero-knowledge compliance solution

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code’s manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Keyring Zero-knowledge compliance solution

Table of Contents

Summary

Code Audit

Keyring Zero-knowledge compliance solution

Summary

Three Sigma audited Keyring in a 4 days engagement. The audit was conducted from 10-07-2024 to 14-07-2024.

Protocol Description

Keyring is a Zero-Knowledge compliance solution for maximum liquidity and composability.

Scope

Code Audit

Keyring Zero-knowledge compliance solution

Scope

Filepath	nSLOC
src/base/KeyringCoreV2Base.sol	162
src/KeyringCoreV2.sol	21
src/unsafe/KeyringCoreV2Unsafe.sol	5
src/lib/RsaVerifyOptimized.sol*	234
Total	**422**

*diff

Assumptions

OpenZeppelin dependencies are considered safe.

The **`RsaVerifyOptimized.sol`** smart contract is based on this implementation and only the differences were reviewed.

Methodology

Code Audit

Keyring Zero-knowledge compliance solution

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Keyring Zero-knowledge compliance solution

Project Dashboard

Application Summary

Name	Keyring
Commit	bfe0c4a68c46230df0ec1d4ea0cf82e2e7b0ce0f
Fix Commit	acef8d6a69210b39ad29d940993429899aa7c227
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	10-07-2024 to 14-07-2024
№ of Auditors	1
Review Time	4 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	1	1	0
High	0	0	0
Medium	2	2	0

Low	3	2	1
None	6	5	1

Category Breakdown

Suggestion	7
Documentation	0
Bug	2
Optimization	0
Good Code Practices	3

Code Maturity Evaluation

Code Audit

Keyring Zero-knowledge compliance solution

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic
Centralization	Weak. Although the access control is very well implemented, there are several privileged roles.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. Transparent proxies are used for most contracts.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Front-Running	Satisfactory. There are no known front-running opportunities.
Monitoring	Satisfactory. Events are generally emitted when there are state changing occurrences.
Specification	Satisfactory. Functions are generally commented and the protocol has public documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

Keyring Zero-knowledge compliance solution

Findings

3S-Keyring-C01

Credentials can be manipulated

Id	3S-Keyring-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #af37b1d.

Description

`KeyringCoreV2::createCredential()` downcasts several variables, **`uint24(policyId), uint32(epoch), uint32(epochExp), uint168(cost)`** when sending them to **`RsaVerifyOptimized::verifyAuthMessage()`**, where they are encoded to a signature. Downcasting does not revert so if a key signs a message with a certain variable smaller than the casting that is being made, attackers can send this variable but with higher upper bits set to 1, such that the encoded message is still the same but the variable that is sent to **`KeyringCoreV2Base::_createCredential()`** is actually much bigger.

One example of an abuse is an attacker setting a much bigger **`epochExp`**, which is downcasted to the much lower value to sign, but sets the expiry of its **`_entityData[policyId][tradingAddress].exp`** to a value very far away in the future (up to **`uint64`**).

Another example is an attacker setting **`epoch`** to a much larger value such that it can set the credentials forever.

Recommendation

All the downcastings in the codebase should use the safe casting lib from OpenZeppelin.

Additionally, signatures could implement a nonce system to complement the deadline **`epoch`** argument.

3S-Keyring-M01

`RsaVerifyOptimized` sets the size of the key to 1024 bits, which is unsafe

Id	3S-Keyring-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Suggestion
Status	Addressed in #02b0fa2.

Description

RSA signatures rely on the size of the key for higher safety assurances. As computational power advances, it becomes possible to brute force signatures up to more bits. This article indicates keys up to 829 bits have been cracked, which is dangerously close to the hardcoded 1024 bits.

The original `RsaVerifyOptimized` repository recommends setting a key size of at least 2048 bits.

Recommendation

Update the key size to be 2048.

3S-Keyring-M02

`RsaVerifyOptimized::pkcs1Sha256()` modified the original code incorrectly in one instance

Id	3S-Keyring-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #02b0fa2.

Description

`RsaVerifyOptimized::pkcs1Sha256()` code has slight changes to the original code. There is one modification that is different from the original, which is checking if the first bytes of **`decipher`** are **`0x00`** and **`0x01`**, respectively. As can be seen in the following code snippet, the first 2 bytes of **`decipher`** may be for example **`0x0101`** and it will not set the result to false.

```
// if (decipher[0] != 0 || decipher[1] != 0x01) {
//     return false;
// }
if iszero(and(mload(add(decipher, 32)),
0x0001ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)) {
    result := false
}
```

Additionally, here should be 111, but the code is not reachable as it only accepts **`digestAlgoWithParamLen == 17 == sha256ExplicitNullParamByteLen`**.

Recommendation

Use the previous optimized assembly code.

3S-Keyring-L01

`KeyringCoreV2Base::collectFees()` should use **`Address::sendValue()`** instead of **`address.transfer()`**

Id	3S-Keyring-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #ffe8b11.

Description

`address.transfer()` hardcodes 2300 gas to forward, which may not be enough if gas costs change in the future or **`to`** is a smart contract wallet that does some logic in its **`receive()/fallback()`** function.

Recommendation

Use **`address::sendValue()`** from OpenZeppelin.

3S-Keyring-L02

Missing admin change events

Id	3S-Keyring-L02
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion
Status	Addressed in #ffe8b11.

Description

When setting the admin in **`KeyringCoreV2Base::setAdmin()`** and the constructor an event should be emitted.

Recommendation

Emit events on all relevant state variable changes.

3S-Keyring-L03

`KeyringCoreV2Base` could use a 2 step admin transfer mechanism

Id	3S-Keyring-L03
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Admin transfer in **`KeyringCoreV2Base`** is performed at once, possibly setting the admin to an incorrect address. To mitigate this risk, protocols usually implement a 2 step mechanism, where a pending admin is set who has to call **`acceptAdmin()`** in order for the admin to actually change. This ensures that the admin is always correctly set.

Recommendation

Implement the 2 step mechanism.

3S-Keyring-N01

`KeyringCoreV2Base::_createCredential()` could use `currentTime` instead of `block.timestamp` on the `creatBefore` check

Id	3S-Keyring-N01
Classification	None
Category	Good Code Practices
Status	Addressed in #ffe8b11.

Description

`KeyringCoreV2Base::_createCredential()` checks that the current **`block.timestamp`** has not exceeded **`creatBefore`** by doing:

```
if (block.timestamp > creatBefore) {
    revert ErrInvalidCredential(policyId, tradingAddress, "EPO");
}
```

It could use **`currentTime`** instead as it has been cached.

Recommendation

```
if (currentTime > creatBefore) {
    revert ErrInvalidCredential(policyId, tradingAddress, "EPO");
}
```

3S-Keyring-N02

`KeyringCoreV2Base::blacklistEntity()` and `KeyringCoreV2Base::unblacklistEntity()` could check that the addresses are not already set

Id	3S-Keyring-N02
Classification	None
Category	Suggestion
Status	Addressed in #ffe8b11.

Description

`KeyringCoreV2Base::blacklistEntity()` and **`KeyringCoreV2Base::unblacklistEntity()`** could perform additional checks such that the address is not already blacklisted or not. It has no impact but could be important to warn the admin about potential mistakes, as not reverting would likely go unnoticed.

Additionally, other permissioned functions such as **`KeyringCoreV2Base::registerKey()`** or **`KeyringCoreV2Base::revokeKey()`** perform similar checks.

Recommendation

Revert if the address is already blacklisted or not blacklisted.

3S-Keyring-N03

Admin checks in **`KeyringCoreV2Base`** could be placed in a modifier to increase readability and reduce code duplication

Id	3S-Keyring-N03
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

Usually access control is performed in modifiers, as is the case with **`onlyOwner`**, for example. The same could be applied to the admin check in each permissioned function to increase readability and decrease code duplication.

Recommendation

Create a modifier and implement the body of a function to reduce deployment costs (modifier's code is duplicated into each function which increases deployment costs significantly).

```
modifier onlyAdmin() {
    _onlyAdmin();
    _;
}
function _onlyAdmin() internal {
    if (msg.sender != _admin) {
        revert ErrCallerNotAdmin(msg.sender);
    }
}
```

3S-Keyring-N04

`KeyringCoreV2Base::registerKey()` could check that `validTo` is equal to or bigger than `block.timestamp`

Id	3S-Keyring-N04
Classification	None
Category	Suggestion
Status	Addressed in #ffe8b11.

Description

`KeyringCoreV2Base::registerKey()` registers a key with a validity between `validFrom` and `validTo`. It checks if `validTo <= validFrom`, but does not confirm that `validTo` is not set in the past.

Recommendation

```
if (validTo < block.timestamp) {
    revert ErrInvalidKeyRegistration("IVT");
}
```

3S-Keyring-N05

Functions can be marked **`external`** when only called from other contracts

Id	3S-Keyring-N05
Classification	None
Category	Suggestion
Status	Addressed in #ffe8b11.

Description

Functions that are never called internally should be marked **`external`**.

Recommendation

Set **`KeyringCoreV2Base::createCredential()`**, **`KeyringCoreV2Base::blacklistEntity()`**, **`KeyringCoreV2Base::unblacklistEntity()`**, **`KeyringCoreV2Base::collectFees()`** to external.

3S-Keyring-N06

Some natspec comments in **`KeyringCoreV2Base`** are outdated

Id	3S-Keyring-N06
Classification	None
Category	Good Code Practices
Status	Addressed in #ffe8b11.

Description

`EntityData` and **`KeyEntry`** structs have a natspec parameter named **`PADDING`** which is not present.

Recommendation

Remove the **`PADDING`** parameters from the comments.