Enclave

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Feb 03 2026

Enclave Audit Report

1 Executive Summary

1.1 Project Information

Description	Enclave is a verifiable multi-recipient asset allocation system based on zero-knowledge proofs.
Type	DeFi
Auditors	L1ght09,Sprig,Leon@BitsLab
Timeline	Thu Jan 22 2026 - Wed Jan 28 2026
Languages	Solidity
Platform	Others
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/enclave-hq/contracts
Commits	14a05ad4d766674d3e25be7d7fca12cbe66d711a f7a04817e86465f30af0475178ed68bcf629757e c17735dfec6802e3d66b7a1916334fedb0195c82

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
ZKP	src/ZKPay.sol	981445541dc7e49fa0b6a070572ce63871e7b7f4
ZKPV	src/ZKPayVault.sol	7f914371e89107069ef5615debc7697509ae2015
ICO	src/bychain/IntentConfig.sol	cd61077f9591f2c32c9b2edd6b9ef71c85b63e46
JLD	src/bychain/adapters/JustLendDelegate.sol	f5413b970456e39653403886a7a865fdd435ac60
AAVE3D	src/bychain/adapters/AAVEv3Delegate.sol	b3d87032eee2228743d2496253a6bc08788e04b5
TUUPS	src/bychain/TreasuryUUPS.sol	5063a9f87364615e6a3913f4313cea1c8757c0b4
TCC	src/bychain/TreasuryConfigCore.sol	7f49adfcdae4c20f800384b47308815ecd635611
IMA	src/bychain/IntentManager.sol	0b908a6c75d368bc9d376261f692db13180813ad

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	9	3
Critical	0	0	0
Major	0	0	0
Medium	3	3	0
Minor	6	4	2
Informational	3	2	1

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Enclave to identify any potential issues and vulnerabilities in the source code of the Enclave smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 12 issues of varying severity, listed below.

ID	Title	Severity	Status
ICO-2	`lifiDiamond` Address Not Initialized in `IntentConfig`	Minor	Fixed
IIC-1	Unused Functions	Informational	Fixed
TCC-3	Missing Setter of `legacyTokenIdToKey` Mapping	Minor	Fixed
ZKP-4	Unfair Gas Fee Deduction in `_calculateWithdrawFee()`	Minor	Acknowledged
ZKP-5	Inconsistent Logic Between Code and Comment	Informational	Fixed
ZKP-6	Single-step Ownership Transfer Can Be Dangerous	Minor	Acknowledged
IMA-10	Missing Validation in `deBridgeReceive()`	Medium	Fixed
TCC-12	Code Optimization	Minor	Fixed
ZKP1-7	Missing Validation of `tokenKey` in `executeCommitment()`	Medium	Fixed

ZKP1-8	Incomplete EOA Validation in `setSP1Verifier()` Function	Informational	Acknowledged
ZKP1-9	Lack of Events Emission	Minor	Fixed
ZKP1-11	Missing Reentrancy Guard and CEI Violation in `executeCommitment()` and `executeWithdraw()`	Medium	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Enclave Smart Contract :

Owner

- Owner can set SP1 verifier address through the `setSP1Verifier()` function.
- Owner can set submit proof verification key hash through the `setSubmitVkeyHash()` function.
- Owner can set withdraw proof verification key hash through the `setWithdrawVkeyHash()` function.
- Owner can set deposit governance contract address through the `setDepositGovernance()` function.
- Owner can manually manage token allowance through the `manualApprove()` function.
- Owner can transfer ownership through the `transferOwnership()` function.
- Owner can renounce ownership through the `renounceOwnership()` function.
- Owner can configure token through the `configureToken()` function.
- Owner can remove token through the `removeToken()` function.
- Owner can set general configuration through the `setConfig()` function.
- Owner can set address configuration through the `setAddressConfig()` function.
- Owner can set boolean configuration through the `setBoolConfig()` function.
- Owner can set string configuration through the `setStringConfig()` function.
- Owner can batch set configurations through the `batchSetConfigs()` function.
- Owner can set minimum deposit amount through the `setMinDepositAmount()` function.
- Owner can batch set minimum deposit amounts through the `setMinDepositAmountBatch()` function.

- Owner can set treasury address through the `setTreasury()` function.
- Owner can create deposits through the `createDeposit()` function.
- Owner can update the owner address through the `updateOwner()` function.
- Owner can request withdrawal of technical fees through the `requestTechFeeWithdraw()` function.
- Owner can set chain withdraw gas fees through the `setChainWithdrawGasFee()` function.
- Owner can set cross-chain fee basis points through the `setCrossChainFeeBps()` function.
- Owner can request withdrawal of withdraw fees through the `requestWithdrawFeeWithdraw()` function.
- Owner can set treasury config core through the `setTreasuryConfigCore()` function.
- Owner can pause the contract in emergency through the `emergencyPause()` function.
- Owner can unpause the contract through the `emergencyUnpause()` function.
- Owner can withdraw specified amount of token in emergency through the `emergencyWithdraw()` function.
- Owner can withdraw all tokens in emergency through the `emergencyWithdrawAll()` function.
- Owner can batch withdraw multiple tokens in emergency through the `emergencyWithdrawBatch()` function.
- Owner can execute payout (withdrawal) through the `payout()` function.
- Owner can withdraw all funds from lending pool through the `withdrawFromLendingPool()` function.
- Owner can batch withdraw funds from lending pool through the `withdrawFromLendingPoolBatch()` function.

- Owner can mark historical requests as processed through the `markRequestsAsProcessed()` function.
- Owner can transfer accumulated yield tokens through the `transferToken()` function.
- Owner can batch transfer multiple tokens through the `transferTokensBatch()` function.

Treasury

- Treasury can execute withdrawal operations through the `IntentManager::executeWithdraw()` function.

DeBridge Gate

- DeBridge Gate can receive cross-chain funds through the `deBridgeReceive()` function.

User

- User can execute commitment with ZK proof through the `ZKPay::executeCommitment()` function.
- User can execute withdraw with ZK proof through the `ZKPay::executeWithdraw()` function.
- User can refund high risk deposits through the `refundHighRiskDeposit()` function.
- User can deposit tokens through the `deposit()` function.

4 Findings

ICO-2 `lifiDiamond` Address Not Initialized in `IntentConfig`

Severity: Minor

Status: Fixed

Code Location:

src/bychain/IntentConfig.sol#27

Descriptions:

The `IntentConfig` contract manages configuration for cross-chain operations. It stores important addresses like bridge integrations. The contract has a state variable called `lifiDiamond`. This variable is declared but never assigned a value. The constructor and the internal `_initializeDefaults()` function do not set the `lifiDiamond` address. Any function that reads this variable will get the default value of a zero address. This means the system cannot correctly interact with the Li.Fi bridge service. It will cause failures in any feature that depends on this address.

Suggestion:

It is recommended to initialize the `lifiDiamond` state variable. A setter function controlled by the owner can be added, or the address can be set in the constructor.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

IIC-1 Unused Functions

Severity: Informational

Status: Fixed

Code Location:

src/bychain/IIntentConfig.sol#457-564

Descriptions:

The `_bytesToAddress()` function in the `IntentManager` contract is designed to convert a bytes input into an address type. This function uses inline assembly to perform the conversion. However, the function is marked with an audit comment indicating it is unused. An unused function increases the contract size unnecessarily and may lead to confusion during maintenance. This results in higher deployment costs and potential inefficiencies in the contract code.

Suggestion:

It is recommended to remove the unused `_bytesToAddress()` function from the contract to reduce code complexity and deployment costs.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TCC-3 Missing Setter of `legacyTokenIdToKey` Mapping

Severity: Minor

Status: Fixed

Code Location:

src/bychain/TreasuryConfigCore.sol#24

Descriptions:

The `TreasuryConfigCore` contract manages token configurations. It contains a mapping variable called `legacyTokenIdToKey`. This mapping is intended to store a relationship between legacy token IDs and their corresponding token keys. The contract provides a function to read data from this mapping. However, there is no function in the contract to write or initialize data into this `legacyTokenIdToKey` mapping. This means the mapping will always be empty. Any function or system that relies on reading a valid token key from this mapping will fail because it will not find any data.

Suggestion:

It is recommended to add a function to initialize or update the `legacyTokenIdToKey` mapping.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ZKP-4 Unfair Gas Fee Deduction in `_calculateWithdrawFee()`

Severity: Minor

Status: Acknowledged

Code Location:

src/ZKPayVault.sol#120-158

Descriptions:

The `_calculateWithdrawFee()` function in the ZKPayVault contract calculates the gas fee and cross-chain fee for a withdrawal. When the user's withdrawal amount is less than the gas fee, the function sets the entire amount as the gas fee, leaving the user with zero actual amount. This is unfair because the user loses all their funds without receiving any service. The issue occurs because the function does not revert or adjust the fee when the amount is insufficient to cover the gas fee.

Suggestion:

It is recommended to revert the transaction or adjust the gas fee when the withdrawal amount is less than the gas fee to prevent unfair fund loss.

Resolution:

The client replied: The system has addressed this issue at the source through the minimum deposit amount restriction.

ZKP-5 Inconsistent Logic Between Code and Comment

Severity: Informational

Status: Fixed

Code Location:

src/ZKPayVault.sol#336-348

Descriptions:

It is recommended to update the comment to accurately reflect the actual "priority-based deduction" logic implemented in the code, or alternatively, modify the code to implement a true proportional deduction as originally described in the comment.

Suggestion:

It is recommended to review the design and implementation of this balance deduction logic. Ensure that the code's behavior aligns with the intended business rules as documented in the comments or specifications.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ZKP-6 Single-step Ownership Transfer Can Be Dangerous

Severity: Minor

Status: Acknowledged

Code Location:

src/ZKPayVault.sol#237-246;

src/bychain/IntentManager.sol#20;

src/bychain/IntentConfig.sol#12;

src/bychain/TreasuryConfigCore.sol#12;

src/bychain/TreasuryUUPS.sol#22

Descriptions:

The `updateMultisigOwner()` function in `ZKPayVault.sol` allows the current multisig owner to transfer ownership to a new address in a single step. This function is used to update the multisig owner address that controls critical vault operations. However, the transfer occurs immediately without a confirmation step from the new owner. If the new address is entered incorrectly—such as a typo or an address the owner does not control—the ownership will be permanently lost to an unreachable address. This can result in the loss of administrative control over the vault, making it impossible to perform any further privileged actions, including future ownership transfers or security updates.

Additionally, `IntentManager.sol#L20` , `IntentConfig.sol#L12` , `TreasuryConfigCore.sol#L12` , `TreasuryUUPS.sol#L22` , these contracts inherit from `Ownable` or `OwnableUpgradeable` contracts. Since the `Ownable` and `OwnableUpgradeable` contract also follow a single-step transfer model, the risks associated with single-step transfer should also be considered.

Suggestion:

It is recommended to implement a two-step ownership transfer process, where the new owner must explicitly accept the ownership to complete the transfer. This prevents accidental loss of ownership due to address errors.

IMA-10 Missing Validation in `deBridgeReceive()`

Severity: Medium

Status: Fixed

Code Location:

src/bychain/IntentManager.sol#380,396

Descriptions:

The `deBridgeReceive()` function in the IntentManager contract handles cross-chain token transfers. It decodes parameters from the payload but does not validate if the decoded `sourceChainId` matches the function parameter `_srcChainId`. It also does not check if `_srcAddress` is a registered IntentManager address. This lack of validation could allow malicious actors to supply inconsistent data, potentially leading to incorrect fund routing or unauthorized operations.

Suggestion:

It is recommended to add validation checks in the `deBridgeReceive()` function to ensure that the payload parameters are consistent with the function arguments and the registered contract addresses.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TCC-12 Code Optimization

Severity: Minor

Status: Fixed

Code Location:

src/bychain/TreasuryConfigCore.sol#116

Descriptions:

The `removeToken()` method uses a swap-and-pop operation to remove elements from the `allTokenKeys` array, but it doesn't check if the element to be removed is already the last one, leading to unnecessary self-assignment.

Suggestion:

It is recommended to check if the element to be removed is already the last one in the `removeToken()` method.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ZKP1-7 Missing Validation of `tokenKey` in `executeCommitment()`

Severity: Medium

Status: Fixed

Code Location:

src/ZKPay.sol#515-589

Descriptions:

The `executeCommitment()` function in the ZKPay contract processes commitments from zero-knowledge proofs. It decodes public values from the proof and validates multiple fields to ensure secure execution. However, the function does not validate the `tokenKey` field from the decoded public values. This allows a non-normalized or mismatched `tokenKey` to be accepted, which can lead to incorrect token handling. Without validation, the contract may process commitments for unintended tokens, resulting in financial losses or protocol instability.

Suggestion:

It is recommended to add validation for the `tokenKey` field to ensure it matches the expected token key for the deposit.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ZKP1-8 Incomplete EOA Validation in `setSP1Verifier()` Function

Severity: Informational

Status: Acknowledged

Code Location:

src/ZKPay.sol#182-195

Descriptions:

The `setSP1Verifier()` function in the `ZKPay` contract is used by the contract owner to set the address of the SP1 verifier contract. The function includes a check to ensure the provided address is not an Externally Owned Account (EOA) by verifying that `addr.code.length` is not zero. However, this validation is incomplete. Under EIP-7702, an EOA can enable special functionality, causing its `addr.code.length` to return a non-zero value (specifically 23). This loophole allows an EOA with EIP-7702 enabled to bypass the intended check and be incorrectly set as the verifier.

Suggestion:

is recommended to implement a more robust validation method to definitively distinguish between contract accounts and EOA accounts, even those with EIP-7702 enabled, before setting the verifier address.

ZKP1-9 Lack of Events Emission

Severity: Minor

Status: Fixed

Code Location:

src/ZKPay.sol#210,226

Descriptions:

The `setSubmitVkeyHash()` and `setWithdrawVkeyHash()` functions in the `ZKPay` contract are used by the contract owner to update the critical verification key hash. The key hash is essential for the protocol's zero-knowledge proof verification process. However, these functions change important state variables without emitting events to log the change. This makes it difficult for users and external systems to track when and to what value the `submitVkeyHash` and `withdrawVkeyHash` were updated. Without the event log, there is no transparent, on-chain record of this administrative action, which reduces accountability and makes off-chain monitoring or reacting to such changes impossible.

Suggestion:

It is recommended to emit events within the `setSubmitVkeyHash()` and `setWithdrawVkeyHash()` functions. The event should include both the old value and the new value as parameters.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ZKP1-11 Missing Reentrancy Guard and CEI Violation in `executeCommitment()` and `executeWithdraw()`

Severity: Medium

Status: Fixed

Code Location:

src/ZKPay.sol#532,612

Descriptions:

The `executeCommitment()` function in the ZKPay contract processes user commitments after verifying a zero-knowledge proof. It first performs several checks on the commitment and deposit data. After these checks, it calls an external contract `ISP1Verifier.verifyProof()` to verify the proof. Only after this external call does it update the contract's internal state, such as marking a commitment as used.

This function lacks a reentrancy guard modifier (`nonReentrant`). More critically, it performs an external call (`verifyProof`) before updating its own state (like setting `$.availableCommitments[publicValues.commitment] = true`). This violates the Checks-Effects-Interactions (CEI) pattern, a best practice for secure smart contract development.

An attacker could potentially create a malicious verifier contract that re-enters the `executeCommitment()` function during the `verifyProof` call. Because the state (marking the commitment as used) hasn't been updated yet, the re-entrant call could pass the initial validation checks again, leading to double-spending of the same commitment or other state inconsistencies. The `executeWithdraw()` function has the same vulnerability.

Suggestion:

It is recommended to add the `nonReentrant` modifier to both `executeCommitment()` and `executeWithdraw()` functions. Furthermore, the code should be refactored to follow the CEI pattern: update all critical internal state variables (like marking commitments or deposits as used) before making any external calls.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.