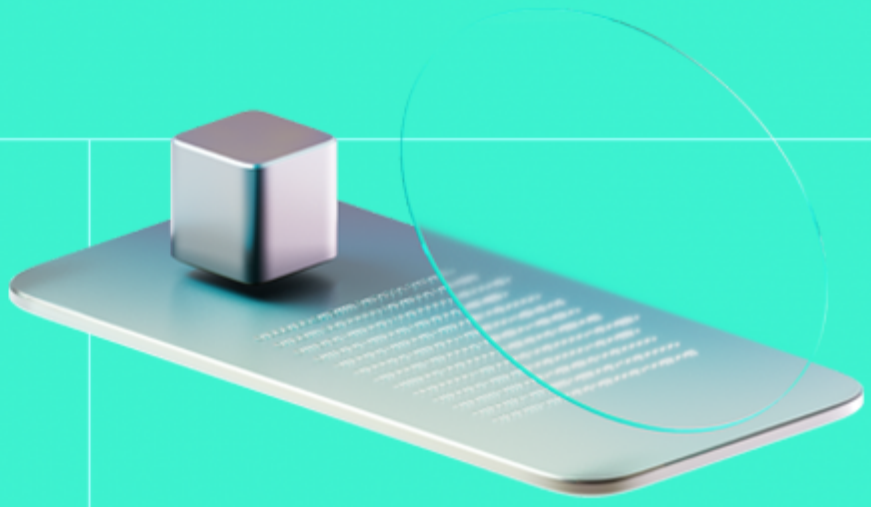
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Dexlyn

Date: 12/02/2025

We express our gratitude to the Dexlyn team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Dexlyn is a dynamic DeFi platform on the Supra Chain, offering secure and efficient token swaps and liquidity management through its automated market maker (AMM) technology. Beyond these capabilities, Dexlyn is evolving into a comprehensive DeFi ecosystem, with plans to integrate advanced functionalities, empowering users with a wide range of decentralized financial solutions. Hyperlane, an open-source interoperability framework, enhances Dexlyn's ecosystem by enabling seamless cross-chain communication and deployment. It connects over 100 blockchains, utilizing customizable security modules and multi-VM support to drive innovation and scalability.

Document

Name	Smart Contract Code Review and Security Analysis Report for Dexlyn
Audited By	Jakub Heba
Approved By	Przemyslaw Swiatowiec
Website	dexlyn.com
Changelog	03/01/2025 - Preliminary Report, 13/01/2025 - Final Report, Updated Final report- 12/02/2025
Platform	Supra
Language	MOVE
Tags	cross-chain
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/DexlynLabs/hyperlane-monorepo
Commit	8d4b098701f84dc440ed88a6dfc2ce61e0ca9067

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

6	5	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	2
High	0
Medium	1
Low	3

Vulnerability	Severity
F-2025-8182 - Incorrect message signature verification parameters	Critical
F-2025-8185 - Incorrect token amount calculation during transfer	Critical
F-2025-8735 - Malicious attacker can make messages sent through the bridge unfinalized	Medium
F-2025-8047 - Unused variable	Low
F-2025-8048 - One-way ownership transfer pattern is Unsafe	Low
F-2025-8049 - In an edge case transfer will revert despite having sufficient funds	Low

Documentation quality

- Functional requirements are partially missed.
- Technical description is not provided.

Code quality

- The code duplicates commonly known contracts instead of reusing them.
- Several template code patterns were found.
- The development environment is configured.

Test coverage

Code coverage cannot be calculated, as only E2E tests were provided, which are covering basic contract iterations only.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

Table of Contents

System Overview

Dexlyn Hyperlane on Aptos implementation consists of the following components:

IGPS (Interchain Gas Payment Service) module manages cross-chain gas payments, allowing users to pay for transaction gas fees on destination chains using the source chain's native token, with features for gas price conversion based on exchange rates and beneficiary management. It is also using the Gas Oracle.

Multisig ISM (Interchain Security Module) contract implements a multi-signature verification scheme for cross-chain messages, allowing a configurable threshold of validators from different domains to sign and verify message authenticity, with features for validator management and ownership control.

Mailbox module serves as the core messaging infrastructure for cross-chain communication, handling message dispatch, verification, and delivery between different domains with features like message merkleization, gas payment integration, and delivery status tracking.

Router module manages cross-chain communication routing by maintaining registries of remote router addresses and providing functionality to enroll and verify router addresses across different domains, with features for ownership management and type-safe router initialization.

Synthetic tokens - Both modules implement cross-chain token bridge functionality, with different token management approaches:

- **HyperCoin** - Implements a synthetic token bridge that can mint and burn tokens, using capabilities for token management.
- **HyperCoinCollateral** - Implements a collateral-based token bridge that locks tokens in a resource account instead of burning, using existing token type FusionCoin.

Validator Announce module manages validator announcements and their storage locations, implementing signature verification for announcements and preventing replay attacks, with functionality to track and query validator registrations and their associated storage metadata.

Privileged roles

- **Validators** - Sign and verify cross-chain messages using multi-signature scheme, announce their storage locations
- **Owner/Admin** - Controls protocol configuration, can update gas prices, beneficiary addresses, and validator settings
- **Relayers** - Pay gas fees and relay messages between chains
- **Beneficiary** - Receives collected gas fees from interchain gas payments

Potential Risks

No additional risks have been found.

Findings

Vulnerability Details

F-2025-8182 - Incorrect message signature verification parameters - Critical

Description:

The code implementation uses an incorrect parameter combination in the signature verification process. The `verify` function in `move/isms/sources/multisig_ism.move` is using `msg_utils::nonce(message)` instead of the proper `merkle_index` when generating the signed digest bytes, which would cause signature validation to fail.

Additionally, the `merkle_index` was being extracted incorrectly as raw bytes rather than being properly converted to a u32 value.

```
/// Requires that m-of-n validators verify a merkle root,
  /// and verifies a merkle proof of `message` against that root.
  public fun verify(
    metadata: &vector<u8>,
    message: &vector<u8>,
  ): bool acquires ISM {
    let state = borrow_global<ISM>(@hp_isms);

    let origin_mailbox = ism_metadata::origin_mailbox(metadata);
    let origin_domain = msg_utils::origin_domain(message);

    let merkle_root = ism_metadata::merkle_root(metadata);
    let merkle_index = ism_metadata::merkle_index(metadata);
    let signed_digest_bytes = utils::eth_signed_message_hash(&utils::ism_chec
kpoint_hash(
      origin_mailbox,
      origin_domain,
      merkle_root,
      msg_utils::nonce(message),
      msg_utils::id(message)
    ));


    let domain_validators = simple_map::borrow(&state.validators_per_domain,
&origin_domain);
    let validator_count = vector::length(&domain_validators.validators);
    let validator_index = 0;
```

The issue itself was found by the Client team during the audit process.

Status: Fixed

Classification

Impact: 4/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Medium

Severity: Critical

Recommendations

Remediation: The fix should implement two key changes:

1. Convert the merkle index bytes to a u32 value,
2. Use `merkle_index` instead of `nonce` in the `ism_checkpoint_hash` calculation.

Resolution: Issue was fixed in `7908859` through implementation of `merkle_index_u32` function and using `merkle_index` instead of `nonce` in the `ism_checkpoint_hash` calculation.

F-2025-8185 - Incorrect token amount calculation during transfer - Critical

Description:

In the `transfer_remote` and `transfer_remote_with_gas` functions of both `hyper_coin_collateral.move` and `hyper_coin.move`, there is an issue in the token amount calculation when source decimals are greater than destination decimals.

The code is incorrectly assigning the scaled-down `data_amount` directly to `amount`, resulting in users being charged less tokens than they should be on the source chain.

```
    /// This function helps to transfer funds from source chain to destinatio
n chain without paying for destination gas
    public entry fun transfer_remote(
        account: &signer,
        dest_domain: u32,
        dest_receipient: vector<u8>,
        amount: u64) acquires CoinCapability, State {
        let state = borrow_global_mut<State>(@synthetic_tokens);
        let data_amount: u256;
        let source_decimals = coin::decimals<HyperSupraCoin>();
        /// assert for destination decimals for graceful exit
        assert!(table::contains(&state.destination_decimals,dest_domain),erro
r::not_found(EDESTINATION_DECIMAL_NOT_SET));
        let destination_decimals = *table::borrow(&state.destination_decimals
, dest_domain);
        if (source_decimals < destination_decimals) {
            data_amount = (amount as u256) * calculate_power(10, ((destinatio
n_decimals - source_decimals) as u16));
        }
        else if (source_decimals == destination_decimals) {
            data_amount = (amount as u256);
        }
        else {
            data_amount = (amount as u256) / calculate_power(10, ((source_dec
imals - destination_decimals) as u16));
            amount = (data_amount as u64);
            assert!(data_amount > 0, EAMOUNT_TOO_SMALL);
        };
        let sender = signer::address_of(account);
        let caps = borrow_global<CoinCapability>(@synthetic_tokens);
        coin::burn_from<HyperSupraCoin>(sender, amount, &caps.burn_cap);
        state.last_id = mailbox::dispatch<HyperSupraCoin>(
            dest_domain,
            token_msg_utils::format_token_message_into_bytes(
```

```
                h256::from_bytes(&dest_receipient),
                data_amount,
                dest_receipient
            ),
            &state.cap
        );
        emit(SentTransferRemote{destination:dest_domain,receipient:dest_recei
pient,amount});
    }

    /// This function helps to transfer funds from source chain to destinatio
n chain by paying the destination gas
    public entry fun transfer_remote_with_gas(
        account: &signer,
        dest_domain: u32,
        dest_receipient: vector<u8>,
        amount: u64) acquires CoinCapability, State {
        let state = borrow_global_mut<State>(@synthetic_tokens);
        let data_amount: u256;
        let source_decimals = coin::decimals<HyperSupraCoin>();
        /// assert for destination decimals for graceful exit
        assert!(table::contains(&state.destination_decimals,dest_domain),erro
r::not_found(EDESTINATION_DECIMAL_NOT_SET));
        let destination_decimals = *table::borrow(&state.destination_decimals
, dest_domain);
        if (source_decimals < destination_decimals) {
            data_amount = (amount as u256) * calculate_power(10, ((destinatio
n_decimals - source_decimals) as u16));
        }
        else if (source_decimals < destination_decimals) {
            data_amount = (amount as u256);
        }
        else {
            data_amount = (amount as u256) / calculate_power(10, ((source_dec
imals - destination_decimals) as u16));
            amount = (data_amount as u64);
            assert!(data_amount > 0, EAMOUNT_TOO_SMALL);
        };
        let sender = signer::address_of(account);
        let caps = borrow_global<CoinCapability>(@synthetic_tokens);
        coin::burn_from(sender, amount, &caps.burn_cap);
        state.last_id = mailbox::dispatch_with_gas<HyperSupraCoin>(
            account,
            dest_domain,
            token_msg_utils::format_token_message_into_bytes(
                h256::from_bytes(&dest_receipient),
                data_amount,
                dest_receipient
            ),
```

```
                DEFAULT_GAS_AMOUNT,
                &state.cap
            );
            emit(SentTransferRemote{destination:dest_domain,receipient:dest_recei
pient,amount});
        }
```

The issue itself was found by the Client team during the audit process.

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Simple

Severity: Critical

Recommendations

Remediation: It is recommended to ensure that proper token amounts are used in the calculation of scaled `data_amount` for the destination chain, checking, if it is greater than zero and staling the amount back up for the source chain deduction.

Resolution: Issue was fixed in `7908859` through implementation of all steps mentioned in the Remediation section.

F-2025-8735 - Malicious attacker can make messages sent through the bridge unfinalized - Medium

Description:

It was identified that on the MOVE side the relayer directly calls `handle_message` in the recipient contract address and the recipient contract in turns calls `handle_message` of the mailbox for verification and validation. However, there is no way to assert in the mailbox that the message that is going to get verified was actually being called from the intended recipient contract or not.

As a consequence, a potential attacker is able to create a malicious contract, handling the call to `handle_message`, which will be called by him in the context of a completely correct message. The recipient is not verified - it may therefore be different than the intended message recipient.

After validation from the mailbox side, the status of the ID of the message will be changed to `delivered`, which will prevent the original relayer from calling `handle_message` on this particular message. Then, the malicious attacker may not finish processing the message on the bridge, which will result in an unsuccessful operation, for example bridging of tokens.

The issue itself was found by the Client team after the audit process.

Status: Fixed

Classification

Impact: 3/5

Likelihood: 1/5

Exploitability: Semi-Dependent

Complexity: Complex

Severity: Medium

Recommendations

Remediation: We recommend implementation of validation on the mailbox side checking, if the recipient of the `handle_message` call is the same as specified in the message.

Resolution: The issue was fixed in the `4399a55a5f5f7d585606b38c826e6819fcd28fff` by asserting the recipient of the message.

F-2025-8047 - Unused variable - Low

Description:

In `move/synthetic-tokens/sources/hyper_coin.move` and `move/tokens/sources/hyper_coin_collateral.move`, the `State` struct contains an unused vector field `received_messages` that stores message data but is never accessed or utilized in the contract logic.

The current implementation:

```
struct State has key {
  cap: router::RouterCap<HyperSupraCollateral>,
  destination_decimals: Table<u32, u8>,
  received_messages: vector<vector<u8>>,
  signer_cap: SignerCapability,
  last_id: vector<u8>
}
```

While the severity is low, this redundant code unnecessarily increases contract size and may cause confusion during code maintenance.

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Complex

Severity: Low

Recommendations

Remediation: Remove the unused `received_messages` vector field from the `State` struct since it serves no purpose in the contract's functionality.

Resolution: Issue was fixed in `7908859` through removal of `received_messages` parameter from the `State` struct.

F-2025-8048 - One-way ownership transfer pattern is Unsafe - Low

Description:

In `router.move`, `mailbox.move`, and `multisig_ism.move`, the ownership transfer pattern implemented in functions like `transfer_ownership()` uses a single-step process where ownership is directly transferred to a new address. If the new owner address is incorrectly specified (typo, wrong address format, etc.), ownership could be permanently lost with no way to recover.

Current implementation:

```
public fun transfer_ownership<T>(account: &signer, new_owner: address) acquir
es RouterRegistry {
   let account_address = signer::address_of(account);
   assert_owner_address<T>(account_address);
   let registry = borrow_global_mut<RouterRegistry>(@hp_router);
   let state = simple_map::borrow_mut(&mut registry.router_state_map, &type_i
nfo::type_of<T>());
   state.owner_address = new_owner;
}
```

This can cause potential permanent loss of contract ownership if wrong address is provided.

Status: Fixed

Classification

Impact: 2/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Implement a two-step ownership transfer pattern similar to OpenZeppelin's `Ownable2Step`.

Resolution: Fixed in `7908859` through the implementation of 2-step ownership transfer mechanism.

F-2025-8049 - In an edge case transfer will revert despite having sufficient funds - Low

Description: In `move/igps/sources/igps.move`, the `pay_for_gas` function's balance check uses a strict greater than (>) comparison instead of greater than or equal to (>=). This causes transactions to revert when a user has exactly the required amount:

```
assert!(coin::balance<AptosCoin>(account_address) > required_amount, ERROR_INSUFFICIENT_INTERCHAIN_GAS);
```

Transactions will fail even when users have exactly the required amount of funds to pay for interchain gas. While not a security vulnerability, this creates unnecessary friction and may confuse users who have allocated precisely the required amount.

Status: Accepted

Classification

Impact: 1/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Update the balance check to use greater than or equal to (>=) to allow transactions with exact amounts.

Resolution: Client accepted the issue with the following note:

> Thanks for point this out but we feel its always safe to have a balance more than whats required just to avoid getting errors (can be caused by difference in exchange rate or gas price) that we might not get during the simulation phase but during the execution phase

Observation Details

F-2025-8050 - Missing event emissions - Info

Description:

Across multiple files in the codebase, state-changing functions like `init_module()` and `transfer_ownership()` do not emit events. State modifications such as initialization operations and ownership transfers occur without any on-chain logging, which reduces transparency.

The lack of events makes it harder to track state changes off-chain. This affects the ability to monitor contract behavior, debug issues, and build indexing solutions. Contract initialization and ownership changes are not visible to external observers and interfaces.

Status: Fixed

Recommendations

Remediation: Add event emissions for state changes by implementing event structs and emitting them in relevant functions.

Resolution: The issue was fixed in `7908859` through the proper events implementation for most of the `init_module` functions and in `transfer_ownership` functions as well.

F-2025-8051 - Unfinished development TODOs in codebase - Info

Description: In `move/validator-announce/sources/announce.move:126` and `move/examples/sources/hello_world.move:1`, TODO comments indicate incomplete or unfinished development work. These comments suggest features or improvements that were planned but not implemented.

The presence of TODO comments points to potential gaps in functionality and incomplete implementations. This affects code maintainability and may indicate missing features or unhandled edge cases.

Status: Fixed

Recommendations

Remediation: Review and resolve all TODO comments. Implement the missing functionality described in the comments.

Resolution: Issue was fixed in `7908859` through the TODO comments removal.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/DexlynLabs/hyperlane-monorepo
Commit	8d4b098701f84dc440ed88a6dfc2ce61e0ca9067
Whitepaper	-
Requirements	https://github.com/DexlynLabs/hyperlane-monorepo/README.md
Technical Requirements	-

Asset	Type
move/examples/sources/hello_world.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/igps/sources/events.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/igps/sources/gas_oracle.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/igps/sources/igps.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/isms/sources/multisig_ism.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/h256.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/ism_metadata.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/merkle_tree.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/msg_utils.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/token_msg_utils.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/library/sources/utils.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/mailbox/sources/events.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/mailbox/sources/mailbox.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/router/sources/events.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/router/sources/router.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract

Asset	Type
move/synthetic-tokens/sources/hyper_coin.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/tokens/sources/hyper_coin_collateral.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/validator-announce/sources/announce.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract
move/validator-announce/sources/events.move [https://github.com/DexlynLabs/hyperlane-monorepo]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.