SHERLOCK

# Security Review For Hyperlane

Collaborative Audit Prepared For: **Hyperlane**
Lead Security Expert(s): **bin2chen**
**S3v3ru5**

Date Audited: **February 6 - February 27, 2025**
Final Commit: **84f886b**

## Introduction

Hyperlane is a permissionless interoperability protocol for cross-chain communication. It enables abstract message passing and asset transfers across different chains, allowing the security to be customized by applications and for anyone to bring Hyperlane to any chain.

## Scope

Repository: hyperlane-xyz/hyperlane-monorepo

Audited Commit: 84f886b7b611a845064822d7dbe6e70396c00523

Final Commit: 84f886b7b611a845064822d7dbe6e70396c00523

Files:

- rust/sealevel/libraries/access-control/src/lib.rs
- rust/sealevel/libraries/account-utils/src/discriminator.rs
- rust/sealevel/libraries/account-utils/src/lib.rs
- rust/sealevel/libraries/ecdsa-signature/src/lib.rs
- rust/sealevel/libraries/hyperlane-sealevel-connection-client/src/gas_router.rs
- rust/sealevel/libraries/hyperlane-sealevel-connection-client/src/lib.rs
- rust/sealevel/libraries/hyperlane-sealevel-connection-client/src/router.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/accounts.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/error.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/instruction.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/lib.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/message.rs
- rust/sealevel/libraries/hyperlane-sealevel-token/src/processor.rs
- rust/sealevel/libraries/interchain-security-module-interface/src/lib.rs
- rust/sealevel/libraries/message-recipient-interface/src/lib.rs
- rust/sealevel/libraries/multisig-ism/Cargo.toml
- rust/sealevel/libraries/multisig-ism/src/error.rs
- rust/sealevel/libraries/multisig-ism/src/interface.rs
- rust/sealevel/libraries/multisig-ism/src/lib.rs
- rust/sealevel/libraries/multisig-ism/src/multisig.rs
- rust/sealevel/libraries/multisig-ism/src/test_data.rs

- rust/sealevel/libraries/serializable-account-meta/src/lib.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/accounts.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/error.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/instruction.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/lib.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/pda_seeds.rs
- rust/sealevel/programs/hyperlane-sealevel-igp/src/processor.rs
- rust/sealevel/programs/hyperlane-sealevel-token-collateral/src/instruction.rs
- rust/sealevel/programs/hyperlane-sealevel-token-collateral/src/lib.rs
- rust/sealevel/programs/hyperlane-sealevel-token-collateral/src/plugin.rs
- rust/sealevel/programs/hyperlane-sealevel-token-collateral/src/processor.rs
- rust/sealevel/programs/hyperlane-sealevel-token-native/src/instruction.rs
- rust/sealevel/programs/hyperlane-sealevel-token-native/src/lib.rs
- rust/sealevel/programs/hyperlane-sealevel-token-native/src/plugin.rs
- rust/sealevel/programs/hyperlane-sealevel-token-native/src/processor.rs
- rust/sealevel/programs/hyperlane-sealevel-token/src/instruction.rs
- rust/sealevel/programs/hyperlane-sealevel-token/src/lib.rs
- rust/sealevel/programs/hyperlane-sealevel-token/src/plugin.rs
- rust/sealevel/programs/hyperlane-sealevel-token/src/processor.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/accounts.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/error.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/instruction.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/lib.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/metadata.rs
- rust/sealevel/programs/ism/multisig-ism-message-id/src/processor.rs
- rust/sealevel/programs/mailbox/src/accounts.rs
- rust/sealevel/programs/mailbox/src/error.rs
- rust/sealevel/programs/mailbox/src/instruction.rs
- rust/sealevel/programs/mailbox/src/lib.rs
- rust/sealevel/programs/mailbox/src/pda_seeds.rs
- rust/sealevel/programs/mailbox/src/processor.rs

- rust/sealevel/programs/mailbox/src/protocol_fee.rs
- rust/sealevel/programs/validator-announce/src/accounts.rs
- rust/sealevel/programs/validator-announce/src/error.rs
- rust/sealevel/programs/validator-announce/src/instruction.rs
- rust/sealevel/programs/validator-announce/src/lib.rs
- rust/sealevel/programs/validator-announce/src/pda_seeds.rs
- rust/sealevel/programs/validator-announce/src/processor.rs

## Final Commit Hash

84f886b7b611a845064822d7dbe6e70396c00523

## Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

## Issues Found

| High | Medium | Low/Info |
|---|---|---|
| 0 | 0 | 8 |

## Issues Not Fixed and Not Acknowledged

| High | Medium | Low/Info |
|---|---|---|
| 0 | 0 | 0 |

Please note: Any submission in this report marked as Disputed or Won't Fix does not necessarily reflect their literal interpretation.

These designations indicate that while the client acknowledges the submission, it may have been classified this way due to: (1) being primarily informational in nature, (2) having minimal security impact, or (3) considerations related to the security review timeline.

Specifically for submissions 65, 66, and 67 marked as 'Won't Fix', the client intends to implement these fixes in the near future, but they were not deemed immediate security risks that would block completion of the current audit.

# Issue L-1: create_replay_protection_account() does not execute store()

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/60

## Summary

`validator-announce##process_announce()` generates a pda to prevent replay by `create_replay_protection_account()`, but this method forgets `store()`.

## Vulnerability Detail

Here is the implementation of `create_replay_protection_account()`.

```
fn create_replay_protection_account<'a>(
    program_id: &Pubkey,
    payer_info: &AccountInfo<'a>,
    system_program_info: &AccountInfo<'a>,
    replay_protection_info: &AccountInfo<'a>,
    replay_id: [u8; 32],
    replay_protection_bump_seed: u8,
) -> Result<(), ProgramError> {
    let replay_protection_account =
↪   ReplayProtectionAccount::from(ReplayProtection(()));
    let replay_protection_account_size = replay_protection_account.size();

    // Create the account.
    create_pda_account(
        payer_info,
        &Rent::get()?,
        replay_protection_account_size,
        program_id,
        system_program_info,
        replay_protection_info,
        replay_protection_pda_seeds!(replay_id, replay_protection_bump_seed),
    )?;

    Ok(())
}
```

As we can see, there is no execution of `replay_protection_account.store()` and the account data is still in an uninitialized state. (But the data is already allocated space)

## Impact

Due to checking replay with `replay_protection_info.data_is_empty()` the data is not initialized, But the data is already allocated space `data.len() > 0`, so it does not affect the function. However, it is recommended to store it, so that the program can be extended in the future.

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/blob/0053d8e839a245335544b9b920b30a8b48e3b33d/hyperlane-monorepo/rust/sealevel/programs/validator-announce/src/processor.rs#L316

## Tool Used

Manual Review

## Recommendation

```
fn create_replay_protection_account<'a>(
...
    let replay_protection_account =
↪   ReplayProtectionAccount::from(ReplayProtection(()));
    let replay_protection_account_size = replay_protection_account.size();

    // Create the account.
    create_pda_account(
        payer_info,
        &Rent::get()?,
        replay_protection_account_size,
        program_id,
        system_program_info,
        replay_protection_info,
        replay_protection_pda_seeds!(replay_id, replay_protection_bump_seed),
    )?;
+   replay_protection_account.store(replay_protection_info, false)?;
```

and test file `validator-announce::tests::functional.rs` there are also some mistakes.

```
async fn assert_successful_announcement(
    banks_client: &mut BanksClient,
    program_id: Pubkey,
    validator_storage_locations_key: Pubkey,
    replay_protection_key: Pubkey,
    expected_validator_storage_locations: ValidatorStorageLocations,
) {
```

```
...

    // Expect the replay protection account to be created
    let replay_protection_account = banks_client
        .get_account(replay_protection_key)
        .await
        .unwrap()
        .unwrap();
    assert_eq!(replay_protection_account.owner, program_id);

-   assert!(!validator_storage_locations_account.data.is_empty());
+   assert!(!replay_protection_account.data.is_empty());
    let replay_protection =
-        ReplayProtectionAccount::fetch_data(&mut
↪   &validator_storage_locations_account.data[..])
+        ReplayProtectionAccount::fetch_data(&mut
↪   &replay_protection_account.data[..])
            .unwrap();
    assert_eq!(replay_protection, Some(Box::new(ReplayProtection(()))),);
}
```

# Issue L-2: set_validators_and_threshold() suggests adding a check for rent exempt

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/61

## Summary

When `set_validators_and_threshold()` is saved, it is possible to extend the space if `validators > 50+`, but currently there is no guarantee that `rent exempt`

## Vulnerability Detail

`set_validators_and_threshold()` creates a pda and pays a rent of `domain_pda_size = 1024`. Subsequent additions of `validators` will extend the space if needed.

```
fn set_validators_and_threshold(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    config: Domained<ValidatorsAndThreshold>,
) -> ProgramResult {
  ...



      // Now store the new domain data according to the config:
    DomainDataAccount::from(DomainData {
       bump_seed,
        validators_and_threshold: config.data,
    })
@>  .store(domain_pda_account, true)?;

    Ok(())
}
```

However, this expansion space is not guaranteed to be rent exempt, it is recommended to check `verify_rent_exempt()` or use `AccountData::store_with_rent_exempt_realloc()`.

## Impact

pda not rent exempt

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/blob/0053d8e839a245335544b9b920b30a8b48e3b33d/hyperlane-monorepo/rust/sealevel/programs/ism/multisig-ism-message-id/src/processor.rs#L449

## Tool Used

Manual Review

## Recommendation

Perform check `verify_rent_exempt()` or use `AccountData::store_with_rent_exempt_realloc()`

# Issue L-3: SyntheticPlugin::transfer_out() should check ata_payer_account rent exempt

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/62

## Summary

When executing `SyntheticPlugin::transfer_out()`, we may need to create an `associated token account`, with `ata_payer_account` as the payer. We need to make sure that after this pda is paid, it still rents exempt, but incorrectly checks for `recipient_ata`.

## Vulnerability Detail

`SyntheticPlugin::transfer_out()` is implemented as follows.

```
    fn transfer_out<'a, 'b>(
        program_id: &Pubkey,
        token: &HyperlaneToken<Self>,
        system_program: &'a AccountInfo<'b>,
        recipient_wallet: &'a AccountInfo<'b>,
        accounts_iter: &mut std::slice::Iter<'a, AccountInfo<'b>>,
        amount: u64,
    ) -> Result<(), ProgramError> {
      ...
        // Create and init (this does both) associated token account if necessary.
        invoke_signed(
            &create_associated_token_account_idempotent(
                ata_payer_account.key,
                recipient_wallet.key,
                mint_account.key,
                &spl_token_2022::id(),
            ),
            &[
                ata_payer_account.clone(),
                recipient_ata.clone(),
                recipient_wallet.clone(),
                mint_account.clone(),
                system_program.clone(),
                spl_token_2022.clone(),
            ],
            &[hyperlane_token_ata_payer_pda_seeds!(
                token.plugin_data.ata_payer_bump
            )],
        )?;
```

```
@>      verify_rent_exempt(recipient_ata, &Rent::get()?)?;
```

Currently checking `recipient_ata`, should check `ata_payer_account`.

## Impact

`ata_payer_account` not rent exempt

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/blob/0053d8e839a245335544b9b920b30a8b48e3b33d/hyperlane-monorepo/rust/sealevel/programs/hyperlane-sealevel-token/src/plugin.rs#L322

## Tool Used

Manual Review

## Recommendation

```
verify_rent_exempt(ata_payer_account, &Rent::get()?)?;
```

# Issue L-4: Igp size is larger than the actual need.

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/63

## Summary

The Igp size is larger than the actual need.

## Vulnerability Detail

The `Igp` size is calculated as follows.

```
pub struct Igp {
    /// The bump seed for the IGP PDA.
    pub bump_seed: u8,
    /// The salt used to derive the IGP PDA.
    pub salt: H256,
    /// The owner of the IGP.
    pub owner: Option<Pubkey>,
    /// The beneficiary of the IGP.
    pub beneficiary: Pubkey,
    /// The gas oracles for each destination domain.
    pub gas_oracles: HashMap<u32, GasOracle>,
}

impl SizedData for Igp {
    fn size(&self) -> usize {
        // 1 for bump_seed
        // 32 for salt
        // 33 for owner (1 byte Option, 32 bytes for pubkey)
        // 32 for beneficiary
        // 4 for gas_oracles.len()
        // M * (4 + (1 + 257)) for gas_oracles contents
        1 + 32 + 33 + 32 + 4 + (self.gas_oracles.len() * (1 + 257))
    }
}
```

257 is too large

```
pub enum GasOracle {
    /// Remote gas data stored directly in the variant data.
    RemoteGasData(RemoteGasData),
    // Future gas oracle variants could include a Pyth type, generalized CPI type,
 ↪  etc.
}
pub struct RemoteGasData {
```

```
    /// The token exchange rate for the remote token, adjusted by the
    /// TOKEN_EXCHANGE_RATE_SCALE.
    /// If this e.g. 0.2, then one local token would give you 5 remote tokens.
    #[cfg_attr(feature = "serde", serde(with =
↪   "hyperlane_core::utils::serde_u128"))]
    pub token_exchange_rate: u128,
    /// The gas price for the remote chain.
    #[cfg_attr(feature = "serde", serde(with =
↪   "hyperlane_core::utils::serde_u128"))]
    pub gas_price: u128,
    /// The number of decimals for the remote token.
    pub token_decimals: u8,
}
```

`GasOracle = 1 + 16 + 16 + 1`

So it should be: `1 + 32 + 33 + 32 + 4 + (self.gas_oracles.len() * (4 + (1 + 16 + 16 + 1)))`

## Impact

pay more rent

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/blob/0053d8e839a245335544b9b920b30a8b48e3b33d/hyperlane-monorepo/rust/sealevel/programs/hyperlane-sealevel-igp/src/accounts.rs#L180

## Tool Used

Manual Review

## Recommendation

```
impl SizedData for Igp {
    fn size(&self) -> usize {
        // 1 for bump_seed
        // 32 for salt
        // 33 for owner (1 byte Option, 32 bytes for pubkey)
        // 32 for beneficiary
        // 4 for gas_oracles.len()
-       // M * (4 + (1 + 257)) for gas_oracles contents
+       // M * (4 + (1 + 16 + 16 + 1)) for gas_oracles contents
```

```
-       1 + 32 + 33 + 32 + 4 + (self.gas_oracles.len() * (1 + 257))
+       1 + 32 + 33 + 32 + 4 + (self.gas_oracles.len() * (4 + (1 + 16 + 16 + 1)))
```

# Issue L-5: Malicious recipient can steal assets from signer accounts passed in inbox_process instruction

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/64

## Summary

The `recipient` program determines which accounts are passed to it. A malicious `recipient` program could include the `payer` account (which is a signer used for paying rent by the mailbox) in its returned account list. Since signer status is preserved during cross-program invocation (CPI), when the recipient program is called, it can use the `payer`'s signer privileges to access and control its assets.

## Vulnerability Detail

The vulnerability stems from two key issues:

1. The accounts passed to the `recipient` program during CPI are determined by the `recipient` program itself through the `InterchainSecurityModuleAccountMetas` instruction.
2. Signer status is preserved when accounts are passed to the CPI, allowing the recipient program to act with the authority of any signer account included in the list.

This creates a scenario where automated bots submitting verified messages can unknowingly pass their own `payer` account (with signer privileges) to a malicious recipient program.

## Impact

A malicious recipient program can steal all assets owned by the `payer` account, which is typically a bot submitting verified messages. This is particularly severe when:

1. Bots automatically construct instructions using account lists returned by the recipient
2. The same `payer` address is reused over a period of time

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/pull/52/files/3d3de0b031def3478d1572efdbd6d6d8df5b7b52#diff-e9b2fa002ebbf4ae6921f9eba8170ae5078c9af6eb561ed724aa3d4448abaf20R271-R287

## Tool Used

Manual Review

## Recommendation

To mitigate this vulnerability, implement one of the following solutions:

- Verify that the payer account is not included in the account list with signer status before proceeding with the CPI to recipient or the ISM program.
- Add user documentation to warn users implementing relayers of this scenario and to ensure payer is not included in the accounts returned by the recipient and ISM.

# Issue L-6: Missing explicit upper bound for threshold in multisig configuration

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/65

## Summary

The MultiSig currently accepts configurations with `threshold > 0` and `#validators <= threshold` without an upper bound for threshold. Solana's transaction size limit (1232 bytes) restricts the practical number of validator signatures (65 bytes each) to fewer than 18. When accounting for other transaction components, this limit is further reduced. An explicit threshold upper bound would prevent impractical configurations.

## Vulnerability Detail

The MultiSig implementation has two configuration requirements:

1. `threshold > 0`
2. `#validators <= threshold`

The validator signatures need to be passed in the transaction, and the mailbox is intended to be called directly rather than through CPI. The vulnerability lies in the absence of an upper bound for the threshold value, which could create configurations that are technically valid but practically impossible to satisfy due to Solana's transaction size constraints.

## Impact

Users can set impractical configurations that are unusable due to Solana transaction size limitations.

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/pull/43/files/ef83718c24b5a6e11c6df11c39d8f2126d855257#diff-d1dcb47cb56aed45e60f24832b149886d34d791a46dbdf0eff51b20d77e0cc38R79-R81

## Tool Used

Manual Review

## Recommendation

Add an explicit upper bound check for threshold values based on Solana's transaction size limit, accounting for signature size and additional transaction components.

**Workaround:** It is possible to circumvent transaction size limitations by storing metadata in chunks within a router program state and calling the mailbox via CPI. This works because argument size in CPI can be much larger than direct transaction calls. However, implementing a reasonable threshold bound would still improve the default user experience.

# Issue L-7: Error-prone implementation of AccountData::fetch function

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/66

## Summary

The `AccountData::fetch` method is used to retrieve data from accounts that should be initialized. However, the implementation does not verify initialization status and returns a default value for uninitialized accounts instead of throwing an error. While no vulnerable instructions have been identified, this pattern is error-prone as it removes a potential safety check and relies entirely on PDA address and program owner validations.

## Vulnerability Detail

The current implementation of `AccountData::fetch` returns a default value when called on an uninitialized account:

```
pub fn fetch(buf: &mut &[u8]) -> Result<Self, ProgramError> {
    Ok(Self::from(Self::fetch_data(buf)?.unwrap_or_default()))
}
```

This means the method provides no security benefit for validation and is only useful for its `store` capabilities.

## Impact

No vulnerable instructions have been identified. However, this implementation pattern may lead developers to incorrectly assume that `fetch` validates account initialization, potentially causing security issues in future code.

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/pull/3/files/b36cb0cebece9a25155655ce73cc666daf34f251#diff-0544fc952eefa8d170bbd200b75827f5bb58e39b132e7c871c1bc8a9d5e09ce5R92-R93

## Tool Used

Manual Review

## Recommendation

Modify `AccountData::fetch` to return an error if the account is not initialized.

# Issue L-8: Missing extensions checks in token collateral program

Source: https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/issues/67

## Summary

The hyperlane-sealevel-token-collateral program does not validate Token2022 mint extensions. Certain extensions may be incompatible with the current implementation.

## Vulnerability Detail

The program incorrectly handles tokens with certain Token2022 extensions. With the TransferFee extension, the program will receive fewer tokens than requested but still assumes the full amount was transferred. Other extensions like TransferHook and Permanent Delegate pose additional risks, as they could prevent tokens from being transferred out of the escrow or allow external actors to remove tokens from it.

## Impact

Tokens with certain extensions may cause unexpected behavior in the system, resulting in accounting discrepancies or other operational issues.

## Code Snippet

https://github.com/sherlock-audit/2025-02-hyperlane-sealevel-audit-wave-2/pull/31/files/272fc8e58963bf9ac5195bb1a92190df4c93833a#diff-c785bd6cedf76e1fcf74972111df07c6857709d8271ca4ca4c64ba2f99ce9b8fR281-R291

## Tool Used

Manual Review

## Recommendation

Perform due diligence before adding support for a Token2022 token. Consider referencing https://neodyme.io/en/blog/token-2022 which covers security considerations for different Token2022 extensions.

# Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.