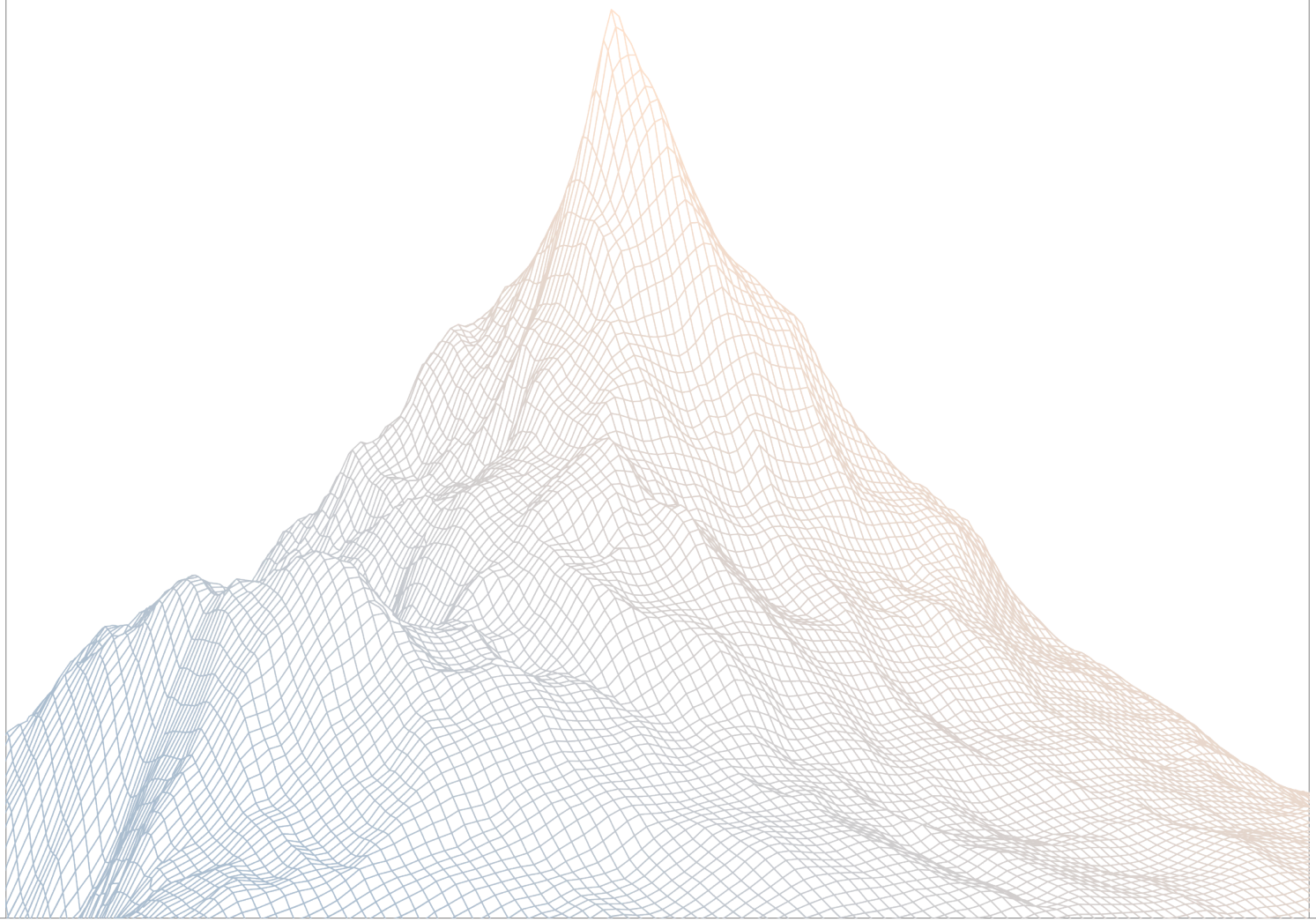
Zenith

Mantle

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: May 8th to May 15th, 2025

AUDITED BY: cccz
ladboy

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Mantle

Mantle Network is dedicated to building an EVM-compatible scaling solution for Ethereum. This means that all contracts and tools running on Ethereum can operate on the Mantle Network with minimal modifications. Taking advantage of its modular architecture, Mantle Network combines an optimistic rollup with various innovative data availability solutions, providing cheaper and more accessible data availability while inheriting the security of Ethereum.

Our protocol design philosophy aims to offer users a less costly and more user-friendly experience, provide developers with a simpler and more flexible development environment, and deliver a comprehensive set of infrastructure for the next wave of mass-adopted dApps.

2.2 Scope

The engagement involved a review of the following targets:

Target	fiat24contracts
Repository	https://github.com/mantle-xyz/fiat24contracts
Commit Hash	`fb84defefdb7c27680c22db5da14703170179f83`
Files	`Changes in PR-22`

Target	fiat24contracts
Repository	https://github.com/mantle-xyz/fiat24contracts
Commit Hash	`431583759901f8a8faf5c899f27ccdc215b490c8`
Files	`Changes in PR-28`

2.3 Audit Timeline

May 8, 2025	Audit start
May 15, 2025	Audit end
May 28, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	2
Medium Risk	8
Low Risk	4
Informational	7
Total Issues	**22**

3

Findings Summary

ID	Description	Status
C-1	Incorrect cross-chain message in permitAndDepositToken-ViaUsdc()	Resolved
H-1	Fiat24CryptoDeposit2._lzReceive() decoding incorrectly	Resolved
H-2	Using quoteExactInputSingle() to calculate amountOutMin-imum is vulnerable to sandwich attacks	Resolved
M-1	permitAndTransferFrom can revert because of insufficient allowance in Fiat24Token.sol	Resolved
M-2	adminProcessFailedRefund does not send the refund back to user	Acknowledged
M-3	Malicious actor can exhaust the max retry and block the refund in Fiat24CryptoDeposit2.sol	Acknowledged
M-4	Refund message cannot be sent back to Fiat24CryptoDeposit2.sol due to lack of native token airdrop when sending the message	Acknowledged
M-5	permitAndMoneyExchangeExactOut() requires the signer to provide exact input	Resolved
M-6	Different token pairs use the same slippage	Resolved
M-7	Fiat24CryptoDeposit2.depoitETH() don't work	Resolved
M-8	Token swaps in Fiat24CryptoRelay have no slippage control	Resolved
L-1	Consider avoid hardcode the fixedNativeFee in Fiat24CryptoRelay.sol	Acknowledged
L-2	CASH_OPERATOR_ROLE can arbitrarily spend funds that users approved to Fiat24CryptoDeposit2 and Fiat24CryptoRelay	Acknowledged
L-3	Fiat24CryptoRelay.processMessage() assumes usdc:usd24 is 1:1 when transferring fees	Acknowledged
L-4	Any user can withdraw mis-sent ETH via depositETH()	Resolved
I-1	Consider rename function quote to quoteLayerzeroFee	Resolved

ID	Description	Status
I-2	Consider follow the layerzero v2 integration checklist and remove the duplicated requirement in _lzReceive	Resolved
I-3	ReentrancyGuardUpgradeable is not initialized in initializer	Acknowledged
I-4	Consider remove unused code	Resolved
I-5	Incomplete update of privileged roles	Resolved
I-6	processMessage() needs to check if outputToken is valid	Resolved
I-7	permitAndDepositTokenViaUsdc() should check that usdcFactAmount rather than usdcAmount is in range.	Resolved

4

Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Incorrect cross-chain message in `permitAndDepositTokenViaUsdc()`

SEVERITY: Critical	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- PR-22

Description:

`permitAndDepositTokenViaUsdc()` incorrectly sets the `user` in the payload to `msg.sender`, which is `CASH_OPERATOR_ROLE`. This causes the cross-chain token to be incorrectly sent to `CASH_OPERATOR_ROLE`, causing the user to lose tokens.

```
bytes memory payload = abi.encode(
    _msgSender(),
    _inputToken,
    _amount,
    usdcAmount,
    _outputToken
);
```

Recommendations:

Change to

```
        bytes memory payload = abi.encode(
          _msgSender(),
          userAddress,
           _inputToken,
           _amount,
```

```
            usdcAmount,
            _outputToken
        );
```

Mantle: Resolved with PR-32

Zenith: Verified

4.2 High Risk

A total of 2 high risk findings were identified.

[H-1] `Fiat24CryptoDeposit2._lzReceive()` decoding incorrectly

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

The recent update changed the encoding of cross-chain data as follows

```
bytes memory payload = abi.encode(
    _msgSender(),
    _inputToken,
    _amount,
    usdcAmount,
    _outputToken
);
```

However, `Fiat24CryptoDeposit2._lzReceive()` still decodes the data in the old format, which causes the `inputToken` to be decoded as `usdcAmount` when refund.

```
function _lzReceive(
    Origin calldata /* _origin */,
    bytes32 _guid,
    bytes calldata _payload,
    address /* _executor */,
    bytes calldata /* _extraData */
) internal override {
    (address user, uint256 usdcAmount, address outputToken)
    = abi.decode(_payload, (address, uint256, address));
```

```
    _handleRefund(_guid, user, usdcAmount, outputToken);
}
```

The worst impact is that a large amount of USDC in `usdcDepositAddress` will be stolen.

Recommendations:

Change to

```
function _lzReceive(
    Origin calldata /* _origin */,
    bytes32 _guid,
    bytes calldata _payload,
    address /* _executor */,
    bytes calldata /* _extraData */
) internal override {

(address user, uint256 usdcAmount, address outputToken) = abi.decode(_
      payload, (address, uint256, address));
   (address user, address inputToken, uint256 inputAmount, uint256
      usdcAmount, address outputToken) = abi.decode(payload, (address,
      address, uint256, uint256, address));
    _handleRefund(_guid, user, usdcAmount, outputToken);
}
```

In addition, some minor changes are needed.

```
        bytes memory payload = abi.encode(
            _msgSender(),
            address(0),
          msg.value,
          amountIn,
            usdcAmount,
            _outputToken
        );
```

```
function quote(
    uint32 _dstEid,
    address _userAddress,
   address _inputToken,
   uint256 _inputTokenAmount,
```

```
        uint256 _usdcAmount,
        address _outputToken
    ) public view returns (MessagingFee memory fee) {
        bytes memory payload = abi.encode(
            _userAddress,
            _inputToken,
            _inputTokenAmount,
            _usdcAmount,
            _outputToken
        );

        bytes memory defaultWorkerOptions = OptionsBuilder
            .newOptions()
            .addExecutorLzReceiveOption(relay_gas_limit, 0);

        fee = _quote(_dstEid, payload, defaultWorkerOptions, false);
    }
```

Mantle: Resolved with PR-32

Zenith: Verified.

[H-2] Using `quoteExactInputSingle()` to calculate `amountOutMinimum` is vulnerable to sandwich attacks

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

Fiat24CryptoDeposit2 supports swapping different input tokens to USDC through UniswapV3 and then cross-chain.

When calculating `amountOutMinimum`, the protocol uses the result of `quoteExactInputSingle()`.

Since the result of `quoteExactInputSingle()` is the same as the result of `exactInputSingle()` in the same transaction, this makes the slippage control not work.

```
function quoteExactInputSingle(
    address tokenIn,
    address tokenOut,
    uint24 fee,
    uint256 amountIn,
    uint160 sqrtPriceLimitX96
) public override returns (uint256 amountOut) {
    bool zeroForOne = tokenIn < tokenOut;

    try
        getPool(tokenIn, tokenOut, fee).swap(
            address(this), // address(0) might cause issues with some tokens
            zeroForOne,
            amountIn.toInt256(),
            sqrtPriceLimitX96 == 0
                ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 :
    TickMath.MAX_SQRT_RATIO - 1)
                : sqrtPriceLimitX96,
            abi.encodePacked(tokenIn, fee, tokenOut)
        )
```

```
    {} catch (bytes memory reason) {
        return parseRevertReason(reason);
    }
}
```

For example, the current WETH/USDC = 2500, the attacker manipulates the uniswap price to 2000. Since the prices in `quoteExactInputSingle()` and `exactInputSingle()` are both 2000, the user will suffer a sandwich attack when performing the exchange.

Recommendations:

It is recommended that users obtain the result of `quoteExactInputSingle()` off-chain and pass it in as a parameter for slippage control.

Mantle: Resolved with PR-32

Zenith: Verified

4.3 Medium Risk

A total of 8 medium risk findings were identified.

[M-1] `permitAndTransferFrom` can revert because of insufficient allowance in `Fiat24Token.sol`

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- Fiat24Token.sol

Description:

In `Fiat24Token`, the permit related feature are added.

```
function permitAndTransferFrom(
    address userAddress,
    address recipient,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (bool) {
    require(hasRole(CASH_OPERATOR_ROLE, _msgSender()), "Fiat24Token: Not a
    Cash Operator");
    try IERC20PermitUpgradeable(address(this)).permit(
        userAddress,
        address(this),
        amount,
        deadline,
        v, r, s
    ) {
    } catch {
        emit PermitFailed(userAddress, address(this), amount);
    }
```

```
    return transferFrom(userAddress,recipient,amount);
}
```

and

```
function transferFrom(address sender, address recipient, uint256 amount)
    public virtual override returns (bool) {
    _transfer(sender, recipient, amount);

    uint256 currentAllowance = allowance(sender, _msgSender());
    require(currentAllowance >= amount, "ERC20: transfer amount exceeds
    allowance");
    unchecked {
        _approve(sender, _msgSender(), currentAllowance - amount);
    }

    return true;
}
```

As we can see, both function `permitAndTransferFrom` and `transferFrom` shares the same `_msgSender()`,

the allowance check validate that `_msgSender()` has sufficient allowance to spend the sender's balance.

However, the user only gives permit to `address(this)`, not `_msgSender()`,

```
/// @inheritdoc IERC20Permit
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public virtual {
```

Then calling `transferFrom` after the permit will still suffer from transaction revert.

This issue exists in other `permit` usage in the smart cotnract `Fiat24Token.sol` as well listed below.

- permitAndClientPayout
- permitAndClientPayout(

Recommendations:

```
function permitAndTransferFrom(
    address userAddress,
    address recipient,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (bool) {
    require(hasRole(CASH_OPERATOR_ROLE, _msgSender()), "Fiat24Token: Not
a Cash Operator");
    try IERC20PermitUpgradeable(address(this)).permit(
        userAddress,
        address(this),
        amount,
        deadline,
        v, r, s
    ) {
    } catch {
        emit PermitFailed(userAddress, address(this), amount);
    }

   return transferFrom(userAddress,recipient,amount);
  return address(this).transferFrom(userAddress,recipient,amount);
}
```

Mantle: Resolved with PR-32

Zenith: Verified

[M-2] `adminProcessFailedRefund` does not send the refund back to user

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Fiat24CryptoDeposit2.sol

Description:

`Fiat24CryptoDeposit2.sol` sent messaget to `Fiat24CryptoRelay.sol`

If `Fiat24CryptoRelay.sol` fails to process the message, a refund message is sent back to `Fiat24CryptoDeposit2.sol`

If `Fiat24CryptoDeposit2.sol` fails to process the refund message, the message is stored and anyone can retry the message later.

Themax number of retry is 3

There is no access control when calling the function `retryFailedRefund`

```
function retryFailedRefund(bytes32 _guid) external {
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund storage refund = failedMessages[_guid];
    require(refund.user != address(0), "No failed message");
    require(refund.retryCount < MAX_RETRY_COUNT, "Max retries reached");

    try IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress,
    refund.user, refund.usdcAmount) {
        delete failedMessages[_guid];
        emit RefundRetried(_guid);
    } catch {
        unchecked { refund.retryCount++; }
        emit RefundProcessFailed(_guid, refund.user, refund.usdcAmount);
    }
}
```

After the max number of retry is reached, the admin should be able to trigger the refund.

However, when admin calls `adminProcessFailedRefund`, the code just delete the refund request without sending the fund the user, then after the max number of retry is reached, fund is locked.

Recommendations:

Consider allowing admin to process the refund any time.

```
function adminProcessFailedRefund(bytes32 _guid) external {
    require(hasRole(OPERATOR_ROLE, msg.sender), "Not Operator");
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund memory refund = failedMessages[_guid];
    require(refund.user ≠ address(0), "No failed message");

    // Only allow manual processing when retries exhausted
    require(refund.retryCount ⩾ MAX_RETRY_COUNT, "Retry count not
        exceeded");
    delete failedMessages[_guid];


    IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress, refund.user,
        refund.usdcAmount)
    emit RefundRetried(_guid);
}
```

Mantle: Acknowledged

[M-3] Malicious actor can exhaust the max retry and block the refund in `Fiat24CryptoDeposit2.sol`

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- Fiat24CryptoDeposit2.sol

Description:

`Fiat24CryptoDeposit2.sol` sent message to `Fiat24CryptoRelay.sol`

If `Fiat24CryptoRelay.sol` fails to process the message, a refund message is sent back to `Fiat24CryptoDeposit2.sol`

If `Fiat24CryptoDeposit2.sol` fails to process the refund message, the message is stored and anyone can retry the message later.

Themax number of retry is 3

There is no access control when calling the function `retryFailedRefund`

```
function retryFailedRefund(bytes32 _guid) external {
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund storage refund = failedMessages[_guid];
    require(refund.user ≠ address(0), "No failed message");
    require(refund.retryCount < MAX_RETRY_COUNT, "Max retries reached");

    try IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress,
    refund.user, refund.usdcAmount) {
        delete failedMessages[_guid];
        emit RefundRetried(_guid);
    } catch {
        unchecked { refund.retryCount++; }
        emit RefundProcessFailed(_guid, refund.user, refund.usdcAmount);
    }
}
```

If there is insufficient USDC in `usdcDepositAddress`, transferring fund from deposit address

to user will always revert.

Thena malicious actor can just pick a `guid` and call `refund` 3 times to exhaust the max retry and lock the fund.

Recommendations:

Only allow refund receiver to retry the refund, allow admin to process the fund any time.

```
function retryFailedRefund(bytes32 _guid) external {
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund storage refund = failedMessages[_guid];
    require(refund.user ≠ address(0), "No failed message");
    require(refund.user ≠ msg.sender, "No failed message");
    require(refund.retryCount < MAX_RETRY_COUNT, "Max retries reached");

    try IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress,
refund.user, refund.usdcAmount) {
        delete failedMessages[_guid];
        emit RefundRetried(_guid);
    } catch {
        unchecked { refund.retryCount++; }
        emit RefundProcessFailed(_guid, refund.user, refund.usdcAmount);
    }
}

function adminProcessFailedRefund(bytes32 _guid) external {
    require(hasRole(OPERATOR_ROLE, msg.sender), "Not Operator");
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund memory refund = failedMessages[_guid];
    require(refund.user ≠ address(0), "No failed message");

    // Only allow manual processing when retries exhausted
    require(refund.retryCount ⩾ MAX_RETRY_COUNT, "Retry count not
        exceeded");
    delete failedMessages[_guid];


    IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress, refund.user,
        refund.usdcAmount)
    emit RefundRetried(_guid);
}
```

Mantle: Acknowledged. If a refund fails, we will try to refund three times off-chain. If all retries fail, the finance team will manually complete the transfer.

[M-4] Refund message cannot be sent back to `Fiat24CryptoDeposit2.sol` due to lack of native token airdrop when sending the message

SEVERITY: Medium	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Fiat24CryptoDeposit2.sol

Description:

`Fiat24CryptoDeposit2.sol` sent messaget to `Fiat24CryptoRelay.sol`

When sending the message, the `workOptions` has no native ETH attached => `addExecutorLzReceiveOption(relay_gas_limit, 0)`

```
bytes memory payload = abi.encode(
    _msgSender(),
    usdcAmount,
    _outputToken
);

bytes memory defaultWorkerOptions = OptionsBuilder
    .newOptions()
    .addExecutorLzReceiveOption(relay_gas_limit, 0);

MessagingFee memory fee = MessagingFee({
    nativeFee: nativeFee,
    lzTokenFee: 0
});
```

- evm/configuration/options#add-options-types

> addExecutorLzReceiveOption is a method that can be used to specify how much gas limit and msg.value the Executor uses when calling lzReceive on the receiving chain.

If `Fiat24CryptoRelay.sol` fails to process the message, a refund message is sent back to

Fiat24CryptoDeposit2.sol

```
function _refundToSource(
    uint32 _srcEid,
    bytes32 _srcOApp,
    bytes calldata refundPayload,
    string memory reason
) internal {

    bytes memory options = OptionsBuilder
        .newOptions()
        .addExecutorLzReceiveOption(RELAY_GAS_LIMIT, 0);

    MessagingFee memory fee = MessagingFee({
        nativeFee: fixedNativeFee,
        lzTokenFee: 0
    });

    this.externalLzSend{value: fixedNativeFee}(
        _srcEid,
        refundPayload,
        options,
        fee,
        payable(address(this))
    );

    emit RefundSent(_srcEid, _srcOApp, reason);
}
```

When sending the message back, the code needs to attach `fixedNativeFee` to pay for layerzero v2 transaction fee.

Give the executor does not attach native ETH by default and if there is insufficient native token in `Fiat24CryptoRelay.sol`, refund transaction still revert aftering catching the `processMessage` failure.

```
function _lzReceive(
    Origin calldata _origin,
    bytes32 _guid,
    bytes calldata payload,
    address /* _executor */,
    bytes calldata /* _extraData */
) internal override {

    require(peers(_origin.srcEid) == _origin.sender, "Invalid sender");
    try this.processMessage(payload) {
```

```
            emit MessageProcessed(_guid);
        } catch Error(string memory reason) {
            _refundToSource(_origin.srcEid, _origin.sender, payload, reason);
        } catch {
            _refundToSource(_origin.srcEid, _origin.sender, payload, "Unknown
        failure");
        }
    }
```

Recommendations:

Consider add the option to specify the msg.value the Executor uses when calling lzReceive on the receiving chain

```
bytes memory defaultWorkerOptions = OptionsBuilder
    .newOptions()
    .addExecutorLzReceiveOption(relay_gas_limit, native_eth_gas);
```

Mantle: Acknowledged.

[M-5] `permitAndMoneyExchangeExactOut()` requires the signer to provide exact input

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

`Fiat24CryptoRelay.permitAndMoneyExchangeExactOut()` will exchange the exact output tokens to the user. `permitAndMoneyExchangeExactOut()` will first consume the user's signature and transfer the inputAmount amount of tokens.

```
function permitAndMoneyExchangeExactOut(
    address userAddress,
    address _inputToken,
    address _outputToken,
    uint256 _outputAmount,
    uint256 _deadline,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
) external returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (!hasRole(CASH_OPERATOR_ROLE, _msgSender()))
    revert Fiat24Token__NotCashOperator(_msgSender());
    if (_outputAmount == 0) revert Fiat24CryptoDeposit__ValueZero();
    if (!validXXX24Tokens[_inputToken])
    revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
    if (!validXXX24Tokens[_outputToken])
    revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);
    if (_inputToken == _outputToken)
    revert Fiat24CryptoDeposit__InputTokenOutputTokenSame(_inputToken,
    _outputToken);

    uint256 inputAmount = _outputAmount * getExchangeRate(_outputToken,
    _inputToken) / XXX24_DIVISOR * getSpread(_outputToken, _inputToken,
    true) / XXX24_DIVISOR;
```

```
    if (inputAmount == 0) revert Fiat24CryptoDeposit__ValueZero();

    // Permit
    try IERC20Permit(_inputToken).permit(
        userAddress,
        address(this),
        inputAmount,
        _deadline,
        _v, _r, _s
    ) {} catch {
        emit PermitFailed(userAddress, address(this), inputAmount);
    }
    TransferHelper.safeTransferFrom(_inputToken, userAddress,
    IFiat24Account(fiat24account).ownerOf(TREASURY_DESK), inputAmount);
    TransferHelper.safeTransferFrom(_outputToken,
    IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK), userAddress,
    _outputAmount);
    emit MoneyExchangedExactOut(userAddress, _inputToken, _outputToken,
    inputAmount, _outputAmount);
    return inputAmount;
}
```

The problem here is that the user needs to include the exact inputAmount in the signature of token.permit(), which breaks the intent of the function.

Recommendations:

It is recommended to make `inputAmount` in the signature `maxInputAmount`, just make sure it is greater than `inputAmount`.

```
        uint256 inputAmount = _outputAmount * getExchangeRate(_outputToken,
    _inputToken) / XXX24_DIVISOR * getSpread(_outputToken, _inputToken,
    true) / XXX24_DIVISOR;
        if (inputAmount == 0) revert Fiat24CryptoDeposit__ValueZero();

        // Permit
        try IERC20Permit(_inputToken).permit(
            userAddress,
            address(this),
            inputAmount,
            maxInputAmount
            _deadline,
            _v, _r, _s
        ) {} catch {
            emit PermitFailed(userAddress, address(this), inputAmount);
```

```
        }
        TransferHelper.safeTransferFrom(_inputToken, userAddress,
    IFiat24Account(fiat24account).ownerOf(TREASURY_DESK), inputAmount);
        TransferHelper.safeTransferFrom(_outputToken,
    IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK), userAddress,
    _outputAmount);
        emit MoneyExchangedExactOut(userAddress, _inputToken, _outputToken,
    inputAmount, _outputAmount);
        return inputAmount;
    }
```

Mantle: Resolved with PR-32

Zenith: Verified

[M-6] Different token pairs use the same slippage

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

Fiat24CryptoDeposit2 supports swapping different input tokens to USDC through UniswapV3 and then cross-chain. When performing UniswapV3 swaps, the protocol uses the same slippage (default is 5%) for different token pairs.

```
uint256 amountOutMininumUSDC = getQuote(_inputToken, usdc, poolFee,
    _amount);
ISwapRouter.ExactInputSingleParams memory params
    = ISwapRouter.ExactInputSingleParams({
    tokenIn: _inputToken,
    tokenOut: usdc,
    fee: poolFee,
    recipient: address(this),
    deadline: block.timestamp + 15,
    amountIn: _amount,
    amountOutMinimum:
    amountOutMininumUSDC.sub(amountOutMininumUSDC.mul(slippage).div(100)),
    sqrtPriceLimitX96: 0
});

usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);
```

Since different token pairs have different pool depths, to ensure the swap success, the slippage must be adapted to the pool with the greatest volatility. For example, for the USDT/USDC pool, a slippage of 1% is sufficient, but to adapt to the ETH/USDC pool, the slippage needs to be 5%, which results in the USDT/USDC swap potentially facing sandwich attacks.

Recommendations:

It is recommended that the slippage parameter be passed in by the user rather than set by the protocol.

Mantle: Resolved with PR-32

Zenith: Verified

[M-7] `Fiat24CryptoDeposit2.depoitETH()` don't work

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

`Fiat24CryptoDeposit2.depositETH()` will return the remaining ETH to the user after the exchange.

```
uint256 amountIn = msg.value - nativeFee;

ISwapRouter.ExactInputSingleParams memory params
    = ISwapRouter.ExactInputSingleParams({
    tokenIn: weth,
    tokenOut: usdc,
    fee: poolFee,
    recipient: address(this),
    deadline: block.timestamp + 15,
    amountIn: amountIn,
    amountOutMinimum: getQuote(weth, usdc, poolFee, amountIn)
.sub(getQuote(weth, usdc, poolFee, amountIn).mul(slippage).div(100)),
    sqrtPriceLimitX96: 0
});

uint256 usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle{value:
    amountIn}(params);
IPeripheryPaymentsWithFee(UNISWAP_PERIPHERY_PAYMENTS).refundETH();

(bool success, ) = msg.sender.call{value: address(this).balance}("");
```

The problem here is that the protocol will return all the ETH balance to the user, including `nativeFee`, which will cause the cross-chain message in `_lzSend()` to fail due to insufficient balance.

```
        MessagingFee memory fee = MessagingFee({
```

```
            nativeFee: nativeFee,
            lzTokenFee: 0
        });

        _lzSend(dstChainId, payload, defaultWorkerOptions, fee,
    payable(msg.sender));
...
    function _lzSend(
        uint32 _dstEid,
        bytes memory _message,
        bytes memory _options,
        MessagingFee memory _fee,
        address _refundAddress
    ) internal virtual returns (MessagingReceipt memory receipt) {
        // @dev Push corresponding fees to the endpoint, any excess is sent
    back to the _refundAddress from the endpoint.
        uint256 messageValue = _payNative(_fee.nativeFee);
        if (_fee.lzTokenFee > 0) _payLzToken(_fee.lzTokenFee);

        return
            // solhint-disable-next-line check-send-result
            endpoint.send{ value: messageValue }(
                MessagingParams(_dstEid, _getPeerOrRevert(_dstEid),
    _message, _options, _fee.lzTokenFee > 0),
                _refundAddress
            );
    }
```

Recommendations:

Change to

```
        uint256 usdcAmount
    = ISwapRouter(UNISWAP_ROUTER).exactInputSingle{value: amountIn}(params);
        IPeripheryPaymentsWithFee(UNISWAP_PERIPHERY_PAYMENTS).refundETH();


    (bool success, ) = msg.sender.call{value: address(this).balance}("");
       (bool success, ) = msg.sender.call{value: address(this).balance -
          nativeFee}("");
        if (!success) revert Fiat24CryptoDeposit__EthRefundFailed();
```

Mantle: Resolved with PR-32

Zenith: Verified

[M-8] Token swaps in Fiat24CryptoRelay have no slippage control

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- PR-22

Description:

When users exchange tokens in Fiat24CryptoRelay via `moneyExchangeExactIn()` and `moneyExchangeExactOut()`, they cannot control slippage via minOutputAmount and maxInputAmount.

```
function moneyExchangeExactIn(address _inputToken, address _outputToken,
    uint256 _inputAmount) external returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (_inputAmount == 0) revert Fiat24CryptoDeposit__ValueZero();
    if (!validXXX24Tokens[_inputToken])
    revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
    if (!validXXX24Tokens[_outputToken])
    revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);
    if (_inputToken == _outputToken)
    revert Fiat24CryptoDeposit__InputTokenOutputTokenSame(_inputToken,
    _outputToken);

    TransferHelper.safeTransferFrom(_inputToken, _msgSender(),
    IFiat24Account(fiat24account).ownerOf(TREASURY_DESK), _inputAmount);
    uint256 outputAmount =
        _inputAmount * getExchangeRate(_inputToken, _outputToken)
    / XXX24_DIVISOR * getSpread(_inputToken, _outputToken, false)
    / XXX24_DIVISOR;
    TransferHelper.safeTransferFrom(_outputToken,
    IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK), _msgSender(),
    outputAmount);

    emit MoneyExchangedExactIn(_msgSender(), _inputToken, _outputToken,
    _inputAmount, outputAmount);
    return outputAmount;
```

```
}
```

If `exchangeRates` changes, users may receive or spend different tokens than expected.

Recommendations:

It is recommended to provide parameters to allow users to control slippage

Mantle: Resolved with PR-32

Zenith: Verified

4.4 Low Risk

A total of 4 low risk findings were identified.

[L-1] Consider avoid hardcode the `fixedNativeFee` in `Fiat24CryptoRelay.sol`

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Fiat24CryptoRelay.sol

Description:

In `Fiat24CryptoRelay.sol`, if the `processMessage` fails, a refund message is sent.

```
function _refundToSource(
     uint32 _srcEid,
     bytes32 _srcOApp,
     bytes calldata refundPayload,
     string memory reason
 ) internal {

     bytes memory options = OptionsBuilder
         .newOptions()
         .addExecutorLzReceiveOption(RELAY_GAS_LIMIT, 0);

     MessagingFee memory fee = MessagingFee({
         nativeFee: fixedNativeFee,
         lzTokenFee: 0
     });

     this.externalLzSend{value: fixedNativeFee}(
         _srcEid,
         refundPayload,
         options,
         fee,
         payable(address(this))
```

```
        );

        emit RefundSent(_srcEid, _srcOApp, reason);
    }
```

The code hardcode the `fixedNativeFee`, which is currently 0.1 native value.

If the dynamic layerzero fee surpasses the hardcoded native value, the refund message cannot be sent.

Recommendations:

Consider quote the value instead of hardcode the `fixedNativeFee` value.

```
function _refundToSource(
        uint32 _srcEid,
        bytes32 _srcOApp,
        bytes calldata refundPayload,
        string memory reason
    ) internal {

        bytes memory options = OptionsBuilder
            .newOptions()
            .addExecutorLzReceiveOption(RELAY_GAS_LIMIT, 0);

        fee = _quote(_dstEid, refundPayload, options, false);

        this.externalLzSend{value: fee.nativeFee}(
            _srcEid,
            refundPayload,
            options,
            fee,
            payable(address(this))
        );
    }
```

Mantle: Acknowledged

[L-2] CASH_OPERATOR_ROLE can arbitrarily spend funds that users approved to Fiat24CryptoDeposit2 and Fiat24CryptoRelay

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- PR-22

Description:

PR-22 Allow `CASH_OPERATOR_ROLE` to call `permitAnd*()` functions of Fiat24CryptoDeposit2 and Fiat24CryptoRelay.

Taking `permitAndDepositTokenViaUsdc()` as an example, `CASH_OPERATOR_ROLE` calls it with the signature provided by the user, which calls `token.permit()` to consume the signature and approve Fiat24CryptoDeposit2 to spend the user's tokens, and then transfers these tokens from the user.

To prevent frontrun attacks on `token.permit()`, the contract wraps the `token.permit()` call with try-catch, but this also gives `CASH_OPERATOR_ROLE` the opportunity to provide an invalid signature to consume the tokens that the user has approved to Fiat24CryptoDeposit2. When `CASH_OPERATOR_ROLE` provides an invalid signature to call `permitAndDepositTokenViaUsdc()`, `token.permit()` fails and is caught, and then `safeTransferFrom()` spends the tokens that the user has approved.

```
function permitAndDepositTokenViaUsdc(
    address userAddress,
    address _inputToken,
    address _outputToken,
    uint256 _amount,
    uint256 _feeAmountViaUsdc,
    uint256 _deadline,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
) external nonReentrant payable returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (!hasRole(CASH_OPERATOR_ROLE, _msgSender()))
    revert Fiat24Token__NotCashOperator(_msgSender());
```

```
    if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
    if (!validXXX24Tokens[_outputToken])
    revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);

    try IERC20PermitUpgradeable(_inputToken).permit(
        userAddress,
        address(this),
        _amount,
        _deadline,
        _v, _r, _s
    ) {
    } catch {
        emit PermitFailed(userAddress, _inputToken, _amount);
    }

    TransferHelper.safeTransferFrom(_inputToken, userAddress, address(this),
    _amount);
    TransferHelper.safeApprove(_inputToken, UNISWAP_ROUTER, _amount);
```

Recommendations:

If `CASH_OPERATOR_ROLE` is trusted, it is recommended to document the issue to inform users.

Mantle: Acknowledged

[L-3] `Fiat24CryptoRelay.processMessage()` assumes usdc:usd24 is 1:1 when transferring fees

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- PR-22

Description:

When transferring the fee in `Fiat24CryptoRelay.processMessage()`, the protocol assumes usdc:usd24 = 1:1.

```
if (walletId == 0 || !walletIdExists) {
    TransferHelper.safeTransferFrom(
        usd24,
        IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK),
        IFiat24Account(fiat24account).ownerOf(FEE_DESK),
        feeInUSDC / USDC_DIVISOR
    );
} else {
    TransferHelper.safeTransferFrom(
        usd24,
        IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK),
        IFiat24Account(fiat24account).ownerOf(walletId),
        feeInUSDC / USDC_DIVISOR
    );
}
```

Given that the protocol sets `exchangeRates[usdc][usd24]`, this will result in incorrect fees being charged by the protocol when `exchangeRates` changes.

```
exchangeRates[usdc][usd24] = 10000;
```

Recommendations:

It is recommended to calculate the amount of usd24 corresponding to usdc based on `exchangeRates[usdc][usd24]` when transferring fees.

Mantle: Acknowledged

[L-4] Any user can withdraw mis-sent ETH via `depositETH()`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- Fiat24CryptoDeposit

Description:

`Fiat24CryptoDeposit2.withdrawETH()` allows `OPERATOR_ADMIN_ROLE` to withdraw ETH that was mistakenly sent in the contract.

```
function withdrawETH(address payable to, uint256 amount) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))
    revert Fiat24CryptoDeposit__NotOperator(_msgSender());
    require(to ≠ address(0), "Invalid recipient");
    require(address(this).balance >= amount, "Insufficient ETH balance");

    (bool success, ) = to.call{value: amount}("");
    require(success, "ETH withdrawal failed");
}
```

Since `depositETH()` will send all ETH in the contract to the user, any user can withdraw ETH that was mistakenly sent in the contract through `depositETH()`.

```
uint256 usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle{value:
    amountIn}(params);
IPeripheryPaymentsWithFee(UNISWAP_PERIPHERY_PAYMENTS).refundETH();

(bool success, ) = msg.sender.call{value: address(this).balance}("");
```

Recommendations:

It is recommended to only return the difference before and after `refundETH()` to the user (also avoiding returning nativeFee to the user).

Mantle: Resolved with PR-32

Zenith: Verified

4.5 Informational

A total of 7 informational findings were identified.

[I-1] Consider rename function `quote` to `quoteLayerzeroFee`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Fiat24CryptoDeposit2.sol

Description:

```
function quote(
    uint32 _dstEid,
    address _userAddress,
    uint256 _usdcAmount,
    address _outputToken
) public view returns (MessagingFee memory fee) {
```

Function `quote` quotes layerzero fee.

```
function getQuote(address _inputToken, address _outputToken, uint24 _fee,
    uint256 _amount) public returns (uint256) {
    return IQuoter(UNISWAP_QUOTER).quoteExactInputSingle(_inputToken,
    _outputToken, _fee, _amount, 0);
}
```

and `function getQuote` quotes swap amount.

The function names quote and getQuote are overly similar, which may cause confusion for users or integrators. While `quote` estimates LayerZero messaging fees, `getQuote` calculates swap amounts.

Recommendations:

Consider rename `quote` to `quoteLayerzeroFee` for external integration.

Mantle: Resolved with PR-32

Zenith: Verified

[I-2] Consider follow the layerzero v2 integration checklist and remove the duplicated requirement in `_lzReceive`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Fiat24CryptoRelay.sol

Description:

Consider follow the layerzero v2 integration checklist

- evm/technical-reference/integration-checklist

Recommendations:

We recommend follow these two item from the checklist.

1. Implement Enforced Options

https://docs.layerzero.network/v2/developers/evm/technical-reference/integration-checklist#implement-enforced-options

> Implement and set enforcedOptions to ensure users pay a predetermined amount of gas for delivery on the destination transaction. This setup guarantees that messages sent from a source have sufficient gas to be executed on the destination chain.

2. Avoid Redundant require Statements

> Do not add require statements that repeat checks in parent contracts, such as those in OAppReceiver.lzReceive.

Currently, the `lzReceive` already validates the sender is a valid peer

```
if (_getPeerOrRevert(_origin.srcEid) ≠ _origin.sender)
    revert OnlyPeer(_origin.srcEid, _origin.sender);
```

Then peer check in `Fiat24CryptoRelay.sol` `lzReceive` function can be removed.

```
    function _lzReceive(
        Origin calldata _origin,
        bytes32 _guid,
        bytes calldata payload,
        address /* _executor */,
        bytes calldata /* _extraData */
    ) internal override {

        require(peers(_origin.srcEid) == _origin.sender, "Invalid sender");
```

Mantle: Resolved with @e7de74d512...

Zenith: Verified.

[I-3] `ReentrancyGuardUpgradeable` is not initialized in `initializer`

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Acknowledged	`LIKELIHOOD:` Low

Target

- Fiat24CryptoRelay.sol

Description:

The smart contract `Fiat24CryptoRelay` inherits from `ReentrancyGuardUpgradeable`, but in the `initializer`, the code does not calls __ReentrancyGuard_init

Recommendations:

Consider call __ReentrancyGuard_init

Mantle: Acknowledged.

[I-4] Consider remove unused code

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Fiat24CryptoDeposit2.sol

Description:

The variable `lzNativeFee` is not used.

```
uint256 private lzNativeFee = 198009600000000;
```

Recommendations:

Consider remove the `lzNativeFee` variable.

Mantle: Resolved with @982e238a43...

Zenith: Verified

[I-5] Incomplete update of privileged roles

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- PR-22

Description:

PR-22 adds new privileged roles such as `OPERATOR_ADMIN_ROLE`, `PAUSE_ROLE`, and `UNPAUSE_ROLE`, but doesn't grant the roles during initialization, although the admin can call `grantRole()` later to grant the roles.

```
bytes32 public constant OPERATOR_ADMIN_ROLE
    = keccak256("OPERATOR_ADMIN_ROLE");
bytes32 public constant PAUSE_ROLE = keccak256("PAUSE_ROLE");
bytes32 public constant UNPAUSE_ROLE = keccak256("UNPAUSE_ROLE");
```

Recommendations:

It is recommended to grant the `OPERATOR_ADMIN_ROLE`, `PAUSE_ROLE`, `UNPAUSE_ROLE` roles during contract initialization.

Mantle: Resolved with PR-32

Zenith: Verified

[I-6] `processMessage()` needs to check if `outputToken` is valid

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- PR-22

Description:

When cross-chain tokens, the source chain checks whether the _outputToken is valid.

```
if (paused()) revert Fiat24CryptoDeposit__Paused();
if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
if (!validXXX24Tokens[_outputToken])
    revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);
```

However, the `processMessage()` function on the target chain does not check it.

Assuming that the `_outputToken` on the target chain is invalid, `exchangeRates[usd24][_outputToken]` is 0 and a division by zero error occurs when `processMessage()` calls `getExchangeRate()`.

```
if (_inputToken == usd24 || _outputToken == usd24) {
    exchangeRate =
        exchangeRates[_inputToken][_outputToken] == 0 ?
    10000 ** 2 / exchangeRates[_outputToken][_inputToken] :
    exchangeRates[_inputToken][_outputToken];
}
```

Recommendations:

Change to

```
    function processMessage(bytes calldata payload) nonReentrant external {

        if (paused()) revert Fiat24CryptoDeposit__Paused();
```

```
        require(msg.sender == address(this), "Only internal calls");

        (address user, uint256 usdcAmount, address outputToken)
    = abi.decode(payload, (address, uint256, address));
        if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__
            NotValidOutputToken(_outputToken);
        require(user != address(0), "Invalid user");
        require(usdcAmount > 0, "Invalid amount");
```

Mantle: Resolved with PR-32

Zenith: Verified

[I-7] `permitAndDepositTokenViaUsdc()` should check that `usdcFactAmount` rather than `usdcAmount` is in range.

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- ERC20.sol

Description:

In `permitAndDepositTokenViaUsdc()`, the actual cross-chain usdc amount is `usdcFactAmount` instead of `usdcAmount`, so it should be checked that `usdcFactAmount` is in the range.

```
if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
if (usdcAmount > maxUsdcDepositAmount)
    revert Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(usdcAmount,
    maxUsdcDepositAmount);
if (usdcAmount < minUsdcDepositAmount)
    revert Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(usdcAmount,
    minUsdcDepositAmount);

if (_feeAmountViaUsdc >= MAX_FEE_AMOUNT_USDC) {
    _feeAmountViaUsdc = MAX_FEE_AMOUNT_USDC;
}

uint256 usdcFactAmount = usdcAmount - _feeAmountViaUsdc;
TransferHelper.safeTransfer(usdc, feeReceiver, _feeAmountViaUsdc);
TransferHelper.safeTransfer(usdc, usdcDepositAddress, usdcFactAmount);
```

Recommendations:

```
    if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero(
        );
```

```
        if (usdcAmount > maxUsdcDepositAmount) revert Fiat24CryptoDeposit__
UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount);
        if (usdcAmount < minUsdcDepositAmount) revert Fiat24CryptoDeposit__
UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount);

        if (_feeAmountViaUsdc >= MAX_FEE_AMOUNT_USDC) {
            _feeAmountViaUsdc = MAX_FEE_AMOUNT_USDC;
        }

        uint256 usdcFactAmount = usdcAmount - _feeAmountViaUsdc;

if (usdcFactAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZe
        ro();

if (usdcFactAmount > maxUsdcDepositAmount) revert Fiat24CryptoDeposit_

_UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount)
        ;

if (usdcFactAmount < minUsdcDepositAmount) revert Fiat24CryptoDeposit_

_UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount);
        TransferHelper.safeTransfer(usdc, feeReceiver, _feeAmountViaUsdc);
        TransferHelper.safeTransfer(usdc, usdcDepositAddress,
usdcFactAmount);
```

Mantle: Resolved with PR-32

Zenith: Verified