BLOCKSEC

Security Audit

Report for fiat24contracts

Date: November 26, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	fiat24contracts

Version History

Version	Date	Description
1.0	November 26, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-11-27 09:49:56+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of fiat24contracts of Mantle.

Fiat24 is a pluggable fiat infrastructure for Web3 projects. It streamlines the fiat money transfer and payments for both traditional and web3 native users by providing experiences that they're most comfortable with. With support for most of the major payment channels such as SEPA, SIC, SWIFT and Debit Card processings, Fiat24 results in a standard payment provider to web3 users and applications.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- src/Fiat24CryptoDeposit_Base.sol
- src/Fiat24CryptoDeposit.sol
- src/Fiat24CryptoDeposit2.sol
- src/interfaces/IFiat24CryptoDeposit.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
project_name	`Version 0`	`482b3481099a98e1a7c4a88fabaf9703c1f4aa79`
	`Version 1`	`33661107245ac6c9c0a3fca51c5965a7587028f4`
	`Version 2`	`5a38a6ad8af8135dfa3de54d1cb9282345656368`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on,

[1] https://github.com/mantle-xyz/fiat24contracts

the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency

- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **three** recommendations and **five** notes.

- Medium Risk: 2
- Recommendation: 3
- Note: 5

ID	Severity	Description	Category	Status
1	Medium	Lack of native token support in the function `_swapViaAggregator()` of the contract `Fiat24CryptoDeposit_Base`	Security Issue	Fixed
2	Medium	Incorrect refund address in cross-chain messaging	Security Issue	Fixed
3	-	Lack of check on `msg.value` to be zero for ERC20 deposits in the contract `Fiat24CryptoDeposit`	Recommendation	Fixed
4	-	Lack of check when updating `minUsdcDepositAmount` and `maxUsdcDepositAmount`	Recommendation	Fixed
5	-	Ensure `_aggregator` is whitelisted before configuring function selector	Recommendation	Fixed
6	-	Design of gas fee handling and refund address in `depositTokenViaAggregator()`	Note	-
7	-	Incompatibility with fee-on-transfer tokens	Note	-
8	-	Potential Centralization Risks	Note	-
9	-	Aggregator integration for native tokens	Note	-
10	-	Commented `initialize()` function	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of native token support in the function `_swapViaAggregator()` of the contract `Fiat24CryptoDeposit_Base`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24CryptoDeposit_Base` implements the function `_swapViaAggregator()`, which processes token swaps through external aggregators. However, the function lacks support for native tokens, while the contract `Fiat24CryptoDeposit's` function `_swapViaAggregator()` implements the support for native tokens. This limitation prevents successful execution of swap operations via aggregators involving native tokens within the contract `Fiat24CryptoDeposit_Base`.

```
531   function _swapViaAggregator(
532       address _inputToken,
533       uint256 _amount,
534       address _aggregator,
535       bytes calldata _swapCalldata,
536       uint256 _minUsdcAmount
537   ) internal returns (uint256 usdcAmount) {
538       // Validate aggregator and function selector
539       if (!whitelistedAggregators[_aggregator]) revert
              Fiat24CryptoDeposit__NotWhitelistedAggregator(_aggregator);
540       if (_swapCalldata.length < 4) revert Fiat24CryptoDeposit__InvalidCalldata();
541
542       bytes4 selector = bytes4(_swapCalldata[0:4]);
543       if (!whitelistedSelectors[_aggregator][selector]) revert
              Fiat24CryptoDeposit__FunctionNotWhitelisted(selector);
544
545       if (_inputToken == usdc) {
546           return _amount;
547       }
548
549       uint256 usdcBalanceBefore = IERC20Upgradeable(usdc).balanceOf(address(this));
550
551       TransferHelper.safeApprove(_inputToken, _aggregator, _amount);
552
553       (bool success, ) = _aggregator.call(_swapCalldata);
554       if (!success) revert Fiat24CryptoDeposit__AggregatorSwapFailed();
555
556       uint256 usdcBalanceAfter = IERC20Upgradeable(usdc).balanceOf(address(this));
557       usdcAmount = usdcBalanceAfter - usdcBalanceBefore;
558
559       TransferHelper.safeApprove(_inputToken, _aggregator, 0);
560
561       // Validate user-specified minimum amount
562       if (usdcAmount < _minUsdcAmount) revert Fiat24CryptoDeposit__SlippageExceeded(usdcAmount,
              _minUsdcAmount);
563
564       return usdcAmount;
565   }
```

Listing 2.1: src/Fiat24CryptoDeposit_Base.sol

```
607   function _swapViaAggregator(
608       address _inputToken,
609       uint256 _amount,
610       address _aggregator,
611       bytes calldata _swapCalldata,
612       uint256 _minUsdcAmount
613   ) internal returns (uint256 usdcAmount) {
614       // Validate aggregator and function selector
615       if (!whitelistedAggregators[_aggregator]) revert
              Fiat24CryptoDeposit__NotWhitelistedAggregator(_aggregator);
616       if (_swapCalldata.length < 4) revert Fiat24CryptoDeposit__InvalidCalldata();
617
```

```
618         bytes4 selector = bytes4(_swapCalldata[0:4]);
619         if (!whitelistedSelectors[_aggregator][selector]) revert
                Fiat24CryptoDeposit__FunctionNotWhitelisted(selector);
620
621         if (_inputToken == usdc) {
622             return _amount;
623         }
624
625         uint256 usdcBalanceBefore = IERC20Upgradeable(usdc).balanceOf(address(this));
626
627         if (_inputToken == address(0) || _inputToken == 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)
                {
628             // For MNT, call aggregator with value
629             (bool success, ) = _aggregator.call{value: _amount}(_swapCalldata);
630             if (!success) revert Fiat24CryptoDeposit__AggregatorSwapFailed();
631         } else {
632             // For ERC20 tokens, approve and call
633             TransferHelper.safeApprove(_inputToken, _aggregator, _amount);
634             (bool success, ) = _aggregator.call(_swapCalldata);
635             if (!success) revert Fiat24CryptoDeposit__AggregatorSwapFailed();
636             TransferHelper.safeApprove(_inputToken, _aggregator, 0);
637         }
638
639         uint256 usdcBalanceAfter = IERC20Upgradeable(usdc).balanceOf(address(this));
640         usdcAmount = usdcBalanceAfter - usdcBalanceBefore;
641
642         // Validate user-specified minimum amount
643         if (usdcAmount < _minUsdcAmount) revert Fiat24CryptoDeposit__SlippageExceeded(usdcAmount,
                _minUsdcAmount);
644
645         return usdcAmount;
646     }
```

Listing 2.2: src/Fiat24CryptoDeposit.sol

Impact This limitation prevents successful execution of swap operations via aggregators involving native tokens within the contract `Fiat24CryptoDeposit_Base`.

Suggestion Revise the logic accordingly.

2.1.2 Incorrect refund address in cross-chain messaging

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24CryptoDeposit_Base` implements the function `depositTokenViaUsdc()`. However, the function designates the `feeReceiver` as the refund address for unused native fees, creating a logical contradiction where users who pay the native fees cannot recover their excess deposits. The function `depositTokenViaUsdc()` in the contract `Fiat24CryptoDeposit2` has the same question.

```
154   function depositTokenViaUsdc(address _inputToken, address _outputToken, uint256 _amount,
          uint256 _amountOutMinimum) nonReentrant payable external returns (uint256) {
155       if (paused()) revert Fiat24CryptoDeposit__Paused();
156       if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
157       if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
              _outputToken);
158
159       TransferHelper.safeTransferFrom(_inputToken, _msgSender(), address(this), _amount);
160       uint256 usdcAmount = _swapToUsdc(_inputToken, _amount, _amountOutMinimum, false);
161
162       if (usdcAmount > maxUsdcDepositAmount) revert
              Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount
              );
163       if (usdcAmount < minUsdcDepositAmount) revert
              Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount)
              ;
164
165       TransferHelper.safeTransfer(usdc, usdcDepositAddress, usdcAmount);
166       _sendLayerZeroMessage(_msgSender(), _inputToken, _amount, usdcAmount, _outputToken, msg.
              value, payable(feeReceiver));
167
168       emit SentDepositedTokenViaUsd(_msgSender(), _inputToken, _outputToken, _amount, usdcAmount)
              ;
169       return usdcAmount;
170   }
```

Listing 2.3: src/Fiat24CryptoDeposit_Base.sol

Impact Users who pay the native fees cannot recover their excess deposits.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Lack of check on `msg.value` to be zero for ERC20 deposits in the contract `Fiat24CryptoDeposit`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit`, the functions `depositTokenViaAggregator()` and `depositTokenViaAggregatorToAccount()` support deposits of both native tokens and ERC20 tokens. If a user invokes these functions to deposit ERC20 tokens but inadvertently attaches a `msg.value`, the transaction will succeed and the user will lose the attached native tokens. It is recommended to check the `msg.value` is zero in the ERC20 branch to avoid loss.

```
219   function depositTokenViaAggregator(
220       address _inputToken,
221       address _outputToken,
222       uint256 _amount,
223       address _aggregator,
```

```
224         bytes calldata _swapCalldata,
225         uint256 _minUsdcAmount,
226         uint256 _feeAmountViaUsdc
227     ) nonReentrant payable external returns (uint256) {
228         if (paused()) revert Fiat24CryptoDeposit__Paused();
229         if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
230         if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
                _outputToken);
231         uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(_msgSender());
232         if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(_msgSender());
233 
234         address sender = _msgSender();
235 
236         // Handle MNT deposit
237         if (_inputToken == address(0) || _inputToken == 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)
                {
238             if (msg.value != _amount) revert Fiat24CryptoDeposit__ValueZero();
239         } else {
240             TransferHelper.safeTransferFrom(_inputToken, sender, address(this), _amount);
241         }
242 
243         uint256 usdcAmount = _swapViaAggregator(_inputToken, _amount, _aggregator, _swapCalldata,
                _minUsdcAmount);
244         uint256 usdcFactAmount = _processFeeAndValidation(usdcAmount, _feeAmountViaUsdc);
245 
246         emit SentDepositedTokenViaAggregator(sender, _inputToken, _outputToken, _amount,
                usdcFactAmount, _aggregator);
247         return _processDeposit(sender, _inputToken, _outputToken, _amount, usdcFactAmount, tokenId)
                ;
248     }
```

Listing 2.4: src/Fiat24CryptoDeposit.sol

```
258     function depositTokenViaAggregatorToAccount(
259         address _targetAccount,
260         address _inputToken,
261         address _outputToken,
262         uint256 _amount,
263         address _aggregator,
264         bytes calldata _swapCalldata,
265         uint256 _minUsdcAmount
266     ) nonReentrant payable external returns (uint256) {
267         if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
268         uint256 targetTokenId = _validateDepositToAccountParams(_targetAccount, _outputToken);
269 
270         address sender = _msgSender();
271 
272         // Handle MNT deposit
273         if (_inputToken == address(0) || _inputToken == 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)
                {
274             if (msg.value != _amount) revert Fiat24CryptoDeposit__ValueZero();
275             // MNT is already in msg.value, will be used in swap
276         } else {
```

```
277             TransferHelper.safeTransferFrom(_inputToken, sender, address(this), _amount);
278         }
279
280         uint256 usdcAmount = _swapViaAggregator(_inputToken, _amount, _aggregator, _swapCalldata,
                _minUsdcAmount);
281
282         if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
283
284         emit SentDepositedTokenViaAggregator(sender, _inputToken, _outputToken, _amount, usdcAmount
                , _aggregator);
285         emit DepositToAccount(sender, _targetAccount, _inputToken, _amount);
286         return _processDeposit(_targetAccount, _inputToken, _outputToken, _amount, usdcAmount,
                targetTokenId);
287     }
```

Listing 2.5: src/Fiat24CryptoDeposit.sol

Suggestion It is recommended to require the `msg.value` to be zero in the ERC20 branch to avoid loss.

2.2.2 Lack of check when updating `minUsdcDepositAmount` and `maxUsdcDepositAmount`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The functions `changeMaxUsdcDepositAmount()` and `changeMinUsdcDepositAmount()` in the contract `Fiat24CryptoDeposit` update the deposit limits without validating that the minimum amount remains less than the maximum amount.

The contracts `Fiat24CryptoDeposit2` and `Fiat24CryptoDeposit_Base` have the same issue.

```
357     function changeMaxUsdcDepositAmount(uint256 _maxUsdcDepositAmount) external {
358         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
359         maxUsdcDepositAmount = _maxUsdcDepositAmount;
360     }
361
362     function changeMinUsdcDepositAmount(uint256 _minUsdcDepositAmount) external {
363         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
364         minUsdcDepositAmount = _minUsdcDepositAmount;
365     }
```

Listing 2.6: src/Fiat24CryptoDeposit.sol

```
418     function changeMaxUsdcDepositAmount(uint256 _maxUsdcDepositAmount) external {
419         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
420         maxUsdcDepositAmount = _maxUsdcDepositAmount;
421     }
422
423     function changeMinUsdcDepositAmount(uint256 _minUsdcDepositAmount) external {
```

```
424         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
425         minUsdcDepositAmount = _minUsdcDepositAmount;
426     }
```

Listing 2.7: src/Fiat24CryptoDeposit2.sol

```
324     function changeMaxUsdcDepositAmount(uint256 _maxUsdcDepositAmount) external {
325         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
326         maxUsdcDepositAmount = _maxUsdcDepositAmount;
327     }
328
329     function changeMinUsdcDepositAmount(uint256 _minUsdcDepositAmount) external {
330         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
331         minUsdcDepositAmount = _minUsdcDepositAmount;
332     }
```

Listing 2.8: src/Fiat24CryptoDeposit_Base.sol

Suggestion It is recommended to add checks in both functions to ensure that the `maxUsdcDepositAmount` is always greater than the `minUsdcDepositAmount` when updating either value.

2.2.3 Ensure `_aggregator` is whitelisted before configuring function selector

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contracts `Fiat24CryptoDeposit`, `Fiat24CryptoDeposit2` and `Fiat24CryptoDeposit_Base`, the function `setFunctionSelector()` allows the `OPERATOR_ADMIN_ROLE` to configure allowed `_selector` for an `_aggregator` address. It is recommended to add a validation step to ensure that the `_aggregator` address is whitelisted before configuring the selector.

```
675     /// @notice Add or remove aggregator from whitelist
676     /// @param _aggregator The aggregator contract address
677     /// @param _isWhitelisted True to whitelist, false to remove
678     /// @dev Only OPERATOR_ADMIN_ROLE can manage aggregator whitelist
679     function setAggregatorWhitelist(address _aggregator, bool _isWhitelisted) external {
680         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
681         require(_aggregator != address(0), "Invalid aggregator address");
682         whitelistedAggregators[_aggregator] = _isWhitelisted;
683         emit AggregatorWhitelistUpdated(_aggregator, _isWhitelisted);
684     }
685
686     /// @notice Add or remove function selector for a specific aggregator
687     /// @param _aggregator The aggregator contract address
688     /// @param _selector The function selector (first 4 bytes of function signature)
689     /// @param _isWhitelisted True to whitelist, false to remove
690     function setFunctionSelector(address _aggregator, bytes4 _selector, bool _isWhitelisted)
            external {
```

```
691         if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
                Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
692         require(_aggregator != address(0), "Invalid aggregator address");
693         require(_selector != bytes4(0), "Invalid selector");
694         whitelistedSelectors[_aggregator][_selector] = _isWhitelisted;
695         emit FunctionSelectorWhitelisted(_aggregator, _selector, _isWhitelisted);
696     }
```

Listing 2.9: src/Fiat24CryptoDeposit.sol

Suggestion Add a validation step to ensure that the `_aggregator` address is whitelisted before configuring the selector.

2.3 Note

2.3.1 Design of gas fee handling and refund address in `depositTokenViaAggregator()`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit_Base`, the function `depositTokenViaAggregator()` is invoked by the project through a user that implements `EIP-7702`. Under this design, the variable `_feeAmountViaUsdc` is used as part of the mechanism for collecting gas-related fees, and `feeReceiver` is designated as the refund address for LayerZero.

The same mechanism is used in the `depositTokenViaAggregator()` function of `Fiat24CryptoDeposit2`.

```
369     function depositTokenViaAggregator(
370         address _inputToken,
371         address _outputToken,
372         uint256 _amount,
373         address _aggregator,
374         bytes calldata _swapCalldata,
375         uint256 _minUsdcAmount,
376         uint256 _feeAmountViaUsdc
377     ) nonReentrant payable external returns (uint256) {
378         if (paused()) revert Fiat24CryptoDeposit__Paused();
379         if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
380         if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
                _outputToken);
381
382         address sender = _msgSender();
383
384         TransferHelper.safeTransferFrom(_inputToken, sender, address(this), _amount);
385
386         uint256 usdcFactAmount;
387         {
388             uint256 usdcAmount = _swapViaAggregator(_inputToken, _amount, _aggregator,
                    _swapCalldata, _minUsdcAmount);
389             usdcFactAmount = _processFeeAndValidation(usdcAmount, _feeAmountViaUsdc);
390             TransferHelper.safeTransfer(usdc, usdcDepositAddress, usdcFactAmount);
```

```
391         }
392
393         // Send cross-chain message via LayerZero
394         _lzSend(
395             dstId,
396             abi.encode(sender, _inputToken, _amount, usdcFactAmount, _outputToken),
397             OptionsBuilder.newOptions().addExecutorLzReceiveOption(relay_gas_limit, 0),
398             MessagingFee({nativeFee: msg.value, lzTokenFee: 0}),
399             payable(msg.sender)
400         );
401
402         emit SentDepositedTokenViaAggregator(sender, _inputToken, _outputToken, _amount,
                usdcFactAmount, _aggregator);
403         return usdcFactAmount;
404     }
```

Listing 2.10: src/Fiat24CryptoDeposit_Base.sol

Feedback from the project The project team states that they use the EIP-7702 mechanism to pay gas fees on behalf of users, and then use the variable `_feeAmountViaUsdc` to collect the corresponding amount of tokens as payment for those gas fees.

2.3.2 Incompatibility with fee-on-transfer tokens

Introduced by Version 1

Description The deposit logic and the aggregator swap logic in the contract directly use the user-provided amount as the input for subsequent swap operations, which makes the contract incompatible with fee-on-transfer tokens.

2.3.3 Potential Centralization Risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, `OPERATOR_ADMIN_ROLE` can add aggregator addresses through the function `setAggregatorWhitelist()`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.4 Aggregator integration for native tokens

Introduced by Version 1

Description In this project, both the `address(0)` and `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeEEeE` are accepted as valid representations for the native token when used as `_inputToken` in aggregator swaps. However, it should be noted that individual aggregators may not consistently support both address formats for input token being as native token.

2.3.5 Commented `initialize()` function

Introduced by `Version 1`

Description In this project, the function `initialize()` is fully commented out in the contracts `Fiat24CryptoDeposit_Base`, `Fiat24CryptoDeposit` and `Fiat24CryptoDeposit2`.

```
83  // function initialize(
84  //     address admin,
85  //     address _delegate,
86  //     address _usd24,
87  //     address _eur24,
88  //     address _chf24,
89  //     address _gbp24,
90  //     address _cnh24,
91  //     address _usdc,
92  //     address _weth,
93  //     address _usdcDepositAddress,
94  //     address _feeReceiver,
95  //     uint32 _dstId
96  // ) public initializer {
```

Listing 2.11: src/Fiat24CryptoDeposit2.sol

```
83  // function initialize(
84  //     address admin,
85  //     address _delegate,
86  //     address _usd24,
87  //     address _eur24,
88  //     address _chf24,
89  //     address _gbp24,
90  //     address _cnh24,
91  //     address _usdc,
92  //     address _weth,
93  //     address _usdcDepositAddress,
94  //     address _feeReceiver,
95  //     uint32 _dstId
96  // ) public initializer {
```

Listing 2.12: src/Fiat24CryptoDeposit_Base.sol

```
98  //     function initialize(
99  //     address admin,
100 //     address _fiat24account,
101 //     address _usd24,
102 //     address _eur24,
103 //     address _chf24,
104 //     address _gbp24,
105 //     address _cnh24,
106 //     address _usdc,
107 //     address _usdcDepositAddress,
108 //     address _feeReceiver,
109 //     address _fiat24CryptoRelayAddress
110 // ) public initializer {
```

Listing 2.13: src/Fiat24CryptoDeposit.sol

Feedback from the project The project team stated that the implementation contract has already been deployed and initialized. During this upgrade of the logic contract, the `initialize()` function is not required and has been temporarily commented out to reduce contract code size.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS