



Security Audit

Report for Fiat24 Contracts

Date: August 6, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction	1
1.1 About Target Contracts	1
1.2 Disclaimer	2
1.3 Procedure of Auditing	2
1.3.1 Security Issues	2
1.3.2 Additional Recommendation	3
1.4 Security Model	3
Chapter 2 Findings	5
2.1 Security Issue	6
2.1.1 Incorrect permission check in function <code>updateExchangeRate()</code>	6
2.1.2 Potential front-running attacks when updating exchange rates	7
2.1.3 Fixed exchange rates during initialization creates front-running risk	8
2.1.4 Potential DoS risk in the function <code>_removeFailedKey()</code>	9
2.1.5 Inconsistent mechanism of updating exchange rates	10
2.1.6 Incorrect rounding direction in the functions <code>authorize()</code> and <code>increment()</code>	11
2.1.7 Lack of checks for the parameters <code>cardCurrency_</code> and <code>originalPaidCurrency_</code>	12
2.1.8 <code>Fiat24</code> tokens received by a <code>fiat24account</code> with specific <code>status</code> will be locked	14
2.1.9 Inconsistent access control	15
2.2 Recommendation	16
2.2.1 Inconsistency between the comment and the codes	16
2.2.2 Lack of duplication check on the <code>fiatName</code> in the function <code>addFiatToken()</code>	17
2.2.3 Add zero address checks	18
2.2.4 Lack of duplication check in the function <code>addTokenAddress()</code>	19
2.2.5 Confusing naming for the variable <code>_amountOutMinimum</code>	19
2.2.6 Lack of non zero value check in the function <code>updateExchangeRates()</code>	21
2.3 Note	23
2.3.1 Atomicity in <code>Fiat24</code> card authorization process	23
2.3.2 Lack of fiat tokens removal mechanism	24
2.3.3 The parameter <code>_amountOutMinimum</code> should be validated in the backend	24
2.3.4 Upgrade the implementation of <code>Fiat24Token</code> properly	25
2.3.5 Ensure that the <code>exchangeRates</code> and <code>validXXX24Tokens</code> are set properly	27
2.3.6 Initialize the implementation contracts immediately after deployments	27
2.3.7 Potential centralization risks	27

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24 Contracts

Version History

Version	Date	Description
1.0	August 6, 2025	First release

Signature

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 Fiat24 Contracts of Mantle.

Fiat24 is a digital banking platform built on blockchain technology that bridges banking services with the crypto ecosystem. The platform provides NFT-based digital accounts as unique identifiers for users, with each account represented as an ERC-721 token featuring customizable features and status management. Fiat24 supports multiple fiat currencies through tokenized representations, including USD24, EUR24, CHF24, GBP24, and CNH24, with real-time exchange rates and seamless cross-currency transactions. The platform features crypto deposit functionality that enables users to deposit USDC and other cryptocurrencies, automatically converting them to fiat tokens at current market rates.

Note this audit only focuses on the smart contracts in the following directories/files:

- fiat24contracts/src/Fiat24CryptoDeposit.sol
- fiat24contracts/src/Fiat24CryptoDeposit2.sol
- fiat24contracts/src/Fiat24CryptoDeposit_Base.sol
- fiat24contracts/src/Fiat24CardAuthorizationMargeta.sol
- fiat24contracts/src/Fiat24CryptoRelay.sol
- fiat24contracts/src/FiatTokenBeacon.sol
- fiat24contracts/src/FiatTokenFactory.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.

Project	Version	Commit Hash
Fiat24 Contracts	Version 1	32b66f10a42b9ba39de279312754160d20d2100d
	Version 2	8fa9f76352a27f901c293552ed1c03b06c9bb3f4

¹<https://github.com/mantle-xyz/fiat24contracts>

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, 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	High	High	Medium
	Low	Medium	Low
		High	Low
		Likelihood	

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.

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

³<https://cwe.mitre.org/>

- **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.

Chapter 2 Findings

In total, we found **nine** potential security issues. Besides, we have **six** recommendations and **seven** notes.

- High Risk: 1
- Medium Risk: 2
- Low Risk: 6
- Recommendation: 6
- Note: 7

ID	Severity	Description	Category	Status
1	High	Incorrect permission check in function <code>updateExchangeRate()</code>	Security Issue	Fixed
2	Medium	Potential front-running attacks when updating exchange rates	Security Issue	Confirmed
3	Medium	Fixed exchange rates during initialization creates front-running risk	Security Issue	Confirmed
4	Low	Potential DoS risk in the function <code>_removeFailedKey()</code>	Security Issue	Fixed
5	Low	Inconsistent mechanism of updating exchange rates	Security Issue	Confirmed
6	Low	Incorrect rounding direction in the functions <code>authorize()</code> and <code>increment()</code>	Security Issue	Confirmed
7	Low	Lack of checks for the parameters <code>cardCurrency_</code> and <code>originalPaidCurrency_</code>	Security Issue	Confirmed
8	Low	<code>Fiat24</code> tokens received by a <code>fiat24account</code> with specific <code>status</code> will be locked	Security Issue	Confirmed
9	Low	Inconsistent access control	Security Issue	Fixed
10	-	Inconsistency between the comment and the codes	Recommendation	Fixed
11	-	Lack of duplication check on the <code>fiatName</code> in the function <code>addFiatToken()</code>	Recommendation	Fixed
12	-	Add zero address checks	Recommendation	Confirmed
13	-	Lack of duplication check in the function <code>addTokenAddress()</code>	Recommendation	Confirmed
14	-	Confusing naming for the variable <code>_amountOutMinimum</code>	Recommendation	Confirmed
15	-	Lack of non zero value check in the function <code>updateExchangeRates()</code>	Recommendation	Confirmed

16	-	Atomicity in Fiat24 card authorization process	Note	-
17	-	Lack of fiat tokens removal mechanism	Note	-
18	-	The parameter <code>_amountOutMinimum</code> should be validated in the backend	Note	-
19	-	Upgrade the implementation of Fiat24Token properly	Note	-
20	-	Ensure that the <code>exchangeRates</code> and <code>validXXX24Tokens</code> are set properly	Note	-
21	-	Initialize the implementation contracts immediately after deployments	Note	-
22	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect permission check in function `updateExchangeRate()`

Severity High

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description Since the permission check for `msg.sender` is implemented in the function `_updateExchangeRate()` in the contract `Fiat24CardAuthorizationMarqeta`, a malicious attacker could exploit this by passing empty arrays (`fiatTokens` and `rates` with length 0) to the function `updateExchangeRates()`. This circumvents the authorization logic in the function `_updateExchangeRate()` while still allowing the attacker to modify the `marketClosed` value arbitrarily. Since the variable `marketClosed` affects spread calculations, this could ultimately lead to potential financial loss. The same problem exists in the contract `Fiat24CryptoRelay`.

```

381 function updateExchangeRates(
382     address[] calldata fiatTokens,
383     uint256[] calldata rates,
384     bool isMarketClosed
385 ) external {
386     require(fiatTokens.length == rates.length, "Arrays length mismatch");
387     marketClosed = isMarketClosed;
388     for (uint256 i = 0; i < fiatTokens.length; i++) {
389         address token = fiatTokens[i];
390         uint256 rate = rates[i];
391         require(validXXX24Tokens[token], "Invalid token");
392         require(rate > 0, "Rate must be >0");
393         _updateExchangeRate(token, rate, isMarketClosed);
394     }
395 }
```

```
396
397 /// @notice Updating the exchange rate between USD and individual fiat currencies
398 function _updateExchangeRate(address _fiatToken, uint256 _rateUsdcToFiat, bool _isMarketClosed
    ) internal {
399
400     uint256 oldRate = exchangeRates[usd24Address][_fiatToken];
401
402     if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
403         exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
404         emit ExchangeRateUpdatedByOperator(_fiatToken, oldRate, _rateUsdcToFiat,
            _isMarketClosed);
405     } else if (hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender())) {
406
407         uint256 rateDiff = oldRate > _rateUsdcToFiat ? (oldRate - _rateUsdcToFiat) : (
            _rateUsdcToFiat - oldRate);
408         rateDiff = rateDiff * 10000 / oldRate;
409         require(rateDiff < 300, "Rate Update Robot: change too large");
410         exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
411         emit ExchangeRateUpdatedByRobot(_fiatToken, oldRate, _rateUsdcToFiat, _isMarketClosed);
412     } else {
413         revert Fiat24CardAuthorizationMarqeta__NotRateUpdater((_msgSender()));
414     }
415 }
```

Listing 2.1: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could ultimately lead to potential financial loss.

Suggestion Add a permission check in the function `updateExchangeRates()`.

2.1.2 Potential front-running attacks when updating exchange rates

Severity Medium

Status Confirmed

Introduced by Version 1

Description In both the contracts `Fiat24CardAuthorizationMarqeta` and `Fiat24CryptoRelay`, exchange rate updates performed by the `RATES_UPDATER_OPERATOR_ROLE` and `RATES_UPDATER_ROBOT_ROLE` are vulnerable to front-running attacks. A malicious user could:

1. Monitor pending rate update transactions in the mempool.
2. Front-run the update by executing advantageous trades. For example, when the exchange rate rises, the user could front-run to exchange for cheaper `fiatTokens`.

```
381 function updateExchangeRates(
382     address[] calldata fiatTokens,
383     uint256[] calldata rates,
384     bool isMarketClosed
385 ) external {
386     require(fiatTokens.length == rates.length, "Arrays length mismatch");
387     marketClosed = isMarketClosed;
388     for (uint256 i = 0; i < fiatTokens.length; i++) {
389         address token = fiatTokens[i];
```

```
390     uint256 rate = rates[i];
391     require(validXXX24Tokens[token], "Invalid token");
392     require(rate > 0, "Rate must be >0");
393     _updateExchangeRate(token, rate, isMarketClosed);
394 }
395 }
```

Listing 2.2: src/Fiat24CardAuthorizationMarqeta.sol

Impact A malicious user could front-run to exchange for cheaper `fiatTokens`.

Suggestion Revise the logic accordingly.

Feedback from the project The project states that an exchange fee is charged during the exchange process, and there is a slight possibility of benefiting from front-running. Furthermore, the project states that the exchanges or spending operations for users are limited.

2.1.3 Fixed exchange rates during initialization creates front-running risk

Severity Medium

Status Confirmed

Introduced by [Version 1](#)

Description In the contracts `Fiat24CardAuthorizationMarqeta`, the `initialize()` function sets hardcoded exchange rates for `Fiat24` tokens (e.g., `EUR24`, `USD24`, `CHF24`). These rates are applied immediately upon deployment and initialization, before any dynamic updates can occur. This introduces a front-running risk:

1. A malicious actor could monitor the contract deployment and initialization, and immediately execute trades at the fixed rates before the protocol updates them.

2. Since the initial rates may not reflect real-time market prices, attackers could arbitrarily profit by exploiting mispriced conversions (e.g., buying undervalued tokens or selling overvalued ones).

This could lead to protocol losses if the initial rates are significantly off-market. The contracts `Fiat24CryptoDeposit` and `Fiat24CryptoRelay` have the same problem.

```
108     exchangeRates[usd24Address][usd24Address] = 10000;
109     exchangeRates[usd24Address][eur24Address] = 9168;
110     exchangeRates[usd24Address][chf24Address] = 8632;
111     exchangeRates[usd24Address][gbp24Address] = 7674;
112     exchangeRates[usd24Address][cnh24Address] = 70885;
```

Listing 2.3: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Revise the code logic accordingly.

Feedback from the project The project states that the rate used in the deployment is up to date at that time.

2.1.4 Potential DoS risk in the function `_removeFailedKey()`

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the contract `Fiat24CryptoRelay`, the function `_lzReceive()` attempts to process messages from `LayerZero` and append the failed `messageId` to the state variable `failedKeys`. The only way to remove a `messageId` is by invoking the function `_removeFailedKey()`, which will attempt to iterate through the entire `failedKeys` array. A malicious actor could create numerous failing messages, potentially causing the array iteration to exceed the block gas limit and eventually resulting in DoS in the functions `retryFailedMessage()` and `adminProcessFailedMessage()`.

```

470 function _removeFailedKey(bytes32 messageId) internal {
471     uint256 len = failedKeys.length;
472     for (uint256 i = 0; i < len; i++) {
473         if (failedKeys[i] == messageId) {
474             if (i < len - 1) {
475                 failedKeys[i] = failedKeys[len - 1];
476             }
477             failedKeys.pop();
478             break;
479         }
480     }
481 }

```

Listing 2.4: `src/Fiat24CryptoRelay.sol`

```

241 function adminProcessFailedMessage(bytes32 messageId) external {
242     if (!hasRole(OPERATOR_ROLE, _msgSender())) revert Fiat24CryptoDeposit__NotOperator(
        _msgSender());
243     bytes memory payload = _failedPayloads[messageId];
244     require(payload.length > 0, "No failed message to retry");
245     delete _failedPayloads[messageId];
246     _removeFailedKey(messageId);
247     emit FailedMessageProcessed(messageId);
248 }

```

Listing 2.5: `src/Fiat24CryptoRelay.sol`

```

218 function retryFailedMessage(bytes32 messageId) external {
219     bytes memory payload = _failedPayloads[messageId];
220     require(payload.length > 0, "No failed message to retry");
221
222     delete _failedPayloads[messageId];
223
224     try this.processMessage(payload) {
225         _removeFailedKey(messageId);
226         emit MessageRetried(messageId, true, "");
227     } catch Error(string memory reason) {
228         _failedPayloads[messageId] = payload;
229         emit MessageRetried(messageId, false, reason);

```

```

230     } catch {
231         _failedPayloads[messageId] = payload;
232         emit MessageRetried(messageId, false, "Unknown failure");
233     }
234 }

```

Listing 2.6: src/Fiat24CryptoRelay.sol

Impact This may cause a potential DoS risk.

Suggestion Implement a mechanism that allows partial processing of `failedKeys` instead of requiring a full traversal.

2.1.5 Inconsistent mechanism of updating exchange rates

Severity Low

Status Confirmed

Introduced by Version 1

Description The contract `Fiat24CardAuthorizationMarqeta` currently implements two different mechanisms for batch updating exchange rates. The first approach updates individual rates that meet the `rateDiff < 300` requirement while skipping non-compliant ones, and always updates the variable `marketClosed`. The second approach rejects the entire batch update if any single rate fails the `rateDiff < 300` check, including preventing the `marketClosed` update. The contract `Fiat24CryptoRelay` has the same problem. This could create unpredictable system behavior depending on which update mechanism is triggered.

```

309     } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender())) {
310         uint256 rateDiff_usd_eur = (exchangeRates[usd24Address][eur24Address] > _usd_eur)
311             ? (exchangeRates[usd24Address][eur24Address] - _usd_eur)
312             : (_usd_eur - exchangeRates[usd24Address][eur24Address]);
313         rateDiff_usd_eur = (rateDiff_usd_eur * 10000) / exchangeRates[usd24Address][
            eur24Address];
314         uint256 rateDiff_usd_chf = (exchangeRates[usd24Address][chf24Address] > _usd_chf)
315             ? (exchangeRates[usd24Address][chf24Address] - _usd_chf)
316             : (_usd_chf - exchangeRates[usd24Address][chf24Address]);
317         rateDiff_usd_chf = (rateDiff_usd_chf * 10000) / exchangeRates[usd24Address][
            chf24Address];
318         uint256 rateDiff_usd_gbp = (exchangeRates[usd24Address][gbp24Address] > _usd_gbp)
319             ? (exchangeRates[usd24Address][gbp24Address] - _usd_gbp)
320             : (_usd_gbp - exchangeRates[usd24Address][gbp24Address]);
321         rateDiff_usd_gbp = (rateDiff_usd_gbp * 10000) / exchangeRates[usd24Address][
            gbp24Address];
322         uint256 rateDiff_usd_cnh = (exchangeRates[usd24Address][cnh24Address] > _usd_cnh)
323             ? (exchangeRates[usd24Address][cnh24Address] - _usd_cnh)
324             : (_usd_cnh - exchangeRates[usd24Address][cnh24Address]);
325         rateDiff_usd_cnh = (rateDiff_usd_cnh * 10000) / exchangeRates[usd24Address][
            cnh24Address];
326         if (rateDiff_usd_eur < 300) exchangeRates[usd24Address][eur24Address] = _usd_eur;
327         if (rateDiff_usd_chf < 300) exchangeRates[usd24Address][chf24Address] = _usd_chf;
328         if (rateDiff_usd_gbp < 300) exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
329         if (rateDiff_usd_cnh < 300) exchangeRates[usd24Address][cnh24Address] = _usd_cnh;

```

```

330     marketClosed = _isMarketClosed;
331     emit ExchangeRatesUpdatedByRobot(
332         _msgSender(),
333         exchangeRates[usd24Address][eur24Address],
334         exchangeRates[usd24Address][chf24Address],
335         exchangeRates[usd24Address][gbp24Address],
336         exchangeRates[usd24Address][cnh24Address],
337         marketClosed
338     );
339 } else {

```

Listing 2.7: src/Fiat24CardAuthorizationMarqeta.sol

```

405 } else if (hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender())) {
406
407     uint256 rateDiff = oldRate > _rateUsdcToFiat ? (oldRate - _rateUsdcToFiat) : (
408         _rateUsdcToFiat - oldRate);
409     rateDiff = rateDiff * 10000 / oldRate;
410     require(rateDiff < 300, "Rate Update Robot: change too large");
411     exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
412     emit ExchangeRateUpdatedByRobot(_fiatToken, oldRate, _rateUsdcToFiat, _isMarketClosed);
413 } else {

```

Listing 2.8: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could create unpredictable system behavior depending on which update mechanism is triggered.

Suggestion Uniform the two mechanisms.

Feedback from the project The project states that they will delete one of the two functions `updateExchangeRates()` in the future.

2.1.6 Incorrect rounding direction in the functions `authorize()` and `increment()`

Severity Low

Status Confirmed

Introduced by Version 1

Description In the contract `Fiat24CardAuthorizationMarqeta`, the functions `authorize()` and `increment()` round down when calculating the value of `paidAmount`, which should be transferred from the user account to the `booked`. This could cause a loss to the protocol.

```

137     if (validXXX24Tokens[XXX24Tokens[transactionCurrency_]]) {
138         if (
139             IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).balanceOf(sender) >=
140                 transactionAmount_
141             && IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).allowance(sender,
142                 address(this)) >= transactionAmount_
143         ) {
144             paidCurrency = XXX24Tokens[transactionCurrency_];
145             paidAmount = transactionAmount_;
146         } else {

```

```

145         paidAmount = transactionAmount_ * getRate(XXX24Tokens[transactionCurrency_],
146             cardCurrency_)
147             * getSpread(XXX24Tokens[transactionCurrency_], cardCurrency_, false) /
148                 100000000;
149     }
150 } else {
151     if (settlementCurrency_ != eur24Address) revert
152         Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(
153             settlementCurrency_);
154     paidAmount =
155         settlementAmount_ * (100 + interchange) * getRate(eur24Address, cardCurrency_) *
156             getSpread(eur24Address, cardCurrency_, false) / 10000000000;
157 }

```

Listing 2.9: src/Fiat24CardAuthorizationMarqeta.sol

```

176     if (validXXX24Tokens[XXX24Tokens[transactionCurrency_]]) {
177         if (
178             IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).balanceOf(sender) >=
179                 transactionAmount_
180             && IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).allowance(sender,
181                 address(this)) >= transactionAmount_
182         ) {
183             paidCurrency = XXX24Tokens[transactionCurrency_];
184             paidAmount = transactionAmount_;
185         } else {
186             paidCurrency = cardCurrency_;
187             paidAmount = transactionAmount_ * getRate(XXX24Tokens[transactionCurrency_],
188                 cardCurrency_)
189                 * getSpread(XXX24Tokens[transactionCurrency_], cardCurrency_, false) /
190                     100000000;
191         }
192     } else {
193         if (settlementCurrency_ != eur24Address) revert
194             Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(
195                 settlementCurrency_);
196         paidCurrency = cardCurrency_;
197         paidAmount =
198             settlementAmount_ * (100 + interchange) * getRate(eur24Address, cardCurrency_) *
199                 getSpread(eur24Address, cardCurrency_, false) / 10000000000;
200     }

```

Listing 2.10: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Round up when calculating the value of the variable `paidAmount` in the functions `authorize()` and `increment()`.

2.1.7 Lack of checks for the parameters `cardCurrency_` and `originalPaidCurrency_`

Severity Low

Status Confirmed

Introduced by Version 1

Description The contract `Fiat24CardAuthorizationMarqeta` does not verify whether the input parameters `cardCurrency_` and `originalPaidCurrency_` have corresponding `validXXX24Tokens` values set to true. This oversight could allow processing of invalid currencies that are not registered in `validXXX24Tokens`, ultimately leading to exchange rate lookups returning zero values, which may cause a loss to the protocol.

```
119 function authorize(  
120     string memory authorizationToken_,  
121     string memory cardId_,  
122     uint256 tokenId_,  
123     address cardCurrency_,  
124     string memory transactionCurrency_,  
125     address settlementCurrency_,  
126     uint256 transactionAmount_,  
127     uint256 settlementAmount_  
128 ) public {  
129     if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert  
130         Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());  
131     if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();  
132     if (!validXXX24Tokens[settlementCurrency_]) revert  
133         Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);  
134     address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);  
135     address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);  
136     address paidCurrency = cardCurrency_;
```

Listing 2.11: src/Fiat24CardAuthorizationMarqeta.sol

```
200 function advice(  
201     string memory authorizationToken_,  
202     string memory originalAuthorizationToken_,  
203     string memory cardId_,  
204     uint256 tokenId_,  
205     string memory transactionCurrency_,  
206     address settlementCurrency_,  
207     uint256 transactionAmount_,  
208     uint256 settlementAmount_,  
209     address originalPaidCurrency_  
210 ) public {  
211     if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert  
212         Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());  
213     if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();  
214     if (!validXXX24Tokens[settlementCurrency_]) revert  
215         Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);  
216     address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);  
217     address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);  
218     address paidCurrency = originalPaidCurrency_; // Always pay back to the same currency
```

Listing 2.12: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Add validations for the parameters `cardCurrency_` and `originalPaidCurrency_`.

Feedback from the project The project states that they will fix this issue in a future version.

2.1.8 Fiat24 tokens received by a fiat24account with specific status will be locked

Severity Low

Status Confirmed

Introduced by Version 1

Description In the contract `Fiat24Token`, the function `tokenTransferAllowed()` allows a `fiat24account` whose `status` is either `Na` or `Tourist` to receive `Fiat24` tokens. However, due to the `tokenTransferAllowed()` check, these accounts are unable to use the received tokens. For example, if the users want to pay out by invoking the function `clientPayout()`, they will fail, since only the accounts with the `status.Live` can transfer `Fiat24` tokens to other addresses. As a result, the `Fiat24` tokens held by the accounts whose `status` is either `Na` or `Tourist` become locked.

```
286 function tokenTransferAllowed(address from, address to, uint256 amount) public view returns (
    bool) {
287     require(!fiat24account.paused(), "Fiat24Token: All account transfers are paused");
288     require(!paused(), "Fiat24Token: All account transfers of this currency are paused");
289     if (sanctionCheck) {
290         SanctionsList sanctionsList = SanctionsList(sanctionContract);
291         bool toIsSanctioned = sanctionsList.isSanctioned(to);
292         require(!toIsSanctioned, "Fiat24Token: Transfer to sanctioned address");
293         bool fromIsSanctioned = sanctionsList.isSanctioned(from);
294         require(!fromIsSanctioned, "Fiat24Token: Transfer from sanctioned address");
295     }
296     if (from != address(0) && to != address(0)) {
297         if (balanceOf(from) < amount) {
298             return false;
299         }
300         uint256 toAmount = amount + balanceOf(to);
301         Fiat24Account.Status fromClientStatus;
302         uint256 accountIdFrom = fiat24account.historicOwnership(from);
303         if (accountIdFrom != 0) {
304             fromClientStatus = fiat24account.status(accountIdFrom);
305         } else if (from != address(0) && fiat24account.balanceOf(from) > 0) {
306             fromClientStatus = Fiat24Account.Status.Tourist;
307             accountIdFrom = fiat24account.tokenOfOwnerByIndex(from, 0);
308         } else {
309             fromClientStatus = Fiat24Account.Status.Na;
310         }
311         Fiat24Account.Status toClientStatus;
312         uint256 accountIdTo = fiat24account.historicOwnership(to);
313         if (accountIdTo != 0) {
314             toClientStatus = fiat24account.status(accountIdTo);
315         } else if (to != address(0) && fiat24account.balanceOf(to) > 0) {
316             toClientStatus = Fiat24Account.Status.Tourist;
317             accountIdTo = fiat24account.tokenOfOwnerByIndex(to, 0);
318         } else {
319             toClientStatus = Fiat24Account.Status.Na;
```

```

320     }
321     uint256 amountInChf = convertToChf(amount);
322     bool fromLimitCheck = fiat24account.checkLimit(accountIdFrom, amountInChf);
323     bool toLimitCheck = fiat24account.checkLimit(accountIdTo, amountInChf);
324     // When the money from 91xx, we don't consider the client limit
325     if (accountIdFrom >= 9100 && accountIdFrom <= 9199) {
326         toLimitCheck = true;
327     }
328     return (
329         fromClientStatus == Fiat24Account.Status.Live
330         && (toClientStatus == Fiat24Account.Status.Live || toClientStatus ==
331             Fiat24Account.Status.SoftBlocked) && fromLimitCheck && toLimitCheck
332     ) || (
333         fromClientStatus == Fiat24Account.Status.Live && fromLimitCheck
334         && ((toClientStatus == Fiat24Account.Status.Na || toClientStatus ==
335             Fiat24Account.Status.Tourist) && toAmount <= LimitWalkin)
336     );
337     return false;
338 }

```

Listing 2.13: src/Fiat24Token.sol

```

179 function clientPayout(uint256 amount, string memory contactId) external {
180     require(amount >= minimalPayoutAmount, "Fiat24Token: amount < minimal payout amount");
181     uint256 tokenId = fiat24account.tokenOfOwnerByIndex(msg.sender, 0);
182     // string memory txid = string(abi.encodePacked(uintToString(tokenId), "-", uintToString(
183         ArbSys(address(100)).arbBlockNumber())));
184     string memory txid = string(abi.encodePacked(uintToString(tokenId), "-", uintToString(block
185         .number)));
186     transferByAccountId(9102, amount);
187     emit ClientPayout(tokenId, msg.sender, 9102, amount, contactId, txid);
188 }

```

Listing 2.14: src/Fiat24Token.sol

Impact Fiat tokens held by the accounts whose `status` is either `Na` or `Tourist` are locked.

Suggestion Revise the code logic accordingly.

Feedback from the project The project states that FiatToken will only be unlocked for users who have successfully passed the KYC verification process.

2.1.9 Inconsistent access control

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the contract `Fiat24CryptoDeposit_Base`, the functions `changeUsdcAddress()` and `changeUsdcDepositAddress()` are executed by the role `OPERATOR_ADMIN_ROLE`. However, in the contract `Fiat24CryptoDeposit2`, the same functions are used by the role `DEFAULT_ADMIN_ROLE`.

The difference in access control for the same function across contracts may lead to misoperations.

```
424 function changeUsdcAddress(address _usdcAddress) external {
425     if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
426     require(_usdcAddress != address(0), "Invalid usdc address");
427     usdc = _usdcAddress;
428 }
429
430 function changeUsdcDepositAddress(address _usdcDepositAddress) external {
431     if (!hasRole(DEFAULT_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
432     address oldUsdcDepositAddress = usdcDepositAddress;
433     usdcDepositAddress = _usdcDepositAddress;
434     emit UsdcDepositAddressChanged(oldUsdcDepositAddress, usdcDepositAddress);
435 }
```

Listing 2.15: src/Fiat24CryptoDeposit2.sol

```
421 function changeUsdcAddress(address _usdcAddress) external {
422     if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
423     require(_usdcAddress != address(0), "Invalid usdc address");
424     usdc = _usdcAddress;
425 }
426
427 function changeUsdcDepositAddress(address _usdcDepositAddress) external {
428     if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
429     address oldUsdcDepositAddress = usdcDepositAddress;
430     usdcDepositAddress = _usdcDepositAddress;
431     emit UsdcDepositAddressChanged(oldUsdcDepositAddress, usdcDepositAddress);
432 }
```

Listing 2.16: src/Fiat24CryptoDeposit_Base.sol

Impact Potential misoperations due to the inconsistent access control.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Inconsistency between the comment and the codes

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the contract `Fiat24CryptoDeposit_Base`, the comment for the function `quoteLayerzeroFee()` states, "Quotes the gas needed to pay for the full omnichain transaction in native gas or ZRO token." However, in the code implementation, when calling the function `_quote()`, the parameter `_payInLzToken` is set to false, indicating that the transaction fee is paid in only native gas, which is inconsistent with the description in the comment.

```
327  /**
328   * @notice Quotes the gas needed to pay for the full omnichain transaction in native gas or
329   *      ZRO token.
330   */
331  function quoteLayerzeroFee(
332      uint32 _dstEid,
333      address _userAddress,
334      address _inputToken,
335      uint256 _inputAmount,
336      uint256 _usdcAmount,
337      address _outputToken
338  ) public view returns (MessagingFee memory fee) {
339      bytes memory payload = abi.encode(
340          _userAddress,
341          _inputToken,
342          _inputAmount,
343          _usdcAmount,
344          _outputToken
345      );
346      bytes memory defaultWorkerOptions = OptionsBuilder
347          .newOptions()
348          .addExecutorLzReceiveOption(relay_gas_limit, 0);
349
350      fee = _quote(_dstEid, payload, defaultWorkerOptions, false);
351  }
```

Listing 2.17: src/Fiat24CryptoDeposit_Base.sol

Suggestion Revise the code logic accordingly.

2.2.2 Lack of duplication check on the fiatName in the function addFiatToken()

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the contract [Fiat24CardAuthorizationMarqeta](#), the function `addFiatToken()` does not verify whether the `_fiatName` string parameter has already been used in the `XXX24Tokens` mapping. This oversight could lead to accidental overwriting of existing token entries in the mapping. The vulnerability affects the `paidCurrency = XXX24Tokens[transactionCurrency_]` logic, potentially returning incorrect token addresses when looking up currencies by name. This could lead to financial loss to the protocol when there are misoperations.

```
366  function addFiatToken(address _fiatToken, uint256 _rateUsdToFiat, string calldata _fiatName)
367      external {
368      if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
369          Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
370
371      require(_fiatToken != address(0), "Zero address");
372      require(!validXXX24Tokens[_fiatToken], "Already exists token");
```

```

372     require(_rateUsdToFiat > 0, "Rate must be > 0");
373
374     validXXX24Tokens[_fiatToken] = true;
375     XXX24Tokens[_fiatName] = _fiatToken;
376     exchangeRates[usd24Address][_fiatToken] = _rateUsdToFiat;
377
378     emit FiatTokenAndRateAddedInMarqeta(_fiatToken, _rateUsdToFiat, _fiatName);
379 }

```

Listing 2.18: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could lead to financial loss to the protocol when there are misoperations.

Suggestion Add duplicate checks accordingly.

2.2.3 Add zero address checks

Status Confirmed

Introduced by Version 1

Description In the function `constructor()` of the contract `Fiat24CardAuthorizationMarqeta`, several address variables (e.g., `eur24Address_`, `usd24Address_`, `chf24Address_`, `gbp24Address_`, `cnh24Address_`) are not checked to ensure they are not zero. Similar checks are also recommended to add for arrays `fiatTokenOperatorRoles`, `cashOperatorRoles`, and `fiatTokenPausers` in the contract `Fiat24TokenFactory`.

```

78     function initialize(
79         address admin,
80         address fiat24AccountAddress_,
81         address eur24Address_,
82         address usd24Address_,
83         address chf24Address_,
84         address gbp24Address_,
85         address cnh24Address_
86     ) public initializer {
87         __AccessControl_init_unchained();
88         __Pausable_init_unchained();
89         _setupRole(DEFAULT_ADMIN_ROLE, admin);
90         _setupRole(OPERATOR_ADMIN_ROLE, admin);
91         fiat24AccountAddress = fiat24AccountAddress_;
92         eur24Address = eur24Address_;
93         usd24Address = usd24Address_;
94         chf24Address = chf24Address_;
95         gbp24Address = gbp24Address_;
96         cnh24Address = cnh24Address_;

```

Listing 2.19: src/Fiat24CardAuthorizationMarqeta.sol

```

24     address[] public fiatTokenOperatorRoles;
25
26     address[] public cashOperatorRoles;
27
28     address[] public fiatTokenPausers;

```

Listing 2.20: src/FiatTokenFactory.sol

Suggestion Add zero address checks accordingly.

2.2.4 Lack of duplication check in the function `addTokenAddress()`

Status Confirmed

Introduced by Version 1

Description In the contract `FiatTokenFactory`, the function `addTokenAddress()` does not verify whether the `tokenAddress` already exists in the `allTokens` array before pushing the new address. This could lead to duplicate entries in the array.

```
221 function addTokenAddress(address tokenAddress) external onlyRole(DEFAULT_ADMIN_ROLE) {
222     allTokens.push(tokenAddress);
223     emit ConfigUpdated("addTokenAddress(address)", "", abi.encode(tokenAddress));
224 }
```

Listing 2.21: src/FiatTokenFactory.sol

Suggestion Add duplicate checks accordingly.

2.2.5 Confusing naming for the variable `_amountOutMinimum`

Status Confirmed

Introduced by Version 1

Description In the contract `Fiat24CryptoDeposit`, the functions `depositTokenViaUsdc()` and `permitAndDepositTokenViaUsdc()` allow users to deposit USDC in exchange for other `Fiat24` tokens. Both functions include a parameter named `_amountOutMinimum`, which is used to validate whether the input `_amount` (representing the USDC amount being deposited) meets a minimum requirement (`_amount < _amountOutMinimum`).

However, the current naming of `_amountOutMinimum` is misleading because it looks like this parameter represents the minimum expected output amount of `Fiat24` tokens, when in reality, it serves as the minimum required input amount of USDC. This inconsistency in naming could cause confusion for developers and users interacting with these functions. The contracts `Fiat24CryptoDeposit_Base`, `Fiat24CryptoDeposit2` have the same problem.

```
107 function depositTokenViaUsdc(address _inputToken, address _outputToken, uint256 _amount,
108     uint256 _amountOutMinimum) nonReentrant external returns (uint256) {
109     if (paused()) revert Fiat24CryptoDeposit__Paused();
110     if (_amount < _amountOutMinimum || _amount == 0) revert
111         Fiat24CryptoDeposit__AmountLessThanMinimum(_amount);
112     if (_inputToken != usdc) revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
113     if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
114         _outputToken);
115     uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(_msgSender());
116     if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(_msgSender());
```

```
115     TransferHelper.safeTransferFrom(_inputToken, _msgSender(), address(this), _amount);
116     return _processDeposit(_msgSender(), _inputToken, _outputToken, _amount, _amount, tokenId);
117 }
```

Listing 2.22: src/Fiat24CryptoDeposit.sol

```
119 function permitAndDepositTokenViaUsdc(
120     address userAddress,
121     address _inputToken,
122     address _outputToken,
123     uint256 _amount,
124     uint256 _amountOutMinimum,
125     uint256 _feeAmountViaUsdc,
126     uint256 _deadline,
127     uint8 _v,
128     bytes32 _r,
129     bytes32 _s
130 ) external nonReentrant payable returns (uint256) {
131     if (paused()) revert Fiat24CryptoDeposit__Paused();
132     if (!hasRole(CASH_OPERATOR_ROLE, _msgSender())) revert Fiat24Token__NotCashOperator(
133         _msgSender());
134     if (_inputToken != usdc) revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
135     if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
136         _outputToken);
137     if (_amount < _amountOutMinimum || _amount == 0) revert
138         Fiat24CryptoDeposit__AmountLessThanMinimum(_amount);
139
140     try IERC20PermitUpgradeable(_inputToken).permit(
141         userAddress,
142         address(this),
143         _amount,
144         _deadline,
145         _v, _r, _s
146     ) {
147     } catch {
148         emit PermitFailed(userAddress, _inputToken, _amount);
149     }
150
151     uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(userAddress);
152     if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(userAddress);
153
154     TransferHelper.safeTransferFrom(_inputToken, userAddress, address(this), _amount);
155
156     if (_feeAmountViaUsdc >= MAX_FEE_AMOUNT_USDC) {
157         _feeAmountViaUsdc = MAX_FEE_AMOUNT_USDC;
158     }
159
160     if (_feeAmountViaUsdc >= _amount) {
161         revert Fiat24CryptoDeposit__FeeAmountExceedsOutput(_feeAmountViaUsdc, _amount);
162     }
163
164     uint256 usdcFactAmount = _amount - _feeAmountViaUsdc;
165     TransferHelper.safeTransfer(usdc, feeReceiver, _feeAmountViaUsdc);
```

```

163
164     return _processDeposit(userAddress, _inputToken, _outputToken, _amount, usdcFactAmount,
        tokenId);
165 }

```

Listing 2.23: src/Fiat24CryptoDeposit.sol

Suggestion It is recommended to rename `_amountOutMinimum` to `_amountUsdcMinimum` or a similar name that accurately reflects its purpose.

2.2.6 Lack of non zero value check in the function `updateExchangeRates()`

Status Confirmed

Introduced by Version 1

Description In the contract `Fiat24CardAuthorizationMarqeta`, the function `updateExchangeRates()` does not validate whether the input rate values (i.e., `_usd_eur`, `_usd_chf`, `_usd_gbp`, `_usd_cnh`) are greater than zero before updating the exchange rates. The function `updateExchangeRates()` in the contract `Fiat24CryptoRelay` has the same problem.

```

294     function updateExchangeRates(uint256 _usd_eur, uint256 _usd_chf, uint256 _usd_gbp, uint256
        _usd_cnh, bool _isMarketClosed) external {
295         if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
296             exchangeRates[usd24Address][eur24Address] = _usd_eur;
297             exchangeRates[usd24Address][chf24Address] = _usd_chf;
298             exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
299             exchangeRates[usd24Address][cnh24Address] = _usd_cnh;
300             marketClosed = _isMarketClosed;
301             emit ExchangeRatesUpdatedByOperator(
302                 _msgSender(),
303                 exchangeRates[usd24Address][eur24Address],
304                 exchangeRates[usd24Address][chf24Address],
305                 exchangeRates[usd24Address][gbp24Address],
306                 exchangeRates[usd24Address][cnh24Address],
307                 marketClosed
308             );
309         } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender()))) {
310             uint256 rateDiff_usd_eur = (exchangeRates[usd24Address][eur24Address] > _usd_eur)
311                 ? (exchangeRates[usd24Address][eur24Address] - _usd_eur)
312                 : (_usd_eur - exchangeRates[usd24Address][eur24Address]);
313             rateDiff_usd_eur = (rateDiff_usd_eur * 10000) / exchangeRates[usd24Address][
                eur24Address];
314             uint256 rateDiff_usd_chf = (exchangeRates[usd24Address][chf24Address] > _usd_chf)
315                 ? (exchangeRates[usd24Address][chf24Address] - _usd_chf)
316                 : (_usd_chf - exchangeRates[usd24Address][chf24Address]);
317             rateDiff_usd_chf = (rateDiff_usd_chf * 10000) / exchangeRates[usd24Address][
                chf24Address];
318             uint256 rateDiff_usd_gbp = (exchangeRates[usd24Address][gbp24Address] > _usd_gbp)
319                 ? (exchangeRates[usd24Address][gbp24Address] - _usd_gbp)
320                 : (_usd_gbp - exchangeRates[usd24Address][gbp24Address]);
321             rateDiff_usd_gbp = (rateDiff_usd_gbp * 10000) / exchangeRates[usd24Address][
                gbp24Address];

```



```

322     uint256 rateDiff_usd_cnh = (exchangeRates[usd24Address][cnh24Address] > _usd_cnh)
323         ? (exchangeRates[usd24Address][cnh24Address] - _usd_cnh)
324         : (_usd_cnh - exchangeRates[usd24Address][cnh24Address]);
325     rateDiff_usd_cnh = (rateDiff_usd_cnh * 10000) / exchangeRates[usd24Address][
        cnh24Address];
326     if (rateDiff_usd_eur < 300) exchangeRates[usd24Address][eur24Address] = _usd_eur;
327     if (rateDiff_usd_chf < 300) exchangeRates[usd24Address][chf24Address] = _usd_chf;
328     if (rateDiff_usd_gbp < 300) exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
329     if (rateDiff_usd_cnh < 300) exchangeRates[usd24Address][cnh24Address] = _usd_cnh;
330     marketClosed = _isMarketClosed;
331     emit ExchangeRatesUpdatedByRobot(
332         _msgSender(),
333         exchangeRates[usd24Address][eur24Address],
334         exchangeRates[usd24Address][chf24Address],
335         exchangeRates[usd24Address][gbp24Address],
336         exchangeRates[usd24Address][cnh24Address],
337         marketClosed
338     );
339 } else {
340     revert Fiat24CardAuthorizationMarqeta__NotRateUpdater((_msgSender()));
341 }
342 }

```

Listing 2.24: src/Fiat24CardAuthorizationMarqeta.sol

```

308 function updateExchangeRates(uint256 _usd_eur, uint256 _usd_chf, uint256 _usd_gbp, uint256
    _usd_cnh, bool _isMarketClosed) external {
309     if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
310         exchangeRates[usd24][eur24] = _usd_eur;
311         exchangeRates[usd24][chf24] = _usd_chf;
312         exchangeRates[usd24][gbp24] = _usd_gbp;
313         exchangeRates[usd24][cnh24] = _usd_cnh;
314         marketClosed = _isMarketClosed;
315         emit ExchangeRatesUpdatedByOperator(
316             _msgSender(), exchangeRates[usd24][eur24], exchangeRates[usd24][chf24],
                exchangeRates[usd24][gbp24], exchangeRates[usd24][cnh24], marketClosed
317         );
318     } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender()))) {
319         uint256 rateDiff_usd_eur =
320             (exchangeRates[usd24][eur24] > _usd_eur) ? (exchangeRates[usd24][eur24] - _usd_eur)
                : (_usd_eur - exchangeRates[usd24][eur24]);
321         rateDiff_usd_eur = (rateDiff_usd_eur * XXX24_DIVISOR) / exchangeRates[usd24][eur24];
322         uint256 rateDiff_usd_chf =
323             (exchangeRates[usd24][chf24] > _usd_chf) ? (exchangeRates[usd24][chf24] - _usd_chf)
                : (_usd_chf - exchangeRates[usd24][chf24]);
324         rateDiff_usd_chf = (rateDiff_usd_chf * XXX24_DIVISOR) / exchangeRates[usd24][chf24];
325         uint256 rateDiff_usd_gbp =
326             (exchangeRates[usd24][gbp24] > _usd_gbp) ? (exchangeRates[usd24][gbp24] - _usd_gbp)
                : (_usd_gbp - exchangeRates[usd24][gbp24]);
327         rateDiff_usd_gbp = (rateDiff_usd_gbp * XXX24_DIVISOR) / exchangeRates[usd24][gbp24];
328         uint256 rateDiff_usd_cnh =
329             (exchangeRates[usd24][cnh24] > _usd_cnh) ? (exchangeRates[usd24][cnh24] - _usd_cnh)
                : (_usd_cnh - exchangeRates[usd24][cnh24]);

```

```
330     rateDiff_usd_cnh = (rateDiff_usd_cnh * XXX24_DIVISOR) / exchangeRates[usd24][cnh24];
331     if (rateDiff_usd_eur < 300) exchangeRates[usd24][eur24] = _usd_eur;
332     if (rateDiff_usd_chf < 300) exchangeRates[usd24][chf24] = _usd_chf;
333     if (rateDiff_usd_gbp < 300) exchangeRates[usd24][gbp24] = _usd_gbp;
334     if (rateDiff_usd_cnh < 300) exchangeRates[usd24][cnh24] = _usd_cnh;
335     marketClosed = _isMarketClosed;
336     emit ExchangeRatesUpdatedByRobot(
337         _msgSender(), exchangeRates[usd24][eur24], exchangeRates[usd24][chf24],
338         exchangeRates[usd24][gbp24], exchangeRates[usd24][cnh24], marketClosed
339     );
340     } else {
341         revert Fiat24CryptoDeposit__NotRateUpdater((_msgSender()));
342     }
```

Listing 2.25: src/Fiat24CryptoRelay.sol

Impact This could lead to operational errors that update exchange rates to zero, which in subsequent currency conversions would cause financial loss to the protocol.

Suggestion Add non zero value checks accordingly.

2.3 Note

2.3.1 Atomicity in Fiat24 card authorization process

Introduced by [Version 1](#)

Description Our current assumption for the contract [Fiat24CardAuthorizationMarqeta](#) is as follows:

- 1.A user makes an offline payment using a physical card.
- 2.The card issuer forwards the transaction details to Fiat24's backend system.
- 3.An address with [AUTHORIZER_ROLE](#) subsequently deducts the corresponding amount from the user's Fiat24 account.

However, if the process is not atomic, the following risks may arise:

- 1.Double-spending attack

A user could swipe the card multiple times before the [AUTHORIZER_ROLE](#) executes the deduction. If the total spent exceeds their [Fiat24](#) account balance, they could obtain goods without sufficient funds, profiting at the protocol's expense.

- 2.Exchange rate & interchange fee risks

The final [paidAmount](#) depends on dynamic factors like, fluctuating exchange rates at settlement time, and interchange fees based on the [paidCurrency](#). If the calculated [paidAmount](#) exceeds the user's balance due to these variables, the deduction could fail after the goods are already taken, leaving the protocol with unrecoverable losses.

Thus, the project should ensure the atomicity in [Fiat24](#) card authorization process.

Feedback from the project The project states that a pre-confirmation mechanism is implemented to ensure transactions are executed atomically to prevent the risks above.

2.3.2 Lack of fiat tokens removal mechanism

Introduced by [Version 1](#)

Description In the contracts [Fiat24CardAuthorizationMarqeta](#), [Fiat24CryptoDeposit_Base](#), [Fiat24CryptoDeposit](#), [Fiat24CryptoDeposit2](#), and [Fiat24CryptoRelay](#), the function `addFiatToken()` is used to add new fiat tokens. They all lack a corresponding removal mechanism for cases when a particular fiat token needs to be discontinued or removed from support.

```

366  function addFiatToken(address _fiatToken, uint256 _rateUsdToFiat, string calldata _fiatName)
      external {
367
368      if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
          Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
369
370      require(_fiatToken != address(0), "Zero address");
371      require(!validXXX24Tokens[_fiatToken], "Already exists token");
372      require(_rateUsdToFiat > 0, "Rate must be > 0");
373
374      validXXX24Tokens[_fiatToken] = true;
375      XXX24Tokens[_fiatName] = _fiatToken;
376      exchangeRates[usd24Address][_fiatToken] = _rateUsdToFiat;
377
378      emit FiatTokenAndRateAddedInMarqeta(_fiatToken, _rateUsdToFiat, _fiatName);
379  }

```

Listing 2.26: `src/Fiat24CardAuthorizationMarqeta.sol`

Feedback from the project The project will add the removal mechanism in the future.

2.3.3 The parameter `_amountOutMinimum` should be validated in the backend

Introduced by [Version 1](#)

Description In the contracts [Fiat24CryptoDeposit2](#) and [Fiat24CryptoDeposit_Base](#), the function `permitAndDepositTokenViaUsdc()` is invoked by the `CASH_OPERATOR_ROLE` to control the `_amountOutMinimum` slippage parameter. This requires backend systems to strictly validate the parameter `_amountOutMinimum`. Otherwise, it may cause loss to users.

```

244  function permitAndDepositTokenViaUsdc(
245      address userAddress,
246      address _inputToken,
247      address _outputToken,
248      uint256 _amount,
249      uint256 _amountOutMinimum,
250      uint256 _feeAmountViaUsdc,
251      uint256 _deadline,
252      uint8 _v,
253      bytes32 _r,
254      bytes32 _s
255  ) external nonReentrant payable returns (uint256) {
256      if (paused()) revert Fiat24CryptoDeposit__Paused();
257      if (!hasRole(CASH_OPERATOR_ROLE, _msgSender())) revert Fiat24Token__NotCashOperator(
          _msgSender());

```

```

258     if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
259     if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
        _outputToken);
260
261     try IERC20PermitUpgradeable(_inputToken).permit(
262         userAddress,
263         address(this),
264         _amount,
265         _deadline,
266         _v, _r, _s
267     ) {
268     } catch {
269         emit PermitFailed(userAddress, _inputToken, _amount);
270     }
271
272     TransferHelper.safeTransferFrom(_inputToken, userAddress, address(this), _amount);
273     TransferHelper.safeApprove(_inputToken, UNISWAP_ROUTER, _amount);
274
275     uint256 usdcAmount;
276     if (_inputToken != usdc) {
277         uint24 poolFee = getPoolFeeOfMostLiquidPool(_inputToken, usdc);
278         if (poolFee == 0) revert Fiat24CryptoDeposit__NoPoolAvailable(_inputToken, usdc);
279
280         ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
281             tokenIn: _inputToken,
282             tokenOut: usdc,
283             fee: poolFee,
284             recipient: address(this),
285             deadline: block.timestamp + 15,
286             amountIn: _amount,
287             amountOutMinimum: _amountOutMinimum,
288             sqrtPriceLimitX96: 0
289         });
290         usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);

```

Listing 2.27: src/Fiat24CryptoDeposit2.sol

2.3.4 Upgrade the implementation of Fiat24Token properly

Introduced by Version 1

Description In the contract `Fiat24TokenFactory`, the function `AuthAndCreateFiatToken()` relies on `__Fiat24Token_init_()` initialization logic automatically granting the factory contract `OPERATOR_ROLE`, `OPERATOR_ADMIN_ROLE`, and `DEFAULT_ADMIN_ROLE` permissions. This creates a risk when the `Fiat24Token` implementation is upgraded with modified initialization logic that stops granting these permissions, the factory will immediately fail to properly configure the new token's permissions for `cashOperatorRoles` and other addresses, causing permanent DoS in the function `AuthAndCreateFiatToken()`.

```

52     function __Fiat24Token_init_(
53         address admin,
54         address fiat24accountProxyAddress,

```

```
55     string memory name_,
56     string memory symbol_,
57     uint256 limitWalkin,
58     uint256 chfRate,
59     uint256 withdrawCharge
60 ) internal onlyInitializing {
61     __AccessControl_init_unchained();
62     __ERC20_init_unchained(name_, symbol_);
63     __ERC20Permit_init(name_);
64     _setupRole(DEFAULT_ADMIN_ROLE, admin);
65     _setupRole(OPERATOR_ADMIN_ROLE, admin);
66     _setupRole(OPERATOR_ROLE, admin);
67     fiat24account = Fiat24Account(fiat24accountProxyAddress);
68     LimitWalkin = limitWalkin;
69     ChfRate = chfRate;
70     WithdrawCharge = withdrawCharge;
71 }
```

Listing 2.28: src/Fiat24Token.sol

```
109 function AuthAndCreateFiatToken(
110     string calldata name,
111     string calldata symbol,
112     uint256 limitWalkin,
113     uint256 chfRate,
114     uint256 withdrawCharge
115 ) external onlyRole(CREATE_ROLE) returns (address) {
116
117     bytes memory initData = abi.encodeWithSignature(
118         "initialize(address,address,string,string,uint256,uint256,uint256)",
119         address(this),
120         accountProxyAddress,
121         name, symbol, limitWalkin, chfRate, withdrawCharge
122     );
123
124     BeaconProxy proxy = new BeaconProxy(beaconAddress, initData);
125     address proxyAddr = address(proxy);
126     allTokens.push(proxyAddr);
127
128     FiatToken token = FiatToken(proxyAddr);
129
130     token.grantRole(token.DEFAULT_ADMIN_ROLE(), fiatTokenAdminAddress);
131     token.grantRole(token.OPERATOR_ADMIN_ROLE(), fiatTokenOperatorAdminRole);
132
133     for (uint256 i = 0; i < fiatTokenOperatorRoles.length; i++) {
134         token.grantRole(token.OPERATOR_ROLE(), fiatTokenOperatorRoles[i]);
135     }
136
137     for (uint256 i = 0; i < cashOperatorRoles.length; i++) {
138         token.grantRole(token.CASH_OPERATOR_ROLE(), cashOperatorRoles[i]);
139     }
140
141     for (uint256 i = 0; i < fiatTokenPausers.length; i++) {
```

```
142         token.grantRole(token.PAUSE_ROLE(), fiatTokenPausers[i]);
143     }
144
145     if (fiatTokenUnpauser != address(0)) {
146         token.grantRole(token.UNPAUSE_ROLE(), fiatTokenUnpauser);
147     }
148
149     token.revokeRole(token.OPERATOR_ROLE(), address(this));
150     token.revokeRole(token.OPERATOR_ADMIN_ROLE(), address(this));
151     token.renounceRole(token.DEFAULT_ADMIN_ROLE(), address(this));
152
153     emit FiatTokenCreated(proxyAddr, fiatTokenAdminAddress);
154     return proxyAddr;
155 }
```

Listing 2.29: src/FiatTokenFactory.sol

2.3.5 Ensure that the `exchangeRates` and `validXXX24Tokens` are set properly

Introduced by [Version 1](#)

Description In the protocol, the `exchangeRates` and `validXXX24Tokens` strongly impact the swapping of different `Fiat24` tokens. The project team should ensure that they are properly set to guarantee the security of the entire project.

2.3.6 Initialize the implementation contracts immediately after deployments

Introduced by [Version 1](#)

Description The contracts `Fiat24CardAuthorizationMarqeta`, `Fiat24CryptoDeposit`, `Fiat24CryptoDeposit2`, `Fiat24CryptoDeposit_Base` and `Fiat24CardAuthorizationMargeta` do not invoke the function `_disableInitializers()` in the constructor. Thus, the protocol should initialize these implementation contracts immediately after deployments, to avoid evil initialization front-running risks.

2.3.7 Potential centralization risks

Introduced by [Version 1](#)

Description In this project, several privileged roles (e.g., `OPERATOR_ROLE`, `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, Fiat token adding operation is controlled by `OPERATOR_ADMIN_ROLE`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

