BLOCKSEC

Security Audit
Report for Fiat24 Contracts

Date: July 16, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About the Audit Target . . . 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 . . . 5
2.1.1 Lack of fee distribution to the wallet provider . . . 5
2.1.2 Redundant swaps for the fee conversion . . . 7
2.1.3 Improper minimum fee guard in `calculateTokenFeeAmount()` . . . 7
2.1.4 Improper removal of the function `__Fiat24Token_init_()` . . . 8
2.1.5 Potential DoS due to inconsistent allowance consumption . . . 8
2.1.6 Incorrect fund source in the function `retryFailedBatchTransfer()` . . . 10
2.1.7 Incorrect quoting in the function `_executeOnrampAndBridge()` . . . 11
2.2 Recommendation . . . 12
2.2.1 Remove redundant and deprecated code . . . 12
2.2.2 Add duplicate checks in the function `setFeeManager()` . . . 13
2.2.3 Add non-reentrant guard to the function `permitAndMoneyExchangeExactIn()` 13
2.2.4 Verify wrapped LZD amount in the function `_wrapPathUsdToLzd()` . . . 14
2.3 Note . . . 15
2.3.1 Potential centralization risks . . . 15

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24 Contracts

Version History

Version	Date	Description
1.0	July 16, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-20 07:53:00+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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Fiat24 Contracts of Mantle.

The Fiat24 Contracts of Mantle is a fiat-crypto payment infrastructure on Mantle that enables card authorization processing and operator-controlled vault management for fiat token operations. This update introduces a unified fee configuration module via the contract Fiat24FeeManager, cross-chain deposit paths on the Arbitrum and Tempo chains via LayerZero, a delegation vault extension for partner-managed payouts in the contract DelegationVault, and batch double-booking transfers in the contract Fiat24CardAuthorizationMarqeta.

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/BufferPool.sol
- src/DelegationVault.sol
- src/Fiat24CardAuthorizationMarqeta.sol
- src/Fiat24CryptoDeposit_Arb.sol
- src/Fiat24CryptoDeposit_Tempo.sol
- src/Fiat24CryptoDeposit.sol
- src/Fiat24CryptoRelay.sol
- src/Fiat24FeeManager.sol
- src/Fiat24Token.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target 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.

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

Project	Version	Commit Hash
fiat24contracts	Version 0	ad908801e2dc30fb805551dc117268312f6fdbc7
	Version 1	b80f6c1521bb96974a5a4af4ad2caf316d27e2df
	Version 2	66f0ff159016f7df0b1979e7d9209457f2d793f5

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 Target, 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 the Target.

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 the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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**,

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

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

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.

Chapter 2 Findings

In total, we found **seven** potential security issues. Besides, we have **four** recommendations and **one** note.

- Low Risk: 7
- Recommendation: 4
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Lack of fee distribution to the wallet provider	Security Issue	Confirmed
2	Low	Redundant swaps for the fee conversion	Security Issue	Fixed
3	Low	Improper minimum fee guard in `calculateTokenFeeAmount()`	Security Issue	Fixed
4	Low	Improper removal of the function `__Fiat24Token_init_()`	Security Issue	Fixed
5	Low	Potential DoS due to inconsistent allowance consumption	Security Issue	Fixed
6	Low	Incorrect fund source in the function `retryFailedBatchTransfer()`	Security Issue	Fixed
7	Low	Incorrect quoting in the function `_executeOnrampAndBridge()`	Security Issue	Fixed
8	-	Remove redundant and deprecated code	Recommendation	Partially Fixed
9	-	Add duplicate checks in the function `setFeeManager()`	Recommendation	Confirmed
10	-	Add non-reentrant guard to the function `permitAndMoneyExchangeExactIn()`	Recommendation	Fixed
11	-	Verify wrapped LZD amount in the function `_wrapPathUsdToLzd()`	Recommendation	Fixed
12	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of fee distribution to the wallet provider

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit`, the function `_collectDepositFee()` collects fees and sends them to the account `FEE_DESK`. However, the function `_collectDepositFee()` lacks distributing fees to the wallet provider. As a result, the wallet provider will lose the fee.

```
525   function _processDeposit(
526       address user,
527       address _inputToken,
528       address _outputToken,
529       uint256 _inputAmount,
530       uint256 _factAmount,
531       uint256 tokenId
532   ) internal returns (uint256 outputAmount) {
533       if (_factAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
534       if (!_isUnlimitedDepositInput(_inputToken) && _factAmount > maxUsdcDepositAmount) {
535           revert Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(_factAmount,
                  maxUsdcDepositAmount);
536       }
537       if (_factAmount < minUsdcDepositAmount) revert
              Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(_factAmount, minUsdcDepositAmount
              );
538
539       uint256 feeInUSDC = _collectDepositFee(tokenId, _factAmount);
540       TransferHelper.safeTransfer(usdc, usdcDepositAddress, _factAmount);
541
542       IFiat24CryptoRelay relay = IFiat24CryptoRelay(fiat24CryptoRelayAddress);
543       uint256 grossOutputAmount = _calculateDepositGrossOutput(_factAmount - feeInUSDC,
              _outputToken, relay);
544       outputAmount = _applySpread(grossOutputAmount, relay.getSpreadForTokenId(tokenId, usd24,
              _outputToken, false));
545
546       TransferHelper.safeTransferFrom(_outputToken, IFiat24Account(fiat24account).ownerOf(
              CRYPTO_DESK), user, outputAmount);
547       emit DepositedFiat24Token(user, _inputToken, _inputAmount, _outputToken, outputAmount);
548   }
```

Listing 2.1: src/Fiat24CryptoDeposit.sol

```
550   function _collectDepositFee(uint256 tokenId, uint256 usdcAmount) internal returns (uint256
          feeInUSDC) {
551       feeInUSDC = IFiat24CryptoRelay(fiat24CryptoRelayAddress).getOfframpFee(tokenId, usdcAmount)
              ;
552       if (feeInUSDC == 0) {
553           return feeInUSDC;
554       }
555
556       address feeAccount = IFiat24Account(fiat24account).ownerOf(FEE_DESK);
557       uint256 feeAmountUsd24 = feeInUSDC / USDC_DIVISOR;
558       TransferHelper.safeTransferFrom(
559           usd24,
560           IFiat24Account(fiat24account).ownerOf(CRYPTO_DESK),
561           feeAccount,
562           feeAmountUsd24
563       );
564       emit DepositFeeCollected(tokenId, feeAccount, usdcAmount, feeInUSDC, feeAmountUsd24);
565   }
```

Listing 2.2: src/Fiat24CryptoDeposit.sol

Impact The wallet provider will lose the fee.

Suggestion Revise the logic accordingly.

Feedback from the project The project stated that the fee for wallet providers will be distributed off-chain.

2.1.2 Redundant swaps for the fee conversion

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDepositTempo`, the function `_swapUsdcToPathUsdFee()` swaps USDC.e into pathUSD via the stablecoin exchange and wraps pathUSD into LZD as the bridging fee. However, USDC.e is a whitelisted backing token in the contract `LZEndpointDollar`, which can directly wrap USDC.e into LZD. As a result, the redundant swaps may lead users to suffer potential losses (e.g., fees and slippages).

```
463       uint256 pathUsdFeeAmount = _swapUsdcToPathUsdFee(_toUint128(feeUsdcAmount));
```

Listing 2.3: src/Fiat24CryptoDeposit_Tempo.sol

```
488       pathUsdAmount = stablecoinExchange.swapExactAmountIn(address(usdc), address(pathUsd),
              feeUsdcAmount, 0);
```

Listing 2.4: src/Fiat24CryptoDeposit_Tempo.sol

```
539       ILzdWrapper(feeToken).wrap(address(pathUsd), address(this), pathUsdAmount);
```

Listing 2.5: src/Fiat24CryptoDeposit_Tempo.sol

Impact The redundant swaps may lead users to suffer potential losses (e.g., fees and slippages).

Suggestion Remove the redundant swaps.

2.1.3 Improper minimum fee guard in `calculateTokenFeeAmount()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24FeeManager`, the function `calculateTokenFeeAmount()` enforces a minimal fee (i.e., `MINIMALCOMMISIONFEE`) via the check `feeAmount > 0 && feeAmount < MINIMALCOMMISIONFEE`. However, the fee computation `feeAmount = (amount * fee) / 10000` may be truncated and resulting in the fee to zero. As a result, users may circumvent the minimal fee floor.

```
313       if (feeAmount > 0 && feeAmount < MINIMALCOMMISIONFEE) {
```

Listing 2.6: src/Fiat24FeeManager.sol

Impact Users may circumvent the minimal fee floor.

Suggestion Change the guard condition from `feeAmount > 0 && feeAmount < MINIMALCOMMISIONFEE` to `feeAmount < MINIMALCOMMISIONFEE`.

2.1.4 Improper removal of the function `__Fiat24Token_init_()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24Token`, the function `__Fiat24Token_init_()` is commented out. However, all fiat24 tokens inherit the contract `Fiat24Token` and invoke the function `__Fiat24Token_init_()` in their initializers. As a result, new fiat24 tokens can not be deployed in the future.

```
62  // function __Fiat24Token_init_(
63  //     address admin,
64  //     address fiat24accountProxyAddress,
65  //     string memory name_,
66  //     string memory symbol_,
67  //     uint256 limitWalkin,
68  //     uint256 chfRate,
69  //     uint256 withdrawCharge
70  // ) internal onlyInitializing {
71  //     __AccessControl_init_unchained();
72  //     __ERC20_init_unchained(name_, symbol_);
73  //     __ERC20Permit_init(name_);
74  //     _setupRole(DEFAULT_ADMIN_ROLE, admin);
75  //     _setupRole(OPERATOR_ADMIN_ROLE, admin);
76  //     _setupRole(OPERATOR_ROLE, admin);
77  //     fiat24account = Fiat24Account(fiat24accountProxyAddress);
78  //     LimitWalkin = limitWalkin;
79  //     ChfRate = chfRate;
80  //     WithdrawCharge = withdrawCharge;
81  // }
```

Listing 2.7: src/Fiat24Token.sol

Impact New fiat24 tokens can not be deployed in the future.

Suggestion Restore the function `__Fiat24Token_init_()`.

2.1.5 Potential DoS due to inconsistent allowance consumption

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24Token`, the function `cashDepositOK()` allows the `CASH_OPERATOR` to transfer fiat tokens (via the function `transferFrom()`) to the recipient and charge fees (via the function `_transferChargeFrom()`). However, the allowance consumption of the functions

is inconsistent. Specifically, the function `cashDepositOK()` requires the owner of the account `9101` to approve fiat24 tokens to both the role `CASH_OPERATOR` and the contract `Fiat24Token`. As a result, this design may lead to potential DoS if the owner of the account `9101` fails to approve tokens to the contract `Fiat24Token`.

```
144   function cashDepositOK(uint256 recipientAccountId, uint256 amount, string memory exaccId,
          string memory bankId, string memory trxId) external onlyRole(CASH_OPERATOR_ROLE) {
145       bytes32 key = keccak256(abi.encodePacked(bankId, "-", trxId));
146       require(pacs008[bytes32ToString(key)] == 0, "Fiat24Token: pacs008 already processed");
147       pacs008[bytes32ToString(key)] = block.number;
148       uint256 feeAmount = Fiat24FeeManager(feeManager).getPayinFee(recipientAccountId, address(
              this), amount);
149       uint256 netAmount = _applyFeeAmount(amount, feeAmount);
150       transferFrom(fiat24account.ownerOf(9101), fiat24account.ownerOf(recipientAccountId),
              netAmount);
151       _transferChargeFrom(fiat24account.ownerOf(9101), feeAmount);
152       emit CashDeposit(recipientAccountId, fiat24account.ownerOf(recipientAccountId),
              recipientAccountId, netAmount, exaccId, bankId, trxId);
153       emit CashDepositFeeCharged(recipientAccountId, amount, feeAmount, netAmount);
154   }
```

Listing 2.8: src/Fiat24Token.sol

```
409   function _transferChargeFrom(address from, uint256 feeAmount) internal {
410       if (feeAmount > 0) {
411           require(IERC20Upgradeable(address(this)).transferFrom(from, fiat24account.ownerOf(9203)
                  , feeAmount));
412       }
413   }
```

Listing 2.9: src/Fiat24Token.sol

```
 96   function transferFrom(address sender, address recipient, uint256 amount) public virtual
          override returns (bool) {
 97       _transfer(sender, recipient, amount);
 98
 99       uint256 currentAllowance = allowance(sender, _msgSender());
100       require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
101       unchecked {
102           _approve(sender, _msgSender(), currentAllowance - amount);
103       }
104
105       return true;
106   }
```

Listing 2.10: src/Fiat24Token.sol

Impact This design may lead to potential DoS if the owner of the account `9101` fails to approve tokens to the contract `Fiat24Token`.

Suggestion Revise the logic accordingly.

2.1.6 Incorrect fund source in the function `retryFailedBatchTransfer()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CardAuthorizationMarqeta`, the function `retryFailedBatchTransfer()` transfers tokens from the recorded sender (i.e., `failed.sender` equals `REVERSE_ROLE`). If the role `REVERSE_ROLE` is reset, the function `retryFailedBatchTransfer()` will transfer tokens from the previous `REVERSE_ROLE` instead of the current `REVERSE_ROLE`. As a result, this may lead to a potential fund loss to the previous `REVERSE_ROLE`.

```
435   function retryFailedBatchTransfer(uint256 failedIndex) external {
436       if (!(hasRole(REVERSE_ROLE, _msgSender()))) revert
              Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
437       if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
438       require(failedIndex < failedBatchTransfers.length, "Invalid failed index");
439
440       FailedBatchTransfer memory failed = failedBatchTransfers[failedIndex];
441       try this.executeBatchDoubleBookingTransferItem(failed.sender, failed.userTokenId, failed.
              token, failed.amount) {
442           _removeFailedBatchTransfer(failedIndex);
443           emit BatchTransferRetried(failedIndex, true, "");
444       } catch (bytes memory reason) {
445           failedBatchTransfers[failedIndex].reason = reason;
446           emit BatchTransferRetried(failedIndex, false, reason);
447       }
448   }
```

Listing 2.11: src/Fiat24CardAuthorizationMarqeta.sol

```
394   function batchDoubleBookingTransfer(BatchDoubleBookingTransferParams calldata params) public {
395       if (!(hasRole(REVERSE_ROLE, _msgSender()))) revert
              Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
396       if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
397       require(params.userTokenIds.length == params.tokens.length && params.userTokenIds.length ==
               params.amounts.length, "Arrays length mismatch");
398       require(params.userTokenIds.length > 0, "Empty batch");
399
400       address sender = _msgSender();
401
402       for (uint256 i = 0; i < params.userTokenIds.length; i++) {
403           uint256 userTokenId = params.userTokenIds[i];
404           address token = params.tokens[i];
405           uint256 amount = params.amounts[i];
406           try this.executeBatchDoubleBookingTransferItem(sender, userTokenId, token, amount) {}
407           catch (bytes memory reason) {
408               _recordFailedBatchTransfer(sender, userTokenId, token, amount, reason);
409           }
410       }
411   }
```

Listing 2.12: src/Fiat24CardAuthorizationMarqeta.sol

```
758   function _recordFailedBatchTransfer(address sender, uint256 userTokenId, address token,
          uint256 amount, bytes memory reason) internal {
759       address user;
760       try IFiat24Account(fiat24AccountAddress).ownerOf(userTokenId) returns (address owner) {
761           user = owner;
762       } catch {}
763
764       failedBatchTransfers.push(FailedBatchTransfer({
765           sender: sender,
766           userTokenId: userTokenId,
767           user: user,
768           token: token,
769           amount: amount,
770           reason: reason
771       }));
772
773       emit BatchTransferFailed(failedBatchTransfers.length - 1, sender, token, userTokenId, user,
              amount, reason);
774   }
```

Listing 2.13: src/Fiat24CardAuthorizationMarqeta.sol

Impact This may lead to a potential fund loss to the previous `REVERSE_ROLE`.

Suggestion Revise the logic accordingly.

2.1.7 Incorrect quoting in the function `_executeOnrampAndBridge()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24PartnerGateway`, the function `_executeOnrampAndBridge()` invokes the function `getQuote()` to calculate the value `usdcOut` for event emission. However, the function `getQuote()` calculates the value `usdcOut` using zero `tokenId` rather than the contract `Fiat24PartnerGateway`'s ID. As a result, an incorrect spread rate might be used in the calculation of the value `usdcOut`, leading to an incorrect event emission.

```
579       // Get quote to return usdcOut for event
580       usdcOut = bufferPool.getQuote(params.tokenIn, params.amountIn);
```

Listing 2.14: src/Fiat24PartnerGateway.sol

```
505   function getQuote(address tokenIn, uint256 amountIn) public view returns (uint256 usdcOut) {
506       (usdcOut, ) = _getQuoteWithFee(tokenIn, amountIn, 0);
507   }
```

Listing 2.15: src/BufferPool.sol

```
517   function getQuoteForTokenId(address tokenIn, uint256 amountIn, uint256 tokenId) public view
          returns (uint256 usdcOut) {
```

Listing 2.16: src/BufferPool.sol

Impact This may lead to an incorrect event emission.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Remove redundant and deprecated code

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description There are several redundant and deprecated functions and declarations.

1) In the contracts `BufferPool`, `Fiat24CryptoRelay`, and `Fiat24CardAuthorizationMarqeta`, the legacy fee and spread setters (i.e., the functions `updateFees()`, `changeExchangeSpread()`, and `setExchangeSpread()`) are redundant.

```
640   function updateFees(uint256 _bufferPoolFee) external onlyRole(OPERATOR_ROLE) {
641       require(_bufferPoolFee <= 1000, "Fee too high"); // Max 10%
642       if (_bufferPoolFee == bufferPoolFee) revert BufferPool__NoChange();
643       uint256 oldFee = bufferPoolFee;
644       bufferPoolFee = _bufferPoolFee;
645       emit FeesUpdated(oldFee, _bufferPoolFee);
646   }
```

Listing 2.17: src/BufferPool.sol

```
423   function changeExchangeSpread(uint256 _exchangeSpread) external onlyRole(OPERATOR_ADMIN_ROLE)
          {
424       uint256 oldSpread = exchangeSpread;
425       require(_exchangeSpread >= 9000 && _exchangeSpread <= 11000, "Spread must be between 9000
              and 11000");
426       require(_exchangeSpread != oldSpread, "No change");
427       exchangeSpread = _exchangeSpread;
428       emit ExchangeSpreadChanged(oldSpread, _exchangeSpread);
429   }
```

Listing 2.18: src/Fiat24CryptoRelay.sol

```
800   function setExchangeSpread(uint256 newExchangeSpread) external {
801       if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
              Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
802
803       require(newExchangeSpread > 9000 && newExchangeSpread <= 11000, "Spread must be between
              9000 and 11000");
804       uint256 old = exchangeSpread;
805       exchangeSpread = newExchangeSpread;
806       emit ExchangeSpreadUpdated(old, newExchangeSpread);
807   }
```

Listing 2.19: src/Fiat24CardAuthorizationMarqeta.sol

2) In the contracts `Fiat24FeeManager`, the function `getExchangeSpreadBps()` with one parameter is redundant.

```
235    function getExchangeSpreadBps(uint256) public view returns (uint256) {
236        return exchangeSpread;
237    }
```

Listing 2.20: src/Fiat24FeeManager.sol

3) In the interface `IFiat24CryptoRelay`, the function declarations `setFundFlowVault()`, `setFundFlowVaultEnabled()`, `setRelayGasLimit()`, and `setFixedNativeFee()` are redundant.

```
46    function setFundFlowVault(address fundFlowVault_) external;
47    function setFundFlowVaultEnabled(bool enabled) external;
48    function setRelayGasLimit(uint128 _newLimit) external;
49    function setFixedNativeFee(uint256 _fixedNativeFee) external;
```

Listing 2.21: src/interfaces/IFiat24CryptoRelay.sol

It is recommended to remove these redundant and deprecated code.

Suggestion Remove redundant and deprecated code.

2.2.2 Add duplicate checks in the function `setFeeManager()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `Fiat24Token`, the function `setFeeManager()` updates the variable `feeManager` without verifying whether the new value differs from the current one. This may result in unnecessary repeated operations and redundant event emissions. It is recommended to add a duplicate check before applying the update.

```
369    function setFeeManager(address feeManager_) public onlyRole(OPERATOR_ADMIN_ROLE) {
370        require(feeManager_ != address(0));
371        feeManager = feeManager_;
372    }
```

Listing 2.22: src/Fiat24Token.sol

Suggestion It is recommended to add a duplicate check before applying the update.

2.2.3 Add non-reentrant guard to the function `permitAndMoneyExchangeExactIn()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description It is recommended to add the non-reentrant guard to the function `permitAndMoneyExchangeExactIn()`.

```
286    function permitAndMoneyExchangeExactIn(
287        address userAddress,
288        address _inputToken,
289        address _outputToken,
290        uint256 _inputAmount,
291        uint256 _amountOutMinimum,
```

```
292         uint256 _deadline,
293         uint8 _v,
294         bytes32 _r,
295         bytes32 _s
296     ) external onlyRole(CASH_OPERATOR_ROLE) whenNotPaused returns (uint256) {
297         if (!validXXX24Tokens[_inputToken]) revert Fiat24CryptoDeposit__NotValidInputToken(
                _inputToken);
298         if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
                _outputToken);
299         if (_inputToken == _outputToken) revert Fiat24CryptoDeposit__InputTokenOutputTokenSame(
                _inputToken, _outputToken);
300
301         uint256 usdAmount = _inputAmount * getExchangeRate(_inputToken, usd24) / XXX24_DIVISOR;
302         if (usdAmount < minUsdExchangeAmount) revert Fiat24CryptoExchange__UsdAmountLowerMinAmount(
                usdAmount);
303
304         // 1) Permit
305         try IERC20Permit(_inputToken).permit(userAddress, address(this), _inputAmount, _deadline,
                _v, _r, _s) {}
306         catch {
307             emit PermitFailed(userAddress, address(this), _inputAmount);
308         }
309
310         return _exchangeExactIn(userAddress, _inputToken, _outputToken, _inputAmount,
                _amountOutMinimum);
311     }
```

Listing 2.23: src/Fiat24CryptoRelay.sol

Suggestion Add non-reentrant guard to the function `permitAndMoneyExchangeExactIn()`.

2.2.4 Verify wrapped LZD amount in the function `_wrapPathUsdToLzd()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit_Tempo`, the function `_wrapPathUsdToLzd()` expected a 1-to-1 wrap ratio from pathUSD to LZD. It is recommended to add a check on `lzdAmount` and `pathUsdAmount` after the wrap.

```
534     function _wrapPathUsdToLzd(uint256 pathUsdAmount) internal returns (uint256 lzdAmount) {
535         if (pathUsdAmount == 0) revert Fiat24CryptoDepositTempo__InvalidAmount();
536         address feeToken = _lzFeeToken();
537         uint256 lzdBefore = IERC20Upgradeable(feeToken).balanceOf(address(this));
538         _approveToken(address(pathUsd), feeToken, pathUsdAmount);
539         ILzdWrapper(feeToken).wrap(address(pathUsd), address(this), pathUsdAmount);
540         IERC20Upgradeable(address(pathUsd)).safeApprove(feeToken, 0);
541         lzdAmount = IERC20Upgradeable(feeToken).balanceOf(address(this)) - lzdBefore;
542         if (lzdAmount == 0) revert Fiat24CryptoDepositTempo__InsufficientSwapOutput(feeToken,
                lzdAmount, pathUsdAmount);
543     }
```

Listing 2.24: src/Fiat24CryptoDeposit_Tempo.sol

```
20/// @notice Minimal interface for the LayerZero alt fee token (EndpointV2Alt.nativeToken(), "LZD")
      ,
21///         a multi-collateral wrapper that mints 1:1 from a whitelisted backing token (e.g.
      pathUSD).
```

Listing 2.25: src/Fiat24CryptoDeposit_Tempo.sol

Suggestion Verify wrapped LZD amount.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations. For example, `OPERATOR_ADMIN_ROLE` can update critical protocol parameters through functions such as `changeStandardFee()`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol. The project should ensure that privileged accounts are managed with multi-signature wallets and that key rotation procedures are in place.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS