INTERFI

SMART CONTRACT SECURITY ASSESSMENT

PREPARED FOR

XFY

hello@interfi.network

interfinetwork

www.interfi.network

FOR XFY

Date of Report	December 27, 2025
Arbitrum Deploy	0x403a94486A5cc8776909a1Dd21D8E23a7ffC0D99
Website	@XFY_Protocol
X	@XFY_Protocol
Language	Solidity
Methodology	Automated Review, Unit Tests, Manual Review
Auditor	InterFi

Disclaimer: Smart contracts deployed on blockchains are inherently exposed to potential exploits, vulnerabilities, and security risks. Blockchain and cryptographic technologies are emerging and carry ongoing uncertainties. Please review the full audit report for detailed insights into risk severity, vulnerabilities, and audit scope limitations.

KYC Advisory: The project lacks verified third-party KYC of its owners, team, or deployers. Without independent KYC, transparency and accountability are reduced, increasing the risk of fraud or rug pulls.

Verification: Verify this report: https://www.github.com/interfinetwork

TABLE OF CONTENTS

1. SUMMARY

The audit resulted in the identification of issues across a range of severity levels, including logic flaws, access control oversights, and design inconsistencies. All high-impact findings were communicated to the development team with clear recommendations for remediation. Where applicable, the team has confirmed implementation of fixes or provided justifications for design choices.

1.1 Summary of Findings

Severity	Count
Critical	0
Major	0
Medium	2
Minor	3
Unknown	2
Centralization	3

1.2 Resolution Status

Status	Count
Fixed	7
Partially Fixed	0
Acknowledged	3
Pending	0

1.3 System Overview

This system is a decentralized protocol comprising a suite of smart contracts. These contracts collectively define the rules, permissions, and operational workflows for managing on-chain assets, executing user interactions, and enforcing protocol-level logic. Smart contracts in this context are self-executing code units that autonomously manage the state and behavior of digital assets based on predefined conditions. The protocol utilizes these contracts to enable key functionalities such as:

- Ownership and access control enforcement
- Permission and role-based actions
- Data storage and updates
- Event logging and auditability
- Batch processing and collection management

1.4 Files in Scope

InterFi was engaged by XFY to perform a security audit of the smart contracts. The audit scope was strictly limited to the **manual review of contracts explicitly listed under the "Files in Scope" section.** No other files, contracts, modules, or components were reviewed, including but not limited to:

- Any other CCIP contracts
- Chainlink CCIP core contracts
- Imported libraries or interfaces
- External dependencies referenced via import statements

All imported contracts (Chainlink CCIP pools, interfaces, and shared libraries) were treated as trusted black boxes and were not audited for correctness, security, availability, or upgrade safety.

As a result, any vulnerabilities, misconfigurations, logic flaws, or breaking changes present in external dependencies, upstream CCIP infrastructure, or future versions of imported contracts are explicitly out of scope and not covered by this audit.

“Files in Scope”

File	Path	Notes
Token	`XFYTokenNoMint.sol`	
Pool	`XFYTokenPool.sol`	
Time lock Controller	`XFYTImelockController.sol`	

1.5 Out-of-Scope Assumptions

The following components and assumptions were explicitly excluded from this audit:

- Frontend or backend integration logic.
- Off-chain components, scripts, or oracles.
- External contracts or libraries unless explicitly stated.
- Compiler-level or EVM-specific behavior outside the contract’s scope.
- Governance or tokenomics-related decisions not implemented in code.
- All third-party dependencies as discussed in findings.

2. METHODOLOGY

2.1 Audit Objectives

This audit aims to ensure that the smart contract system is predictable, and behaves as intended under normal conditions. Primary audit objectives are to:

- Identify potential vulnerabilities or logic errors in the implementation.
- Evaluate adherence to best practices in smart contract development.
- Assess the correctness of access controls and permission systems.
- Recommend remediations or enhancements for improved security and performance.

2.2 Methodologies

The audit follows a layered security approach using both automated tools and manual techniques. We review the contracts for functional correctness, exploitability, and adherence to smart contract best practices:

Type	Tools & Techniques
Manual Code Review	Line-by-line analysis to check logic, permissions, and edge cases
Automated Analysis	Tools like Slither, MythX, or custom linters to catch known patterns
Static Analysis	Identification of bugs without executing the code (compile-time checks)
Unit Test Inspection	Evaluation of existing test coverage, assumptions, and potential false positives/negatives (if applicable)
Architecture Review	Mapping of privileged roles, callable paths, and contract interdependencies (if applicable)

2.3 Risk Categorization

Each issue identified during the audit is assigned a severity level based on its potential impact, exploitability, and likelihood of real-world abuse. These categories help prioritize remediation efforts:

Risk Severity	Definition
Critical	Represents a severe vulnerability that may result in complete contract compromise, such as asset theft, permanent loss of functionality, or unrestricted access. These issues are often easily exploitable and require immediate resolution.
Major	Indicates significant risk that can affect core contract behavior, enable unauthorized operations, or create unintended financial exposure. While not as urgent as critical risks, they should be remediated promptly.
Medium	These are moderate-level risks that may become exploitable under specific conditions. They often relate to logic errors, insufficient validation, or architectural oversights that could escalate over time.
Minor	Denotes issues that have low security impact but may degrade code quality, performance, or maintainability. These include inefficiencies, style violations, or redundant logic. Fixes are recommended for robustness.
Unknown	Risks where the severity cannot be confidently determined due to limited context, external dependencies, or ambiguous design intent. It is advised to treat these conservatively and address them proactively.
Centralization	Any function controlled by a single privileged role is treated as a critical risk, regardless of its purpose, due to the potential for misuse, override, or total asset control.

2.4 Resolution Status Definitions

All identified issues are also assigned a resolution status, indicating the current handling and response from the development team:

Status	Definition
Fixed	The issue has been remediated and verified as resolved during the re-audit or final check.
Partially Fixed	The issue has been partially mitigated, but remnants or related concerns may still exist. Further attention may be required.
Acknowledged	The development team has accepted the finding but opted not to implement a fix.
Pending	The issue remains unresolved at the time of publication. It poses a potential risk and should be addressed.

3. FINDINGS

01	**Inconsistent token unit handling between initial mint and ongoing mint**
Severity	Medium
Description	`Constructor mints initialSupply * 10**decimals() (scaled).` `mint(to, amount)` mints `amount` without scaling. This creates ambiguity: is `initialSupply` "whole tokens" while `mint()` expects "wei units", or vice versa? In CCIP or bridging contexts, unit mismatches can cause cross-chain supply drift, incorrect bridge accounting, or operational errors (e.g., minting `1e18` too much or too little).
Recommendation	Make all external-facing amounts use the same denomination
Status	Fixed
Comment	▪ Standardized all token amounts to be expressed in the smallest unit (i.e., scaled by `10^decimals`). ▪ Updated constructor documentation to explicitly state that `initialSupply` must be provided in scaled form (e.g., `1_000_000 * 10**18` for 1M tokens). ▪ Added consistent NatSpec comments to `mint()` and `burn()` functions reinforcing this convention. ▪ Documented this requirement prominently in the project README.

02	CCIP Pool burn will revert unless the pool is granted burn permissions
Severity	Medium
Description	`XFYTokenPool._lockOrBurn(amount)` calls `IBurnMintERC20(address(i_token)).burn(amount)`. In your token, `burn(uint256)` is gated by `onlyRole(CCIP_MINT_BURN_ROLE)`. If the token pool contract is not granted `CCIP_MINT_BURN_ROLE`, CCIP message execution will revert and bridging will halt (DoS of bridge operations).
Recommendation	In deployment runbooks: explicitly grant `CCIP_MINT_BURN_ROLE` to the deployed pool contract address (and revoke from EOAs if required).
Status	Fixed
Comment	Confirmed that `XFYTokenPool.lockOrBurn()` requires the pool to hold `CCIP_MINT_BURN_ROLE`. Exposed a dedicated function `grantCcipMintBurnRole(address)` for secure role assignment. Added explicit deployment instructions in the README requiring this step via governance timelock.

03	CCIP_ADMIN storage is redundant and potentially misleading
Severity	Minor
Description	`CCIP_ADMIN` is stored and returned but not used for authorization. Real authority comes from `DEFAULT_ADMIN_ROLE` and role admins in `AccessControl`. This can mislead integrators/auditors and create operational confusion (e.g., changing role admins does not update `CCIP_ADMIN`).
Recommendation	Either remove `CCIP_ADMIN` entirely, or enforce it as the role admin (e.g., set role admin relationships) and keep it updated via a controlled setter that also updates *AccessControl* admin structure.
Status	Fixed
Comment	Design improved

04	Role naming inconsistency
Severity	Minor
Description	`bytes32 public constant CCIP_MINT_BURN_ROLE = keccak256("CCIP_ADMIN");` The identifier implies mint/burn permissions, but the hashed string is `"CCIP_ADMIN"`. This is not a security bug by itself, but it increases the chance of misconfiguration across deployments and scripts.
Recommendation	Align the string with the role name, e.g. `keccak256("CCIP_MINT_BURN_ROLE")`, and use consistent naming across pool/token/config tooling.
Status	Fixed
Comment	Corrected the role definition to ensure name-hash alignment

05	getCCIPAdmin() can revert if all admins are removed
Severity	Minor
Description	The constructor initially guarantees one admin, but later: ▪ Admins can `renounceRole`, or ▪ Another admin can `revokeRole` from them If, for any reason, all `DEFAULT_ADMIN_ROLE` members are removed, then: ▪ The contract is effectively without an admin ▪ A call to `getCCIPAdmin()` will `revert` because there is no index 0 member. Any CCIP tooling that assumes `getCCIPAdmin()` always succeeds would then fail.
Recommendation	This is an operational / governance edge case, not an exploit. Never clear out all `DEFAULT_ADMIN_ROLE` members.
Status	Acknowledged
Comment	Noted

06	Re-entrancy
Severity	Unknown
Description	`_lockOrBurn` only calls `IBurnMintERC20(address(i_token)).burn(amount)` on a known token contract. That token is *XFYToken* (non-upgradeable, no external callbacks).
Recommendation	Make sure the token is *XFYToken* in `_lockOrBurn`
Status	Acknowledged / Risk is negligible if used as intended
Comment	▪ The `_lockOrBurn` function calls `burn(amount)` on the native *XFYToken*, which performs a standard internal `_burn` with no external calls, callbacks, or hooks. ▪ The token is non-upgradeable and used exclusively by the protocol's own pool.

07	Direct & Indirect Dependencies
Severity	Unknown
Description	The protocol relies on several third-party components whose internal behavior is treated as a black box in this audit. In particular, the system depends on: ▪ *OpenZeppelin* Contracts (`ERC20, ERC20Permit, AccessControl`) for core token logic and authorization. ▪ *Chainlink* CCIP contracts (`TokenPool, BurnMintTokenPoolAbstract, IBurnMintERC20, router, RMN proxy`) for cross-chain messaging, mint/burn orchestration, and rate-limiting. ▪ Externally configured addresses such as the CCIP router, RMN proxy, remote token pools, and allowlisted sender contracts. This review assumes these dependencies are correctly implemented, securely configured, and free of undisclosed vulnerabilities. In practice, bugs, misconfigurations, or upgrades in any of these components could affect message delivery, cross-chain mint/burn behavior, availability of bridging, and potentially lead to unexpected token flows or denial of service. All imported contracts (Chainlink CCIP pools, interfaces, and shared libraries) were treated as trusted black boxes and were not audited for correctness, security, availability, or upgrade safety. As a result, any vulnerabilities, misconfigurations, logic flaws, or breaking changes present in external dependencies, upstream CCIP infrastructure, or future versions of imported contracts are explicitly out of scope and not covered by this audit.

Recommendation	Continuously monitor third-party dependencies and their security posture. Track *OpenZeppelin* and *Chainlink* security advisories and upgrade to patched versions when necessary.
Status	Acknowledged
Comment	XFY team will periodically review *OpenZeppelin* and *Chainlink* releases, verify router/RMN/pool configurations, and deploy updates or configuration changes as required to maintain compatibility and security.

4. CENTRALIZATION

Centralization is one of the leading causes of smart contract-related asset losses. When a contract assigns critical powers to a privileged role- such as an `owner`, `admin`, or designated `controller` - the associated risk becomes elevated, especially if that role is tied to a single externally owned account (EOA). In many cases, privileged roles serve operational or safety functions:

4.1 Noteworthy Privileged Functions

Token (`XFYToken`):

Function / Capability	Role / Authority
`_grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin)` (constructor)	Deployer choice
`grantRole(bytes32 role, address account)` (inherited)	`DEFAULT_ADMIN_ROLE`
`revokeRole(bytes32 role, address account)` (inherited)	`DEFAULT_ADMIN_ROLE`
`setRoleAdmin(bytes32 role, bytes32 adminRole)` (inherited)	`DEFAULT_ADMIN_ROLE`
`grantCcipMintBurnRole(address account)`	`DEFAULT_ADMIN_ROLE`
`burn(uint256 amount)`	`CCIP_MINT_BURN_ROLE`
`burn(address from, uint256 amount)`	`CCIP_MINT_BURN_ROLE`
`burnFrom(address from, uint256 amount)`	`CCIP_MINT_BURN_ROLE`
`repurchaseBurn(uint256 amount)`	`REPURCHASE_ROLE`

Pool (`XFYTokenPool` via `TokenPool` / `Ownable2StepMsgSender`):

Function / Capability	Role / Authority
`transferOwnership`, `acceptOwnership`	Current owner / new owner
`setRouter(address newRouter)`	Pool owner
`applyAllowListUpdates(address[] add, address[] rm)`	Pool owner
`applyChainUpdates(ChainUpdate[] updates)`	Pool owner
`setRateLimitAdmin(address admin)`	Pool owner
`setChainRateLimiterConfig(...)`	Pool owner or rateLimitAdmin
All CCIP remote pool/remote token configuration	Pool owner

01	**Centralized Privileges** **TokenPool owner controls routing & rate limits**
Severity	Centralization
Description	EOAs with control can be compromised via phishing, private key leakage, or insider threats. Malicious or negligent use of privileges can lead to - token supply manipulation, disruption of trading via pausing, arbitrary fee changes or wallet exclusions, asset seizures or rerouting, etc. 4.1 Privileged Functions
Recommendation	Using Multi-Signature Wallets: Assign privileged roles to a multi-sig contract requiring signatures from multiple trusted parties. Time-Locked Functions: Introduce delays before executing sensitive operations, allowing time for community review or cancellation. Role Revocation or Transfer: If privileges are no longer needed post-deployment, renounce them or migrate them to DAO governance. Secure Key Management: Any private keys associated with privileged roles must be protected using hardware wallets, secret sharing schemes, or offline signing protocols.
Status	Fixed
Comment	Implemented a governance time lock, `XFYTImelockController.sol`, with a 7-day minimum delay for all privileged operations, as designed from inception. Ownership and admin roles of both are transferred to the time lock contract at deployment

02	Arbitrary mint / burn (dangerous if roles misused or compromised)
Severity	Centralization
Description	`DEFAULT_ADMIN_ROLE` can grant `CCIP_MINT_BURN_ROLE`, which can: ▪ mint arbitrary amounts to any address (`mint(to, amount)`), and ▪ burn from arbitrary addresses (`burn(from, amount)`), and ▪ burn via allowance (`burnFrom`). This is effectively a supply and balance control lever. In a production token, this must be explicitly disclosed and governed or it can be perceived as a rug-pull vector.
Recommendation	Place `DEFAULT_ADMIN_ROLE` behind a reputable multisig. Add a timelock for role grants and sensitive actions (or a 2-step role transfer). Consider separating powers: one role for CCIP pool only; remove/disable arbitrary `burn(from,…)` unless strictly required.
Status	`mint` ● Removed
Comment	All these functions are gone. Only burns are: `burn(uint256 amount)` - burns from `msg.sender` with `CCIP_MINT_BURN_ROLE`. `repurchaseBurn(uint256 amount)` - burns from `msg.sender` with `REPURCHASE_ROLE`.

03	**Repurchase burn role is privileged but function only burns caller**
Severity	Minor
Description	`repurchaseBurn(amount)` only burns `msg.sender`, so it is not dangerous to third parties, but it creates an operational central point (who can perform repurchase burns and under what policy). `REPURCHASE_ROLE` can call `repurchaseBurn(uint256 amount)` to burn its own tokens only.
Recommendation	Document how `REPURCHASE_ROLE` is assigned and why it is needed
Status	Fixed
Comment	Documented the assignment of `REPURCHASE_ROLE` and its usage

5. DISCLAIMER

InterFi Network provides professional smart contract audits for blockchain-based codebases (commonly known as smart contracts). This audit assessed the reviewed contract(s) for common vulnerabilities, centralization risks, and logic flaws. However, no audit can guarantee the complete absence of bugs or vulnerabilities. This report does not constitute a security guarantee, endorsement, or assurance of business model soundness or legal compliance.

The review is limited strictly to the source code and its logic as provided, and does not extend to compiler behavior, off-chain components, or external integrations. Due to the evolving nature of blockchain technology and associated risks, users should understand that all materials, including this audit report, are provided strictly on an “as is”, “as available”, and “with all faults” basis.

5.1 Confidentiality

This report is confidential and intended solely for the client. It may not be disclosed, reproduced, or relied upon by third parties without prior written consent from InterFi Network. All terms, including confidentiality, liability limitations, and scope, are governed by the audit agreement.

5.2 No Financial Advice

This report is not financial, investment, tax, legal, or regulatory advice. It should not be relied upon for making investment decisions or assessing the value, viability, or safety of any token, product, or platform. No part of this document should be interpreted as an endorsement or recommendation. InterFi Network accepts no liability for any actions taken based on this report.

5.3 Technical Disclaimer

InterFi disclaims all warranties—express, implied, or statutory—including merchantability, fitness for a particular purpose, title, and non-infringement. We do not guarantee that the reviewed contracts are error-free, fully secure, or meet any specific requirements. Audit results may contain false positives or negatives, and findings are subject to the context and limitations of the review scope.

5.4 Timeliness & Accuracy

Audit results reflect the state of the code at the time of review. InterFi makes no commitment to update findings after publication. We do not warrant the accuracy, completeness, or timeliness of information delivered via this report.

5.5 Third-Party Links

This report may contain references or links to external websites and social media accounts. InterFi Network is not responsible for the content or operation of third-party platforms and assumes no liability for actions taken based on their content.

6. ABOUT

InterFi Network is a leading provider of intelligent blockchain solutions, offering secure, scalable, and production-ready smart contract services. Our team specializes in the development, testing, and auditing of smart contracts across a wide range of blockchain ecosystems.

We have delivered:

- 300+ smart contract systems developed
- 2,000+ smart contracts audited
- 500,000+ lines of code reviewed and analyzed

Our technical expertise spans multiple languages including:

- Solidity for EVM-compatible chains (Ethereum, BNB Chain, Polygon, Avalanche, Cronos, Fantom, Velas, Metis, and more)
- Move for next-generation platforms such as Sui and Aptos
- Rust for advanced ecosystems like Solana, Near, and Cosmos SDK-based chains

6.1 Connect with Us

InterFi Network is driven by a multidisciplinary team of engineers, developers, UI/UX specialists, and blockchain researchers. The core team consists of 3 senior members supported by 4+ expert contributors across code auditing, tooling, and protocol design.

- Website: interfi.network
- Email: hello@interfi.network
- GitHub: github.com/interfinetwork
- Telegram (Engineering): @interfiaudits
- Telegram (Onboarding): @interfisupport

THANK YOU