CRYPTO LEGACY × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

April 21, 2025

Abstract ... 3
Disclaimer ... 3
Summary ... 3
General recommendations ... 4
Project overview ... 5
Project description ... 5
Codebase update #1 ... 6
Audit process ... 7
Manual analysis ... 8
Critical issues ... 8
Medium severity issues ... 9
M01. Accumulated fee not reduced on withdrawal (fixed) ... 9
M02. Approval does not work for lifetimeNFT transfer (fixed) ... 9
M03. Inaccurate documentation (addressed) ... 10
M04. Locked NFT may be not considered during deployment (fixed) ... 10
M05. Missing quorum setting check (fixed) ... 11
M06. Missing withdrawal function (fixed) ... 11
M07. Supply limit is not checked (commented) ... 11
M08. Referrer code replacement (fixed) ... 11
M09. User's data reset (fixed) ... 12
M10. Using of zero referral code (fixed) ... 12
M11. Privacy level is lower than expected (commented) ... 13
M12. Beneficiary claim donation can be ignored (fixed) ... 13
M13. Beneficiary can vote multiple times (fixed) ... 14
M14. Incorrect replacement of target address (fixed) ... 14
Low severity issues ... 15
L01. Beneficiaries are not guardians by default (fixed) ... 15
L02. Exact amount of gas (fixed) ... 15
L03. Incorrect event order (fixed) ... 15
L04. Uncalculated totalFee (fixed) ... 15
L05. Unused field (commented) ... 16
Notes ... 17
N01. Locked assets (fixed) ... 17
N02. Missing _disableInitializers (fixed) ... 17
N03. Missing setter event (fixed) ... 17

N04. Non-resetting voting results (fixed) 17
N05. Project's owner role (commented) 18
N06. Redundant check (fixed) 19
N07. Storage layout (fixed) 19
N08. Redundant code (fixed) 19

ABSTRACT

In this report, we consider the security of smart contracts of CryptoLegacy project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of CryptoLegacy smart contracts. We described the audit process in the section below.

The audit showed multiple issues of medium severity:
Accumulated fee not reduced on withdrawal,
Approval does not work for lifetimeNFT transfer, Inaccurate documentation,
Locked NFT may be not considered during deployment (was fixed during the process),
Missing quorum setting check, Missing withdrawal function, Supply limit is not checked,
Referrer code replacement, User's data reset, Using of zero referral code,
Privacy level is lower than expected, Beneficiary claim donation can be ignored,
Beneficiary can vote multiple times. Also, several low-severity issues were found.

All the tests passed.

After the initial audit, the codebase was updated. The developers either fixed or provided comments for all identified issues. The statuses of the medium severity issues:

- Fixed: Accumulated fee not reduced on withdrawal, Approval does not work for lifetimeNFT transfer, Locked NFT may be not considered during deployment, Missing quorum setting check, Missing withdrawal function, Referrer code replacement, User's data reset, Using of zero referral code, Beneficiary claim donation can be ignored, Beneficiary can vote multiple times.
- Commented: Supply limit is not checked, Privacy level is lower than expected.
- Addressed: Inaccurate documentation.

The medium severity issue Incorrect replacement of target address was discovered by us during the recheck and was subsequently fixed by the developers. The number of tests and the code coverage increased.

GENERAL RECOMMENDATIONS

We recommend avoiding scope splitting in audits of projects that use the Diamond pattern. Reviewing only part of the diamond can compromise audit quality and lead to missed critical issues, as all facets function operate as a unified system.

PROJECT OVERVIEW

Project description

For the audit, we were provided with CryptoLegacy project on a private repository. The audit was started on commit 53e5d4fa731c5c794402b1efca02bcb201d04dfb and finished on commit c4ee1858b5e66560c0c9ea3dc2cf10c78d9dc8fb.

The scope of the audit included:

- contracts/BeneficiaryRegistry.sol;
- contracts/BuildManagerOwnable.sol;
- contracts/CryptoLegacy.sol;
- contracts/CryptoLegacyBuildManager.sol;
- contracts/CryptoLegacyFactory.sol;
- contracts/CryptoLegacyOwnable.sol;
- contracts/CustDiamondBase.sol;
- contracts/FeeRegistry.sol;
- contracts/LegacyMessenger.sol;
- contracts/LifetimeNft.sol;
- contracts/LockChainGate.sol;
- contracts/PluginsRegistry.sol;
- contracts/libraries/LibCreate2Deploy;
- contracts/libraries/LibCryptoLegacy;
- contracts/libraries/LibCryptoLegacyPlugins;
- contracts/libraries/LibSafeMinimalMultisig;
- contracts/libraries/LibTrustedGuardiansPlugin;
- contracts/plugins/CryptoLegacyBasePlugin;
- contracts/plugins/LegacyRecoveryPlugin;
- contracts/plugins/TrustedGuardiansPlugin.

The documentation for the project included https://docs.cryptolegacy.app/ and chatGPT.

All 40 tests passed successfully. The code coverage was 84.69%.

The total LOC of audited sources is 2505.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 9a1e03b14160d4391bf0f623030e7f822f620f27.

The scope of the audit included the contracts from the inital audit and the new contract **SignatureRoleTimelock**. We identified one medium severity issue, and the developers have either fixed or provided comments for all reported issues.

The number of tests increased. All 58 tests passed. The code coverage was 89.67%.

After this update, the developers uploaded the current version of the code to the public repository at commit 3d9e1b1cdc95e514ef285c945f792d772f62af6c.

AUDIT PROCESS

We started the audit on March 10 and finished on April 4, 2025. The initial audit was conducted in two parts, which resulted in some issues being fixed or commented during the process. Also, several issues were found and fixed by the developers.

We inspected the materials provided for the audit and contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether the code maintains a sufficient privacy level;
- Whether the referral system functions as expected;
- Integration with deBridge;
- Whether plugin functions work together without blocking each other;
- Whether the code corresponds to the documentation;
- Standard Solidity issues.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

We made the recheck on April 14-18, 2025. We checked fixes for issues from the initial audit, checked the new contract, re-ran tests and recalculated the code coverage. We also scanned the project with the Audit Agent tool.

Finally, we updated the report.

We checked the public version of the code to see if it matches the code from the last recheck on April 21, 2025.

MANUAL ANALYSIS

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. **Accumulated fee not reduced on withdrawal** (fixed)

The `custAccumulatedFee` value is not reduced when withdrawing fees in the `FeeRegistry.withdrawAccumulatedFee` function. As a result, `accumulatedFee` no longer matches the expected value (contract balance minus the sum of all referrers' fees). This could lead to:

- Locking subsequent withdrawals if the contract balance does not have enough native tokens.
- Ability of withdrawing accumulated referrers fee improperly.

> **The issue has been fixed and is not present in the latest version of the code.**

M02. **Approval does not work for lifetimeNFT transfer** (fixed)

The `LockChainGate._updateLifetimeNftOwnerOnChain` function internally calls the `_checkDestinationLockedChain` function, which verifies that `msg.sender` has a locked NFT. This implies that the sender must be the owner of the NFT.

However, the sender may also have approval (`lockedNftApprovedTo`) for unlocking and transferring the NFT. Despite this, the check prevents NFT ownership transfers (via the `transferLifetimeNftTo` function) if the sender is an approved address.

> **The issue has been fixed and is not present in the latest version of the code.**

M03. Inaccurate documentation (addressed)

The documentation is well-written and quite complete, but it has inaccuracies, for example:

- Recovery addresses "are never directly linked to the CryptoLegacy contract" according to the documentation (see #M11).
- "All assets are automatically withdrawn from the CryptoLegacy contract to another address after recovery address voting" according to the documentation. However, only ERC-20 tokens can be withdrawn. NFTs cannot be withdrawn if the **NftLegacyPlugin** (out of scope) is used.

The documentation is required to streamline both development and audit processes. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

> `Comment from the developers:`
>
> We agree that some parts of the documentation should be clarified. Recovery addresses are indeed stored as hashes and never linked to the contract state directly. Once they are used, they naturally reveal themselves — but by that point, their purpose has already been fulfilled. The important part is that they remain hidden until activation; after that, privacy is no longer critical, as the recovery process is already underway. From there, users are free to manage their assets however they choose, including via privacy-preserving protocols like 0xbow.io.
>
> When we say that "all assets can be withdrawn," we are referring to currently supported assets — specifically, ERC-20 tokens. NFTs are not yet included in recovery flows, but this is not a limitation of the architecture. NFT recovery will be implemented directly in the NftLegacyPlugin by accessing the same shared storage used by the LegacyRecoveryPlugin, without requiring changes to it. Additionally, support for LP tokens, staked positions, and other protocol-native assets is planned through our modular plugin system and will be added progressively based on demand and audit readiness.

M04. Locked NFT may be not considered during deployment (fixed)

If a user already has a locked **LifetimeNFT** before deploying a **CryptoLegacy** contract via the `CryptoLegacyBuildManager.buildCryptoLegacy` function, their `initialFeeToPay` should be 0 (as specified in `_getAndPayBuildFee` at line 305). However, if the user includes `msg.value > 0` during the deployment of the **CryptoLegacy** contract to pay for cross-chain referralCode creation, they are required to pay an additional fee (`_getAndPayBuildFee` function at line 299), and their locked NFT is not considered. This leads to a transaction revert due to an insufficient amount of native tokens in the `_payFee` function at line 123.

> This issue has been fixed on commit `c4ee1858b5e66560c0c9ea3dc2cf10c78d9dc8fb`.

M05. Missing quorum setting check (fixed)

If the `requiredConfirmations` value is initialized to be greater than the number of voters in the **LegacyRecoveryPlugin** contract, it becomes impossible to reach a voting threshold.

After the distribution begins, the owner cannot change this value due to the `onlyOwner` modifier in the `lrSetMultisigConfig` function. This modifier includes a check that prevents changes once the distribution has started. As a result, the entire multisig functionality becomes inaccessible.

Also, if `requiredConfirmations = 0` and there is only one voter, their vote should be sufficient to trigger execution. However, the `LibSafeMinimalMultisig._execute` function is not called in the `LibSafeMinimalMultisig._propose` function due to the `s.requiredConfirmations == 1` check, preventing the proposal from being executed.

> The issues have been fixed and are not present in the latest version of the code.

M06. Missing withdrawal function (fixed)

The **FeeRegistry** contract does not have a method for withdrawing fees earned by referrers (in case the fee transfer fails in the `takeFee` function at line 200).

> The issue has been fixed and is not present in the latest version of the code.

M07. Supply limit is not checked (commented)

The `CryptoLegacyBuildManager._mintAndLockLifetimeNft` function, which is called inside the `CryptoLegacyBuildManager._payFee` function, does not check the supply limit of the **LifetimeNFT** before minting the token.

> The NatSpec comment: "This variable is used to block mass minting until the NFT supply is at least the configured amount".

M08. Referrer code replacement (fixed)

The `FeeRegistry.changeCodeReferrer` function allows setting a new owner for a `referralCode`, even if they already have an active referral code. This leads to the overwriting of the `codeByReferrer[_newReferer]` mapping at line 416, causing the previous owner to lose control over their original code, as ownership check is performed through the same mapping in `_checkSenderIsReferrer` at line 402.

Additionally, this function can be used to manipulate `sharePct` and `discountPct` values, which are set via the `setRefererSpecificPct` function for the corresponding referral code.

> The issue has been fixed and is not present in the latest version of the code.

M09. User's data reset (fixed)

The `FeeRegistry._setCustomCode` function has several vulnerabilities:

- During a cross-chain referral code information update via the `crossUpdateCustomCode` function, the `discountPct`, `sharePct`, and `accumulatedFee` fields are reset (as a new structure is initialized) in the `_setCustomCode` function. As a result, the referrer loses any unwithdrawn accumulated fees.
- By initializing a new `Referrer` structure, it is possible to overwrite the `codeByReferrer` mapping for another referrer by assigning them as the new owner. This would prevent the referrer from managing their original referral code due to the `_checkSenderIsReferrer` function.
- If only the recipient is changed and `prevOwner` and `_referrer` are the same address, the value of the `codeByReferrer[prevOwner]` mapping is deleted at line 238, causing the owner to lose control over their original code due to the `_checkSenderIsReferrer` function.

> **The issues have been fixed and are not present in the latest version of the code.**

M10. Using of zero referral code (fixed)

According to the logic, the zero `referralCode` should not belong to anyone, and the `discountPct` and `sharePct` values should be zero (as specified at line 459 in the `FeeRegistry.getCodePct` function).

However, any user (who does not have a `referralCode`) can set any owner and recipient to the zero `referralCode` through the `FeeRegistry.changeCodeReferrer` and `FeeRegistry.changeRecipientReferrer` functions.

> **The issues have been fixed and are not present in the latest version of the code.**

M11. Privacy level is lower than expected (commented)

According to the documentation, guardians and recovery addresses are securely stored as hashes, preventing any direct link to **CryptoLegacy** contract. And guardians can decrypt the encrypted asset data from transaction events only once the threshold condition is met, allowing emergency transfer of assets into **CryptoLegacy** contract.

However, guardians and recovery addresses are stored as `bytes32` hashes in the mappings for privacy. However, they reveal themselves upon their first interaction with the project, long before the distribution begins — when they start voting.

The same applies to beneficiary addresses — they reveal themselves when calling `CryptoLegacyBasePlugin.initiateChallenge` function to start the distribution process. And although they have the `CryptoLegacyBasePlugin.beneficiarySwitch` function, they will still reveal themselves when making a claim.

> `Comment from the developers:`
>
> CryptoLegacy keeps guardians, beneficiaries, and recovery addresses private by storing them as hashes. This means no one can see who they are on-chain until they actually take action. Once someone starts a challenge or claims assets, their address becomes visible — and that's expected. At that point, they're simply executing their role, and privacy is no longer a concern. The assets have already entered the claim phase, and control shifts to the recipient. For added safety, we recommend using fresh wallet addresses for each role. If the owner returns during the challenge period, they can reset participants and restore privacy for the next cycle. The current setup provides strong privacy where it matters most — before any action is taken — and ensures the system works reliably once distribution begins. In the future, we plan to offer optional zero-knowledge plugins, including potential integrations with projects like 0xbow.io, for users who require additional privacy. And of course, after claiming, users are free to manage or shield their assets using any private solution they prefer.

M12. Beneficiary claim donation can be ignored (fixed)

During the claim process via the `CryptoLegacyBasePlugin.beneficiaryClaim` function, a beneficiary can donate native tokens to referrers and the **CryptoLegacyBuildManager**. However, this functionality may be ignored if the owner's **LifetimeNFT** is locked inside the `LibCryptoLegacy._takeFee` call as it has the `_isLifetimeActiveAndUpdate` check. And the native tokens will be locked in the **CryptoLegacy** contract.

> The issue has been fixed and is not present in the latest version of the code.

M13. Beneficiary can vote multiple times (fixed)

If the guardians list is not explicitly initialized in the **TrustedGuardiansPlugin** contract, the `CryptoLegacyStorage.beneficiaries` are used as guardians by default.

During the voting process in `guardiansVoteForDistribution`, the address hash of each voting guardian is added to the `guardiansVoted` mapping. However, when a beneficiary changes their address using the `CryptoLegacyBasePlugin.beneficiarySwitch` function, both their actual address and its corresponding hash are updated. This change is not reflected in the `guardiansVoted` array.

As a result, the same beneficiary (as a guardian) can vote multiple times by switching to a new address each time.

> **The issue has been fixed and is not present in the latest version of the code.**

M14. Incorrect replacement of target address (fixed)

The `SignatureRoleTimelock._removeSignatureRole` function is intended to remove the contract address from the `targets` array if it no longer has any function selectors in the `targetSigs` mapping. However, there is a typo at line 259, where the code attempts to remove a different, nonexistent selector.

Consider replacing `targetSigs[_target]` with `targets` at line 259 of the **SignatureRoleTimelock** contract.

> **The issue has been fixed and is not present in the latest version of the code.**

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Beneficiaries are not guardians by default (fixed)

If the `TrustedGuardiansPlugin.guardians` list is not initialized, the `BeneficiaryRegistry.getCryptoLegacyListByGuardian` function will not return any **CryptoLegacy** contract addresses, as the provided guardian address hash is not included in the `cryptoLegacyByGuardian` mapping.

> **The issue has been fixed and is not present in the latest version of the code.**

L02. Exact amount of gas (fixed)

There are external calls in the code that specify an exact amount of gas. In case of a new fork, the cost of opcodes may change, causing the calls to revert due to running out of gas in the project.

> **The issues have been fixed and are not present in the latest version of the code.**

L03. Incorrect event order (fixed)

Consider moving the `ConfirmSafeMinimalMultisigProposal` event before the `if (p.confirms >= s.requiredConfirmations)` check in the `LibSafeMinimalMultisig._confirm` function. Otherwise, the event order may be disrupted. If the proposal is executed in the same transaction as the confirmation, the `ExecuteSafeMinimalMultisigProposal` event could be emitted before the `ConfirmSafeMinimalMultisigProposal` event.

> **The issue has been fixed and is not present in the latest version of the code.**

L04. Uncalculated totalFee (fixed)

The value of the `totalFee` variable is always 0 in the `FeeRegistry._setCrossChainsRef` funciton as it is not increased inside the loop. It does not lead to anything since this function has the same check of cross-chain fees at line 308.

> **The issue has been fixed and is not present in the latest version of the code.**

L05. Unused field (commented)

The `CryptoLegacyStorage.defaultFuncDisabled` field from the **ICryptoLegacy** interface is not used in the current version of the code.

> `Comment from the developers:`
> The `defaultFuncDisabled` field is there on purpose, even if it's not used yet. In the future, some plugins might need to turn off certain default functions, and this field will make that possible. It's part of making the system ready for upgrades and new features later on.

Notes

N01. Locked assets (fixed)

If the owner misses the `update`, the beneficiary calls `initiateChallenge`, waits for the `challengeTimeout`, and then transfers assets to the **CryptoLegacy** contract via `transferTreasuryTokensToLegacy` in the **CryptoLegacyBasePlugin** contract. The assets will be distributed according to the beneficiaries' claims through vesting (the owner cannot withdraw them and cannot call the update to stop distribution). This is the expected behavior.

However, if the `transferTreasuryTokensToLegacy` and `update` functions are called in the same block at the `distributionStartAt` time, the tokens are sent to the **CryptoLegacy** contract and `distributionStartAt` is set to 0. This prevents the beneficiaries from claiming the tokens, but the owner also cannot withdraw them, and they remain locked in the contract. The only way to withdraw them is to go through the procedure again via the `initiateChallenge` function.

> The issue has been fixed and is not present in the latest version of the code.

N02. Missing _disableInitializers (fixed)

According to OpenZeppelin recommendations, the `_disableInitializers` function should be called inside the constructor. However, the constructor of the **FeeRegistry** contract is missing the `_disableInitializers` call.

> The issue has been fixed and is not present in the latest version of the code.

N03. Missing setter event (fixed)

The constructor of the **CryptoLegacyBuildManager** contract does not emit the `SetRegistries` and `SetFactory` events. And the constructor of the **LifetimeNft** contract does not emit the `SetBaseURI` event. Consider adding events to provide more transparent interaction with the code.

> The issues have been fixed and are not present in the latest version of the code.

N04. Non-resetting voting results (fixed)

Consider resetting the `pluginStorage.guardiansVoted` array if the voting is successful in the `TrustedGuardiansPlugin.guardiansVoteForDistribution` function, as it can only be reset through the `resetGuardianVoting` function. Otherwise, in the case of a re-vote, a single non-voted guardian may call `guardiansVoteForDistribution` and succeed in the voting, as previous votes are not reset, and their number exceeds the threshold.

> The issue has been fixed and is not present in the latest version of the code.

N05. Project's owner role (commented)

In the current implementation, the system depends on the owner role. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if owner's private keys become compromised.

The owner can in the project:

- Change the fees charged for updates in the **FeeRegistry** contract;
- Withdraw accumulated fees of referrers if they are accumulated in the **FeeRegistry** contract;
- Change the fees charged for sending cross-chain messages in the **LockChainGate** contract.

The admin role of the **SignatureRoleTimelock** contract can add new target contracts, add function signatures, and grant roles to any address.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> According to the documentation, the project will be run by DAO.

> `Comment from the developers:`
> While some protocol-level contracts initially include an owner role, this role is not held by an individual. Instead, it is assigned to a DAO-controlled multisig system with enforced timelocks and role-based permissions. To strengthen this model, we implemented the SignatureRoleTimelock contract, which provides a structured and transparent way to manage sensitive functions like fee updates, plugin registry changes, and cross-chain settings. Each function is tied to a specific role and requires scheduled execution with a mandatory delay (e.g., 5 days), making abuse or rushed changes practically impossible. Full decentralization is part of our roadmap, with DAO NFT holders gradually gaining more governance power. In this transition phase, the system already operates in a trust-minimized manner, backed by multisigs involving core team members, partner protocols, and external security advisors. Additional details here.

N06. Redundant check (fixed)

The code has following redundant checks:

- The `isLifetimeNftLocked` check in the `CryptoLegacyBuildManager._payFee` function duplicates a check that is called later inside the `_mintAndLockLifetimeNft` function (`_mintAndLockLifetimeNft` -> `IFeeRegistryLocker(address(feeRegistry)).lockLifetimeNft` -> `LockChainGate._writeLockLifetimeNft` at Line 176).
- The `FeeRegistry.createCustomCode` function is called inside the `FeeRegistry.createCode` function, and both functions have the same `_checkSenderIsOperator` check. Consider removing this check from the `createCustomCode` function.
- The `updateCrossChainsRef`, `createCustomRef`, and `createRef` functions of the **CryptoLegacyBuildManager** contract calculate the total cross-chain fees and compare it with `msg.value`. These functions call `FeeRegistry._setCrossChainsRef`, which calculates the cross-chain fees and compares it with `msg.value` twice.

> **The issues have been fixed and are not present in the latest version of the code.**

N07. Storage layout (fixed)

Consider using ERC7201: Namespaced storage layout to simplify future **LockChainGate**, **FeeRegistry** contracts upgrades, as well as OpenZeppelin libraries for upgradeable contracts (such as OwnableUpgradeable).

> **The issue has been fixed and is not present in the latest version of the code.**

N08. Redundant code (fixed)

The **Initializable** contract is imported and inherited in **CryptoLegacyFactory**, but it is not used.

> **The issue has been fixed and is not present in the latest version of the code.**

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Evgeny Bokarev, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

April 21, 2025