CryptAudit

Protocol Audit Report

Version 1.0

Cyfrin.io

January 1, 2026

Protocol Audit Report

Dennis Kilonzo

Dec 30, 2025

Prepared by: CryptAudit Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

A user should be abl;e to store and/or retrieve the password. No other person other than the user should be able to see the password

Disclaimer

The CryptAudit team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings corresponding to the document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
#-- PasswordStore.sol
```

Roles

-Owner: The person who can set and read the password -Outsiders: No other person should be alloweed to set or read the password

Executive Summary

TBA

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	1
Gas Optimizations	0
Total	0

Findings

High Risk Findings

[H-1] Storing the password on-chain makes it visible to anyone and no longer private

Description: All data stored on chain is public and visible to anyone. The `PasswordStore::s_password` variable is intended to be hidden and only accessible by the owner through the `PasswordStore::getPassword` function. I show one such method of reading any data off chain below.

Impact: Anyone is able to read the private password, severely breaking the functionality of the protocol.

Proof of Concept: The below test case shows how anyone could read the password directly from the blockchain. We use foundry's cast tool to read directly from the storage of the contract, without being the owner. Create a locally running chain

make anvil Deploy the contract to the chain

make deploy Run the storage tool

We use 1 because that's the storage slot of s_password in the contract.

cast storage 1 –rpc-url http://127.0.0.1:8545 You'll get an output that looks like this:

0x6d7950617373776f726400000000000000000000000000000000000000000014 You can then parse that hex to a string with:

cast parse-bytes32-string 0x6d7950617373776f726400000000000000000000000000000000000000000014 And get an output of:

myPassword

Recommended Mitigation: The current architecture is fundamentally insecure as it stores sensitive data in plaintext on a public ledger. A more robust approach involves off-chain encryption: the user encrypts their password locally and only stores the ciphertext on-chain. This ensures that even though the storage is public, the data remains unreadable without a private decryption key held only by the user.

Furthermore, you should remove the getPassword() view function entirely. While view functions are intended for gasless "calls," there is a significant risk of user error. If a user—or a poorly configured frontend—accidentally submits a transaction to this function instead of a simple call, the following occurs:

- Mempool Exposure: The request, potentially including parameters or the intent to decrypt, is broadcast to public nodes before it is even mined.
- On-Chain Logging: The transaction is recorded in the blockchain's history. If the function returns the decrypted password as a result of a state-changing transaction, that value can be leaked in the transaction's execution trace.
- Permanent Record: Once the decryption request is "sent" as a transaction, it exists forever in the block history, effectively "doxing" the secret you were trying to protect.

By removing the function, you force the user to retrieve the encrypted data directly from storage and decrypt it locally on their own machine, ensuring the sensitive decryption process never touches the network.

[H-2] PasswordStore::setPassword‘ has no access controls, meaning a non-owner could change the password

Low Risk FIndings

[I-1] `PasswordStore::getPassword` natspec indicates a parameter that doesn't exist, causing the natspec to be incorrect.

Description: ”’ / @notice This allows only the owner to retrieve the password. @> * @param new-Password The new password to set. */ function getPassword() external view returns (string memory) {} ”’

The `PasswordStore::getPassword` function signature is `getPassword()` while the natspec says it should be `getPassword(string)`.

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

”’diff / @notice This allows only the owner to retrieve the password. - * @param newPassword The new password to set. */ ”’