# Protocol Audit Report

Version 1.0

August 27, 2025

# Protocol Audit Report

Kevin Lee

August 20, 2025

Lead Auditors:

- Kevin Lee

## Table of Contents

## Protocol Summary

A simple smart-contract application for storing a password. The intent is:

- The **owner** can set a password.
- The **owner** can later retrieve that password.
- **Non-owners** must not be able to view or modify the password.

The audited contract is referred to as `PasswordStore` (some places in the code/comments say `PasswordStone`; this report assumes `PasswordStore` as the canonical name).

## Disclaimer

Kevin Lee makes best efforts to identify vulnerabilities in the allotted time, but provides **no warranty** that the code is free of vulnerabilities. This audit is **not** an endorsement of the business or product. The review focused only on the security aspects of the Solidity implementation provided.

## Audit Details

### Scope

- Contracts: `PasswordStore.sol`
- Primary state: `string s_password`
- Functions of interest: `setPassword(string)`, `getPassword()`

Out of scope: external infrastructure, front ends, off-chain services, chain configuration.

### Roles

- **Owner**: The account intended to set and read the password.
- **External user**: Any other account interacting with the contract.
- **Observer**: Any party reading blockchain state (including raw storage).

## Executive Summary

The contract, as currently designed and implemented, **does not protect password secrecy** and **lacks access control** on password updates.

- Storing plaintext secrets on-chain is inherently insecure: all node operators and chain observers can read storage directly.
- `setPassword` is **missing an authorization check**, allowing **anyone** to overwrite the stored password.
- Minor documentation/natspec mismatch was observed.

Overall risk: **High**.

### Issues found

| ID | Title | Severity | Status |
|---|---|---|---|
| H-1 | Storing the password on-chain makes it visible to anyone | High | Open |
| H-2 | `setPassword` has no access control; anyone can set/change the password | High | Open |
| I-1 | `getPassword` natspec documents a parameter that does not exist (doc mismatch) | Informational | Open |

## Findings

### [H-1] Storing the password on-chain makes it visible to anyone (no privacy on public chains)

**Description** All data stored on-chain is publicly readable. The variable `PasswordStore::s_password` is intended to be private and only read via `getPassword` by the owner. However, storage can be queried directly from any full node or via RPC, bypassing Solidity visibility.

**Impact** Anyone can read the password, completely breaking the protocol's core security goal.

**Proof of Concept** The following commands demonstrate reading the password from storage in a local devnet:

1. Start a local chain:

```
make anvil
```

2. Deploy the contract:

```
make deploy
```

3. Read the storage slot and decode:

```
# Replace the address with your deployed contract address
cast storage 0x5FbDB2315678afecb367f032d93F642f64180aa3 0
cast parse-bytes32-string 0x6d7950617373776f7264000000000000000000000000000000000000000000000014
```

**Recommended Mitigation** The architecture must change. Never store plaintext secrets on-chain. Suggested pattern:

- Store **only ciphertext** (encrypted password) on-chain.
- Keep **decryption keys off-chain** and never expose them via public endpoints or view functions.
- Remove any view function that would return the secret or permit reconstructing it from public inputs.
- Consider alternative designs: e.g., store a **hash** (for verification) instead of the secret itself; or use hybrid approaches where the contract only validates commitments while the secret lives entirely off-chain.

---

### [H-2] `PasswordStore::setPassword` has no access control (anyone can set the password)

**Description** `setPassword` is `external` and lacks an authorization check, despite the intention that **only the owner** should be able to set a new password.

```
function setPassword(string memory newPassword) external {
    // @audit - There is no access control here
    s_password = newPassword;
    emit SetNetPassword();
}
```

**Impact** Any address can overwrite the stored password, defeating the intended functionality and enabling griefing/DoS against the owner.

**Proof of Concept** Add this test to `PasswordStore.t.sol`:

```
function test_anyone_can_set_password(address randomAddress) public {
    vm.assume(randomAddress != owner);
    vm.prank(randomAddress);
    string memory expectedPassword = "myNewPassword";
    passwordStore.setPassword(expectedPassword);

    vm.prank(owner);
    string memory actualPassword = passwordStore.getPassword();
    assertEq(actualPassword, expectedPassword);
}
```

**Recommended Mitigation** Add an access-control guard. For example:

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

function setPassword(string memory newPassword) external onlyOwner {
    s_password = newPassword;
    emit SetNetPassword();
}
```

Also consider standard libraries (e.g., OpenZeppelin's `Ownable`) to reduce implementation risk.

### [I-1] `PasswordStore::getPassword` natspec indicates a non-existent parameter (documentation mismatch)

**Description** The natspec for `getPassword` references a parameter that does not exist in the function signature, making the documentation inaccurate.

**Impact** Low. Mismatched docs can confuse integrators, auditors, and tooling that rely on natspec.

**Recommended Mitigation** Update natspec to match the current signature. If a parameter was removed during refactoring, delete it from the docs.

## High

- **H-1** Storing plaintext password on-chain reveals it publicly.
- **H-2** Missing access control on `setPassword` allows anyone to overwrite the password.

## Medium

*No medium-severity issues found.*

## Low

*No low-severity issues found.*

## Informational

- **I-1** `getPassword` natspec/doc mismatch.

## Gas

*No gas optimizations proposed; security issues should be addressed first.*