[High-1] Storing the password on-chain makes it visible to everyone, despite the usage of private keyword

Description: All the data stored on-chain is visible to everyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only access through the `PasswordStore::s_password`, which is only intended to be called by the owner of the contract.

Impact: Anyone can read the private password, severaly 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 (https://github.com/foundry-rs/foundry) tool to read directly from the storage of the contract, without being the owner.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

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

```
cast storage <ADDRESS_HERE> 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: Due to this, the overall architecture of the contract should be rethought. One could encrypt the password off-chain, and then store the encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you wouldn't want the user to accidentally send a transaction with the password that decrypts your password.

[High-2] `PasswordStore::setPassword` has no access control, means a non-owner can change the password

Description: `The PasswordStore::setPassword` function is set to be an `external` function, however the natspec of the function and overall purpose of the smart contract is that this function allows only the owner to set a new password.

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

Impact: Anyone can set/change the password of the contract, severely breaking the contracts intended functionality.

Proof of Concept: Add the following to `PasswordStore.t.sol` test file.

```
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 access control conditionals to `setPassword` function.

```
    if(msg.sender != s_owner){
        revert PasswordStore_NotOwner;
    }
```

[Informational-1] The `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 newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
```

The natspec for the function `PasswordStore::getPassword` indicates it should have a parameter with the signature `getPassword(string)`. However, the actual function signature is `getPassword()`.

Impact: The natspec is incorrect.

Recommended Mitigation: Remove the incorrect natspec line.

```
-     * @param newPassword The new password to set.
```