

**LAB 12**

**Part 4: Password Storage**

**Copyright © (2022-2025) All Rights Reserved**

**Dr. Matthew Kisow**

VERSION 1

22-NOV-2024

**LAB**:NaCL Cryptography Programming – Part 4: Password Storage

**OBJECTIVE**: Create a Python script to decrypt a user-specified encrypted file and a password to restore the original secret payload.

**INSTRUCTIONS**: This script will decrypt files encrypted with the **pass\_decrypter.py** script by reversing the encryption process. It will use the scrypt password-hashing mechanism to derive a key from the user-provided password, decrypt the random key (outer box), and then decrypt the payload (inner box) using that random key. Please follow the instructions for submitting this assignment on Blackboard.

1. **Prerequisites**
2. **Setting Up Your Python Environment**

Before you start programming your client, set up your Python environment:

1. Create a virtual environment:

python -m venv .venv

1. Activate the virtual environment:

* **Linux/macOS:**

source .venv/bin/activate

* **Windows:**

.venv\Scripts\activate

1. Install the necessary dependencies:

pip install wheel pynacl requests

1. **Understand the Decryption Process:**
2. Use the script password-hashing mechanism to produce a 32-byte key from a password. CPU and memory difficulty levels (iterations) were changed from the default values to **SCRYPT\_OPSLIMIT\_INTERACTIVE** and **SCRYPT\_MEMLIMIT\_INTERACTIVE** to provide a balance between difficulty and time.
3. Generate and protect a random key using symmetric key encryption, using **Salsa20/Poly1305 MAC** and the password-derived key. This will be the **outer box**.
4. Take the secret payload and protect it via symmetric key encryption with the random key. This will be the **inner box**. Doing this allows the password to be changed in the future without needing to re-encrypt the entire secret payload.
5. The **pass\_encrypter.py** file and the **ciphertext.bin** file have been provided in BlackBoard. The **ciphertext.bin** file has been saved in the following format:
6. **Password salt:** 32 bytes
7. **Outer box**: The encrypted random key is 72 bytes long (32-byte key + 24-byte nonce + 16-byte auth).
8. **Inner box**: The encrypted variable length payload contains (variable-length payload + 24-byte nonce + 16-byte auth). The inner box contains the secret payload.
9. **Writing the pass\_decrypter.py Script**
10. **Import Necessary Modules**

Use the **argparse**, **getpass** and **nacl.utils** modules for file handling.

1. **Define Helper Functions**

**Prompt for Password:** Use a function to collect the password securely:

def get\_password():

"""Prompt the user for a password."""

return getpass.getpass("Enter password: ").encode()

1. **Decrypt Outer and Inner Boxes:**
   * Derive the password key and decrypt the random key (outer box).
   * Decrypt the payload (inner box) with the random key:

def decrypt\_file(password, input\_file, output\_file):

"""Decrypt a file."""

try:

with open(input\_file, "rb") as f:

encrypted\_data = f.read()

except FileNotFoundError:

print("Error: Encrypted file not found.")

exit(1)

*# Extract components*

salt = encrypted\_data[:SALT\_SIZE]

outer\_encrypted = encrypted\_data[SALT\_SIZE:SALT\_SIZE + 72]

inner\_encrypted = encrypted\_data[SALT\_SIZE + 72:]

*# Derive the password key*

password\_key = scrypt.kdf(

SecretBox.KEY\_SIZE,

password,

salt,

opslimit=SCRYPT\_OPSLIMIT,

memlimit=SCRYPT\_MEMLIMIT,

)

*# Decrypt the outer box to retrieve the inner key*

try:

outer\_box = SecretBox(password\_key)

inner\_key = outer\_box.decrypt(outer\_encrypted)

except Exception as e:

print("Error: Invalid password or corrupted outer encryption.")

exit(1)

*# Decrypt the inner box to retrieve the payload*

inner\_box = SecretBox(inner\_key)

try:

payload = inner\_box.decrypt(inner\_encrypted)

with open(output\_file, "wb") as f:

f.write(payload)

print(f"File decrypted and saved to {output\_file}")

except Exception as e:

print("Error: Corrupted inner encryption.")

exit(1)

*# Calculate and display the SHA-256 hash of the decrypted file*

sha256\_hash = calculate\_sha256(output\_file)

print(f"SHA-256 hash of the decrypted file: {sha256\_hash}")

1. **Build the Main Script Logic**
2. Collect the encrypted file path, password, and output file path from the user.
3. Use the functions above to perform decryption.
4. Handle errors (e.g., incorrect password or file not found).
5. **Testing**
   1. Save the script as **pass\_decrypter.py**.
   2. Run it from the command line, providing the **ciphertext.bin** and **otp.bin** files as arguments:

python3 pass\_decrypter.py ciphertext.bin

**DELIVERABLE**

Write a Python3 script named **part4.py** that decrypts the secret payload and saves the resulting file to disk. Your program should print the following output on the screen:

* The SHA-256 hash of the secret payload file. Use the pyNaCl library to compute this hash.
* A human description of what kind of file was produced (in your own words)
* A human description of what kind of viewer could display the file. Name one specific viewer program that would work
* Write a Python 3 program named **part4\_pw\_change.py** that allows the user to change the password without needing to re-encrypt the entire payload. Use the new password **cybersecurity**. The new output file should be named **part4.ciphertext-new.bin**.