

**LAB 12**

**Part 4: Password Storage**

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

**Dr. Matthew Kisow**

VERSION 2

22-DEC-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. **Using the Lab Python Environment**

Navigate to the “*LAB12/Part 2*” folder and enter the Python3 virtual environment.

1. Activate the virtual environment:

source venv/bin/activate

1. **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**.

*This is the working pseudocode solution to the program.*

BEGIN PROGRAM

# Source Code File: Part 4: DECRYPTION: Based on the parameters below.

# Name: part4.py

# Author: <students name>

IMPORT argparse

IMPORT getpass

IMPORT hashlib

IMPORT nacl.utils

IMPORT SecretBox FROM nacl.secret

IMPORT scrypt FROM nacl.pwhash

# Parameters

SET SALT\_SIZE TO 32

SET SCRYPT\_OPSLIMIT TO scrypt.OPSLIMIT\_INTERACTIVE

SET SCRYPT\_MEMLIMIT TO scrypt.MEMLIMIT\_INTERACTIVE

DEFINE FUNCTION get\_password()

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

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

END FUNCTION

DEFINE FUNCTION calculate\_sha256(file\_path)

"""Calculate the SHA-256 hash of a file."""

INITIALIZE sha256\_hash AS SHA-256 object

TRY

OPEN file\_path IN binary mode AS f

FOR byte\_block IN ITERATE f.read(4096) UNTIL byte\_block IS EMPTY

UPDATE sha256\_hash WITH byte\_block

RETURN sha256\_hash.hexdigest()

EXCEPT FileNotFoundError

PRINT "Error: File not found for hash calculation."

TERMINATE PROGRAM WITH EXIT CODE 1

END FUNCTION

DEFINE FUNCTION decrypt\_file(password, input\_file, output\_file)

"""Decrypt a file."""

TRY

OPEN input\_file IN binary mode AS f

READ contents INTO encrypted\_data

EXCEPT FileNotFoundError

PRINT "Error: Encrypted file not found."

TERMINATE PROGRAM WITH EXIT CODE 1

# Extract components

EXTRACT encrypted\_data[:SALT\_SIZE] TO salt

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

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

# Derive the password key

ASSIGN RESULT OF scrypt.kdf(

SecretBox.KEY\_SIZE, password, salt,

opslimit=SCRYPT\_OPSLIMIT, memlimit=SCRYPT\_MEMLIMIT

) TO password\_key

# Decrypt the outer box to retrieve the inner key

TRY

CREATE outer\_box AS SecretBox(password\_key)

ASSIGN RESULT OF outer\_box.decrypt(outer\_encrypted) TO inner\_key

EXCEPT Exception AS e

PRINT "Error: Invalid password or corrupted outer encryption."

TERMINATE PROGRAM WITH EXIT CODE 1

# Decrypt the inner box to retrieve the payload

CREATE inner\_box AS SecretBox(inner\_key)

TRY

ASSIGN RESULT OF inner\_box.decrypt(inner\_encrypted) TO payload

OPEN output\_file IN write binary mode AS f

WRITE payload TO f

PRINT "File decrypted and saved to {output\_file}"

EXCEPT Exception AS e

PRINT "Error: Corrupted inner encryption."

TERMINATE PROGRAM WITH EXIT CODE 1

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

ASSIGN RESULT OF calculate\_sha256(output\_file) TO sha256\_hash

PRINT "SHA-256 hash of the decrypted file: {sha256\_hash}"

END FUNCTION

IF SCRIPT IS EXECUTED AS MAIN PROGRAM THEN

CREATE argparse.ArgumentParser AS parser

ADD argument "input" TO parser WITH description "Path to the encrypted input file"

ADD argument "output" TO parser WITH description "Path to the output decrypted file"

PARSE arguments FROM COMMAND LINE INTO args

CALL get\_password() AND ASSIGN TO password

CALL decrypt\_file(password, args.input, args.output)

END PROGRAM

*This is the working pseudocode solution to the program.*

BEGIN PROGRAM

# Source Code File: Part 4: ENCRYPTION/DECRYPTION: Based on the parameters below.

# Name: part4\_pw\_change.py

# Author: <students name>

IMPORT argparse, os, getpass, tempfile

IMPORT nacl.utils

IMPORT SecretBox FROM nacl.secret

IMPORT scrypt FROM nacl.pwhash

# Parameters

SET SALT\_SIZE TO 32

SET SCRYPT\_OPSLIMIT TO scrypt.OPSLIMIT\_INTERACTIVE

SET SCRYPT\_MEMLIMIT TO scrypt.MEMLIMIT\_INTERACTIVE

DEFINE FUNCTION get\_password(prompt="Enter password: ", confirm=False)

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

WHILE True

ASSIGN getpass.getpass(prompt) TO password

IF confirm IS True THEN

ASSIGN getpass.getpass("Confirm password: ") TO confirm\_password

IF password EQUALS confirm\_password THEN

RETURN password.encode()

ELSE

PRINT "Passwords do not match. Please try again."

ELSE

RETURN password.encode()

END FUNCTION

DEFINE FUNCTION encrypt\_file(password, input\_file, output\_file)

"""Encrypt a file."""

GENERATE random salt AS salt

DERIVE 32-byte key FROM password USING salt AND scrypt PARAMETERS TO password\_key

GENERATE random inner\_key

INITIALIZE inner\_box USING inner\_key

TRY

OPEN input\_file IN binary mode AS f

READ contents TO payload

ASSIGN inner\_box.encrypt(payload) TO inner\_encrypted

EXCEPT FileNotFoundError

PRINT "Error: Input file not found."

TERMINATE PROGRAM WITH EXIT CODE 1

INITIALIZE outer\_box USING password\_key

ASSIGN outer\_box.encrypt(inner\_key) TO outer\_encrypted

OPEN output\_file IN binary write mode AS f

WRITE (salt + outer\_encrypted + inner\_encrypted) TO f

PRINT "File encrypted and saved to {output\_file}"

END FUNCTION

DEFINE FUNCTION decrypt\_file(password, input\_file, output\_file)

"""Decrypt a file."""

TRY

OPEN input\_file IN binary mode AS f

READ contents TO encrypted\_data

EXCEPT FileNotFoundError

PRINT "Error: Encrypted file not found."

TERMINATE PROGRAM WITH EXIT CODE 1

EXTRACT salt FROM encrypted\_data[:SALT\_SIZE]

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

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

DERIVE password\_key FROM password USING salt AND scrypt PARAMETERS

TRY

INITIALIZE outer\_box USING password\_key

ASSIGN outer\_box.decrypt(outer\_encrypted) TO inner\_key

EXCEPT Exception

PRINT "Error: Invalid password or corrupted outer encryption."

TERMINATE PROGRAM WITH EXIT CODE 1

INITIALIZE inner\_box USING inner\_key

TRY

ASSIGN inner\_box.decrypt(inner\_encrypted) TO payload

OPEN output\_file IN binary write mode AS f

WRITE payload TO f

PRINT "File decrypted and saved to {output\_file}"

EXCEPT Exception

PRINT "Error: Corrupted inner encryption."

TERMINATE PROGRAM WITH EXIT CODE 1

END FUNCTION

DEFINE FUNCTION change\_password(input\_file)

"""Change the password of an encrypted file."""

PROMPT user FOR old\_password USING get\_password("Enter old password: ")

PROMPT user FOR new\_password USING get\_password("Enter new password: ", confirm=True)

TRY

OPEN input\_file IN binary mode AS f

READ contents TO encrypted\_data

EXCEPT FileNotFoundError

PRINT "Error: Encrypted file not found."

TERMINATE PROGRAM WITH EXIT CODE 1

EXTRACT salt, outer\_encrypted, inner\_encrypted FROM encrypted\_data

DERIVE old\_password\_key FROM old\_password USING salt AND scrypt PARAMETERS

TRY

INITIALIZE outer\_box USING old\_password\_key

ASSIGN outer\_box.decrypt(outer\_encrypted) TO inner\_key

EXCEPT Exception

PRINT "Error: Invalid old password or corrupted outer encryption."

TERMINATE PROGRAM WITH EXIT CODE 1

GENERATE new\_salt

DERIVE new\_password\_key FROM new\_password USING new\_salt AND scrypt PARAMETERS

INITIALIZE new\_outer\_box USING new\_password\_key

ASSIGN new\_outer\_box.encrypt(inner\_key) TO new\_outer\_encrypted

CREATE temporary file AS temp\_file

WRITE (new\_salt + new\_outer\_encrypted + inner\_encrypted) TO temp\_file

REPLACE input\_file WITH temp\_file

PRINT "Password changed successfully."

END FUNCTION

IF THIS SCRIPT IS EXECUTED AS MAIN PROGRAM THEN

CREATE argparse.ArgumentParser AS parser

ADD subcommands encrypt, decrypt, change-password TO parser

PARSE command-line arguments TO args

IF args.command == "encrypt" THEN

PROMPT user FOR password USING get\_password(confirm=True)

CALL encrypt\_file(password, args.input, args.output)

ELSE IF args.command == "decrypt" THEN

PROMPT user FOR password USING get\_password()

CALL decrypt\_file(password, args.input, args.output)

ELSE IF args.command == "change-password" THEN

CALL change\_password(args.input)

END PROGRAM