**Summary**

Part A presents a comprehensive breakdown of a secure shell script named "code.sh", developed in Kali Linux 2023.1, aimed at **encrypting** and **decrypting** files using AES-256 encryption via OpenSSL. The script uses a sample file named "data.txt", containing the text **"Hi, my name is Aman Shaikh"**, which it encrypts to **"data.txt.enc"** and decrypts back to "data.txt\_decrypted". The process starts with a user-friendly menu that prompts the user to select encryption or decryption. During encryption, the user is asked to securely enter and confirm a passphrase, which is then used with PBKDF2 and a salt to derive a strong key. File permissions are tightly controlled using chmod (400 for encrypted files, 600 for decrypted ones) to ensure only the owner has access. Decryption similarly requires the original passphrase and includes a verification step to prevent unauthorized access. All actions, including **SHA256** checksums and timestamps, are logged in **"audit\_log.txt"** to ensure traceability and integrity of the process. Overall, the script offers a practical example of **cryptographic** file protection with secure handling, verification, and detailed auditing.

**Table of Contents**

1. **Introduction**
   1. **Overview of the Script**
   2. **Linux Distribution**
2. **Shell Script Overview**

**1.1 Overview**

1. **Encryption Process**

**3.1 Passphrase Entry and Confirmation**

**3.2 Key Derivation and Encryption**

**3.3 Secure File Storage**

1. **Decryption Process**

**4.1 Passphrase Verification**

**4.2 Decryption**

**4.3 Secure File Permissions**

1. **Audit Logging**

**5.1 Timestamped Activity Logs**

**5.2 Ensuring Confidentiality of Logs**

**Introduction**

**Overview of the Script**

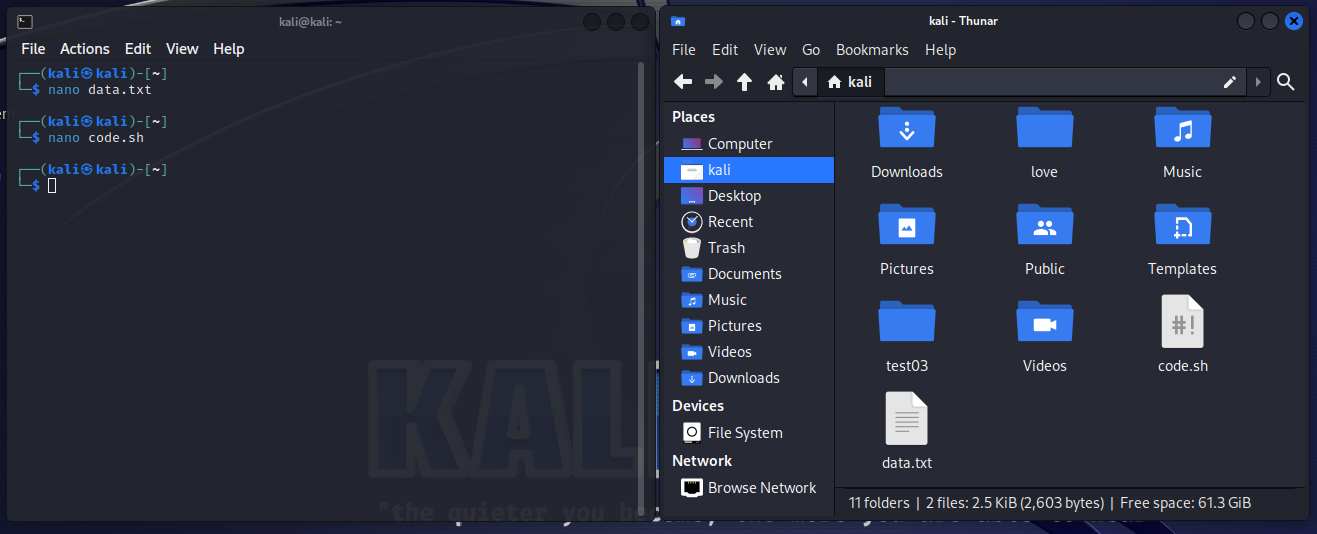
This assignment explores a robust shell script, named "code.sh," designed to provide users with a secure and user-friendly means of encrypting and decrypting files, using the example file "data.txt." The content of "data.txt" is "Hi, my name is Aman Shaikh." After encryption, the new file is named "data.txt.enc," and after decryption, the new file is named "data.txt\_decrypted." The content "Hi, my name is Aman Shaikh" will be written to the "data.txt\_decrypted" file. The audit logs are saved in the file "audit\_log.txt." Leveraging strong symmetric encryption algorithms, such as AES, the script ensures data confidentiality, integrity, and traceability through passphrase protection, proper permissions, and comprehensive audit logging.

**Linux Distribution**

For the Kali Linux operating system, version 2023.1, known for its expertise in security and penetration testing tools, the shell script "code.sh" was painstakingly created. This script uses the sample file "data.txt" to provide users with a safe and simple way to encrypt and decrypt files. The strong AES-256 algorithm and OpenSSL are used to encrypt the text "Hi, my name is Aman Shaikh" into the file "data.txt.enc." The script uses password security, appropriate permissions, and thorough audit logging to guarantee confidentiality, integrity, and traceability.A screenshot of a computer

Description automatically generated

**Shell Script Overview**

The script, "code.sh," opens with a menu that is easy to understand and gives users the option to either encrypt or decrypt the file "data.txt." After that, it walks users through entering their key, starts encryption or decryption depending on the option selected, and records all pertinent activity in the "audit\_log.txt" file.

**Encryption Process**

**Passphrase Entry and Confirmation**

Users are required to input a key to encrypt the "data.txt" file. To protect the passphrase's security, the script hides user input and asks for confirmation to avoid input errors. A screenshot of a computer

Description automatically generated

**Key Derivation and Encryption**

The provided passphrase is used to derive an encryption key using the AES algorithm. The script then encrypts the content "Hi, my name is Aman Shaikh" in the file "data.txt" securely using OpenSSL with the AES-256 encryption algorithm. The new encrypted file is named "data.txt.enc."

* **-salt**: The **-salt** flag adds a random salt to the encryption process, enhancing security by preventing identical passphrases from producing the same ciphertext.
* **-pbkdf2**: The **-pbkdf2** flag enables the use of the Password-Based Key Derivation Function 2 (PBKDF2), a key derivation function that strengthens the passphrase and improves resistance against brute-force attacks.
* **-iter**: The **-iter** flag specifies the number of iterations for the key derivation process, adding an additional layer of security. In this script, the number of iterations is set to 10,000.

**Secure File Storage**

Only authorised users can access the encrypted file "data.txt.enc" because it is saved with restricted rights (400). Furthermore, checksums are computed both prior to and following encryption in order to identify any possible manipulation that may occur during storage.

* **chmod**: The **chmod** command is used to change the permissions of a file. In this case, **chmod 400** restricts access to the encrypted file "data.txt.enc" to the file owner only.

**Decryption Process**

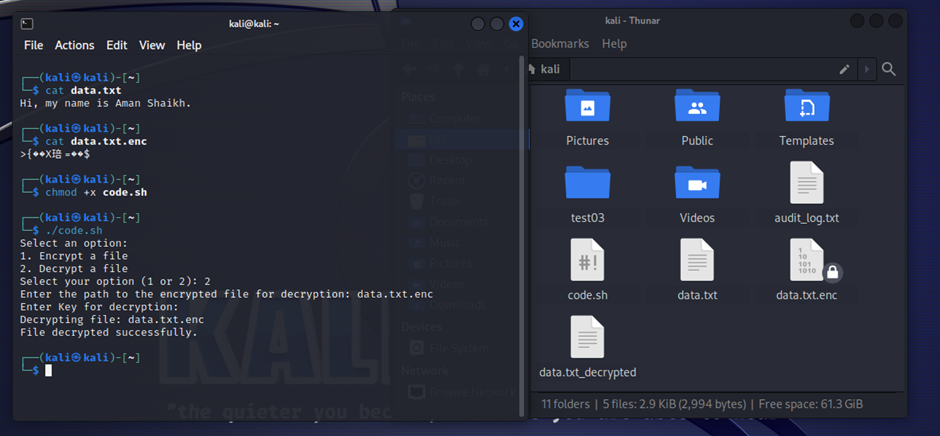
**Passphrase Verification**

The passcode used during encryption must be provided by users to decode the file "data.txt.enc." In the event that the passphrase is entered incorrectly, the script uses OpenSSL for a key verification phase that prevents unauthorised access without actually decrypting the file.

* **exit 1**: The **exit 1** command is used to terminate the script with an exit status of 1, signalling an error condition. In this case, it is triggered when the passphrase provided for decryption is incorrect.

**Decryption**

After the key has been successfully verified, the script uses OpenSSL to safely decode the text "Hi, my name is Aman Shaikh" from the file "data.txt.enc."



The AES-256 algorithm ensures robust protection during the decryption process. The new decrypted file "data.txt\_decrypted" will have the content "Hi, my name is Aman Shaikh."

A screenshot of a computer

Description automatically generated

* **-in and -out**: The **-in** flag specifies the input file (in this case, "data.txt.enc"), and the **-out** flag specifies the output file (in this case, "data.txt\_decrypted").
* **-pass**: The **-pass** flag is used to provide the passphrase for the decryption process. In this script, it's specified as "pass:$passphrase."
* **-pbkdf2 and -iter**: Similar to the encryption process, these flags are used for key derivation during decryption, ensuring consistency in the security measures applied.

**Secure File Permissions**

The decrypted file "data.txt\_decrypted" is assigned restrictive permissions (600) to restrict access, maintaining the security posture established during encryption.

**Audit Logging**

**Timestamped Activity Logs**

Every encryption and decryption activity for the file "data.txt" using the script "code.sh" is logged in the "audit\_log.txt" file, capturing essential details such as file names, timestamps, original checksum, checksum and encrypted or decrypted.

A computer screen shot of a computer screen

Description automatically generated

* **sha256**: The **sha256** command is used to calculate the checksum (hash) of a file. I generated checksums for both the original "data.txt" and the encrypted "data.txt.enc" files.

**Ensuring Confidentiality of Logs**

The audit log is safely kept under the name "audit\_log.txt" to guard against illegal access.

**Code**

# Main Program

echo "Select an option:"

echo "1. Encrypt a file"

echo "2. Decrypt a file"

read -p "Select your option (1 or 2): " choice

# Function to encrypt a file

ITERATIONS=10000

encrypt\_file() {

echo -n "Enter Key for encryption: "

stty -echo

read keyy

stty echo

echo

echo -n "Confirm Key: "

stty -echo

read confirm\_key

stty echo

echo

if [ "$keyy" != "$confirm\_key" ]; then

echo "Keys do not match. Aborting."

exit 1

fi

echo "Encrypting file: $1"

openssl enc -aes-256-cbc -salt -in "$1" -out "$1.enc" -pass "pass:$keyy" -pbkdf2 -iter "$ITERATIONS"

# Set proper permissions for the encrypted file

chmod 400 "$1.enc"

# Calculate checksum before and after encryption for integrity check

original\_checksum=$(sha256sum "$1" | awk '{print $1}')

encrypted\_checksum=$(sha256sum "$1.enc" | awk '{print $1}')

# Log encryption activity with checksums

echo "$(date) - Encrypted file: $1, Original Checksum: $original\_checksum, Encrypted Checksum: $encrypted\_checksum" >> audit\_log.txt

}

# Function to decrypt a file

decrypt\_file() {

echo -n "Enter Key for decryption: "

stty -echo

read keyy

stty echo

echo

# Check the Key is correct

if ! openssl enc -d -aes-256-cbc -in "$1" -out /dev/null -pass "pass:$keyy" -pbkdf2 -iter "$ITERATIONS" 2>/dev/null; then

echo "Error: Incorrect Key. Aborting."

exit 1

fi

decrypted\_file="${1%.enc}\_decrypted"

echo "Decrypting file: $1"

openssl enc -d -aes-256-cbc -in "$1" -out "$decrypted\_file" -pass "pass:$keyy" -iter "$ITERATIONS"

# Set permissions for decrypted file

chmod 600 "$decrypted\_file"

# Log decryption activity

echo "$(date) - Decrypted file: $1 -> $decrypted\_file" >> audit\_log.txt

}

case $choice in

1)

read -p "Enter the path to the file for encryption: " file\_to\_encrypt

if [ -f "$file\_to\_encrypt" ]; then

encrypt\_file "$file\_to\_encrypt"

echo "File encrypted successfully."

else

echo "File not found. Aborting."

exit 1

fi

;;

2)

read -p "Enter the path to the encrypted file for decryption: " file\_to\_decrypt

if [ -f "$file\_to\_decrypt" ]; then

decrypt\_file "$file\_to\_decrypt"

echo "File decrypted successfully."

else

echo "File not found. Aborting."

exit 1

fi

;;

\*)

echo "Invalid option. Aborting."

exit 1

;;

esac