# Cryptography Tools in Linux

## [CONFIDENTIAL]

### Introduction

Congratulations on joining the Federal Bureau of Instigation! We are pleased to offer you a competitive starting salary of $[redacted] plus benefits such as [redacted], [redacted] and free pizza on Mondays! Throughout this training, you'll embark on four critical assignments: Encryption, Decryption, Hashing, and Salting. You'll use OpenSSL as your primary tool to execute these tasks successfully. If you encounter difficulties with an assignment, refer to the corresponding section of the employee handbook using the **cat** command. To begin, navigate to the FBI directory by entering the following command: **cd FBI**

### Encryption

*Encryption* is the process of converting data into a coded format to prevent unauthorized access. Today, you will be utilizing symmetric encryption, which uses the same key for both encryption and decryption, meaning both the sender and receiver must share the same secret key to exchange data securely.

29 February 1982: We are in possession of a confidential file for NASA titled "Apolloll.txt," which must be transferred to the CIA. To locate the file, use basic Linux navigation commands to find the directory named Encryption_Dept. Once there, use the **cat** command to display the contents of the file. The document preview is shown below, but the full text is redacted. You will need to access the actual file to view the complete information.

"The moon landing sequence is scheduled to be [redacted] in a [redacted] location in [redacted], with President John P. Kennedy overseeing the [redacted], and Neil Legweak [redacted] the lead astronaut.

[redacted] will [redacted] at the [redacted] the Trinity test [redacted] was [redacted] under the direction of Bobert J. Cllosenheimer. The previous [redacted] involved extensive use of [redacted] and [redacted], so it's critical to ensure that any [redacted] of [redacted] is thoroughly removed.

This document is highly classified and must not be shared with anyone outside the project. As everything is now prepared for the [redacted], please remind the [redacted] that once Neil Legweak delivers the line, "One small step for man," it signals the beginning of the [redacted]."


1. To start, generate a random 256-bit (32 bytes) key and name it `secretKey.key` using the command below:

**openssl rand -out secretKey.key 32**

2. Next, to encrypt the file, use the command below:

**openssl rc4 -e -a -kfile secretKey.key -in Apolloll.txt -out encrypted.txt**

| | |
|---|---|
| **rc4:** | RC4 stream cipher for encrypting. |
| **-e:** | Represents the encryption command. |
| **-a:** | Base64 encoding for the output. |
| **-kfile:** | Key file used for encryption. |
| **-in Apolloll.txt:** | Input file to encrypt. |
| **-out encrypted.txt:** | Redirects the output to a file. |



3. A file named "`encrypted.txt`" should now be generated in the `Encryption_Dept` directory. Use the **ls** command to verify its existence, and the **cat** command to view its contents. Once confirmed, transfer both the key and the encrypted file to the `Decryption_Dept` directory of the `CIA` using the following **mv** commands.

**mv secretKey.key /home/kali/CIA/Decryption_Dept**
**mv encrypted.txt /home/kali/CIA/Decryption_Dept**

## Decryption

1 April 1982: Because you are the new hire, you've been assigned the task of decrypting and storing the file in the other department. Navigate to the directory where you previously transferred the key and the file.

1. The process of decrypting is nearly the same as encrypting. However, being new, they denied you clearance to parts of the command, so you'll have to figure out the rest:

**openssl rc4 [REDACTED] -a -kfile [REDACTED] -in [REDACTED] -out [REDACTED]**


2. Once you decrypt the file, verify its creation with the **ls** command and read its contents using the **cat** command. Check it matches the original file and report your findings.

3. Did the decrypted file match the original?
   - ☐ YES (The file was decrypted successfully).
   - ☐ NO (The file was not decrypted successfully).

4. Once you are done verifying your results, make your way to the `Hashing_Dept` directory of the `CIA` directory.

## Hashing

*Hashing* is the process of converting data into a fixed-size string of characters, typically a hash code, using a mathematical algorithm. It ensures data integrity by producing a unique output for each unique input, making it nearly impossible to reverse or retrieve the original data.

20 April 1982: Earlier this year, there was an incident where someone modified the Zybooks Files to ensure the correct answers were always selected. To prevent this from happening again, generate a unique hash for the file. This hash will serve as a fingerprint—any future alterations to the file will produce a different hash, allowing you to detect unauthorized changes.

1. To create and store a hash of the Zybooks file, use the following command:

**openssl dgst -sha256 [file name].txt > [file name].sha256**

| | |
|---|---|
| **dgst:** | Specifies the digest (hash) command. |
| **-sha256:** | Uses the SHA-256 hashing algorithm. |
| **[file name].txt:** | Input file to hash. |
| **> [file name].sha256:** | Redirects the output to a file. |

2. To verify the hash, regenerate the file's hash using the following commands. Compare the newly generated hash with the existing hashed version. Ensure the hashes match and report your findings.

**openssl dgst -sha256 [file name].txt**

**cat [file name].sha256**


3. Are the hashes different or the same?
   - ☐ SAME (file was not altered).
   - ☐ DIFFERENT (file was altered).

   If different, state why:_____

31 June 1982: A rogue agent has taken the Zybooks files and is holding them for ransom. Once we recover them, we suspect he will modify the files and corrupt the data. He will likely FOLLOW these steps to alter the files.

4. Modify the Zybooks file by using the nano command. Once you are done modifying it, press CTRL+ O, then ENTER to save your changes, and then press CTRL+ X to exit the file.

13 August 1982: We suspect that the Zybooks files have been modified again, and naturally, as the new hire, you're going to be held responsible. Your task is to verify the integrity of the files by comparing the hashes. Hopefully, they match. Report your findings once completed.

5. Regenerate the hash and compare it to the stored hash using the previous commands from step two.

6. Are the hashes different or the same?
   - ☐ SAME (file was not altered).
   - ☐ DIFFERENT (file was altered).

   If different, state why:_____

7. Once you are done verifying your results, make your way to the `Salting_Dept` directory of the `FBI` directory.

## Salting

*Salting* is the practice of adding random data, called a "salt," to a password or other input before hashing it. This enhances security by ensuring that even identical inputs produce different hash values, making it harder for attackers.

IMPORTANT UPDATE: Recently, the password to our Google Drive containing all of our [redacted] files has been compromised. Simply hashing passwords is not sufficient


for security because attackers can use precomputed tables (rainbow tables) to reverse-engineer hashes. Adding a unique salt to each password before hashing significantly enhances security by ensuring that identical passwords result in different hashes.

1. To understand why hashing alone is insufficient, hash the password in the "password.txt" file twice and store the results in separate files named "password1.hash" and "password2.hash" using the following command:

   **openssl dgst -sha256 password.txt > password[x].hash**

   Check the contents of the files by using the **cat** command on both.

2. What do you notice? Are the hashes different or the same?
   - ☐ SAME (hashes are NOT unique and MORE susceptible to attacks).
   - ☐ DIFFERENT (hashes ARE unique and LESS susceptible to attacks).

3. To add uniqueness to the passwords, begin by creating two salts named "salt1.hex" and "salt2.hex" using the following command:

   **openssl rand -hex 16 > salt[x].hex**

4. Using the same password file, repeat the command from step one twice, adding the line **-hmac "$(cat salt[x].hex)"** between **-sha256** and **password.txt.** Store the results in files named "password3.hash" and "password4.hash." This process ensures the hashes are unique by using the two salts you created.

   Now, Check the contents of the files by using the **cat** command on both.

5. What do you notice? Are the hashes different or the same?
   - ☐ SAME (hashes are NOT unique and susceptible to attacks).
   - ☐ DIFFERENT (hashes ARE unique and less susceptible to attacks).


## Overview

In this workshop, we explored essential cryptographic techniques using OpenSSL. You learned how encryption, decryption, hashing, and salting work and gained hands-on experience applying these techniques through the command line.

Key topics included:

1. Symmetric Encryption: Using algorithms like RC4, we encrypted and decrypted files, learning how symmetric encryption relies on the same key for both processes.
2. Key Generation: We used OpenSSL to generate strong cryptographic keys and explored how key length impacts security.
3. Hashing: Participants saw how hashing is used to verify data integrity, focusing on one-way functions that produce unique outputs from input data.
4. Salting: We applied salting to hashes to enhance security, understanding how salts prevent common attacks like rainbow table lookups.

By the end, you will be able to encrypt and decrypt data, generate secure keys, and apply advanced techniques like Base64 encoding and salting. These skills are essential for ensuring data confidentiality and integrity in real-world applications.


Agent Name _____ Completion Date _____

Agent Signature _____