# **Secret-Key Encryption Lab Report**

**By: Zainab Arif**

## **TABLE OF CONTENTS**

1. **Introduction**
2. **Setup & Requirements**
3. **Steps Performed** 
   * Step 1: Setting Up OpenSSL
   * Step 2: Encrypting and Decrypting Using a Password
   * Step 3: Generating a Key and IV
   * Step 4: Encrypting and Decrypting Using a Key File
   * Step 5: Encrypting Using Different AES Modes
   * Step 6: Checking Data Integrity Using HMAC
4. **Observations and Learnings**
5. **Conclusion**

## **1. Introduction**

This report details the implementation of the **Secret-Key Encryption Lab** using **OpenSSL** in **Kali Linux**. The purpose of this lab is to understand the process of encrypting and decrypting files using symmetric-key encryption techniques. Symmetric encryption ensures that data remains confidential by converting readable information into an unreadable format, which can only be decrypted using a specific secret key.

This lab provides practical experience in using the **Advanced Encryption Standard (AES)** and its different key sizes. Additionally, it explores the importance of **initialization vectors (IVs)** and **data integrity verification** through **HMAC (Hash-Based Message Authentication Code)**.

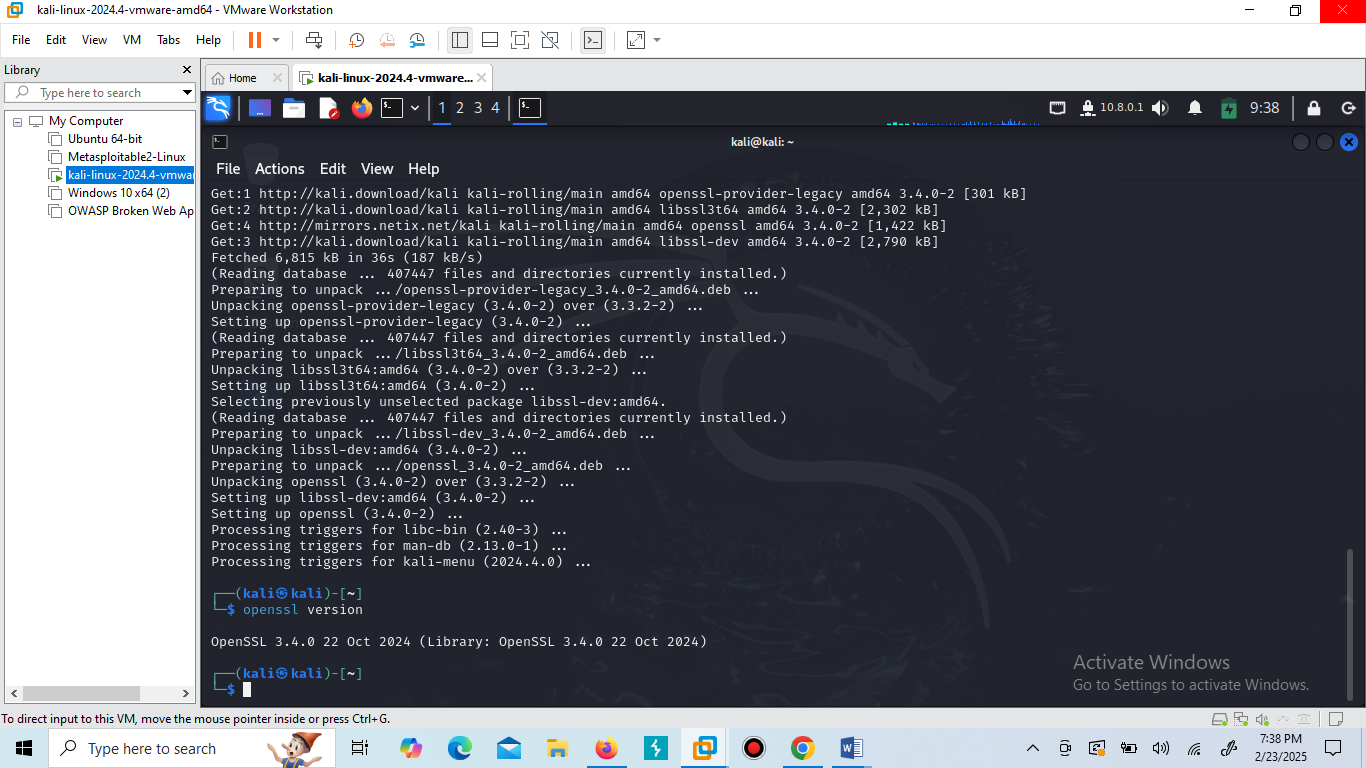
## **2. Setup & Requirements**

* **Operating System:** Kali Linux
* **Software Used:** OpenSSL
* **Test File:** message.txt (Plaintext file used for encryption)
* **Encryption Key:** "seedlabs" (used as a password for encryption)

## **3. Steps Performed**

### ****Step 1: Checking OpenSSL Installation****

OpenSSL is pre-installed in Kali Linux. To verify its version, the following command was executed:

openssl version

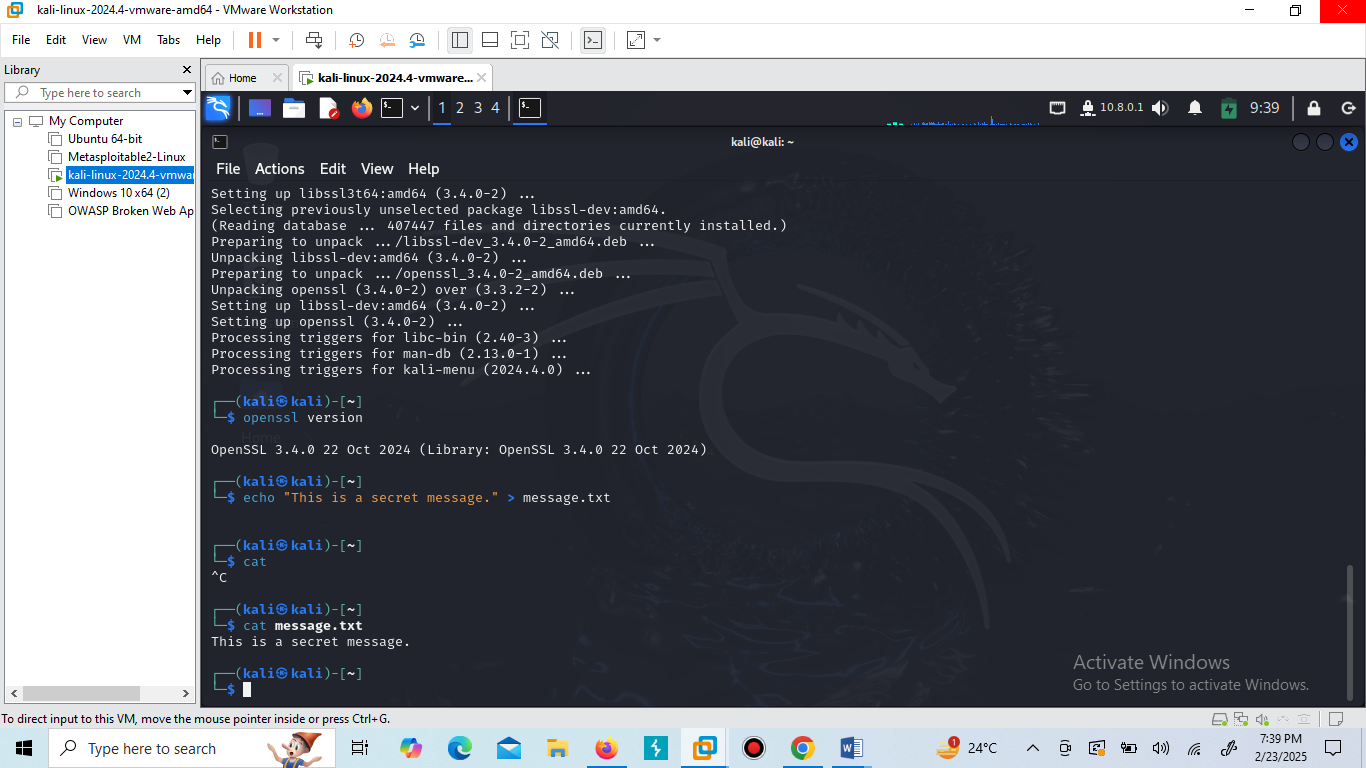
If OpenSSL is not installed, it can be installed using:

sudo apt update && sudo apt install openssl libssl-dev -y

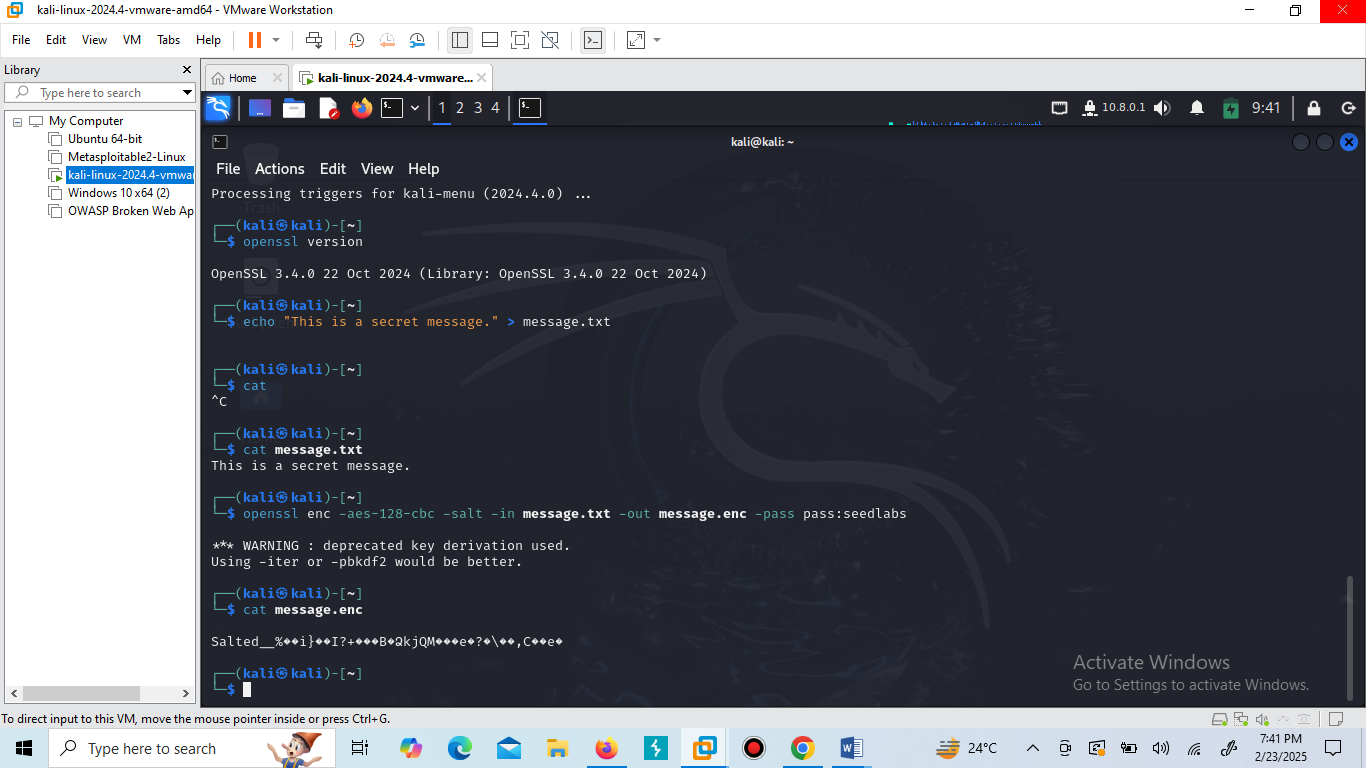
### ****Step 2: Encrypting and Decrypting Using a Password****

#### **1. Creating a Sample File**

A sample plaintext file was created for encryption:  
echo "This is a secret message." > message.txt

To verify its contents:  
cat message.txt

#### **2. Encrypting the File Using AES-128-CBC**

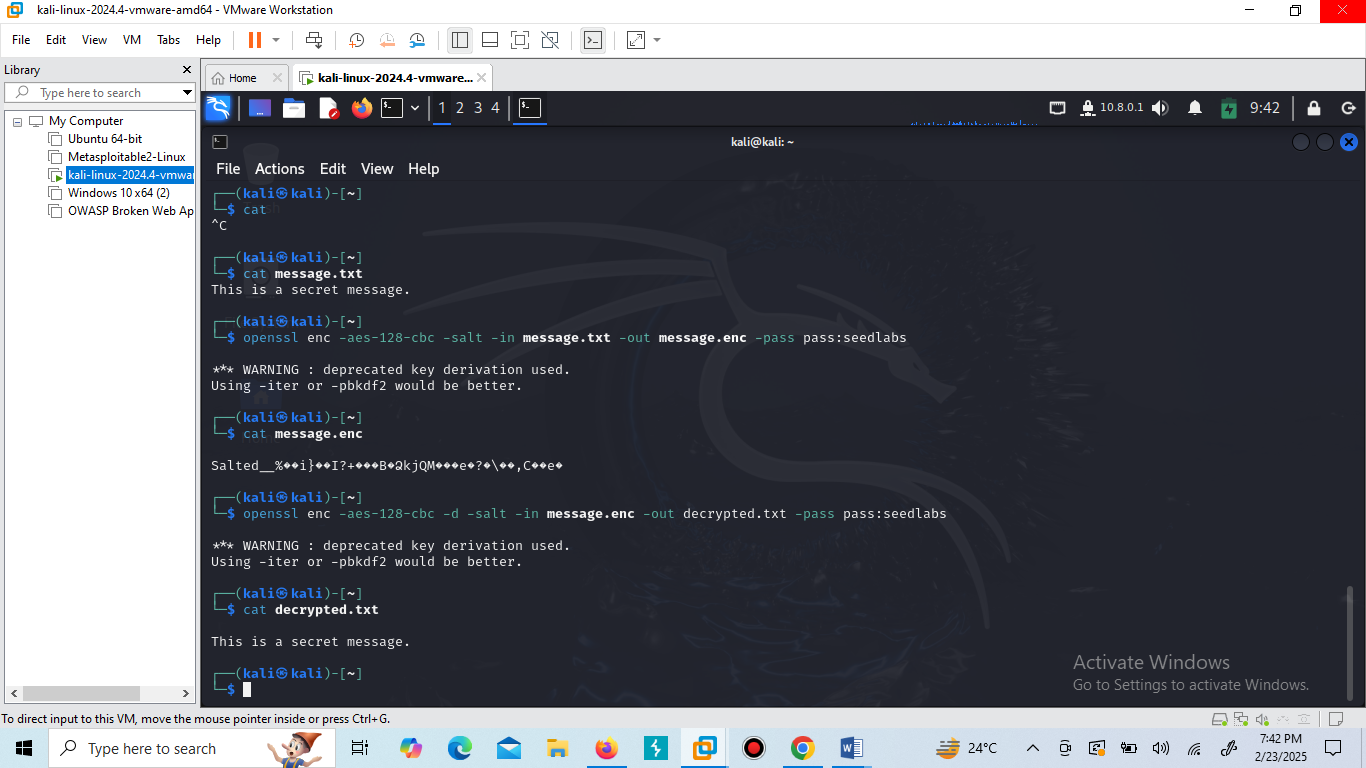
The file was encrypted using **AES-128-CBC** mode with a password:  
openssl enc -aes-128-cbc -salt -in message.txt -out message.enc -pass pass:seedlabs  


After encryption, the file message.enc contains unreadable ciphertext.

#### **3. Decrypting the Encrypted File**

To decrypt and restore the original message:  
openssl enc -aes-128-cbc -d -salt -in message.enc -out decrypted.txt -pass pass:seedlabs

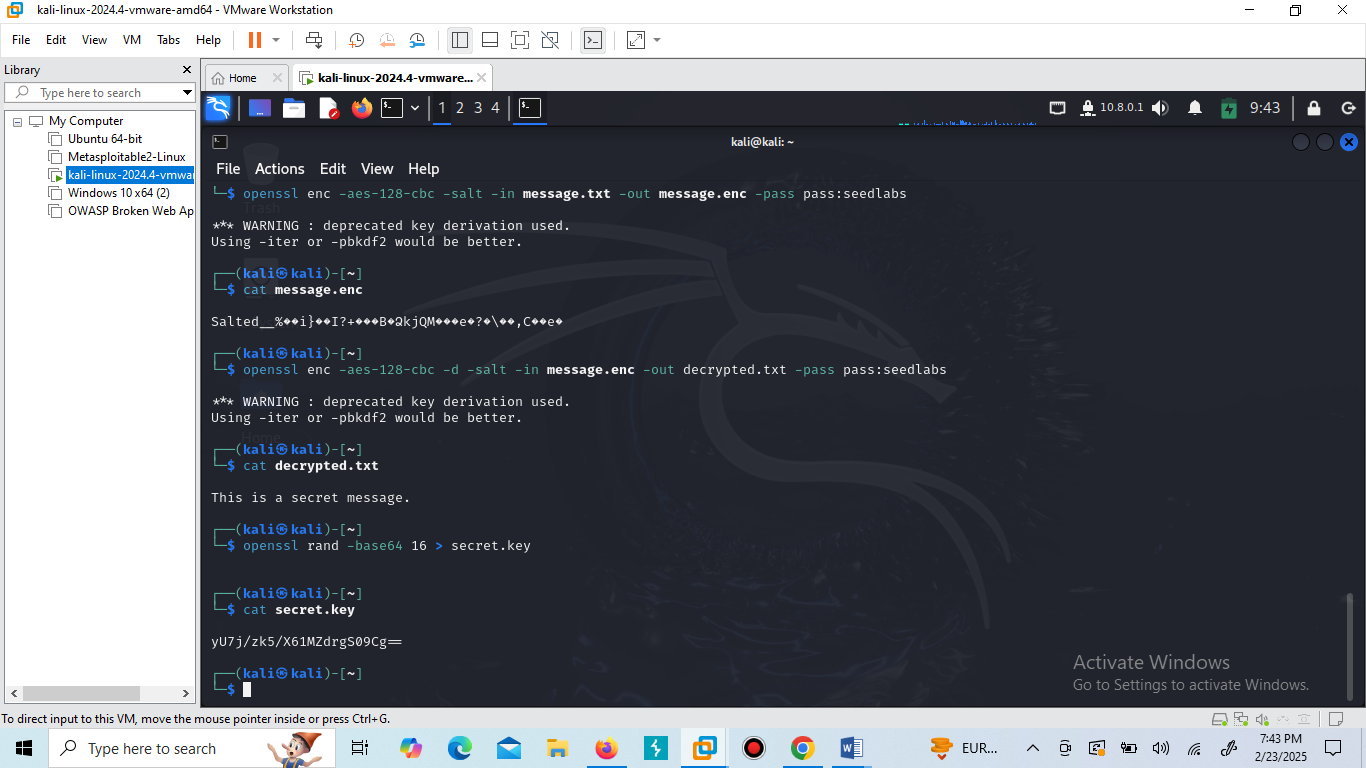
To confirm successful decryption:  
cat decrypted.txt



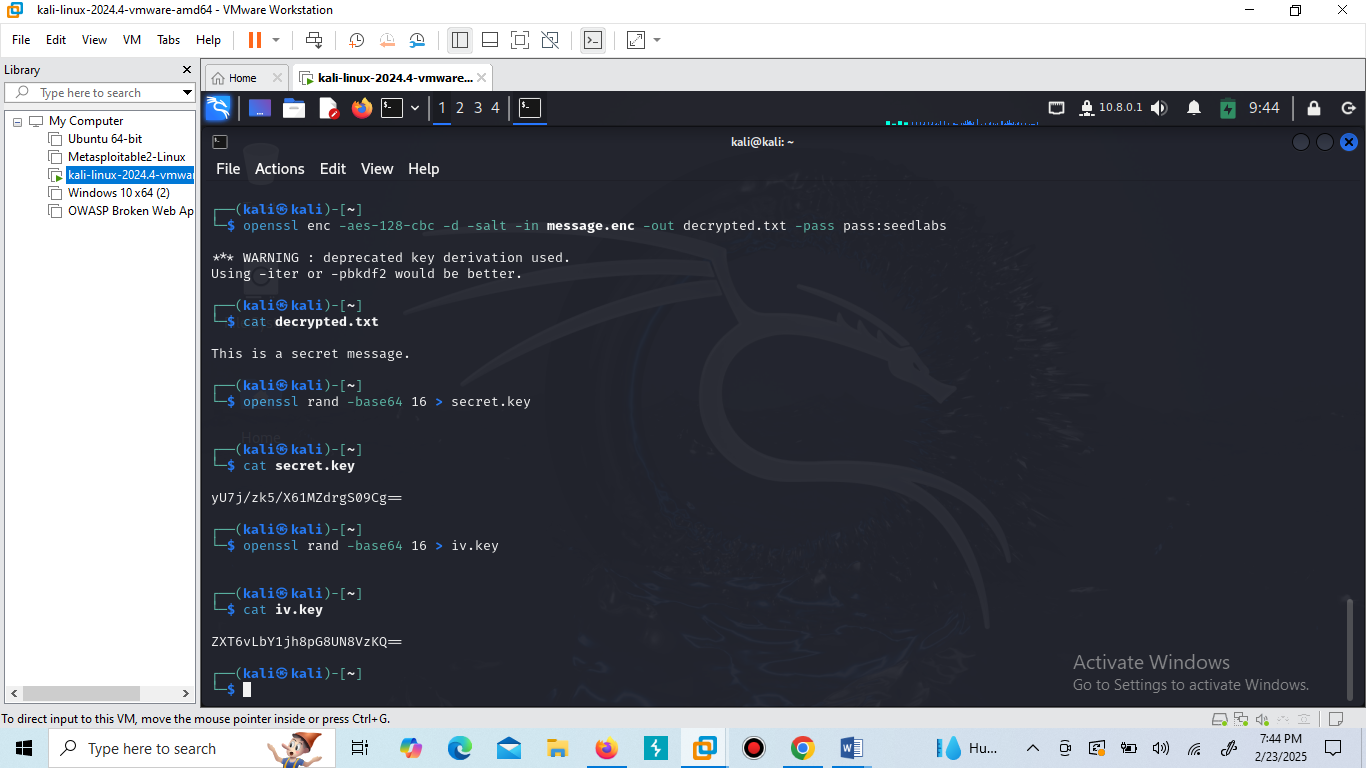
### ****Step 3: Generating a Key and Initialization Vector (IV)****

Instead of using a password, a random encryption key and IV were generated.

#### **1. Generating a Secret Key (16 Bytes)**

openssl rand -base64 16 > secret.key  


#### **2. Generating an Initialization Vector (IV)**

openssl rand -base64 16 > iv.key  


These values ensure added security in encryption.

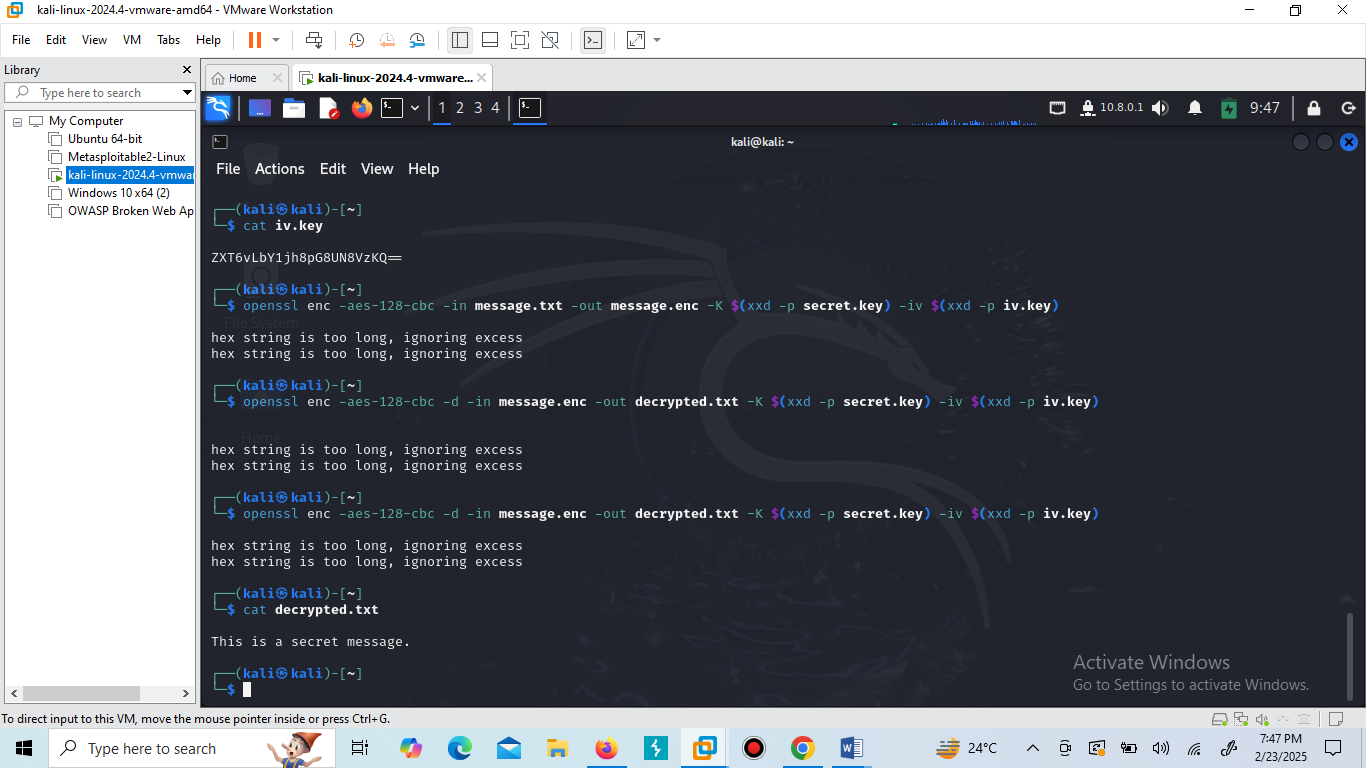
### ****Step 4: Encrypting and Decrypting Using a Key File****

#### **1. Encrypting the File Using the Key and IV**

openssl enc -aes-128-cbc -in message.txt -out message.enc -K $(xxd -p secret.key) -iv $(xxd -p iv.key)

#### **2. Decrypting the File Using the Key and IV**

openssl enc -aes-128-cbc -d -in message.enc -out decrypted.txt -K $(xxd -p secret.key) -iv $(xxd -p iv.key)



This method eliminates the need for a password.

### ****Step 5: Encrypting Using Different AES Modes****

#### **1. AES-128-CBC**

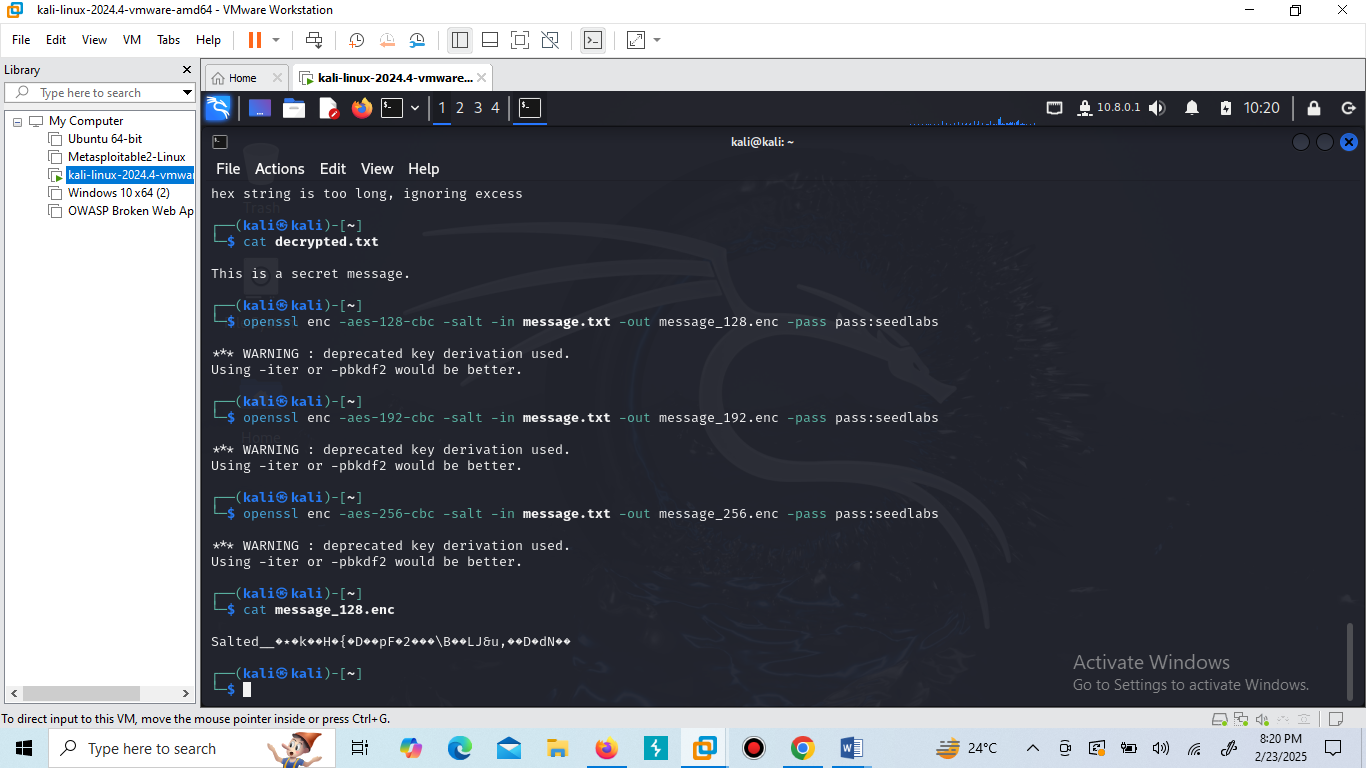
openssl enc -aes-128-cbc -salt -in message.txt -out message\_128.enc -pass pass:seedlabs

#### **2. AES-192-CBC**

openssl enc -aes-192-cbc -salt -in message.txt -out message\_192.enc -pass pass:seedlabs

#### **3. AES-256-CBC**

openssl enc -aes-256-cbc -salt -in message.txt -out message\_256.enc -pass pass:seedlabs

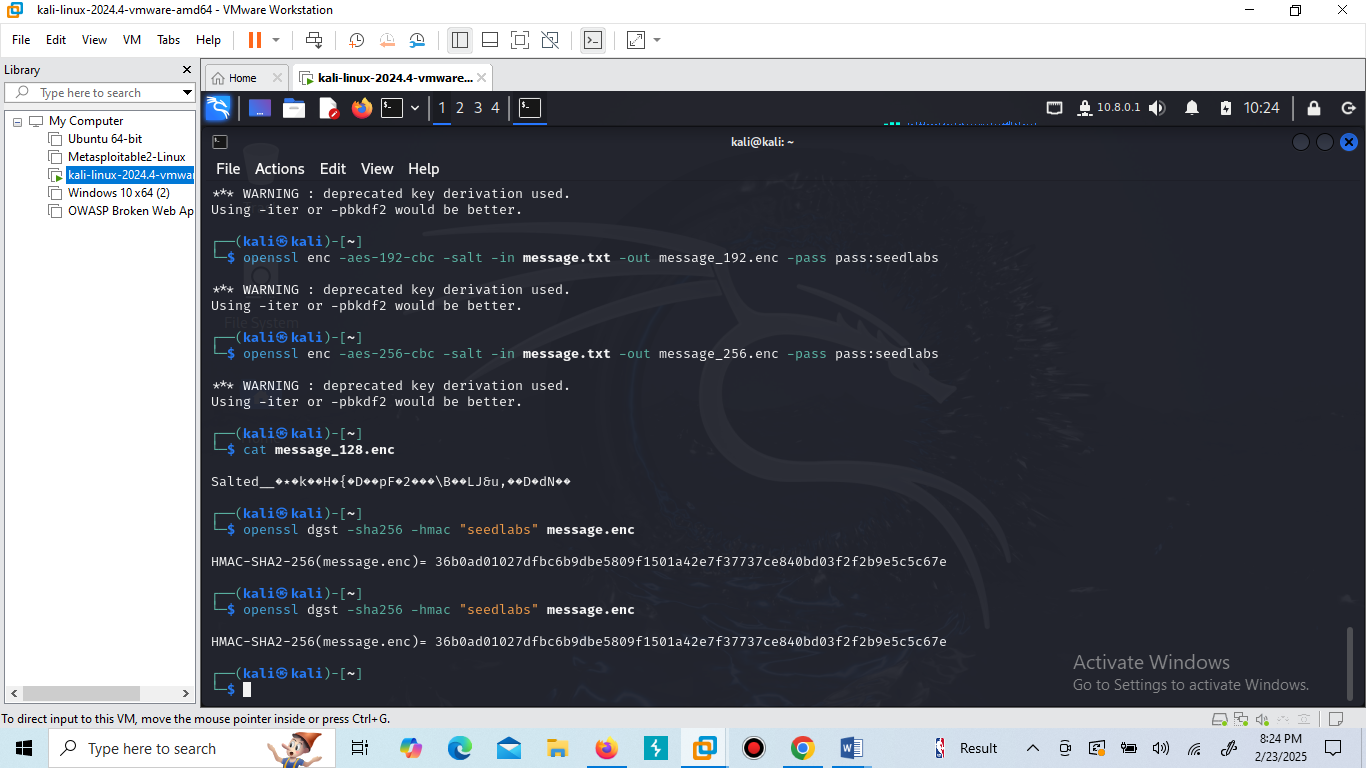


### ****Observations:****

* **AES-128** is faster but less secure.
* **AES-256** offers better security but requires more processing power.

### ****Step 6: Checking Data Integrity Using HMAC****

To verify if the encrypted data remains unaltered, an **HMAC-SHA256** was generated:

openssl dgst -sha256 -hmac "seedlabs" message.enc  


A unique hash value for message.enc. If any changes are made to message.enc, the HMAC value will change, indicating tampering.

## **4. Observations and Learnings**

* AES encryption ensures **confidentiality** by making data unreadable without a key.
* Different AES key sizes offer varying levels of security and performance trade-offs.
* **Initialization Vectors (IVs)** add randomness, preventing pattern-based attacks.
* **Key-based encryption** is more secure than password-based encryption.
* **HMAC provides integrity verification**, ensuring encrypted data is not altered.

## **5. Conclusion**

The **practical application of AES encryption** using OpenSSL. By performing **encryption, decryption, and integrity verification**, a deeper understanding of cryptographic techniques was developed.