Lab 6 - AES Encryption Using Python

Quick Overview of AES:

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. AES has become the most widely used symmetric cipher. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192 or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256 depending on the key length.

Import Important Libraries

```
!pip install pycrypto # To Use Crypto, we need to install pycrypto first.
```

```
from base64 import urlsafe_b64encode, urlsafe_b64decode
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Random import get_random_bytes
import time
```

AES Encryption Implementation

```
# AES Encryption Program
time.clock = time.time



block_size = AES.block_size # Assign AES block size we will use for our code

def pad(s):
  return s + (block_size - len(s) % block_size) * chr(block_size - len(s) % block_size)
def unpad(s):
  return s[:-ord(s[len(s) - 1:])]

def base64pad(s):
  t = "="
  return s + bytes(t, encoding="utf8") * (4 - len(s) % 4)
def base64unpad(s):
  a = "="
  return s.rstrip(bytes(a, encoding="utf8"))

# Encrypt function
def encrypt(key, msg):
    iv = Random.new().read(block_size)
    cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=128)
    encrypted_msg = cipher.encrypt(pad(str(msg)))
    return base64unpad(urlsafe_b64encode(iv + encrypted_msg))

# Decrypt function
def decrypt(key, msg):
    decoded_msg = urlsafe_b64decode(base64pad(msg))
    iv = decoded_msg[:block_size]
    encrypted_msg = decoded_msg[block_size:]
    cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=128)
    return unpad(cipher.decrypt(encrypted_msg))
```

Here we perform AES Encryption testing.

```
# Example 1

wrong_key = 'LKHlhb899Y09olUu'   # Used for testing wrong key or incorrect key
encrypt_key = get_random_bytes(32) # Generate random key or you can use your own. In this code we use AES_256 (32).

hidden_msg = encrypt(encrypt_key, "Attack on December 20th at noon.") # Encrypt message
result = decrypt(encrypt_key, hidden_msg) # Decrypt message

# When incorrect encryption key is used, `decrypt` will return empty string
# So here we check if the string is empty or not and return results accordingly
if result == b'':
      print("Error: Wrong Key Used Or Message Corrupted")
else:
    print("The message is: ", result.decode('utf-8'))
```

```
The message is:  Attack on December 20th at noon.
```

```
# Example 2 - Using Wrong Key

wrong_key = 'LKHlhb899Y09olUu'   # Used for testing wrong key or incorrect key
encrypt_key = get_random_bytes(32) # Generate random key or you can use your own. In this code we use AES_256 (32).

hidden_msg = encrypt(encrypt_key, "Attack on December 20th at noon.") # Encrypt message
result = decrypt(wrong_key, hidden_msg) # Decrypt message

# When incorrect encryption key is used, `decrypt` will return empty string
# So here we check if the string is empty or not and return results accordingly
if result == b'':
      print("Error: Wrong Key Used Or Message Corrupted")
else:
    print(result.decode('utf-8'))
```

```
Error: Wrong Key Used Or Message Corrupted
```

Observations:

In the first example, we use a right encryption key and the AES Encryption program does do its job and encrypts and decrypts correctly. In the second example, we use the wrong key to see what would happen and if our program will handle it correctly instead of decrypting it. As we see on the second example, the error message was printed; hence, the AES Encryption program is doing its job correctly.

References Used:

Cryptography-and-Network-Security-Principles-and-Practice-Global-Edition-by-William-Stallings

https://gist.github.com/stupidbodo/46ffe9b43bc11b43cb02#file-aes_encryption-py

https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/