HOMEWORK A

~Purva Naresh Rumde (pr23b)

Problem ID	Captured Flag/Answer	Steps
P1	Decoded String: **fsuCTF{vuln3rabl3_s33d}**	In this problem I have wrote a python program that works on a AES. SO this program prints key, IV and the flag.
P2	Decoded String: **fsuCTF{n0w_d0n7_G0_pU771ng_4LL_Y0uR_3gg5_1n_4_R4ND0m_b45K37}**	In this problem I have wrote a program using the pwn tools, a function find_seed() which tries to determine the seed used by the server's random number generator. It does this by generating random numbers for different seeds and checking if they match the known output from the server.
P3	Decoded String: **fsuCTF{WH47_4_W1LD_9oO53_Ch423}**	In this problem, the cipher is decoded using base64 and later used ROT on it. Then I used AES decrypt with the key that I received from the message. And that is how I got the flag.

Q1.

In this problem I created a program,

1. Importing Necessary Libraries:

   - The code imports libraries for AES encryption (`Crypto.Cipher.AES`), padding operations (`Crypto.Util.Padding`), handling timestamps (`time`), and generating random numbers (`random`).

2. Reading the Encrypted Message from File:

   - It reads the encrypted message from a file named "ciphertext.txt".

3. Checking Ciphertext Length:

   - Ensures that the length of the ciphertext is a multiple of 16 bytes, which is required for AES decryption.

4. Iterating Through Timestamps:

   - It loops through a range of timestamps, using each timestamp as a seed for generating random numbers.

5. Generating Key and IV:

   - For each timestamp, it generates a random 16-byte key and a random 16-byte Initialization Vector (IV).

6. AES Initialization and Decryption:

   - It initializes an AES cipher with the generated key and IV, then decrypts the ciphertext using AES in CBC mode.

7. Unpadding Decrypted Message:

   - Attempts to unpad the decrypted message to retrieve the original plaintext.

8. Checking for Flag Format:

   - Checks if the decrypted message looks like the flag format (`fsuCTF{...}`). If it does, it prints the key, IV, and decrypted message.

9. Handling Decryption Errors:

   - Catches any decryption errors, typically due to incorrect padding.

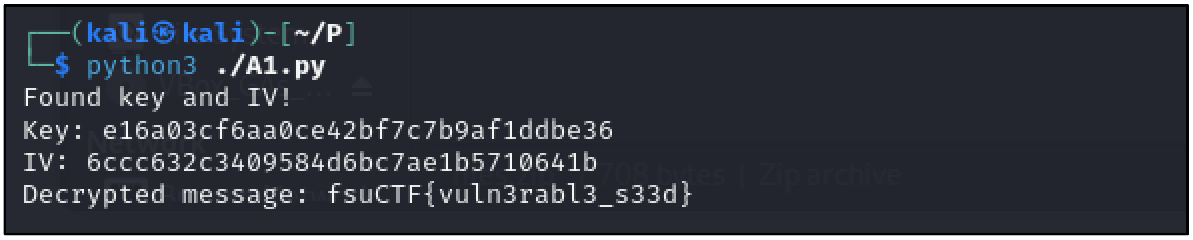
10. Exiting Loop If Key and IV Not Found:

   - If the loop completes without finding a valid key and IV, it prints a message indicating they were not found.

Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import time
import random
# Read ciphertext from file
with open("ciphertext.txt", 'rb') as file:
    enc_msg = file.read()
# Ensure the length of the ciphertext is a multiple of 16 bytes
if len(enc_msg) % 16 != 0:
    print("Error: Ciphertext length is not a multiple of 16 bytes.")
    exit()
# Iterate through a range of timestamps (adjust this range based on your knowledge/guess)
start_t = 1711929600
end_t = int(time.time())  # Current time
for tstamp in range(start_t, end_t + 1):
    random.seed(tstamp)
    enc_key = bytes([random.randint(0, 255) for _ in range(16)])
    enc_iv = bytes([random.randint(0, 255) for _ in range(16)])
    # Initialize AES Cipher with the generated key and IV
    cipher = AES.new(enc_key, AES.MODE_CBC, enc_iv)
    # Decrypt the ciphertext
    dec_msg = cipher.decrypt(enc_msg)
    # Unpad the decrypted message
    try:
        orig_msg = unpad(dec_msg, AES.block_size).decode()
        # Check if the decrypted message looks like the flag
        if orig_msg.startswith("fsuCTF{") and orig_msg.endswith("}"):
            print("Found key and IV!")
            print("Key:", enc_key.hex())
            print("IV:", enc_iv.hex())
            print("Decrypted message:", orig_msg)
            break
    except ValueError:
        # Decryption failed, likely due to incorrect padding
        pass
else:
 print("Key and IV not found.")
```

Flag: **fsuCTF{vuln3rabl3_s33d}**



```
┌──(kali㉿kali)-[~/P]
└─$ python3 ./A1.py
Found key and IV!
Key: e16a03cf6aa0ce42bf7c7b9af1ddbe36
IV: 6ccc632c3409584d6bc7ae1b5710641b
Decrypted message: fsuCTF{vuln3rabl3_s33d}
```

Q2.

In this problem,

1. **Importing Libraries**: The code imports necessary libraries such as pwn, random, and time.
2. **Seed Search**: There's a function find_seed() which tries to determine the seed used by the server's random number generator. It does this by generating random numbers for different seeds and checking if they match the known output from the server.
3. **Main Function**:
   - It establishes a connection with the server.
   - Receives the initial message and the first number from the server.
   - Uses find_seed() to find the seed used by the server.
   - If the seed is found, it proceeds to predict the next numbers generated by the server.
   - It verifies the seed by comparing the first generated number from the server with the predicted one.

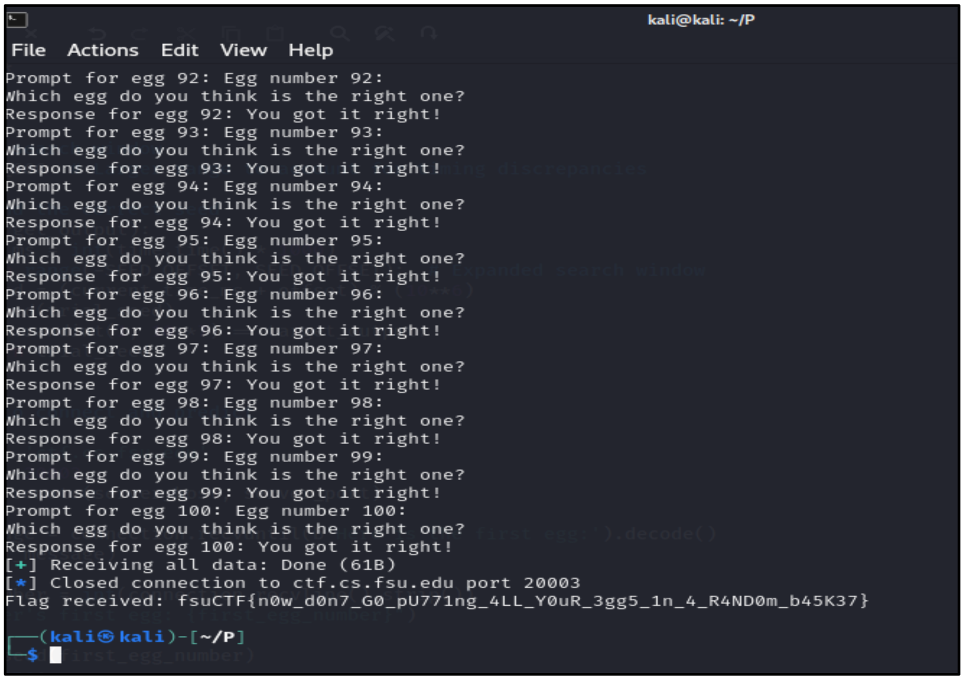
    - Then, it sends 100 guesses to the server and receives responses.
    - If all guesses are correct, it receives and prints the flag.
4. **Proper Execution Check**: It ensures that the main() function is executed only when the script is run directly, not when it's imported as a module.

Overall, the script is designed to connect to a server, predict the server's random number generation, and obtain the flag if successful.

Flag**: fsuCTF{n0w_d0n7_G0_pU771ng_4LL_Y0uR_3gg5_1n_4_R4ND0m_b45K37}**

Code:

```
from pwn import *
import random
import time
# Increased seed search window
SEED_OFFSET = 10_000  # Larger range to account for timing discrepancies
# Function to find the correct seed
def find_seed(target_output):
    current_time_ms = int(time.time() * 1000)
    for offset in range(-SEED_OFFSET, SEED_OFFSET):  # Expanded search window
        trial_seed = (current_time_ms + offset) % (10**6)
        random.seed(trial_seed)
        if random.randint(1, 10**7) == target_output:
            return trial_seed
    return None
# Main function to connect and predict
def main():
    server_host = 'ctf.cs.fsu.edu'
    server_port = 20003
    connection = remote(server_host, server_port)
    initial_message = connection.recvuntil(b'Here is the first egg:').decode()
    print(initial_message)
    first_egg_number = int(connection.recvline().strip())
    print(f"Server's first egg: {first_egg_number}")
    seed = find_seed(first_egg_number)
    if seed is None:
        print("Failed to find the seed.")
        connection.close()
        return
        print(f"Found seed: {seed}")
    # Predict based on the found seed
    random.seed(seed)
    first_random = random.randint(1, 10**7)
    # Re-verify the seed with the first guess to ensure consistency
    if first_random != first_egg_number:
        print("Seed mismatch, exiting.")
        connection.close()
        return
    try:
        for guess_number in range(100):
            # Predict the next value
            next_egg = random.randint(1, 10**7)
            # Receive the prompt for the current guess
            prompt = connection.recvuntil(b'Which egg do you think is the right
one?').decode().strip()
            print(f"Prompt for egg {guess_number+1}: {prompt}")
            # Send the prediction
            connection.sendline(str(next_egg))
            response = connection.recvline().decode().strip()
            print(f'Response for egg {guess_number+1}: {response}')
            if 'wrong' in response:
                print("Incorrect guess, exiting.")
                break
        if guess_number == 99:
            # Expected outcome after all 100 correct guesses
            flag = connection.recvall().decode().strip()
            print("Flag received:", flag)
    finally:
        connection.close()
# Ensure proper script execution check
if __name__ == "__main__":
    main()
```



```
Prompt for egg 92: Egg number 92:
Which egg do you think is the right one?
Response for egg 92: You got it right!
Prompt for egg 93: Egg number 93:
Which egg do you think is the right one?
Response for egg 93: You got it right!
Prompt for egg 94: Egg number 94:
Which egg do you think is the right one?
Response for egg 94: You got it right!
Prompt for egg 95: Egg number 95:
Which egg do you think is the right one?
Response for egg 95: You got it right!
Prompt for egg 96: Egg number 96:
Which egg do you think is the right one?
Response for egg 96: You got it right!
Prompt for egg 97: Egg number 97:
Which egg do you think is the right one?
Response for egg 97: You got it right!
Prompt for egg 98: Egg number 98:
Which egg do you think is the right one?
Response for egg 98: You got it right!
Prompt for egg 99: Egg number 99:
Which egg do you think is the right one?
Response for egg 99: You got it right!
Prompt for egg 100: Egg number 100:
Which egg do you think is the right one?
Response for egg 100: You got it right!
[+] Receiving all data: Done (61B)
[*] Closed connection to ctf.cs.fsu.edu port 20003
Flag received: fsuCTF{n0w_d0n7_G0_pU771ng_4LL_Y0uR_3gg5_1n_4_R4ND0m_b45K37}

┌──(kali㉿kali)-[~/P]
└─$ 
```

Q3.

Flag: **fsuCTF{WH47_4_W1LD_9oO53_Ch423}**

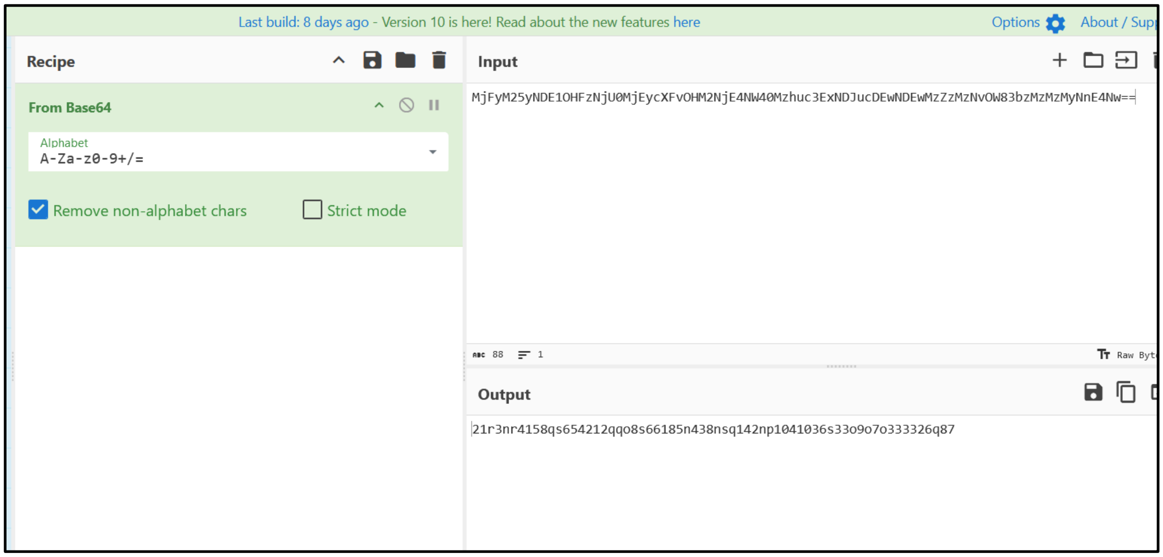
Code to get decoded cipher text and message

```
import base64

# Function to decode base64 encoded text from a file

def decode_base64_from_file(file_path):

    with open(file_path, 'r') as file:

        encoded_text = file.read().strip()

        decoded_text = base64.b64decode(encoded_text).decode('utf-8')

   return decoded_text

# Decode the cipher text

cipher_text = decode_base64_from_file('cipher.txt')

print("Decoded Cipher Text:\n", cipher_text)

# Decode the message text

message_text = decode_base64_from_file('message.txt')

print("\nDecoded Message Text:\n", message_text)

# Function to decode a message using the Caesar cipher with a given shift

def caesar_cipher_decrypt(ciphertext, shift):

    decrypted_text = ""

    for char in ciphertext:

        if char.isalpha():

            # Shift the character by the specified amount

            shifted_char = chr((ord(char) - shift - 65) % 26 + 65) if char.isupper() else chr((ord(char) - shift - 97) % 26 + 97)

            decrypted_text += shifted_char

        else:

            # Keep non-alphabetic characters unchanged
            decrypted_text += char
    return decrypted_text
# Decrypt the message text using rot13 (Caesar cipher with a shift of 13)
decrypted_message = caesar_cipher_decrypt(message_text, 13)
print("Decrypted Message Text:\n", decrypted_message)
```

Output:

```
Decoded Cipher Text:
 21r3nr4158qs654212qqo8s66185n438nsq142np1041036s33o9o7o333326q87
Decoded Message Text:
 Guvf zrffntr unf orra rapelcgrq fb frpheryl gung abobql jvyy rire or noyr gb svther vg bhg vgf fb frpher gung V'yy rira gryy lbh gur xrl: 6n6p776r66753470273o5q3q6n6q6s70
Decrypted Message Text:
 This message has been encrypted so securely that nobody will ever be able to figure it out its so secure that I'll even tell you the key: 6a6c776e66753470273b5d3d6a6d6f70
```

Cipher text from base64 (also used CyberChef)


Last build: 8 days ago - Version 10 is here! Read about the new features here
Options
About / Sup
Recipe
Input
Output
Tт Raw Byt

From Base64

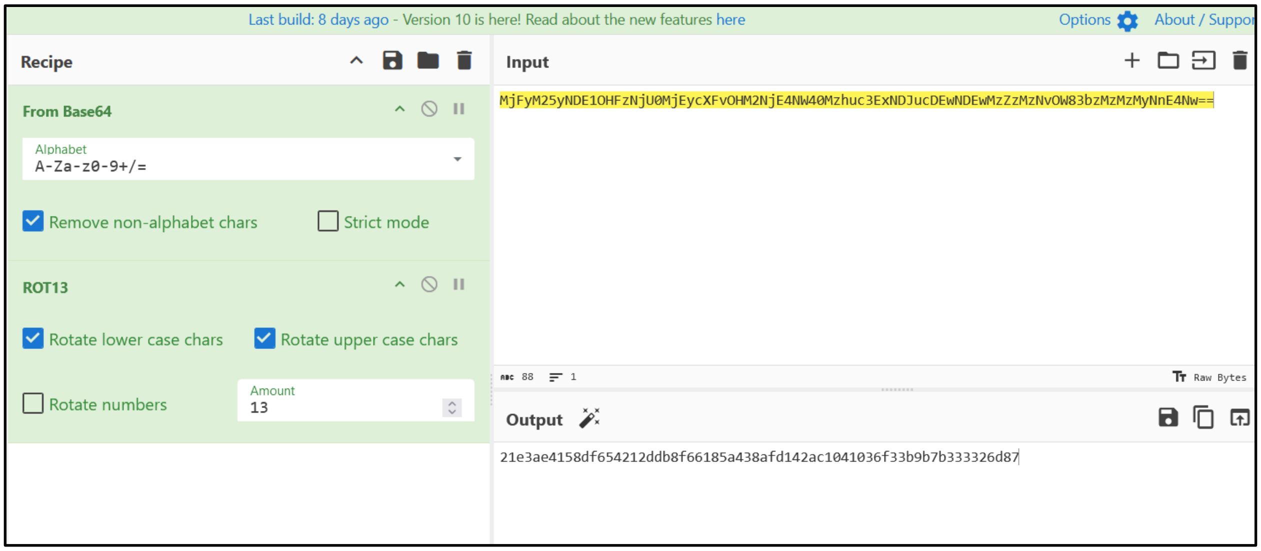
Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars   □ Strict mode

```
MjFyM25yNDE1OHFzNjU0MjEycXFvOHM2NjE4NW40Mzhuc3ExNDJucDEwNDEwMzZzMzNvOW83bzMzMzMyNnE4Nw==
```

ABC 88 1

```
21r3nr4158qs654212qqo8s66185n438nsq142np1041036s33o9o7o333326q87
```



Last build: 8 days ago - Version 10 is here! Read about the new features here
Options
About / Suppor
Raw Bytes

Recipe

From Base64

Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

ROT13

☑ Rotate lower case chars ☑ Rotate upper case chars

☐ Rotate numbers Amount 13

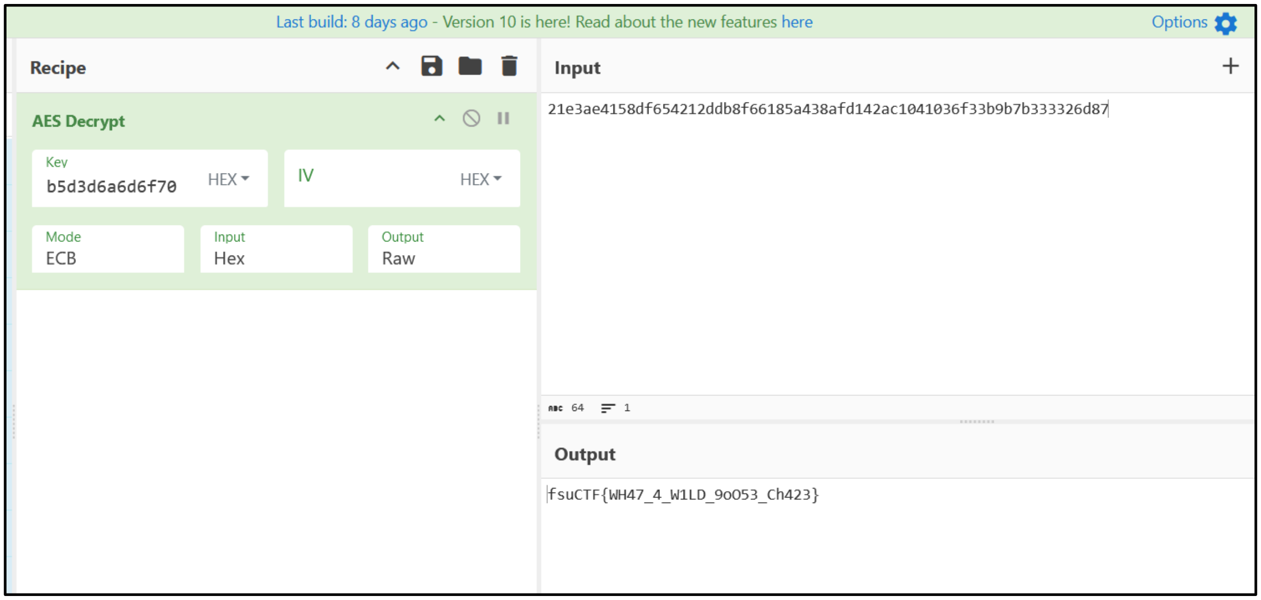
Input

```
MjFyM25yNDE1OHFzNjU0MjEycXFvOHM2NjE4NW40Mzhuc3ExNDJucDEwNDEwMzZzMzNvOW83bzMzMzMyNnE4Nw==
```

88 1

Output

```
21e3ae4158df654212ddb8f66185a438afd142ac1041036f33b9b7b333326d87
```



Last build: 8 days ago - Version 10 is here! Read about the new features here
Options

Recipe

AES Decrypt

Key b5d3d6a6d6f70 HEX

IV HEX

Mode ECB

Input Hex

Output Raw

Input

```
21e3ae4158df654212ddb8f66185a438afd142ac1041036f33b9b7b333326d87
```

64 1

Output

```
fsuCTF{WH47_4_W1LD_9oO53_Ch423}
```