Faculty of Engineering & Technology

Electrical & Computer Engineering Department

APPLIED CRYPTOGRAPHY- ENCS4320

Report for Project

Crypto Lab—Padding Oracle Attack

Prepared by:

Ahmad Zubaidia   1200105

Partners: there is no partners.

Instructor: Dr. Ahmad Alsadeh

Section: 1

Date:26/6/2023

ABSTRACT

The purpose of this experiment, was to study the details of cryptographic padding, especially PKCS#5, and its importance in protecting data using block ciphers. We developed a simulation intended to simulate actual encryption situations using Docker, a platform that makes it easier to create regulated and consistent environments. We started out by looking at the core concepts of PKCS#5 padding and its function in block cipher encryption. After that, we focused our attention to a serious flaw caused by incorrect implementation known as the Padding Oracle Attack. We manually carried out this attack using careful, step-by-step analysis, modifying padding data to decrypt encrypted information. The experiment's next step included speeding the procedure by creating an automated attack program. As a result, we were able to observe and evaluate the effectiveness and speed of a Padding Oracle Attack when it executed out automatically. In the experiment's final stages, the Padding Oracle Attack was used to enable the decoding of a fictitious secret message without the need for prior knowledge of the encryption key. This showed the potential consequences of cryptographic padding issues. This experiment highlighted the value of strong encryption techniques for preventing weaknesses while offering important insights into the area of cryptography.

Table of Contents

Table of figures

Introduction:

In today’s world, almost everything we do involves computers and the internet. From sending emails and sharing pictures to storing important documents, we rely heavily on digital technology. However, as we all know, the online world isn't always safe, and that's why we need ways to protect our information from prying eyes. This is where cryptography comes in.

Cryptography is like a digital lock and key system that helps keep our data safe. It’s a way to change the information into a secret code that only someone with the right key can understand. One of the ways cryptography does this is through something called block ciphers. Imagine you have a long message you want to secure. Block ciphers take this message and break it into smaller, equal-sized pieces, called blocks. These blocks are then scrambled up with a key so that no one can read them unless they have the key to unscramble them.

But what happens if the last piece of your message isn't big enough to make a full block? This is where PKCS#5 padding steps in. PKCS#5 padding adds extra bits to the last block so that it’s the right size. Think of it as adding extra stuffing to a package so it fits snugly in its box.

Now, you might think that with all this scrambling and padding, your message is safe. However, like any lock, there are people who try to pick it. In cryptography, one way they do this is through something called a Padding Oracle Attack. This is a sneaky way for a hacker to figure out the padding pattern used in the last block of the message, and eventually, decode the entire message without needing the key!

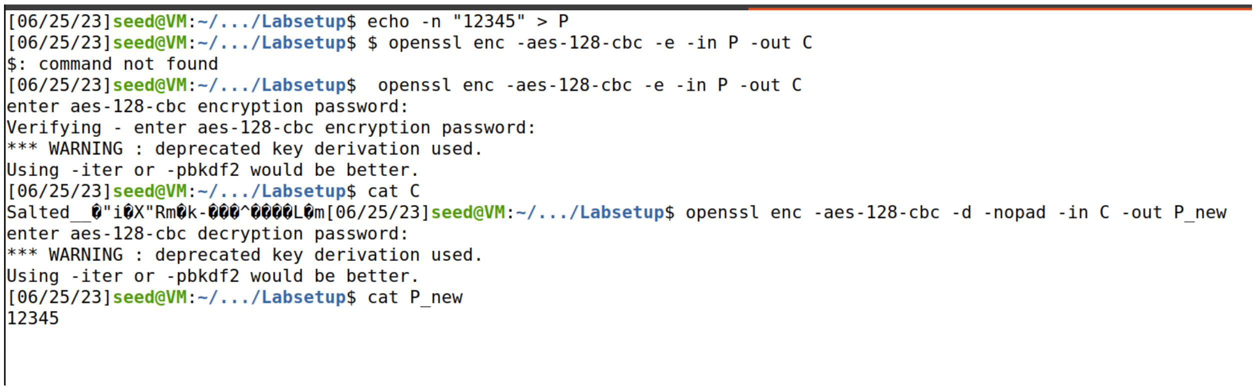
Understanding the details of how PKCS#5 padding works and how it can be exploited through the Padding Oracle Attack is important. It helps us build more secure systems and teaches us how to protect sensitive data.

In this experiment, we will take a close look at PKCS#5 padding, learn how it adds those extra bits to make the data just the right size, and how this method can sometimes be used against us through the Padding Oracle Attack. We will also try to understand how these attacks work and think about ways we can defend against them.

By diving deep into these topics, we're not just learning about codes and ciphers. We are contributing to a safer and more secure digital world for everyone.

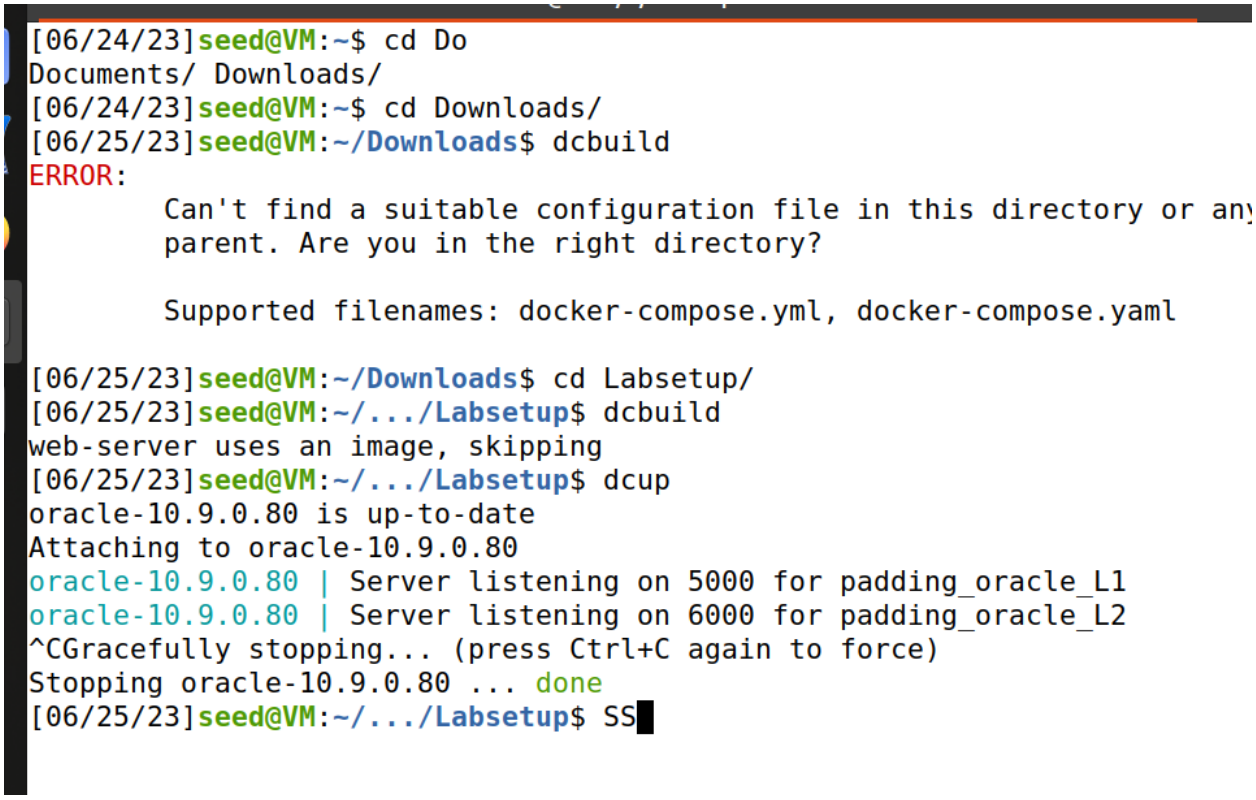
Procedure

Task 1



```
[06/25/23]seed@VM:~/.../Labsetup$ echo -n "12345" > P
[06/25/23]seed@VM:~/.../Labsetup$ $ openssl enc -aes-128-cbc -e -in P -out C
$: command not found
[06/25/23]seed@VM:~/.../Labsetup$  openssl enc -aes-128-cbc -e -in P -out C
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ cat C
Salted__�"i�X"Rm�k-���^����L�m[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -nopad -in C -out P_new
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ cat P_new
12345
```

Figure 1: makeing the first file and see padding



```
[06/24/23]seed@VM:~$ cd Do
Documents/ Downloads/
[06/24/23]seed@VM:~$ cd Downloads/
[06/25/23]seed@VM:~/Downloads$ dcbuild
ERROR:
        Can't find a suitable configuration file in this directory or any
        parent. Are you in the right directory?

        Supported filenames: docker-compose.yml, docker-compose.yaml

[06/25/23]seed@VM:~/Downloads$ cd Labsetup/
[06/25/23]seed@VM:~/.../Labsetup$ dcbuild
web-server uses an image, skipping
[06/25/23]seed@VM:~/.../Labsetup$ dcup
oracle-10.9.0.80 is up-to-date
Attaching to oracle-10.9.0.80
oracle-10.9.0.80 | Server listening on 5000 for padding_oracle_L1
oracle-10.9.0.80 | Server listening on 6000 for padding_oracle_L2
^CGracefully stopping... (press Ctrl+C again to force)
Stopping oracle-10.9.0.80 ... done
[06/25/23]seed@VM:~/.../Labsetup$ SS
```

Figure 2: setting up the servers

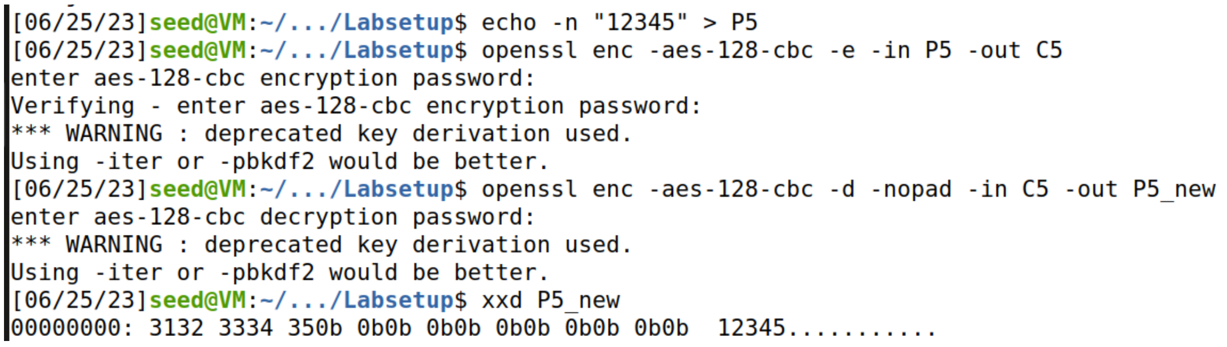

```
[06/25/23]seed@VM:~/.../Labsetup$ echo -n "12345" > P5
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in P5 -out C5
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -nopad -in C5 -out P5_new
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ xxd P5_new
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
```

Figure 3 : task 1 , 5 bytes

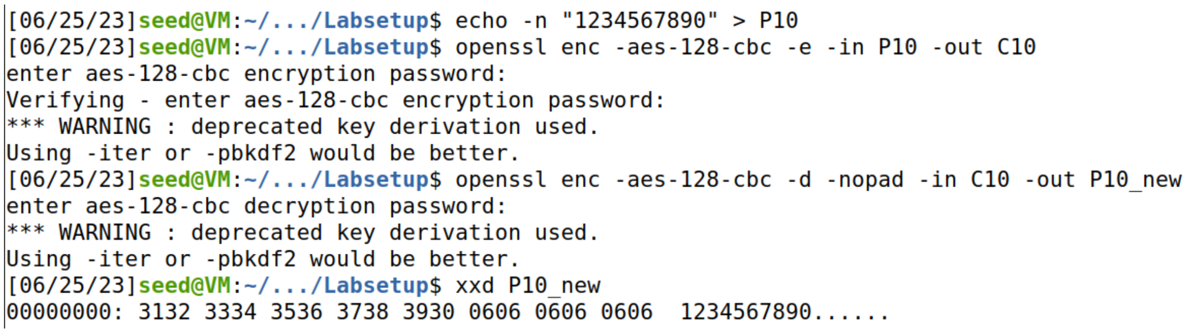

```
[06/25/23]seed@VM:~/.../Labsetup$ echo -n "1234567890" > P10
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in P10 -out C10
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -nopad -in C10 -out P10_new
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ xxd P10_new
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
```

Figure 4:task 1 , 10 bytes

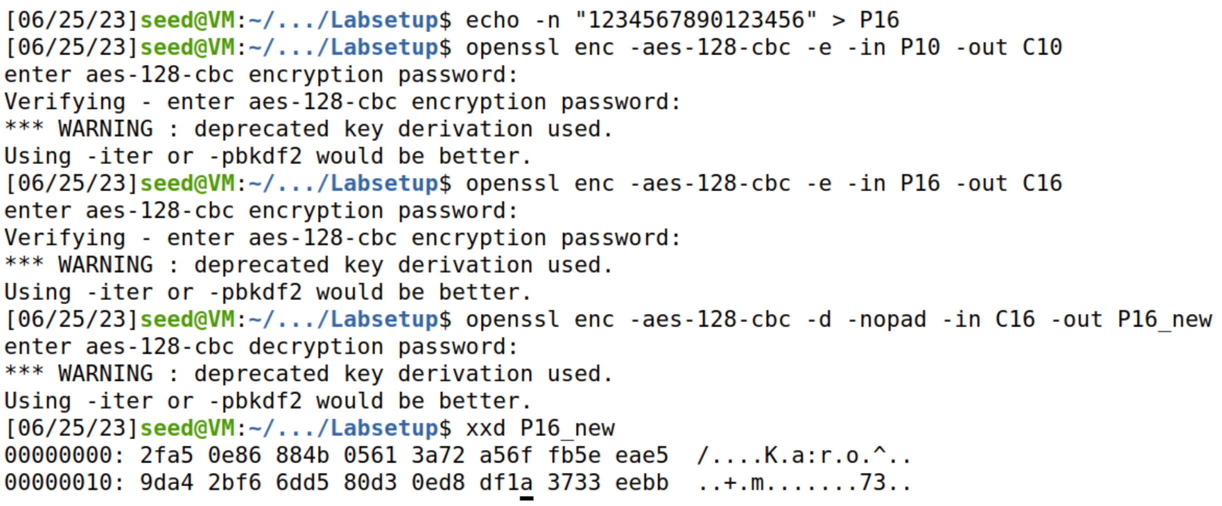

```
[06/25/23]seed@VM:~/.../Labsetup$ echo -n "1234567890123456" > P16
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in P10 -out C10
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in P16 -out C16
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -nopad -in C16 -out P16_new
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[06/25/23]seed@VM:~/.../Labsetup$ xxd P16_new
00000000: 2fa5 0e86 884b 0561 3a72 a56f fb5e eae5  /....K.a:r.o.^..
00000010: 9da4 2bf6 6dd5 80d3 0ed8 df1a 3733 eebb  ..+.m.......73..
```

Figure 5: task1 , 16 bit

What Was Done:
The objective of Task 1 was to become familiar with how padding, specifically PKCS#5 padding, functions in block ciphers. For block ciphers, padding may be required when the size of the plaintext is not a multiple of the block size. In this task, three files were created with sizes of 5 bytes, 10 bytes, and 16 bytes, respectively. The OpenSSL command-line tool was utilized for encrypting and decrypting these files with 128-bit AES in CBC mode. This was done to observe what data was added during the padding process.

The following **commands were executed:**

Created a file with 5 bytes: **echo -n "12345" > P**

Encrypted the file: **openssl enc -aes-128-cbc -e -in P -out C**

Decrypted the file without removing the padding: **openssl enc -aes-128-cbc -d -nopad -in C -out P_new**

Displayed the content in hex format: **xxd P_new**

These steps were repeated for files containing 10 bytes and 16 bytes.

Observations:
Upon decrypting and examining the contents of the file **P_new** in hex format, it was noted that additional bytes were appended to the original data. The padded data was in a non-printable form. For the file with 5 bytes, the padding was 11 bytes long, for the file with 10 bytes, the padding was 6 bytes long, and for the file with 16 bytes, no padding was added.

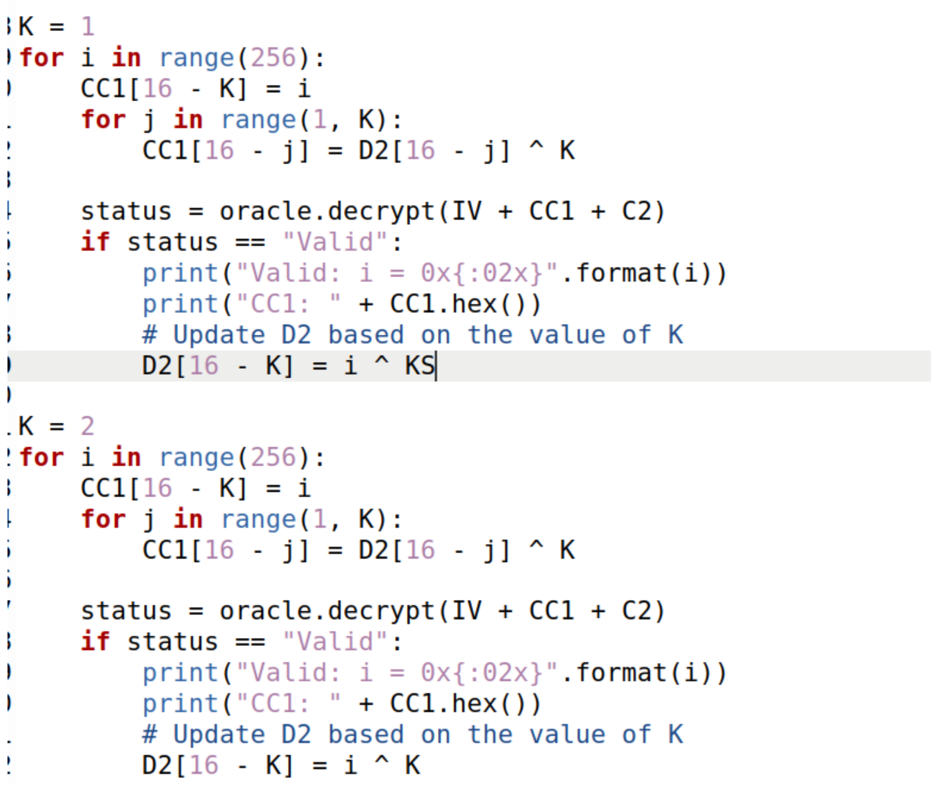
Explanation of Observations:
The PKCS#5 padding scheme works by filling up the block with bytes, all of which represent the number of bytes added. For instance, in the case of the file with 5 bytes, 11 bytes of padding were added, each having the value 0x0B (which is 11 in hexadecimal). This makes the total block size 16 bytes, which is a multiple of the AES block size.

For the file with 10 bytes, 6 bytes of padding were added, each with the value 0x06.

Interestingly, for the file with exactly 16 bytes, no padding was added. This is because the data already fits perfectly into the block size required by AES.

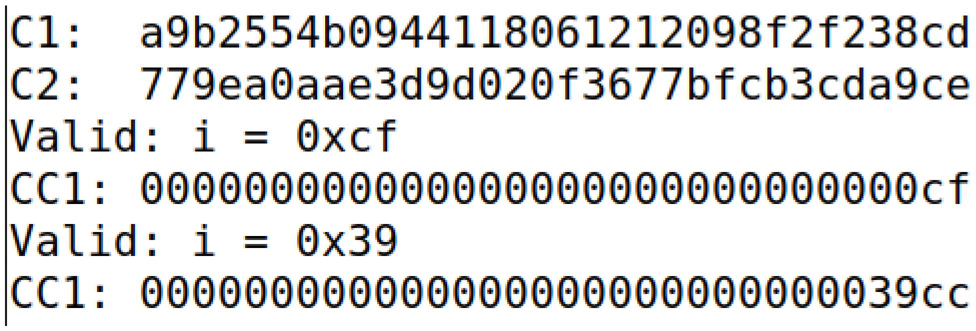
Understanding how padding works is essential as it has implications for security and the proper functioning of cryptographic systems.

Task 2



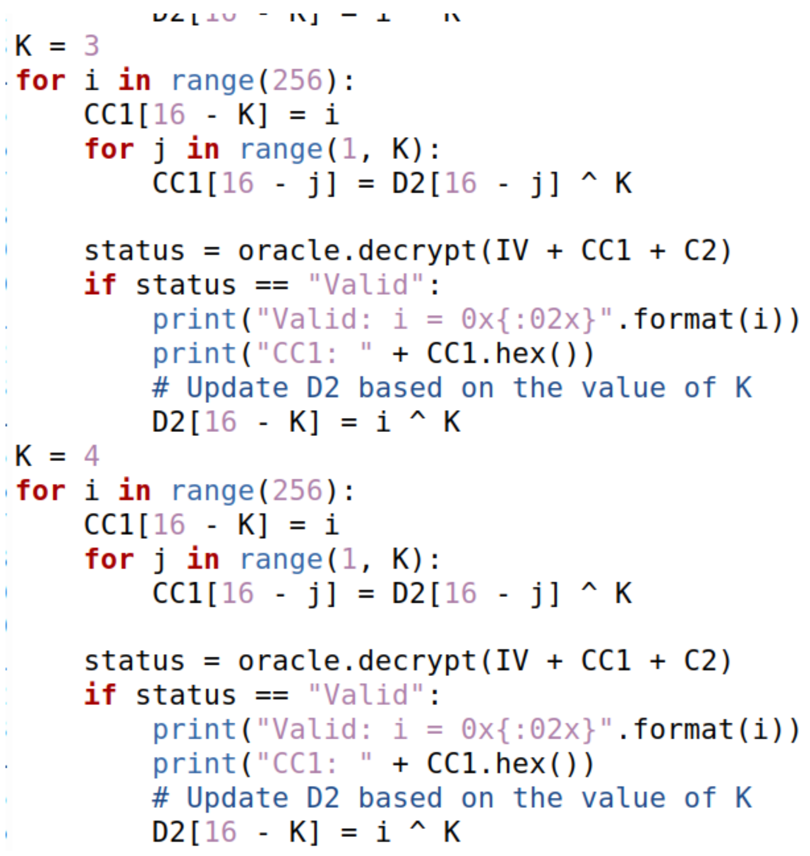
```
K = 1
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ KS

K = 2
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ K
```

Figure 6: iteration ,K=1, D[15]. , K=2 ,D[14].



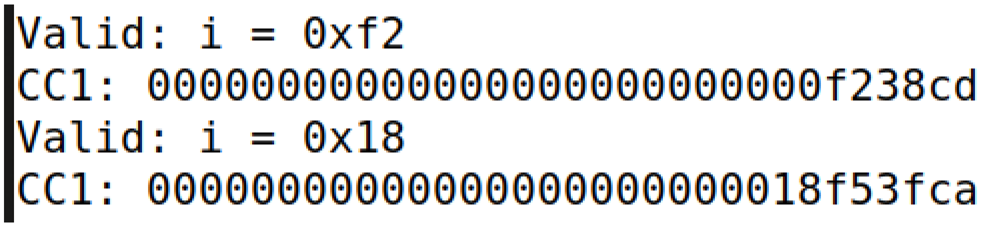
```
C1:  a9b2554b0944118061212098f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xcf
CC1: 000000000000000000000000000000cf
Valid: i = 0x39
CC1: 0000000000000000000000000000039cc
```

Figure 7: the values which makes the padding valid K=1,K=2.



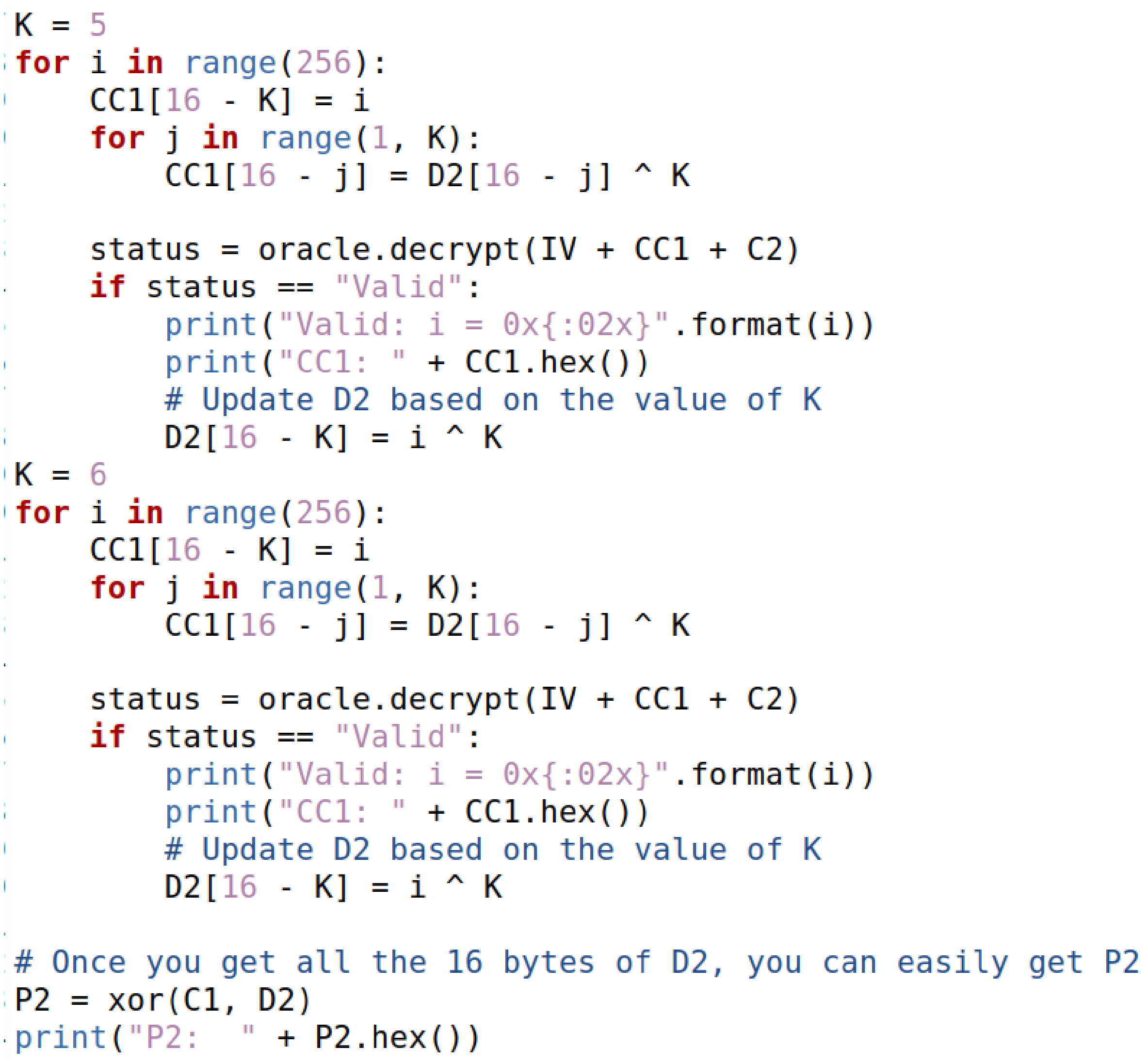
```
        D2[16 - K] = i ^ K
K = 3
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ K
K = 4
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ K
```

Figure 8: Bytes ,K=3, D[13]. , K=4 ,D[12].



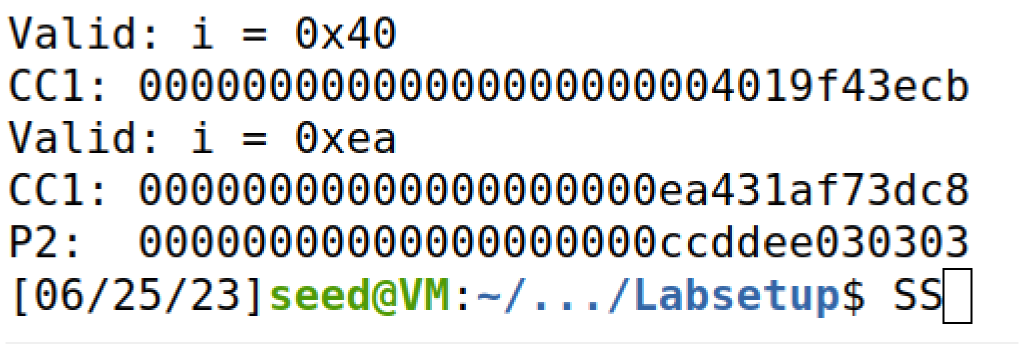
```
Valid: i = 0xf2
CC1: 00000000000000000000000000f238cd
Valid: i = 0x18
CC1: 00000000000000000000000018f53fca
```

Figure 9:the values which makes the padding valid K=3,K=4



```
K = 5
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ K
K = 6
for i in range(256):
    CC1[16 - K] = i
    for j in range(1, K):
        CC1[16 - j] = D2[16 - j] ^ K

    status = oracle.decrypt(IV + CC1 + C2)
    if status == "Valid":
        print("Valid: i = 0x{:02x}".format(i))
        print("CC1: " + CC1.hex())
        # Update D2 based on the value of K
        D2[16 - K] = i ^ K

# Once you get all the 16 bytes of D2, you can easily get P2
P2 = xor(C1, D2)
print("P2:  " + P2.hex())
```

Figure 10: Bytes ,K=5, D[11]. , K=6 ,D[10].



```
Valid: i = 0x40
CC1: 00000000000000000000004019f43ecb
Valid: i = 0xea
CC1: 000000000000000000000ea431af73dc8
P2:  000000000000000000000ccddee030303
[06/25/23]seed@VM:~/.../Labsetup$ SS
```

Figure 11: the values which makes the padding valid K=5,K=6 & P(plain text for 6 Bytes)

What Was Done:
Task 2 focused on performing a padding oracle attack. Some systems, during decryption, verify the validity of the padding and return an error if the padding is invalid. This behavior can be exploited through a padding oracle attack. The goal of Task 2 was to decipher a secret message by exploiting the padding oracle without knowing the encryption key. AES-CBC was used as the encryption algorithm.

The padding oracle was set up on port 5000 and provided a ciphertext of a secret message encrypted with an unknown key K. The ciphertext consisted of an IV (initialization vector) and the encrypted message, split into blocks.

The ciphertext was fetched from the padding oracle.

Two 16-byte arrays, C1 and C2, were used to store the content of the two blocks of the ciphertext.

The padding oracle was then interacted with by sending modified versions of C1 (referred to as CC1) along with C2 and observing whether the padding was valid or not.

Through an iterative process, each byte of an array D2 was deciphered. D2 represents the output of the block cipher before XORing with the previous ciphertext block (or IV for the first block).

Upon figuring out D2, the plaintext P2 was calculated as **P2 = C1 ⊕ D2**.

Observations:
Through multiple iterations and trying all possible values for each byte, one byte at a time, it was possible to derive values for D2 such that the padding was valid. By continuously modifying CC1 and observing the oracle's responses, the entire block D2 was eventually derived.

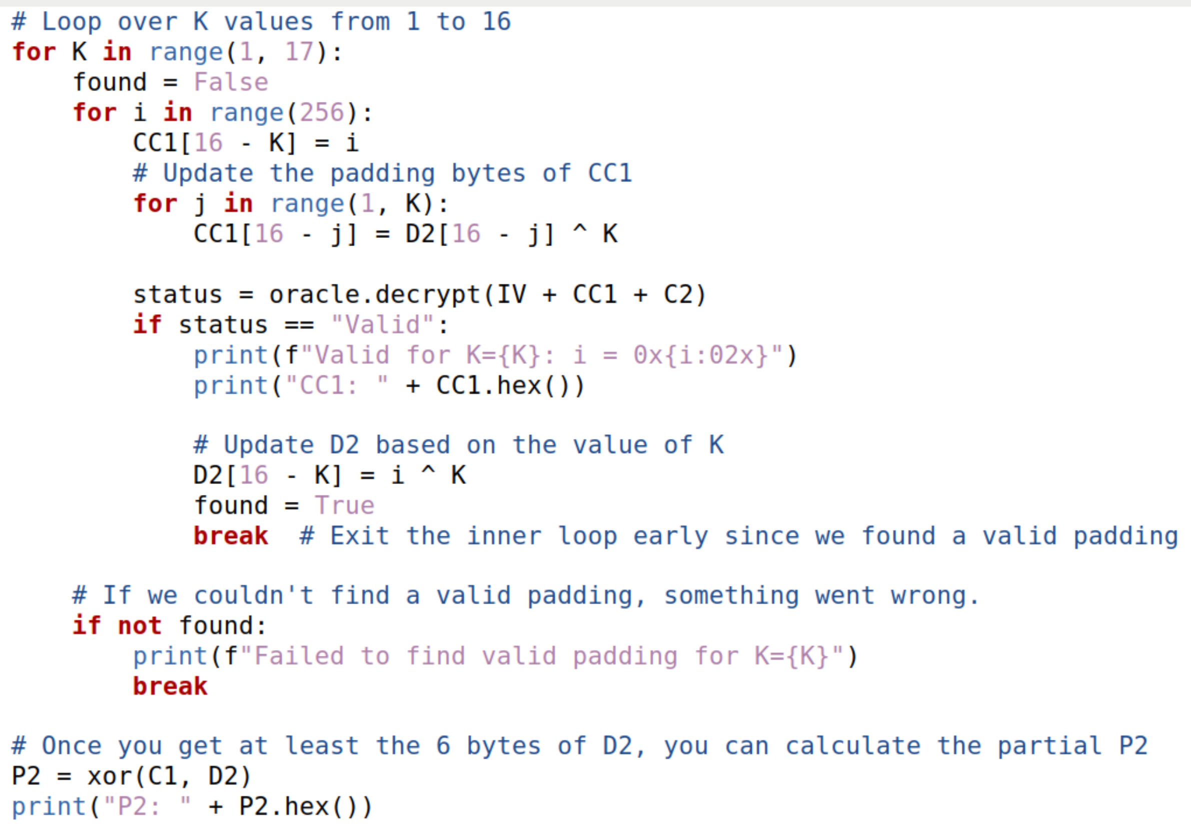
Explanation of Observations:
The padding oracle attack exploits the fact that an attacker can learn whether the padding of a modified ciphertext is valid or not. By carefully choosing the modifications and observing the responses, the attacker can make educated guesses about the plaintext.

In this task, the focus was on decrypting the second block of the ciphertext. To decrypt the second block, values for D2 were needed, which could then be XORed with C1 to obtain the plaintext. By iteratively changing the bytes in CC1 and sending it with C2 to the oracle, information about the actual D2 was gathered. This iterative process was done for each byte of D2, and as the values were derived, they were used in subsequent rounds to derive the next byte.

The process was manual, and at least six bytes of D2 needed to be deciphered for the task. Once the required number of bytes for D2 was obtained, P2 could be calculated.

This task demonstrated how a seemingly harmless behavior of checking for valid padding can be exploited to derive sensitive information. It highlights the importance of careful implementation and consideration of cryptographic systems.

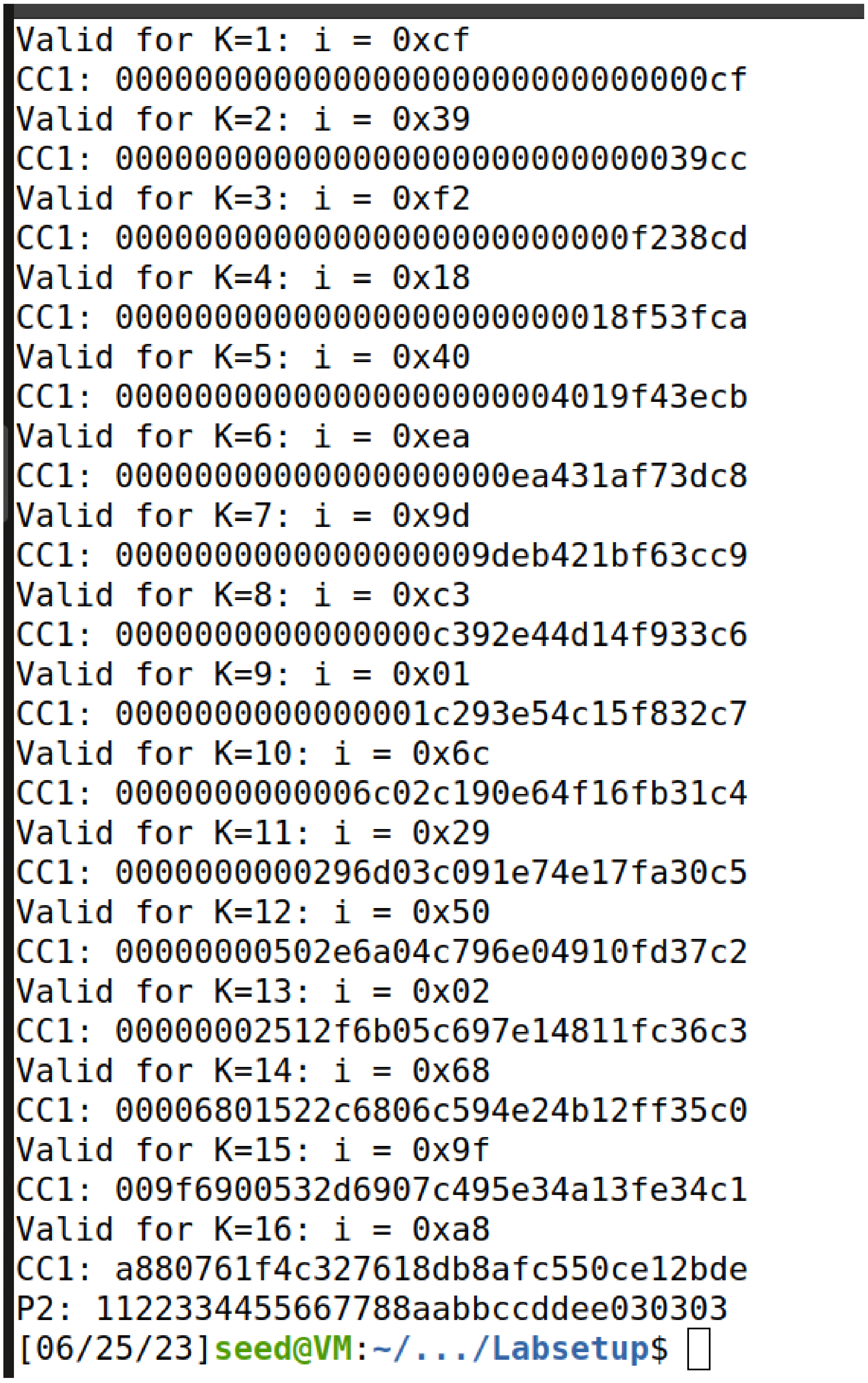
Task 3



```python
# Loop over K values from 1 to 16
for K in range(1, 17):
    found = False
    for i in range(256):
        CC1[16 - K] = i
        # Update the padding bytes of CC1
        for j in range(1, K):
            CC1[16 - j] = D2[16 - j] ^ K

        status = oracle.decrypt(IV + CC1 + C2)
        if status == "Valid":
            print(f"Valid for K={K}: i = 0x{i:02x}")
            print("CC1: " + CC1.hex())

            # Update D2 based on the value of K
            D2[16 - K] = i ^ K
            found = True
            break  # Exit the inner loop early since we found a valid padding

    # If we couldn't find a valid padding, something went wrong.
    if not found:
        print(f"Failed to find valid padding for K={K}")
        break

# Once you get at least the 6 bytes of D2, you can calculate the partial P2
P2 = xor(C1, D2)
print("P2: " + P2.hex())
```

Figure 12: Loop over K values from 1 to 16

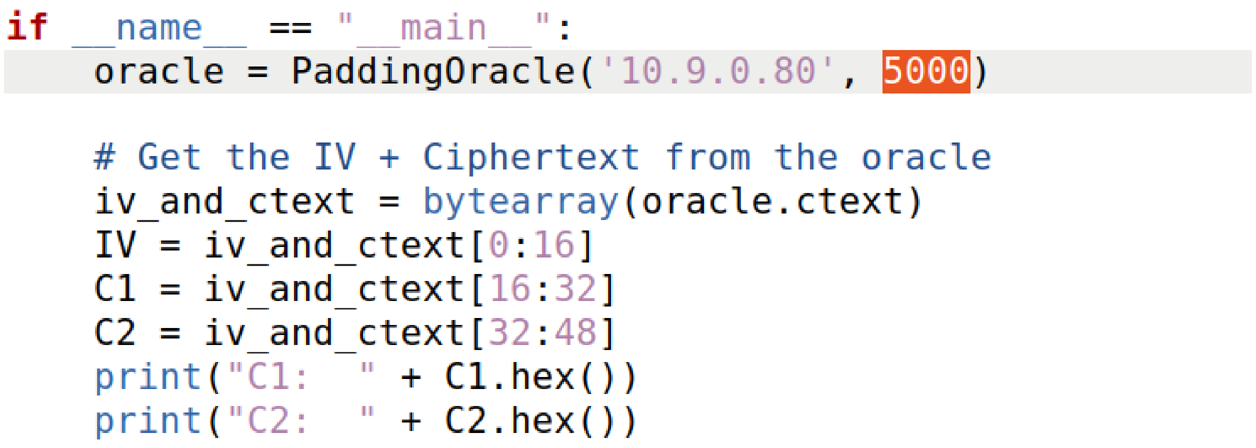
Just the reason of this section , to make the previous process, automatic , by using for loops.



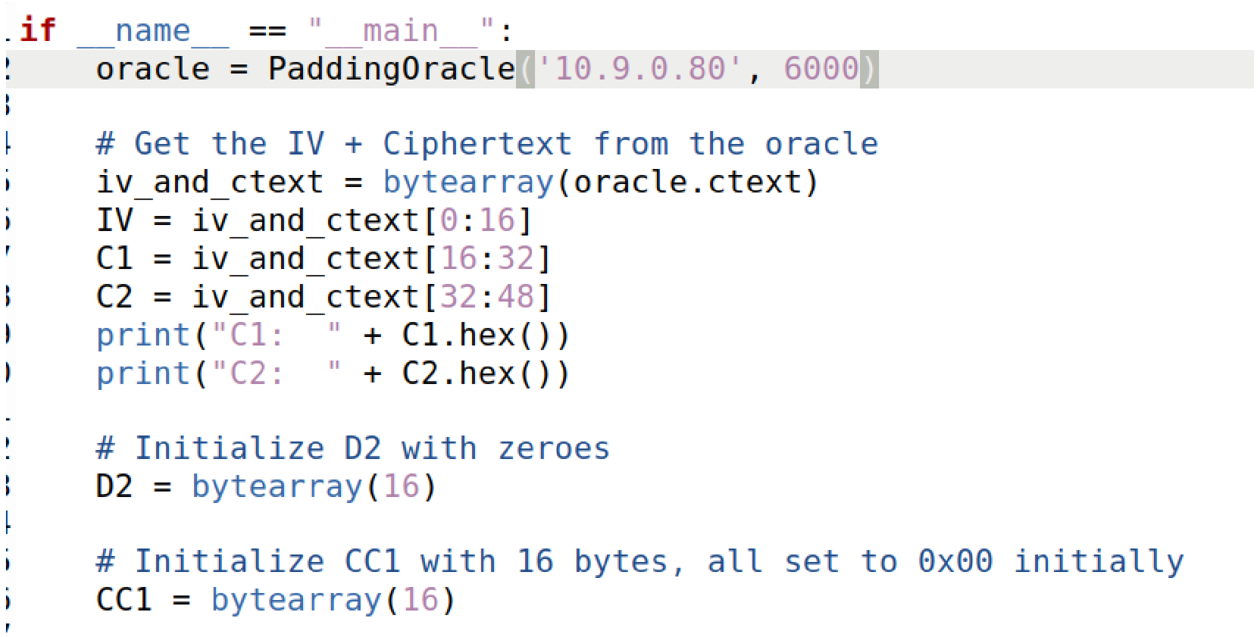
```
Valid for K=1: i = 0xcf
CC1: 000000000000000000000000000000cf
Valid for K=2: i = 0x39
CC1: 0000000000000000000000000000039cc
Valid for K=3: i = 0xf2
CC1: 00000000000000000000000000f238cd
Valid for K=4: i = 0x18
CC1: 00000000000000000000000018f53fca
Valid for K=5: i = 0x40
CC1: 0000000000000000000004019f43ecb
Valid for K=6: i = 0xea
CC1: 00000000000000000000ea431af73dc8
Valid for K=7: i = 0x9d
CC1: 000000000000000009deb421bf63cc9
Valid for K=8: i = 0xc3
CC1: 0000000000000000c392e44d14f933c6
Valid for K=9: i = 0x01
CC1: 000000000000001c293e54c15f832c7
Valid for K=10: i = 0x6c
CC1: 000000000006c02c190e64f16fb31c4
Valid for K=11: i = 0x29
CC1: 0000000000296d03c091e74e17fa30c5
Valid for K=12: i = 0x50
CC1: 00000000502e6a04c796e04910fd37c2
Valid for K=13: i = 0x02
CC1: 00000002512f6b05c697e14811fc36c3
Valid for K=14: i = 0x68
CC1: 00006801522c6806c594e24b12ff35c0
Valid for K=15: i = 0x9f
CC1: 009f6900532d6907c495e34a13fe34c1
Valid for K=16: i = 0xa8
CC1: a880761f4c327618db8afc550ce12bde
P2: 1122334455667788aabbccddee030303
[06/25/23]seed@VM:~/.../Labsetup$ 
```

Figure 13: the values of valid padding.

Using port 6000

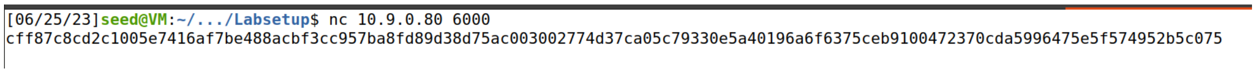


```python
if __name__ == "__main__":
    oracle = PaddingOracle('10.9.0.80', 5000)

    # Get the IV + Ciphertext from the oracle
    iv_and_ctext = bytearray(oracle.ctext)
    IV = iv_and_ctext[0:16]
    C1 = iv_and_ctext[16:32]
    C2 = iv_and_ctext[32:48]
    print("C1:  " + C1.hex())
    print("C2:  " + C2.hex())
```



```python
if __name__ == "__main__":
    oracle = PaddingOracle('10.9.0.80', 6000)

    # Get the IV + Ciphertext from the oracle
    iv_and_ctext = bytearray(oracle.ctext)
    IV = iv_and_ctext[0:16]
    C1 = iv_and_ctext[16:32]
    C2 = iv_and_ctext[32:48]
    print("C1:  " + C1.hex())
    print("C2:  " + C2.hex())

    # Initialize D2 with zeroes
    D2 = bytearray(16)

    # Initialize CC1 with 16 bytes, all set to 0x00 initially
    CC1 = bytearray(16)
```

Figure 14:The Level-2 server listens to port 6000



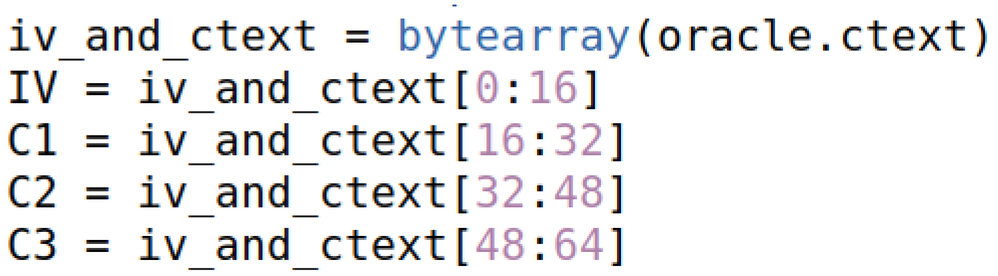
```
[06/25/23]seed@VM:~/.../Labsetup$ nc 10.9.0.80 6000
cff87c8cd2c1005e7416af7be488acbf3cc957ba8fd89d38d75ac003002774d37ca05c79330e5a40196a6f6375ceb9100472370cda5996475e5f574952b5c075
```

Figure 15: Cypher from port 6000

After seeing the cypher form port 6000 , we conclude some points

For this port , we got cypher which equal to 98 character which equal to 48 Byte means 3 Blocks +IV .

16 IV + 16 block 1 , 16 block 2 , 16 block 3.

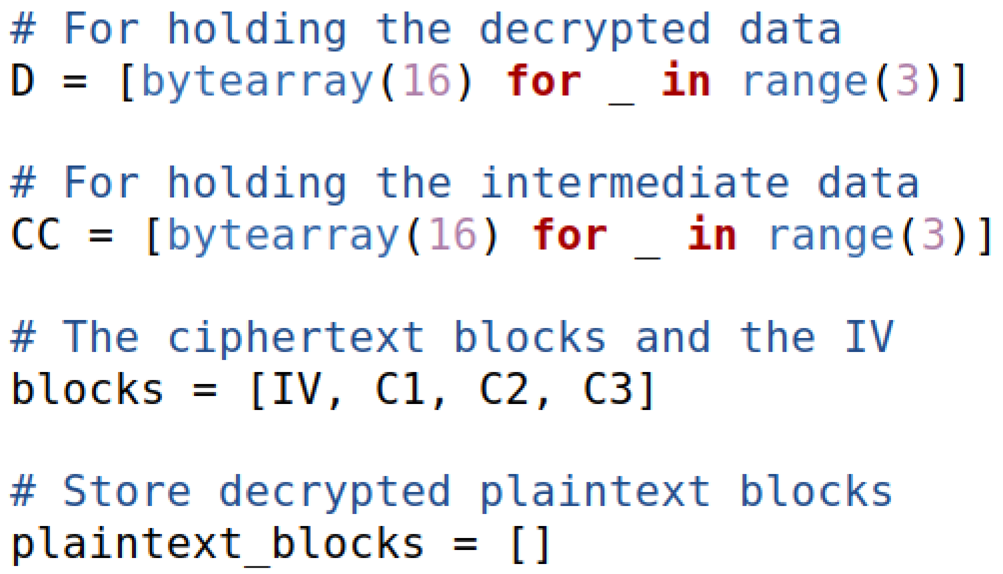


```
iv_and_ctext = bytearray(oracle.ctext)
IV = iv_and_ctext[0:16]
C1 = iv_and_ctext[16:32]
C2 = iv_and_ctext[32:48]
C3 = iv_and_ctext[48:64]
```

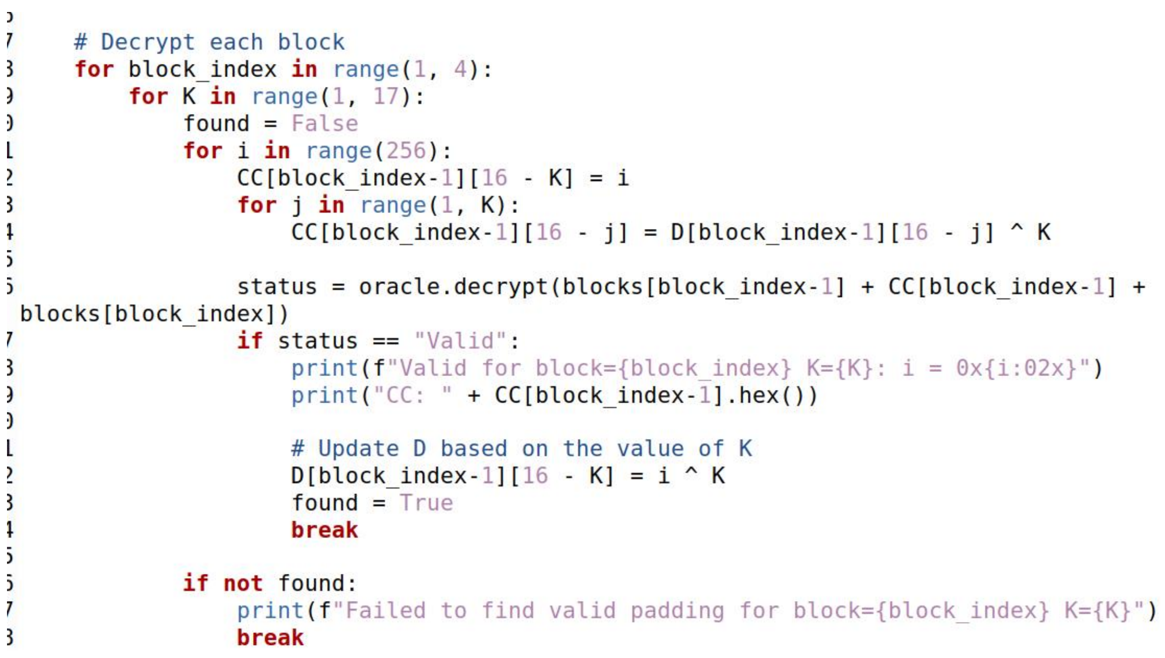
Figure 16: Cypher Blocks with IV

D1, D2 ,D3 for holding the decrypted data

And CC1, CC2, CC3 for holding the intermediate data

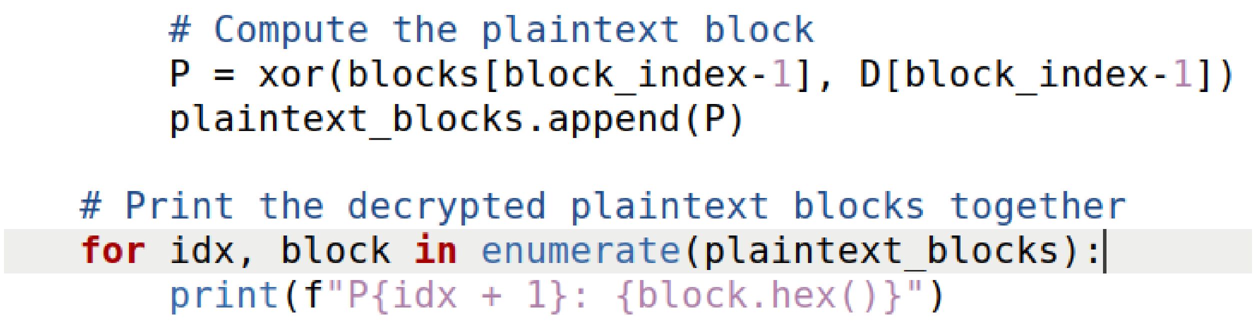


```
# For holding the decrypted data
D = [bytearray(16) for _ in range(3)]

# For holding the intermediate data
CC = [bytearray(16) for _ in range(3)]

# The ciphertext blocks and the IV
blocks = [IV, C1, C2, C3]

# Store decrypted plaintext blocks
plaintext_blocks = []
```



```python
    # Decrypt each block
    for block_index in range(1, 4):
        for K in range(1, 17):
            found = False
            for i in range(256):
                CC[block_index-1][16 - K] = i
                for j in range(1, K):
                    CC[block_index-1][16 - j] = D[block_index-1][16 - j] ^ K

                status = oracle.decrypt(blocks[block_index-1] + CC[block_index-1] +
blocks[block_index])
                if status == "Valid":
                    print(f"Valid for block={block_index} K={K}: i = 0x{i:02x}")
                    print("CC: " + CC[block_index-1].hex())

                    # Update D based on the value of K
                    D[block_index-1][16 - K] = i ^ K
                    found = True
                    break

            if not found:
                print(f"Failed to find valid padding for block={block_index} K={K}")
                break
```

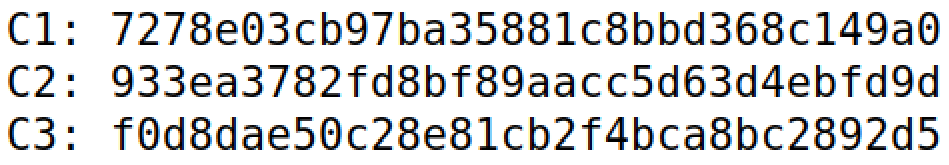
Firs loop for blocks , second for K (Bytes from 1-16) , the third one for itreation to find the vaild padding , the last one for updating .



```python
        # Compute the plaintext block
        P = xor(blocks[block_index-1], D[block_index-1])
        plaintext_blocks.append(P)

    # Print the decrypted plaintext blocks together
    for idx, block in enumerate(plaintext_blocks):
        print(f"P{idx + 1}: {block.hex()}")
```

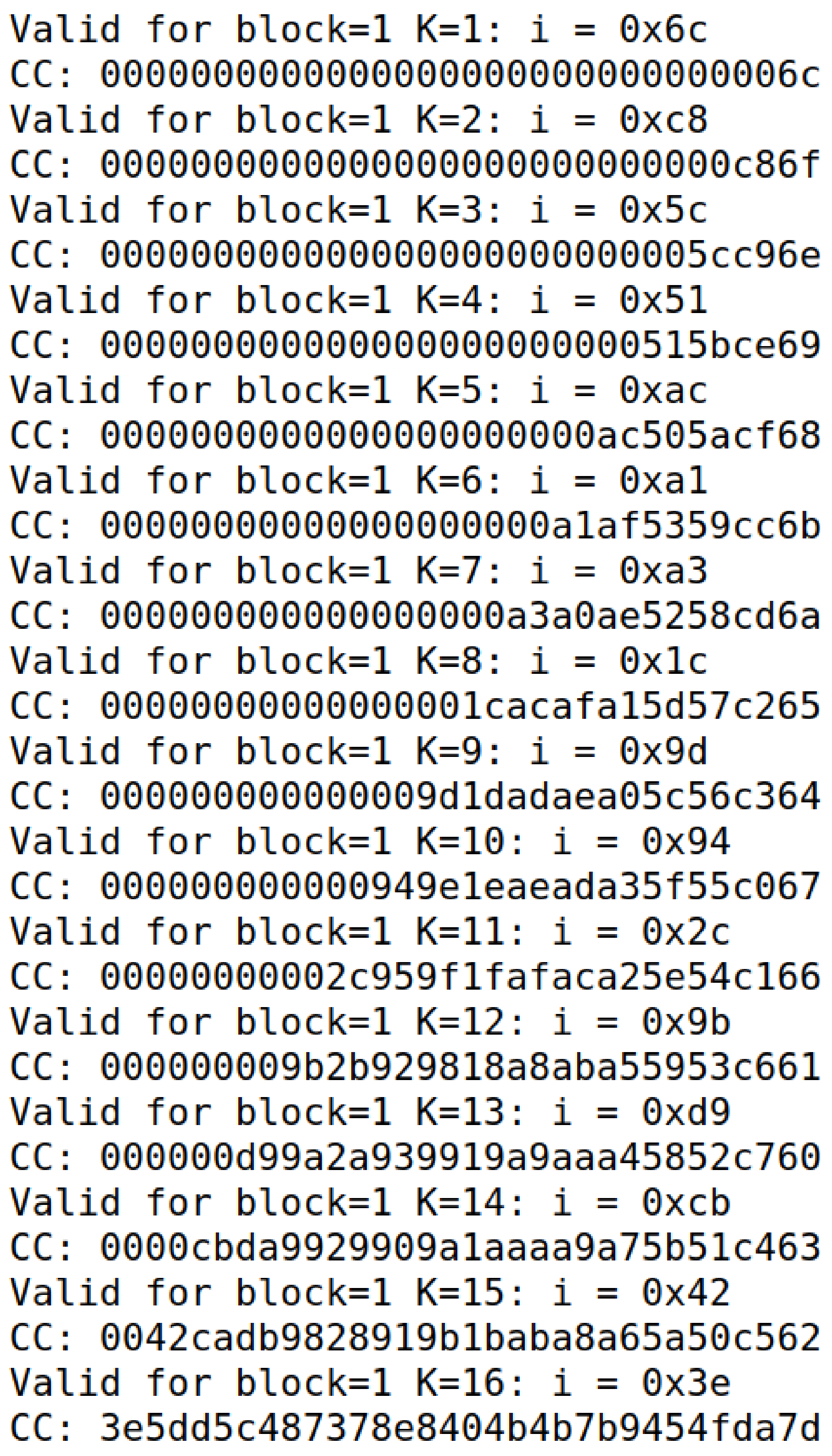
Figure 17: ***Automated attack process for three blocks.***

The output:



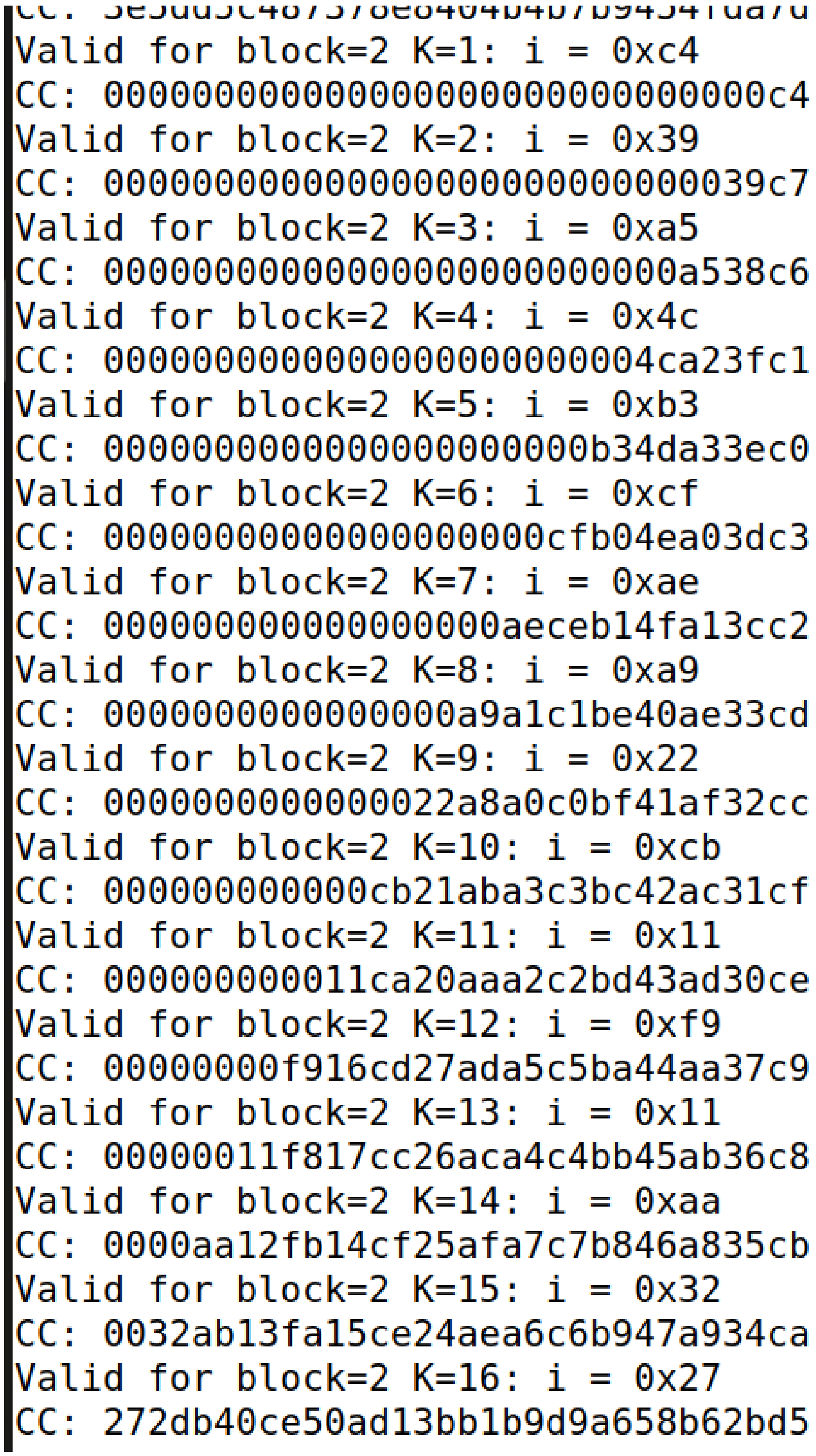
```
C1: 7278e03cb97ba35881c8bbd368c149a0
C2: 933ea3782fd8bf89aacc5d63d4ebfd9d
C3: f0d8dae50c28e81cb2f4bca8bc2892d5
```

Figure 18: three blocks' cyphers



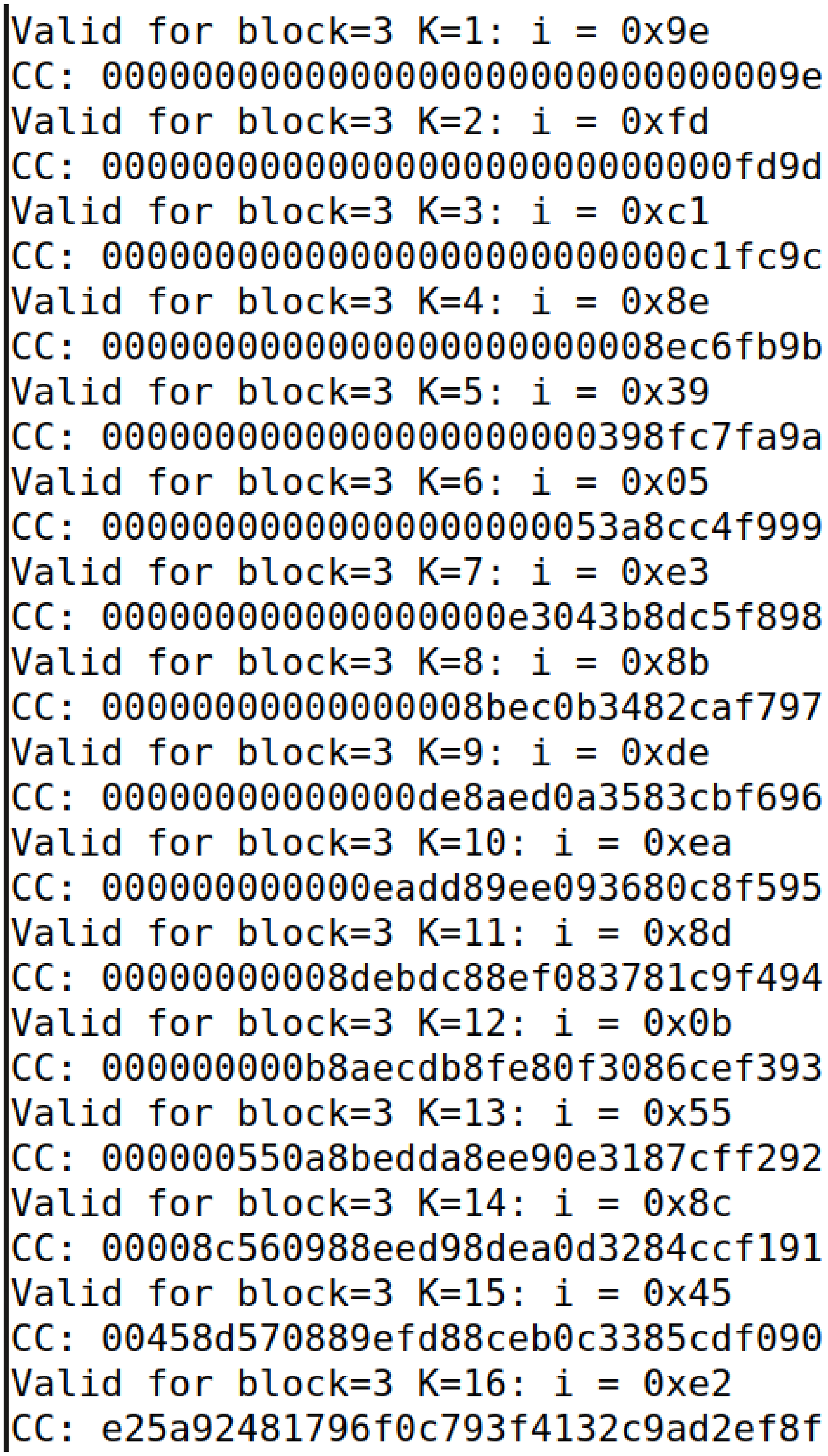
```
Valid for block=1 K=1: i = 0x6c
CC: 0000000000000000000000000000006c
Valid for block=1 K=2: i = 0xc8
CC: 000000000000000000000000000c86f
Valid for block=1 K=3: i = 0x5c
CC: 00000000000000000000000005cc96e
Valid for block=1 K=4: i = 0x51
CC: 000000000000000000000000515bce69
Valid for block=1 K=5: i = 0xac
CC: 0000000000000000000000ac505acf68
Valid for block=1 K=6: i = 0xa1
CC: 00000000000000000000a1af5359cc6b
Valid for block=1 K=7: i = 0xa3
CC: 000000000000000000a3a0ae5258cd6a
Valid for block=1 K=8: i = 0x1c
CC: 00000000000000001cacafa15d57c265
Valid for block=1 K=9: i = 0x9d
CC: 000000000000009d1dadaea05c56c364
Valid for block=1 K=10: i = 0x94
CC: 000000000000949e1eaeada35f55c067
Valid for block=1 K=11: i = 0x2c
CC: 00000000002c959f1fafaca25e54c166
Valid for block=1 K=12: i = 0x9b
CC: 000000009b2b929818a8aba55953c661
Valid for block=1 K=13: i = 0xd9
CC: 000000d99a2a939919a9aaa45852c760
Valid for block=1 K=14: i = 0xcb
CC: 0000cbda9929909a1aaaa9a75b51c463
Valid for block=1 K=15: i = 0x42
CC: 0042cadb9828919b1baba8a65a50c562
Valid for block=1 K=16: i = 0x3e
CC: 3e5dd5c487378e8404b4b7b9454fda7d
```

Figure 19: for Block one Valid padding values



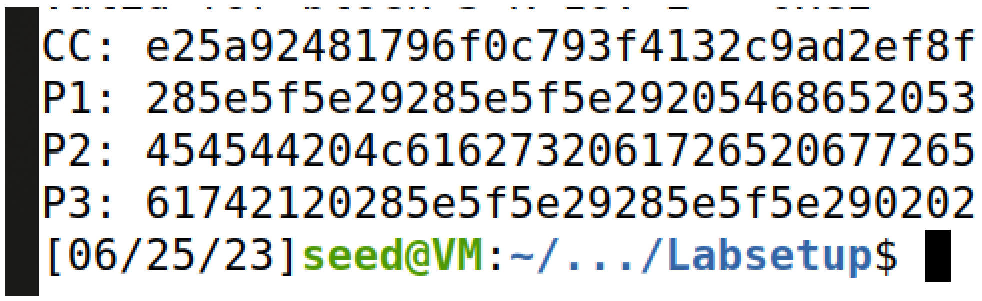
```
CC: 3e5dd5c487378e8404b4b7b9454fda7d
Valid for block=2 K=1: i = 0xc4
CC: 000000000000000000000000000000c4
Valid for block=2 K=2: i = 0x39
CC: 000000000000000000000000000039c7
Valid for block=2 K=3: i = 0xa5
CC: 0000000000000000000000000000a538c6
Valid for block=2 K=4: i = 0x4c
CC: 000000000000000000000004ca23fc1
Valid for block=2 K=5: i = 0xb3
CC: 0000000000000000000000b34da33ec0
Valid for block=2 K=6: i = 0xcf
CC: 00000000000000000000cfb04ea03dc3
Valid for block=2 K=7: i = 0xae
CC: 000000000000000000aeceb14fa13cc2
Valid for block=2 K=8: i = 0xa9
CC: 0000000000000000a9a1c1be40ae33cd
Valid for block=2 K=9: i = 0x22
CC: 00000000000000022a8a0c0bf41af32cc
Valid for block=2 K=10: i = 0xcb
CC: 000000000000cb21aba3c3bc42ac31cf
Valid for block=2 K=11: i = 0x11
CC: 000000000011ca20aaa2c2bd43ad30ce
Valid for block=2 K=12: i = 0xf9
CC: 00000000f916cd27ada5c5ba44aa37c9
Valid for block=2 K=13: i = 0x11
CC: 00000011f817cc26aca4c4bb45ab36c8
Valid for block=2 K=14: i = 0xaa
CC: 0000aa12fb14cf25afa7c7b846a835cb
Valid for block=2 K=15: i = 0x32
CC: 0032ab13fa15ce24aea6c6b947a934ca
Valid for block=2 K=16: i = 0x27
CC: 272db40ce50ad13bb1b9d9a658b62bd5
```

Figure 20: For Block Two



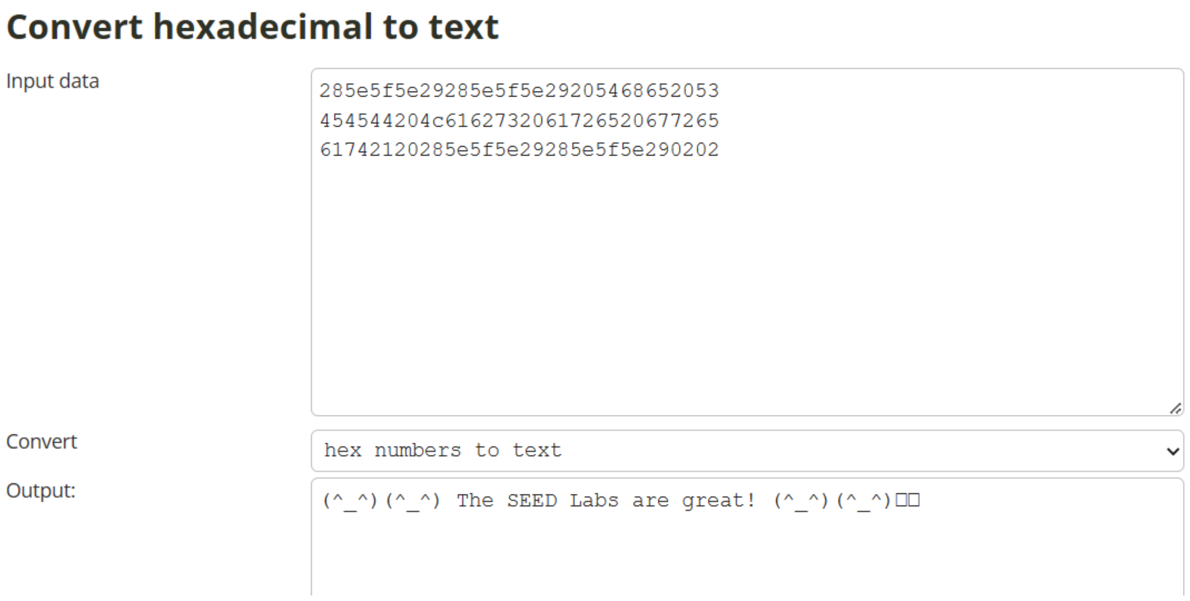
```
Valid for block=3 K=1: i = 0x9e
CC: 0000000000000000000000000000009e
Valid for block=3 K=2: i = 0xfd
CC: 000000000000000000000000000fd9d
Valid for block=3 K=3: i = 0xc1
CC: 00000000000000000000000000c1fc9c
Valid for block=3 K=4: i = 0x8e
CC: 000000000000000000000008ec6fb9b
Valid for block=3 K=5: i = 0x39
CC: 000000000000000000000398fc7fa9a
Valid for block=3 K=6: i = 0x05
CC: 0000000000000000000053a8cc4f999
Valid for block=3 K=7: i = 0xe3
CC: 000000000000000000e3043b8dc5f898
Valid for block=3 K=8: i = 0x8b
CC: 00000000000000008bec0b3482caf797
Valid for block=3 K=9: i = 0xde
CC: 00000000000000de8aed0a3583cbf696
Valid for block=3 K=10: i = 0xea
CC: 000000000000eadd89ee093680c8f595
Valid for block=3 K=11: i = 0x8d
CC: 00000000008debdc88ef083781c9f494
Valid for block=3 K=12: i = 0x0b
CC: 000000000b8aecdb8fe80f3086cef393
Valid for block=3 K=13: i = 0x55
CC: 000000550a8bedda8ee90e3187cff292
Valid for block=3 K=14: i = 0x8c
CC: 00008c560988eed98dea0d3284ccf191
Valid for block=3 K=15: i = 0x45
CC: 00458d570889efd88ceb0c3385cdf090
Valid for block=3 K=16: i = 0xe2
CC: e25a92481796f0c793f4132c9ad2ef8f
```

Figure 21: For Block Three



```
CC: e25a92481796f0c793f4132c9ad2ef8f
P1: 285e5f5e29285e5f5e29205468652053
P2: 454544204c6162732061726520677265
P3: 61742120285e5f5e29285e5f5e290202
[06/25/23]seed@VM:~/.../Labsetup$
```

Figure 22: the plain text of three blocks in hex



Convert hexadecimal to text

Input data

```
285e5f5e29285e5f5e29205468652053
454544204c6162732061726520677265
61742120285e5f5e29285e5f5e290202
```

Convert: hex numbers to text

Output:

```
(^_^)(^_^) The SEED Labs are great! (^_^)(^_^)□□
```

Figure 23: text of plain text

What Was Done:
Task 3 involved automating the padding oracle attack process and extending it to decrypt all blocks of the secret message. The objective was to derive the entire plaintext by exploiting the padding oracle without prior knowledge of the encryption key. AES-CBC was again used as the encryption algorithm. A padding oracle server for Level 2 was set up on port 6000, allowing interaction for the automated attack. The server responded to queries with a ciphertext containing an IV and encrypted message blocks. The process began by fetching the ciphertext from the padding oracle. Four 16-byte arrays, including an IV and three ciphertext blocks (C1, C2, and C3), were created to store the respective components of the ciphertext. To perform the automated attack, arrays were initialized to hold the decrypted data (D) and intermediate data (CC). The blocks of the ciphertext and the IV were assigned to corresponding variables. The attack involved decrypting each block iteratively. For each block, an iterative process was conducted, focusing on one byte at a time. The goal was to modify CC1 in each round, generate a modified ciphertext, and send it to the padding oracle to determine if the padding was valid. By trying all possible byte values for each round, one byte of D2 could be derived. The code provided in the task served as a foundation for constructing the attack. The program looped through different byte values of CC1, checking the padding validity through interaction with the padding oracle. Upon finding valid padding, the derived value was used to update D2. This process was repeated until all bytes of D2 were derived for each block. Finally, the plaintext blocks were calculated by performing XOR operations between the corresponding ciphertext blocks and derived D values.

Observations:
The automated padding oracle attack successfully decrypted all blocks of the secret message. By automating the process, it became feasible to derive the entire plaintext without manual intervention.

Explanation of Observations:
Automating the padding oracle attack in Task 3 proved highly effective in deriving all blocks of the secret message without manual iterations. By streamlining the attack process through the use of appropriate arrays and iterative procedures for each block, the attacker systematically derived the correct values for D. The automated attack program modified CC1, checked padding validity, and updated D values accordingly. This automation showcased the scalability and efficiency of the padding oracle attack, highlighting the vulnerability of systems relying on padding validation during decryption. The successful decryption of all blocks underscores the importance of implementing robust encryption algorithms and secure padding schemes to prevent padding oracle attacks and maintain the confidentiality of sensitive information.

Conclusion

The experiment provided important light on the function of PKCS#5 padding and the significance of block cipher encryption. The Padding Oracle Attack made it clear how cryptographic padding flaws may be used to decode data without having access to the encryption keys. The attack's human execution exposed its specifics and difficulties, but the programmed method demonstrated how automation can make such attacks more effective. This experiment emphasizes how important it is to use strong encryption procedures in order to protect data from attacks like this.