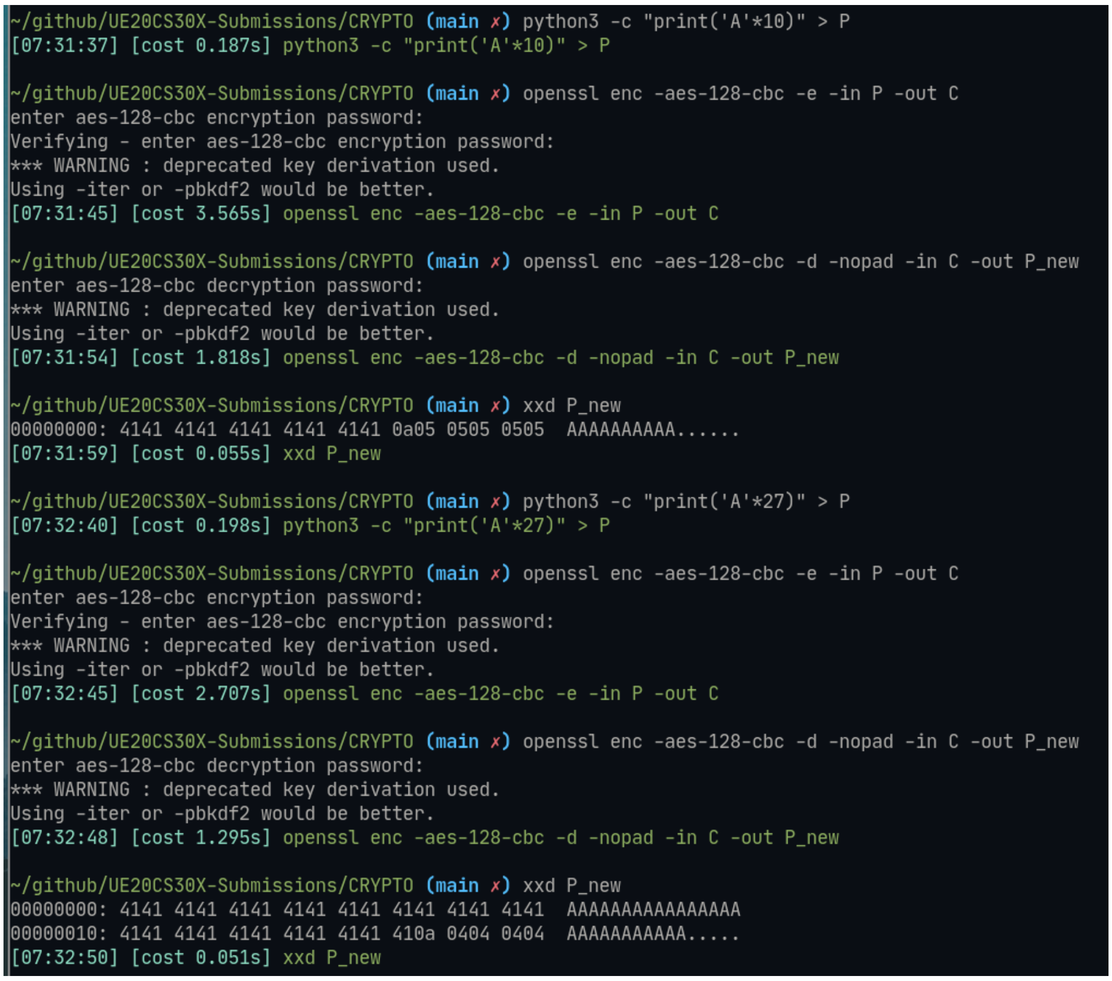
Cryptograhphy Hands-On submission 10

Details :

- SRN : PES2UG20CS237
- Name : P K Navin Shrinivas
- Section : D

TASK 1 : Understanding padding

Screenshots :



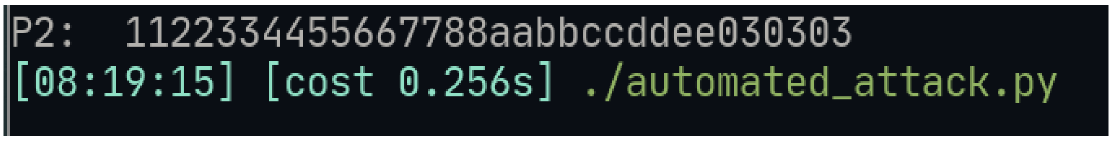
```
~/github/UE20CS30X-Submissions/CRYPTO (main ✗) python3 -c "print('A'*10)" > P
[07:31:37] [cost 0.187s] python3 -c "print('A'*10)" > P

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) 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.
[07:31:45] [cost 3.565s] openssl enc -aes-128-cbc -e -in P -out C

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) 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.
[07:31:54] [cost 1.818s] openssl enc -aes-128-cbc -d -nopad -in C -out P_new

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) xxd P_new
00000000: 4141 4141 4141 4141 4141 0a05 0505 0505  AAAAAAAAAA......
[07:31:59] [cost 0.055s] xxd P_new

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) python3 -c "print('A'*27)" > P
[07:32:40] [cost 0.198s] python3 -c "print('A'*27)" > P

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) 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.
[07:32:45] [cost 2.707s] openssl enc -aes-128-cbc -e -in P -out C

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) 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.
[07:32:48] [cost 1.295s] openssl enc -aes-128-cbc -d -nopad -in C -out P_new

~/github/UE20CS30X-Submissions/CRYPTO (main ✗) xxd P_new
00000000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000010: 4141 4141 4141 4141 4141 410a 0404 0404  AAAAAAAAAAA.....
[07:32:50] [cost 0.051s] xxd P_new
```

Observation :

- The encrpytion scheme is reversable and adds padding for to make the inputs as even multiples.

TASK 2 : Level 1 oracle attack

Screenshots :



```
P2:  1122334455667788aabbccddee030303
[08:19:15] [cost 0.256s] ./automated_attack.py
```

Observation :

- After trying all 256 combination for 16 hex codes, we can get the entire plain text quite easily

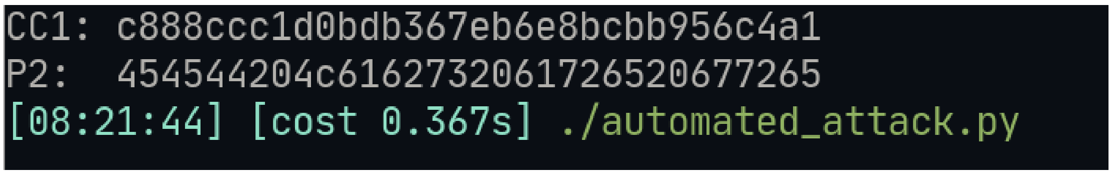
TASK 3

Code :

```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[00:16]
    C1    = iv_and_ctext[16:32]  # 1st block of ciphertext
    C2    = iv_and_ctext[32:48]  # 2nd block of ciphertext
    print("C1:  " + C1.hex())
    print("C2:  " + C2.hex())
    D2 = bytearray(16)
    CC1 = bytearray(16)
    D2[0]  = C1[0]
    D2[1]  = C1[1]
    D2[2]  = C1[2]
    D2[3]  = C1[3]
    D2[4]  = C1[4]
    D2[5]  = C1[5]
    D2[6]  = C1[6]
    D2[7]  = C1[7]
    D2[8]  = C1[8]
    D2[9]  = C1[9]
    D2[10] = C1[10]
```

```
        D2[11] = C1[11]
        D2[12] = C1[12]
        D2[13] = C1[13]
        D2[14] = C1[14]
        D2[15] = C1[15]

        CC1[0]  = 0x00
        CC1[1]  = 0x00
        CC1[2]  = 0x00
        CC1[3]  = 0x00
        CC1[4]  = 0x00
        CC1[5]  = 0x00
        CC1[6]  = 0x00
        CC1[7]  = 0x00
        CC1[8]  = 0x00
        CC1[9]  = 0x00
        CC1[10] = 0x00
        CC1[11] = 0x00
        CC1[12] = 0x00
        CC1[13] = 0x00
        CC1[14] = 0x00
        CC1[15] = 0x00
        for j in range(1,18):
            K = j
            for i in range(256):
                  CC1[16 - K] = i
                  P2 = xor(C1, D2)
                  status = oracle.decrypt(IV + CC1 + C2)
                  if status == "Valid":
                      print("Valid: i = 0x{:02x}".format(i))
                      print("CC1: " + CC1.hex())
                      D2[16-K] = K ^ i#C1[16-K]
                      for j in range(16-K,16):
                        CC1[j] = (D2[j] ^ K+1)
                      break

    ################################################################

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

Screenshots :



```
CC1: c888ccc1d0bdb367eb6e8bcbb956c4a1
P2:  454544204c61627320617265206772
[08:21:44] [cost 0.367s] ./automated_attack.py
```

Observation :

- Using the code above, we can crack the code in one time, hence we have the decrpyted text using the same logic!