Joseph Shaffer

CSE 5473

Lab 2

2. Padding 01

Encrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -e 02020202020202020202020202020202
Plaintext after padding 02020202020202020202020202020201
Ciphertext: a5aff8141e959e377aa38cb2c6d62609
```

Decrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -d a5aff8141e959e377aa38cb2c6d62609
Plaintext: 02020202020202020202020202020202
```

Padding 0202

Encrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -e 01010101010101010101010101010101
Plaintext after padding 01010101010101010101010101010202
Ciphertext: ed21e157c5c5373ab27a7d2960eb80c0
```

Decrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -d ed21e157c5c5373ab27a7d2960eb80c0
Plaintext: 01010101010101010101010101010101
```

Padding 030303

Encrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -e 01010101010101010101010101010101
Plaintext after padding 01010101010101010101010101030303
Ciphertext: 8982e488d8385632cb761ebf36590cd8
```

Decrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -d 8982e488d8385632cb761ebf36590cd8
Plaintext: 01010101010101010101010101010101
```

Padding 04040404

Encrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -e 01010101010101010101010101010101
Plaintext after padding 01010101010101010101010104040404
Ciphertext: 848d8659e97646bef6f7cdb310b7a34b
```

Decrypt

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -d 848d8659e97646bef6f7cdb310b7a34b
Plaintext: 01010101010101010101010101010101
```

As can be seen from the screenshots above, 4 hexadecimal strings were selected so that their last block contains the padding 01, 0202, 030303, and 04040404 respectively. After encrypting these strings, the plaintexts were printed after padding along with the ciphertexts after encryption. After that, those ciphertexts, were decrypted and unpadded, which allowed us to find that after decryption, we get the original plaintexts back.

String Repeated 3 Times Encryption

```
joe@joe-VirtualBox:~/Documents$ python ecb.py -e 0102030405060708080706050403020101020304050607080807060504030201010203040506070808070605040302
01
Plaintext after padding 0102030405060708080706050403020101020304050607080807060504030201010203040506070808070605040302011010101010101010101010101
010101010
Ciphertext: dc1836ee4a8a87f3186a2a1f1bd3bd00dc1836ee4a8a87f3186a2a1f1bd3bd00dc1836ee4a8a87f3186a2a1f1bd3bd0000657ea140655a44782747705d422fad
```

From the screenshot above, we can see a string that is exactly the length of one block and is also repeated 3 times to construct a new string. Also, from the screenshot, we can see the padding is one whole block of 10 ('16' in hexadecimal) and we can see repeated ciphertext blocks after encryption. Meaning ECB will produce the same ciphertext after encryption for the same plaintext with the same key.

3.

Encrypt

```
joe@joe-VirtualBox:~/Documents$ python cbc.py -s 'Congratulations! You have earned the extra credit!'
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617274686520657874726120637265646974210e0e0e0e0e0e0e0e0e0e0
e0e0e0e0e
PlainText Block: 436f6e67726174756c6174696f6e7321
Rest of Plaintext: 20596f7520686176652065617274686520657874726120637265646974210e0e0e0e0e0e0e0e0e0e0e0e0e0e0e
Encrypted Block: e8dd36746bf44af4537f7e18b254c678
Ciphertext: e8dd36746bf44af4537f7e18b254c678
PlainText Block: 20596f7520686176652065617265206e6564
Rest of Plaintext: 2074686520657874726120637265646974210e0e0e0e0e0e0e0e0e0e0e0e0e0e0e
Encrypted Block: 00e799df57a2478a19c17d646fdb661d
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d
PlainText Block: 2074686520657874726120637265646469
Rest of Plaintext: 74210e0e0e0e0e0e0e0e0e0e0e0e0e0e
Encrypted Block: 6c7bcf7af6b6c19ec26937e31ac3d14c
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14c
PlainText Block: 74210e0e0e0e0e0e0e0e0e0e0e0e0e0e
Rest of Plaintext:
Encrypted Block: bdc53dcb158a8cae58ed03e8c1110823
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
```

Plaintext: 'Congratulations! You have earned the extra credit!'

Ciphertext:
e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d1
4cbdc53dcb158a8cae58ed03e8c1110823

IV: 0f1e2d3c4b5a69788796a5b4c3d2e1f0

The ciphertext is created from encrypting the given plaintext with the given key and the IV created by me.

Decrypt

```
joe@joe-VirtualBox:~/Documents$ python cbc.py -u e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d1
4cbdc53dcb158a8cae58ed03e8c1110823
Ciphertext Block: e8dd36746bf44af4537f7e18b254c678
Rest of Ciphertext: 00e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Plaintext Block: 436f6e67726174756c6174696f6e7321
Ciphertext Block: 00e799df57a2478a19c17d646fdb661d
Rest of Ciphertext: 6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617265206e6564
Ciphertext Block: 6c7bcf7af6b6c19ec26937e31ac3d14c
Rest of Ciphertext: bdc53dcb158a8cae58ed03e8c1110823
Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617274686520657874726120637265646469
Ciphertext Block: bdc53dcb158a8cae58ed03e8c1110823
Rest of Ciphertext:
Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617274686520657874726120637265646974210e0e0e0e0e0e0e0e0e0e0e0e0e0e
0e
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617274686520657874726120637265646974210e0e0e0e0e0e0e0e0e0e0e0
e0e0e0e0e
Plaintext Congratulations! You have earned the extra credit!
```

This screenshot shows when decrypting the ciphertext from the encryption of the given plaintext, I end up with the same plaintext that was encrypted.

4.

Invalid Padding Recognition

Did not pad inputted plaintext, and inputted one block with incorrect padding. Oracle recognizes incorrect padding of 05 for the block of 01010101010101010101010101010505.

```
joe@joe-VirtualBox:~/Documents$ python oracle.py -e 01010101010101010101010101010505
Padded Plaintext Block: 01010101010101010101010101010505
No: Invalid Padding
Ciphertext: 2236fde995bdea92de76bb8963cd5452
```

Valid Padding Recognition

```
joe@joe-VirtualBox:~/Documents$ python oracle.py -e 01010101010101010101010101010505
Padded Plaintext Block: 0101010101010101010101010101050510101010101010101010101010101010
Yes: Valid Padding
Padding is: 10101010101010101010101010101010
Ciphertext: 2236fde995bdea92de76bb8963cd545279f8b45bcd483c6e44c0e3aaba1e9d6b
```

```
joe@joe-VirtualBox:~/Documents$ python oracle.py -e 0101010101010101010101010101
Padded Plaintext Block: 01010101010101010101010101010202
Yes: Valid Padding
Padding is: 0202
Ciphertext: f5cdae3f34233734c6e72bb1770c26de
```

```
joe@joe-VirtualBox:~/Documents$ python oracle.py -s 'Congratulations! You have earned the extra credit!'
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f752068617665206561726e65642074686520657874726120637265646974210e0e0e0e0e0e0e0e0e0
e0e0e0e0e
Yes: Valid Padding
Padding is: 0e0e0e0e0e0e0e0e0e0e0e0e0e0e
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
```

I outputted the padding to confirm that my oracle was identifying correct padding.

The screenshots above show the oracle returns “Yes: Valid Padding” when valid padding is used for the corresponding plaintext being encrypted and returns “No: Invalid Padding” when invalid padding is used for the corresponding plaintext being encrypted.

5.

```
joe@joe-VirtualBox:~/Documents$ python attack.py -e 0102030405060708090a0b0c0d0e0f100203040506
Padded Plaintext Block: 0102030405060708090a0b0c0d0e0f10020304050600b0b0b0b0b0b0b0b0b0b0b
Ciphertext: e28133a831d3d5ec50921f141ba2a9f2f1518cde5a94a09f86e62dc38dc0b75e
Attack Decryption Plaintext: 0102030405060708090a0b0c0d0e0f10
```

```
joe@joe-VirtualBox:~/Documents$ python attack.py -e 02020202020202020202020202020202020202
Padded Plaintext Block: 020202020202020202020202020202020202020d0d0d0d0d0d0d0d0d0d0d0d0d
Ciphertext: 0f1e2d3c4b5a69788796a5b4c3d2e1f0b5ce6473d389d59f18baf459e6550b9136c5400fb21a76d0e3a347811c813135
Attack Decryption Plaintext: 02020202020202020202020202020202
```

```
joe@joe-VirtualBox:~/Documents$ python attack.py -s 'Congratulations! You have earned the extra credit!'
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617261206372656469742120e0e0e0e0e0e0e0e0e0e0
e0e0e0e0e
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Attack Decryption Plaintext: Congratulations!
```

After implementing the padding oracle attack, I was able to decrypt the first block of a given ciphertext, but as you can see from the first screen shot, the padding oracle attack cannot remove the padding, if the padding is in the first block. If I extended this further, the padding oracle attack would decrypt all blocks of the ciphertext, but would still leave the last block of the plaintext with padding. I addressed the removal of the padding in the extra credit section, which shows the decryption of all blocks of the ciphertext. For this section, I only focused on decrypting the first block.

```
joe@joe-VirtualBox:~/Documents$ python attack.py -s 'Congratulations! You have earned the extra credit!'
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617261206372656469742120e0e0e0e0e0e0e0e0e0e0
e0e0e0e0e
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Attack Decryption Plaintext: Congratulations! You have earned the extra credit!=0E=0E=0E=0E=0E=0E=0E=0E=
=0E=0E=0E=0E=0E=0E
```

```
joe@joe-VirtualBox:~/Documents$ python attack.py -e 02020202020202020202
Padded Plaintext Block: 02020202020202020202020505050505
Ciphertext: 2ea0815c56e57ae619c5ee70d41c2675
Attack Decryption Plaintext: 02020202020202020202020505050505
```

The two screen shots above, show extending the padding attack oracle to decrypt all blocks of the ciphertext, but it can be seen that this not remove the padding from the last block of plaintext. I just extended the padding oracle attack to address all blocks of the ciphertext and not just the first block.

Extra Credit:

```
joe@joe-VirtualBox:~/Documents$ python attack.py -s 'Congratulations! You have earned the extra credit!'
Padded Plaintext Block: 436f6e67726174756c6174696f6e732120596f7520686176652065617261206372656469742120e0e0e0e0e0e0e0e0e0e0
e0e0e0e0e
Ciphertext: e8dd36746bf44af4537f7e18b254c67800e799df57a2478a19c17d646fdb661d6c7bcf7af6b6c19ec26937e31ac3d14cbdc53dcb158a8cae58ed03e8c1110823
Attack Decryption Plaintext: Congratulations! You have earned the extra credit!
```

After using the attack to decrypt all the blocks of the ciphertext, I end up with all the plaintext blocks, with the last plaintext block being padded. To remove the padding, I send CN-1 and CN to the oracle to see if there is valid padding in CN, if the padding is invalid, I know CN is not the last block and does not have any padding. If there is valid padding, I can assume, CN is the last block and does have padding that needs to be removed. To do this I changed the first byte of CN-1 and sent CN-1 and CN to the oracle to check if there is valid padding. If there is still valid padding, I know that the byte I changed was data, and the whole block is not padding. I repeat this by then changing the second byte of CN-1 and send CN-1 and CN to the oracle to see if there is valid padding. I repeat this step for all the bytes of CN-1 until I find out from the oracle that there is no longer valid padding, which means that the byte I changed in CN-1 did not affect the data in CN but the padding in it. Thus, I know where the padding starts in PN. I already have PN from attacking and decrypting all the blocks with the attack, but now have to remove the padding from the last block, PN, of the plaintext. From knowing where the padding begins, I can just remove the end of the plaintext block from where the padding begins and I end up with the original plaintext. For example, the padding in this case was ‘0e’ 14 times, and thus I removed 14 bytes from the end of PN to get the original PN block without padding.

The difficult part was determining where the padding begun, which was different from the original attack, where you decrypted a block byte by byte from the last byte of CN-1, but in this case, I started with the first byte of CN-1. The difficulty lay in the fact that I needed to know where in PN the padding begun, since I knew it had to end at the end of the block. If I used the same approach as decrypting the first block, I would not be able to see where the padding begun. Basically, it came down to how to determine if the block had padding and where the padding began in the block. And the method used to decrypt each block byte by byte did not allow for me to know where the padding begun, thus I did not address the padding until I already decrypted all the plaintext blocks from the attack. I assume that you could combine the decryption and padding removal together as well as start at the last byte to determine where the padding begins in PN, but I felt the combination of both could be complicated, especially as CN-1 prime is being changed along with PN prime to determine D(K, C).