5th Feb,2018	**CS 5321** **Assignment 1**	
		Due: 18th Feb 2018, 11:59PM

Assignment 1 is about attacking a cipher using a padding oracle. For this assignment, you will implement the padding attack discussed in class. This attack was first discussed in the paper "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..." by S. Vaudenay. You should read the paper (http://dx.doi.org/10.1007/3-540-46035-7_35) and implement the described attack.

Submission will be made via IVLE and must be received by ***Sunday, 18th Feb 2018, 11.59pm.***

1 Padding Oracle Attack [70 pts]

In Problem 1, you will have the opportunity to create your own padding attack discussed in class to extract a message from a ciphertext. You can find a set of valid ciphertexts on the IVLE handout section. The ciphertexts will be given as a 128-bit hexadecimal number (starting with 0x...) computed as follows:

$$C = C_0 \| C_1 = CBC_K^{Encrypt}(PAD(M)),$$

where the word M is shorter than or equal to 8 ASCII characters (i.e., 64 bits). The block size of the cipher and the length of the secret key K are 64 bits. The first 64-bit block of C is the initialization vector (IV), which we denote C_0, and the last 64-bit block is the first cipher block, C_1. Note that we assume the randomized CBC encryption; that is, the IV (C_0) is randomly selected for every message.

You are given a padding oracle, which can be accessed by function calls; see Section 3 for the details. You can access the padding oracle as many times as you want. You will have two language options to develop your padding attack program: Java or Python. The padding oracle will accept both C_0 and C_1 (each 64 bits long), and return a 1 or a 0, indicating correct or incorrect padding respectively. The padding oracle works as follows:

$$\{0, 1\} = \text{Check PAD}(CBC_K^{Decrypt}(C')),$$

where you provide the C'; i.e., you are a chosen-ciphertext attacker.

1.1 Submission & Grading

Submission: Your code has name *p1_S_#.java (or .py)* where # denotes the student id. Tar all your codes (*S_#.tar*) and upload it to IVLE. Your code should accept two command-line arguments C_0 and C_1 in the following format:
For java

- javac -cp pad_oracle.jar p1_S_#.java &&
  java -cp pad_oracle.jar:bcprov-jdk15-130.jar: p1_S_# C_0 C_1

For python

- python p1_S_#.py C_0 C_1

The ciphertext blocks (C_0 and C_1) have hex format starting with 0x. The output must be the plaintext in ASCII format.

Grading: Your code should successfully obtain the plaintext from a ciphertext. You will get **40 pts** if your code works for the ciphertexts that are available on handouts. If it works with all the other ciphertexts (not given to students), an additional **30 pts** will be given.

2 Turning Decryption Oracle into Encryption Oracle [30 pts]

What you will create in Problem 1 is a single-block decryption oracle that takes a two-block cipher-text (one of which is an IV) and returns a single-block plaintext. Now, you are asked to create an encryption oracle using the single-block decryption oracle. Your code needs to take an arbitrary message M, pad it if needed, and encrypt it without knowing the secret key. Note that the message M, can be longer than one plaintext block.

We provide a single-block decryption oracle so that you can solve Problem 2 without solving Problem 1. The single-block decryption oracle is accessed via calling the oracle function. Note that the secret key used in Problem 2 is different from the one used in Problem 1.[1]

2.1 **Submission & Grading**

Submission: Your code has name *p2_S_#.java(or .py)*, where # denotes the student id. Tar all your codes (*S_#.tar*) and upload it to IVLE. Your code should accept any arbitrary message, M in the command-line argument:

For java

- javac -cp dec_oracle.jar p2_S_#.java &&
  java -cp dec_oracle.jar:bcprov-jdk15-130.jar: p2_S_# "This is the message that needs to be encrypted."

For python

- python p2_S_#.py "This is the message that needs to be encrypted."

The output must be the one or more 64-bit hex formats; e.g., 0x12...ef 0x98..ab 0xb7..2d.

Grading: Your code should encrypt all the multiple test plaintext messages and generate valid ciphertexts to get **30 pts**. Partial points will be given when not all test plaintext messages are encrypted successfully.

[1]Since the two keys are different, you cannot use the decryption oracle provided in Problem 2 to solve Problem 1.

3 How to access the oracles?

You will be given the following list of files:

- pad_oracle.jar: Padding oracle (Java bytecode[2]) for Problem 1.
- dec_oracle.jar: Decryption oracle (Java bytecode) for Problem 2.
- oracle_python_v1_2.py: Padding and decryption oracles (Python functions) for Problem 1 and 2, respectively (version 1.2).
- python_interface_v1_2.jar: Java bytecode for interfacing Python scripts and oracle Java bytecodes.
- bcprov-jdk15-130.jar: Crypto library.

Here are the example codes that access the oracles.
Java:

```
// query padding oracle
pad_oracle p = new pad_oracle ();
boolean isPaddingCorrect = p.doOracle("0x1234567890abcdef", "0x1234567890abcdef");

// query decryption oracle
dec_oracle d = new dec_oracle ();
String plaintext = d.doOracle("0x1234567890abcdef", "0x1234567890abcdef");
```

Python:

Execute the Java-Python interface class by
java -cp pad_oracle.jar:dec_oracle.jar:bcprov-jdk15-130.jar:python_interface_v1_2.jar python_interface_v1_2

In your Python script,
from oracle_python_v1_2 import pad_oracle, dec_oracle

```
# query padding oracle
ret_pad = pad_oracle('0x1234567890abcdef', '0x1234567890abcdef');
# query decryption oracle
ret_dec = dec_oracle('0x1234567890abcdef', '0x1234567890abcdef');
```

[2]Note that (1) you should not reverse-engineer the bytecodes and (2) even if you did, it would not help.