Introduction to Cyber Security 2020
Project: CBC Encryption

Submit by July 31, 2020

You should implement your solution in Python3 (if you plan on using special modules in Python then ask us before using them, to make sure that we allow it). **For each function, you should also write a short description documentation (as a comment) of the algorithm that you implemented.**

Remark. Use Python3 and the pycryptodome package for cipher-related operations (see recitation 6 for an example). **You are only allowed to use AES.MODE_ECB throughout the exercise**!

Submission Instructions

The submission should contain a single file. The first line of your file must be a comment with:
$\langle firstname\rangle\langle comma\rangle\langle lastname\rangle\langle comma\rangle\langle ID\rangle$
The second line of your file must be a comment:
$\langle Python\_Version\rangle$
Example:

```
#Israel,Israeli,123456789
#Python 3.6
```

The submission would be via the Moodle system (`https://lemida.biu.ac.il/`).
Submission that does not follow the above format might be skipped (without checking and grading).

Submit a file named 'project.py' with the implementation of the following functions:

- cbc_custom_decrypt(k, n, cipher)
- cbc_flip_fix(k, n, cipher)

Reminder

A hex string is a string composed of hexadecimal characters `0`, ..., `F`. Every ASCII character is represented by 2 hexadecimal characters. For example, the NULL character which is 0 in ASCII is represented by the hex string `00`, the characters 'a',...,'z', which are 97,...,122 in ASCII, are represented by the hex strings `61`,...,`7A` respectively. Sometimes, to make sure that we are talking about a hex string we add the '0x' in the beginning of it ('\x' is another common way), but it is not necessary.

For instance, the hex string "`68656C6C6F`" represents the ASCII string "hello", and the hex string
"`68656C6C6F2C20776F726C64`" represents the ASCII string "hello, world".

CBC Encryption Mode

Recall that CBC (cipher block chaining) is an encryption mode for block cipher. Let E be a block cipher which receives a 16 byte key k and a 16 byte plaintext m, and outputs the 16 byte ciphertext $E(k, m)$.

CBC encryption receives as inputs a message composed of n blocks, each of length 16 bytes: $m_1, \ldots, m_n$. It also receives as input a 16 byte known initialization vector IV.

Given this input, CBC encryption produces a ciphertext of $n + 1$ blocks, each of length 16 bytes:

- $c_0 = IV$
- $c_i = E(k, m_i \oplus c_{i-1})$, for $1 \leq i \leq n$.

Question 1

We will implement CBC with AES as the block cipher.

Write a function `cbc_custom_decrypt(k, n, cipher)` which receives three inputs: a key k, an integer n, and a string of $n + 1$ blocks of 16 bytes.

Your function should implement CBC decryption by using *only ECB decryption.* Your function output should be the n block CBC decryption of this ciphertext (overall, your output should be $16 \cdot n$ bytes – see output examples below).

Note. You may use the sample code from recitation 6 as a reference. **You are only allowed to use AES.MODE_ECB throughout the exercise**!

Input Example

```
k = b'\x81\x0ff\t\x04\xb6\xcf\x1f.\x10\x8frd\xb4E\x19'

>>> type(k)
<class 'bytes'>
```

If you would like to transform k into a hex string you may use .hex() as follows:

```
>>> k.hex()
'810f660904b6cf1f2e108f7264b44519'
```

Consider we chose:

```
IV = b'e|\x92\xd0\x8b\xd9\x00\xc8X\xf2Noi\xa1\x155'
>>> len(IV)
16
```

and we encrypted a *single* block of 16 bytes (in this example, n=1). So, a possible input cipher might be:

```
cipher = b'e|\x92\xd0\x8b\xd9\x00\xc8X\xf2Noi\xa1\x155
\x8b\xa5\xb7\xdcka\xaa\x94=a_!x\x1a\xcf\xf4'
>>> type(cipher)
<class 'bytes'>
>>> len(cipher)
32
```

Note that the number of blocks we encrypt depends on n which we pass as a parameter to your function.

Output Example

Your output should also be of type 'bytes'. Example for a possible output (n=1):

```
k = b'\x81\x0ff\t\x04\xb6\xcf\x1f.\x10\x8frd\xb4E\x19'
n = 1
cipher = b'e|\x92\xd0\x8b\xd9\x00\xc8X\xf2Noi\xa1\x155
\x8b\xa5\xb7\xdcka\xaa\x94=a_!x\x1a\xcf\xf4'

output = cbc_custom_decrypt(k, n, cipher)
>>> type(output)
<class 'bytes'>
>>> output
b'1111111111111111'
```

Example for a possible output (some unknown k and cipher, n=2):

```
output = cbc_custom_decrypt(k, n, cipher)
>>> output
b'54artn;yt2\xccd\xacgq212341124a\xac2\xad0\x00\xce1'
```

Verification

You may verify your code works properly by using AES.MODE_CBC as follows. Consider the following parameters:

```
k = b'\xfcV\xc8\x7f\xcf\x8f\x9ff\x8c\xadX\xaf\xa1
\x0fs\x1e'

IV = b'\xf51\xf7\xe4\xb1m\xda\xed\xddz\xb4
\xff.\x8dN\xe6'

cipher = b'\xf51\xf7\xe4\xb1m\xda\xed\xddz\xb4
\xff.\x8dN\xe6|8\xa1x\x18@\xb1\x82\x98\x01
\xb3"\xdc\x95\xc2\\d{\xe8(\xb6\x93G\x8a#\x04q
\xb6\x89\xbfN\x9a'
```

You can verify your function output by running the following code:

```
>>> crypto = AES.new(k, AES.MODE_CBC, cipher[:16])
>>> crypto.decrypt(cipher[16:])
b'aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbb'
```

Question 2

Suppose that bit number j of block c_i of the ciphertext got flipped (namely, if the original value of the bit was 0 then it changed to 1, and if the original value of the bit was 1 then it changed to 0). Convince yourself that the decryption process will decrypt all blocks correctly, except for blocks i and $i+1$. The decryption of block i will be completely random, and the decryption of block $i+1$ will be correct, except for bit j in this block that will be flipped.

Write a function cbc_flip_fix(k, n, cipher) which receives three inputs:

- A key k
- An integer n
- A string of $n+1$ blocks of 16 bytes. This string was generated in the following way:
  - Each of the n plaintext blocks m_i, was generated by choosing a *random* byte and repeating it 16 times.
  - The plaintext message $m_1, \ldots, m_n$ was encrypted using the key k in CBC mode. The result is $c_0, \ldots, c_n$.
  - A random bit in one of the blocks $c_1, \ldots, c_{n-1}$ was flipped.
  - The resulting $n+1$ blocks are the input given to the function.

Your function should output the original value of the block whose encryption was completely corrupted.

Example

Assume we encrypted:

```
msg = b'aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbb'
IV = b'\xd1\xd9\x00\x926)\x030i:\x11/B\x89\xfb\x8b'
```

The result is *cipher* $= c_0, c_1, c_2$ (where c_0 is the IV). Now, assume a bit was flipped in c_1, so your *decryption* might look like this:

```
'\xd1\xd9\x00\x926)\x030i:\x11/B\x89\xfb\x8b
\xb4V*6e\xb7\xcf\xbc\xa5\n\xe9~\xd2f\x93\x86
bbbbbbbbbfbbbbbb'
```

Your function should return `aaaaaaaaaaaaaaa` as 'bytes'.