Decoding a Secret Text Encrypted using a Cipher based on AES

CE235 Assignment 3

Specification

Aim

You are going to complete a Python 3 program so that it successfully decodes a secret message which is encoded using a baby block cipher related to AES.

Blocks, Bytes and Nibbles

A block is 8 bits (or one byte, or one ASCII character). A key is 8 bits (or one byte). A block is treated as consisting of four nibbles, each nibble being two bits.

A block is 8 bits: 1 2 3 4 5 6 7 8.

Nibbles are: 1 2, 3 4, 5 6, 7 8.

A nibble can be treated either as two bits or as a number in arithmetic modulo 4 (that is, a number in the range 0−3).

Elements of the Cipher

- Bitwise XOR of the block and the key;
- Replacing each nibble in a block by another nibble according to a substitution table (sometimes called an S-box);
- Swapping the last two nibbles of the block;
- Adding the last two nibbles to the first two nibbles, or subtracting the last two nibbles from the first two nibbles.

Bitwise XOR

This is simply a bitwise XOR of the block and the key:

```
11101110
11010100
--------
00111010
```

S-Box

When encrypting, **FOR INFORMATION ONLY, WE ARE NOT DOING THIS**, substitute the value of each nibble of the block as follows:

A value 0 becomes 1,
A value 1 becomes 3,
A value 2 becomes 0,
A value 3 becomes 2.

When decrypting, **THIS IS WHAT WE ARE DOING**, substitute the value of each nibble of the block in the opposite direction, as follows:

A value 0 becomes 2,
A value 1 becomes 0,
A value 2 becomes 3,
A value 3 becomes 1.

Swapping Two Nibbles

Permute the positions of the last two nibbles of a block. In other words, a block

1 2, 3 4, 5 6, 7 8

becomes

1 2, 3 4, 7 8, 5 6

Adding and Subtracting Nibbles

When encrypting (**for information, we are not doing this**), add the 3rd nibble to the 1st nibble and make that the new value of the 1st nibble; add the 4th nibble to the 2nd nibble and make that the new value of the 2nd nibble. The 3rd nibble and 4th nibble remain as they were before.

When decrypting (**we are doing this**), do the opposite: subtract the 3rd nibble from the 1st nibble and make that the new value of the 1st nibble; subtract the 4th nibble from the 2nd nibble and make that the new value of the 2nd nibble. The 3rd nibble and 4th nibble remain as they were before.

When doing addition and subtraction, treat each nibble as a number in arithmetic modulo 4 (that is, a number in the range 0−3). This is easy in Python even for negative numbers - see below.

Encryption Algorithm (We are Not doing)

- Bitwise XOR of the block and the key
- Use the S-box substitution on each nibble of the block
- Swap the last two nibbles of the block
- Add the last two nibbles to the first two
- Bitwise XOR of the block and the key
- Use the S-box substitution on each nibble of the block
- Swap the last two nibbles of the block
- Bitwise XOR of the block and the key

Decryption algorithm (We Are doing)

- Bitwise XOR of the block and the key

- Swap the last two nibbles of the block
- Use the S-box substitution on each nibble of the block
- Bitwise XOR of the block and the key
- Subtract the last two nibbles from the first two
- Swap the last two nibbles of the block
- Use the S-box substitution on each nibble of the block
- Bitwise XOR of the block and the key

A Cipher to Decrypt

The following list of numbers is a text encrypted with the cipher:

```
[132, 201, 141, 74, 140, 94, 141, 140, 141, 15, 31, 164, 90, 229, 201, 141,
78, 114, 241, 217, 141, 217, 140, 180, 141, 164, 51, 141, 188, 221, 31,
164, 241, 177, 141, 140, 51, 217, 141, 201, 229, 152, 141, 78, 241, 114,
78, 102, 94, 141, 74, 152, 31, 152, 141, 94, 201, 31, 164, 102, 164, 51,
90, 141, 201, 229, 164, 31, 201, 152, 152, 51, 115]
```

The key used for encryption is `84`.

Your task is to implement the decryption algorithm in Python and decrypt the cipher above. We also specify the encryption algorithm in case you would like to do that as well, for your own interest.

Worked Example 1

```
Block 132                   10 00 01 00
Key 84                      01 01 01 00
XOR block and key           11 01 00 00
Swap last 2 nibbles         11 01 00 00
S-Box                       01 00 10 10
Key                         01 01 01 00
XOR Block & Key             00 01 11 10
Subtract 2nd 2 from 1st 2   01 11 11 10
Swap last 2 nibbles         01 11 10 11
S-Box                       00 01 11 01
Key                         01 01 01 00
XOR Block & Key             01 00 10 01

Answer: 73, i.e. 'I'.
```

How to Solve it

First of all read all the instructions carefully, go through the worked example and make sure you understand it exactly! Then look at the program.

The starting point is `dec_decrypt_aes_part.py`. You are provided with this program. You can run this from the command line as follows:

```
python dec_decrypt_aes_part.py a
```

```
python dec_decrypt_aes_part.py b
python dec_decrypt_aes_part.py c
python dec_decrypt_aes_part.py d
python dec_decrypt_aes_part.py e
```

Each argument causes a different function to be called:

a function dec_decrypt applied to cipher text above (ie this checks your complete solution)

b function dec_xor applied to two binary numbers in nibble form

c function dec_swap_last_two applied to a binary number in nibble form

d function dec_s_box applied to a binary number in nibble form

e function dec_subtract_last_two_first_two applied to a binary number in nibble form

Bad News: You have got to write those five functions! They are only stubs at present.

Good News: I have written everything else including a lot of the tricky parts. **You must not change any other code at all except in those five functions above.**

Note: Everything must be written in Python 3 running under Windows, exactly as configured in the CSEE Labs. **You must use the program given as the basis**. Please do not use Python 2.7 or any other programming language.

Hints and Tips

You need to start by taking a careful look at the program and noting how it is organised. *You do not need to understand every aspect of the program* in order to complete the assignment. *What you do need to understand* is:

- How to write each function - what it should be doing exactly.
- What the provided inputs are for and what types they are.
- What the required output is, which a function is going to return, and what type it is.

Because types in Python are implicit, you need to pay careful attention to this aspect of the task. The comments in the program are very detailed and they tell you this information, but you must read them carefully!

Now you can try running the program from the command line in CMD exactly as shown above and seeing what happens in each case. **The program already runs but it does not yet compute the value of any of the functions correctly.** You have got to write the functions first!

Let us comment on the program and outline the data types involved.

At the top, we define a global variable which contains the cipher text. Below it is a variable containing the decryption key.

`dec_decrypt( int_list, int_key )` is the top-level function. It takes a list of integers comprising the cipher text and it converts them to a bytearray. It then converts them to nibbles. Each byte becomes four nibbles in a bytearray consisting of four binary numbers; the value of each is one nibble, i.e. a two-bit binary number whose value is 0, 1, 2 or 3. We split into nibbles because all our operations are nibble-based.

We also convert the key into nibble form to facilitate subsequent operations. So the key is now a bytearray as well.

Next, we go though the bytes, four nibbles at a time, decrypting. The result is a list of bytearrays, one for each byte. Now, each bytearray corresponds to a character in the text which is no longer encrypted.

Next we convert the list of nibble quadruples (bytearrays) into a list of byte codes, each one an integer.

Finally, we convert the list of integer codes into a string. This is the message!

The above is for information only. You are not going to change anything at all in this function.

`dec_byte_to_nibbles( a_byte )` converts a single byte into four nibbles and returns the four nibbles as four numbers in a bytearray. You must not change this in any way but you should look at it to understand how we have extracted the bits using shift right (`>>`) and bit AND (`&`).

Now we should look at
`dec_decrypt_byte_as_nibbles( nibble_quadruple, key_quadruple )`
which you are going to write. This function carries out the decryption process, exactly as described above. Note the the arguments nibble_quadruple (the byte to be decoded) and key_quadruple (the key to decode with) are both in nibble form already, i.e. each is a bytearray of four numbers corresponding to the first, second, third and fourth nibble. This function **must** use the following functions:

```
dec_xor( nibbles, key )

dec_swap_last_two( nibbles )

dec_s_box( nibbles )

dec_subtract_last_two_first_two( nibbles )
```

So, you must write these four functions first before you write
`dec_decrypt_byte_as_nibbles( nibble_quadruple, key_quadruple )`

They are specified in the text above about the algorithm.

To make a running program for you to look at before starting work, all these functions return a result but it is not the correct result! Therefore the decryption does not work when you first run the program.

When you are developing your program, you can import it in the normal way from CMD, starting in the directory where you program is:

```
python from dec_decrypt_aes_part
import *
```

You can also run the program direct from the command line with the options a-e, as described above. As you will have noticed, options a-e carry out handy tests of the five functions we are interested in.

Labs

We have a two-hour lab on the Monday and Thursday of the submission week. Once again, we will be there to help you.

In this assignment you need to read the instructions carefully, go through the worked example line by line, study the code, think about a solution and perhaps start work before the lab to get the best use out of it.

At the start of the lab, you must know exactly WHAT you are trying to do, even though you may not be sure HOW to do it.

How to submit

Submit one .py file to Faser called:

cs_registrationnumber.py .

So, if your number is 1234567, your filename will be:

cs_1234567.py

Note that the filename is all lower case, it does not contain any spaces and it is a .py file.

Submission Deadline

11h59 Friday 01 March, Week 22.

Marking Scheme

There is 10% of the overall module for this assignment.

1. We will run your program from the command line on Windows with all five arguments:

```
python cs_registrationnumber.py a

python cs_registrationnumber.py b

python cs_registrationnumber.py c

python cs_registrationnumber.py d

python cs_registrationnumber.py e
```

2. We will check that the output produced by the program is correct in each case.

3. We will inspect the program to check that you have indeed altered the five functions as instructed, that nothing else has been changed and that the program is properly computing the results of each function.

We will allocate marks as follows:

Part a: 6%
Part b: 1%
Part c: 1%
Part d: 1%
Part e: 1%

Total: 10%

If everything is OK, you get full marks. Parts b-e either work or they do not; for part a, we may award some partial marks if it does not work but is close to working.

Plagiarism

You should work individually on this project. Anything you submit is assumed to be entirely your own work. The usual Essex policy on plagiarism applies: http://www.essex.ac.uk/plagiarism/ .

Acknowledgement

This assignment is based on an earlier one by Alexei Vernitski and Richard Sutcliffe. Many thanks to both for their ingenious ideas.