**SCHOOL OF COMPUTER SCIENCE**
**COURSEWORK ASSESSMENT PROFORMA**

**MODULE & LECTURER:** CM3110 Security, George Theodorakopoulos

**DATE SET:** 19 October 2015

**SUBMISSION DATE:** 27 November 2015, 11:59pm

**SUBMISSION ARRANGEMENTS:** Online, via Learning Central

**TITLE:** Security Coursework

This coursework is worth 30% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity. Your work should be submitted using the official Coursework Submission Cover sheet.

## INSTRUCTIONS

This coursework comprises three (3) parts. In each part, you are asked to write a Python function that implements the given specification of a particular cryptographic algorithm or attack. You should submit these three functions together with a single pdf report that briefly describes any important decisions you had to make in your code that are not explicitly listed in the specifications. Your report should also answer any questions that you are explicitly asked in the detailed description in the following pages.

## SUBMISSION INSTRUCTIONS

All files (1 .pdf cover sheet, 3 .py files, 1 .pdf report) should be in a single .zip archive. Below are the descriptions for each file separately, but you should only submit the .zip archive.

| Description | | Type | Name |
|---|---|---|---|
| Cover sheet | **Compulsory** | One PDF (.pdf) file | cover.pdf |
| Part A | **Compulsory** | One Python source file (.py) | parta.py |
| Part B | **Compulsory** | One Python source file (.py) | partb.py |
| Part C | **Compulsory** | One Python source file (.py) | partc.py |
| Report | **Compulsory** | One PDF (.pdf) file | report.pdf |
| Zip archive | **Compulsory** | One zip file containing all of the above files | [student number].zip |

## CRITERIA FOR ASSESSMENT

Each of the three parts is worth 10 marks, for a total of 30 marks.
Credit will be awarded against the following criteria:

1. Correctness of implementation as compared to the specification
2. Quality, clarity, and correctness (where applicable) of arguments in the report

Where applicable, more details on mark allocation are provided in the description below.

Feedback on your performance will address each of these criteria.

## FURTHER DETAILS

Feedback on your coursework will address the above criteria and will be returned in approximately three (3) weeks

This will be supplemented with oral feedback in the revision lecture.

Individual feedback will be provided on Learning Central. Further individual feedback will be available upon request.

## CM3110 Security Coursework

## Part A: Implement and evaluate a stream cipher (10 marks)

In this part, your task is to implement the stream cipher, based on the pseudocode given below (8 marks), and then to evaluate how long it takes to encrypt plaintexts of various sizes (2 marks).

### A1 – Implementation (8 marks)

The following pseudocode describes a stream cipher. The first function is the Key Setup phase, which uses a user-supplied secret key to initialize a permutation S that the cipher needs. The second function uses the permutation to generate a byte stream that can be XOR-ed with the plaintext (ciphertext) to encrypt (decrypt) it.

```
Key Setup
(
input: an array key[] containing k.len integers
output: a permutation K of the numbers 0…255
)

for i from 0 to 255
    K[i] := i
j := 0
for i from 0 to 255
    j := (j + K[i] + key[i mod k.len]) mod 256
    swap K[i], K[j]

Byte Stream Generator
(
input: Permutation K computed above
output: An infinite-length sequence of bytes (integers in 0…255)
)

i := 0
j := 0
start:
    i := (i + 1) mod 256
    j := (j + K[i]) mod 256
    swap K[i], K[j]
    output K[ (K[i] + K[j]) mod 256 ]
    goto start
```

Based on this pseudocode, you should write a program that takes as input

1. a character (either ‘e’ or ‘d’) that determines whether you should perform encryption or decryption, and
2. a file that contains a secret key in the form of an ASCII string, and
3. a file that contains the input text, either the plaintext to be encrypted or the ciphertext to be decrypted, depending on the character given earlier, and
4. a file that your program should create and store the output in.

Your program should run using

`python parta.py` *option keyfile inputfile outputfile*

Notes:

1. You will need to convert the key and the plaintext from an ASCII string to a sequence of integers. Convert it character-by-character using the function `ord()`, as follows:
   ```
   def convert(s):
       return [ord(c) for c in s]
   ```

2. After XOR-ing the byte stream with the (integer) plaintext, store the ciphertext in the output file as a string of hexadecimal digits (0123456789ABCDEF). Keep in mind that the result of XOR-ing one integer from the byte stream with one integer from the plaintext is an integer in 0…255, so you need two hex digits to represent it.

**Example 1 (encryption)**

```
python parta.py e key.txt plaintext.txt output1.txt
```

and the contents of the files are:

`key.txt:`
Secret key

`plaintext.txt:`
Transfer 100GBP to account 1234.

The correct output is
4E40B844800742DB4F9B8879E75727B244988FA23854AF93F3252EE07935FBB4

**A2 – Running time evaluation (2 marks)**

To evaluate the performance of your program, use Python's time function, as in the example code below.

```
import time

start = time.time()
# your function goes here
end = time.time()
# the function took 'end - start' seconds
```

You should measure the time it takes to encrypt plaintexts of increasing size (up to a few KB in size). Note that the time you measure will be influenced by other processes running at the same time, so for each plaintext size that you encrypt, repeat the measurement 10 times and take the average. Your report should include a plot and a characterization of any trend that you observe (i.e. How does the running time of your program scale with the plaintext size?).

## Part B: Attacking the Stream Cipher – Integrity (10 marks)

In this part, you will play the role of an attacker who intercepts a ciphertext and modifies it.
The attacker knows that the intercepted ciphertext, say of size *n* bytes, is an e-banking message that transfers some (known) amount to the attacker's bank account. He also knows that the amount information is contained in bytes *k1*to *k2*, inclusive (1<= k1 <= k2 <= n). His objective is to replace the amount with an arbitrary amount of his choice (probably larger).
Your task is to write a program that takes as input

1. a ciphertext of size *n* bytes (in the same form as in Part A: a hexadecimal number, i.e. a string of hexadecimal digits 0123456789ABCDEF), and
2. two integers k1 and k2 (1<= k1 <= k2 <= n), and
3. an ASCII string of size k2-k1+1 (original plaintext in bytes k1 to k2), and
4. an ASCII string of size k2-k1+1 (replacement plaintext for bytes k1 to k2)

and produces a ciphertext that, when decrypted with the key used for the original ciphertext, produces the original plaintext except that the attacker's chosen string is in bytes k1 to k2.

Your program should run using

`python partb.py` *ciphertextfile k1 k2 original replacement*

**Example 2 (attack)**

```
python partb.py ciphertext.txt 10 15 100GBP 999EUR
```

and the contents of the file are:

`ciphertext.txt:`
4E40B844800742DB4F9B8879E75727B244988FA23854AF93F3252EE07935FBB4

Note that the ciphertext is the same as in Examples 1 and 2, so the original plaintext is

Transfer 100GBP to account 1234.

Bytes 10 to 15 are the amount and the currency (100GBP), so the correct output in this example is a ciphertext that, when decrypted with the key that was used to encrypt `ciphertext.txt`, produces the plaintext

Transfer 999EUR to account 1234.

Note: You cannot use the encryption key in your code (the attacker does not have it!), but you can and you should use it when testing.

In your report, describe the main idea in your attack. Why does the attacker need to know the exact value of the original plaintext in bytes k1 to k2?

## Part C: Attacking the Stream Cipher – Confidentiality (10 marks)

In this part, you will again play the role of an attacker, only this time your objective is to *read* an intercepted ciphertext, rather than modify it.

### C1 – Implementation (8 marks)

You have intercepted two ciphertexts, A and B, of the same length. You happen to know the plaintext for A, and you also know that, because of a mistake by the sender, the same keystream bytes that were used to produce A were also used for B. Your objective is to find the plaintext that corresponds to B.
Your task is to write a program that takes as input

1. two ciphertexts, A and B, and
2. the plaintext for A

and produces as output the plaintext for B.

Your program should run using

`python partc.py` *ciphertextfileA ciphertextfileB plaintextfileA*

In your report, describe the main idea in your attack.

### C2 – Question (2 marks)

Answer the following question, looking back at the pseudocode in Part A: If you keep using successive bytes from the bytestream, are these bytes ever going to start repeating from the beginning in the same order? Mathematically, is there any number T such that the 1$^{st}$ byte in the byte stream is the same as the (T+1)th byte, the 2$^{nd}$ byte is the same as the (T+2)th byte, and so on? If such a T exists, it is called the *period* of the byte stream. Justify your response. There is no need for a formal proof or, if you claim that T exists, for an exact computation of T's value.