SCHOOL OF COMPUTER SCIENCE
COURSEWORK ASSESSMENT PROFORMA

MODULE & LECTURER: CM3110 Security, George Theodorakopoulos

DATE SET: 18 October 2016

SUBMISSION DATE: 25 November 2016, 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 provided specification of a particular cryptographic 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. Efficiency of the implementation
3. 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

You are given Python 3 code (file `cipher.py`) that implements a stream cipher. In the three parts of this coursework, you are asked to perform various attacks against this cipher.

How the cipher code works

The cipher code takes as input

1. An opcode character ‘e’ (for encryption) or ‘d’ (for decryption)
2. A file that contains the key (ASCII string)
3. A file that contains the input text (plaintext or ciphertext)

and encrypts or decrypts the input as appropriate, printing the output to stdout.

Regardless of whether it encrypts or decrypts, the cipher first converts the key from an ASCII string to a list of integers. This list of integers is then used to generate a keystream (a never-ending sequence of integers between 0 and 255 inclusive).

In encryption mode (opcode ‘e’), each keystream integer is XORed with a plaintext character that has been converted to an integer (ord() function). The resulting ciphertext integer is converted to **two** hexadecimal characters (0123456789ABCDEF), which are then written out. We need **two** hex characters per ciphertext integer, because the ciphertext integers are between 0 and 255, so we need **two** bytes (16 bits) to encode each.

In decryption mode (opcode ‘d’), each keystream integer is XORed with a ciphertext integer that corresponds to a **pair** of hexadecimal characters. The resulting integer is converted to a plaintext character (chr() function). The output is written out as an ASCII string.

Example 1 (encryption)

Assume you execute

```
python cipher.py e key.txt plaintext.txt
```

and the contents of the files are:

`key.txt`:
Secret key

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

Then, the correct output is
4E40B844800742DB4F9B8879E75727B244988FA23854AF93F3252EE07935FBB4

Part A: Brute-force attack
(10 marks: 8 for the code; 2 for the report)

In this part, your task is to implement a brute-force attack against a given ciphertext. You should recover both the key that was used and the plaintext that was encrypted. You should also measure the time it takes to perform the brute force attack.

A1 – Implementation

More specifically, you should write a program that takes as input a file that contains the input ciphertext and produces as output at least two lines: The first line should contain only the key, and the second line onwards should contain only the plaintext. Do not add any extra newlines to the plaintext.

Your program should run using

`python parta.py` *ciphertextfile*

Notes:

1. You can assume that the encryption key is 6 characters long. Only lowercase letters are allowed (a-z).
2. You can assume that the plaintext is valid English, i.e. it contains words that you can find in a dictionary.
3. The ciphertext inside the file is in the form of a string of hexadecimal digits 0123456789ABCDEF.

A2 – Running time evaluation

To evaluate the performance of the brute-force attack, 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
```

Note that the time you measure may be influenced by other processes running on your computer at the same time, so repeat the measurement 10 times and take the average. Evaluate the running time of the brute force attack when increasing the length of the encryption key to 7 characters and also to 8 characters.

In your report, describe the main components in your implementation of the attack. Your report should also include a characterization of any trend that you observe in the running time (i.e. How does the running time of the brute-force attack scale with the key size?).

Part B: Integrity Attack
(10 marks: 8 for the code; 2 for the report)

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 file that contains the 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 (Integrity 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 Example 1, 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 why you expect your attack to work, by referring to how stream ciphers work. Why does the attacker need to know the exact value of the original plaintext in bytes k1 to k2?

Part C: Confidentiality Attack
(10 marks: 8 for the code; 2 for the report)

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.

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 why you expect your attack to work, by referring to how stream ciphers work.