COSC2539: Security in Computing and IT
Assignment 1
Part 2 & 3
Cryptography
Semester A, 2016

Denis Rinfret

Learning Objectives

Learning Objective	RMIT Graduate Attribute	AQF LOD
Describe basic system security mechanisms	Innovative	K1, S1, S2, A1
Cryptography and cryptanalysis	Active and lifelong learners	K1, S1, S2, A1

Assignment to be done in teams of 2 students

Due: end of Week 7, Sunday April 10, 2016 at 23:59.

Objectives

The main objective of this assignment is to understand the concepts of encryption, decryption and cryptanalysis. Part 1 of this assignment has to be done in class. Instructions for part 1 will be provided separately.

You will have to decrypt a number of cipher text messages, sometimes knowing which algorithm or which key has been used, sometimes **not** knowing which algorithm **and** which key has been used. In order to do this, you will need to understand many classic encryption algorithms and you will need to write programs to help you decrypt ciphertext messages. The following programming languages are allowed: Python, C and Java. For C, you cannot use Visual Studio or something similar, but you can use `gcc` or `clang` in a Linux environment similar to the one used for the Programming Techniques course. For Java, using NetBeans or IntelliJ is acceptable, and for python, any developing environment is OK. If you want to use another programming language and/or programming environment not listed above, you need to have your choice approved before the submission deadline.

Assume all messages use this alphabet:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ .,:;()-!?$'"\n0123456789
```

Please note that there is a space between the `Z` and the `.` in the alphabet given above. Also, `\n` represents only one character and has the usual meaning of a *new line* character. The 2 characters between the dollar sign `$` and the new line character are the *single quote* and the *double quote* characters.

Part 2: Implementation of Ciphers

Write programs to implement the four ciphers given next.

Ceasar Cipher

Write a program to implement the *Ceasar cipher*. The key should be an integer. Your program should be called `ceasar.py` if using Python, `ceasar` if using C, or `Ceasar` if using Java, and take the following command line arguments:

- `e` or `d`: encrypt or decrypt mode.
- `inputfilename`: the name of the input file. If in encrypt mode, the input file contains the plaintext, and if in decrypt mode, the input file contains the ciphertext.
- `key`: an integer.

If using Java, you should invoke your program like this:

```
java Ceasar e msg.txt 12
```

or

```
java Ceasar d msg.enc 12
```

For Python and C, replace `java Ceasar` with, respectively, `python ceasar.py` or `ceasar` (or `./ceasar`). The output of your program will either be ciphertext (if in encrypt mode) or plaintext (if in decrypt mode), and should be printed on the standard output.

Random Substitution Cipher

Write a program to implement a *random substitution cipher*. The key should be a random bijective mapping of the alphabet. Your program should be called `subst.py` if using Python, `subst` if using C, or `Subst` if using Java, and take the following command line arguments:

- `e` or `d` or `g`: encrypt or decrypt or generate mode.
- `inputfilename`: the name of the input file. If in encrypt mode, the input file contains the plaintext, and if in decrypt mode, the input file contains the ciphertext. If in generate mode, the input file name will be ignored.
- `keyfilename`: the name of the file containing the key. If in generate mode, the key will be written into that file name. Otherwise in encrypt and decrypt mode, the key will be read from that file.

You should invoke your program in a similar way to the Ceasar cipher, but by replacing the program names of course. The output of your program will either be ciphertext (if in encrypt mode) or plaintext (if in decrypt mode) and should be printed on the standard output. If in generate mode, there will be no output on the standard output.

Transposition Cipher

Write a program to implement the *columnar transposition cipher*. The key should be an integer, the number of columns in the table. Your program should be called `coltrans.py` if using Python, `coltrans` if using C, or `ColTrans` if using Java, and take the following command line arguments:

- `e` or `d`: encrypt or decrypt mode.
- `inputfilename`: the name of the input file. If in encrypt mode, the input file contains the plaintext, and if in decrypt mode, the input file contains the ciphertext.

- `key`: an integer.

You should invoke your program in a similar way to the Ceasar cipher, but by replacing the program names of course. The output of your program will either be ciphertext (if in encrypt mode) or plaintext (if in decrypt mode), and should be printed on the standard output.

Vernam Cipher

Write a program to implement a *Vernam cipher*, a kind of *One-Time-Pad (OTP).* The key should be a sequence of random letters of the alphabet to be added to the plaintext or subtracted from the ciphertext. Adding or subtracting letters really mean adding or subtracting the letter numerical codes. Your program should be called `vernam.py` if using Python, `vernam` if using C, or `Vernam` if using Java, and take the following command line arguments:

- `e` or `d` or `g`: encrypt or decrypt or generate mode.
- `inputfilename`: the name of the input file. If in encrypt mode, the input file contains the plaintext, and if in decrypt mode, the input file contains the ciphertext. If in generate mode, the input file name will be ignored.
- `keyfilename`: the name of the file containing the key. If in generate mode, the key will be written into that file name. Otherwise in encrypt and decrypt mode, the key will be read from that file.
- `n`: an integer. In generate mode, `n` is the length of the key to generate. In encrypt and decrypt mode, `n` is the offset in the key file, i.e. it is the position of the first random letter in the key to use in the addition or subtraction of letters.

You should invoke your program in a similar way to the Ceasar cipher, but by replacing the program names of course. The output of your program will either be ciphertext (if in encrypt mode) or plaintext (if in decrypt mode) and should be printed on the standard output. If in generate mode, there will be no output on the standard output.

Part 3: Decrypting Messages

Decrypt the following messages. All of these messages are available in separate text files, inside the file named `COSC2451_A1_msgs_2016A.7z`. Using separate text files will help avoid any issues with counting the number of spaces and new line characters correctly. To avoid issues with some text editors occasionally adding new line characters at the end of a file and other issues related to counting characters, the messages to decrypt in the text files have been enclosed between the characters `>` and `<`. These 2 characters are not part the message since they are not part of the alphabet. You should skip these 2 characters when decrypting the messages.

Since some information is missing (for example: missing key), you cannot simply run your program from the previous part to decrypt the messages. You have to do some cryptanalysis to be able to decrypt the messages. This is why it is not sufficient to give only the final decrypted message as a final answer: you also have to show your work for every question. Showing your work means in this case:

- explaining how you used your programs from the previous parts to find the answer, or
- explaining how you modified your programs to find the answer, or
- explaining which cryptanalysis techniques you used to help you find the answer, or
- anything else showing how you came up with the answer.

Messages to Decrypt

1. File: `msg1.enc`, algorithm: `Ceasar`, key: `unknown`.
2. File: `msg2.enc`, algorithm: `Columnar Transposition`, key: `unknown`.
3. File: `msg3.enc`, algorithm: `unknown`, key: `23`.
4. File: `msg4.enc`, algorithm: `Random Substitution`, key: `unknown`.
5. File: `msg5.enc`, algorithm: `Vernam (addition)`, key: `unknown`.

   Hint: `msg5.enc` is related to message 2 in some way.
6. File: `msg6.enc`, algorithm: `Columnar Transposition` first, then `Ceasar`, key: `unknown, but same key for both encryption steps`.
7. File: `msg7.enc`, algorithm: `unknown` but **not** `Vernam`, key: `unknown`.

Submission and Marking

In you Google Drive, create a folder named `COSC2539_A1_s1234567_s1234568` and put all your files in it. Create a Google Docs inside your folder, named `COSC2539_A1_s1234567_s1234568_part3`. This document should contain the decrypted messages and the explanations of how you decrypted them. Do not include source code in these documents. Create an archive (either in `zip` or `7z` format) containing your source code. Name your archive `COSC2539_A1_s1234567_s1234568_src.zip` or the equivalent in 7z format. Don't forget to replace `s1234567` and `s1234568` with the student IDs of students in the group.

Share your `COSC2539_A1_s1234567_s1234568` folder with the lecturer in *edit* mode. Don't wait until the last day to share your folder. You can share it early, and keep adding files to it or making updates to your files.

Do not modify your documents after the due date. The usual 10%/day, maximum 5 late days will apply. This assignment is worth 20% of your total mark for the course. Part 1 is worth 5%, part 2 is worth 8%, and part 3 is worth 7%.