Computer Security (300569)

Assignment

For this assignment, you are given two options. Option 1 involves writing a literature review on one of the major areas of computer security, and Option 2 involves writing a program for decryption of Caesar and columnar ciphers. Each option is worth the same value of 10 marks that counts into the final assessment of this unit. You should pick **one and only one** option below. Working on both options will not give you extra credits. In this case, only your programming assignment (option 2) will be taken for marking. Hence you are responsible for making the right choice for yourself at the beginning based on your experience, expertise and preference. Details of both options can be found below, including the description of tasks and the marking criteria.

Option 1. Literature Review (15 Marks)

Task Description

You are required to write an open-topic review essay on one of the major areas of computer security outlined below

1. Program security – discussed in Chapter 3 of textbook
2. Web user security – discussed in Chapter 4 of textbook
3. Web application security – discussed in Lecture 6
4. Operating system security – discussed in Chapter 5 of textbook
5. Network security – discussed in Chapter 6 of textbook
6. Database security – discussed in Chapter 7 of textbook

…

You can also choose to write in other areas of computer security subject to the approval of unit coordinator. Your essay should focus on one and only one area in security, and provide an in-depth and comprehensive discussion on important issues in the area of your focus. Depth is much more important than coverage for this assignment. Hence concentrating on a small topic with your own insight is much better than briefly touching everything superficially. For example, a broad review essay on program security and operating system security together is probably not as good as review on program security alone, provided that they are about the same length. A review essay focusing on buffer overflow issues in program security is even better than a review essay on program security in general.

Doing research is an important step for writing a literature review, where everything starts with reading and finding ideas from the reading materials. As a starting point, you can read the relevant chapters in the textbook as well as the lecture notes corresponding to the area of focus for your review article. However, to gain more understanding and insight, you should not confine your reading to the teaching materials alone. Instead, you need to reach out and look further and deeper into the issues of interest by checking out and delving into relevant literatures, including but

not limited to, published conference papers and journal articles, reference books and possibly online resources. For example, if you are to write a literature review on buffer overflow program errors, which is discussed between pages 134 and 152 in Section 3.1 of the textbook. You can find a lot more information from reference materials [ALE96], [MUD95], [PIN04] cited on those pages, which can be located by referring to the bibliography section in the end of the book. Each article referenced here also contains more references where you can look for further information. This is especially true with [MUD95] and [PIN04], two review articles on buffer overflow. Similar readings and research can be done with other topics of your choice.

Your review essay should be well structured into sections and paragraphs. Each section should come with a heading and focus on a single main point, for e.g. introduction, aims, issues, models, solutions, summary, etc. A section may further be divided into paragraphs depending on the points that they cover. You should also include a bibliography or reference section in the end of your essay, which contains references to all articles, books and online links that have been drawn upon and cited in the main essay. Your essay should contain a minimum of **2,000** words and a maximum of **5,000** words excluding the reference section in the end. Moreover, it is most important you include in your essay not only facts and findings from the textbook and other references but also your own opinions and understandings of the issues discussed in your essay that reflects your critical thinking on the topics covered.

Due to the amount of literature reading and analytic writing needed, you are advised to start early and proceed in an iterative cycle, much like working on a large software project. Your first draft may not cover all the points and be free from errors, but should contain all the critical points and the backbone of the whole essay. You can then improve upon your initial draft by adding more points and refining the existing content. This can be repeated for more iterations, and each iteration you will have an improved version. This approach will enable you to maintain efficiency in writing and maximize your productivity, and importantly help minimize last minute shock which often results from procrastination of everything.

Marking Criteria

Your final submission will be marked against the criteria listed on the following page with a total of 100 marks. Fractional marks may be awarded for each criterion. Your final mark for the assignment is 0.15 times the raw marks you receive.

A Note on original work

All submissions will be examined by the turnitin software (http://turnitin.com) which was integrated into the vUWS system for the detection of possible plagiarism. Any submission that fails the turnitin examination will be given **ZERO** mark and may lead to further investigation on academic misconduct.

Marking Criteria	Level	Marks Awarded	Description
Objective (10 marks)	Good	10	The essay has clear aim and is well focused on a single important issue in security
	Average	5	Mainly focusing on a single area of security, with some deviations and irrelevant discussions
	Poor	0	Not focusing on a single area of security
Presentation (20 marks)	Good	20	The essay reads very well, easy to understand and free from grammatical errors, spelling mistakes and use of colloquial English
	Average	10	The essay reads well in general, though there might be certain degree of grammatical mistakes and lack of clarity in presentation
	Poor	0	The essay is poorly written and can hardly be understood by readers
Originality (20 marks)	Good	20	The essay contains some original thoughts and analysis from the author alongside discussions on facts and observations
	Average	10	There are some original points made by the author and not sufficient enough
	Poor	0	The essay is basically a repetition of what is being discussed in the literature and there is virtually no insight included or input made on the author's part
Technical quality (30 marks)	Excellent	30	The essay is up to a high calibre of technicality and reflects a high level of critical thinking for the author.
	Good	20	The essay has good quality in general, but contains some errors and inaccuracies in the discussion.
	Average	10	The essay does not meet the requirement on minimum word count, and contains many errors and inaccuracies in the discussion.
	Poor	0	The essay has low technical quality and virtually no scientific value.
Organisation (10 mark)	Good	10	The essay is well structured and preferably separated into sections. Each section should cover one major point and could be further divided into paragraphs for clarity.
	Average	5	The essay is structured into sections and paragraphs, although not all of them make sense.
	Poor	0	The essay is poorly organised.
References (10 mark)	Sufficient	10	Contains sufficient references in bibliography section and references are cited the main text
	Inadequate	5	References included but are insufficient; or references not linked in the main text
	Absent	0	Lacking references

Option 2. Programming Task (10 Marks)

Task Description

You are required to create a program to *automatically* decrypt cipher text messages encoded using one of the four encryption algorithms below.

- Caesar cipher
- Columnar transposition cipher
- Modified Caesar cipher
- Diagonal columnar cipher

whereas the first two algorithms are simple substitution and transposition ciphers discussed in lecture 3. The last two ciphers are briefly described below.

Modified Caesar cipher

This is a substitution cipher which modifies the Caesar cipher in two steps. In the first step, a word is chosen and inserted into the beginning of the alphabet. Duplicated characters are deleted to keep the alphabet to 26 letters. For example, with the word "hello" (for simplicity assume everything is lower case), the alphabet is modified by

h	e	l	o	a	b	c	d	f	g	i	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

Note that except the letters appearing in the word "hello" ('h','e','l' and 'o'), the remaining letters are in the original alphabetic order.

In the second step, the whole modified alphabet is shifted circularly in one direction with fixed step size. This is the same step as the Caesar cipher. An example of right shift with step size 3 for the above alphabet is shown below

o	a	b	c	d	f	g	i	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z	h	e	l

The random word added to the beginning of the alphabet in the first step adds more perturbation to the substitution scheme, increasing the search space for breaking the cipher. To achieve best result, the word used should be moderately long and the step size should not equal the number of unique letters in the word.

To break the cipher, one needs to determine both the word inserted in step 1 and the step size for shifting of alphabet in step 2. This can be done by exhaustive or brute force search by checking all possible words and shifts. Note that the step size could be either positive (right shift) or negative (left shift) for different directions of shift for the alphabet. To make it simple, we only used a few technical words for altering the alphabet in step 1, which will be provided with this assignment in a text file.

Diagonal transposition cipher

Diagonal transposition cipher is a transposition cipher similar to the columnar transposition. Firstly, it rearranges the plaintext message into a block of fixed number of columns by fitting the input text on each row of the block consecutively. This is the same step as columnar transposition. However, different from columnar transposition which produces the ciphertext message by scanning the columns of the rearranged

text block, the diagonal transposition cipher produces the ciphertext message by scanning along the diagonal directions. An example of diagonal transposition is shown in the figure below with two variants for diagonal and reverse diagonal ciphers, where the red dotted lines show the scanlines along which the ciphertext messages are produced. For the diagonal version, it starts from the bottom-left corner and scans the diagonal lines sweeping from top-left to bottom-right corners. For the reverse diagonal version, it starts from the top-left corner and scans the reverse diagonal lines sweeping from top-right to bottom-left corners. With the illustrated example, for the plaintext message "`CRYPT OGRAP HYISF UN`" wrapped into 5 columns, the ciphertext messages produced by diagonal transformation ciphers are "`uhnoy cgirr syafp pt`" for the diagonal version and "`corhg ynyrp niats pt`" for the reverse diagonal version.

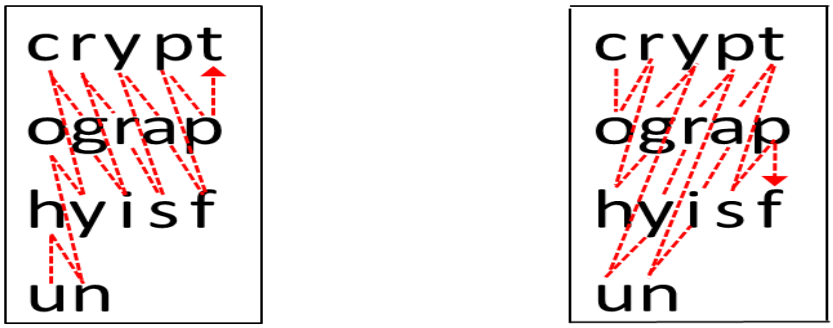

Figure 1. Illustration of the diagonal transposition cipher: left: diagonal version; right: reverse diagonal version.

Program Flow and I/O

You will be provided with a ciphertext file. Each line of the file contains a ciphertext message to be hacked. Unlike the decryption exercises you did in lab 3, you are not given any information on the types of cipher and the keys used for encryption. You need to determine these and obtain the decrypted message ***automatically*** with your program. However, the plaintext messages are meaningful English expressions containing alphabetical letters only. An English dictionary is also provided in a text file format for your hacking program. You need to develop an algorithm to effectively hack the ciphertext messages in the input file with high accuracy.

For a ciphertext message on each line of the input file, your program should output the following information in comma separated fields,

```
PlainText, CipherType, KeyVal
```

where `PlainText` is the estimated plaintext message, `CipherType` is the type of the cipher used for encryption, which can take any of the four values 'C', 'T', 'M', 'D', corresponding to four different ciphers used, namely Caesar, columnar transposition, modified Caesar, diagonal transformation. `KeyVal` represents the key value of the encryption algorithm, which is the number of alphabetical shifts (positive for right shift and negative for left shift) for Caesar and modified Caesar ciphers, number of

columns for rearrangement of the text block for columnar transposition and diagonal transposition (positive for diagonal version and negative for reverse diagonal version) ciphers. The modified Caesar has one more key, the word inserted at the beginning of the alphabet for the substitution scheme. Some examples are shown below

Input: `dgyzr kcoez nkdo`
Output: `TWOPH ASEUP DATE,C,10`
(Caesar cipher - right shift the alphabet 10 times)

Input: `saupw nrsed cesht oi`
Output: `SWEET ANDSO URCHI PS,T,5`
(Columar transposition - fold the text in 5 columns)

Input: `bxsyx rnppy hydcy`
Output: `MAKEA DIFFE RENCE,M,cryptanalysis,-2`
(Modified Caesar cipher - insert cryptanalysis in the alphabet, remove duplicates and left shift the alphabet 2 times)

Input: `meanr radsk lcakh s`
Output: `MARKE RSAND CHALK S,D,-4`
(Diagonal transposition - fold the text in 4 columns, scan in reverse diagonal direction)

Following the same convention in Lab 3, the ciphertext messages are organised in groups of five lowercase letters and the output plaintext messages should be organised in groups of five uppercase letters.

Ideally, your program should read the ciphertext messages from the input file, produce decryption results in above format, and output the results line by line to an output text file. You can also display your results at the standard output. Alternatively, you could choose to read ciphertext messages from the keyboard. There'll be some reduction of mark though for doing so. Apart from keyboard input, your program should not require any user intervention to perform the decryption steps.

Programming Language and Supplementary Material

There is no restriction on the programming language you use to write the above program. You can choose whichever language you feel most comfortable with. However, if the programming language you pick is not from one of the following - `C, C++, Java, Python,` you will have to provide additional information on how to compile and run the program. You only have to upload the source code for your submission. There is no need to upload the exe or class files.

For this assignment, we provide you with an English dictionary in text file format, where each line contains a word in the dictionary, an example input file and the corresponding output file containing decryption results for ciphertext messages in the input file. We also provide a list of technical words in a separate file. These words were used to shift the alphabet for all ciphertext messages produced by the modified Caesar cipher. You can develop your program and use the provided test files to verify your program. Note that you need to make sure your program and algorithm are general enough as we will use different test files for marking purpose.

Marking Criteria

The marking of programming assignment is broken into two separate criteria – the functionality of the program, as well as the operation and style of the program. The full achievable mark is 100, and marks for each section are described below.

I. Functionality (80 marks)

This is the single most important criterion. The program has to be functional in order to receive high marks here. Your program will be marked against each of the following criterion ranked in increasing level of difficulty. Fractional marks may be awarded for each criterion and your final mark for this section is the sum of marks for individual criterion.

Manual decryption (40 marks)

Your program is supposed to perform automatic decryption of input plaintext messages. To achieve that, the first step is being able to decrypt a message given the cipher and key. You will get 10 marks for each of the four ciphering algorithms that your program can handle manually.

Automation (30 marks)

You will be marked against level and accuracy of automatic decryption. To achieve full mark here, your program needs to automate everything and achieve high decryption accuracy. Marks will be given in steps of 5 marks, i.e. 10, 15, 20, 25, 30 marks corresponding to >20%, >40%, >60%, >80%, >95% accuracy rates respectively. The accuracy rate is calculated based on the percentage of correct decryptions achieved for all ciphertext messages in the input file. For example, if the input file contains 100 messages and your program can correctly decrypt 70 of them, this is 70% accuracy rate and thus will give you 20 marks for this section.

Your marks will be reduced by half if your program can break the ciphertext message semi-automatically with some partial information given (e.g. knowing the ciphering algorithm but not the key value, and vice versa).

Error tolerance (10 marks)

The test file used for marking contains some errors, i.e. some of the cipher text messages were encrypted from plaintext messages with spelling mistakes. Your program should also be able to correctly handle these instances.

Compiling and running

This is a pre-requisite of functionality. You need to ensure that your submitted source code can compile and run successfully. Otherwise you will get 0 mark here.

II. Operation and style (20 marks)

This has to do with the program in operation. Marks for this section are worked out on a deduction basis. If your program has any of the issues below, marks will be deduced

by the corresponding value displayed alongside the criterion. Note that the maximum deduction is 20 marks. If your program has many issues below and receives more than 20 marks deduction, you will lose all 20 marks in this section but not more.

Poor usablity **(-5 marks)**

Your program should be easy to use. It should have proper display messages for user interaction, e.g. asking the user to enter the name of the input and output files, or an input ciphertext message if your program does not support reading from the file. If your program receives filenames through command line arguments, a help message should be displayed if the user runs the program without arguments.

Inadequate error handling **(-5 marks)**

Your program should check whether the input cipher text messages are correctly formatted and ensure that only letters are received as input. In case of incorrect input, you should return an error message (e.g. 'INVALID INPUT') at output. If your program receives input messages from the file, it should check whether the file exists and handle non-existence errors.

Inconsistent display format of output results **(-5 marks)**

Your program should produce the output results in the same format described in the examples above. There is an up-to 5 mark deduction for inconsistent format.

Program does not support file input mode **(-15 marks)**

If your program does not support reading from the file and decrypting line by line, there is a 15 mark deduction even if the rest of the program functions correctly.

Poor readability **(-5 marks)**

This involves poor indentation, poor naming of variables and functions, declaration of variables that are not used in the program, etc.

Inadequate comments **(-5 marks)**

You need to briefly write the main idea of your decryption algorithm in the comments of your program. You may also need to include comments where-ever your code may cause confusion and requires extra explanation.

Poor modularity **(-3 marks)**

No or limited use of subroutines or functions. Everything or majority of the code is written in the main function.

A Note on original work

All submissions will be examined by the MOSS software developed by Stanford University (http://theory.stanford.edu/~aiken/moss/) for possible code plagiarism. Any submission that fails the MOSS examination will be given **ZERO** mark and may lead to further investigation on academic misconduct.

Submission Instructions:

Please take some time to read the important submission information below. Failure to comply with the instructions will incur a 5% mark reduction of your assignment.

Whichever option you choose, you need to submit your assignment through the online submission link on the vUWS site of this unit. You can submit your work multiple times. However, only one file can be submitted with each submission and your most recent submission will be taken for marking. Thus, **you need to include all your submission files in one zip file.** The zip file should include the completed cover sheet and the essay (in pdf or word document format) for Option 1 and the cover sheet plus source code files and other files for Option 2. For Option 1, your essay can be in either pdf or word document format.

Make sure you rename your submission file as **`SID_SNAME.zip`**, where `SID` should be substituted by your student number, `SNAME` should be substituted by your surname. Other zip file formats are acceptable, but you have to change the extension (e.g. tar.gz for Tar Gzip file, 7z for 7-Zip file).

For example, if a student's surname is Smith and his/her student number is 18000000, then the submission file should be renamed to `18000000_Smith.zip`. If there is apostrophe, dash or space in your surname, you should remove them in the filename, so O'Neal->ONeal, al-Assad->alAssad, Del Piero->DelPiero.

Submission deadline is **17:00 Tuesday, 13 October 2015**. The deadline is firm and no extension will be given.