Assessed Coursework 1 — Symmetric Encryption

Avoiding Plagiarism

In addition to the slides on plagiarism that were given in the first lecture, I have also uploaded a set of slides, entitled "Academic Misconduct" with more details on plagiarism as well as other forms of academic misconduct as well as a few tips on how to avoid them. You can find the slides in the same folder as this description. Notable things to take into consideration:

- Coursework reports must be written in your own words and any code in your coursework must be your own code. If some text or code in the coursework has been taken from other sources, these sources must be properly referenced.
- Failure to reference work that has been obtained from other sources or to copy the words and/or code of another student is plagiarism and if detected, this will be reported to the School's Discipline Committee. If a student is found guilty of plagiarism, the penalty could involve voiding the course.
- Students must never give hard or soft copies of their coursework reports or code to another student. Students must always refuse any request from another student for a copy of their report and/or code.
- Sharing a coursework report and/or code with another student is collusion, and if detected, this will be reported to the School's Discipline Committee. If found guilty of collusion, the penalty could involve voiding the course.
- And remember: the consequences of taking unacceptable short cuts in coursework are much worse than getting a bad mark (or even no marks) on a piece of coursework. There has been one case this year where a student was awarded on Ordinary degree (rather than an Honours degree) because of the sanction imposed by the University's Discipline Committee. The offence was plagiarism of coursework.
- Further information on academic misconduct can be found in: https://www.hw.ac.uk/students/doc/discguidelines.pdf

Your coursework submissions will be automatically checked for plagiarism.

1 Overview

The learning objective of this coursework is for you to become familiar with the concepts in the *symmetric encryption*. After finishing the coursework, you should be able to gain first-hand experience on encryption

algorithms, encryption modes, and initialisation vectors (IV). Moreover, you will be able to use tools and write simple scripts or programs to encrypt/decrypt messages.

This is an independent coursework and you are expected to work **on your own** to complete the coursework tasks. You are also required to submit a coursework report. At the end of each task, the information that you need to provide to answer the task is indicated. For further information, including submission, see Section 4.

2 Lab Environment for Coursework

The software needed for this lab is installed on all machines in ITLab 1. You should also be able to use your own laptop, though you may need to download or update certain tools, including an application for editing hex-files. Make sure you are using a CentOS Linux for the lab. Using other Linux/Unix distributions (including MacOS) might result in unpredictable behaviour.

OpenSSL. In this lab, we will use `openssl` commands and libraries, which are installed on all Linux lab machines. To confirm that you have `openssl` on your personal machine, your can type either of the following commands:

```
   %  openssl version
OpenSSL 1.0.2k-fips 26 Jan 2017

   %  which openssl
/usr/bin/openssl
```

Editing hex files In some of the tasks, you'll need to modify hex files.

`emacs`*:* To edit hex-files, you can use the installed version of the `emacs` editor. Start it like this:

```
   % emacs
```

In the editor type `M-x hexl-find-file` and enter the file name. You will see a screen with the hex-contents of the file. Use your cursor to move to a position that you want to edit. To insert a hex value do `C-M-x <value>` or use the `Hexl menu`. Use `C-x C-s` to save the file and `C-x C-c` to exit the editor.

`ghex2`*:* An easy-to-use graphical hex editor, `ghex2`, is available on all Linux machines running the gnome desktop. You can start it by typing `ghex2` from the command-line or selecting it from the graphics section of the system menus.

`shed`*:* A simple hex editor, `shed`, is available on the Linux machines, if you have run `source f21cn.sh`. Call it with the filename to edit as argument, e.g. `shed file.bin`. Use your cursor to move to a file position and the SPACE key insert a value. Prefix with a number to edit that many bytes. Its full path is `/usr/bin/shed`. Documentation is available via a man-page, i.e. `man shed`.

GNU/Linux coreutils: Alternatively, you can also use the `coreutils` on any GNU/Linux system. This way you can avoid any manual editing of the files. Type `info coreutils` to get information about the entire toolset, browse the index and look for the tools for outputting parts of a file.

3 Coursework Tasks

3.1 Task 1: Frequency Analysis: Monoalphabetic Substitution Cipher

It is well-known that monoalphabetic substitution cipher is not secure because it can be subjected to frequency analysis. In this task, you are given a ciphertext that is encrypted using a monoalphabetic cipher; namely, each letter in the original text is replaced by another letter, where the replacement does not vary (i.e., a letter is always replaced by the same letter during the encryption). However, note that the shift amount for each letter varies (i.e., amongst the alphabet of plaintext characters, the shift amount for the encryption varies). Your job is to find out the original text using frequency analysis. It is known that the original text is an English article. Download the ciphertext file `ciphertext-task1.txt` from Vision.

Note that the plaintext was modified before encrypting, so that all upper case letters have been converted to lower cases, all the punctuations and numbers were removed. However, the spaces between words were maintained in the plaintext (and appear as corresponding spaces in the ciphertext), so that you can still see the boundaries of the words in the ciphertext.

Your task is to use the frequency analysis to figure out the **encryption key** and the **original plaintext**. Using the frequency analysis, you can find out the plaintext for some of the ciphertext characters quite easily. For those characters, you may want to change them back to its plaintext, as you may be able to get more clues. It is better to use capital letters for plaintext, so for the same letter, we know which is plaintext and which is ciphertext. You can use the tr command to do this. For example, in the following, we replace letters `a`, `e`, and `t` in `in.txt` with letters `X`, `G`, `E`, respectively; the results are saved in `out.txt`.

```
$ tr aet XGE < in.txt > out.txt
```

There are many online resources that you can use. We list four useful links in the following:

- `http://www.richkni.co.uk/php/crypta/freq.php`: This website can produce the statistics for a ciphertext, including the single-letter frequencies, bigram frequencies (2-letter sequence), and trigram frequencies (3-letter sequence), etc.
- `https://en.wikipedia.org/wiki/Frequency_analysis`: This Wikipedia page provides frequencies for a typical English plaintext.
- `https://en.wikipedia.org/wiki/Bigram`: Bigram frequency.
- `https://en.wikipedia.org/wiki/Trigram`: Trigram frequency.

In your report, include

- A short description (no more than a page) that explains what you did in your attempt to discover the encryption key and the original plaintext. This description should indicate how you decrypted various ciphertext letters, e.g., whether you used the single letter frequency analysis, bigram or trigram frequencies, or some other reason to guess a particular plaintext character.
- The encryption key. You should write this as an encryption table that shows the mapping of each plaintext letter to its corresponding ciphertext letter.
- The plaintext, included as an appendix in your report.

3.2 Task 2: Symmetric encryption: Ciphers and modes

In this task, we will use various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

```
% openssl enc ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-aes-128-cfb`, `-bf-cbc`, etc. In this task, **you should use at least 3 different ciphers with each of three different modes to encrypt a file of your choosing (thus, you should have 9 total encryption examples)**. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the `openssl enc` command in the following:

```
-in <file>     input file
-out <file>    output file
-e             encrypt
-d             decrypt
-K/-iv         key/iv in hex, with length appropriate to cipher
-base64        base64 encode after encryption, or before decryption
               Otherwise, input file is encrypted to binary output
```

In your report, include

- A screenshot(s) that shows your execution of the nine encryption commands If you choose to write a script to execute the commands, include the script along with a screenshot of the script running – your script should write useful statements to the terminal in this case in order to indicate which command is running.
- For each of the nine encryption commands, please indicate which part of the command refers to the cipher name, and which part refers to the mode of operation that is used with the cipher.

3.3 Task 3: Encryption Mode — Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1,000 bytes long.
2. Encrypt the file using the AES-128 cipher, separately for each of the encryption modes of operation: ECB, CBC, CFB, and OFB.
3. Unfortunately, a **single bit** of the 55th byte in the encrypted file got corrupted. You can achieve this corruption by editing the hex-file for each of the four encryptions of the text file. modifying the 55th byte in this file using `emacs`.
4. Decrypt the corrupted ciphertext file using the correct key and initialisation vector (IV).

In your report, include

- An initial hypothesis of what you anticipate the result will be for each mode of operation (**do this before you run the commands**). In other words, how many bits of information do you anticipate that you will be able to recover from the corrupted files for each of the modes? Your explanation should use the equations for each mode (as discussed in our lecture) to justify your reasoning. (NB: You won't lose marks if your initial hypotheses were wrong. Marks are allocated for using sound reasoning.)
- Screenshots to demonstrate your encryption, file modifications, and decryption for each of the four modes of operation.
- An explantion of the results, and whether your results differ from your hypotheses. Your explanations should use the equations for each mode to justify your reasoning, should compare the results between the modes, and should discuss the impact of the corrupted ciphertext on the plaintext file in terms of the number of bits/bytes, and in terms of cipher blocks, impacted.

3.4 Task 4: Initial Vector (IV)

Most of the encryption modes require an initial vector (IV). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode. The objective of this task is to help you to understand the problems if an IV is not selected properly.

One may argue that if the plaintext does not repeat, using the same IV is safe. Let us look at the Output Feedback (OFB) mode. Assume that the attacker gets hold of a plaintext (`P1`) and a ciphertext (`C1`) , can he/she decrypt other encrypted messages if the IV is always the same? You are given the following information, please try to figure out the actual content of `P2` based on `C2`, `P1`, and `C1`.

```
Plaintext   (P1):  This is a known message!
Ciphertext  (C1):  a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

Plaintext   (P2):  (unknown to you)
Ciphertext  (C2):  bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
```

The attack used in this experiment is a *known-plaintext attack*. Note that you might find a site such as `http://xor.pw/` useful for working through this task.

In your report, include

- An explanation of how you attempted to determine the value of the plaintext `P2`. Your explanation should use the equations for OFB mode, as discussed in our lecture.
- The value for P2, in ASCII.

3.5 Task 5: Known-plaintext attack

Download the English word list (`words.txt`), plaintext (`plain.txt`) and ciphertext files for this task from Vision. Each student is assigned a different ciphertext file, for which a different key was used for encryption. The instructions to obtain your ciphertext file will be made available on Vision.

You know that `aes-128-cbc` was used to generate the ciphertext from the plaintext. **No salting** was used during encryption. You also know that **no IV was specified during encryption**. Another clue is that the key, used to encrypt this plaintext, is an English word shorter than 16 characters; the word that can be found from a typical English dictionary (we use the English word list that you downloaded from Vision to

represent this dictionary). Your goal is to write a program to find out this key, using the downloaded English word list, and plaintext and ciphertext files. For this task you will use the commandline tools provided by `openssl` to encrypt and decrypt messages in order to perform a *brute-force* determination of the key.

In your report, include

- A short description (no more than a page) that explains what you did in your attempt to discover the encryption key. Your description should talk about the key components of your script, and should explain how may have validated the correct operation of key components.
- A copy of your script. Your script should be well commented to explain what is happening in the script. Your script should also be flexible (e.g., using variables and not hard-coded values, and taking options from the command line), efficient (e.g., finds the key in the least number of steps) and clearly presents output (e.g., redirects unnecessary errors).
- A screen shot(s) to show the execution of your script.

Using shell scripting and command line openssl

Use the `openssl` commands for en-/de-cryption in a shell script to determine the key that was used to encode the file. As an example of how to process the word list file, check the line count script on Vision (line_count.sh) which reads text from a file and counts the number of lines. You will have to replace the body of the loop by calls to the appropriate `openssl` command and checking the result file. Hans-Wolfgang Loidl has kindly provided a screen-cast explaining the steps of the line count script available at the following: http://www.macs.hw.ac.uk/~hwloidl/Courses/LinuxIntro/

Hint: to speed up development, test your script on a self encrypted, short text, and use a much shorter word list; once your solution works for your own text, run it on the given text and the full word list.

4 Submission

You should submit a report to describe what you have done and what you have observed. With each task above, some guidance is provided on what to include in your report. You should also explain any observations that are interesting or surprising. Document any difficulties that you met while doing this exercise.

In your report, you need to answer all the points listed for each task. Structure the report in sections, addressing each of the tasks. Include an introduction section, discussing what you expect to learn from the assignment and clarifying which assumptions you made in performing the tasks. Include a summary/conclusion section, where you discuss whether your expectations were met, highlighting issues of particular importance, and suggesting further work, i.e. what kind of exercise could be done next building on the material from this assignment. In writing-up the work you performed on the above tasks, make sure to assess the strengths and weaknesses of the ciphers and modes used in for this assignment.

The **deadline for submitting the report is 11:59pm on Wednesday 10^{th} of October, 2018**. You should submit an electronic version via Vision: one pdf file for the report and one file for the program sources. Your source code has to be **one .txt, .pdf, .doc(x) or .odt** file, **do not submit a zipped file** for this. Create the one source code file by copy-pasting your codes from the different sections into it.

You should expect **feedback on your submitted coursework** by Wednesday 31^{th} of October, 2018.