Symmetric Encryption Lab

Contents

1 Aims & Objectives

The aim of this lab is to give students a hands-on experience of using OpenSSL command-line and C libraries, and symmetric encryption (i.e. stream and block) ciphers. This lab is *assessed* and consists of **7 tasks**. All tasks must be answered. By finishing the lab, students will learn how to use OpenSSL to perform encryption using different ciphers, the different modes of encryption, and the need for padding for some ciphers. Also, the lab will help students learn to practice how to cryptanalyze some ciphers. Additionally, the students will learn the requirements of post-quantum secure ciphers compared to classically-secure ciphers. Special attention should be paid to Section 3 and the What to Submit subsections of each task which list what exactly needs to be submitted for this lab.

1.1 Optional Additional Reading Resources

- Chapter 21 of the SEED Book, Computer & Internet Security: A Hands-on Approach, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.
- Chapters 6 & 7 of the Handbook of Applied Cryptography, 5th Edition, by Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. See details at `http://cacr.uwaterloo.ca/hac`.
- OpenSSL command-line tool `https://www.openssl.org/docs/man1.0.2/man1/openssl.html`.
- OpenSSL C library `https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html`.

2 Lab Tasks

This section contains the specification of the required tasks.

2.1 Task 1: Crypanalysis of Substitution Cipher

This task requires you to decrypt a ciphertext encrypted using the substitution cipher. Since the substitution cipher has a relatively large key space (26! possible keys), brute-force is not effective against such a cipher. Utilizing English language letter and word frequency is a more effective approach in attacking such a cipher.

2.1.1 Task Requirements

You are given the following (highlighted) ciphertext (which can also be found in the text file `Task1CT.txt` in Blackboard) and your task is to recover the corresponding plaintext using frequency analysis. Note that in practice, spaces as well as numbers and punctuation marks are removed from the plaintext before encryption but to make it easier for you, we have left the spaces in the text.

```
tsl qvrcjl xajj yveli tsl kiazyakjld mzr tlyszaocld vu dlyciazg
yvqkctlid mzr zltxvipd at ad jvgaymjjb dkjat aztv txv kmitd uaidt
kmit lfkjmazd tsl yibktvgimksay tslvib azyjcrazg qmzb yjmddaymj md
xljj md kvkcjmi yakslid mzr tsl dlyvzr kmit yvelid aztlizlt mzr
zltxvip dlyciatb kivtvyvjd mzr mjgviatsqd md xljj md tsilmtd lg
imzdvqxmil slzyl bvc xajj jlmiz hvts tslviltaymj mdklytd vu yvqkctli
 mzr zltxvip dlyciatb md xljj md svx tsmt tslvib ad mkkjalr az
kimytayl tsl pzvxjlrgl xajj sljk bvc az rldagzazg mzr rleljvkazg
dlycil mkkjaymtavzd mzr zltxvip kivtvyvjd mzr hcajrazg dlycil
yvqkctli zltxvipd
```

What to Submit: You need to include the recovered plaintext as well as the recovered key. You must detail the steps you have followed in your cryptanalysis. Also, you are free to include any interesting observations you have made/learnt from doing the task.

2.1.2 Guidelines

You can use whatever way you prefer when substituting the letters. Example of tools/resources which can come in handy include:
`https://cryptii.com/pipes/alphabetical-substitution`
`https://www.dcode.fr/frequency-analysis`
`http://www.richkni.co.uk/php/crypta/freq.php`

The following resource might be useful for the frequency analysis:
`https://norvig.com/mayzner.html`

For substituting the letters, you can use any of the above (or any other) tool of your choice. You can also use the Linux command `tr`. Below is an example of how we can use the command to substitute `'a'→'Y'`, `'f'→'R'` and `'n'→'T'` in the text in the file `infile.txt` and store the translated text in the file `outfile.txt`. When doing the translation for this task, it is advisable to use a different letter case from the letters in the plaintext and those in the ciphertext.

```
tr 'afn' 'YRT' < infile.txt > outfile.txt
```

2.2 Task 2: Using OpenSSL Command-Line

In this task you will explore using the OpenSSL command-line commands to encrypt using different symmetric ciphers and different modes of encryption.

2.2.1 Task Requirements

You need to provide 2 examples of encrypting/decrypting with 2 different block ciphers.

What to Submit: You need to include screenshots of the commands/steps you used as well as screenshot of plaintext/ciphertext files content. Also, you are free to include any interesting observations you have made/learnt from doing the task.

2.2.2 Guidelines

To encrypt/decrypt, you can use the OpenSSL `enc` command. The general syntax of the `enc` command is as follows where you replace `-e` with `-d` when decrypting:

```
openssl enc -ciphertype -e -in Inputfile -out Outputfile   \
            -K KeyValue  \
            -iv IVValue
```

Note: The \ symbol is to ensure that we are continuing on the same line and not inserting a newline character.
Where:

`-ciphertype`: is the name of the cipher you want to use, e.g. `-aes-128-cbc`, `-aes-256-cfb`. To find out the list of supported ciphers, you can run the command `man enc`.

`Inputfile`: is the name of the file containing the plaintext if encrypting or the ciphertext if decrypting.

`Outputfile`: is the name of the file where to output the ciphertext if encrypting or the plainext if decrypting. We already explained how to view text in ASCII/Hex but if you prefer a tool for such translation than using a command line, this might come in handy `https://www.rapidtables.com/convert/number/hex-to-ascii.html`

`KeyValue`: is the value (in hexadecimal) of the key you want to use.

`IVValue`: is the value (in hexadecimal) of the initialization vector (IV) you want to use. Note that this is not required for some ciphers/modes.

Note: If the number you provide as a key/IV is shorter than the required length, it will be padded with zeros to reach the required length. This can be problematic in practice.

To geneate random keys/IVs you can use the OpenSSL `rand` command. The below example generates a random 16-byte (128-bit) long hexadecimal random value:

```
openssl rand -hex 16
```

If you need help converting between ASCII/hexadecimal, the below examples which use the Linux hex dump tool `xxd` might come in handy. In all the examples, the option `-p` ensures that the output is in plain hexdump style.

- Below is an example of how to convert from hexadecimal to ASCII:

```
echo "41420a" | xxd -p -r
```

- Below is an example of how to convert from ASCII to hexadecimal, where we add the option -n so that the echo command does not append the newline character (ASCII code: `0x0a`) to the output:

```
echo -n "AB"  |xxd -p
```

- Below is an example which produces a hexdump of a *(supposedly) existing file test.txt:*

```
xxd -p test.txt
```

Also, the tool `https://www.rapidtables.com/convert/number/hex-to-ascii.html` can be used to translate between ASCII and hexadecimal.

2.3 Task 3: CBC mode vs. ECB Mode

This task requires you to compare the use of electronic codebook (ECB) and cipher block chaining (CBC) modes of encryption.

2.3.1 Task Requirements

In Blackboard you will find a BMP image called `Task3Image.bmp` ([Source: https://www.pngkit.com/]). You are required to encrypt this image using AES 128 once in ECB mode and another time in CBC mode.

What to Submit: You need to submit a screenshot of both versions of the encrypted image, the commands/steps you used, and briefly discuss the difference of the quality of the confidentiality of the obtained encrypted images. Also, you are free to include any other interesting observations you have made/learnt from doing the task.

2.3.2 Guidelines

In BMP images, the first 54 characters in the image file are reserved for the header which identifies this image format. Thus, unless you replace the header of an encrypted image, it would not be classified as a valid image anymore. Follow the below example steps to copy the file header from the original BMP image (e.g. `original.bmp`) and the encrypted image (e.g. `encrypted.bmp`) into a new image file (e.g. `newencrypted.bmp`).

1. Copy the first 54 characters from `original.bmp` to a file named `header`.

```
head -c 54 original.bmp > header
```

2. Copy the rest of the encrypted image (excluding the encrypted header) into a file named `body`.

```
tail -c +55 encrypted.bmp > body
```

3. Merge the original header and the encrypted body into file `newencrypted.bmp`.

```
cat header body > newencrypted.bmp
```

2.4 Task 4: Plaintext Padding

Since block ciphers operate on blocks of the plaintext, there might be a need to pad the plaintext if its length is not a multiple of the required blocksize. The PKCS#5 padding scheme is widely used by many block ciphers (see e.g. `https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html` for details). This task requires you to observe how the different encryption modes use padding.

2.4.1 Task Requirements

In this task you are required to observe how the ECB, CBC, CFB, and OFB modes of encryption use (if any) padding. You can use any block cipher of your choice.

What to Submit: You need to report which of those modes require padding and which do not, justifying your answer in each case. You need to submit a screenshot of the commands/steps you used to reach your conclusion. Also, you are free to include any other interesting observations you have made/learnt from doing the task.

2.4.2 Guidelines

Here you should attempt to encrypt a file containing a plaintext whose length is not a multiple of the block size in question and then attempt to decrypt the ciphertext file and compare the decrypted text with the original plaintext. Note that by default the `openssl enc -d` command removes the padding when decrypting so that you always obtain the original plaintext when decrypting the ciphertext. To stop the decryption command from removing the added padding, you need to add the option `-nopad` when invoking the decryption command. Also, you should be aware of any (invisible) special characters, e.g. the newline character, as those would affect the length of the plaintext. Below is an example of using the Linux `echo` command to write the plaintext `ABC` (ASCII) to the file `plaintext.txt` while ensuring no newline character is added.

```
echo -n "ABC" > plaintext.txt
```

2.5 Task 5: Incorrect use of IVs

It is crucial for the security of block ciphers that the used IVs do not repeat, and to achieve stronger security when using some encryption modes, it is required that the IVs are generated randomly so that they are unpredictable. This task requires you to recover the plaintext from insecurely encrypted ciphertext without knowing the key.

2.5.1 Task Requirements

In this task you are provided with a pair of plaintext (P_1) and corresponding ciphertext (C_1) which was encrypted with AES 128 OFB mode. Also, we provide you with another ciphertext (C_2) corresponding to an unknown plaintext (P_2) which was also produced using AES 128 OFB and the same key and IV as that used in encrypting P_1. Your task is to use the provided information to recover the plaintext P_2. Below is the information you need:

P_1 **(ASCII):**	`UWE is G-ACE-CSE`
C_1 **(Hex):**	`ace4564cb988b435d9a3c1d633cb59a2`
C_2 **(Hex):**	`addb7218f092e7529390e7f26ab227ce`

What to Submit: You need to provide the plaintext P_2 in ASCII as well as screenshots of the commands/steps you have used and a brief explanation of the strategy you used. You need to submit a screenshot of the commands you used to reach your conclusion. Also, you are free to include any other interesting observations you have made/learnt from doing the task.

2.5.2 Guidelines

Figure 1 shows encryption and decryption using the OFB mode. Below is an example of how to compute the XOR of 2 hexadecimal values `a` and `b` using the Linux command line:

```
echo $(( 0xa ^ 0xb ))
```

The guidelines we provided for the previous tasks might come in handy for answering this task. Also, the calculators in `http://xor.pw/` or `https://toolslick.com/math/bitwise/xor-calculator` might be an alternative to using the command-line.

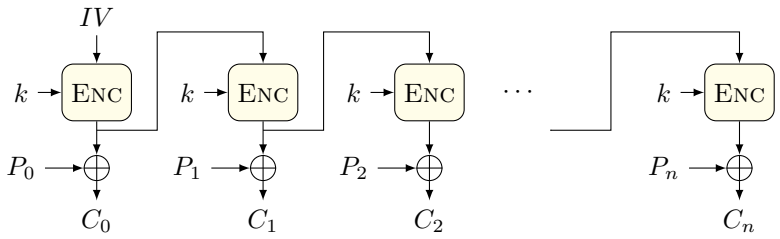
2.6 Task 6: Brute-Force using the OpenSSL C Library

This task requires you to use the OpenSSL C library to perform a brute-force attack to recover the encryption key.

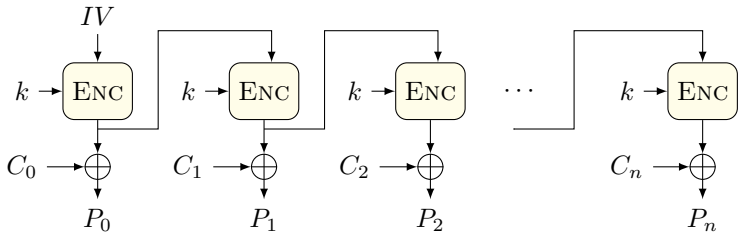
2.6.1 Task Requirements

We provide you with a plaintext and a corresponding ciphertext (encrypted using AES 128 in CBC mode) as well as the IV used, and your task is to use brute-force to recover the key. You must use the OpenSSL C library for this task as using the command-line commands will not be accepted as an answer for this task.

To make your task feasible, the key we used is an English word and is contained in the file `WordList.txt` which can be found in Blackboard. Note that if the word (key) is shorter than 16 characters (the required 128-bit length), we have appended



(a) Encryption



(b) Decryption

Figure 1: OFB Mode of Encryption

the character `'*'` (ASCII Code 0x2a) to the word to reach the required key length. For example, if the word is `hello`, the key used is `hello**********`.

Below is the information you need:

Plaintext (ASCII):	`Gold ACE-CSE!`
Ciphertext (Hex):	`e2773a09c0e095da9eedebf34ed636da`
IV (Hex):	`ffeeddccbbaa00112233445566778899`

What to submit: You need to submit the recovered key as well as the code you have used to recover the key. Also, you are free to include any other interesting observations you have made/learnt from doing the task.

2.6.2 Guidelines

There are many online resources which explain how to use the OpenSSL C library, e.g.:
`https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html`
Also, in Blackboard we included a toy example `ToyEnc.c` which shows how to encrypt/decrypt a short message using the OpenSSL library.

Note: When using `gcc` to compile a program that uses the OpenSSL library, you need to include the option `-lcrypto`. Below is a command-line example showing

how to compile `ToyEnc.c`.

```
gcc -o ToyEnc ToyEnc.c -lcrypto
```

Also, be aware of the case of the words when performing your brute-force as the word case of the key you attempt should be as in the provided word list file.

2.7 Task 7: Post-Quantum Ciphers (Research Task)

It is well-known that some of the currently deployed cryptographic constructs used in protecting communication cannot withstand attacks by quantum computers and thus as soon as a high-scale quantum computer sees light, such protocols will be immediately rendered insecure. This task requires you to undertake some research regarding how quantum computers would affect currently deployed symmetric encryption ciphers.

2.7.1 Task Requirements

Write a short report (1000 words max) discussing how would quantum computers affect currently deployed symmetric encrypting schemes and what measures need to be taken to offer protection against quantum computers in the context of symmetric encryption. Your report should include relevant references from the literature and your discussion should cover efficiency and security aspects, e.g. key sizes.

3 What to Submit

You need to submit a detailed report, with screenshots, to describe what you have done, what you have observed, and how you reached your conclusion/answer for each task. The format of the report is up to you but you must adhere to the requirements mentioned in What to Submit at the end of the tasks. The report should be of a professional standard. You need to provide explanation to the observations that are interesting or surprising. Please also list any important code snippets you have written followed by explanation. Simply attaching code or screenshots without any explanation will not receive credits. The report must demonstrate your understanding of the subject and material and not just be a log of your actions. *All screenshots in the report must have your student number and date stamp in the user prompt. Failure to include these details in the screenshots will invalidate the report and receive a mark of zero.*

3.1 Plagiarism

This is an assessed lab sheet and while it is acceptable to discuss your assignment with your peers as per the university rules, this assignment is intended as an individual assignment. Submissions that are substantially similar will be subject to investigation according to university regulations and any proven cases will be dealt with according to the regulations. More details can be found here `http://www1.uwe.ac.uk/students/academicadvice/assessments/assessmentoffences.aspx`

4 Marking Criteria

	0-29%	30-39%	40-49%	50-59%	60-69%	70-84%	85-100%
Tasks 1-6 (65%)	Little to no attempt/Serious gaps or errors in understanding the topic	Some tasks completed with major omission/Some evidence of understanding the topic with major errors or gaps	Most tasks completed but with minor omissions/Evidence of understanding the topic but with minor errors or gaps	All tasks completed in full. Evidence incomplete or unclear in places/Adequate understanding of the topic	All tasks completed in full. Evidence of a good standard to detail tasks/Clear understanding of topic	All tasks completed in full. Excellent use of evidence to detail tasks/Thorough and comprehensive understanding of topic	All tasks completed in full. Highly reflective use of evidence to develop argument /Impressive and original depth of understanding of topic
Task 7 (25%)	Little to no discussion	Poor analysis. Demonstrates little or no insight into the problem	Below-average analysis. Analysis is lacking in most aspects	Adequate analysis of relevant works but lacks depth; demonstrates some insight into the problem	Good analysis of relevant works but could be more critical; demonstrates good insight into the problem. Good use of sources	Very good analysis of relevant works; demonstrates excellent insight into the problem. Good use of sources and all sources are appropriately referenced	Outstanding analysis of relevant works; demonstrates outstanding insight into the problem and fully covers all aspects. Excellent use of sources and all sources are appropriately referenced
Report Presentation (10%)	Very poor presentation	Weak presentation	Has not followed required conventions; poor proofreading	Partially follows required practices; some issues to be addressed e.g., typos, punctuation	Follows required presentational practices; a few typos/errors in punctuation or grammar	Excellent presentation: typos/errors in punctuation etc. are rare	Excellent presentation

5 Document Revision History

Version	Date	Changes
1.0	05th March 2021	Initial Release