Computer and Network Security
Lab 1: Symmetric Encryption

Jeroen Famaey and Esteban Municio- University of Antwerp
(esteban.municio@uantwerpen.be - M.G.325)

2016 – 2017

Name student:

1 Introduction

In this first lab session you will get familiar with the basic and practical concepts in Computer and Network security. Specifically, you will work on secret-key encryption and symmetric ciphers throughout different encryption challenges. After finishing the lab, you will have gained first-hand experience on the algorithms and concepts you have previously studied, such as mono-alphabetic ciphers, cryptanalysis, the AES algorithm, padding, initial vector (IV) and block ciphering. Through using different algorithms, tools and cipher mechanisms you will be able to encrypt/decrypt information for different applications and requisites.

This lab session is divided in four challenges:

1. Mono-alphabetic cyphers and cryptanalysis
2. AES-ECB
3. AES-CBC
4. ECB vs CBS

In this lab, you will use *openssl*. It is a powerful open-source tool that implements most of the state of the art methods in computer and network security. Also you will use the Python Crypto libraries that implement most of the *openssl* primitives

2 Mono-alphabetic-cyphers and cryptanalysis

2.1 Challenge 1

Mono-alphabetic cyphers were the first used in history. Although they could seem hard to break, most of the mono-alphabetic cyphers are highly vulnerable to cryptanalysis. If the nature of the plain text is known, the analyst can exploit the regularities of the language/protocol by analysing the standard frequency letter distribution. In this challenge you will have to break a coded message by analysing the repetition of the letters, without knowing neither the key nor the plain text. For this exercise, you only know that the plain text is in English and that the mono-alphabetic cypher used maps each of the 26 letters (plus the space character 0x20) of the English alphabet to another one. You will be able to read properly the plain text by finding these relationships.

The message to be decoded is the following:

```
iyzzfwpmzcyawcixvwxvwciywpfavcwryvvm ywqafrwciy
wbfs fwhzymvywuywmpmaywcimcwciywrmvbmamgywimvwu
yyswcfcmzzewgxvbfkyaygwsfwrfaywbfrrjsxbmcxfsvwv
xsbywciyewmaywcffwgms yafjvwqfawjvwuewciywpmewc
iywrfvcwhfhjzmawzyccyavwxswys zxviwmaywycmfxswv
iagzj
```

Notes:

1. The key that encrypts this message is actually a shuffled English alphabet with the space character, which is equivalent to 27 character length key. The message has 240 characters, be sure you are using the entire message.

2. Using force brute, you should check $27! > 1^{28}$ combinations to get the correct key. As you can guess, checking this amount of combinations is quite time demanding. We give you the first 19 characters of the key to make it quicker: *wmubgyq ixldzrsfhta*

3. Even within a small range, it is still necessary to know which of the solutions is the correct one. In order to asses this, you could do this manually, which would not work in a real scenario. Devise some method for scoring a piece of English plain text. We suggest using character frequency in order to evaluate each output and choose the one (or the ones) with the best score.

4. In order to get an absolute score in character frequency for the English language, we provide you through Blackboard with text file which you can parse and use to calculate the character frequency. The name of the file is: book.txt

2.2 Questions Challenge 1

1. Program in Python a script that decrypts the coded message by scoring English plain text. Provide the code and explain the steps you have followed.

2. What is the plain text? What is the used key?

3 Advanced Encryption Standard with ECB

3.1 Challenge 2

After the mono-alphabetic cyphers, poly-alphabetic cyphers were suggested for symmetrical encryption in order to increase complexity and security. However, most of them were still weak, and the only proven unbreakable poly-alphabetic cypher was the one-time pad scheme. However this one requires to generate a huge amount of random keys and distribute them securely over the network. For this reason more advanced techniques have been devised based in block substitution and complex algebra operation such as DES and AES. As DES is currently breakable, most of the current security solutions implement the AES scheme. In this challenge you will have to use the AES libraries to decode an encrypted message. Each programming language provides their own cipher algorithms and the bindings over the original openssl API. For this challenge use the native Python implementation Crypto. Information about this module can be found online (https://pypi.python.org/pypi/pycrypto).

The message to be decoded is the following:

```
y4TrHG1xZgvdRcnn2lU+el785PPKBRVgS7IvIa
onruraSDs9rhsGQrgmbI9TM5pxhTWEaBrqxJMN
vmwIHuP9WfR7O1QsokIZcyqVO/GSYyDIqM1Nil
U5kBSsI7Tfg21ayxOo04h60fgAm5B+GxdgYiGj
hw/LK04o9Hch6PvVowQ=
```

Notes:

1. The key that encrypts/decrypts this message is the following 16 character string: *"VIRTUAL INSANITY"* (case sensitive, without the quotes). The encoded text has 172 characters, be sure you are using the entire message.

2. You do not have to use force brute, you already know the password!

3. As you may know, AES usually uses Base64 code as input and output. For this reason you will want to use the Base64 Python library (https://docs.python.org/2/library/base64.html)

4. For AES-128-ECB, it is usually necessary to use padding for those text blocks that are less than 16 bytes long. There are several RFCs for performing this padding (filling with zeros, filling with the number of remaining empty bytes, etc). In order to make it easier, the plain text is exactly a multiple of 16 bytes.

3.2 Questions Challenge 2

1. Program in Python a script that decrypts the coded message given the password. Provide the code and explain the steps you have followed. No credit will be given by using the *openssl* command line tool. You have to use the libraries.

2. What is the plain text message?

4 Advanced Encryption Standard with CBC

4.1 Challenge 3

As you know, the ECB operation mode performs the encryption individually in each block, which involves several unwanted "features" in the encrypted code (see Challenge 4), and thus, its use is not recommended. However other more secure operation modes exist such as de Cipher Block Chaining (CBC). This mode is much more secure since each block depends on the previous blocks. This challenge consist in giving the plain text and the encrypted text coded with AES-256-CBC, guess the key used. As you will notice, performing a brute force attack is a gargantuan task, since the key is 256 bits long. However in some cases, getting the key is not that difficult if the user sets a poor key. In this challenge we have set a key that is easily guessed by a dictionary attack, so the attack should be performed in less than 1 second.

The plain text and the encrypted text is the following:

P:*"This is a really hidden message."*
E:*"chiVBb5bMmiqhsm3Gpp6IFd/CalXMXKMrEMQJCl1u+0="*

Notes:

1. The key that encrypts this message is unknown. As before the plain and code messages are case sensitive and without the quotes). The plain text has 32 characters and the encoded message has 44 characters, be sure you are using them correctly.

2. In this challenge you DO have to a use brute-force attack but in a slightly smart way. The key is 265 bit SHA-2 hash generated by an English word. Hash functions have not been explained yet, however you only have to

know that they perform something similar to a "unique summary". That means that any text string can be summarize in a 256 bit pattern which can not be generated by other text (even if you only change a bit, the summary can be totally different). To perform this hash you only have to use the library Hashlib in Python (https://docs.python.org/2/library/hashlib.html)

3. The easiest way to go towards all possible English words, is by using a dictionary. We provide you with one in Blackboard. The name of the file is *words.txt*

4. As before, AES usually uses Base64 code as input and output. For this reason you will want to use the Base64 Python library (https://docs.python.org/2/library/base64.html)

5. As you may know, AES-CBC uses a initialization vector (IV). For this challenge you have to assume a IV of 16 bytes length filled with NUL (0x00 in Hex).

4.2 Questions Challenge 3

1. Program in Python a script that guesses the key used for encode the given plain text. Provide the code and explain the steps you have followed. No credit will be given by using the *openssl* command line tool. You have to use the libraries.

2. Which is the key?

5 ECB vs CBC

5.1 Challenge 4

Now that you have created your own implementations of AES-ECB and AES-CBC, it is time to assess practically the difference between them. We provide you with the image *"excelent.bmp"* (in Blackboard) and you have to coded it using both ECB and CBC modes. You can use the number of bits, the IV and the key you want. Notes:

1. Since you are encoding a file, the file header will be lost since will be also encoded, and most of the image viewers will not recognize it. In order to visualize the encoded image, substitute the first 54 bytes of the file, with a non encrypted legitimate file header. You can copy-paste from the image *"excelent.bmp"* using an hexadecimal editor (i.e. *ghex*). If you do not have any hexadecimal editor you can implement a easy Python script which copy these 54 bytes from the original to the

encoded ones. The Python library mmap will help you with this task (https://docs.python.org/2/library/mmap.html).

2. You already have implemented both modes, so you may want to use your own code. However you are now free to use the command line tool *openssl* if you want.

5.2 Questions Challenge 4

1. Provide the encrypted images resulting from encrypting *"excelent.bmp"* with AES-ECB and AES-CBC respectively. You can use your own code or the command line tool *openssl*.

2. Display the images with a generic picture viewing software. What is the difference between them? Can you infer something from any of the images?

6 Submitting your solution

The deadline for submitting your solution is Thursday October 27th, 2016 23:59. To submit your solution do the following:

- Create a text file with your answers.
- Clean the code files and prepare other required files.
- Create a compressed file with everything. Name the compressed file **cns-lab1-name.zip** (where *"name"* is your UA user name) and upload it to Blackboard under "Assignments".