

COMP 524 - Cyber Security: Assignment #2

Due on November 7th, 2018

Professor Michael Soltys

Rihan Stephen Pereira

email - rihanstephen.pereira576@myci.csuci.edu, studentID - 002497665

David Byrne

email - david.byrne@myci.csuci.edu

Marco Cabrera

email - marco.cabrera561@myci.csuci.edu, studentID - 002456715

Kaveh Arashvand

email - kaveh.arashvand075@myci.csuci.edu, studentID - 002377350

November 7, 2018

Problem 1

Problem 6.1 Implement CBC on a plaintext of arbitrary length; your program should be called as follows: `python3 des-cbc.py key plaintext.txt` and it should output `ciphertext.txt` using DES in ECB as a building block to implement DES in CBC. Essentially you need to loop over the characters in `plaintext.txt` in groups of 8 characters at a time, applying a modification of the above snippet each time. Propose a solution for the case that the length of plaintext is not divisible by 8.

Solution:

As we know, we are working with blocks of 8 characters to be sent into DES as the valid input and due to this we chunk our plain text into blocks of 8 characters, now what will happen if the last remained block doesn't have the exact 8 characters?!! This is where we require some approaches to overcome this situation and PADDING is the general solution: adding some characters to the end of the last validate character remained from plain text to create a complete (8 char length) block. At the following paragraphs we will delve more into the possible obstacles and approaches. Let's have a look at the characteristics that an appropriate **PADDING_CHARACTER** should have:

- It Should be **DISTINGUISHABLE**: for sure the character which we want to use, whatever it is, should be completely distinguishable from the valid part of plain text otherwise during the decryption process the recipient of the file has no mean to ensure which character is in the message and which one is for padding part, for example consider the following 8-bytes (characters) chunk which represents the last chop on the plain text. Also, assume that the **mod (length of plain-text and 8) is 2** which means we have 2 legitimate characters in the last chop and consequently we have to add six **PADDING.CHARACTERS** to the end of the last legitimate character to make a 8 characters chop, I am going to name the legitimate 2 characters by "L" and the padding part by "P":

LL**PPPPPP**

Our suggestion is to use un-readable ASCII characters for **P** because if we use any other ASCII code there is no way to achieve the **DISTINGUISHABILITY** characteristic. We know that in ASCII codes the first unreadable character is **10000000** in binary which is equal to 128. In the other words if we try to use this code as our **P** then we ensure that the recipient of the message can distinguish it from the rest of the valid text because definitely he would not be able to read it. For example we can make the last block in our example as follow:

LL **10000000 10000000 10000000 10000000 10000000 10000000**

This approach still can be a little nicer if we include the exact number of padded characters into it and then encrypt it, with this modification the recipient of the message can easily understand how many characters had been padded to the plain text before being encrypted and this can show him how many unreadable characters he can easily cut from the end of the last block to get the valid part, our approach to meet this demand is very straight forward, just use the binary equivalent of the number of padding characters and add it to **10000000 (128)**, in our example we have added 6 (**00000110** in binary) as Padding characters so we can easily make our last block as following:

LL **10000110 10000110 10000110 10000110 10000110 10000110**

This block will be encrypted using DES and then will be sent to the recipient of the message, then after being decrypted by recipient, the first 1 shows that this character is used as padding and the subtraction of 128 shows the number of padding characters which should be cut to make the decrypted file meaningful.

The python code for Padding and the Pseudocode for DE-Padding are as following codes:

```
### PADDING the PLAIN TEXT:
def checkpad(plaintext):
    length = len(plaintext)
    if ((length % 8) != 0):
        rem = length % 8
    # Pad indicates how many characters we have to add to the end of the file
    pad = 8 - rem
    # the first meaningless number in binary is (10000000) which is equal to 128
    for i in range (pad):
        plaintext= plaintext + chr (128 + pad)
    return (plaintext)

### Pseudocode for DE-PADDING this is used by the Recipient
trim_padding (plaintext)
    get last char of file
    subtract it by 128 to indicates the number of padded characters
    if result is 1-7 :
        then number of pads = result
    return (plain-text the number of pads) # just cut the padding part
    else no padding we had and simply return the whole plain text.
```