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 Bryne
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 5, 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:

For padding part: This is the scenario where we require padding: when the length of our plan test is not dividable by 8 because for CBC we require chunks of 8 bits as input, thereby we have to padding the rest with something like ZERO to be ready to be sent into the next step, although this approach at the encryption seems complete, we require some approach for decryption part to indicate clearly how many bits the sender has already added to the last chop as Padding part. This is obvious that the lack of that declaration will lead to incorrect decryption of last character in the plain text after being decrypted by Recipient of the message. Lets demonstrate it step by step as following:

At first, consider the following 8-bit chunk which represents the last chop on the plain text with assuming that the mod (length of plain-text and 8) is 2 which means we have 2 legitimate bits in the last chop and consequently we have two add six ZEROs to the end of the last chop, I am going to name the legitimate 2 bits by L and the padding part by P:

LLPPPPPP which is LL000000

this can be encrypted using DES readily with no issue. The problem is finding a way to differ between LLLLLLLL with LLPPPPPP when they are exactly the same! For example when we require no padding for the last chop!!!

Our approach is:

The Sender of the message should add ONE mandatory 8 bits chop to the end of the message before being encrypted and on that chop somehow he mentions how many bits he wants to add to the actual last chop of the plain text. In our example: after padding the last chop to become LL000000 he has to simply create another 8 bits block and somehow indicates that our padding is 6 bits, this block should have another interesting character, it should be unreadable in ASCII codes because with this especial feature it shows its difference usage, actually it announces that I am not a part of the actual message and I am just an indicator, for getting this special character, we are going to add our 8 bits block with 1 ! we know that there is no ASCII code which starts with 1 and this is a great characteristic for being utilized to our goal, for example for showing 6 bit padding we create this block : 10000110 and then simply add it to the end of our plain text after being padded by zeros: LL00000010000110

- If we were not required to padding we just simply add: 10000000

With this approach the recipient of the message first decrypts the incoming message and from last 8 bits can find how many Bits were actually used as padding, lets name it PADDING, then he simply ignores (8 + PADDING) last bits to gain the legitimate plain text.