Politecnico di Torino

Cybersecurity and National Defence

Technical Report

Ali Fuat Sakaci S307607
Edip Shahinyoz S312087
Bruno Petutschnig S311940
Sendege Junior Urlich S297958
Divine Umuganwa S315758

Contents

1 Introduction iii
2 Fundamentals of 3DES iii
3 How Did We Implement iv
4 Steps of DES Encryption vi
5 3DES With Operation Mode vii

1 Introduction

This is a report on how the 3DES encoding algorithm works on the surface and how we implemented the algorithm with the coding language Python. We were tasked with writing the algorithm using the knowledge provided in the published book "Understanding Cryptography" by Christof Paar. The book goes on to mention that the 3DES is no longer considered a safe encoding algorithm on today's standards due to being subject to easy brute force attacks given the powerful computers of our time. We decided to use the operation mode ECB for our project. The triple DES algorithm we used shows results for both EEE (encrypt, encrypt, encrypt) and EDE (encrypt, decrypt, encrypt) for both overall 3DES encryption and decryption.

2 Fundamentals of 3DES

3DES is what is called a block encryption, meaning, the algorithm encrypts strings of bits of a certain length, instead of encrypting bit by bit. 3DES makes use of many types of set tables for encrypting. As a result of this 3DES is highly deterministic and so it is expected to yield the same result every time.

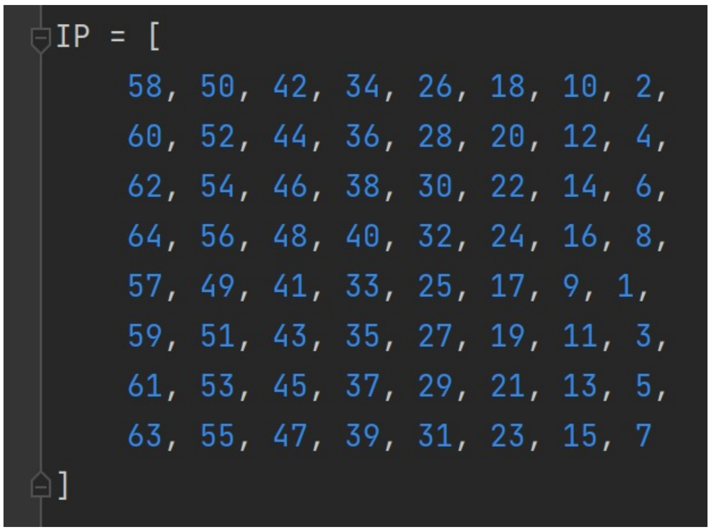
The tables used are:

- Initial permutation table
- Inverted initial permutation table
- E-box
- S-boxes (1 through 8)
- Permutation P
- PC-1
- PC-2

3DES with ECB works by receiving a string of certain length in binary and preparing to fit it into the DES algorithm by partitioning it into pieces of 64 bits after padding bits of mostly 0s to make it a perfect multiple of 64 bits as it is needed because DES algorithm works with receiving a 64 bit plain text.

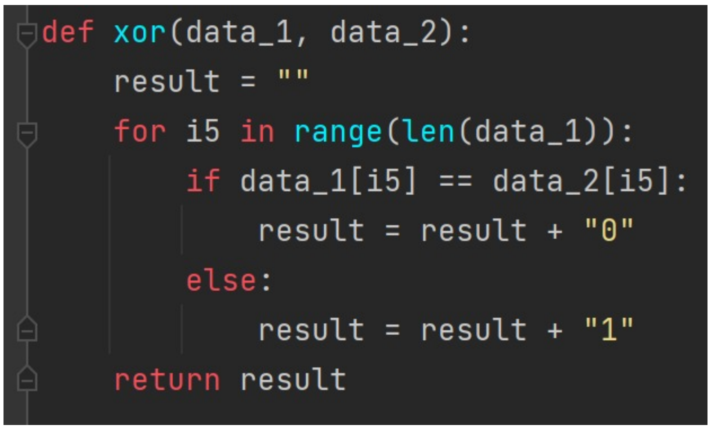
3 How Did We Implement

Before writing the code all the tables shown above were defined in python in forms of lists that can be called/referenced with the table name and an index number to call for a certain element in the table. Here is an example of how the Initial permutation table looks on code.



```python
IP = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]
```

Next step was to define fundamental functions like “xor”ing two strings or applying a table to a string to receive a new string. This was done by def function of python. Here is an example of a function defined to xor two data and return the result.



```python
def xor(data_1, data_2):
    result = ""
    for i5 in range(len(data_1)):
        if data_1[i5] == data_2[i5]:
            result = result + "0"
        else:
            result = result + "1"
    return result
```

Rest of the code works by making use of these functions and tables in correct orders to manufacture the correct algorithm for DES.

List of all the custom functions written by us:

- integer_to_binary(¡integer¿) , receives an integer value of maximum 15 and returns a string of binary corresponding to that integer.
- binary_to_integer(¡binary¿) , receives a string of binary with length of 4 and returns an integer value corresponding to that binary value.
- hex_to_binary(¡text¿) , receives a string of hex digits of any length and returns a string of binary values making use of a lookup table with corresponding values. The look up table is also defined by us in form of a list.
- binary_to_hex(¡binary¿) , receives a string of binary with a length of divisible by 4 and returns a string of hex digits making use of a lookup table defined by us in form of a list.
- table_apply(data, table) , receives a string of binary as "data" and the name of a table as "table". The amount of elements in the table must be the same as the length of the string of binary. The function returns a new string of binary applying the rules of the given table.
- left_shift(data,amount) , receives a string of binary and the amount of bits needed for shifting. The function returns the shifted string.
- Xor(data_1,data_2) , receives two string of binary values of the same length and returns the xor'd result.

The entire DES encryption and decryption part of the code is also defined as separate functions that are called later for the 3DES encryption and decryption.

DES encryption is defined as DES_e(ptxt,ky) , receiving a string of hex digits with length of 16 as the plain text and receiving a key also in the form of a string with 16 hex digits. The function returns the DES output in the form of a string of hex digits. Des decryption is defined as DES_d(ciper_text_hex, key_hex) with same rules for input and returns the plain text in the form of a string of hex digits. These functions are later called for 3DES application in this form:

For EEE:

DES_e(DES_e(DES_e(plain_text , key1) , key2) , key3)

For EDE:

DES_e(DES_d(DES_e(plain_text , key1) , key2) , key3)

4 Steps of DES Encryption

Inside the function for DES encryption we followed the rules mentioned on the book. Starting with converting both the plain text input and the key input from hex to binary ("...A7" -¿ "...10100111"). Plain text gets put though the initial permutation table. The key gets put through the PC-1, returning the 56 bit version of the key. Two separate string are defined for the left and right side of the plain text returning two elements of 32 bits. Same is done for the key returning two elements of 28 bits. From this point the consecutive rounds start.

For each round first the number of the round is checked to see if it is in the rounds that require a one bit or two bit shift to the key. This key is then used again for the next round before proper shifting. Once the key is shifted appropriately the key is put through PC-2 in order to yield the round key that is 48 bits. This is where the f-function starts. The f-function repeats for each round with the following instructions. Right part of the plain text is then expanded into 48 bits using the E-box in preparation for S-boxes. The expanded right part of the plain text and the current round key is xor'd. The result is used for the S-boxes. The S-box application works like this, for the input string is split into 8 groups of 6 bits. Each group has their own designated S-box. For example the first group is designated for the S-Box 1. Each S-box is unique and same for each round. The first and last bits of the 6 bit group combined as a 2 bit value defines the row, while the remaining middle part of 4bit defines the column of which the data is extracted from the designated S-box.

For example:

For the S-box input of "101110 010010"

The group 1 ("101110"); refers to the element found in S-box 1, row (10 = 2),
column (0111 = 7) that is (11 = 1011).

Making the first 4 bits of the output from the S-boxes "1011".

The group 2 ("010010"); refers to the element found in S-box 2, row (00 = 0),
column (1001 = 9) that is (7 = 0111) .

Making the next 4 bits of the output from the S-boxes "0111".

Output = "10110111..."

The full output of 32 bits is then put though the Permutation P table ending the f-function. After the right side of the plain text is put through these applications it is xor'd with the left side of the plain text and result is then assigned to be the right side of the plain text for the next round. The right part of the plain text that was before the instructions of the f-function is assigned to be the left side of the next plain text and the round ends. After 16 rounds, the left and right side of the resulting plain text is switched before putting the entire text through the inverted initial permutation table. Giving us the resulting finished cipher text. In our code it made so the result comes out with its hex version.

We have used the following online link for verification on our implementation of the DES algorithm.

https://simewu.com/des/

Steps of the DES decryption were not trivial to complete since every function and application used in encryption can be reversed simply. The decryption code works by using the same functions as before but backwards. One key difference between our encryption and decryption was the key schedule. On the encryption the round key used is manufactured at each round before being used, while on the decryption the round keys are prepared beforehand and called the right keys when needed.

5 3DES With Operation Mode

We decided to use the operation mode ECB. As stated before the ECB work by taking a hex string of any length, converting is into binary string and partitioning it into 64 blocks. If the string does not come out to be a multiple of 64, the last block is padded to fit into a 64 range. The padding happens like this, first a single bit of "1" is added and then as many "0" bits as needed to complete is added making the block 64 bits long. Then each block is separately put in 3DES. For the structure of the 3DES we decided to do both EEE and EDE versions and allow the user to put up to 3 different keys for each DES. For the 3DES decryption, the EEE encryption required a decrypt -¿ decrypt - ¿ decrypt while the EDE encryption required a decrypt -¿ encrypt -¿ decrypt schedule. Making it simple to code with the previous tools at hand.

The encryption program will request:

- A plain text, consisting of hex digits of any length.
- A key to be used for the first DES, consisting of 16 hex digits.
- A key to be used for the second DES, consisting of 16 hex digits.
- A key to be used for the third DES, consisting of 16 hex digits.

And return:

- A cipher text made from EEE.
- A cipher text made from EDE.

The decryption program will request:

- A cipher text, consisting of hex digits of length multiple of 16.
- A key to be used for the first DES, consisting of 16 hex digits.
- A key to be used for the second DES, consisting of 16 hex digits.
- A key to be used for the third DES, consisting of 16 hex digits.

And return:

- A plain text made from EEE.
- A plain text made from EDE.

We also made the program check for the padded bits before and remove them before printing the resulting plain text for the decryption.