EECE 5850 - Computer Network Security
Project Phase-I

Submitted by-

Md Zahidul Islam
(Contribution: DES and CBC implementation, Debugging, Diagram Drawing, and Report Writing)

And

Parikshit Jadav
(Contribution: DES debugging, ECB implementation, Debugging, and Report Writing)

Overview of the DES and working modes Implementation:

In the first phase of the project, we have implemented DES encryption/decryption algorithm and used the algorithm with two working modes namely ECB (Electronic Code Book)) and CBC (Cipher Block Chaining). The overall process of our Phase-I is described below:

- The implementation starts with the input from the user. The user will provide his/her 8-bytes secret key, IV (in case of CBC mode), and some preference instructions. In the instructions, the user will provide preferred working mode, input type and security task (encryption/decryption).
- The program reads the input type, namely *string*/*file* and asks the user to provide the input *file* or *string* message. Finally, the *work_modes.py* python module is called with all the inputs from the user. These steps are shown in Fig. 1.
- The conceptual diagram of *work_modes.py* module is shown in fig. 2. The program runs the desired operations based on the user inputs. For example, if the user wants to encrypt a message with the CBC mode, the program runs along the left arrow as shown in Fig. 2. Then, it takes each of the 8-bytes padded message block, saved in the *msg_block* variable (the process of generating this block is discussed next), and performs XOR operation with the initialization vector (IV). Next, the output block is encrypted using our defined DES encryption algorithm (*DES_EncDec* function inside *des_base.py* module). Finally, the encrypted block is saved in a variable named *encrypt_msg* and is also used instead of IV to perform XOR operation with the next message block, as shown in Fig. 2, which is the concept of CBC working mode. The implementation of other options e.g., decryption, ECB modes are also conceptualized in Fig. 2.
- The process of generating 8-bytes padded message block from the input is shown in Fig. 3. The process starts with reading each 8-bytes data from input and pad the data with “PKCS5” Padding [1] if it is not 8-bytes long. The padding function is defined inside *base_des.py* module. The padded data block is saved in variable named *msg_block* as binary data.
- The basic DES encryption/decryption algorithm is coded inside *base_des.py* module. Here, we define a python class with several methods (functions). The important methods listed below:

- The *key_gen* method is defined for generating 16 sub keys from the 8-bytes secret key. Note that each 48-bits key is named sub key is our implementation.
- The *mangler_func* method is defined to perform the Mangler operation of DES.
- The main encryption/decryption is performed in the *desEncDec* method. This method can encrypt/decrypt each 8-bytes message block based on the instruction. If the instruction is to encrypt the block, then the iteration starts with the first sub key (K_1) and ends at the 16th sub key (K_{16}) generated by *key_gen* method and complete encryption operation using each key (K_i). Whereas, the iteration starts with the last sub key (K_{16}) and ends at the 1st sub key (K_1) in case of decryption instruction.

- In addition to the above methods, there are some other defined methods in *base_des.py* module such as *padding* and *un_padding* which are used to pad and un-pad the input block, respectively. We also defined two methods (*byte2bitList* and *bitList2byte*) for converting data type between byte and bit python list. These methods are necessary because DES works on individual bit and the input bytes should be converted to bit list so that we can perform bit wise operation for the successful implementation of DES algorithm.
- Fig. 4 shows the final output process of our algorithm, where we remove the padding and save the desired output message.

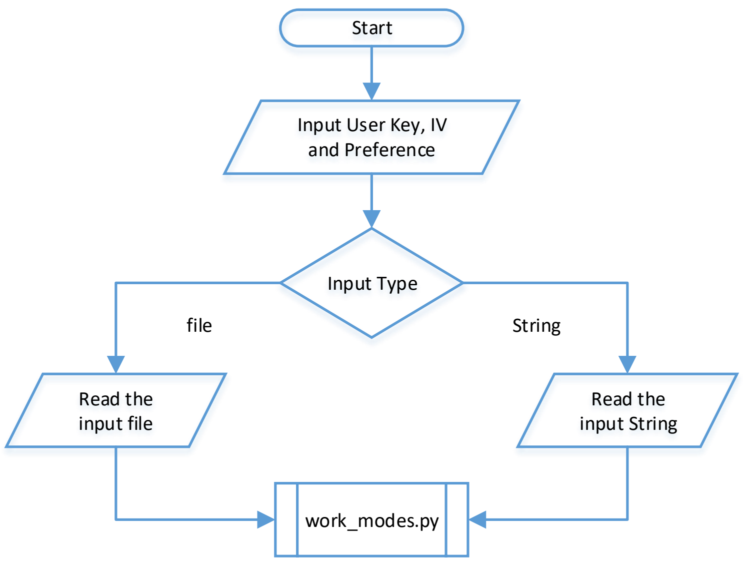

Fig. 1: Input processing and calling *work_modes.py* module.

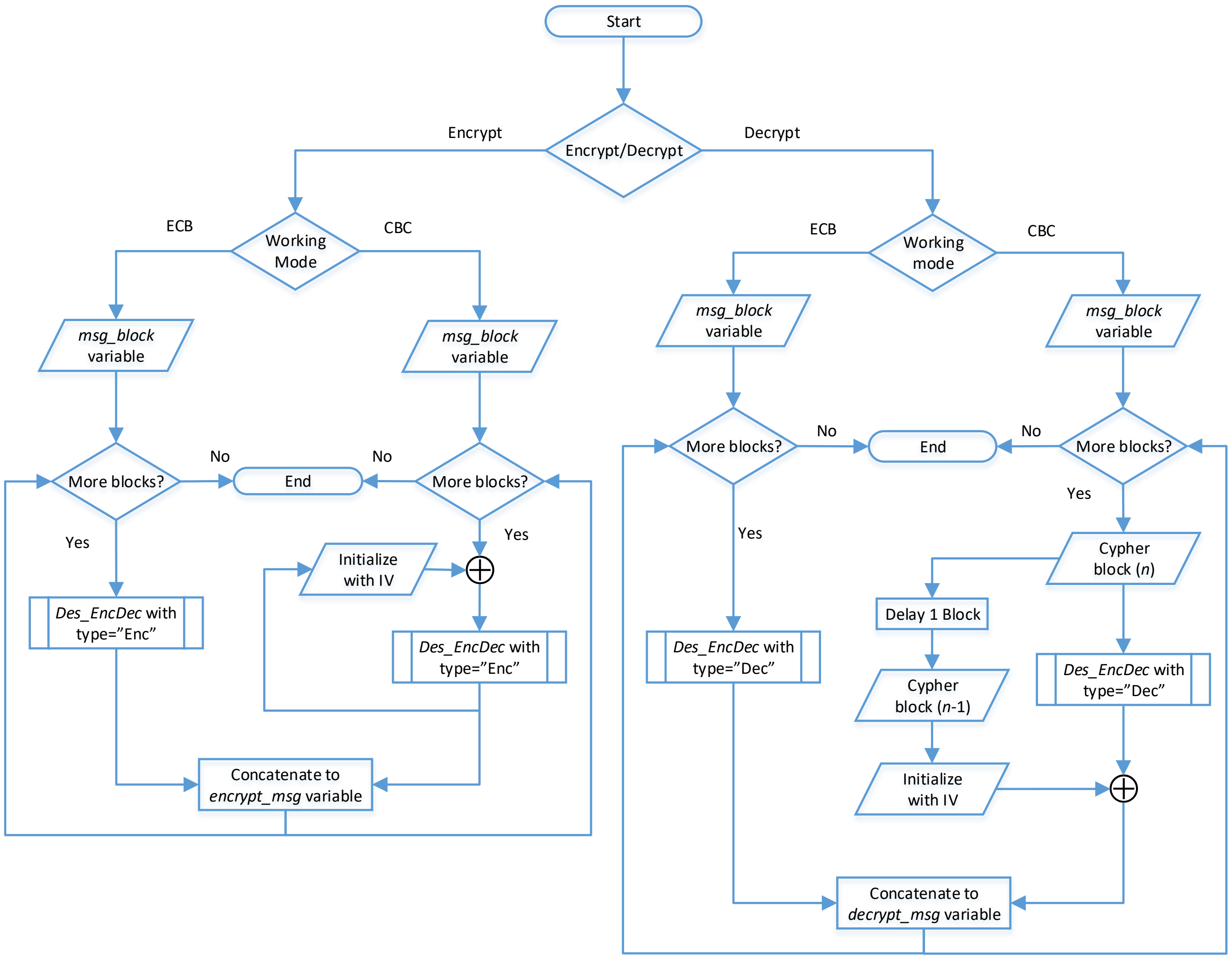

Fig. 2: Encryption/ decryption with two working modes (inside *work_modes.py* module).

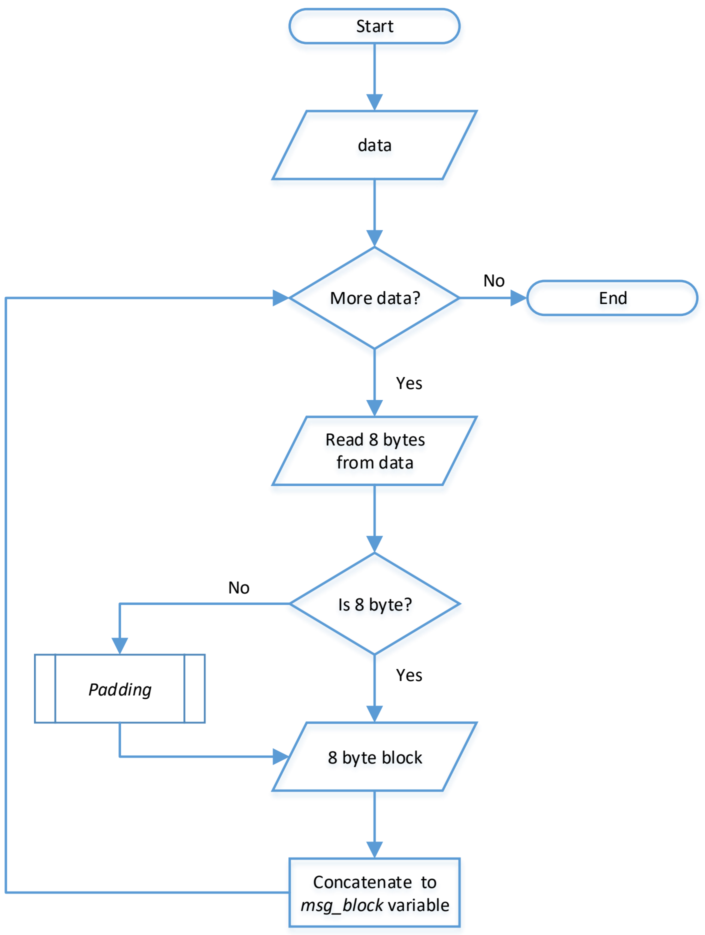

Fig. 3: Dividing message into 8-byte block with padding.

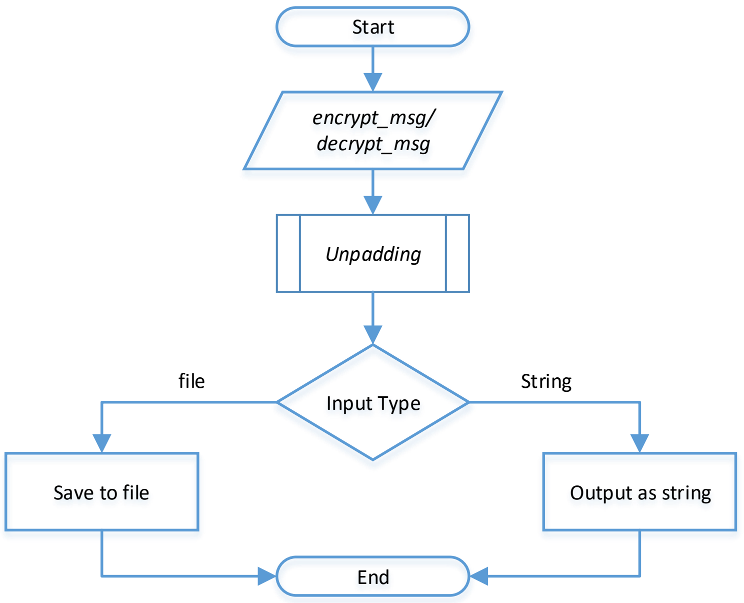

Fig. 4. Output processing.

Debug process:

First, the tricky part in Phase-I is the bitwise operation of DES algorithm. Each bit needs to be treated individually, which makes it inefficient for software implementation. Moreover, frequent permutation operation on individual bits also needs good attention to implement, otherwise, there is a higher chance of getting an error. Second, we need to check the data type carefully and perform expected type conversion operation before passing the data to the functions. For example, if the input message data type is string, then it needs to be encoded using utf-8 binary encoding. Third, for working mode implementation, dividing the input data into blocks, and padding and un-padding the blocks also needs to be handled carefully. In addition, in the CBC modes, the use of the recent encrypted/decrypted blocks needs to be ensured instead of IV.

Discussion:

This project has given us an opportunity to improve our programming skill through the hands-on implementation of DES algorithm along with its two working modes (i.e. ECB and CBC). The understanding developed through this project are helpful in implementing other algorithms like Triple DES, AES or RSA.

Though we successfully implemented the algorithms, there are a few things to consider to make the implementation more efficient and readable. For example, in the DES algorithm (*desEncDec* method), the input bytes are converted to a bit list at first and then it is converted back to the bytes after performing the encryption/decryption, which can be avoided with some modification of the logic.

Reference:

1. https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html
2. https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm
3. https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/DES.k