Practical Assignment 1

Note:

This assignment is focussed on the implementation and characterisation of the DES algorithm. For your convenience, a chapter on DES from the textbook "Cryptography and Network Security" by William Stallings has been linked on the Moodle page, although you may use any reference work you like. Whenever tables or figures are referred to, however, they refer to the tables and figures of the aforementioned chapter.

While you may use any programming language, I would encourage you to use a language that can easily support matrix manipulation. I have used Matlab. If you use a language that does not natively support simple matrix manipulation, try use a library to do this for you. As a last resort, you can write the functions to do this yourself.

The standard rules on cheating and plagiarism apply. You must reference every source that you use. You must not copy code from another source, including current or past students of UKZN or any other University, regardless of whether you reference it or not. If you are unsure, please check with me first.

Question 1 (40 marks)

Write the following functions in the programming language of your choice. In your report, give a short description of what each function does and, if necessary, its role the overall algorithm. Furthermore, any significant design or programming decisions should be reported on and relevant code should be appropriately commented.

Depending on the language you choose, some of these functions may be very short, perhaps a single line, but for readability and reusability, it is important that you write these as functions.

InVec and OutVec are bit arrays of unspecified length. You may use different names in your own code, they are merely presented here as place holders.

- **OutVec = LeftShift(InVec)** – performs a left shift on a bit array InVec and returns it as OutVec.
- **OutVec = ParityDrop(InVec)** – performs the parity bit drop and compression according to Table 6.12
- **OutVec = SW(InVec)** – swops the left and right halves of InVec and returns it as OutVec.
- **OutVec = SBoxLU(InVec, S)** – uses InVec it to generate the row and column. The corresponding element in the S-box, S, is then returned as OutVec. You might also want a function that returns the S-Boxes – see Tables 6.3 to 6.10
- **OutVec = ExDBox(InVec)**[1] – performs an expansion and permutation on InVec and returns it as OutVec – see Table 6.2
- **OutVec = DBox(InVec)** – performs a permutation on InVec and returns it as OutVec – see Table 6.11

[1] You may instead wish to have a single generic function for performing all permutations which takes two arguments – the vector being shuffled and a vector with the indices detailing the shuffling.

Using the above functions, write the following functions which will form the larger building blocks of DES encryption.

- **K_round = RoundKeyGen(K)** – generates round keys from the bit array K. K_round is a matrix, where each row is a different round key.
- **OutVec = InitPerm(InVec)** – performs the initial permutation on InVec and returns it as OutVec.
- **OutVec = fDES(InVec, K_round_row)** – performs the DES function on InVec using round key K_round_row. In practice, OutVec and InVec are likely to be different rows of the same matrix and K_round_row will be a row of K_round.
- **OutVec = InitPermInv(InVec)** – performs the inverse of the initial permutation on InVec and returns it as OutVec.

Using these functions, write the function **Encrypt** which performs DES encryption and decryption.

- **Cipher = Encrypt(Plain, K, Enc, AllCipher) –** encrypts the bit array Plain using the Key K and returns it as Cipher. Enc is a binary flag – when true, the algorithm is to perform encryption, when false the algorithm is to perform decryption. AllCipher is a binary flag. When AllCipher is true, the algorithm is to return a matrix Cipher, where each row is the intermediate result after each round and when AllCipher is false, the algorithm simply returns the final output ciphertext.

For display purpose, the bit arrays should be represented as hexadecimals. Depending on the language of implementation, you may already have an inbuilt function to perform this operation. Otherwise, you can write one yourself. Whenever you are asked to report on the plaintext, ciphertext, key or intermediate encryption result, you must use hexadecimal.

Question 2 (10 marks)

Using the functions written above, write another function called Q2() that performs a few tests using the test data below and the Encrypt() function from Question 1. In your report, include tables like the one below reflecting the results you obtain.

Make sure that your code produces the expected outcome before you move on to the rest of the assignment.

Encryption

Plaintext	Key	Ciphertext (expected)	Ciphertext (actual)
6C31CF791B12932A	51043F77F9E9E50B	9AC1FCFF2B60F480	
AB7C3ED2E238A947	145297E2D918921B	6EC3B87248D751F0	

Decryption

Ciphertext	Key	Plaintext (expected)	Plaintext (actual)
2B75D7C9704BDC53	A5C516D871971C50	7DA16C7B7AD81C4E	
A190A87B3F9293AA	9D71BDF38CA994FF	5E2A221DD2166A19	

Question 3 (15 marks)

Now that you have implemented a working DES encryption algorithm, aside from planning your well-deserved celebratory parties, you can now characterise the algorithm. In particular, we will look at confusion and diffusion. As you will be running many of these experiments in both Questions 3 and 4, you should write a function that returns a random bit array to generate random plaintexts and keys.

- **Write a function Q3a() that compares the intermediate results of the encryption algorithm between two different encryptions, where the plaintexts are identical but the keys have a bitwise difference of 1[2].**
- **Write a function Q3b()that compares the intermediate results of the encryption algorithm between two different encryptions, where the keys are identical but the plaintexts have a bitwise difference of 1.**

In each instance, give the intermediate results of the encryption algorithm with the bitwise difference between the encryptions after every round. Discuss what is meant by confusion and diffusion and report on whether or not the results obtained are consistent with DES offering good encryption.

Question 4 (35 marks)

Run the experiment of Question 3 many times over and statistically analyse the results.

Pick a large number of encryptions to perform, such that your code runs for about 10 minutes[3][4][5]. Try get at least 10000 encryptions. Each encryption will be independent i.e. the plaintexts and keys of one encryption are generated randomly and not related to the plaintexts and keys of a different experiment. Report on how many encryptions you use. This should be done once to test diffusion and once to test confusion.[6]

Once you have the performed a large number of encryptions, construct a histogram of the bitwise difference after every round of encryption for both the confusion and diffusion experiments. In other words, there should be 32 histograms. For each distribution, you should also give the mean and standard deviation of the bitwise difference. Plot the mean and standard deviation against the number of rounds of encryption. Give these histograms and means and standard deviation plots in your report.

Using this information, imagine you wished to reduce the computational cost of DES by reducing the number of rounds. Ignoring all other considerations and only looking at the measures we have been considering (bitwise difference resulting from a single bit change in plaintext or key), how many rounds of DES encryption are necessary? Justify with support from your statistical analysis.

---

[2] Make sure that when you randomly flip a bit you do not flip a parity bit as this will be thrown away and you will end up with functionally identical keys

[3] Outputting text to the console in some IDEs can be time-consuming in comparison to the substantive parts of the algorithm, so remove this if it is slowing your code down. Also, before you run this for the full number of encryptions, try running it for a lower number, say 100 experiments to make sure the mechanics of your code works well, before you increase the accuracy of your results by running it for a longer time.

[4] If you are using Matlab, it is quite important that you preallocate the variables you are working with inside a loop, changing the variable size in the middle of a loop is time consuming.

[5] If you are extremely smart, you may want to rework your code to work in batch – i.e. perform encryptions and decryptions in parallel – this will mean converting all your vectors to matrices and your matrices to tensors. Also, it is probably not worth it, as you won't be running your code long enough to benefit from this speed up.

[6] NOTE: You might want to separate out your code for running the experiments and your code for analysing the results. You should also save your results before you analyse them so that if you inadvertently mess with your results while performing your analysis, you can simply reload the results.