DES Decryption Demystified

Scot Anderson, Ph.D.

January 22, 2013

1 Introduction

DES provides an excellent sample of symmetric encryption for students. We provides theoretical and practical instruction for the implementation of DES in C♯. For instruction purposes we use the original 56-bit keys and 64-bit data blocks. However, we do not consider block chaining. We examine DES from an outside in approach describing the algorithm at a high level and then taking more detailed looks at each part. Section 2 describes the elements of the algorithm. Section 3 takes an in-depth at each of the operations. Section 4 develops the concepts for students will need to code the algorithm for the class.

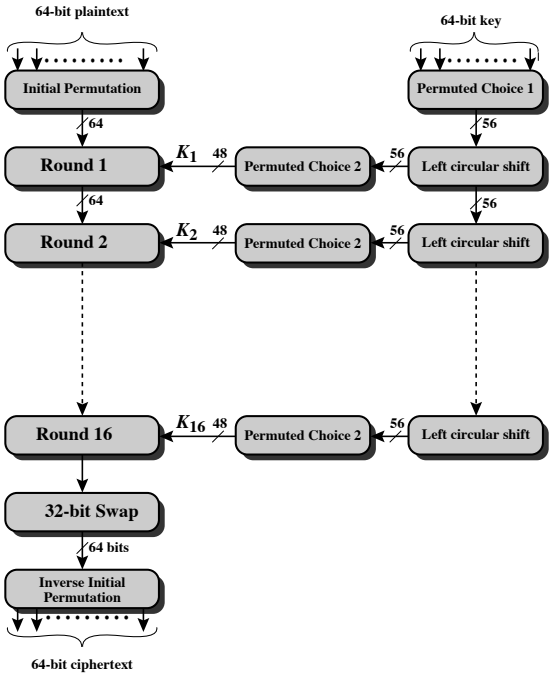

Figure 3.4 General Depiction of DES Encryption Algorithm

Figure 1: High-level view of the Feistal Cipher.

2 Theory from the outside in

DES uses the Feistal Cypher to encrypt and decrypt messages with the same key. The basic structure encrypts one 64-bit block of data at a time. Each block is encrypted by the following steps shown in Figure 1

1. An initial permutation
2. Multiple rounds (16) that repeatedly modify the block using a a function of the key
3. A final 32-bit swap
4. Inverse of the initial permutation

To start we have the Data D permuted as follows.

$$D' = IP(D)$$

Then it is separated into two parts such that

$$D' \rightarrow L_0||R_0$$

Each round uses the left and right 32-bit halves from the previous step and a unique 56-bit key derived from the 64-bit key given by the user. Hence we generate a unique key for each of the 16 rounds. The key generation algorithm does not depend on the encryption and can execute before the operations performed in the rounds. Your algorithm *should perform this step and store the keys* for later use when encrypting multiple blocks. (i.e. for data that contains more than 64 bits, break up the data into 64-bit blocks where each block must go through the same encryption using the same keys.)

The rounds themselves are fairly straightforward. The i^{th} round is given by the following.

1. Input of a Left 32-bit (labeled L_{i-1}) and right 32-bit block (labeled R_{i-1}).
2. The new left output $L_i = R_{i-1}$
3. The new right input $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

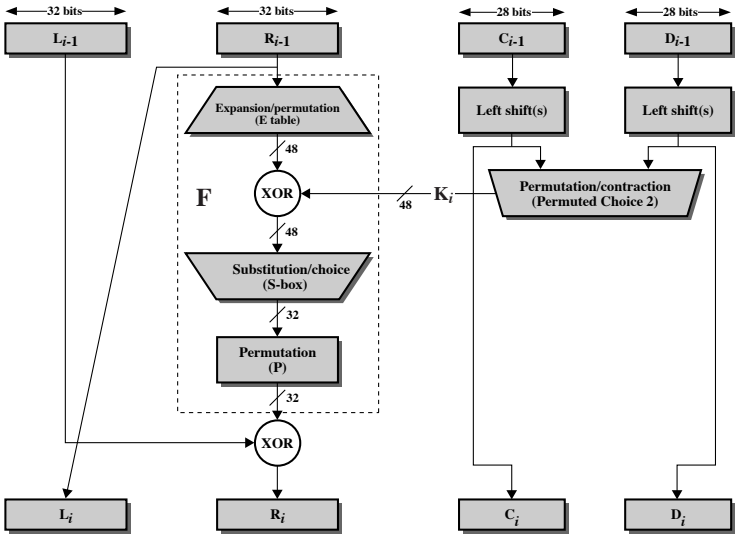

Figure 3.5 Single Round of DES Algorithm

Figure 2: Single Round

Figure 2 shows the single round for both the block encryption and the key generation. The number of rounds in DES is not important to see that decryption works exactly the same way except with inverted key order.

Consider the following theoretical example that walks through the process of encryption and decryption.

Example 1 *Let L0 and R0 represent the left and right 32-bit halves of the block after the initial permutation. These two 32-bit blocks go through the rounds in DES as shown below where rows labeled e_i show the results of encryption after the i^{th} round and d_i show decryption after the i^{th} round.*

e_0	$L0$	$R0$
e_1	$R0$	$F(R0, K0) \oplus L0$
e_2	$F(R0, K0) \oplus L0$	$F(F(R0, K0) \oplus L0, K1) \oplus R0$
$\dots$	$\dots$	$\dots$
$swap$	$F(F(R0, K0) \oplus L0, K1) \oplus R0$	$F(R0, K0) \oplus L0$
$\dots$	$\dots$	$\dots$
d_0	$F(F(R0, K0) \oplus L0, K1) \oplus R0$	$F(R0, K0) \oplus L0$
d_1	$F(R0, K0) \oplus L0$	$F(F(R0, K0) \oplus L0, K1) \oplus F(F(R0, K0) \oplus L0, K1) \oplus R0$
d_1	$F(R0, K0) \oplus L0$	$R0$
d_2	$R0$	$F(R0, K0) \oplus R(R0, K0) \oplus L0$
d_2	$R0$	$L0$
$Swap$	$L0$	$R0$

The left and right sides are combined to get D' and the decrypted dated is given by

$$D = IP^{-1}(D')$$

Including more rounds will produce much more complicated expressions at each level, but the decryption process is identical. More rounds may be added in the ellipses. Adding more rounds would require an ellipses before and after the swap ending the encryption part of table.

This gives a good overview of the flow of operations in the DES algorithm, but it does not tell us much about the details of each operation. Let us consider that details next.

3 Operations In-depth

Consider Figure 2. The operations shown there and in the previous section consist of:

1. IP, initial permutation that happens before the rounds in DES.
2. E, Expansion Permutation (in F)
3. S-Box, Substitution Choice (in F)
4. P, permutation (in F)
5. P2, permutation choice 2 (for key generation)
6. XOR (in various places in the round)
7. Left Shift (for key generation)

3.1 Permutations: All Types

All permutations behave identically even though they may reduce or expand the original input. Consider the following useful definition of a permutation.

Definition 2 *Let A be a binary string of length a and let B be a binary string of length b. A permutation consists of a integer array P of length b such that the number in the i^{th} position of P identifies the value at the position in A that should be copied to the i^{th} position in B. Graphically the center block in Figure 3 represents the permutation.*

In an expansion permutations, elements from the input must appear multiple times in the output. Whereas in contraction permutations, elements from the input may not appear in the output.

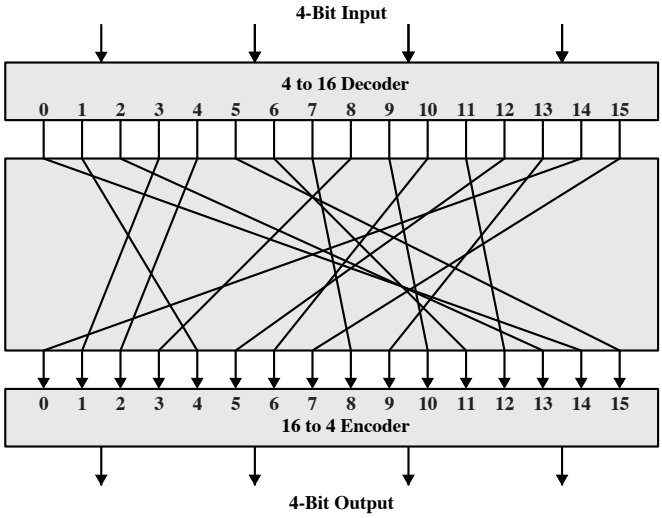

Figure 3.1 General *n*-bit-*n*-bit Block Substitution (shown with *n* = 4)

Figure 3: Permutation

3.2 S-Box Substitutions

Figure 4 shows the F function logic that occurs in each round of DES. The expansion takes the right 32-bit half and expands it to 48. This result is then XOR'd with the 48-bit round key. The 48-bit result of the XOR is then input into the mysterious S-Boxes. As we see in Figure 4, each S-Box takes 6 bits as input and outputs 4 bits.

S-Boxes are no more than plain ordinary tables. The output of an S-Box is determined by the row and column that the input specifies. The row is identified by concatenating the first bit to the last bit to yield a two bit number indicating row 0, 1, 2, or 3. The middle 4 bits are used as a number 0, 1, ..., 15 to identify the column. The four bits that are in that intersection of the row and column are the output of the S-Box.

3.3 XOR and Shift

These last two operations are extremely simple to perform on unsigned integers and we leave a discussion of these two operators to the next section.

4 Coding Considerations and the Assignment

This programming assignment must be designed and built using test driven design. That means you need to build each individual piece of functionality and test it thoroughly before integrating it with the rest of the program. Trust me, you will save yourself hours of troubleshooting by following this model. The C# project that I give you contains several tests that your code must pass.

To implement test driven design you may choose to download and install Visual Studio (latest version) from Dream Spark on the computing resources website. MSDN's unit testing walkthrough gives an extensive document that you may find useful.

I will grade your program based on an interface called IDesGradable that you will find along with other requirements at the class website.

Hints AND additional requirements:

For **S-Boxes** use an array of two dimensional arrays to store the actual S-Boxes. Then the first index identifies the S-Box, The second index identifies the row and the last index identifies to column.

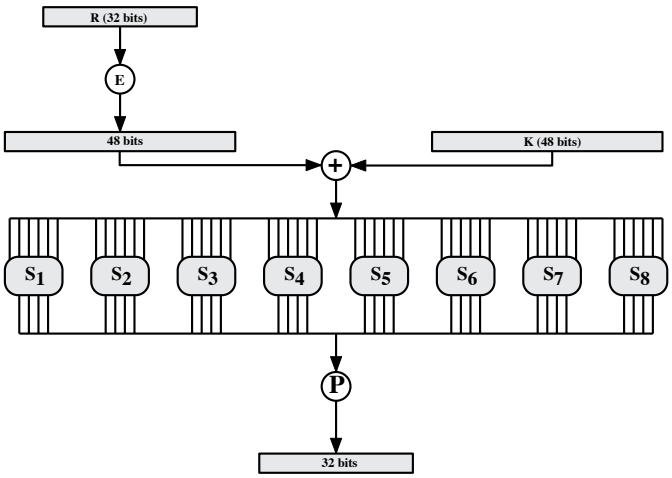

Figure 3.6 Calculation of F(R, K)

Figure 4: The F function

For **permutations** you probably should pass in the input and an array that maps the input to the output as discussed above. Standardizing on left justifying your output and input will significantly simplify your life. You will see in the skeleton code and in the documentation that this is assumed.

Numerous operations require you to *select a specific bit from an unsigned long*. It may be worth your time to code a simple move bit operation that takes your unsigned long and an input position and output position. The function should return the input position value OR'd in the proper output with zero. That way you possibly have just one bit, in the output, located at the proper location.

Not all files are multiples of 64 bits. RFC 3852 Section 6.3 describes how you should include padding when encrypting a file so that you can recover the original file sans padding.