Data Encryption Standard(DES)

Data Encryption Standard(DES)

Note: All of the look up tables (referred to later in these notes) for DES can be found here:

http://orion.towson.edu/~mzimand/cryptostuff/DES-tables.pdf

Here is the basic algorithm used for DES:

To encrypt a plaintext x of 64 bits and a secret key K of 56 bits do the following:

1) Compute $x_0 = IP(x)$, a fixed permutation of the bits in x. IP is specified in the link above.

2) Let $x_i = L_iR_i$, for $0 \leq i \leq 16$, where L_i is the 32 leftmost bits of x_i and R_i is the 32 rightmost bits of x_i. Make the following sequence of computations:

for (i=1 to 16) {
$L_i = R_{i-1}$
$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$
}

Essentially, each loop iteration is known as a Feistel round. (Feistel is the creator of DES.) DEs comprises 16 of these rounds. Each round encrypts ½ of the bits from the pervious round. The function f and the key for the ith round K_i will be discussed in detail later in these notes.

3) $y = IP^{-1}(R_{16}L_{16})$, this means applying the inverse permutation applied in step 1 to the string $R_{16}L_{16}$. (Notice the “reverse” order of the two blocks L_{16} and R_{16}.)

In essence, you would repeat this process for every block of 64 bits that needs to be encrypted.

Now, we need to mention the details of step 2. First the function f:

The first input to f, R_{i-1} is 32 bits, while the second input K_i is 48 bits from the 56 bits of the key K.

1) Expand the 32 bits of R_{i-1} to 48 bits using the matrix E, which is also shown in the link on the first page. This matrix delineates an ordering of the bits of R_{i-1} where 16 of the bits are repeated. Let this computed value be $E(R_{i-1})$.

2) Compute $E(R_{i-1}) \oplus K_i$. Let this computation produce $B = B_1B_2...B_8$, where each B_j, $1 \leq j \leq 8$ is 6 bits of B.

3) This is probably the strangest part of the algorithm. In this step the 48 bits of B need to be reduced to 32 bits. This is done via 8 S-boxes, S_1, S_2, ... S_8. One way to think about these S-boxes is the following. Each is a lookup table with 4 rows and 16 columns with 64 entries. Each entry corresponds to the output for a given input. In essence, an S-box specifies a function from 6 binary bits to 4 binary bits. Compute $C_j = S_j(B_j)$ for , $1 \leq j \leq 8$. Let $C = C_1C_2...C_8$. The details of how to use the S-boxes are below.

4) $f(R_{i-1}, K_i) = P(C)$, where P is a fixed permutation of the bits in C. (P is included in the link.)

<u>How to use the S-boxes (in link)</u>

Let the 6 input bits be $b_1b_2b_3b_4b_5b_6$. Let $R = b_1b_6$, a binary value that ranges from 0 to 3, and $C = b_2b_3b_4b_5$, a binary value ranging from 0 to 15. R will tell you the row to look on in the S-box. (Top row is 0, bottom is 3.) S will tell you the column to look on in the S-box. Each value in an S-box is from 0 to 15. This corresponds to 4 binary bits, the output. Let's practice a little bit:

Calculate $S_1(101110)$

We are using S-box 1. We are going to row = 10 = 2, column = 0111 = 7. The entry in row 2, column 7 in S-box 1 is **11.** Just to make sure you're in the right place, directly to the left of the 11 is 2 and to its right is 15.

Calculate $S_3(010110)$

We are using S-box 3. We are going to row = 00 = 0, column = 1011 = 11. The entry in row 0, column 11 in S-box 3 is **7.** Just to make sure you're in the right place, directly to the left of the 7 is 12 and to its right is 11.

Calculate $S_8(100101)$

We are using S-box 8. We are going to row = 11 = 3, column = 0010 = 2. The entry in row 3, column 0 in S-box 8 is **2.** Just to make sure you're in the right place, directly the right of the 2 is a 1 and to the right of that 1 is a 14.

Examples of Applying any of the permutation matrices (IP, P, E)

The method to apply the permutation matrices IP, IP^{-1}, E and P are all the same.

Let's just use E as an example:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Let's say the input to E, in hexadecimal is:

```
A687 DE29
```

Expanded to binary it's

```
1010
0110
1000
0111
1101
1110
0010
1001
```

Now, the matrix says, first grab the 32^{nd} bit, then the 1^{st} bit, 2^{nd} bit, 3^{rd} bit, 4^{th} bit and 5^{th} bit:

```
110100
```

Continuing, we have the following (with the first row repeated):

```
110100
001101
010000
001111
111011
111100
000101
010011
```

Converted back to HEX we have: `D0D 40F EFC 163`