编码引论第二次仿真实验报告

无35　陈馨瑶　2013011166

2015年12月2日

目录

1 接口及分工

本次实验所采取的加解密方式为DES，我们小组将实验内容分为如下三个部分，每个部分由一个人完成，由我完成的是第二部分。各部分接口说明如下（这里只列出了本次实验新加入的接口）：

1.1 PART I

1.1.1 子密钥生成 keygen

- 输入：key: 64bits original key
- 输出：subkeys: 16 cells, each contains one 48bits subkey

1.2 PART II

1.2.1 密码函数 Feistel

- 输入：R: Half Block（长度32，logical array），key: 密钥（长度48，logical array)
- 输出：feistel_out: f(R, k)（长度32，logical array）

1.3 PART III

1.3.1 加密 encrypt

- 输入：data: 输入数据流，key: 64 位密钥
- 输出：encrypted: 加密数据流

1.3.2 解密 decrypt

- 输入：encrypted: 加密数据流，key: 64 位密钥
- 输出：data: 解密后数据流

1.3.3 密钥生成 create_key

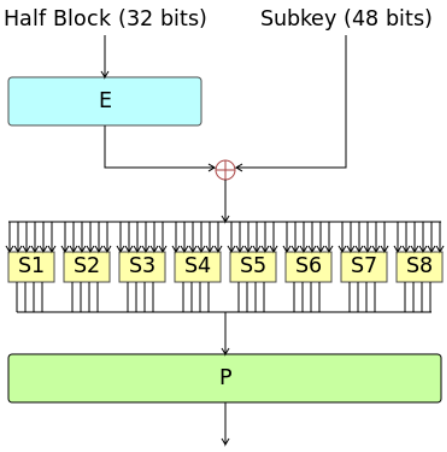
- 输入：num: 所需密钥数目，默认为 1，
- 输出：key: num 个密钥，每列为一个 64 位密钥

1.3.4 主程序 main

- 计算误比特率。
- 以 64 bit 为单位，画出误码图案。

2 模块设计 Feistel

实验中由我完成的Feistel模块流程如下图所示：



在DES算法中，Feistel函数的一个输入为以64bit长度分组后的数据的一半，在加密过程中为每一轮的R_i，另一输入为当前轮48bit子密钥。Feistel函数的步骤如下：

- 扩展置换
- 密钥混合
- 压缩置换
- 重排输出

2.1 扩展置换 Expansion

将输入的32bit数据扩展成48bit。采用的方式为：首先将该32bit数据分成8组，每组4bit，然后在这4bit头尾加上上一组和下一组与之紧邻的1bit数据，将得到的每组6bit数据拼接成48bit，由此完成扩展。

这一步可以采用的完成方式有查找表和循环，我采用的是循环方式。需要注意的是最后一组的末尾加上的是第一位，第一组的头部加上的是最后一位。

2.2 密钥混合 Key-mixing

这一步直接将上一步得到的48bit数据和当前轮子密钥进行逐位异或即可。

2.3 压缩置换 Substitution

将异或得到的48bit数据分成8个6bit块，利用每个块的6bit数据运算得到4bit输出。

这里的运算方式为查找表。实验时，利用randi生成8个元素范围在0-15的矩阵，每个块对应一个不同的4×16矩阵，

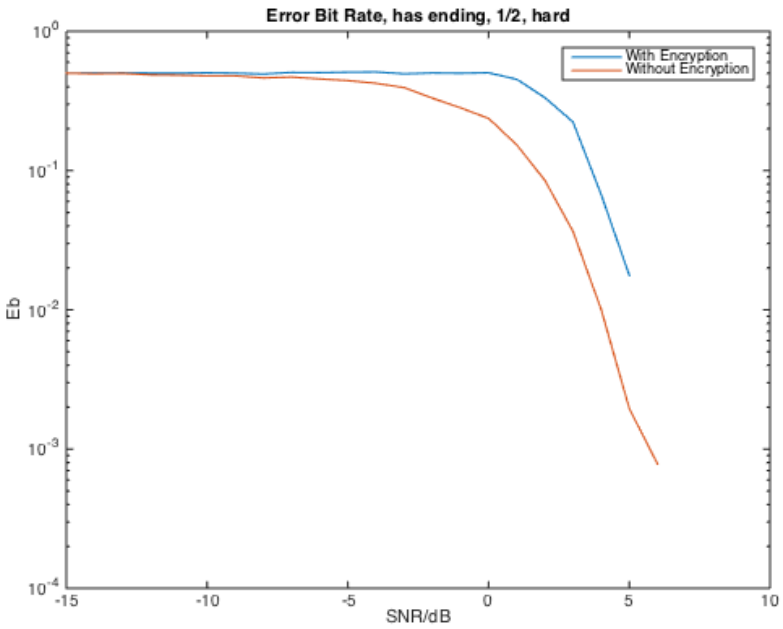
2.4 重排输出 Permutation

利用查找表方式，将上一步得到的每块4bit打散到4个不同的块中，最后得到32bit数据作为输出。

3 仿真结果

3.1 误比特率

误比特率曲线如下图所示，其中，蓝色表示加密过的结果，红色表示未经过加密的结果。



从图中可以看到，蓝色曲线和红色曲线的变化趋势一致，但在相同SNR下经过加密的信号的误比特率明显高于未经加密的信号，这说明在经过加密后，由于信道噪声带来的误码得到了放大。基于窃听者的信道质量较接收方差得多的假设，此次实验中所设计的加密机制的确起到了保障信息安全的作用。

3.2 误码图案

为了更直观地说明加密的效果，我们画出了SNR = 1dB时经过加密和未经加密的误码图案，如下：

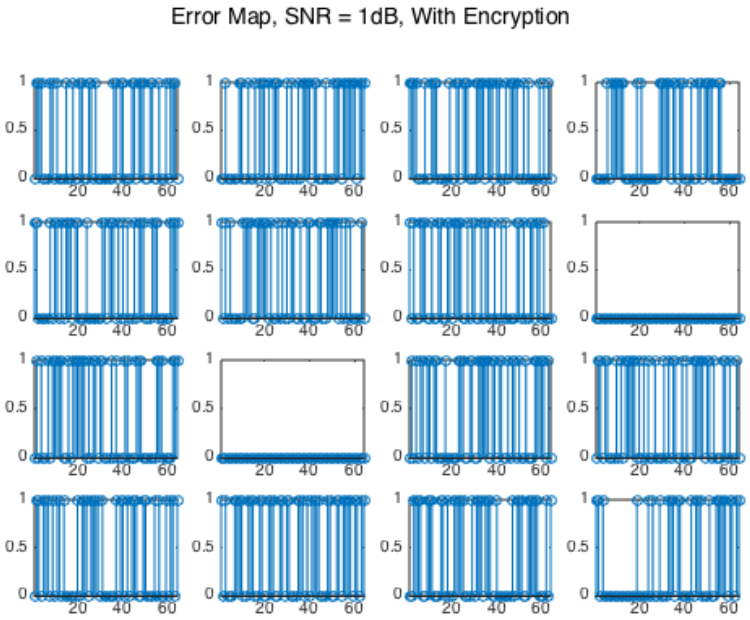


图 1: SNR=1dB，经过加密

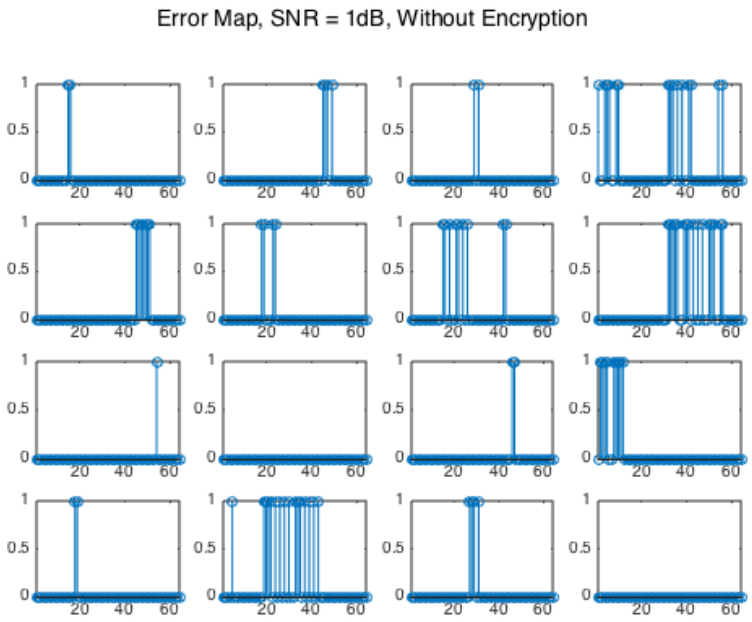


图 2: SNR=1dB，未经加密

两图对比可以看出，其实当SNR = 1dB时由信道噪声本身造成的误码已经很少，而且在大多数情况下都是属于零散的突发错，然而，在经过加密后，误码几乎完全是以“误块”形式出现的。这说明在解密时，很小的一点（几比特）偏差就会影响与其相邻的很多比特数据。这也说明了设计的加密机制的雪崩效应足够好。

4 附：代码

我所完成部分的代码如下：Feistel.m

```
function [ feistel_out ] = Feistel( R, key )
    %% expansion
    S = zeros(8, 6);
    for k = 1: 8
        S(k, 2: 5) = R(4*(k-1)+1: 4*k);
        if k == 1
            S(1, 1) = R(32);
        else
            S(k, 1) = R(4*(k-1));
```

```
        end
        S(k, 6) = R(mod(4*k+1, 32));
    end
    S = reshape(S', 48, 1);

    %% key mixing
    key_mixed = xor(S, key);
    key_mixed = reshape(key_mixed, 6, 8);
    key_mixed = key_mixed';

    %% Substitution
    % randomly generated
    S_box = {[2, 10, 13, 8, 9, 10, 8, 0, 12, 0, 8, 1, 0, 9, 5, 10;...
              6, 8, 15, 5, 6, 10, 11, 12, 2, 13, 10, 2, 2, 4, 4, 2;...
              13, 11, 11, 1, 14, 10, 8, 3, 13, 9, 0, 2, 10, 2, 11, 11;...
              12, 10, 5, 9, 0, 15, 15, 7, 15, 15, 9, 9, 5, 3, 0, 1],...
             [0, 2, 9, 12, 7, 3, 3, 11, 8, 8, 5, 9, 14, 14, 0, 1;...
              6, 2, 1, 6, 6, 11, 1, 5, 14, 7, 0, 0, 1, 1, 10, 10;...
              8, 15, 14, 1, 7, 3, 1, 11, 9, 12, 7, 14, 15, 3, 9, 5;...
              6, 2, 14, 4, 12, 1, 1, 6, 2, 3, 3, 11, 8, 0, 8, 10],...
             [10, 0, 13, 2, 5, 9, 6, 10, 3, 7, 1, 11, 11, 7, 11, 11;...
              10, 8, 4, 4, 12, 7, 7, 11, 6, 14, 3, 1, 15, 0, 11, 9;...
              4, 14, 9, 7, 7, 7, 5, 7, 11, 9, 2, 13, 4, 14, 12, 11;...
              6, 10, 0, 8, 0, 10, 12, 0, 13, 13, 3, 14, 6, 3, 4, 3],...
             [0, 3, 6, 7, 2, 12, 10, 5, 12, 11, 0, 15, 7, 1, 11, 11;...
              15, 5, 5, 14, 11, 5, 12, 6, 5, 9, 10, 13, 12, 4, 8, 15;...
              2, 7, 2, 8, 7, 10, 14, 4, 8, 3, 4, 12, 13, 7, 6, 13;...
              1, 15, 2, 15, 2, 6, 15, 3, 1, 10, 8, 8, 1, 1, 0, 1],...
             [5, 2, 6, 10, 5, 13, 3, 13, 1, 1, 11, 2, 2, 15, 12, 5;...
              3, 13, 1, 15, 9, 13, 2, 6, 2, 10, 7, 6, 5, 5, 5, 5;...
              7, 10, 9, 3, 3, 4, 11, 14, 10, 10, 8, 2, 0, 4, 9, 10;...
              5, 6, 7, 10, 11, 9, 1, 6, 7, 11, 7, 0, 8, 0, 11, 9],...
             [15, 3, 11, 4, 3, 9, 8, 12, 3, 14, 1, 15, 5, 4, 1, 12;...
              14, 6, 11, 10, 14, 8, 8, 6, 7, 15, 7, 4, 2, 0, 2, 5;...
              0, 7, 10, 11, 4, 13, 13, 12, 2, 12, 13, 4, 3, 8, 8, 3;...
              11, 1, 0, 1, 12, 4, 7, 12, 0, 9, 13, 5, 14, 12, 7, 1],...
```

```
46              [4,  9,  1,  4,  3,  5,  6,  6,  13,  14,  4,  7,  10,  10,  14,  12;...
47               6,  3,  5,  3,  4,  1,  10,  3,  8,  9,  3,  10,  7,  1,  12,  3;...
48               8,  6,  8,  10,  1,  15,  11,  12,  14,  0,  9,  0,  14,  1,  11,  6;...
49               15,  9,  10,  13,  9,  10,  8,  15,  11,  1,  10,  13,  1,  12,  0,  8],...
50              [6,  4,  6,  5,  10,  7,  5,  5,  9,  13,  6,  8,  11,  14,  1,  3;...
51               15,  4,  13,  12,  8,  10,  2,  10,  13,  7,  3,  13,  11,  8,  1,  10;...
52               4,  9,  11,  10,  6,  8,  9,  7,  14,  13,  15,  5,  8,  1,  12,  7;...
53               11,  4,  15,  0,  10,  10,  4,  13,  15,  3,  1,  7,  2,  13,  15,  2]};
54
55      sub_S = zeros(8, 4);
56      for k = 1: 8
57          row = 2*key_mixed(k, 1) + key_mixed(k, 6) + 1;
58          col = 8*key_mixed(k, 2) + 4*key_mixed(k, 3) + 2*key_mixed(k, 4) +
                key_mixed(k, 5) + 1;
59          d = S_box{k}(row, col);
60          for p = 4: -1: 1
61              sub_S(k, p) = mod(d, 2);
62              d = floor(d/2);
63          end
64      end
65      sub_S = reshape(sub_S', 1, 32);
66
67      %% Permutation
68      P = [16,  7,  20,  21,...
69           29,  12,  28,  17,...
70           1,  15,  23,  26,...
71           5,  18,  31,  10,...
72           2,  8,  24,  14,...
73           32,  27,  3,  9,...
74           19,  13,  30,  6,...
75           22,  11,  4,  25];
76       feistel_out  = zeros(1, 32);
77      for k = 1: 32
78          feistel_out (k) = sub_S(P(k));
79      end
80       feistel_out  =  feistel_out ';
```

```
81
82 end
```