Report for course:

Introduction to Information Security

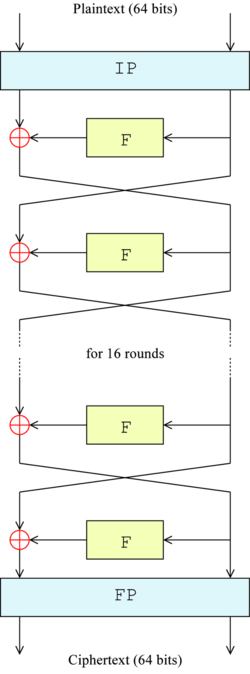
2019.10

\*本报告包含三个部分，本别为DES，AES，Blowfish

# Data Encryption Standard

1. Problem Description

DES是一种对称密钥加密块密码算法，1976年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS）。尽管过短的密钥长度令其一经提出就备受质疑，但DES仍是一个成功的算法，作为标准使用了20余年。本次实验使用python进行DES算法的加密与解密。



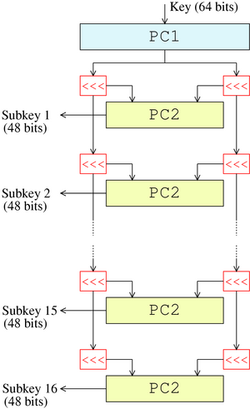
2. Algorithm Description

2.1. Overall structure

右图为DES的流程图，IP为置换，之后按前后顺序分为左32bits和右32bits。接着进行16轮加密运算，每轮内交换左右块（最后一轮不再交换）。F为一个运算，输入为右块和子密钥，其输出与左块相加（异或）。最后一轮结束后进行FP（IP的逆置换）得到密文。由于DES是对合运算，解密仍按照此流程图，区别仅为子密钥反序。

2.2. Details

**IP置换**

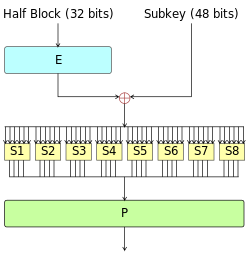
即1-64到1-64的一个一一对应，置换后的串第位等于原串的位。计算机实现为查表

**子密钥生成**

右图为子密钥生成流程图

首先经过PC1置换将64位密钥变为56位，并且分为左右两块各28位。然后每轮各循环左移位，取决于轮数，每轮输出为左右两块一起进行PC2置换得到此轮子密钥

**F函数**

右块32位首先进行E置换扩展位48位，然后与子密钥相加（异或）。得到48位串，按位置分割为8个6位字串。依次输入8个S盒。每个S盒为4\*16矩阵，S盒每个元素为四位子串。

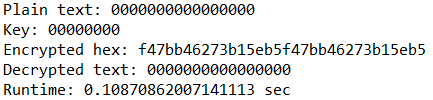
对应关系，例：

**101111** 首尾 **11** = 3 中间 **0111** = 7，

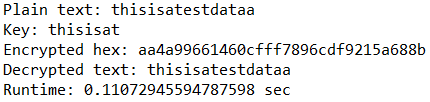
则其S盒输出为S盒底3行第7列的元素

得到8个四位字串再组合到一起进行P置换得到F函数输出。

3. Result



Encrypted hex: f47bb46273b15eb5f47bb46273b15eb5



Encrypted hex: aa4a99661460cfff7896cdf9215a688b

4. Code

# -\*- coding: utf-8 -\*-

"""

@author: Chenghai Li

"""

import time

IP = [58, 50, 42, 34, 26, 18, 10, 2,

      60, 52, 44, 36, 28, 20, 12, 4,

      62, 54, 46, 38, 30, 22, 14, 6,

      64, 56, 48, 40, 32, 24, 16, 8,

      57, 49, 41, 33, 25, 17, 9, 1,

      59, 51, 43, 35, 27, 19, 11, 3,

      61, 53, 45, 37, 29, 21, 13, 5,

      63, 55, 47, 39, 31, 23, 15, 7]

PC\_1 = [57, 49, 41, 33, 25, 17, 9,

        1, 58, 50, 42, 34, 26, 18,

        10, 2, 59, 51, 43, 35, 27,

        19, 11, 3, 60, 52, 44, 36,

        63, 55, 47, 39, 31, 23, 15,

        7, 62, 54, 46, 38, 30, 22,

        14, 6, 61, 53, 45, 37, 29,

        21, 13, 5, 28, 20, 12, 4]

PC\_2 = [14, 17, 11, 24, 1, 5, 3, 28,

        15, 6, 21, 10, 23, 19, 12, 4,

        26, 8, 16, 7, 27, 20, 13, 2,

        41, 52, 31, 37, 47, 55, 30, 40,

        51, 45, 33, 48, 44, 49, 39, 56,

        34, 53, 46, 42, 50, 36, 29, 32]

E = [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]

S\_box = [

[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

 [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

 [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],

 [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],

],

[[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],

 [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],

 [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],

 [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],

],

[[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],

 [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],

 [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],

 [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],

],

[[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],

 [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],

 [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],

 [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],

],

[[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],

 [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],

 [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],

 [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],

],

[[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],

 [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],

 [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],

 [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],

],

[[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],

 [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],

 [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],

 [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],

],

[[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

 [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],

 [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],

 [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],

]

]

P = [16, 7, 20, 21, 29, 12, 28, 17,

     1, 15, 23, 26, 5, 18, 31, 10,

     2, 8, 24, 14, 32, 27, 3, 9,

     19, 13, 30, 6, 22, 11, 4, 25]

IP\_1 = [40, 8, 48, 16, 56, 24, 64, 32,

        39, 7, 47, 15, 55, 23, 63, 31,

        38, 6, 46, 14, 54, 22, 62, 30,

        37, 5, 45, 13, 53, 21, 61, 29,

        36, 4, 44, 12, 52, 20, 60, 28,

        35, 3, 43, 11, 51, 19, 59, 27,

        34, 2, 42, 10, 50, 18, 58, 26,

        33, 1, 41, 9, 49, 17, 57, 25]

Shift = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def encode (s):

    return [int(d) for d in ''.join([bin(ord(c)).replace('0b', '').zfill(8) for c in s])]

def decode (s):

    return ''.join([chr(int(''.join([str(c) for c in s[i:i+8]]),2)) for i in range(0, len(s), 8)])

def get\_hex(s):

    return ''.join([hex(int(''.join([str(c) for c in s[i:i+8]]),2)).replace('0x', '').zfill(2) for i in range(0, len(s), 8)])

def change(s, M):

    length = len(M)

    out = [0] \* length

    for i in range (length):

        out[i] = int(s[M[i]-1])

    return out

def generate\_key(s,t):

    if t == 0:

        s = change(s, PC\_1)

    s = s[Shift[t]:28] + s[:Shift[t]] + s[Shift[t]+28:] + s[28:Shift[t]+28]

    out = change(s, PC\_2)

    return out, s

def F(r, k):

    r = change(r, E)

    for i in range (48):

        r[i] = r[i] ^ k[i]

    out = [0] \* 32

    for i in range (8):

        out[4\*i:4\*i+4] = [int(d) for d in list(bin(S\_box[i][int(str(r[6\*i])+str(r[6\*i+5]),2)][int(''.join([str(c) for c in r[6\*i+1:6\*i+5]]),2)]).replace('0b', '').zfill(4))]

    out = change(out, P)

    return out

def encrypt(t, k):

    t = encode(t)

    k = encode(k)

    t = change(t, IP)

    tl, tr = t[0:32], t[32:64]

    for i in range (16):

        ki, k = generate\_key(k, i)

        for j in range (32):

            tl[j] ^= F(tr, ki)[j]

        if i != 15:

            tl, tr = tr, tl

    t = tl+tr

    t = change(t, IP\_1)

    return t

def decrypt(t, k):

    #t = encode(t)

    k = encode(k)

    t = change(t, IP)

    tl, tr = t[0:32], t[32:64]

    k\_list = []

    for i in range (16):

        ki, k = generate\_key(k, i)

        k\_list.append(ki)

    for i in range (16):

        for j in range (32):

            tl[j] ^= F(tr, k\_list[15-i])[j]

        if i != 15:

            tl, tr = tr, tl

    t = tl+tr

    t = change(t, IP\_1)

    return t

text = 'abcdef0123456789'

key = 'abcdefgh'

print('Plain text:',text)

print('Key:',key)

start = time.time()

code1, code2 = encrypt(text[0:8], key), encrypt(text[8:16], key)

print('Encrypted hex:', get\_hex(code1) + get\_hex(code2))

plain1, plain2 = decrypt(code1, key), decrypt(code2, key)

print('Decrypted text:', decode(plain1) + decode(plain2))

end = time.time()

print('Runtime:',end - start,'sec')

2. Advanced Encryption Standard (AES)

1. Problem Description

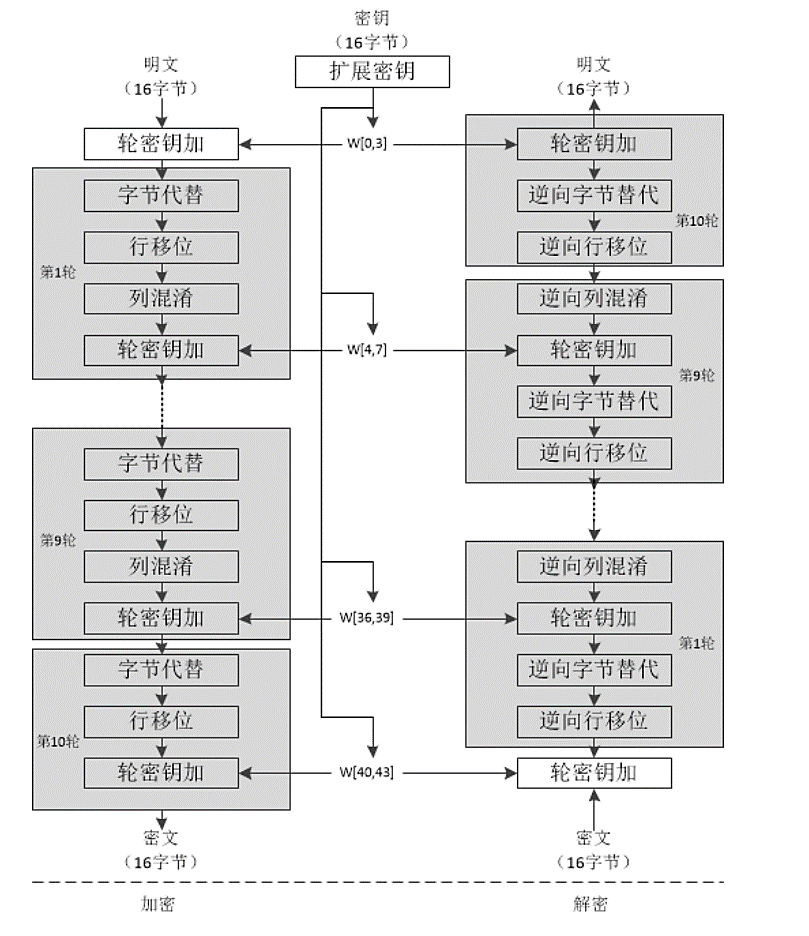
AES又称Rijndael加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于2001年11月26日发布于FIPS PUB 197，并在2002年5月26日成为有效的标准。2006年，高级加密标准已然成为对称密钥加密中最流行的算法之一。本次实验使用python进行AES-128算法的加密与解密。

| **AES** | **密钥长度（32位比特字)** | **分组长度(32位比特字)** | **加密轮数** |
| --- | --- | --- | --- |
| AES-128 | 4 | 4 | 10 |
| AES-192 | 6 | 4 | 12 |
| AES-256 | 8 | 4 | 14 |

2. Algorithm Description

2.1. Overall structure

AES-128输入为16字节的明文及16字节的密钥。加密过程共进行10轮（最后一轮无列混淆）。密钥则进行10次扩展分别作用于每一轮加密。解密过程为加密过程的逆向操作。先进行密钥扩展，然后逐轮解密还原明文。



2.2. Details

**轮密钥加**

即按位异或。例：**1101 ^ 1010 = 0111**

其反操作即再次进行异或：**0111 ^ 1010 = 1101**

故自身构成自身反操作

**字节代替**

即0-255的一个一一映射S

我们有表Sbox [16\*16] 其中的元素值为0-255

现在我们有数10011101，做一个分割1001|1101

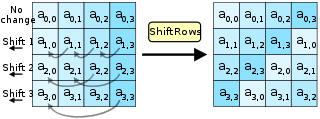
1001 = 9，1101 = 13，则S (10011101) = Sbox [9][13]

由于为一一映射，我们可以构造InvS，及对应的InvSbox [16\*16]

同样的InvS (10011101) = InvSbox [9][13]

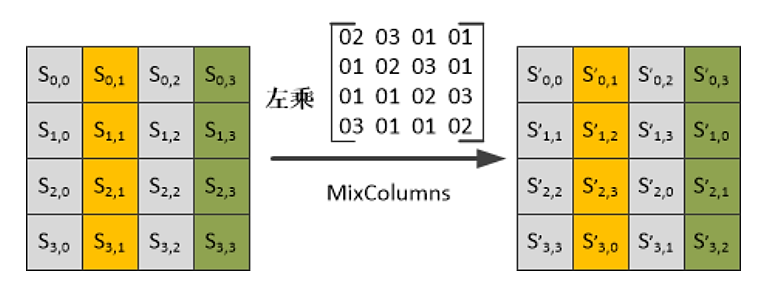
InvS即为S的逆操作

**行位移**



如图，行位移即第一行不动，第二行左移1格，第三行左移2格，第四行左移3格。明显的，行位移的逆操作即第一行不动，第二行右移1格，第三行右移2格，第四行右移3格

**列混淆**



如图，即左乘矩阵A。需注意的是这里的乘法是有限域内的乘法。我们取的既约多项式为。

考虑乘10，首先将待乘数右侧加0，若待乘数第九位为1，则需要模掉，即与100011011进行异或；若为0则不需要。例：

10 \* 10110011 = 101100110 ^ 100011011 = 01111101

10 \* 00110011 = 01100110

任意数相乘则分解后相加（异或）：

00010011 \* a = (10000 \* a) ^ (10 \* a) ^ (1 \* a)

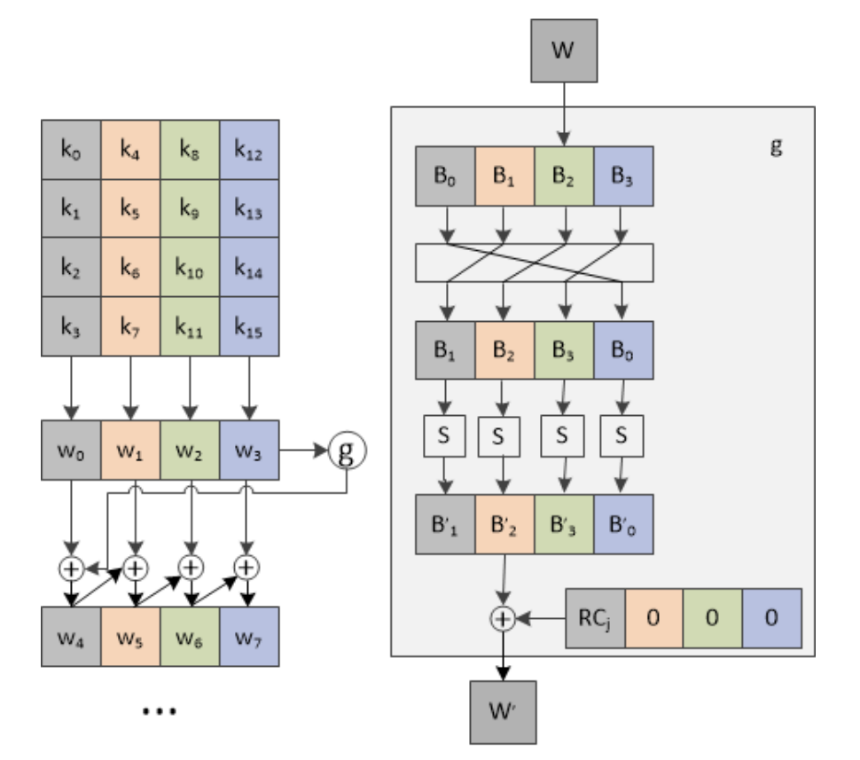
其中10000 \* a = 10 \* 10 \* 10 \* 10 \* a

列混淆的逆操作即左乘A的逆矩阵B，要求BA = I

=

如图最左端即为我们找到的B

**扩展密钥**



扩展密钥的操作不需要求逆。如上4\*4矩阵为16字节key组成的矩阵，考虑其列向量。首先对于，进行g操作。g操作首先对左移1位（此时为向量），然后进行字节代替，

最后第一位与给定的（表示轮数）

进行相加（异或），得到。

与相加（异或）得到，

与相加（异或）得到，

与相加（异或）得到，

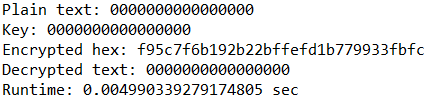
与相加（异或）得到，

完成一轮密钥扩展，依此类推。

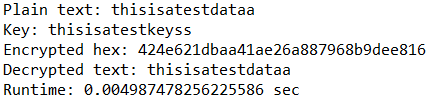
**解密**

由于每个主线步骤都有逆操作，则直接进行密钥扩展，逐轮解密。

3. Result



Encrypted hex: f95c7f6b192b22bffefd1b779933fbfc



Encrypted hex: 424e621dbaa41ae26a887968b9dee816

4. Code

# -\*- coding: utf-8 -\*-

"""

@author: Chenghai Li

"""

import numpy as np

import time

S = [

    [0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76],

    [0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0],

    [0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15],

    [0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75],

    [0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84],

    [0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF],

    [0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8],

    [0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2],

    [0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73],

    [0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB],

    [0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79],

    [0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08],

    [0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A],

    [0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E],

    [0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF],

    [0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16]

]

InvS = [

    [0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB],

    [0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB],

    [0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E],

    [0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25],

    [0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92],

    [0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84],

    [0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06],

    [0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B],

    [0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73],

    [0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E],

    [0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B],

    [0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4],

    [0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F],

    [0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF],

    [0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61],

    [0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D]

]

mix\_f = [

    [2,3,1,1],

    [1,2,3,1],

    [1,1,2,3],

    [3,1,1,2]

]

mix\_b = [

    [0x0E,0x0B,0x0D,0x09],

    [0x09,0x0E,0x0B,0x0D],

    [0x0D,0x09,0x0E,0x0B],

    [0x0B,0x0D,0x09,0x0E]

        ]

def encode (s):

    return [ord(c) for c in s]

def decode (s):

    return ''.join([chr(i) for i in s])

def s\_f (matrix):

    for i in range(4):

        for j in range(4):

            matrix[i][j] = S[matrix[i][j]//16][matrix[i][j]%16]

def s\_b (matrix):

    for i in range(4):

            for j in range(4):

                matrix[i][j] = InvS[matrix[i][j]//16][matrix[i][j]%16]

def shift\_row\_f (s):

    s[1][0], s[1][1], s[1][2], s[1][3] = s[1][1], s[1][2], s[1][3], s[1][0]

    s[2][0], s[2][1], s[2][2], s[2][3] = s[2][2], s[2][3], s[2][0], s[2][1]

    s[3][0], s[3][1], s[3][2], s[3][3] = s[3][3], s[3][0], s[3][1], s[3][2]

def shift\_row\_b (s):

    s[1][1], s[1][2], s[1][3], s[1][0] = s[1][0], s[1][1], s[1][2], s[1][3]

    s[2][2], s[2][3], s[2][0], s[2][1] = s[2][0], s[2][1], s[2][2], s[2][3]

    s[3][3], s[3][0], s[3][1], s[3][2] = s[3][0], s[3][1], s[3][2], s[3][3]

def xtime (a):

    if a & 0b10000000:  #mod x^8 + x^4 + x^3 + x + 1

        return ((a << 1) ^ 0b100011011)

    else:

        return a << 1

def mul (a, b):

    s = list(bin(a))

    i = 0

    out = 0

    while s[-i-1] != 'b':

        if s[-i-1] == '1':

            c = b

            for j in range(i):

                c = xtime(c)

            out = out ^ c

        i += 1

    return out

def mul\_matrix (a, b):

    out = np.empty((4,4),dtype=int)

    for i in range(4):

        for j in range(4):

            index = 0

            for k in range(4):

                index = index ^ mul(a[i][k], b[k][j])

            out[i][j] = index

    return out

def mix\_column\_f (s):

    return mul\_matrix(mix\_f, s)

def mix\_column\_b (s):

    return mul\_matrix(mix\_b, s)

RC = [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36]

def g(vector,t):

    v = [0]\*4

    for i in range(4):

        v[i] = vector[i]

    v[0],v[1],v[2],v[3] = v[1],v[2],v[3],v[0]

    for i in range (4):

        v[i] = S[v[i]//16][v[i]%16]

    v[0] = v[0] ^ RC[t]

    return v

def expand\_key (s):

    key\_list = [s]

    for i in range (10):

        new = np.empty((4,4),dtype=int)

        gw = g(key\_list[i][:,3],i)

        for j in range (4):

            for k in range (4):

                if j == 0:

                    new[k][j] = key\_list[i][k][j] ^ gw[k]

                else:

                    new[k][j] = key\_list[i][k][j] ^ new[k][j-1]

        key\_list.append(new)

    return key\_list

def build\_matrix(text):

    matrix = []

    code = encode(text)

    for i in range(16):

        byte = code[i]

        if i % 4 == 0:

            matrix.append([byte])

        else:

            matrix[int(i / 4)].append(byte)

    matrix = np.array(matrix).T

    return matrix

def decode\_matrix (s):

    out = []

    for i in range(4):

        for j in range(4):

            out.append(chr(s[j][i]))

    return ''.join(out)

def add\_matrix(a, b):

    for i in range (4):

        for j in range (4):

            a[i][j] = a[i][j] ^ b[i][j]

def forward(plain, key):

    key\_matrix = build\_matrix(key)

    expanded\_key = expand\_key(key\_matrix)

    plain\_matrix = build\_matrix(plain)

    for i in range (9):

        add\_matrix(plain\_matrix, expanded\_key[i])

        s\_f(plain\_matrix)

        shift\_row\_f(plain\_matrix)

        plain\_matrix = mix\_column\_f(plain\_matrix)

    add\_matrix(plain\_matrix, expanded\_key[9])

    s\_f(plain\_matrix)

    shift\_row\_f(plain\_matrix)

    add\_matrix(plain\_matrix, expanded\_key[10])

    return plain\_matrix

def back(code, key):

    key\_matrix = build\_matrix(key)

    expanded\_key = expand\_key(key\_matrix)

    plain\_matrix = build\_matrix(code)

    add\_matrix(plain\_matrix, expanded\_key[10])

    shift\_row\_b(plain\_matrix)

    s\_b(plain\_matrix)

    add\_matrix(plain\_matrix, expanded\_key[9])

    for i in range(8,-1,-1):

        plain\_matrix = mix\_column\_b(plain\_matrix)

        shift\_row\_b(plain\_matrix)

        s\_b(plain\_matrix)

        add\_matrix(plain\_matrix, expanded\_key[i])

    return plain\_matrix

def get\_hex(s):

    out = ''

    for i in range(4):

        for j in range(4):

            out += hex(s[j][i]).replace('0x', '').zfill(2)

    return out

text = 'abcdef0123456789'

key = 'abcdef0123456789'

print('Plain text:',text)

print('Key:',key)

start = time.time()

forward\_m = forward(text, key)

code\_hex = get\_hex(forward\_m)

print('Encrypted hex:',code\_hex)

code = decode\_matrix(forward\_m)

back\_m = back(code, key)

ori = decode\_matrix(back\_m)

print('Decrypted text:',ori)

end = time.time()

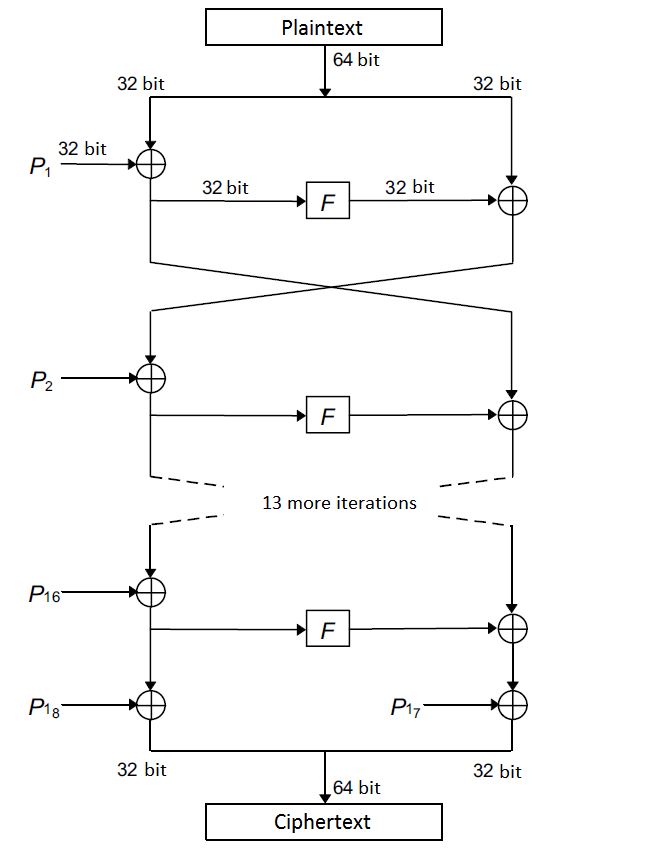
print('Runtime:',end - start,'sec')

# Blowfish

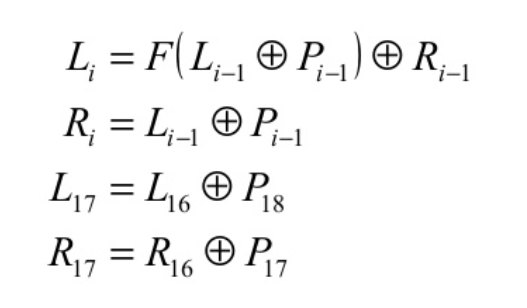
1. Problem Description

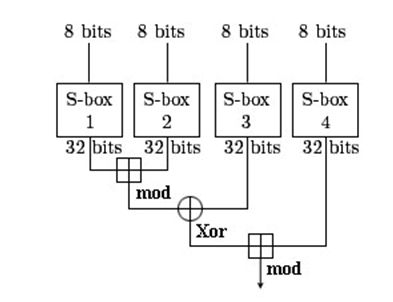
Blowfish是一个对称密钥加密算法，由布鲁斯·施奈尔于1993年设计，现已应用在多种加密产品。Blowfish能保证很好的加密速度，并且当前为止没有发现有效地破解方法。当前为止AES比Blowfish有更广的知名度。

施奈尔设计的Blowfish算法用途广泛，意在替代老旧的DES及避免其他算法的问题与限制。Blowfish刚刚研发出的时候，大部分其他加密算法是专利所有的或属于商业(政府)机密，所以发展起来非常受限制。施奈尔则声明Blowfish的使用没有任何限制，任何国家任何人任何时候都可以随意使用Blowfish算法。

2. Details

右图为Blowfish加密流程图，与DES类似，其有16轮加密，并且每轮交换左右块。基本公式见下。

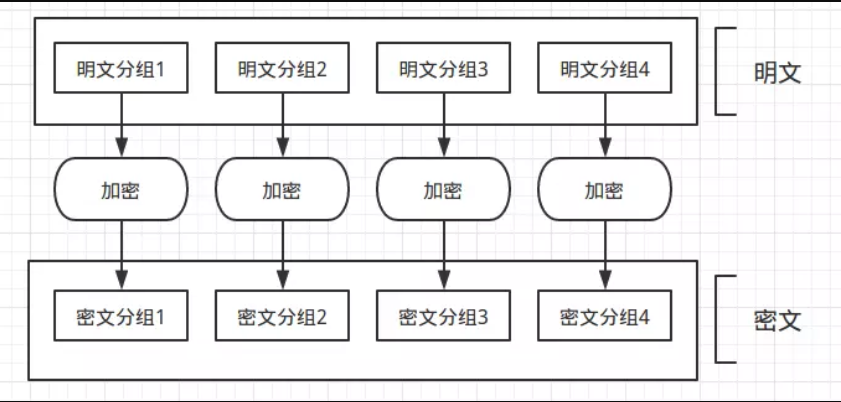


右图为F函数操作，mod为取模操作，为异或操作。

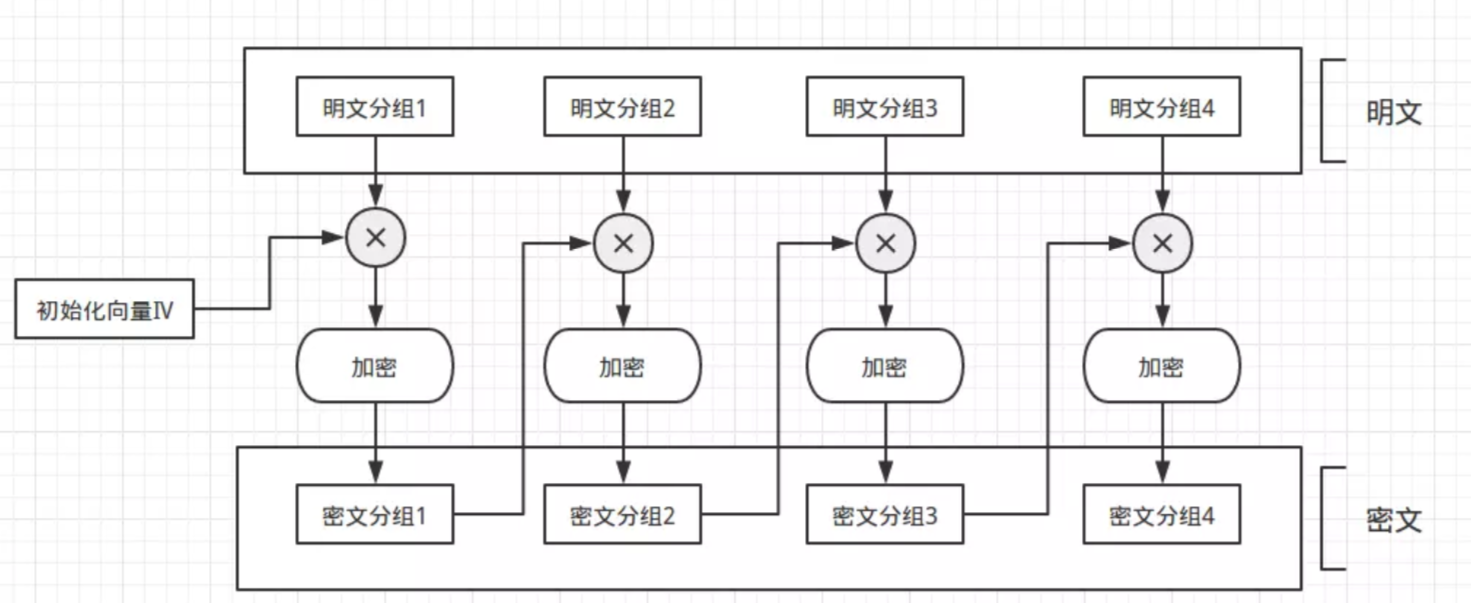
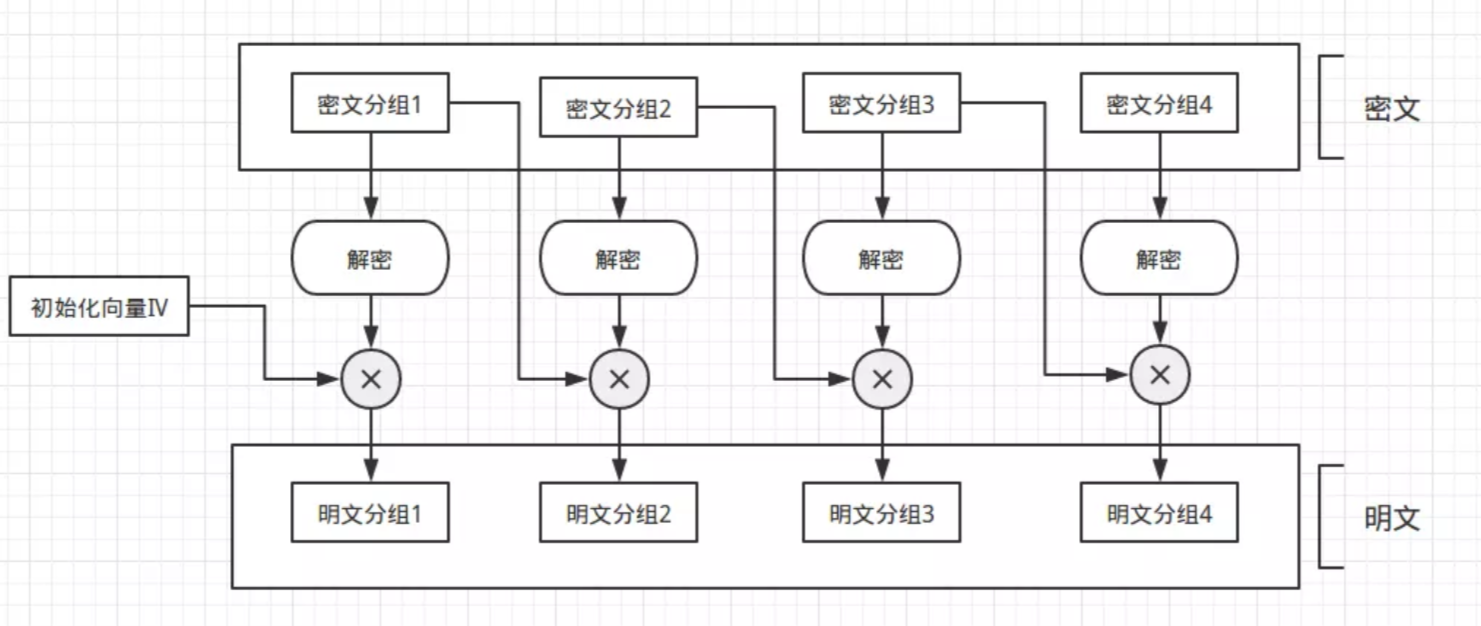
与DES类似，Blowfish为对合运算。解密的时候同加密过程，只需要反序密钥。

# Appendix

分组密码，每次只能处理特定长度的一块数据的一类密码算法，这里的“一块”就称为分组。一个分组的比特数就称为分组长度。这样对长序列加密就需要分组逐个加密。



ECB模式：即直接分组然后每组加密，缺点很明显，若加密图像(或其他具有长结构明文)，则加密后的图像仍保持基本结构。

CBC模式加密

CBC模式解密

全称Cipher Block Chaining mode，密码分组链接模式。

分组方式：将明文分组与前一个密文分组进行异或运算，然后再进行加密。每个分组的加解密都依赖于前一个分组。而第一个分组没有前一个分组，因此需要一个初始化向量（initialization vector）。

初此之外还有CFB、OFB等模式。