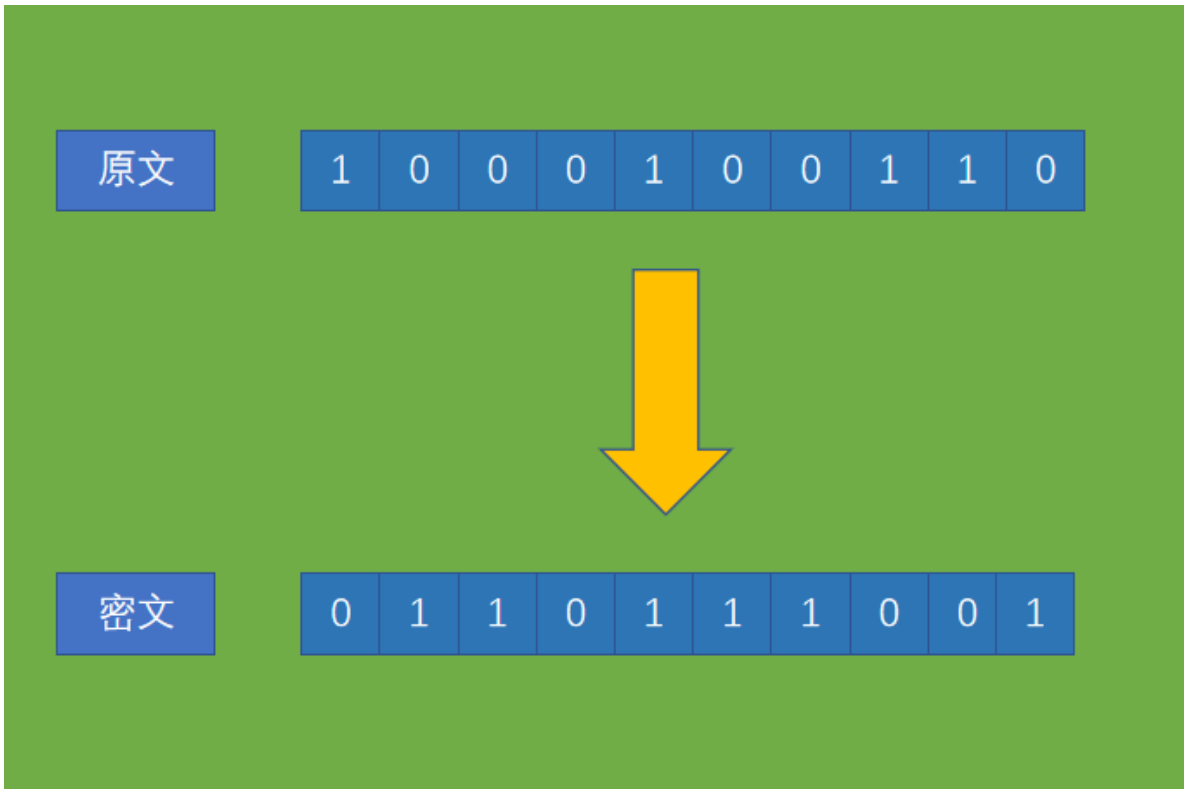
DES算法——从小白到登堂入室

从字符串信息到01比特数

首先什么是加密？加密，是以某种特殊的算法改变原有的信息数据，使得未授权的用户即使获得了已加密的信息，但因不知解密的方法，仍然无法了解信息的内容。(百度百科)

简单来说比如像下面这样：原文经过加密算法之后被加密成密文。



原文被加密成密文之后意义发生了改变窃听者就不能发现消息内容，这正是加密信息的意义。计算机中的信息在经过物理层的时候最终都会变成01比特流，加密也正是基于比特流，如果想变成信息只需经过编码即可。例如想Java/Python将字符串或数字变成比特流。

Python将数字变成01字符

```
num = 100
print(bin(num)) # 0b1100100
```

其中1100100正是100的二进制数，如果我们自己来实现可以这样

```
def to_bits(num, length):
    return [num >> (length - i - 1) & 1 for i in range(length)]

num = 100
print(to_bits(num, 7)) # [1, 1, 0, 0, 1, 0, 0]
```

对于字符串来说首先得进行字符编码编码，具体如下：

```
s = "Hello World"

encoded = s.encode("utf-8") # 使用 UTF-8 对 s 进行编码，编码的结果便是数据

for num in encoded:
    print(num, end=" ")
# 72 101 108 108 111 32 87 111 114 108 100
```

编码后的结果是数据(72 101 108 108 111 32 87 111 114 108 100)，那么就可以使用上述方法将数字变成01比特数，把得到结果连接起来便得到了字符串的01比特流。

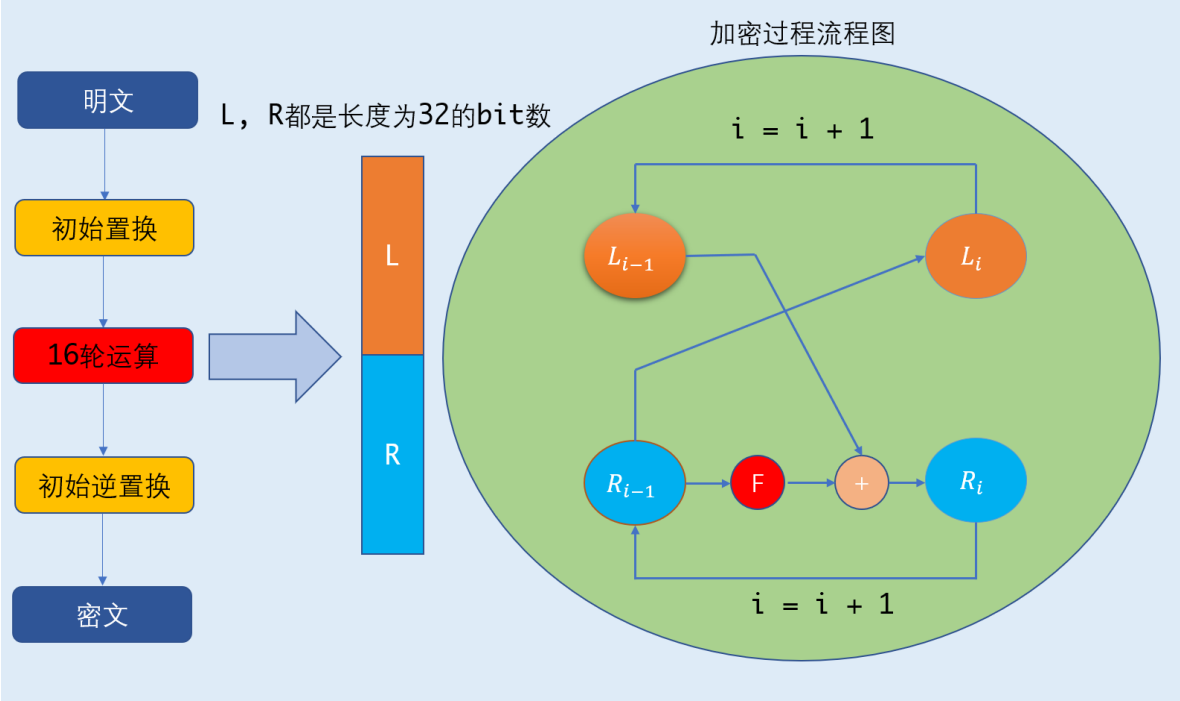
DES(Data Encryption Standard)是第一个广泛应用于商用数据保密的密码算法，虽然DES由于秘钥空间限制已经能被破解而被高级加密标准AES取代，但是它设计思想仍然有很重要的参考价值。下面就具体说明DES加密算法~~~

DES加密的关键过程主要有下面三个。

- 由初始秘钥生成子秘钥
- 轮函数
- 置换

DES算法流程

首先从整体上来了解一下DES加密的流程。



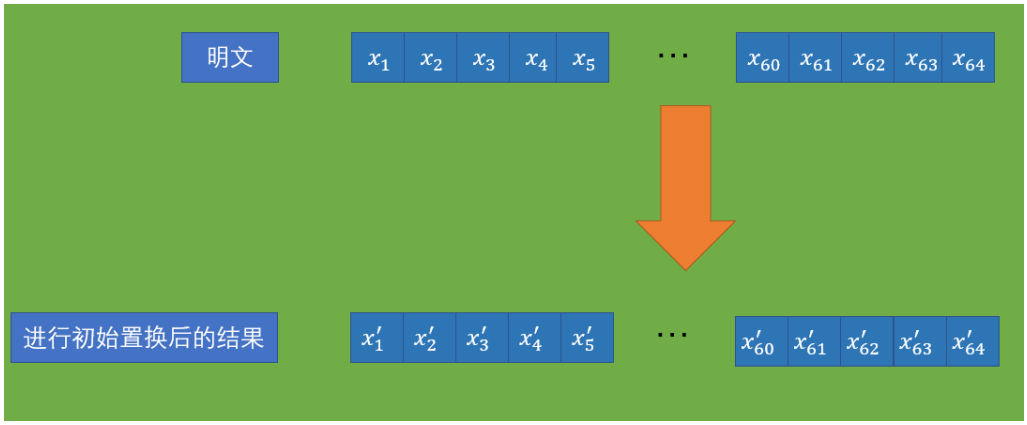
DES加密算法的明文的长度是确定的，是由64个0，1数字组成，秘钥也是如此由64个01数字组成。

- 首先明文经过初始置换得到初始置换后的信息T。
  - 进行初始置换首先需要一张初始置换表，如下

```
public static byte[][] initIPSub = {
            {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}
  };
```

一共8行8列64个数。置换规则(第i行第j列，原文为M置换结果为T，初始置换表为I):
$T_{(i*8+j+1)} = M_{I[i][j]}$，i，j从0开始。

例如第一行第一列58，即原文的M第58位为置换后的信息T的第一位，第二列：原文M第50位为T的第二位,以此类推.....

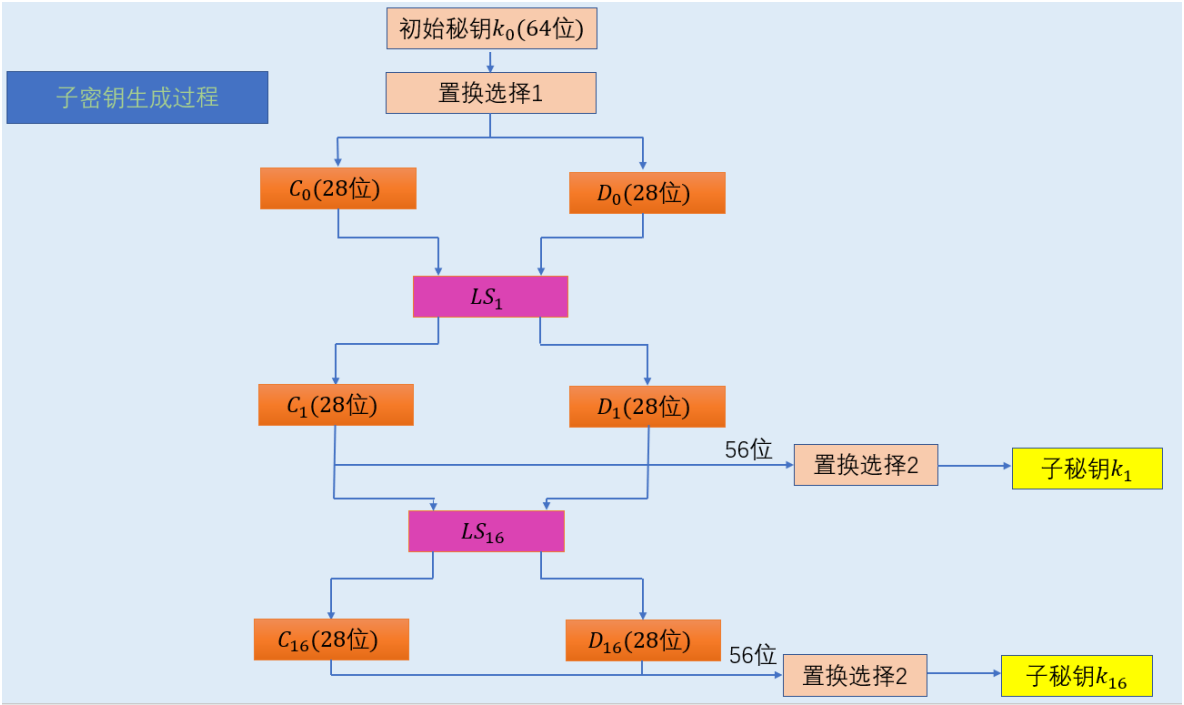


- 16轮运算，由于16轮运算比较复杂稍微再进行详解。
- 初始逆置换和初始置换表一样，都是8x8的置换表，只是置换表内容发生了变化，除此之外其余步骤一样。
    - 初始逆置换表

```
public static byte[][] initIPInverseSub = {
            {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}
    };
```

秘钥生成

在之后进行16轮轮函数运算时需要用到子秘钥，接下来将讲解具体的由初始秘钥生成子秘钥的过程。子秘钥生成流程图如下：



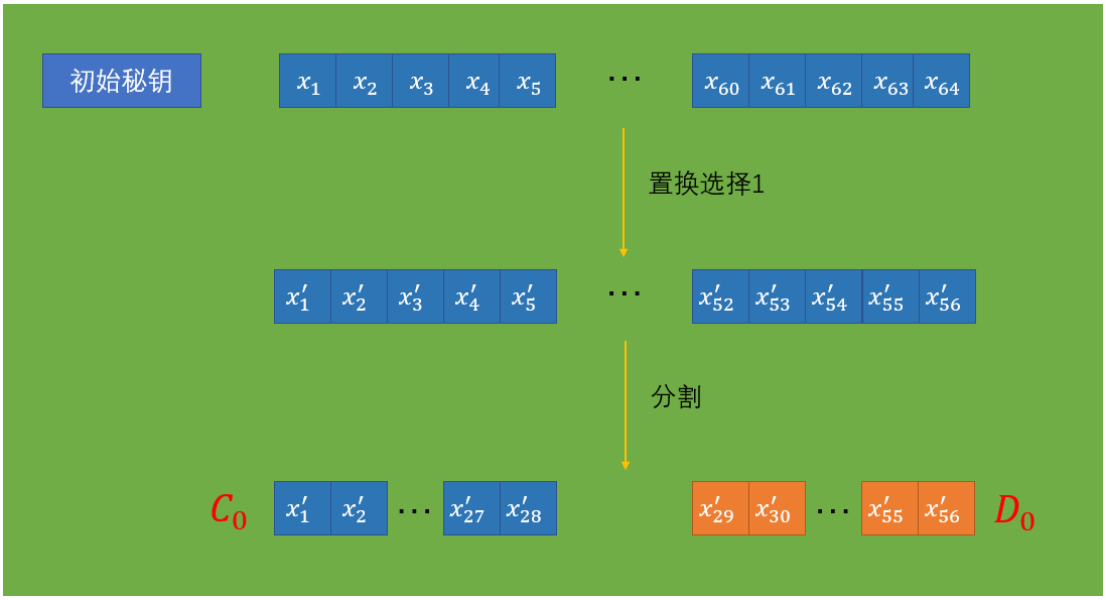
由上图初始秘钥(64位，提供用来加密的秘钥，整个DES加密需要提供的就是明文和秘钥)

- 置换选择1

  置换还是和之前初始置换和初始置换一样，只是表不同而已，置换选择1所用到的表如下：

```
public static byte[][] keySub1 = {
            {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}
    };
```

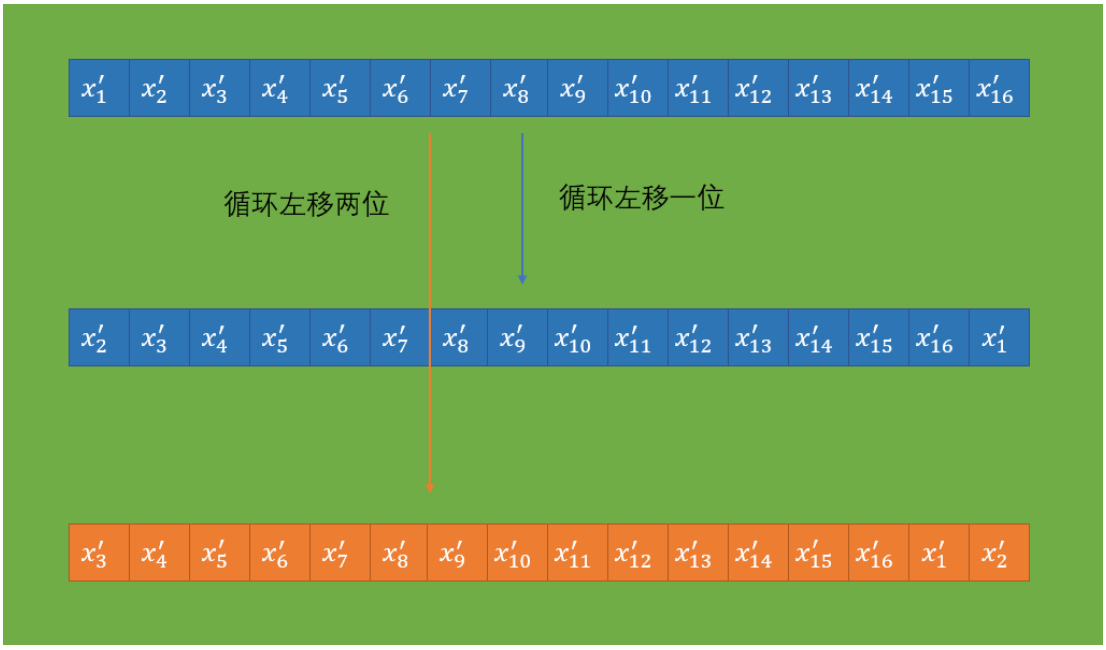
置换选择表1和之前的表不同的地方在于它是一个8x7的表，也就是说会将初始秘钥的64位变成56位，但是规则仍然是一样，只不过前面一共换了64位，这里只换56位而已。在进行初始置换之后再进行分割，将得到的56比特数字分割成左右两部分。



- LS_i函数

$$LS_i = \begin{cases} \text{循环左移一位}, & \text{i} \in \{1, 2, 9, 16\} \\ \text{循环左移两位}, & \text{i} \in \{3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15\} \end{cases}$$

循环左移示例



每一个C_i和D_i都需要经过LS_i输出C_{i+1}和D_{i+1}，并且C_{i+1}和D_{i+1}将作为LS_{i+1}的输入得到C_{i+2}和D_{i+2}以此类推.....

- 在经过LS_i函数之后得到C_{i+1}和D_{i+1}将他们拼接成$C_{i+1}D_{i+1}$再经过置换选择2得到一把子秘钥，置换规则和之前一样，置换选择2的内容如下：

```
public static byte[][] keySub2 = {
            {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}
  };
```

这是一张8x6的表，也就是说只置换48次，置换的子钥是一个48位的比特数。

加密过程

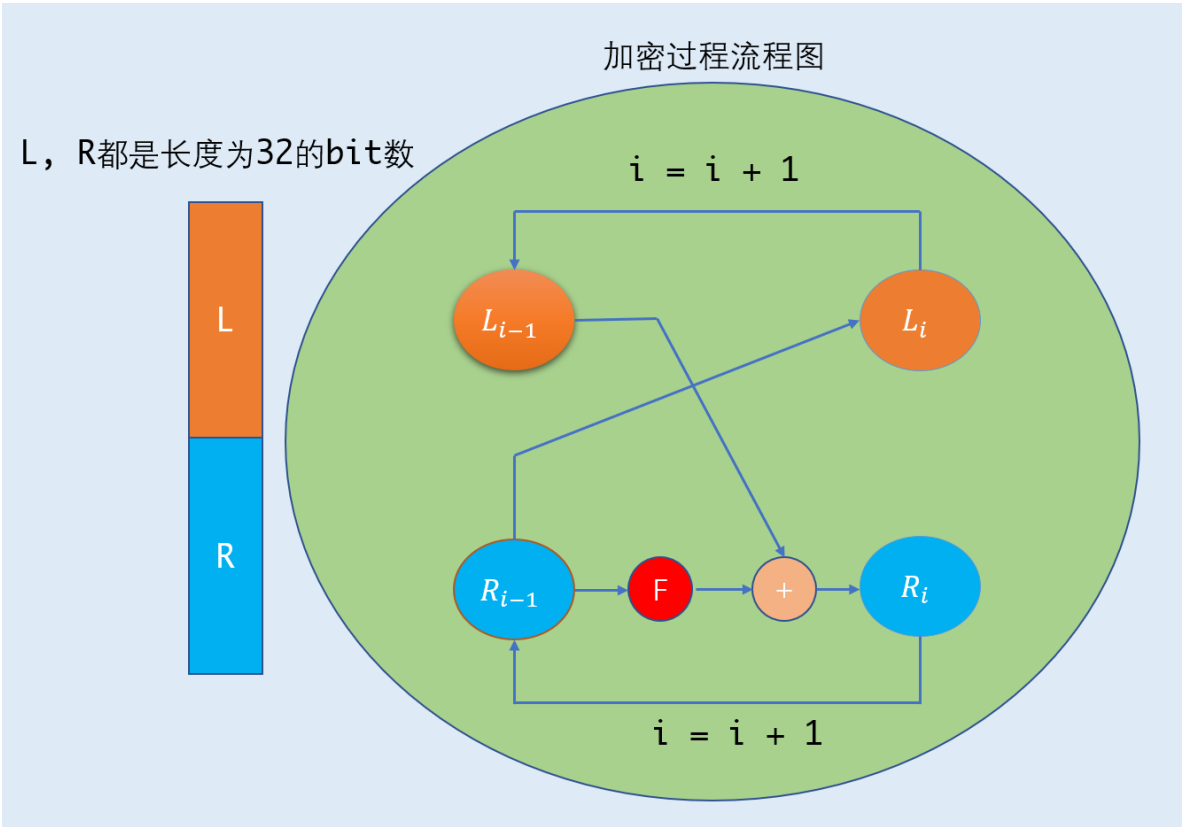
在上面我们完成了DES加密过程中的置换和子秘钥生成了，接下就开始具体了解加密过程了。首先我们将经过初始置换后的T分成左右两个部分，拆分方法和上面秘钥经过置换选择1的分法一样，只不过位数发生了变化，将T分成左右两部分L,R他们各占32位。其中：

$$L_0 = x_1^{'}x_2^{'}x_3^{'}\dots x_{30}^{'}x_{31}^{'}x_{32}^{'}$$
$$R_0 = x_{33}^{'}x_{34}^{'}x_{35}^{'}\dots x_{62}^{'}x_{63}^{'}x_{64}^{'}$$

然后就需要利用L_0和R_0进行16次的加密过程了，加密过程的算法如下，其中F是轮函数，它的输出一个32位的比特数之后会说明它的具体实现，$\bigoplus$是异或运算：

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \bigoplus F(R_{i-1}, K_i) \end{cases} \qquad \begin{array}{l} i = 1, 2, \dots, 16 \\ K_i\text{是子秘钥生成过程中生成的第}i\text{把秘钥} \end{array}$$

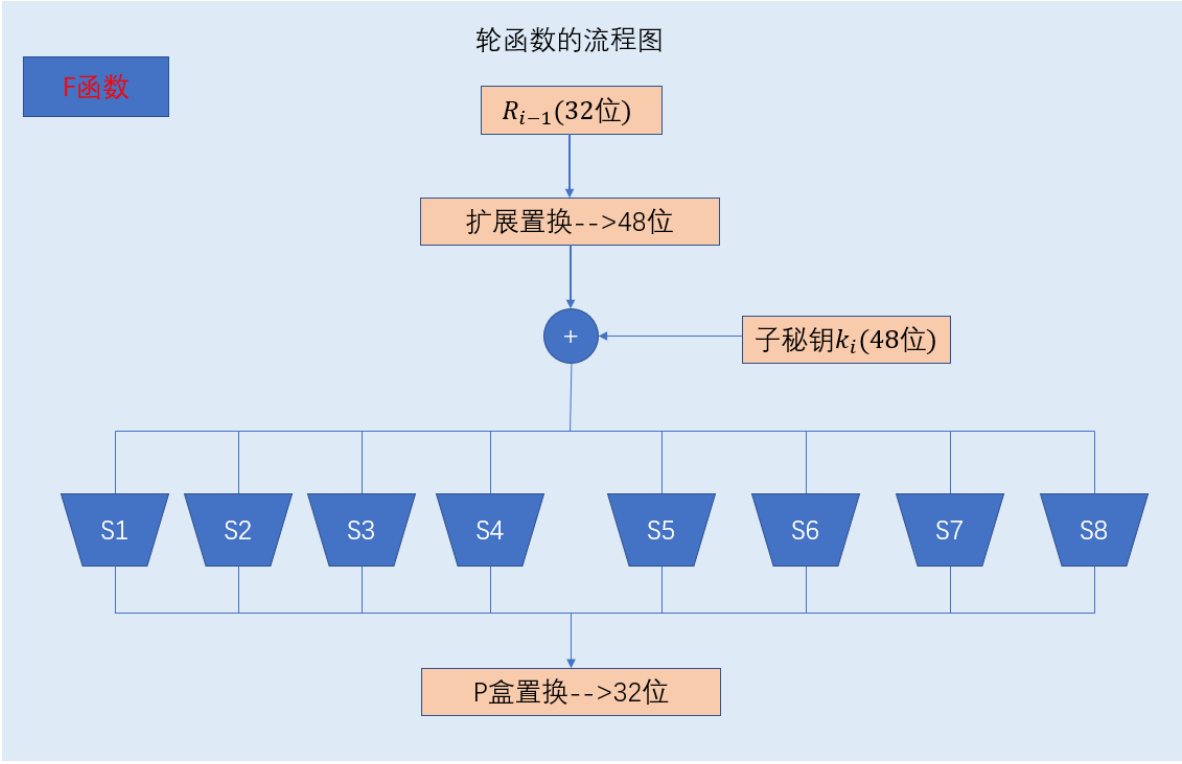
加密流程图如下图所示：



在经过16轮加密运算之后得到的R_{16}, L_{16}都是32位的比特数，将他们拼接成$R_{16}L_{16}$，注意不是$L_{16}R_{16}$，而是$R_{16}L_{16}$。在经过初始逆置换(具体可以查看最开始的DES加密流程图)就可以得到密文了。

轮函数

根据上面很容易知道，在进行加密的时候对信息L_i, R_i的迭代需要使用到轮函数F，接下来对它进行详解。轮函数流程图如下：



根据公式：

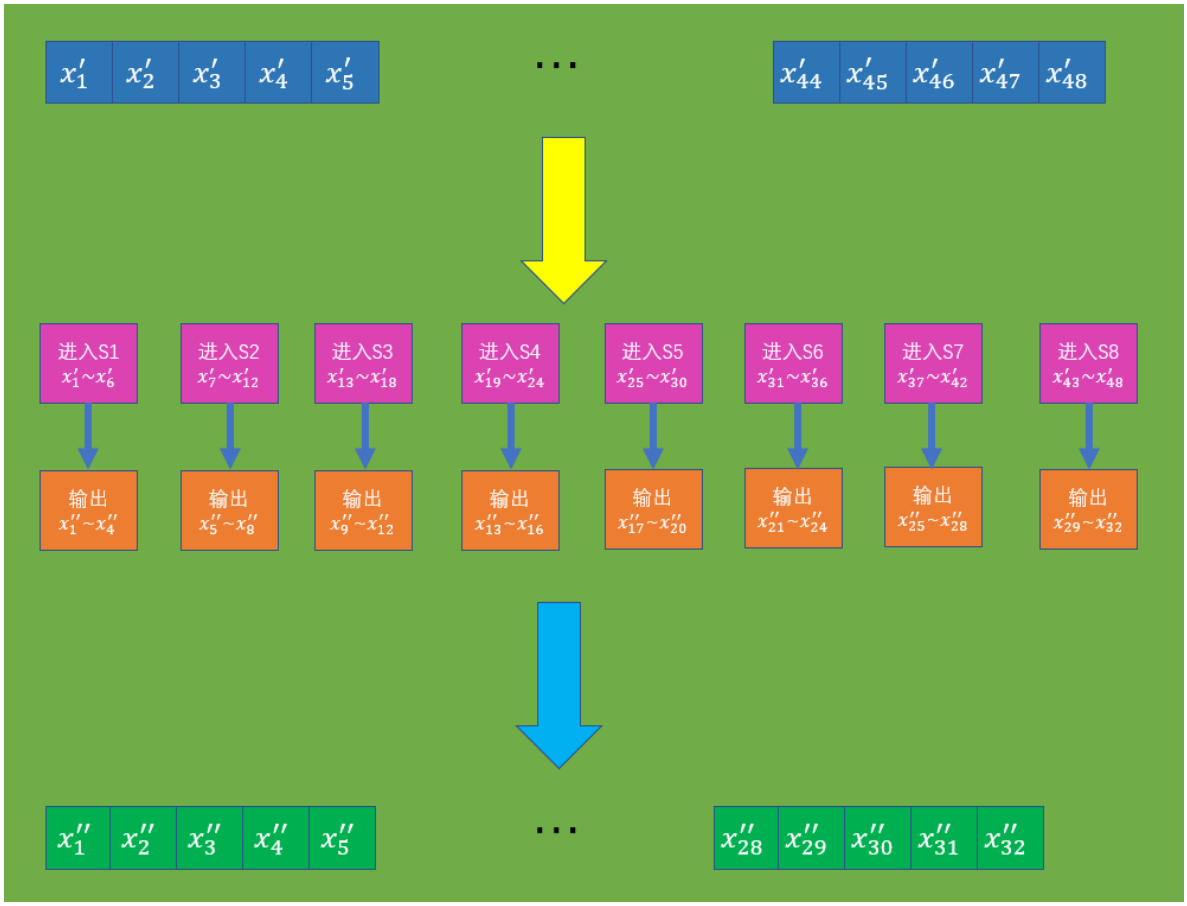
$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \bigoplus F(R_{i-1}, K_i) \end{cases} \qquad i = 1, 2, \ldots, 16$$

K_i是子秘钥生成过程中生成的第i把秘钥

轮函数的输入是R_{i-1}和K_i，首先R_{i-1}通过扩展置换由32变成48位，扩展置换也和之前的置换方式一致，它也有它的置换表，如下：

```
public static byte[][] extendsTable = {
            {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}
    };
```

一共8行6列，因此置换结果有48位。扩展置换得到结果再与48为的子秘钥K_i进行异或运算，得到一个新的48位的结果，下面就需要将得到的48为的结果分解成8份，每份含有6位的比特数，分割方式如下：



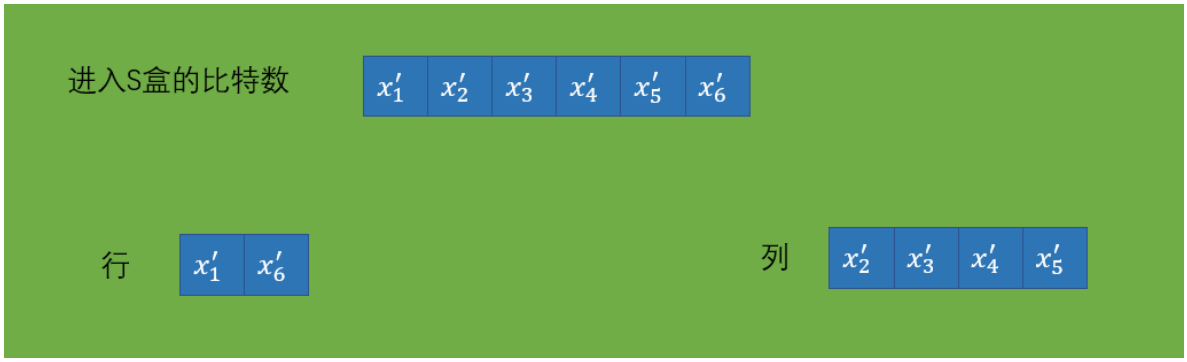
这里的S_i统称为S盒，一共8，每个S的输出是一个4位的比特数，范围就是：0000 ~ 1111，十进制表示就是从0到15。那么S盒是如何映射的呢？

首先先来了解S的结构，一共8个盒，每个盒的构成都一样，均为行为4列为16的表，具体如下：

```
public static byte[][] S1 = {
            {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}
    };
    public static byte[][] S2 = {
            {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}
    };
    public static byte[][] S3 = {
            {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}
    };
    public static byte[][] S4 = {
            {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}
    };
    public static byte[][] S5 = {
            {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}
    };
    public static byte[][] S6 = {
            {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}
    };
    public static byte[][] S7 = {
            {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}
    };
    public static byte[][] S8 = {
            {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}
    };
```

所有数的范围都是0~15，现在就是需要从每一个对应的S盒中找到要输出的数，然后将他转成二进制的形式输出。从表中找一个数就需要行和列，每个S的输入都是6个二进制数，取首尾两个数组成行的二进制数，范围在0~3，中间四位二进制数组成列范围在0~15，如下如所示：



举个例子：输入为110110，则行为10=2，列为1011=11(下标从0开始，对应表的第三行第12列)

经过S盒的结果在通过P盒置换就可以得到最终的F函数的输出结果了，P盒置置换还是盒之前一样，只是换了一张表而已，P盒置换的表如下图所示：

```
public static byte[][] 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}
    };
```

以上就是DES加密的全部内容了，再来重新理一下思路：

- 工具准备——子密钥生成。
- 明文经过初始置换。
- 置换的结果先分割成左右两部分，在进行十六轮迭代，最终合并得到结果，这一步的主要目的是：混淆和扩散，让信息扩散开来。
- 最后经过初始逆置换得到最终的密文。

DES解密

DES解密过程和加密过程完全一样，只不过在子密钥的使用上稍微有点不同，加密的时候子密钥是从第一把使用到第十六把，而解密是从第十六把用到第一把，完全相反的步骤，除此之外其余步骤一模一样，步骤如下：

- 密文经过初始置换。
- 置换的结果先分割成左右两部分，在进行十六轮迭代，密钥使用顺序相反。
- 最后经过初始逆置换得到最初的明文。

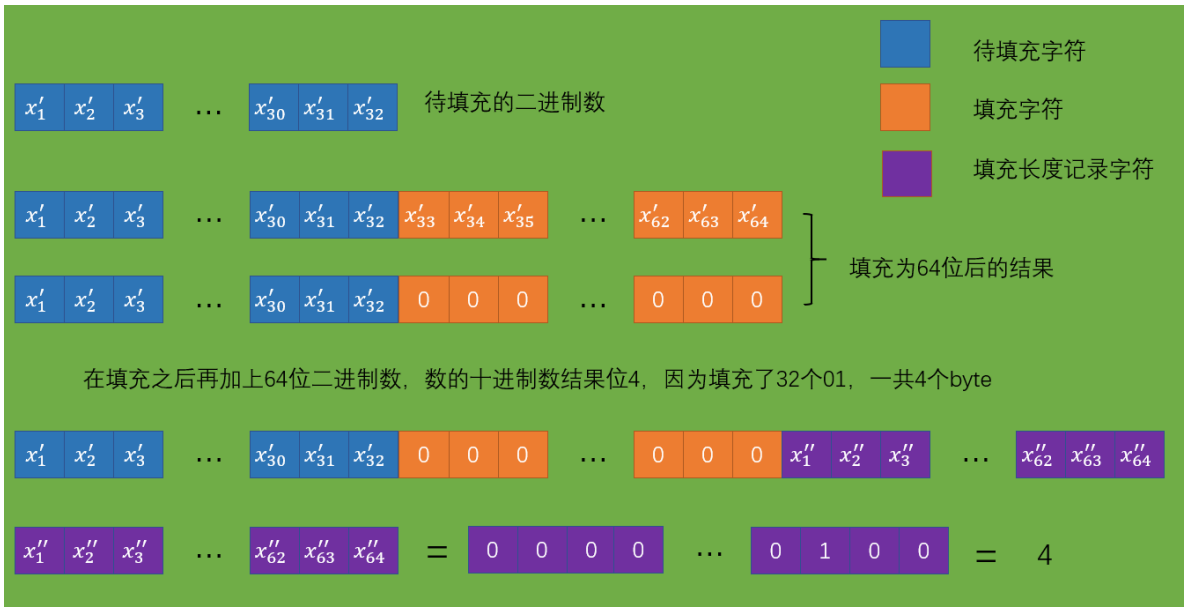
填充模式

前文我们已经知道在DES加密过程中，使用的密钥和明文都是64位的，也就是8个byte但是如果我们需要加密的信息不满足8byte的整数倍怎么办呢？

那就进行填充，如果待加密的信息大小是Mbytes，填充k个byte， 这k个byte都填充为0(即二进制为00000000)，这样我们在得到解密的时候如果后面的byte结果为0则丢掉它。

$$k = \begin{cases} M\%8 = m \\ 0 & if\ m == 0; \\ 8 - m & if\ m\ != 0; \end{cases}$$

除了填充之外，还需要知道一共填充了多少位，因为可能原文中最后一个byte的结果可能是0，即和我们填充的一样，如果不记录填充长度的话，会误将它归结为填充字段，解密结果就会缺失一部分数据，为此我们可以再增加64位用来记录填充的byte数目，即将k变成64位二进制数加载填充数据后面，举个例子，如果待加密的信息最后一组(每组64位)的长度位32位，则填充情况如下：

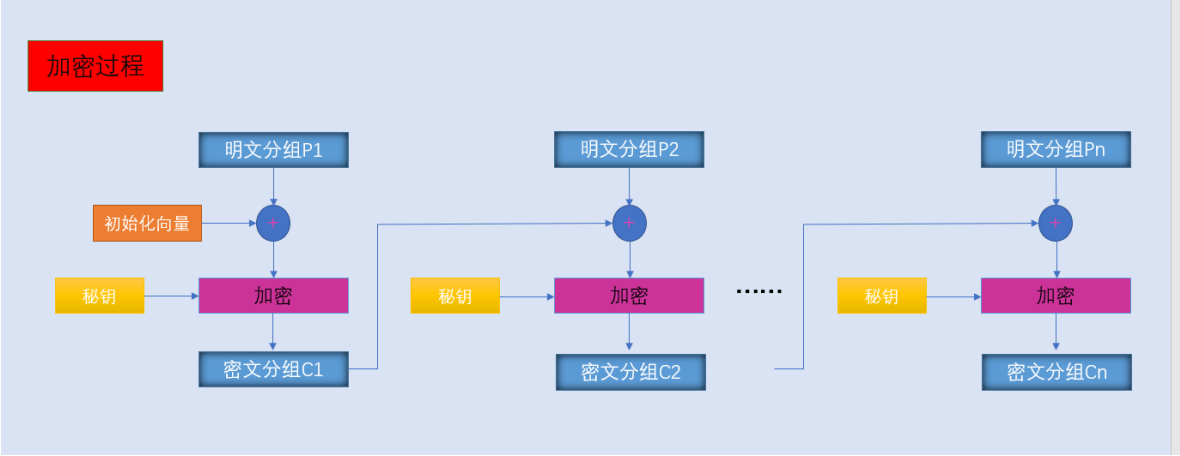


这样我们在解密完成之后读取最后一组信息(即最后64位)的到填充长度k，则原文信息就是去掉最后64位之后再从后往前去掉k个byte就能得到原文了。

CBC工作模式

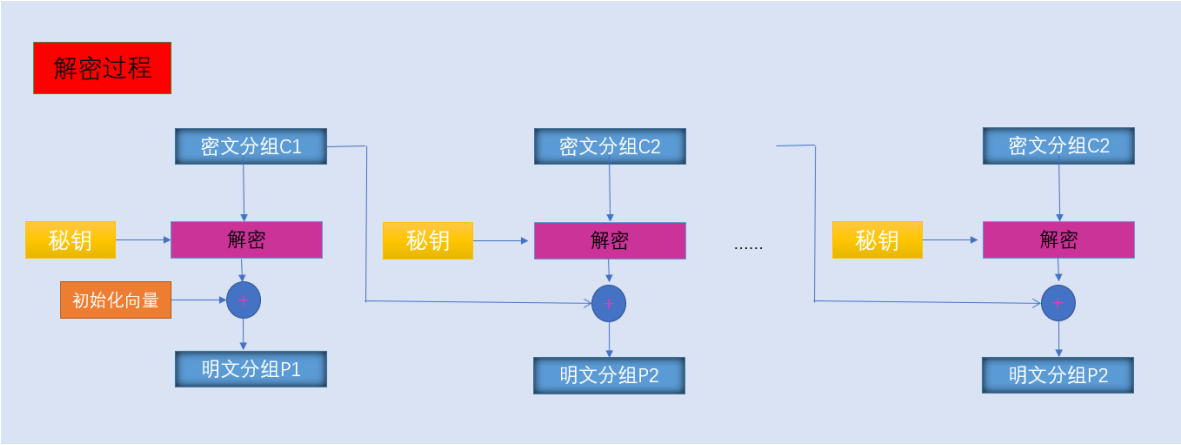
通常情况下我们需要加密的信息可能是一个文件，也可能是其他比较大的信息。当消息的长度大于分组的长度(64)时，需要分成几个分组分别进行处理，于是就有了分组密码的工作模式，接下来主要介绍其中的一种工作模式CBC加密模式

CBC加密模式主要流程如下图所示：



在CBC模式中，首先需要进行分组，将待加密的消息分成N组，如果最后一组没有满足64位则进行填充，除此之外，在CBC模式中还有一个初始化向量，它是一个64位的二进制数，它首先跟明文分组P1进行异或运算，然后使用DES算法对异或得到的结果进行加密得到密文分组C1，然后密文分组C1和明文分组P2进行异或运算，得到的结果再进行DES加密得到密文分组C2，然后密文分组C2再和明文分组P3进行异或运算.....，最终得到N个密文分组。

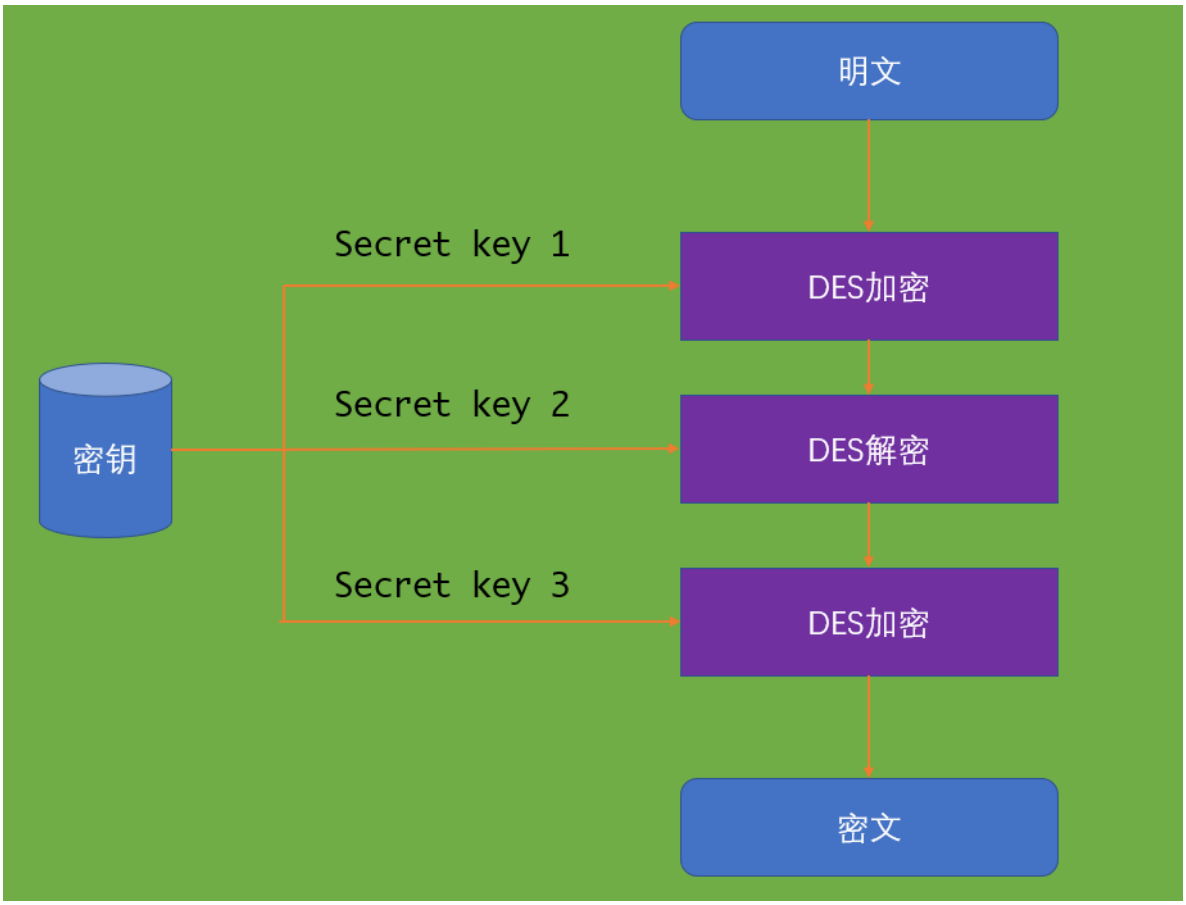
CBC解密模式主要流程如下图所示：



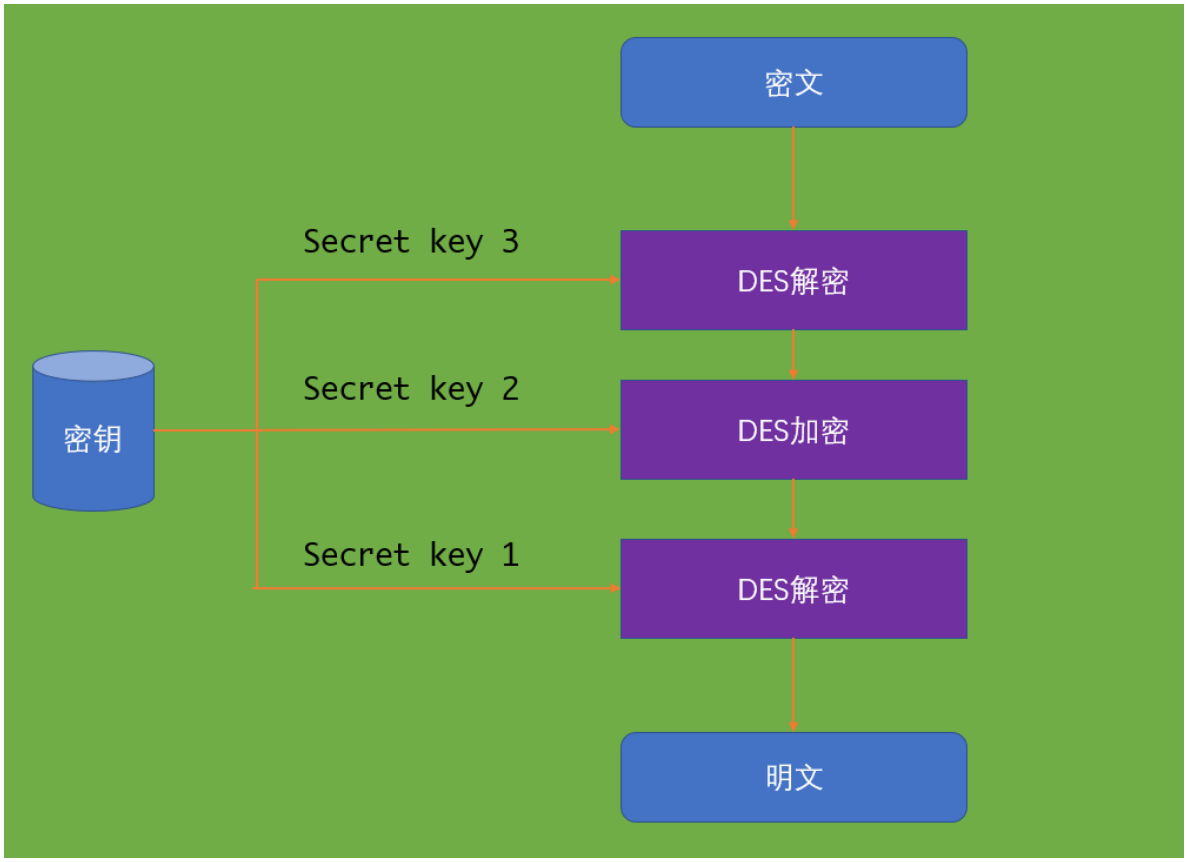
CBC模式解密就时加密的逆过程根据上图很容易理解，这里不再进行赘述。

Triple DES

从上面可以知道DES加密的密钥长度为64，实际上这个长度还不够，已经能被暴力破解，为了增加密钥的长度于是就诞生了3DES， 在了解DES之后，Triple DES就非常简单了，它的具体工作流程如下图所示：



上述就是3DES加密过程，解密将其反过来就行，即：



以上就是DES算法的全部内容~~~~