目录

[第一章 需求分析 1](#_Toc92628975)

[1.1 研究背景及意义 1](#_Toc92628976)

[1.2 国内外相关研究现状及发展趋势 1](#_Toc92628977)

[1.3 全文内容安排 2](#_Toc92628978)

[第二章 DES算法原理 3](#_Toc92628979)

[2.1 DES简介 3](#_Toc92628980)

[2.2 DES的加密解密 3](#_Toc92628981)

[2.3 DES工作原理 3](#_Toc92628982)

[2.4 DES解密原理 4](#_Toc92628983)

[2.5 三重DES 5](#_Toc92628984)

[第三章 基于3DES的图片加密程序具体实现 6](#_Toc92628985)

[3.1 图片数据的处理及分组处理 6](#_Toc92628986)

[3.1.1 读入图片的二进制流 6](#_Toc92628987)

[3.1.2 字符串转二进制bit 6](#_Toc92628988)

[3.1.3 des填充及分组 7](#_Toc92628989)

[3.2 组内加密 8](#_Toc92628990)

[3.2.1 密钥拆分 8](#_Toc92628991)

[3.2.2 DES初始化 10](#_Toc92628992)

[3.2.3 密钥置换 10](#_Toc92628993)

[3.2.4 ip盒置换 12](#_Toc92628994)

[3.2.5 E扩展置换 14](#_Toc92628995)

[3.2.6 S盒代替 14](#_Toc92628996)

[3.2.7 P盒运算 15](#_Toc92628997)

[3.2.8 IP逆置换 16](#_Toc92628998)

[3.3 CBC分组运行模式 16](#_Toc92628999)

[3.3.1 数据的预处理 17](#_Toc92629000)

[3.3.2 CBC代码 18](#_Toc92629001)

[3.4 密文图像显示 19](#_Toc92629002)

[3.5 密文图像解密 21](#_Toc92629003)

[3.5.1 CBC分组组合阶段 21](#_Toc92629004)

[3.5.2 子密钥运算阶段 21](#_Toc92629005)

[3.6 主函数编写 22](#_Toc92629006)

[第四章 问题处理 25](#_Toc92629007)

[第五章 实验数据 26](#_Toc92629008)

[第六章 总结 28](#_Toc92629009)

[第七章 完整程序 29](#_Toc92629010)

[第八章 参考文献 48](#_Toc92629011)

1. 绪论

本章是全篇的绪论部分，首先描述了DES算法的整体研究背景并论述了加密技术的重大意义;其次在阅览大量中英文文献后针对当前国内外对于DES应用的研究现状做了综述;最后概括性地总结了本文的主要研究工作以及全篇的结构安排。

* 1. 研究背景及意义

随着信息与通信技术的迅猛发展和广泛应用,网络与信息系统基础性、全局性的作用日益增强, 人们通过互联网进行信息交流，难免涉及到密码保护问题，这不仅关系到个人隐私甚至可能是国家机密，国家安全。隐私与机密网络信息的安全与保护问题日益成为亟待解决的关键问题。从这个意义上来说，提出一种对密码的有效加密算法，就是我们必须面对的问题，但是密码的复杂性和无序性给加密算法的设计带来难题，于是，我们必须寻找一种对简单密码普遍适用的规律性算法，那么，对对称密码的加密就成为最实用最普遍的问题。随着中国经济，社会的发展与崛起，生活的便捷性与快捷性越来越深入人心，网上购物，网上交易会越来越频繁，藉此，密码安全就成为我们最关注的问题。对密码的保护就成为我们的共识与需求，而加密最能体现出它的分量，那对对称

密码的加密同样就有深远的研究价值与现实意义。

不仅国内，国外对数据加密同样重视，1977年，美国数据加密标准（Data Encryption Standard, DES)颁布，在此后的30年，DES成为世界范围内的标准，，相关算法的不断改进，和新方法，新思路的不断涌现，这些，对对称密码加密算法的深入研究都起到很大的作用。

* 1. 国内外相关研究现状及发展趋势

国外目前不仅在密码基础理论方面的研究做的很好，而且在实际应用方面也做的非常好。制定了一系列特别规范的密码标准。尽管算法的征集和讨论都已经公开化，但密码技术作为一种关键技术，各国都不会放弃自主权和控制权，争夺霸权地位。

目前国际上对非数学的密码理论与技术非常关注，讨论也非常活跃。信息隐藏将在未来网络中保护信息免于破坏起到重要作用，信息隐藏是网络环境下把机密信息隐藏在大量信息中不让对方发觉的一种方法。特别是图象叠加、数字水印、潜信道、隐匿协议等的理论与技术的研究已经引起人们的重视。1996年以来，国际上召开了多次有关信息隐藏的专业研讨会。基于生物特征的识别理论与技术已有所发展，形成了一些理论和技术，也形成了一些产品。

* 1. 全文内容安排

本文由六个章节构成，全文安排如下:

第一章，绪论。主要通过一些数据，强调研究背景及意义的重要性， 介绍国内外的关键技术研究现状。在互联网时代，加密算法对于维护国家安全、维护社会和平发展有着举足轻重的作用。

第二章，DES算法原理。首先是对本文所使用的DES技术进行介绍，然后介绍了本文中所涉及的工作原理和3DES算法。

第三章，基于3DES的图片加密程序具体实现。首先介绍了图片数据的处理及分组实现，然后针对组内加密和CBC分组运行模式，详细地举例并阐释了程序的运行原理，最后通过密文图像显示和解密验证了方法的可行性，给出了最终的实验结果。

第四章，列出在程序编写过程中所遇到的问题并给出自己的解决思路。

第五章，实验数据，给出程序最终运行时所产生的加解密文件和运行效果图。

第六章，总结。对本文所做的研究工作做了概括总结。

第七章，给出整个程序的代码实现。

第八章，给出程序撰写中所引用的参考文献。

1. DES算法原理

在第一章绪论中介绍了本文的研究背景、选题意义，了解到随着互联网的飞速发展，加密算法在扮演着越来越举足轻重的作用，故逐渐成为政府机构和研究学者们关注的热点问题，本章对DES算法的基本原理做深入探索和研究。

* 1. DES简介

DES全称叫(Data Encryption Standard), 是1977年美国联邦信息处理标准（FIPS）中所采用的一种对称加密算法。DES之前一直使用很普遍，但是随着计算机的进步，现在DES已经可以被暴力破解了，除研究的原因外，我们不再建议使用DES算法。

* 1. DES的加密解密

DES的密钥长度是64比特，也就是8个字节。但是DES每隔7 比特会设置一个错误校验位，所以真正的密钥长度是56比特。

因为DES的长度是64比特，每次只能将长度为64比特的原文进行加密，如果原文长于64比特的话，需要多次分组进行加密，这64比特的单位也叫做分组。

* 1. DES工作原理

DES由Horst Feistel设计，也称作Feistel网络。Feistel网络，加密的各个步骤称为轮（round)，整个加密过程就是进行若干轮的循环。

下图展示了DES中的一轮加密过程：

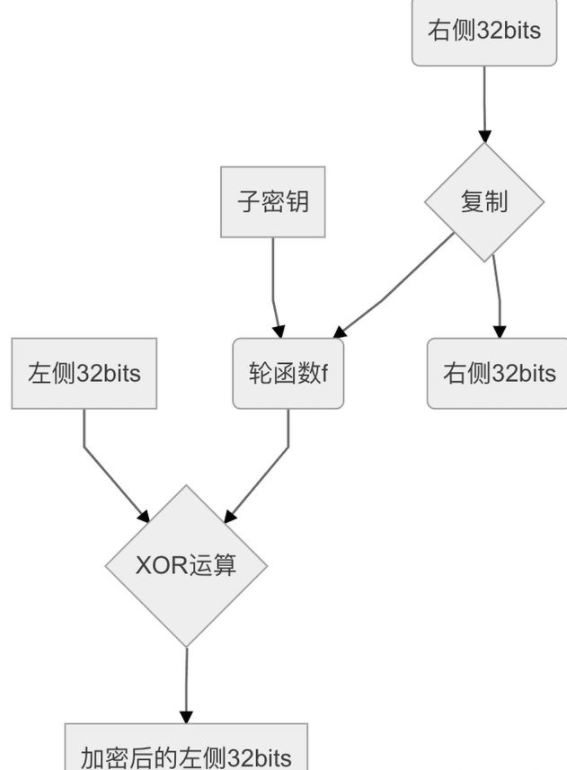


图2-1 DES加密过程

在每一轮的操作中，输入的数据被分成左右两部分，中间的子密钥指本轮加密所使用的密钥。在Feistel网络中，每一轮都使用不同的密钥，该密钥只是一个局部密钥，所以被称为子密钥。

其运算步骤如下：

1. 将输入的数据分成左右两部分
2. 将输入的右边直接发送到输出的右边
3. 将输入的右边发送到轮函数f
4. 轮函数根据右侧数据和子密钥计算出一个随机数
5. 将4得到的随机数和左侧数据进行XOR运算，将运算后的结果作为加密的左侧。

这一轮操作，只加密了左侧数据，右侧的没有加密，交换左右两侧的数据，再来一轮加密。这样右侧的数据也被加密。

* 1. DES解密原理

XOR的特性，即： A XOR B = C C XOR B = A

按照这样的特性，我们只需要将加密后的结果跟步骤4中计算出来的随机数再做一次XOR运算即可还原加密前的数据。

如下图所示：

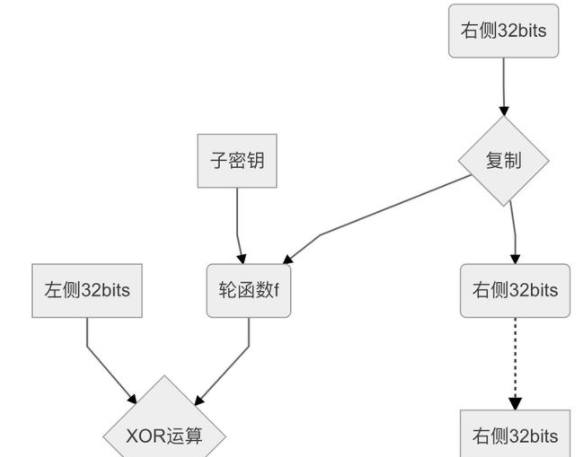


图2-2 DES解密过程

* 1. 三重DES

三重DES是为了增加DES的强度，将DES重复3次所得到的一种密码算法，通常缩写为3DES。

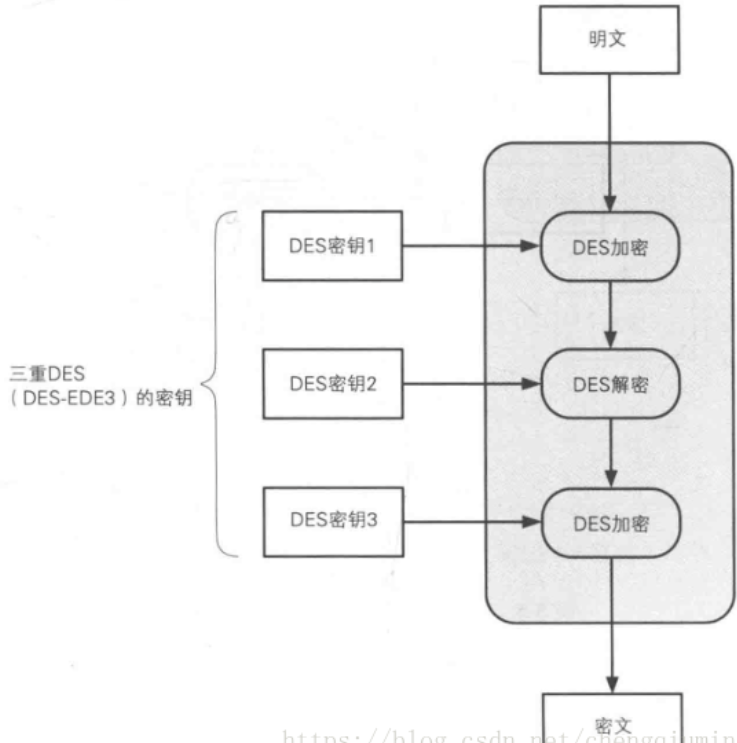


图2-3 3DES示意图

1. 基于3DES的图片加密程序具体实现
   1. 图片数据的处理及分组处理
      1. 读入图片的二进制流

使用rb方式读入二进制byte流

file = open(cwd,'rb')

* + 1. 字符串转二进制bit

该算法逐字读入byte将其转换为二进制形式bit

通过类似8进制转2进制的方法将以字节为单位占据8位的字节流转换为以比特为单位的占据1位的01比特

def \_\_String\_to\_BitList(*self*, *data*):

        """Turn the string data, into a list of bits (1, 0)'s"""

        if \_pythonMajorVersion < 3:

            # Turn the strings into integers. Python 3 uses a bytes

            # class, which already has this behaviour.

            data = [ord(c) for c in data]

        l = len(data) \* 8

        result = [0] \* l

        pos = 0

        for ch in data:

            i = 7

            while i >= 0:

                if ch & (1 << i) != 0:

                    result[pos] = 1

                else:

                    result[pos] = 0

                pos += 1

                i -= 1

* + 1. des填充及分组

填充：

des算法使用64位为单位的数据流，读入的图片数据要先经过数据的填充和切分使之符合要求。

if len(data) % **self**.block\_size != 0:

            if crypt\_type == des.DECRYPT: *# Decryption must work*

on 8 byte blocks

                raise ValueError("Invalid data length, data must be a multipl e of " + str(**self**.block\_size) + " bytes\n.")

            if not **self**.getPadding():

                raise ValueError("Invalid data length, data must be a multipl e of " + str(**self**.block\_size) + " bytes\n. Try setting

the optional padding character")

            else:

                data += (**self**.block\_size –

(len(data) % **self**.block\_size)) \* **self**.getPadding()

            # print "Len of data: %f" % (len(data) / self.block\_size)

该部分代码检查传入数据是否符合规则，若解密模式传入错误长度密文则报错

否则为其填充*len(data) / self.block\_size*个pad字符，默认为空，此为第一次数据的补位。

CBC模式下的补位不能直接补空或0，需要一个初始的填充值，该值需要我们自己手动给出并且01比特化

if **self**.getMode() == CBC:

            if **self**.getIV():

                iv = **self**.\_\_String\_to\_BitList(**self**.getIV())

            else:

                raise ValueError("For CBC mode, you must supply the Initial V alue (IV) for ciphering")

分组：

该段程序将数据流拆成8字节即64位，每段数据分别01比特化后检验，

若是CBC模式，将源数据按位与给出的初始补充数据异或实现将上下文连接起来增强安全性的CBC功能，之后将预处理好的64位数据送往DES加密。

while i < len(data):

            block = **self**.\_\_String\_to\_BitList(data[i:i+8])

            if **self**.getMode() == CBC:

                if crypt\_type == des.ENCRYPT:

                    block = list(map(lambda *x*, *y*: x ^ y, block, iv))

                processed\_block = **self**.\_\_des\_crypt(block, crypt\_type)

                if crypt\_type == des.DECRYPT:

                    processed\_block = list(map(lambda *x*, *y*: x ^ y,

processed\_block, iv))

                    iv = block

                else:

                    iv = processed\_block

            else:

                processed\_block = **self**.\_\_des\_crypt(block, crypt\_type)

* 1. 组内加密

本实验使用3des加解密方式，该算法与des不同之处在于，需要给出3个8字节密钥或者一次给出24字节密钥，通过加密-解密-加密三次不同密钥的普通des加密获得比普通加密更高的安全性。

* + 1. 密钥拆分

该段代码实现：检查密钥长度并在不符合要求的情况下弹出提示，将24字节密钥拆分为三段密钥，最后连续进行三次des算法。特别指出的是，本算法同样接收16字节长度的密钥，在这种情况下，3des算法退化为des算法。

当模式为CBC且没有指定初始补充数据时，初始补充数据为密钥的前8字节

def setKey(*self*, *key*):

**self**.key\_size = 24  *# Use DES-EDE3 mode*

        if len(key) != **self**.key\_size:

            if len(key) == 16: *# Use DES-EDE2 mode*

**self**.key\_size = 16

            else:

                raise ValueError("Invalid triple DES key size. Key must be ei ther 16 or 24 bytes long")

        if **self**.getMode() == CBC:

            if not **self**.getIV():

                # Use the first 8 bytes of the key

**self**.\_iv = key[:**self**.block\_size]

            if len(**self**.getIV()) != **self**.block\_size:

                raise ValueError("Invalid IV, must be 8 bytes in length")

**self**.\_\_key1 = des(key[:8], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

**self**.\_\_key2 = des(key[8:16], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

        if **self**.key\_size == 16:

**self**.\_\_key3 = **self**.\_\_key1

        else:

**self**.\_\_key3 = des(key[16:], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

        \_baseDes.setKey(**self**, key)

* + 1. DES初始化

该段代码传入一些关键信息包括密钥，初始补充数据，填充模式等同时声明了一些运算所必须的容器包括左右数组，16个子密钥等。

def \_\_init\_\_(*self*, *key*, *mode*=ECB, *IV*=None, *pad*=None, *padmode*=PAD\_NORMAL):

        # Sanity checking of arguments.

        if len(key) != 8:

            raise ValueError("Invalid DES key size. Key must be exactly 8 bytes long.")

        \_baseDes.\_\_init\_\_(**self**, mode, IV, pad, padmode)

**self**.key\_size = 8

**self**.L = []

**self**.R = []

**self**.Kn = [ [0] \* 48 ] \* 16 *# 16 48-bit keys (K1 - K16)*

**self**.final = []

**self**.setKey(key)

* + 1. 密钥置换

映射函数，定义了一个建立子密钥到某个列表的映射

def \_\_permutate(*self*, *table*, *block*):

        """Permutate this block with the specified table"""

        return list(map(lambda *x*: block[x], table))

该段代码调用映射函数，实现了密钥的pc1变换，该步骤后密钥变为56位

key = **self**.\_\_permutate(des.\_\_pc1, **self**.\_\_String\_to\_BitList(**self**.getKey()))

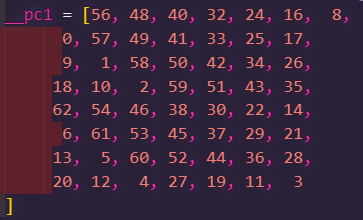


图3-1 pc1表

将56位密钥分为左边和右边两部分，对该28位，进行按照左移表的重复左移。

**self**.L = key[:28]

**self**.R = key[28:]

        while i < 16:

            j = 0

            # Perform circular left shifts

            while j < des.\_\_left\_rotations[i]:

**self**.L.append(**self**.L[0])

                del **self**.L[0]

**self**.R.append(**self**.R[0])

                del **self**.R[0]

                j += 1

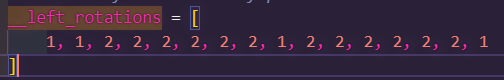


图3-2 左移表

类似pc1变换地，调用映射函数并进行pc2变换

**self**.Kn[i] = **self**.\_\_permutate(des.\_\_pc2, **self**.L + **self**.R)

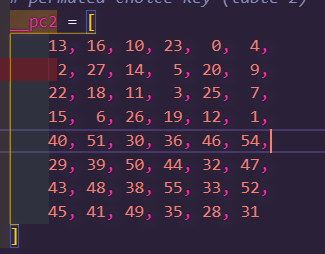


图3-3 pc2表

通过上述操作，我们就得到了16组48位的子密钥

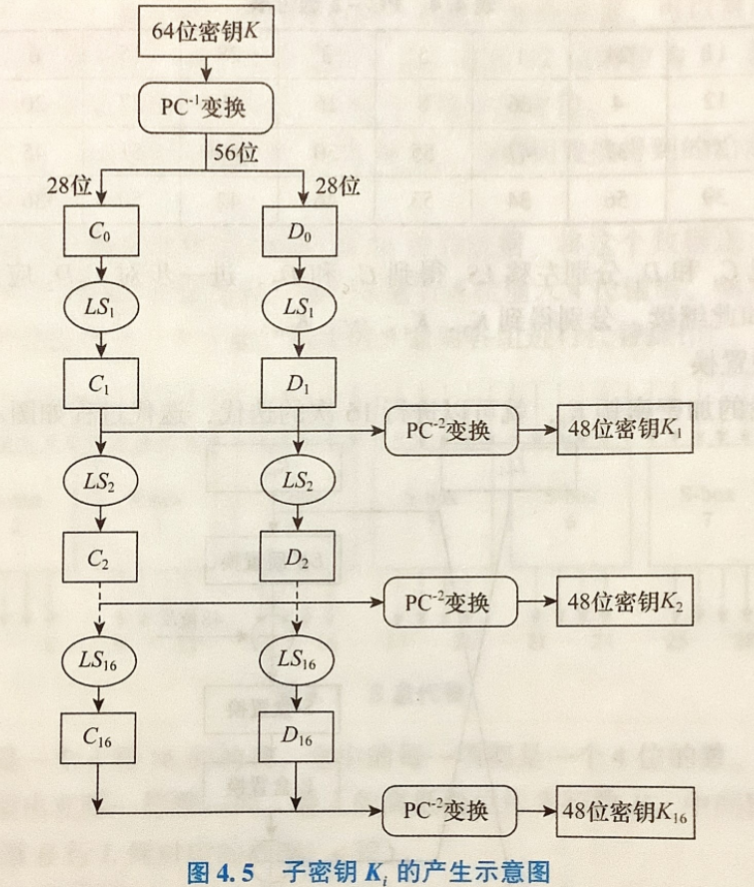


图3-4 子密钥产生图

* + 1. ip盒置换

IP置换的目的是将转化后的64位比特数据按位重新组合，并把输出分为L0和R0两部分，各部分长32位。  
 例如ip盒中的第一个数是57，那么就是把原数据中的第57位放到新数据的第1位

block = **self**.\_\_permutate(des.\_\_ip, block)

**self**.L = block[:32]

**self**.R = block[32:]

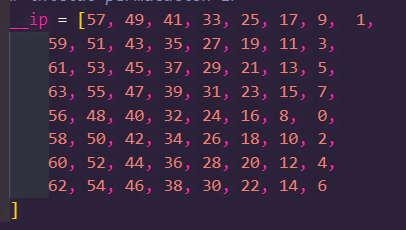


图3-5 IP盒

轮函数f迭代

有了每一轮的加密密钥Ki，就可以进行16次迭代

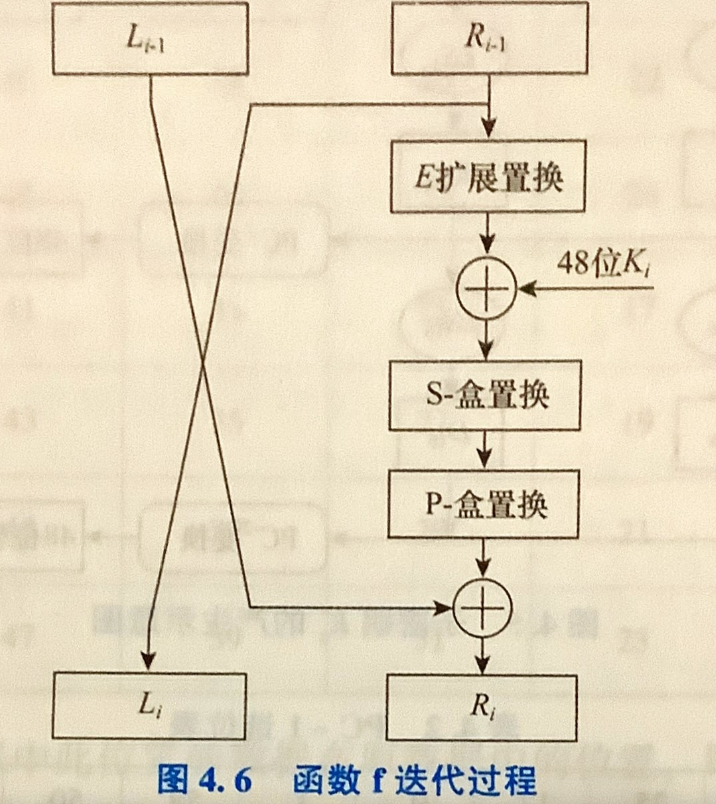


图2-6 f函数迭代过程

根据加密或是解密确定运算方向

if crypt\_type == des.ENCRYPT:

            iteration = 0

            iteration\_adjustment = 1

        # Decryption starts from Kn[16] down to Kn[1]

        else:

            iteration = 15

            iteration\_adjustment = -1

* + 1. E扩展置换

实现明文拓展的功能，使用E盒将明文的部分bit重复排列，造成冗余，来实现拓展。E扩展置换的目的有两个：

* 生成与密钥相同长度的数据以进行异或运算
* 提供更长的结果，再后续的替代运算中可以进行压缩。

**self**.R = **self**.\_\_permutate(des.\_\_expansion\_table, **self**.R)

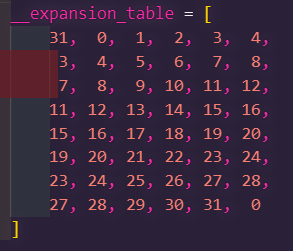


图3-7 E扩展盒

子密钥异或，将结果存储到创建的容器B中

**self**.R = list(map(lambda *x*, *y*: x ^ y, **self**.R, **self**.Kn[iteration]))

            B = [**self**.R[:6], **self**.R[6:12], **self**.R[12:18], **self**.R[18:24], **self**.R[24:30], **self**.R[30:36], **self**.R[36:42], **self**.R[42:]]

* + 1. S盒代替

异或运算结束后需要将48bit缩减为32bit，分组使用S盒置换。替代由8个不同的S盒完成，每个S盒有6个输入4个输出。48位输入分为8个6位分组，每个分组对应一个S盒，对应的S盒对各组进行代替操作。替代过程中产生的8个4位的分组，组合在一起形成32位数据

  j = 0

            Bn = [0] \* 32

            pos = 0

            while j < 8:

                m = (B[j][0] << 1) + B[j][5]

                n = (B[j][1] << 3) + (B[j][2] << 2) + (B[j][3] << 1) + B[j][4

                v = des.\_\_sbox[j][(m << 4) + n]

                 Bn[pos] = (v & 8) >> 3

                Bn[pos + 1] = (v & 4) >> 2

                Bn[pos + 2] = (v & 2) >> 1

                Bn[pos + 3] = v & 1

                pos += 4

                j += 1

例如S盒8的输入为110011，第1位和第6位组合为11，对应S盒8的第3行，第2位到第5位为1001，对应于S盒8的地9列。S盒的第3行第9列的数字为12，那么就用1100来代替110011。

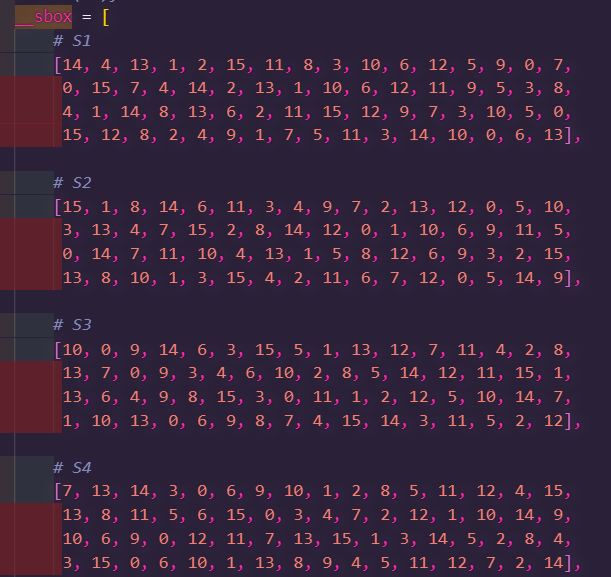


图3-8 S盒

* + 1. P盒运算

将B[1]到B[8]的串联进行排列组合

**self**.R = **self**.\_\_permutate(des.\_\_p, Bn)

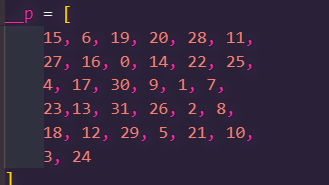


图3-9 P盒

明文右侧与子密钥进行轮函数f后的结果与明文左边的数据进行xor运算，然后交换左边和右边

**self**.R = list(map(lambda *x*, *y*: x ^ y, **self**.R, **self**.L))

**self**.L = tempR

至此 我们完成了一轮的Feistel变换，如此变换16轮，特殊地，在进行最后一轮Feistel变换的时候不需要交换左边和右边

* + 1. IP逆置换

得到了加密后的56位比特后用初始置换函数的逆函数，得到正确的密文序列。

**self**.final = **self**.\_\_permutate(des.\_\_fp, **self**.R + **self**.L)

        return **self**.final

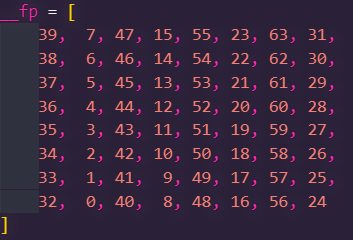


图3-10 fp盒

至此，一次DES组内加密结束。

* 1. CBC分组运行模式

CBC即Cipher Block Chaining ，密码分组链接模式，这种模式是先将明文切分成若干小段，然后每一小段与初始块(IV)或者上一段的密文段进行异或，再与密钥进行加密。

* + 1. 数据的预处理

该段代码分别生成三个对应3des的3次加解密部分的对象，该对象将被初始化函数赋予包括密钥，加密模式（CBC），IV初始块，填充字符，填充模式（PKCS5）的信息。

**self**.\_\_key1 = des(key[:8], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

**self**.\_\_key2 = des(key[8:16], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

        if **self**.key\_size == 16:

**self**.\_\_key3 = **self**.\_\_key1

该部分代码为初始化函数，赋予对象信息

def \_\_init\_\_(*self*, *mode*=ECB, *IV*=None, *pad*=None, *padmode*=PAD\_NORMAL):

        if IV:

            IV = **self**.\_guardAgainstUnicode(IV)

        if pad:

            pad = **self**.\_guardAgainstUnicode(pad)

**self**.block\_size = 8

        # Sanity checking of arguments.

        if pad and padmode == PAD\_PKCS5:

            raise ValueError("Cannot use a pad character with PAD\_PKCS5")

        if IV and len(IV) != **self**.block\_size:

            raise ValueError("Invalid Initial Value (IV), must be a multiple of " + str(**self**.block\_size) + " bytes")

        # Set the passed in variables

**self**.\_mode = mode

**self**.\_iv = IV

**self**.\_padding = pad

**self**.\_padmode = padmode

该部分代码为校验函数，检测参数是否符合要求

    def \_guardAgainstUnicode(*self*, *data*):

        if \_pythonMajorVersion < 3:

            if isinstance(data, unicode):

                raise ValueError("can only work with bytes, not Unicode strings.")

        else:

            if isinstance(data, str):

                try:

                    return data.encode('ascii')

                except UnicodeEncodeError:

                    pass

                raise ValueError("can only work with encoded strings, not Unicode.")

        return data

* + 1. CBC代码

CBC的每第一小段需要初始值来异或，该段代码确保对象拥有初始值

**self**.\_\_key1.setIV(**self**.getIV())

**self**.\_\_key2.setIV(**self**.getIV())

**self**.\_\_key3.setIV(**self**.getIV())

该段代码搭建一个单元为8位比特的循环，一个循环中对该8比特数据连续进行三次des操作实现3des，同时每次循环结束后重新对IV进行赋值，确保下一次操作以当前数据为参数异或，实现CBC模式。

i = 0

            result = []

            while i < len(data):

                iv = data[i:i+8]

                block = **self**.\_\_key3.crypt(iv,    DECRYPT)

                block = **self**.\_\_key2.crypt(block, ENCRYPT)

                block = **self**.\_\_key1.crypt(block, DECRYPT)

**self**.\_\_key1.setIV(iv)

**self**.\_\_key2.setIV(iv)

**self**.\_\_key3.setIV(iv)

                result.append(block)

                i += 8

该段代码实现数据与参数异或的功能

if **self**.getMode() == CBC:

                if crypt\_type == des.ENCRYPT:

                    block = list(map(lambda *x*, *y*: x ^ y, block, iv))

* 1. 密文图像显示

由于图片数据在经过3des加密之后原有的文件头尾和数据结构被打乱，无法使用正常的方式读取，我们需要重新开发一个程序将乱码的图片展示。

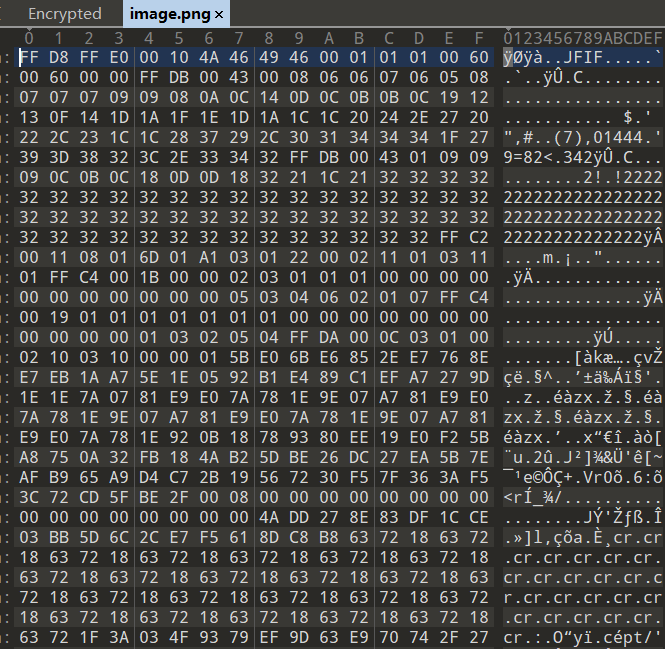


图3-11 格式完好的图片

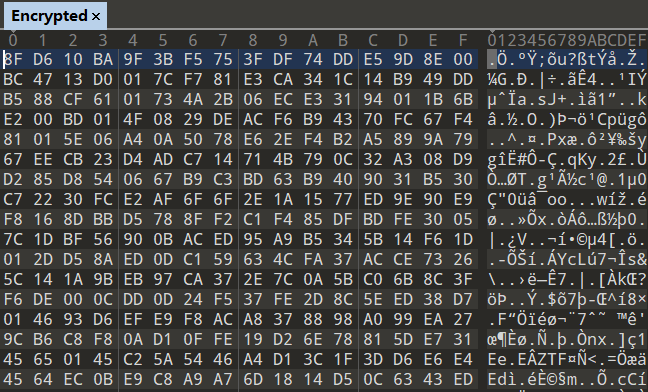


图3-12 加密后的图片结构，可以看出失去了文件头

该段代码实现了将二进制数据转换为图片的功能

首先获取二进制数据的长度将其换算为将要生成的图片的长和宽，之后通过一个二重循环遍历全部数据，向图片中插入算法计算出的各色像素点，从而实现功能。

def bin2img(*data*, *isgrey*):

    colorfunc = calcgrayshade if isgrey else calccolor

    xsize, ysize = size = determine\_size(data)

    img = Image.new("RGB", size, *color*=(255, 255, 255, 0))

    draw = ImageDraw.Draw(img)

    try:

        i = 0

        for y in range(ysize):

            for x in range(xsize):

                draw.point((x, y), *fill*=colorfunc(data[i]))

                i += 1

    except IndexError:

        pass



图3-13 算法生成的图片

* 1. 密文图像解密

3des算法的解密基本上是加密的逆运算

* + 1. CBC分组组合阶段

从原本的EN——DE——EN变为DE——EN——DE

iv = data[i:i+8]

                block = **self**.\_\_key3.crypt(iv,    DECRYPT)

                block = **self**.\_\_key2.crypt(block, ENCRYPT)

                block = **self**.\_\_key1.crypt(block, DECRYPT)

* + 1. 子密钥运算阶段

从加密的iteration0-15变为iteration15-0，相当于p盒，f轮函数，S盒，E扩展盒等都是反向运算的。

if crypt\_type == des.ENCRYPT:

            iteration = 0

            iteration\_adjustment = 1

        else:

            iteration = 15

            iteration\_adjustment = -1

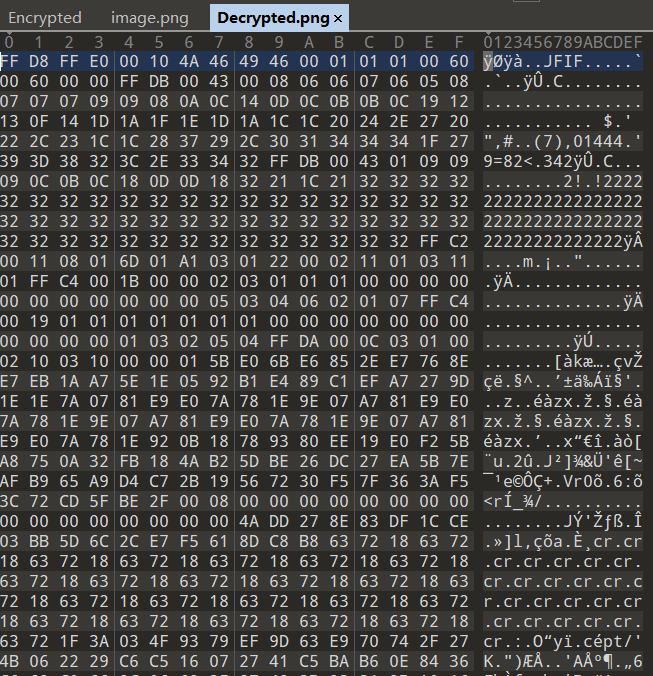


图3-14 解密后的图片数据，与加密前一致

* 1. 主函数编写

将3des算法和二进制转图片算法整合到一个主函数中并为其包装位=为图形处理界面。

该段代码创建图形界面幕布并给出初始值和密钥

root = Tk()

root.geometry("400x400")

root.title("刘世卓8003119050 图像加解密")

ivInt = bytearray("AStringOfIV=", "utf-8")                                 *#key1*

keyInt = bytearray("liushizhuotailihailebahahahahaha", "utf-8")            *#key2*

该段代码引用编写的3des算法并包装成为函数同时给出命令行的提示，给出按钮在幕布的位置

def encrypt\_image():

    with open('image.png', 'rb') as f:

        jpgdata=f.read()

    k = triple\_des(base64.b64decode(keyInt), *mode*=CBC, *IV*=base64.b64decode(ivInt),*pad*=None, *padmode*=PAD\_PKCS5)    *#encryption*

    print ("please wait for a moment")

    jpgdata = k.encrypt(jpgdata)

    with open('Encrypted', 'wb') as f:

        f.write(jpgdata)

    print("Encryption Complete")

b = Button(root, *text*="Encrypt", *command*=encrypt\_image)

b.place(*x*=100, *y*=210)

def decrypt\_image():

    with open('Encrypted', 'rb') as f:

        jpgdata=f.read()

    k = triple\_des(base64.b64decode(keyInt), *mode*=CBC, *IV*=base64.b64decode(ivInt), *padmode*=PAD\_PKCS5)       *#decryption*

    print ("please wait for a moment")

    jpgdata = k.decrypt(jpgdata)

    with open('Decrypted.png', 'wb') as f:

        f.write(jpgdata)

    print("Decryption Complete")

b = Button(root, *text*="Decrypt", *command*=decrypt\_image)

b.place(*x*=200, *y*=210)

该段代码实现了两个查看图片的按钮和退出按钮，一个查看解密后的图片，一个调用二进制转图片算法，查看加密后的图片。

def check\_image():

    bin2img.it()

    img = Image.open('Encrypted.png')

    plt.imshow(img)

    plt.show()

b = Button(root, *text*="checkdecryimg", *command*=check\_image)

b.place(*x*=300, *y*=210)

def check\_image1():

    img = Image.open('image.png')

    plt.imshow(img)

    plt.show()

b = Button(root, *text*="checkimg", *command*=check\_image1)

b.place(*x*=300, *y*=110)

exit\_button = Button(root, *text*="Exit", *command*=root.quit)

exit\_button.place(*x*=310, *y*=310)

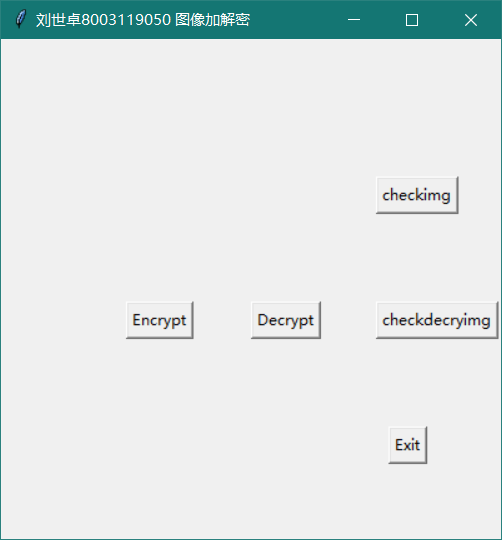


图3-15 实现的图形化界面

1. 问题处理

本次大作业实现了利用3des加解密图片功能的程序并为其添加了图形化操作界面，开发过程中遇到了许多问题与挑战，以下是解决它们的思路。

* des算法的精度问题

使用C++ 语言实现的第一版des算法可能某个函数的底层逻辑精度较差，总是在多次连续加解密的过程中丢失信息导致无法将信息还原，在多次代码审计无果后痛下决心使用python语言重构，终于获得了较好的加解密效果。

* 3des算法的构造问题

3des算法本身并不复杂，只是des以不同密钥对同一段目标数据运行三次罢了。但如果仅仅是对一整个目标数据使用三次des算法整个程序就显得十分臃肿而丧失了编程的优雅。最终选择了以64比特为一段运行程序，这样的代码在优化运行速度的同时给予CBC模式的编写以便利。

* CBC模式的实现问题

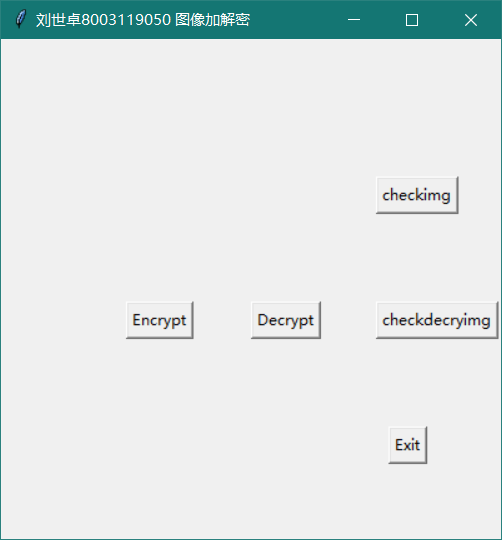
类似3des算法，解决思路主要是将数据拆成小块并标记，在每次运行主要加解密模块之前将其与上一次的数据依照标记所示进行异或达到目的。

* 加密图片的显示问题

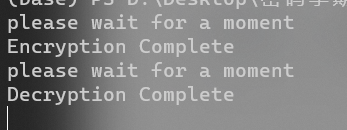
图片数据在进行加密后文件头尾和其数据结构被完全破坏，常规的图片读取手段无法查看图片，于是选择另外编写一个显示图片的算法。

1. 实验数据

图形化操作界面



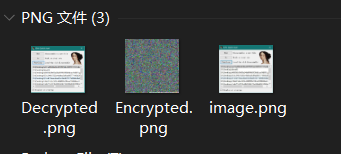
加解密提示



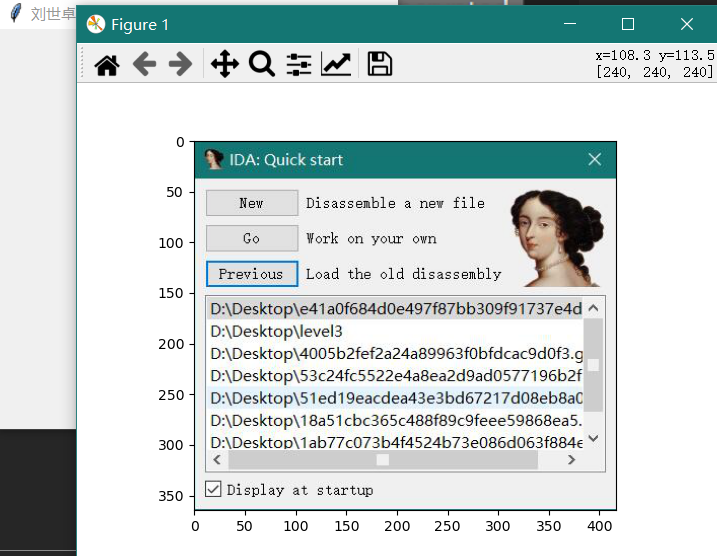
生成的加密文件

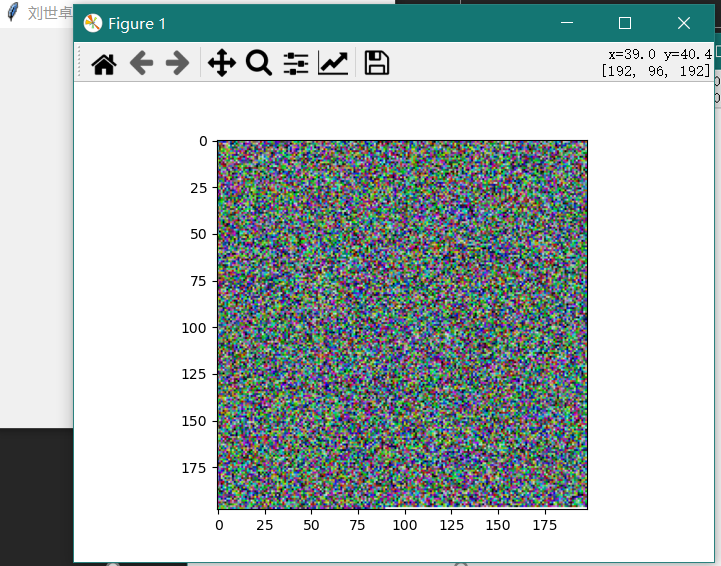


由一个image生成的加密中图片和解密后图片



查看图片功能



查看加密图片功能

1. 总结

在这次实验中,我全心全意地投入到3des加解密实现的乐趣和挑战当中。通过查阅资料，学习基础知识和原理架构，遇到问题及时解决，持续不断地尝试探索，我在其实现方面有了全新的认识和突破。

在此次实验前，我对3des的认识和了解还只是单纯地停留在老师课堂上所讲的理论基础，可以说是一知半解；但通过此次实验的完整过程，我更加深入地理解和体会了3des算法的工作原理，同时在实现程序过程中更加深入的理解了许多相关函数的底层逻辑，也更加熟练地掌握了程序编写所需要遵守的架构和体系，增强了我的编写能力。

1. 完整程序

import sys

import base64

import io

from io import BytesIO

import matplotlib.pyplot as plt

import pickle

from tkinter import \*

from tkinter import filedialog

from PIL import Image

import bin2img

\_pythonMajorVersion = sys.version\_info[0]

ECB =   0

CBC =   1

PAD\_NORMAL = 1

PAD\_PKCS5 = 2

class \_baseDes(object):

    def \_\_init\_\_(*self*, *mode*=ECB, *IV*=None, *pad*=None, *padmode*=PAD\_NORMAL):

        if IV:

            IV = **self**.\_guardAgainstUnicode(IV)

        if pad:

            pad = **self**.\_guardAgainstUnicode(pad)

**self**.block\_size = 8

        if pad and padmode == PAD\_PKCS5:

            raise ValueError("Cannot use a pad character with PAD\_PKCS5")

        if IV and len(IV) != **self**.block\_size:

            raise ValueError("Invalid Initial Value (IV), must be a multiple of " + str(**self**.block\_size) + " bytes")

**self**.\_mode = mode

**self**.\_iv = IV

**self**.\_padding = pad

**self**.\_padmode = padmode

    def getKey(*self*):

        return **self**.\_\_key

    def setKey(*self*, *key*):

        key = **self**.\_guardAgainstUnicode(key)

**self**.\_\_key = key

    def getMode(*self*):

        return **self**.\_mode

    def setMode(*self*, *mode*):

**self**.\_mode = mode

    def getPadding(*self*):

        return **self**.\_padding

    def setPadding(*self*, *pad*):

        if pad is not None:

            pad = **self**.\_guardAgainstUnicode(pad)

**self**.\_padding = pad

    def getPadMode(*self*):

        return **self**.\_padmode

    def setPadMode(*self*, *mode*):

**self**.\_padmode = mode

    def getIV(*self*):

        return **self**.\_iv

    def setIV(*self*, *IV*):

        if not IV or len(IV) != **self**.block\_size:

            raise ValueError("Invalid Initial Value (IV), must be a multiple of " + str(**self**.block\_size) + " bytes")

        IV = **self**.\_guardAgainstUnicode(IV)

**self**.\_iv = IV

    def \_padData(*self*, *data*, *pad*, *padmode*):

        if padmode is None:

            padmode = **self**.getPadMode()

        if pad and padmode == PAD\_PKCS5:

            raise ValueError("Cannot use a pad character with PAD\_PKCS5")

        if padmode == PAD\_NORMAL:

            if len(data) % **self**.block\_size == 0:

                return data

            if not pad:

                pad = **self**.getPadding()

            if not pad:

                raise ValueError("Data must be a multiple of " + str(**self**.block\_size) + " bytes in length. Use padmode=PAD\_PKCS5 or set the pad character.")

            data += (**self**.block\_size - (len(data) % **self**.block\_size)) \* pad

        elif padmode == PAD\_PKCS5:

            pad\_len = 8 - (len(data) % **self**.block\_size)

            if \_pythonMajorVersion < 3:

                data += pad\_len \* chr(pad\_len)

            else:

                data += bytes([pad\_len] \* pad\_len)

        return data

    def \_unpadData(*self*, *data*, *pad*, *padmode*):

        if not data:

            return data

        if pad and padmode == PAD\_PKCS5:

            raise ValueError("Cannot use a pad character with PAD\_PKCS5")

        if padmode is None:

            padmode = **self**.getPadMode()

        if padmode == PAD\_NORMAL:

            if not pad:

                pad = **self**.getPadding()

            if pad:

                data = data[:-**self**.block\_size] + \

                       data[-**self**.block\_size:].rstrip(pad)

        elif padmode == PAD\_PKCS5:

            if \_pythonMajorVersion < 3:

                pad\_len = ord(data[-1])

            else:

                pad\_len = data[-1]

            data = data[:-pad\_len]

        return data

    def \_guardAgainstUnicode(*self*, *data*):

        if \_pythonMajorVersion < 3:

            if isinstance(data, unicode):

                raise ValueError("can only work with bytes, not Unicode strings.")

        else:

            if isinstance(data, str):

                try:

                    return data.encode('ascii')

                except UnicodeEncodeError:

                    pass

                raise ValueError("can only work with encoded strings, not Unicode.")

        return data

class des(\_baseDes):

    \_\_pc1 = [56, 48, 40, 32, 24, 16,  8,

          0, 57, 49, 41, 33, 25, 17,

          9,  1, 58, 50, 42, 34, 26,

         18, 10,  2, 59, 51, 43, 35,

         62, 54, 46, 38, 30, 22, 14,

          6, 61, 53, 45, 37, 29, 21,

         13,  5, 60, 52, 44, 36, 28,

         20, 12,  4, 27, 19, 11,  3

    ]

    \_\_left\_rotations = [

        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

    ]

    \_\_pc2 = [

        13, 16, 10, 23,  0,  4,

         2, 27, 14,  5, 20,  9,

        22, 18, 11,  3, 25,  7,

        15,  6, 26, 19, 12,  1,

        40, 51, 30, 36, 46, 54,

        29, 39, 50, 44, 32, 47,

        43, 48, 38, 55, 33, 52,

        45, 41, 49, 35, 28, 31

    ]

    \_\_ip = [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,

        56, 48, 40, 32, 24, 16, 8,  0,

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

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

        62, 54, 46, 38, 30, 22, 14, 6

    ]

    \_\_expansion\_table = [

        31,  0,  1,  2,  3,  4,

         3,  4,  5,  6,  7,  8,

         7,  8,  9, 10, 11, 12,

        11, 12, 13, 14, 15, 16,

        15, 16, 17, 18, 19, 20,

        19, 20, 21, 22, 23, 24,

        23, 24, 25, 26, 27, 28,

        27, 28, 29, 30, 31,  0

    ]

    \_\_sbox = [

*# 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],

*# 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],

*# 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],

*# 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],

*# 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],

*# 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],

*# 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],

*# 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],

    ]

    \_\_p = [

        15, 6, 19, 20, 28, 11,

        27, 16, 0, 14, 22, 25,

        4, 17, 30, 9, 1, 7,

        23,13, 31, 26, 2, 8,

        18, 12, 29, 5, 21, 10,

        3, 24

    ]

    \_\_fp = [

        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,

        32,  0, 40,  8, 48, 16, 56, 24

    ]

    ENCRYPT =   0x00

    DECRYPT =   0x01

    def \_\_init\_\_(*self*, *key*, *mode*=ECB, *IV*=None, *pad*=None, *padmode*=PAD\_NORMAL):

        if len(key) != 8:

            raise ValueError("Invalid DES key size. Key must be exactly 8 bytes long.")

        \_baseDes.\_\_init\_\_(**self**, mode, IV, pad, padmode)

**self**.key\_size = 8

**self**.L = []

**self**.R = []

**self**.Kn = [ [0] \* 48 ] \* 16

**self**.final = []

**self**.setKey(key)

    def setKey(*self*, *key*):

        \_baseDes.setKey(**self**, key)

**self**.\_\_create\_sub\_keys()

    def \_\_String\_to\_BitList(*self*, *data*):

        if \_pythonMajorVersion < 3:

            data = [ord(c) for c in data]

        l = len(data) \* 8

        result = [0] \* l

        pos = 0

        for ch in data:

            i = 7

            while i >= 0:

                if ch & (1 << i) != 0:

                    result[pos] = 1

                else:

                    result[pos] = 0

                pos += 1

                i -= 1

        return result

    def \_\_BitList\_to\_String(*self*, *data*):

        result = []

        pos = 0

        c = 0

        while pos < len(data):

            c += data[pos] << (7 - (pos % 8))

            if (pos % 8) == 7:

                result.append(c)

                c = 0

            pos += 1

        if \_pythonMajorVersion < 3:

            return ''.join([ chr(c) for c in result ])

        else:

            return bytes(result)

    def \_\_permutate(*self*, *table*, *block*):

        return list(map(lambda *x*: block[x], table))

    def \_\_create\_sub\_keys(*self*):

        key = **self**.\_\_permutate(des.\_\_pc1, **self**.\_\_String\_to\_BitList(**self**.getKey()))

        i = 0

**self**.L = key[:28]

**self**.R = key[28:]

        while i < 16:

            j = 0

            while j < des.\_\_left\_rotations[i]:

**self**.L.append(**self**.L[0])

                del **self**.L[0]

**self**.R.append(**self**.R[0])

                del **self**.R[0]

                j += 1

**self**.Kn[i] = **self**.\_\_permutate(des.\_\_pc2, **self**.L + **self**.R)

            i += 1

    def \_\_des\_crypt(*self*, *block*, *crypt\_type*):

        block = **self**.\_\_permutate(des.\_\_ip, block)

**self**.L = block[:32]

**self**.R = block[32:]

        if crypt\_type == des.ENCRYPT:

            iteration = 0

            iteration\_adjustment = 1

        else:

            iteration = 15

            iteration\_adjustment = -1

        i = 0

        while i < 16:

            tempR = **self**.R[:]

**self**.R = **self**.\_\_permutate(des.\_\_expansion\_table, **self**.R)

**self**.R = list(map(lambda *x*, *y*: x ^ y, **self**.R, **self**.Kn[iteration]))

            B = [**self**.R[:6], **self**.R[6:12], **self**.R[12:18], **self**.R[18:24], **self**.R[24:30], **self**.R[30:36], **self**.R[36:42], **self**.R[42:]]

            j = 0

            Bn = [0] \* 32

            pos = 0

            while j < 8:

                m = (B[j][0] << 1) + B[j][5]

                n = (B[j][1] << 3) + (B[j][2] << 2) + (B[j][3] << 1) + B[j][4]

                v = des.\_\_sbox[j][(m << 4) + n]

                Bn[pos] = (v & 8) >> 3

                Bn[pos + 1] = (v & 4) >> 2

                Bn[pos + 2] = (v & 2) >> 1

                Bn[pos + 3] = v & 1

                pos += 4

                j += 1

**self**.R = **self**.\_\_permutate(des.\_\_p, Bn)

**self**.R = list(map(lambda *x*, *y*: x ^ y, **self**.R, **self**.L))

**self**.L = tempR

            i += 1

            iteration += iteration\_adjustment

**self**.final = **self**.\_\_permutate(des.\_\_fp, **self**.R + **self**.L)

        return **self**.final

    def crypt(*self*, *data*, *crypt\_type*):

        if not data:

            return ''

        if len(data) % **self**.block\_size != 0:

            if crypt\_type == des.DECRYPT:

                raise ValueError("Invalid data length, data must be a multiple of " + str(**self**.block\_size) + " bytes\n.")

            if not **self**.getPadding():

                raise ValueError("Invalid data length, data must be a multiple of " + str(**self**.block\_size) + " bytes\n. Try setting the optional padding character")

            else:

                data += (**self**.block\_size - (len(data) % **self**.block\_size)) \* **self**.getPadding()

        if **self**.getMode() == CBC:

            if **self**.getIV():

                iv = **self**.\_\_String\_to\_BitList(**self**.getIV())

            else:

                raise ValueError("For CBC mode, you must supply the Initial Value (IV) for ciphering")

        i = 0

        dict = {}

        result = []

        while i < len(data):

            block = **self**.\_\_String\_to\_BitList(data[i:i+8])

            if **self**.getMode() == CBC:

                if crypt\_type == des.ENCRYPT:

                    block = list(map(lambda *x*, *y*: x ^ y, block, iv))

                processed\_block = **self**.\_\_des\_crypt(block, crypt\_type)

                if crypt\_type == des.DECRYPT:

                    processed\_block = list(map(lambda *x*, *y*: x ^ y, processed\_block, iv))

                    iv = block

                else:

                    iv = processed\_block

            else:

                processed\_block = **self**.\_\_des\_crypt(block, crypt\_type)

            result.append(**self**.\_\_BitList\_to\_String(processed\_block))

            i += 8

        if \_pythonMajorVersion < 3:

            return ''.join(result)

        else:

            return bytes.fromhex('').join(result)

    def encrypt(*self*, *data*, *pad*=None, *padmode*=None):

        data = **self**.\_guardAgainstUnicode(data)

        if pad is not None:

            pad = **self**.\_guardAgainstUnicode(pad)

        data = **self**.\_padData(data, pad, padmode)

        return **self**.crypt(data, des.ENCRYPT)

    def decrypt(*self*, *data*, *pad*=None, *padmode*=None):

        data = **self**.\_guardAgainstUnicode(data)

        if pad is not None:

            pad = **self**.\_guardAgainstUnicode(pad)

        data = **self**.crypt(data, des.DECRYPT)

        return **self**.\_unpadData(data, pad, padmode)

class triple\_des(\_baseDes):

    def \_\_init\_\_(*self*, *key*, *mode*=ECB, *IV*=None, *pad*=None, *padmode*=PAD\_NORMAL):

        \_baseDes.\_\_init\_\_(**self**, mode, IV, pad, padmode)

**self**.setKey(key)

    def setKey(*self*, *key*):

**self**.key\_size = 24

        if len(key) != **self**.key\_size:

            if len(key) == 16:

**self**.key\_size = 16

            else:

                raise ValueError("Invalid triple DES key size. Key must be either 16 or 24 bytes long")

        if **self**.getMode() == CBC:

            if not **self**.getIV():

**self**.\_iv = key[:**self**.block\_size]

            if len(**self**.getIV()) != **self**.block\_size:

                raise ValueError("Invalid IV, must be 8 bytes in length")

**self**.\_\_key1 = des(key[:8], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

**self**.\_\_key2 = des(key[8:16], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

        if **self**.key\_size == 16:

**self**.\_\_key3 = **self**.\_\_key1

        else:

**self**.\_\_key3 = des(key[16:], **self**.\_mode, **self**.\_iv,

**self**.\_padding, **self**.\_padmode)

        \_baseDes.setKey(**self**, key)

    def setMode(*self*, *mode*):

        \_baseDes.setMode(**self**, mode)

        for key in (**self**.\_\_key1, **self**.\_\_key2, **self**.\_\_key3):

            key.setMode(mode)

    def setPadding(*self*, *pad*):

        \_baseDes.setPadding(**self**, pad)

        for key in (**self**.\_\_key1, **self**.\_\_key2, **self**.\_\_key3):

            key.setPadding(pad)

    def setPadMode(*self*, *mode*):

        \_baseDes.setPadMode(**self**, mode)

        for key in (**self**.\_\_key1, **self**.\_\_key2, **self**.\_\_key3):

            key.setPadMode(mode)

    def setIV(*self*, *IV*):

        \_baseDes.setIV(**self**, IV)

        for key in (**self**.\_\_key1, **self**.\_\_key2, **self**.\_\_key3):

            key.setIV(IV)

    def encrypt(*self*, *data*, *pad*=None, *padmode*=None):

        ENCRYPT = des.ENCRYPT

        DECRYPT = des.DECRYPT

        data = **self**.\_guardAgainstUnicode(data)

        if pad is not None:

            pad = **self**.\_guardAgainstUnicode(pad)

*# Pad the data accordingly.*

        data = **self**.\_padData(data, pad, padmode)

        if **self**.getMode() == CBC:

**self**.\_\_key1.setIV(**self**.getIV())

**self**.\_\_key2.setIV(**self**.getIV())

**self**.\_\_key3.setIV(**self**.getIV())

            i = 0

            result = []

            while i < len(data):

                block = **self**.\_\_key1.crypt(data[i:i+8], ENCRYPT)

                block = **self**.\_\_key2.crypt(block, DECRYPT)

                block = **self**.\_\_key3.crypt(block, ENCRYPT)

**self**.\_\_key1.setIV(block)

**self**.\_\_key2.setIV(block)

**self**.\_\_key3.setIV(block)

                result.append(block)

                i += 8

            if \_pythonMajorVersion < 3:

                return ''.join(result)

            else:

                return bytes.fromhex('').join(result)

        else:

            data = **self**.\_\_key1.crypt(data, ENCRYPT)

            data = **self**.\_\_key2.crypt(data, DECRYPT)

            return **self**.\_\_key3.crypt(data, ENCRYPT)

    def decrypt(*self*, *data*, *pad*=None, *padmode*=None):

        ENCRYPT = des.ENCRYPT

        DECRYPT = des.DECRYPT

        data = **self**.\_guardAgainstUnicode(data)

        if pad is not None:

            pad = **self**.\_guardAgainstUnicode(pad)

        if **self**.getMode() == CBC:

**self**.\_\_key1.setIV(**self**.getIV())

**self**.\_\_key2.setIV(**self**.getIV())

**self**.\_\_key3.setIV(**self**.getIV())

            i = 0

            result = []

            while i < len(data):

                iv = data[i:i+8]

                block = **self**.\_\_key3.crypt(iv,    DECRYPT)

                block = **self**.\_\_key2.crypt(block, ENCRYPT)

                block = **self**.\_\_key1.crypt(block, DECRYPT)

**self**.\_\_key1.setIV(iv)

**self**.\_\_key2.setIV(iv)

**self**.\_\_key3.setIV(iv)

                result.append(block)

                i += 8

            if \_pythonMajorVersion < 3:

                data = ''.join(result)

            else:

                data = bytes.fromhex('').join(result)

        else:

            data = **self**.\_\_key3.crypt(data, DECRYPT)

            data = **self**.\_\_key2.crypt(data, ENCRYPT)

            data = **self**.\_\_key1.crypt(data, DECRYPT)

        return **self**.\_unpadData(data, pad, padmode)

root = Tk()

root.geometry("400x400")

root.title("刘世卓8003119050 图像加解密")

ivInt = bytearray("AStringOfIV=", "utf-8")                                 *#key1*

keyInt = bytearray("liushizhuotailihailebahahahahaha", "utf-8")            *#key2*

def encrypt\_image():

    with open('image.png', 'rb') as f:

        jpgdata=f.read()

    k = triple\_des(base64.b64decode(keyInt), *mode*=CBC, *IV*=base64.b64decode(ivInt),*pad*=None, *padmode*=PAD\_PKCS5)    *#encryption*

    print ("please wait for a moment")

    jpgdata = k.encrypt(jpgdata)

    with open('Encrypted', 'wb') as f:

        f.write(jpgdata)

    print("Encryption Complete")

b = Button(root, *text*="Encrypt", *command*=encrypt\_image)

b.place(*x*=100, *y*=210)

def decrypt\_image():

    with open('Encrypted', 'rb') as f:

        jpgdata=f.read()

    k = triple\_des(base64.b64decode(keyInt), *mode*=CBC, *IV*=base64.b64decode(ivInt), *padmode*=PAD\_PKCS5)       *#decryption*

    print ("please wait for a moment")

    jpgdata = k.decrypt(jpgdata)

    with open('Decrypted.png', 'wb') as f:

        f.write(jpgdata)

    print("Decryption Complete")

b = Button(root, *text*="Decrypt", *command*=decrypt\_image)

b.place(*x*=200, *y*=210)

def check\_image():

    bin2img.it()

    img = Image.open('Encrypted.png')

    plt.imshow(img)

    plt.show()

b = Button(root, *text*="checkdecryimg", *command*=check\_image)

b.place(*x*=300, *y*=210)

def check\_image1():

    img = Image.open('image.png')

    plt.imshow(img)

    plt.show()

b = Button(root, *text*="checkimg", *command*=check\_image1)

b.place(*x*=300, *y*=110)

exit\_button = Button(root, *text*="Exit", *command*=root.quit)

exit\_button.place(*x*=310, *y*=310)

root.mainloop()

1. 参考文献

[1] 杨波.现代密码学（第4版）[M].北京：清华大学出版社.2017,7.