DES 加密算法实现

张境文 16337305

一、DES 的结构

DES 加密算法的结构和 Feistel 密码结构是一致的，DES 是 Feistel 密码的一种。DES 算法处理的整个的单位是 bit，DES 是分组加密算法。DES 一次加密的是 64 位的字符串。最后产生也是 64 位的密文。

二、DES 的加密过程

输入的 64 位明文首先需要通过 IP 置换后形成新的明文序列。而加密的过程需要进行 16 轮，每一轮都需要一个子密钥来进行加密。原来输入的密钥是 64 位的，我们需要先去掉 8 位的奇偶校验位，然后在剩下的 56 位密钥中产生 16 个子密钥。56 位的密钥首先分为 28 位的两个部分，每个部分在每一轮中都要循环左移一定的位数，然后合并产生新的 56 位的密钥，这个 56 位的密钥通过 PC2 置换矩阵之后，压缩为 48 位的密钥用于每一轮加密。轮密钥产生过程如下图：

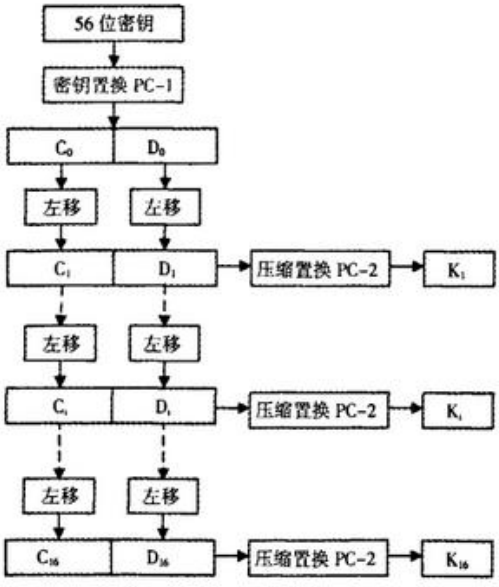
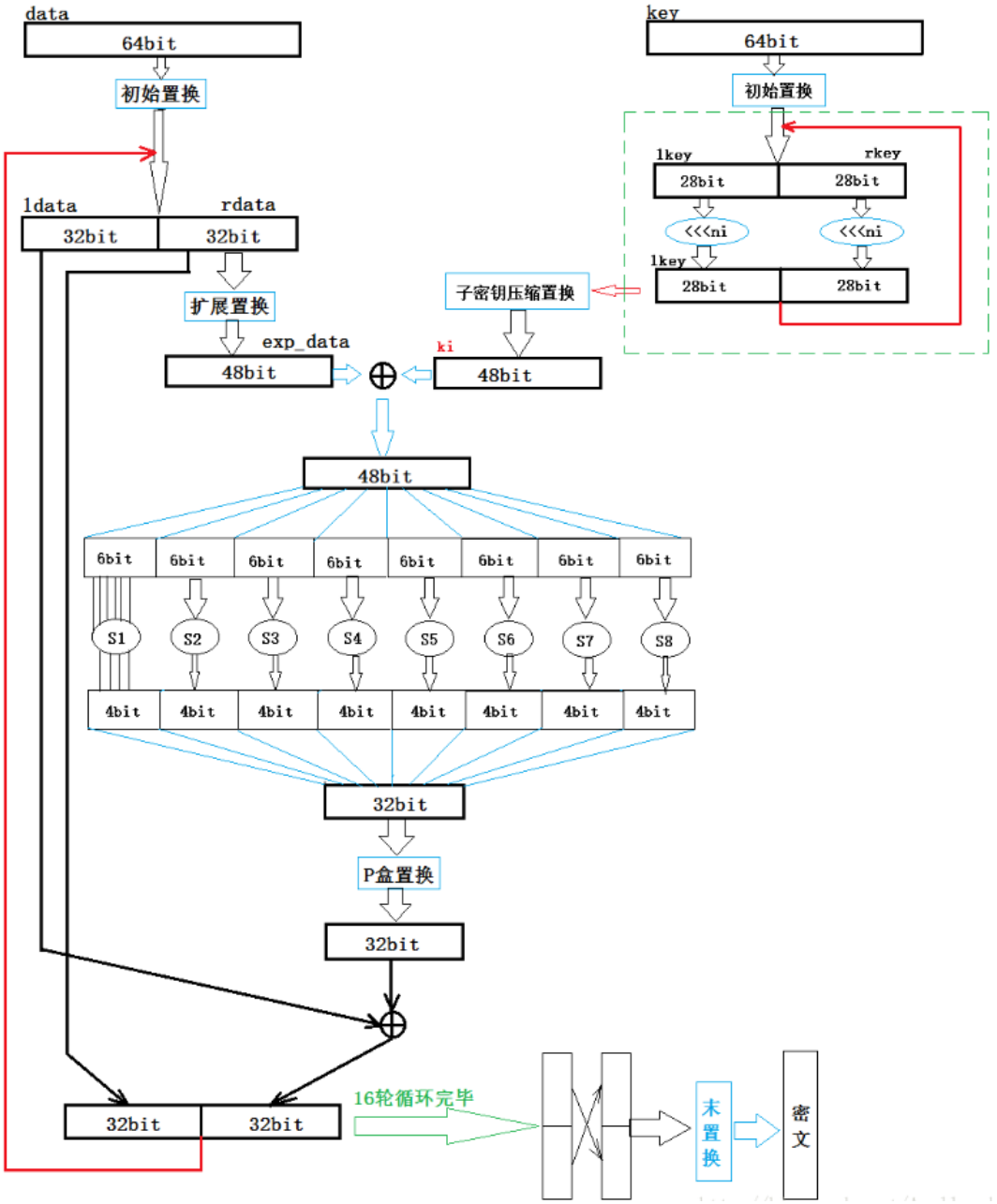


图 2 子密钥 Ki 的产生

输入的 64 位明文分为 32 位的左半部分和 32 位的右半部分。通过 e 盒变换，将右半部分扩展位 48 位的，然后和轮密钥异或。再通过 8 个 s 盒压缩为 32 位，最后经过 P 盒完成得到一个新的 32 位串。新的 32 位串再与左半部分异或放到右半部分，而原来的右半部分放在左半部分。这样就完成了一轮的加密。后面的十五轮加密是一样的过程。最后同归逆 IP 置换产生最终的密文。DES 的一轮加密过程如图：



DES 的解密过程和加密过程差不多。区别在于第一次解码过程用的是第 16 轮的

轮密钥，第 2 轮用的是第 15 轮的轮密钥，以此类推。

三、算法实现

(1)16 轮子密钥产生

```
    def saveKey(self):
        key = self.key.text()
        if key:
            self.subKey.clear()
            bits = self.string2bits(key)
            if len(bits) < 64:
                bits += [0] * (64 - len(bits))
            elif len(bits) > 64:
                bits = bits[:64]

            bits_56 = self.permutate(bits, np.array(self.PC_1)-1)

            left_key_28 = bits_56[:28]
            right_key_28 = bits_56[28:]
            for i in range(16):
                for j in range(self.leftShiftNum[i]):
                    left_key_28.append(left_key_28[0])
                    left_key_28.pop(0)

                    right_key_28.append(right_key_28[0])
                    right_key_28.pop(0)
                self.subKey.append(self.permutate(left_key_28 +
right_key_28, np.array(self.PC_2)-1))
```

（2）明文加密

```
    def onEncryptionClicked(self):
        plainText = self.textEdit_1.toPlainText()

        if plainText and len(self.subKey):
            textBits = self.string2bits(plainText)#正确
            if len(textBits) % 64 != 0:
                textBits.extend([0] * (64 - (len(textBits) % 64)))
            blocks = [textBits[i:i+64] for i in range(0, len(textBits),
64)]

            result = []
```

```
            for block in blocks:
                result.extend(self.encryption(block))
            text = self.bits2string(result)
            hexText = self.string2hex(text)
            self.textBrowser_1.setText(hexText)
```

（3）单个分组加密过程

```
    def encryption(self, block):#block 是 64 位的 0-1
        permutate_block = self.permutate(block, np.array(self.IP)-1)
        left_block = permutate_block[:32]
        right_block = permutate_block[32:]
        for i in range(16):
            extand_right_block = self.permutate(right_block,
np.array(self.e)-1)
            right_block_48 = (np.array(extand_right_block) ^
np.array(self.subKey[i])).tolist()
            right_block_32 = []
            for j in range(8):
                right_block_6 = right_block_48[j*6:(j+1)*6]
                right_block_4 = self.compressBlock(right_block_6, j)
                right_block_32.extend(right_block_4)
            permutate_right_block_32 = self.permutate(right_block_32,
np.array(self.P)-1)
            temp_block = right_block
            right_block = (np.array(left_block) ^
np.array(permutate_right_block_32)).tolist()
            left_block = temp_block
        return self.permutate(right_block + left_block,
np.array(self.invIP)-1)
```

(4)解密密文

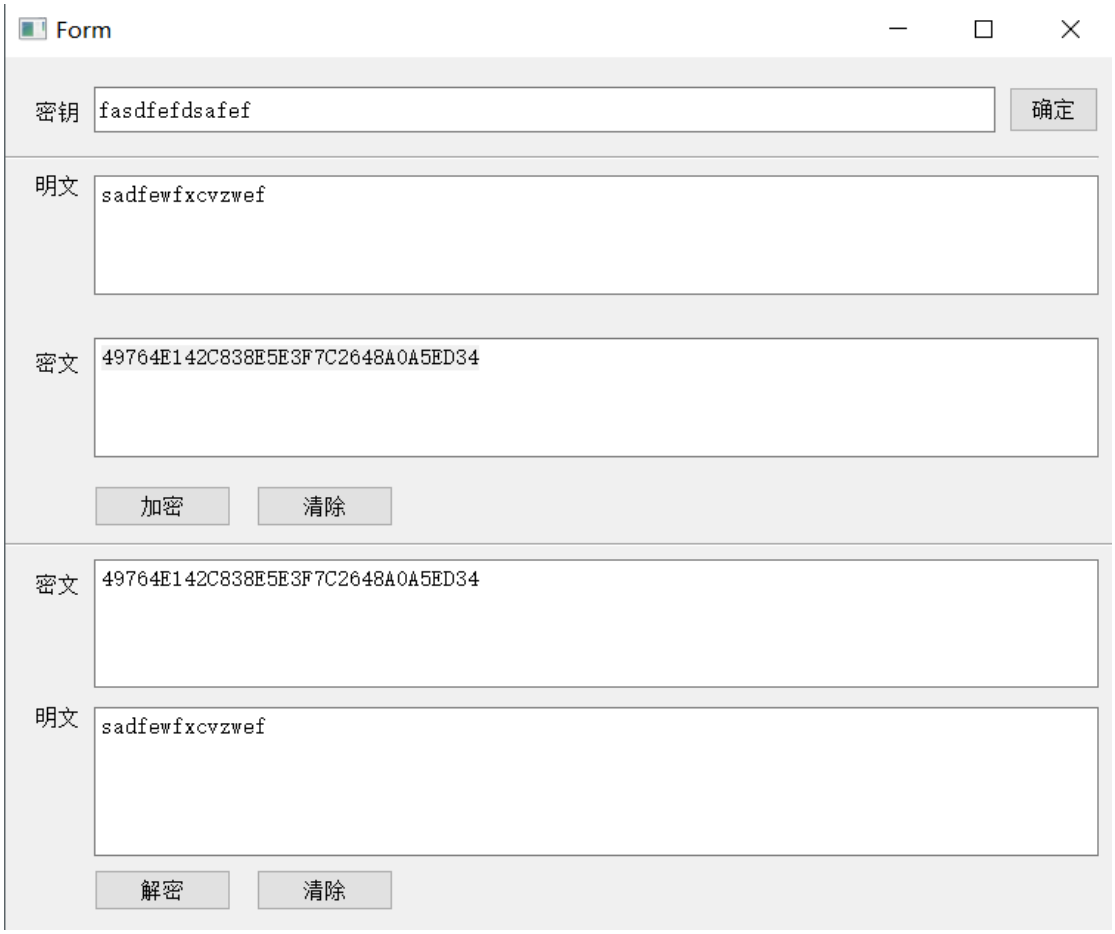
```
    def onDecryptionClicked(self):
        enText = self.textEdit_2.toPlainText()#十六进制
        if enText and len(self.subKey):
            enText = str.upper(enText)
            textBits = self.hex2bits(enText)
            if len(textBits) % 64 != 0:
                textBits.extend([0] * (64 - len(textBits) % 64))
            blocks = [textBits[i: i+64] for i in range(0, len(textBits),
64)]
            result = []
            for block in blocks:
                result.extend(self.decryption(block))
            text = self.bits2string(result)
```

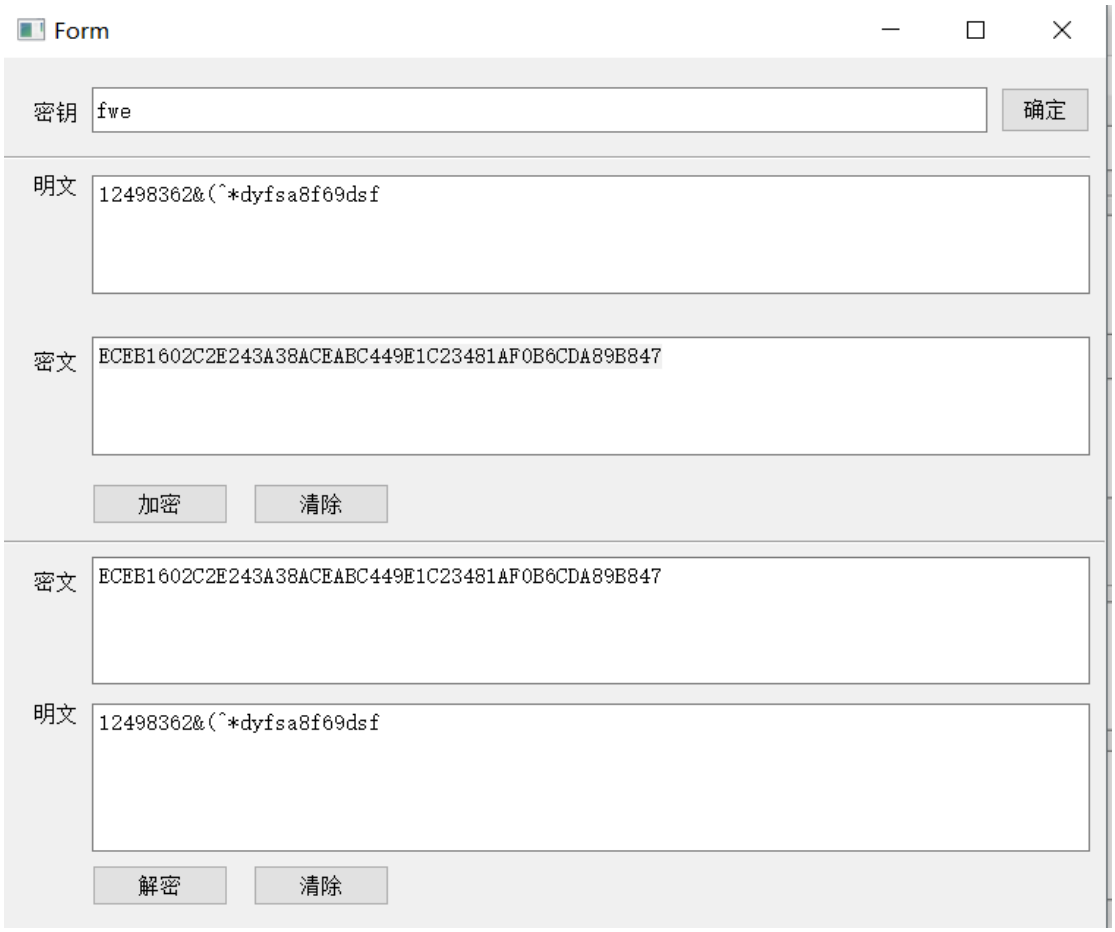
```
            self.textBrowser_2.setText(text)
```

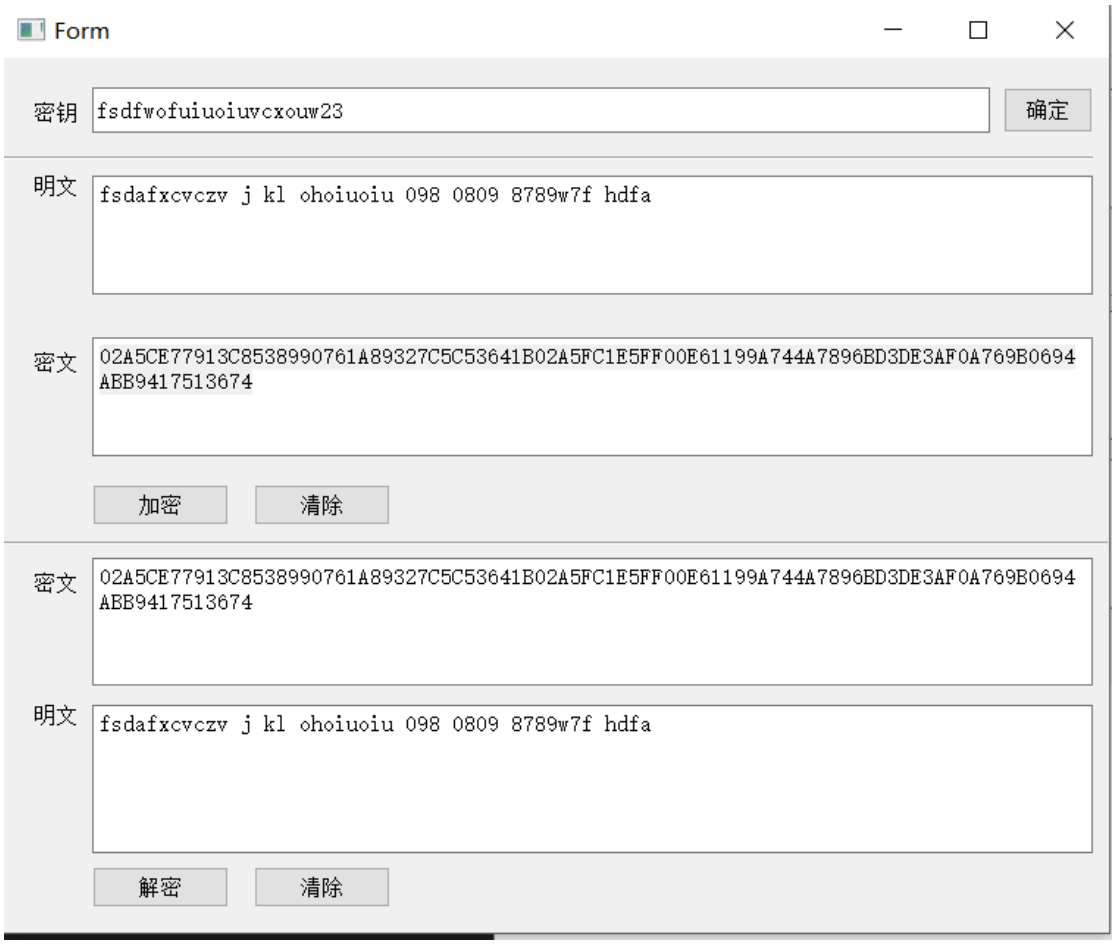
（5）解密一个分组

```
    def decryption(self, block):
        permutate_block = self.permutate(block, np.array(self.IP)-1)
        right_block = permutate_block[32:]
        left_block = permutate_block[:32]
        for i in range(15, -1, -1):
            extand_right_block = self.permutate(right_block,
np.array(self.e) - 1)
            right_block_48 = (np.array(extand_right_block))
^(np.array(self.subKey[i])).tolist()
            right_block_32 = []
            for j in range(8):
                right_block_6 = right_block_48[j*6: (j+1)*6]
                right_block_4 = self.compressBlock(right_block_6, j)
                right_block_32.extend(right_block_4)
            permutate_right_block_32 = self.permutate(right_block_32,
np.array(self.P)-1)
            temp_block = right_block
            right_block = (np.array(left_block) ^
np.array(permutate_right_block_32)).tolist()
            left_block = temp_block
        return self.permutate(right_block+left_block,
np.array(self.invIP) - 1)
```

四、运行结果


Form
密钥
fasdfefdsafef
确定
明文
sadfewfxcvzwef
密文
49764E142C838E5E3F7C2648A0A5ED34
加密
清除
密文
49764E142C838E5E3F7C2648A0A5ED34
明文
sadfewfxcvzwef
解密
清除


Form
密钥
fwe
确定
明文
12498362&(^*dyfsa8f69dsf
密文
ECEB1602C2E243A38ACEABC449E1C23481AF0B6CDA89B847
加密
清除
密文
ECEB1602C2E243A38ACEABC449E1C23481AF0B6CDA89B847
明文
12498362&(^*dyfsa8f69dsf
解密
清除


Form
密钥
fsdfwofuiuoiuvcxouw23
确定
明文
fsdafxcvczv j kl ohoiuoiu 098 0809 8789w7f hdfa
密文
02A5CE77913C8538990761A89327C5C53641B02A5FC1E5FF00E61199A744A7896BD3DE3AF0A769B0694ABB9417513674
加密
清除
密文
02A5CE77913C8538990761A89327C5C53641B02A5FC1E5FF00E61199A744A7896BD3DE3AF0A769B0694ABB9417513674
明文
fsdafxcvczv j kl ohoiuoiu 098 0809 8789w7f hdfa
解密
清除