

中国矿业大学计算机学院

2020 级本科生课程报告

课程名称 密码学课程设计

学生姓名 周子欣

学 号 12204202

专 业 信息安全

任课教师 张立江

报告时间 2022.11.3

目录

一. 古典密码算法	4
1. 维吉尼亚密码	4
1.1 算法原理	4
1.2 已知密钥维吉尼亚加解密核心代码实现	4
1.3 基于词频统计的长密文爆破核心代码实现	5
1.4 字典攻击核心代码实现	6
2. 希尔密码	7
2.1 算法原理	7
2.2 核心代码实现	8
3. 实验总结	9
二. 分组加密算法	10
1. AES 算法	10
1.1 算法原理	10
1.2 aes_128 位加解密核心代码实现	10
1.3 工作模式核心代码实现	12
1.4 正确性验证	13
2. SM4 国密算法	14
2.1 算法原理	14
2.2 核心代码实现	15
3. SM4 与 AES 的区别	16
4. 实验总结	17
三. 序列密码	17
1. 线性反馈移位寄存器	17
1.1 算法原理	17
1.2 核心代码实现	18
1.3 LFSR 周期性正确性验证	19
2. 祖冲之序列密码	20
2.1 算法原理	20
2.2 核心代码实现	20
2.3 ZUC 安全性分析	21
3. 实验总结	22
四. HASH 函数	22
1. MD5	22
1.1 算法原理	22
1.2 代码实现	22
1.3 正确性验证	23
2. SM3	24
2.1 算法原理	24

2.2 核心代码实现	24
2.3 正确性验证	24
3. SM3 与 MD5, SHA 算法的区别	25
4. 实验总结	25
五. 公钥密码	25
1. RSA	25
1.1 算法原理	25
1.2 核心代码实现	26
1.3 RSA-OAEP 填充模式	27
1.4 OAEP 模式核心代码实现	28
1.5 正确性验证	29
2. 实验总结	30
六. 综合实验	30
1. 需求分析	30
2. 功能流程图	30
3. 核心代码实现	31
4. 实现效果	33
5. 加密传输分析	35
6. 实验总结	36

一.古典密码算法

1. 维吉尼亚密码

1.1 算法原理

维吉尼亚密码是使用一系列凯撒密码组成密码字母表的加密算法，属于多表密码的一种简单形式。 encrypt

用数字 0-25 代替字母 A-Z，维吉尼亚密码的加密算法可以写成同余的形式：

$$C_i = P_i + K_i(\text{mod } 26)$$

解密的过程则与加密相反：

$$C_i = P_i - K_i(\text{mod } 26)$$

1.2 已知密钥维吉尼亚加解密核心代码实现

(1)加密函数

```
Letters="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def encrypt(key,message):

    result=''
    l=Len(key)
    key_all=key*(Len(message)//l)
    j=0 #密钥指针

    for i in message:
        index=(Letters.find(i)+key[j])%26
        j+=1
        result=result+Letters[index]

    return result
```

(2)解密函数

```
def decrypt(key,message):

    result=''
    l=Len(key)
    j=0

    for i in message:
        index=(Letters.find(i)-Letters.find(key[j]))%26
        j=(j+1)%l
        result=result+Letters[index]

    text=' '.join(wordninja.split(result))

    return text
```

(3)特点

①采用查表的方式实现位移，而不是直接用 ASCII 码值相加的方法，节省了时间，但增加了空间，而且如果有超出字母外的字符，需要扩充字母表。

②利用了分词工具 wordninja,在解密结果为连续文本的情况下,自动分词,保证了解密文本的可读性。

(4)正确性验证

在网络上找到的一段被维吉尼亚加密的文本,解密后得到一串有意义是字符,且解密后的文本已被分词,有较强可读性。

```

加密输入e,解密输入d,退出输入0
d
输入要解密的文本
krkpewxvftksopztecxbuhfvycgxouflihoffptrcwfwhkcevxiuzfposdvccyctpmjtbtfymilctiwxacsmjmoncwnawjrwjtjgsuystvbxgvcmgczbqecllttfkjlacpft
tjgeegtbvkfpmhjzqaxhvvpgxoeychrcwumchhyigixhqdciaunmjerfekcozqttnfdjlopuyqhjrjawcpfrgxhwiljrgiycrqkajfgvrlrxgkghdbqnliaovzrltgaf
slacjvjexrjrdzsvruprttfkwxfglstdznmjerdvjdihkwdngjfsawgjfunhitjcaykgrptzicibtwrcpycwbkxfibrqemivotvwdnotvldmvgicshbqkzmtfqlzaxrqekntq
efscmbqkfxguyzjaaorgccmcovrwxckgdgonrqhxadcclbnzjfdpzegegtqawygkgcjasofqiecxvbdyageztjikvrxymlapgh
cbcrtfghdnhitjcaytqiknlnswgrtbpfrlkwvvyccraqicqnhpfrwbizliasyfpawqqljshlgkmtccumgxmqlemcvcysxovy
输入密钥
CRYPTO
解密后的结果为:IAM ALIVE HERE MY BELOVED FOR THE REASON TO ADORE YOU OH HOW ANXIOUS I HAVE BEEN FOR YOU AND HOW SORRY IAM ABOUT ALL YOU
MUST HAVE SUFFERED IN HAVING NO NEWS FROM US MAY HEAVEN GRANT THAT THIS LETTER REACHES YOU DO NOT WRITE TOME THIS WOULD COMPROMISE
ALL OF US AND ABOVE ALL DO NOT RETURN UNDER ANY CIRCUMSTANCES IT IS KNOWN THAT IT WAS YOU WHO HELPED US TO GETAWAY FROM HERE AND ALL
WOULD BE LOST IF YOU SHOULD SHOW YOURSELF WE ARE GUARDED DAY AND NIGHT IDO NOT CARE YOU ARE NO THERE DO NOT BE TROUBLED ON MY
ACCOUNT NOTHING WILL HAPPEN TOME THE NATIONAL ASSEMBLE WILL SHOW LENIENCY FAREWELL THE MOST LOVED OF MEN BE QUIET IF YOU CAN TAKE
CARE OF YOURSELF FOR MYSELF I CANNOT WRITE ANYMORE BUT NOTHING IN THE WORLD COULD STOP ME TO ADORE YOU UP TO THE DEATH
加密输入e,解密输入d,退出输入0

```

图 1.1 维吉尼亚已知密钥解密验证

1.3 基于词频统计的长密文爆破核心代码实现

(1)利用重合指数法得到密钥长度

```

def getIC(Zd):
    #计算重合指数
    fi=0
    n=0 #总数
    for k in Zd:
        fi=fi+Zd[k]*(Zd[k]-1)
        n=n+Zd[k]
    p=fi/(n*(n-1))

    return p

def getLenKey(message):
    #得到密钥长度
    #方法：计算子串重合指数
    cnt=0
    pi=0
    for i in range(1,Len(Letters)):
        #print('key_length=%s'%i)
        for j in range(1,i+1):
            str=message[j:i]
            LettersZd=getCnt(str)

            pi=getIC(LettersZd)

            if pi>0.055 and pi<0.075:
                cnt+=1
        if cnt==i:
            return i
            break
    cnt=0
    pi=0

```

(2)利用重合互指数法爆破偏移得到密钥

```

def getMI(str_1):
    # 计算字符串与标准英语之间的重合互指数
    Zd_1=[0.082,0.015,0.028,0.043,0.127,0.022,0.020,0.061,0.070,0.002,0.008,0.040,0.024,0.067,0.075,0.019,0.001,0.060,0.063,0.091,0.028,0.010,0.023,0.001,0.020,0.001]
    Zd_2=getCnt(str_1)

    matrix_1=np.array(Zd_1)
    matrix_2=np.array(list(Zd_2.values())) # 转成数组便于计算内积

    n2=np.sum(matrix_2)
    pi=matrix_2/n2

    Hp=np.sum(pi*matrix_1)

    return Hp

def getKey(message):
    # 破解密钥
    key_len=getLenKey(message)

    group=[] # 按照密钥长度分组
    h_key={} # 存储偏移k
    MI={} # 存储偏移值
    Letter_key='' # 储存字母加密密钥

    for i in range(key_len):
        # 按照密钥长度分组
        group.append(message[i::key_len])

    for i in range(0,key_len):
        # 求出每组的所有重合互指数和最大重合互指数对应的偏移
        MI[i],h_key[i]=getListMI(i,group)

    for i in range(key_len):
        # 根据偏移写出密钥, 因为偏移是相对密文的, 而密钥则是相对明文的
        m=Letters[(26-h_key[i])%26]
        Letter_key=Letter_key+m

    return Letter_key

```

(3)特点

①利用 numpy 库生成数组, 进行内积运算, 代码功能更明确

②利用了分词工具 wordninja,在解密结果为连续文本的情况下, 自动分词, 保证了解密文本的可读性。

(4)正确性验证

以教材 P65 页习题 5(6)为例, 解密题中所给密文

待解密得到密文为:CHREE VOAHM AERAT BIAXX WTNXB EEOPH BSBQM QEQR BWRVX UOAKX AOSXX WEAHB WGIJMM QMNKG RFVGX WTRZX WIAKL
XFPSK AUTEM NDCMG TSXMX BTUIA DNGMG PSREL XNJEL XVRVP RTULH DNQWT WDTYG BPHXT FALJH ASVBF XNGLL CHRZB WELEK MSJIK
NBHWR JGNMG JSGLX FEYPH AGNRB IEQJT AMRVL CRREM NDGLX RRING NSNRW CHRQH AEYEV TAQEB BIPEE WEVKA KOEWA DREMX MTBHH
CHRTK DNRVZ CHRCL QOHPW QAIIW XNRMG WOIF KEE
密钥为: JANET
解密结果为: THE ALMOND TREE WAS IN TENTATIVE BLOSSOM THE DAYS WERE LONGER OFTEN ENDING WITH MAGNIFICENT EVENINGS OF
CORRUGATED PINK SKIES THE HUNTING SEASON WAS OVER WITH HOUNDS AND GUNS PUT AWAY FOR SIX MONTHS THE VINEYARDS WERE BUSY
AGAIN AS THE WELL ORGANIZED FARMERS TREATED THEIR VINES AND THE MORE LACKADAISICAL NEIGHBORS HURRIED TO DO THE PRUNING
THEY SHOULD HAVE DONE IN NOV BR VE

图 2.2 维吉尼亚基于词频统计的爆破长密文解密验证

1.4 字典攻击核心代码实现

(1)判断是否为英文函数

```

def isEnglish(message, wordPercentage=20, letterPercentage=85):
    wordsMatch = getEnglishCount(message) * 100 >= wordPercentage
    numLetters = len(removeNonLetters(message))
    messageLettersPercentage = float(numLetters) / len(message) * 100
    lettersMatch = messageLettersPercentage >= letterPercentage
    return wordsMatch and lettersMatch

```

(2)爆破函数

```
def Hack(message):

    fo=open('dictionary.txt','r')
    words=fo.readlines()    #返回一个字符串列表，每个字符串是文件的一行
    fo.close()

    #开始爆破
    for word in words:
        word=word.strip()
        st=decrypt(word,message)

        #message=message.replace(" ", "") #清除输入中的空格
        #print(st)
        if detectEnglish.isEnglish(st,wordPercentage=40):
            print("密钥可能为%s"%word)
            print("对应明文可能为%s"%st)
```

(2)特点

- ①针对密钥为单词的情况进行爆破，判断结果是否为有意义的英文
- ②通过对解密后的文本中单词百分比和英文字符百分比是否达到一定含量来判断是否为有意义的英文

(3)正确性验证

同样以教材 P65 页习题 5（6）为例，解密题中所给密文

```
42 def main():
43     message='CHREE VOAHM AERAT BIAXX WTNXB EEOPH BSBQM QEQR BWRVX UOAKX AOSXX WEAHB WJMM QMNKG RFVGX WTRZ'
44     message=message.replace(" ", "")
45     Hack(message.upper())
46
```

密钥可能为JANET
 对应明文可能为THE ALMOND TREE WAS IN TENTATIVE BLOSSOM THE DAYS WERE LONGER OFTEN ENDING WITH MAGNIFICENT EVENINGS OF CORRUGATED
 PINK SKIES THE HUNTING SEASON WAS OVER WITH HOUNDS AND GUNS PUT AWAY FOR SIX MONTHS THE VINEYARDS WERE BUSY AGAIN AS THE WELL
 ORGANIZED FARMERS TREATED THEIR VINES AND THE MORE LACKADAISICAL NEIGHBORS HURRIED TO DO THE PRUNING THEY SHOULD HAVE DONE IN
 NOV BR VE

图 3.3 维吉尼亚字典攻击解密验证

2. 希尔密码

2.1 算法原理

Hill 密码的基本思想是通过线性变换将 n 个连续的明文字母替换为 n 个密文字母。Hill 密码的实质就是通过一个变换矩阵把明文变换为密文的一种密码体制。

加密过程：

$$[c]_{1*n} \equiv [p]_{1*n} \times [k]_{n*n} (\text{mod} 26)$$

解密过程：

$$[p]_{1*n} \equiv [c]_{1*n} \times [k]_{n*n}^{-1} (\text{mod} 26)$$

2.2 核心代码实现

(1)加密函数

```
def encrypt(message,m,key):
    #Hill 加密
    result='' #储存加密后的字符串

    m_arr=strToArr(message) #将明文变成int型一维数组
    #print(m_arr)
    m_arr=m_arr.reshape(-1,m) #将数组变成m列的向量
    #print(m_arr)
    c_arr=(np.matmul(m_arr,key))%len(Letters) #加密

    c_arr=c_arr.reshape(-1) #将结果变为一维数组
    c_arr=np.around(c_arr)
    c_arr=c_arr.astype(int)
    c_list=c_arr.tolist()
    #print(c_list)
    result=ListToStr(c_list) #将列表中的数字转成对应字母

    return result
```

(2)整数域上的矩阵求逆函数

```
def matrix_inv(key):
    #整数域上求矩阵逆元
    k_1=np.linalg.inv(key) #求key的逆矩阵,但由于其不是整数,故通过行列式的乘法逆元乘上伴随矩阵得到整数逆元
    k_det=np.linalg.det(key) #求key矩阵的行列式
    k_det_inv=mul_mod_inv(k_det) #求行列式模26的乘法逆元 本质上是1/|A|
    k_2=k_1*k_det%len(Letters) #矩阵的逆乘行列式=伴随矩阵
    k_2=np.around(k_2)
    k_2=k_2.astype(np.int64)#浮点型转int型
    k_3=k_det_inv*k_2%26 #行列式的乘法逆元乘上伴随矩阵得到整数逆元
    k_3=np.around(k_3) #由于伴随矩阵得到的可能是浮点数矩阵,故需要对其进行四舍五入取整
    k_3=k_3.astype(np.int64) #并将每个元素成员强制转换为int类型

    return k_3
```

(2)解密函数

```
def decrypt(c,m,key):
    k_3=matrix_inv(key)#行列式的乘法逆元乘上伴随矩阵得到整数逆元
    return encrypt(c,m,k_3)
```

(3)破译密钥函数

```
def hackKey(p,c,m):
    if len(p)<m*m:
        print('数据过少不可爆破密钥')
        return False
    p_arr=strToArr(p[:m*m])
    p_arr=p_arr.reshape(-1,m)
    c_arr=strToArr(c[0:m*m])
    c_arr=c_arr.reshape(-1,m)
    p_inv=matrix_inv(p_arr)
    key_arr=(np.matmul(p_inv,c_arr))%len(Letters)
    return key_arr.astype(np.int64)
```

(3)特点:

①求加密密钥的逆矩阵因为是求整数域上的逆矩阵,所有不能直接调用求逆函数,要通过行列式的乘法逆元乘上伴随矩阵得到整数逆元。

②采用查表的方式实现位移，而不是直接用 ASCII 码值相加的方法，节省了时间，但增加了空间，而且如果有超出字母外的字符，需要扩充字母表。

(4)正确性验证

①以教材 P65 页习题 5（4）为例，进行希尔加解密。

```

63 def main():
64     message='hill'
65     key=np.array([[8,6,9,5],
66                  [6,9,5,10],
67                  [5,8,4,9],
68                  [10,6,11,4]],dtype=int)
69     m=4
70     c=encrypt(message.upper(),m,key)
71     p=decrypt(c,m,key)
72     print("加密后的密文为"+c)
73     print("对密文解密后的明文为"+p)
74
加密后的密文为JIIY
对密文解密后的明文为HILL
[Finished in 464ms]

```

图 4.4 希尔加解密验证

②以教材 P65 页习题 5（7）为例，检验破译密钥

```

110 p='friday'
111 c='POCFKU'
112 m=2
113 print("明文:"+p+" 与密文: "+c+" 加密密钥为:")
114 hac=hackKey(p.upper(),c.upper(),m)
115 print(hac)
116 print("用求解出的密钥加密"+p+"结果为")
117 print(encrypt(p.upper(),m,hac))
118
119 if __name__ == '__main__':
120     main()
明文:friday 与密文: POCFKU 加密密钥为:
[[ 7  1]
 [ 8 25]]
用求解出的密钥加密friday结果为
POCFKC
[Finished in 230ms]

```

图 5.4 希尔已知密文破解密钥验证

3. 实验总结

此次实验通过对希尔密码以及维吉尼亚密码的实现与分析，对单表、多表替换密码有了较深的学习，并第一次通过编程实现了加密、解密以及破解算法。同时，也认识到了古典密码对于现代计算机的运算能力而言，已经几乎无安全性可言，无论是穷举攻击还是利用统计分析方法，破译它们都没有太大的难度。

二.分组加密算法

1. AES 算法

1.1 算法原理

AES 是作为 DES 的替代标准出现的。AES 明文分组长度为 128 位，即 16 个字节，密钥长度可以为 16 个字节、24 个字节、或 32 个字节，即 128 位密钥、192 位密钥、或 256 位密钥。AES 中没有使用 Feistel 网络，其结构为 SP 结构。

128 位密钥加密结构示意图：

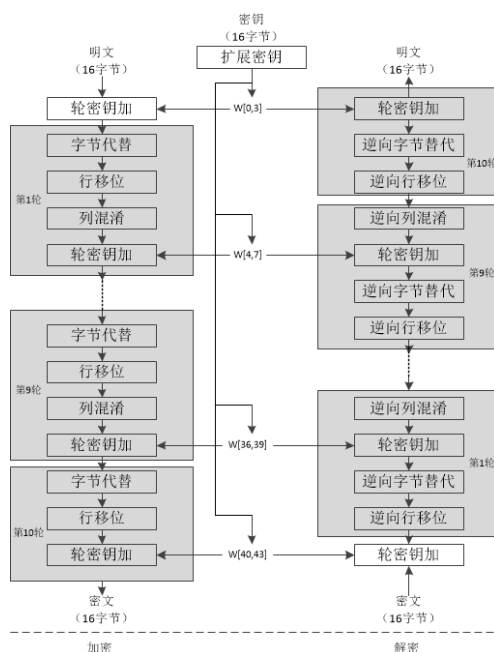


图 2.1 128 位密钥加密结构示意图

1.2 aes_128 位加解密核心代码实现

(1)加密函数

```

def encrypt(message,key):
    mix=message
    #mix=StreamListToMix(message)
    Words=WordExp(key)

    #轮密钥加
    w_mix_1=StreamListToMix(Words[0])
    Result_1=RoundKeyAdd(mix,w_mix_1)
    for i in range(9):
        #字节代换
        Result_2=SubByte(Result_1)
        #行位移
        Result_2=RotHang(Result_2)
        #列混淆
        Result_2=ColMix(Result_2)
        #轮密钥加
        w_mix_i=StreamListToMix(Words[i+1])
        Result_1=RoundKeyAdd(Result_2,w_mix_i)
    #第10轮
    w_mix_11=StreamListToMix(Words[10])
    #字节代换
    Result_2=SubByte(Result_1)
    #行位移
    Result_2=RotHang(Result_2)
    #轮密钥加
    Result_1=RoundKeyAdd(Result_2,w_mix_11)

    return Result_1

```

(2)解密函数

```

def decrypt(message,key):
    #mix=StreamListToMix(message)
    mix=message
    Words=WordExp(key)
    Words.reverse()
    #print(Words)
    #轮密钥加
    w_mix_1=StreamListToMix(Words[0])
    Result_1=RoundKeyAdd(mix,w_mix_1)
    for i in range(9):
        #逆字节代换
        Result_2=ReSubByte(Result_1)
        #逆行位移
        Result_2=ReRotHang(Result_2)
        #逆列混淆
        Result_2=ReColMix(Result_2)
        #轮密钥加
        w_mix_i=ReColMix(StreamListToMix(Words[i+1]))
        Result_1=RoundKeyAdd(Result_2,w_mix_i)
    #第10轮
    w_mix_11=StreamListToMix(Words[10])
    #逆字节代换
    Result_2=ReSubByte(Result_1)
    #逆行位移
    Result_2=ReRotHang(Result_2)
    #轮密钥加
    Result_1=RoundKeyAdd(Result_2,w_mix_11)

    return Result_1

```

(3)特点

- ①对密钥进行列混合使加解密过程相同
- ②在 aes_128.py 中还编写了一些格式转换函数，便于各种输入格式的转换

1.3 工作模式核心代码实现

(1)ECB 模式代码实现

```
def ECBEncrypt(message,key):
```

```
#调整输入格式
m=aes.StrToBinList(message)
if Len(key)<=16:
    k=aes.StrToBinList(key)
else:
    print('密钥长度超出128bit,取前128bit加密')
    k=aes.StrToBinList(key)[:4]

result=''

for i in range(Len(m)//4):
    mix=aes.StreamListToMix(m[4*i:4*i+4])
    res_mix=aes.encrypt(mix,k)
    tmp=aes.MixToStr(res_mix)
    result+=tmp
print(result)
result=aes.StrToBase64(result)
return result
```

```
def ECBDecrypt(message,key):
```

```
#调整输入格式
m=aes.Base64ToStr(message)
m=aes.StrToBinList(m)
if Len(key)<=16:
    k=aes.StrToBinList(key)
else:
    print('密钥长度超出128bit,取前128bit解密')
    k=aes.StrToBinList(key)[:4]

result=''

for i in range(Len(m)//4):
    mix=aes.StreamListToMix(m[4*i:4*i+4])
    res_mix=aes.decrypt(mix,k)
    tmp=aes.MixToStr(res_mix)
    result+=tmp

return result
```

(2)CBC 模式代码实现

```
def CBCEncrypt(message,key,iv):
```

```
#调整输入格式
m=aes.StrToBinList(message)
if Len(key)<=16:
    k=aes.StrToBinList(key)
else:
    print('密钥长度超出128bit,取前128bit加密')
    k=aes.StrToBinList(key)[:4]
if Len(iv)<=16:
    iv=aes.StrToBinList(iv)
else:
    print('IV超出128bit,取前128bit加密')
    iv=aes.StrToBinList(iv)[:4]

result=''

iv_mix=aes.StreamListToMix(iv) #初始化向量矩阵

for i in range(Len(m)//4):
    mix=aes.StreamListToMix(m[4*i:4*i+4])
    mix=iv_mix^mix
    res_mix=aes.encrypt(mix,k)
    tmp=aes.MixToStr(res_mix)
    iv_mix=res_mix
    result+=tmp

result=aes.StrToBase64(result)
return result
```

```
def CBCDecrypt(message,key,iv):
```

```
#调整输入格式
m=aes.Base64ToStr(message)
#print(m)
m=aes.StrToBinList(m)
#print(m)
if Len(key)<=16:
    k=aes.StrToBinList(key)
else:
    print('密钥长度超出128bit,取前128bit解密')
    k=aes.StrToBinList(key)[:4]
if Len(iv)<=16:
    iv=aes.StrToBinList(iv)
else:
    print('IV超出128bit,取前128bit解密')
    iv=aes.StrToBinList(iv)[:4]

result=''

iv_mix=aes.StreamListToMix(iv) #初始化向量矩阵

for i in range(Len(m)//4):
    mix=aes.StreamListToMix(m[4*i:4*i+4])
    res_mix=aes.decrypt(mix,k)
    res_mix=iv_mix^res_mix
    tmp=aes.MixToStr(res_mix)
    iv_mix=mix
    result+=tmp

return result
```

(3)注意

在 base64 编码时采用 latin 编码方式而不是 UTF-8，因为 UTF-8 是变长编码，会和解密网站解出的结果不一致

(4)特点

将 AES 以模块形式导入，便于将工作模式的实现代码用于其他分组密码

1.4 正确性验证

明文: Zhouzixin is a handsome girl.If you like her, she also like you.

密钥: chongyanisyyds

IV: sacsfdvsv

(1)ECB 模式

```

加密输入e,解密输入d,退出输入0
e
ECB模式输入1,CBC模式输入2,退出输入0
1
输入明文(格式是字符串,如:chongyanisyyds):
Zhouzixin is a handsome girl.If you like her, she also like you.
输入密钥(格式是字符串,如:auefawifs):
chongyanisyyds
密文(格式为base64)= n7FDWa6XsU8JiHMHuJBS7aiPYgVT1E8dk9Z7DrbT+Vb6sXghkfoVNA7L8dv27thav4A+bNXGIU+OeEoUPTuAtA==
加密输入e,解密输入d,结束输入0
d
ECB模式输入1,CBC模式输入2,退出输入0
1
输入密文(格式为base64,如:wpjDgM0bw4MRwqrD1kbDjcK1EcKOT8Kpwo1u):
n7FDWa6XsU8JiHMHuJBS7aiPYgVT1E8dk9Z7DrbT+Vb6sXghkfoVNA7L8dv27thav4A+bNXGIU+OeEoUPTuAtA==
输入密钥(格式是字符串,如:auefawifs):
chongyanisyyds
明文= Zhouzixin is a handsome girl.If you like her, she also like you.
加密输入e,解密输入d,结束输入0

```

图 2.2 AES-ECB 模式加解密

在线网站上加密对比,结果相同验证成功

图 2.3 在线 AES-ECB 加解密结果

(2)CBC 模式

```
---AES_128bit ECB与CBC模式加密 明文密钥初始化向量不足均填充0---
加密输入e,解密输入d,退出输入0
e
ECB模式输入1,CBC模式输入2,退出输入0
2
输入明文(格式是字符串,如:chongyanisyyds):
Zhouzixin is a handsome girl.If you like her, she also like you.
输入密钥(格式是字符串,如:auefawifs):
chongyanisyyds
输入IV(格式是字符串,如:sacsfdvsv):
sacsfdvsv
密文(格式为base64)= pua9bpIObPvjH0veuDFDtc28RZHoUi7DyMaQ40N7VDIPVWtnttxFGiQhyx5hqbirNBMF+X9SgjLurnOKTqWwtg==
加密输入e,解密输入d,结束输入0
d
ECB模式输入1,CBC模式输入2,退出输入0
2
输入密文(格式为base64,如:CMKiwpZawppRVs0sNRF4HcOwODLDhA==):
pua9bpIObPvjH0veuDFDtc28RZHoUi7DyMaQ40N7VDIPVWtnttxFGiQhyx5hqbirNBMF+X9SgjLurnOKTqWwtg==
输入密钥(格式是字符串,如:auefawifs):
chongyanisyyds
输入IV(格式是字符串,如:sacsfdvsv):
sacsfdvsv
明文= Zhouzixin is a handsome girl.If you like her, she also like you.
加密输入e,解密输入d,结束输入0
```

图 2.4 在线 AES-CBC 加解密结果

在线网站上加密对比,结果相同验证成功

UTF-8

CBC

None

128bits

chongyanisyyds

sacsfdvsv

加密

解密

pua9bpIObPvjH0veuDFDtc28RZHoUi7DyMaQ40N7VDIPVWtnttxFGiQhyx5hqbirNBMF+X9SgjLurnOKTqWwtg==

图 2.5 在线 AES-CBC 加密结果

2. SM4 国密算法

2.1 算法原理

SM4 是一种 Feistel 结构的分组密码算法,其分组长度和密钥长度均为 128bits。加密算法和密钥扩展算法迭代轮数均为 32 轮。SM4 加解密过程的算法相同但是轮密钥的使用顺序相反。SM4 密码算法使用模 2 加和循环移位作为基本运算。

密钥扩展算法：SM4 算法使用 128 位的加密密钥，并采用 32 轮迭代加密结构，每一轮加密使用一个 32 位的轮密钥，总共使用 32 个轮密钥。因此需要使用密钥扩展算法，从加密密钥中产生 32 个轮密钥。

加密流程大致如下：

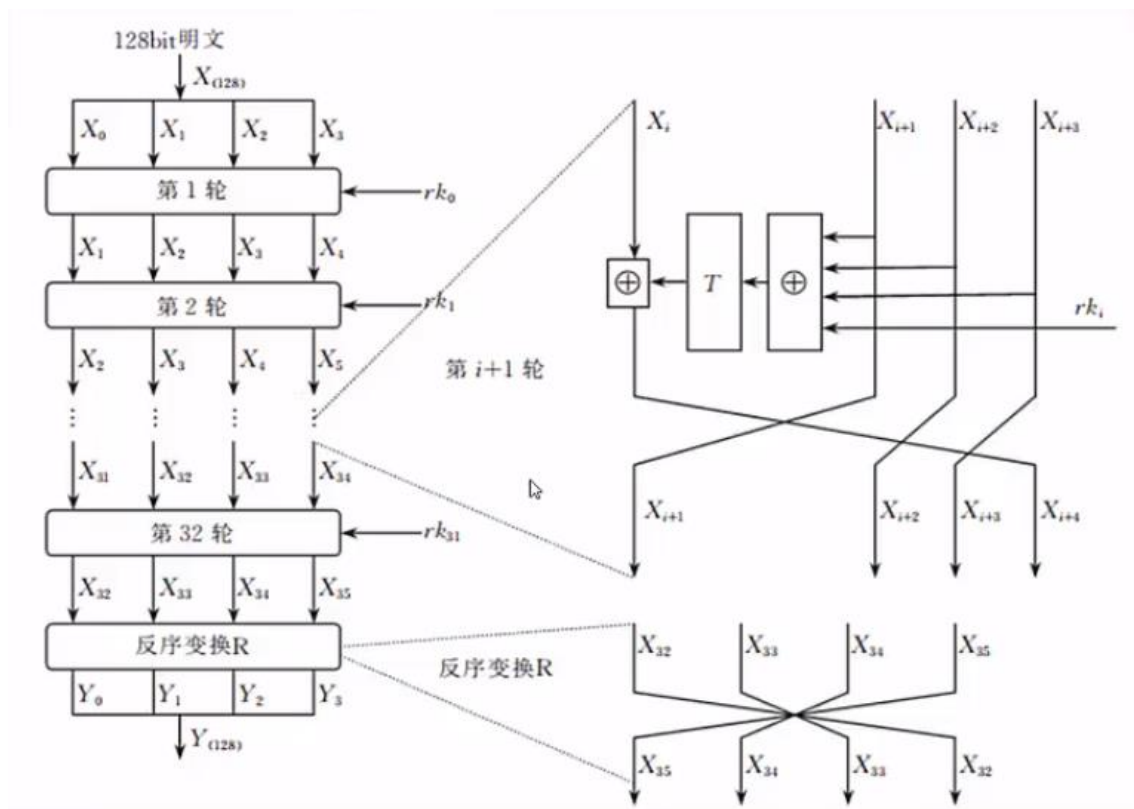


图 2.6 SM4 加密流程

2.2 核心代码实现

(1)加密函数

```
def encrypt(message, key):
    r = ''
    set_Ck()
    RoundKey(key)
    m = message
    if m == '0x0123456789abcdefdcba9876543210':
        r += f(intTobit(int(m, 16), 128))
        print(hex(int(r, 2)))
    else:
        while (len(m) % 16 != 0):
            m += '0'
        for i in range(0, len(m), 16):
            r += f(m[i:i+16])
        return r
```

(2)解密函数

```
def decrypt(message, key):
    r = ''
    set_Ck()
    RoundKey(key)
    Roundk.reverse() #解密密钥是加密密钥的逆序
    m = message
    for i in range(0, len(m), 128):
        r += ff(m[i:i+128])
    r = binTostr(r)

    return r
```

(3)正确性验证

明文: Zhouzixin is a handsome girl.If you like her, she also like you.

密钥: chongyanisyydshh

模式: ECB



图 2.7 SM4 加解密验证

在在线解密网站解密发现结果相同，验证成功

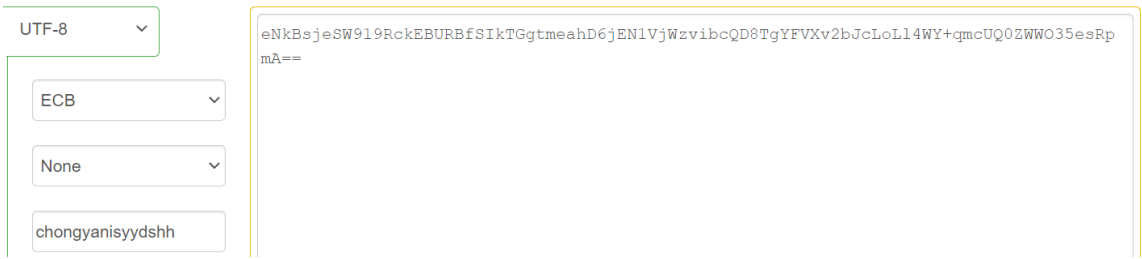


图 2.8 在线 SM4 加密结果

3. SM4 与 AES 的区别

- (1)结构上: AES 基于 SP 网络, SM4 基于 Feistel 网络。
- (2)密钥长度上: SM4 密钥长度只能是 128bit,而 AES 可以选择 128/192/256bit,从密钥长度上来看, AES 更能满足随着计算机能力增强的防止穷举等攻击手段的安全需求。

算法名称	密钥长度	分组长度	循环次数	算法结构
AES	128/192/256	128	10/12/14	Substitution-Permutation
SM4	128	128	32	非平衡Feistel

图 2.9 AES SM4 基本比较

(3)具体操作上：在安全性设计方面，AES 每轮的操作包括使用 S 盒完成分组的字节到字节的代替，简单的行移位，列混淆，密钥轮加（异或）。而 SM4 每轮的操作包括将 32 位明文组与轮密钥异或，基于 S 盒的字节到字节的代替，基于移位的线性变换。两个算法的安全性均是基于 S 盒的非线性性以及线性变换提供的扩散作用。密钥的使用方式也均是将密钥与明文或加密结果异或。一点小区别在于，AES 算法在每轮的最后使用密钥，而 SM4 算法在每轮的开始使用密钥。

4. 实验总结

分组密码是现代密码学的重要体制之一，可以说时现代密码中最常用、应用最多的密码技术之一，而高级加密标准（AES）则是现在密码学比较安全的典型的分组密码。在本次实验中对 AES 进行了实现与分析，在代码编写过程中对 AES 的实现步骤有了更清晰的认知，也对 AES 通过对密钥列混淆使加解密结构相同有了更清楚的了解，并以此对分组密码中很常见的 SP 网络结构有了更深入的理解。但在编写过程中许多格式转换的问题依然处理的不是很好，由于时间问题没有对输入格式的转化进行优化，但在后续实验中找到了更好的处理方法。

除此之外，由于如 AES 的分组密码自身并不是加密的实用手段，而必须以某种工作模式进行实际操作，所以我对 ECB、CBC 两种种常见的工作模式进行了实现，密码学课程设计报告通过对工作模式的深入学习，更加了解了 AES 及工作模式在实际场景下的应用与选择。同时，还复现了比 DES 安全性更高的国密算法 SM4, 熟悉了 SM4 的实现步骤，对 Feistel 网络也有更深刻的理解。

三.序列密码

1. 线性反馈移位寄存器

1.1 算法原理

LFSR 是属于 FSR（反馈移位寄存器）的一种，除了 LFSR 之外，还包括 NFSR（非线性反馈移位寄存器）。FSR 是流密码产生密钥流的一个重要组成部分，在 $GF(2)$ 上的一个 n 级 FSR 通常由 n 个二元存储器和一个反馈函数组成，如下图所示：

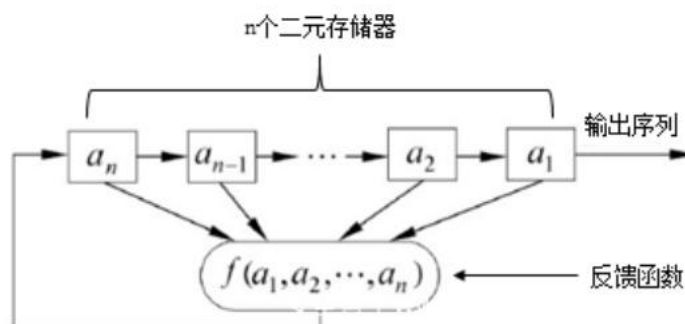


图 3.1 LFSR 示意图

如果这里的反馈函数是线性的，我们则将其称为 LFSR，此时该反馈函数可以表示为：

$$f(a_1, a_2, \dots, a_n) = c_1 a_1 \oplus c_2 a_2 \oplus \dots \oplus c_n a_n$$

1.2 核心代码实现

(1) 线性移位寄存器函数

```
def LFSR(key, start):
    s_k=key #可约多项式序列 确定反馈函数参与异或位数

    for i in range(len(s_k)):
        if s_k[i]=='1':
            bitxor.append(i)

    s_0=start #初始序列 右端为输出
    s_len=len(s_0)
    s_i=s_0 #现在寄存器状态

    output=[] #存放每次是输出

    S_history=[] #存放每一次在寄存器中的序列，用于判断周期结束
    S_history.append(s_0)
    T=0 #内循环周期数
    T_Max=0 #最大周期数
    Sb=0 #哨兵，重复的标志

    while True:
        tmp=str(FeedBack(s_i))
        output.append(s_i[-1:])
        s_i=tmp+s_i[:-1]

        if s_i==s_0:
            #print("周期结束")
            T_Max=len(output)
            T=T_Max
            Sb=1

        for i in range(len(S_history)-1): #如果当前序列与历史状态的序列有相同，表示周期结束，停止循环
            if s_i==S_history[i+1]:
                Sb=1
                #print("构成内循环，周期结束"+str(i)+'-'+S_history[i+1])
                T=len(S_history)-(i+1)
                T_Max=len(output)
                break

        if Sb!=0:
            break

        S_history.append(s_i)

    return output, S_history, T_Max, T
```

(2)特点

①区分了内循环周期结束和与初始值相同时的周期结束，能更好的验证多项式的周期性

②通过 history 存放的每一次的寄存器状态，output 存放输出，能够查看每一次循环的详细过程

1.3 LFSR 周期性正确性验证

在网站 https://wims.univ-cotedazur.fr/wims/cn_tool~algebra~primpoly.cn.html 中查找 16 次的本原多项式，得到如图所示的一些多项式，取第一个为我们验证的多项式，写成逆序的多项式序列 0110111101000111,作为输入，初始状态可以随意选择，这里选择 0101010101010101。

在 \mathbb{F}_2 上总共有 2048 个 16 次本原多项式.

搜索结果: [往后>>](#)

1	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^7+x^6+x^4+x^3+x^2+1$
2	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^7+x^6+x^5+x^2+x+1$
3	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^7+x^6+x^5+x^4+x+1$
4	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^7+x^6+x^5+x^4+x^2+1$
5	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^7+x^6+x^5+x^4+x^3+1$
6	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^5+x^3+x^2+x+1$
7	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^6+x^3+1$
8	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^6+x^5+x^2+x+1$
9	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^7+x+1$
10	$x^{16}+x^{15}+x^{14}+x^{12}+x^{10}+x^8+x^7+x^3+1$

图 3.2 16 次本原多项式

运行结果: (周期序列和详细过程过长截图略)

```
请输入对应反馈函数的可约多项式序列逆序(如10100100101010):
0110111101000111
请输入寄存器初始化序列(如01010101010101):
01010101010101
周期序列为:
1010101010101000110010000100100011110101010110100000100100010100001111100000111011001101011001110001000000000010001101000010001010001
-----
最大周期数为: 65535    内循环周期数为: 65535
是否查看详细过程 y or n
```

图 3.3 周期性验证

运行得到的周期数为 65535，16 次最大的循环周期为 $2^{16}-1=65535$,验证成功。

2. 祖冲之序列密码

2.1 算法原理

ZUC 算法由线性反馈移位寄存器 LFSR、比特重组 BR、非线性函数 F 三个基本部分组成，成功结合了模 $(2^{31}-1)$ 素域、模 2^{32} 域以及模 2 高维向量空间这三种不同代数范畴的运算，采用了线性驱动加有限状态自动机的经典流密码构造模型。公开文献表明，该算法具有很高的理论安全性，能够有效抵抗目前已知的攻击方法，具有较高的安全冗余。祖冲之算法的运行分为两个阶段：初始化阶段和工作阶段。首先进行初始化阶段。这一阶段将初始密钥和初始向量装入 LFSR 作为其初态，并置 R1 和 R2 为 0，然后进行一系列操作。然后进入工作阶段。工作阶段产生密钥流。

算法整体结构分为上中下三层。上层为线性反馈移位寄存器 LFSR，中层为比特重组 BR，下层为非线性函数 F。如下图：

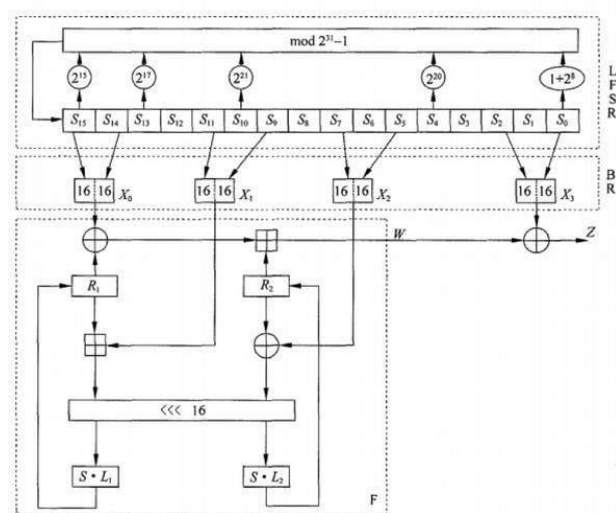


图 3.4 ZUC 算法结构示意图

2.2 核心代码实现

(1)初始化函数

```
def Init():
    global W,S,R1,R2
    print("初始化阶段线性反馈移位寄存器的初态:")
    for i in range(16):
        S[i] = IV[i] / (D[i]<<8) / (KEY[i]<<23)
        print('S'+str(i)+':'+hex(S[i]),end=' ')
    print()
    for i in range(32):
        BitReconstruction()
        F(X[0],X[1],X[2])
        LFSRWithInitialisationMode(W >> 1)
        for j in range(4):
            print('X'+str(j)+':'+hex(X[j]),end=' ')
        print('R1'+':'+hex(R1),end=' ')
        print('R2'+':'+hex(R2),end=' ')
        print('W'+':'+hex(W),end=' ')
        print('S15'+':'+hex(S[15]),end=' ')
        print()
    print("初始化后线性反馈移位寄存器的状态:")
    for i in range(16):
        print('S'+str(i)+':'+hex(S[i]),end=' ')
```

(2) 密钥流生成函数

```
def Keys_Print():
    Keys = []
    print("产生的密钥流:")
    for i in range(3):
        BitReconstruction()
        F(X[0],X[1],X[2])
        Keys.append(W^X[3])
        LFSRWithWorkMode()
        for j in range(4):
            print('X'+str(j)+':'+hex(X[j]),end=' ')
        print('R1'+':'+hex(R1),end=' ')
        print('R2'+':'+hex(R2),end=' ')
        print('W'+':'+hex(W),end=' ')
        print('S15'+':'+hex(S[15]),end=' ')
        print('Z'+str(i)+':'+hex(Keys[i]))
```

2.3 ZUC 安全性分析

课堂上没有对 ZUC 算法进行安全性分析，但经过学习发现它很好的进行了扩散混淆，也有很好的冗余度。具体体现在：

- ①在 LFSR 设计上：ZUC 算法的 LFSR 设计首次采用素域 $GF(231 - 1)$ 的 m 序列。该类序列周期长、统计特性好。此外，由于素域 $GF(231 - 1)$ 上的乘法可以快速实现，LFSR 在设计时充分考虑到安全和效率两方面的问题，在达到高安全目标的同时可以拥有非常高效地软硬件实现。
- ②在 BR 设计上：在比特重组部分，精心选用数据使得重组的数据具有良好的随机性，并且出现重复的概率足够小。
- ③在 F 函数设计上：在非线性函数 F 的设计上，F 函数采用了两个非线性变换 S 盒 S0 和 S1，从而为祖冲之密码提供了非线性。又由于 LFSR 和 BR 都是线性变换，所以非线性函数 F 就成为 ZUC 算法中唯一的非线性部件，从而成为确保 ZUC 算法安全的关键。其中，S 盒为密码提供混淆作用，L 为密码提供扩散作用，二者互相配合提高密码安全性。

3. 实验总结

流密码（又称序列密码）是一类重要的对称密码体制，具有算法实现简单、速度快、传播 错误少的特点，其简单的加解密算法需要对密钥流生成器有较高的要求。本次实验中，首先实现了 LFSR 来深入了解伪随机密钥流生成器的原理，并对其周期性进行了一定的验证。还实现了 ZUC 祖冲之序列密码，进一步了解了流密码的线性结构和非线性结构。

四.HASH 函数

1. MD5

1.1 算法原理

MD5 消息摘要算法（MD5），一种被广泛使用的密码散列函数，可以产生出一个 128 位（16 字节）的散列值，用于确保信息传输完整一致。

MD5 算法输入不定长度信息，输出固定长度 128-bits 的算法。经过程序流程，生成四个 32 位数据，最后联合起来成为一个 128-bits 散列。基本方式有求余、取余、调整长度、与链接变量进行循环运算，得出结果。基本流程如下图所示：

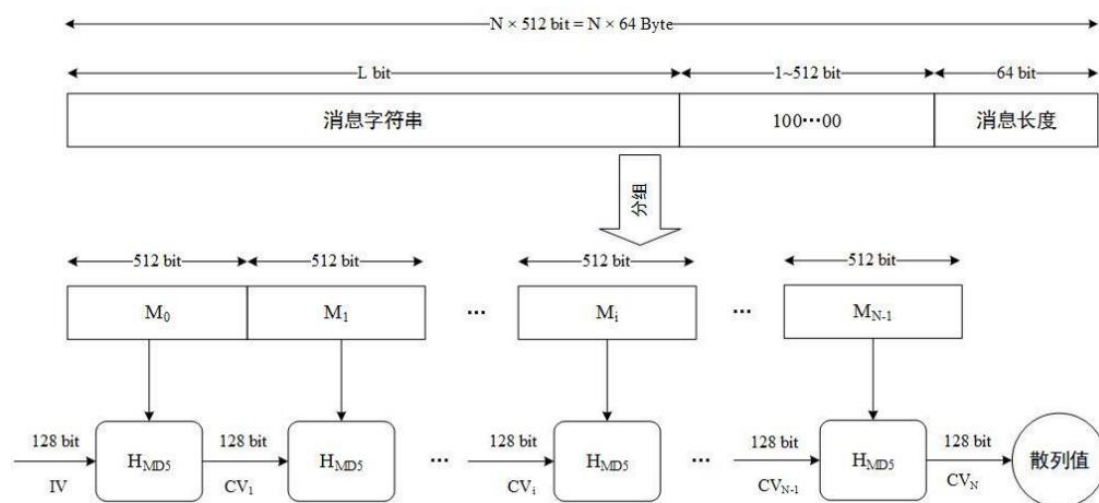


图 4.1 md5 算法结构示意图

1.2 代码实现

(1) MD5 运算流程

```
def md5(m):
    # 转为2进制
    m = str2bin(m)
    # 消息填充
    m = message_padding(m)
    # 对消息分组, 每组512位
    m_list_512 = re.findall(r'.{512}', m)
    # 初始链接变量
    A, B, C, D = IV_A, IV_B, IV_C, IV_D
    # 对每512bit进行分组处理, 前一组的4个输出连接变量作为下一组的4个输入链接变量
    for i in m_list_512:
        A, B, C, D = compress_func(A, B, C, D, i)
    # 把最后一次的分组4个输出连接变量再做一次大端小端转换
    A = hex2little(hex(A)[2:]).zfill(8)
    B = hex2little(hex(B)[2:]).zfill(8)
    C = hex2little(hex(C)[2:]).zfill(8)
    D = hex2little(hex(D)[2:]).zfill(8)
    # 拼接到一起的得到最终的md5值
    return A + B + C + D
```

(2)注意：在编写 md5 算法的过程种，遇到的最大的麻烦就是关于大小端序的转换问题，初始链接遍历应该是小端序输入的，消息的填充也应该以小端序方式，并且在最后运算结束后还要再进行一次大小端序的转换。

1.3 正确性验证

需要加密的内容：Zhouzixin is a handsome girl.If you like her, she also like you.

请输入要进行md5的内容：

Zhouzixin is a handsome girl.If you like her, she also like you.

md5后的散列值为：

b6aac2204ba128f4fcc0af888c62b6da

图 4.2 md5 加密结果

在线网站加密，对比结果相同，验证成功。



MD5在线加密

要加密的字符串:

字符串	Zhouzixin is a handsome girl.If you like her, she also like you.
16位 小写	4ba128f4fcc0af88
16位 大写	4BA128F4FCC0AF88
32位 小写	b6aac2204ba128f4fcc0af888c62b6da
32位 大写	B6AAC2204BA128F4FCC0AF888C62B6DA

图 4.3 在线 md5 加密结果

2. SM3

2.1 算法原理

对长度为 $l(l < 264)$ 比特的消息 m ，SM3 杂凑算法经过填充和迭代压缩，生成杂凑值，杂凑值长度为 256 比特。

具体过程如下图：

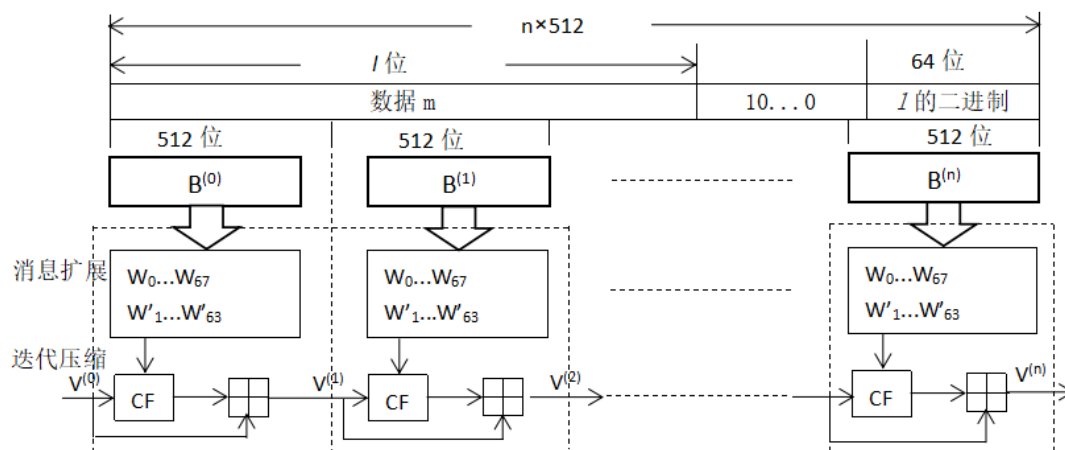


图 4.4 SM3 算法结构示意图

2.2 核心代码实现

SM3 加密主函数

```
def sm3(msg):
    #sm3 加密主函数
    # 字符串转化为比特串
    s_bin = str2bin(msg)
    # 对消息进行填充
    s_fill = msg_fill(s_bin)
    # 对填充后的消息进行迭代压缩
    s_sm3 = iteration_func(s_fill)

    return s_sm3.upper()
```

2.3 正确性验证

需要加密的内容：Zhouzixin is a handsome girl.If you like her, she also like you.

输入需要加密的文本：

Zhouzixin is a handsome girl.If you like her, she also like you.

SM3杂凑值为0A590D36063F285BF64AB15F4148C5846D7BB54CC5B6ECB0B8E71A9AAB54BD7A

Repl Closed

图 4.5 SM3 加密结果

结果与在线加密结果相同


```

==== SM3 Hash Result ====
Hex: 0A590D36063F285BF64AB15F4148C5846D7BB54CC5B6ECB0B8E71A9AAB54BD7A
Base64: C1kNNgY/KFv2SrFfQUjFhG1/tUzFtuywuOcamqtUvXo=
Base32: BJMQ2NQGH4UFX5SKWFPUCSGFQRWXXNKMYY3OZMFY44NJVK2UXV5A====
Array:
0x0A, 0x59, 0x0D, 0x36, 0x06, 0x3F, 0x28, 0x5B, 0xF6, 0x4A, 0xB1, 0x5F, 0x41, 0x48, 0xC5, 0x84,
0x6D, 0x7B, 0xB5, 0x4C, 0xC5, 0xB6, 0xEC, 0xB0, 0xB8, 0xE7, 0x1A, 0x9A, 0xAB, 0x54, 0xBD, 0x7A

```

图 4.5 在线 SM3 加密结果

3. SM3 与 MD5, SHA 算法的区别

(1)算法结构上：在消息填充方面，几个 hash 算法基本相同，都是先在原始消息的最后加一位“1”，再添加 k 个“0”，最终要使 $1+1+k$ 除以 512 后的余数为 448，取其最小的非负整数。然后用一个 64 位的比特串标识原始消息的长度 l。填充后的消息 M 正好是 512 位的倍数。SM3 算法的压缩函数与 SHA-256 的压缩函数具有相似的结构,但是 SM3 算法的设计更加复杂,比如压缩函数的每一轮都使用 2 个消息字。

(2)安全性上：2005 年,王小云等人给出了 MD5 算法和 SHA-1 算法的碰撞攻击方法，现今被广泛应用的 MD5 算法和 SHA-1 算法不再是安全的算法。

SM3 密码摘要算法是中国国家密码管理局 2010 年公布的中国商用密码杂凑算法标准。SM3 算法适用于商用密码应用中的数字签名和验证，是在 SHA-256 基础上改进实现的一种算法。SM3 算法采用 Merkle-Damgard 结构，消息分组长度为 512 位，摘要值长度为 256 位。现今为止，SM3 算法的安全性相对较高。

4. 实验总结

本次实验实现了 MD5 和国密 SM3 算法，不仅掌握了 hash 函数的实现流程和思路，同时也让我对字节序方面的有了更多的理解。

五.公钥密码

1. RSA

1.1 算法原理

RSA 是 1977 年由 Ron Rivest、Adi Shamir 和 Leonard Adleman 共同提出，其安全性基于大素数因子分解的困难问题，整体算法流程简单清晰，分为如下三个部分：

(1) 密钥生成算法:

① 选取两个保密的大素数 p 和 q , 满足 $p \neq q$, 计算 $n = p \times q$, $\varphi(n) = (p - 1)(q - 1)$, $\varphi(n)$ 为 n 的欧拉函数。

② 随机选取整数 e , 满足 $1 < e < \varphi(n)$ 且 $\gcd(e, \varphi(n)) = 1$, 即 e 与 $\varphi(n)$ 互素。

③ 计算 d , 满足 $ed \equiv 1 \pmod{\varphi(n)}$, 则公钥为 (e, n) , 私钥为 d 。

(2) 加密 对明文进行比特串分组, 使每个分组十进制小于 n , 然后对每个分组 m ($0 \leq m < n$), 计算 $c = m^e \pmod{n}$, 则得到密文 c 。

(3) 解密 对于密文 c ($0 \leq c < n$), 计算 $m = c^d \pmod{n}$, 得到对应明文 m 。

1.2 核心代码实现

RSA 的加密解密函数非常简单, 其中保证算法安全关键的是密钥的生成。

(1) Miller-Rabin 素性检验函数

除了密钥生成以及加密解密算法之外, RSA 加密体制还有一个重要的问题, 那就是大素数的生成, 这就要求了还需要掌握素性检验的算法。在本实验中, 我使用了 Miller-Rabin 素性检验结合随机数的生成来得到所需要的大素数, 它是一个基于概率的算法, 是费马小定理的一个改进。简单来说, 要测试 n 是否为素数, 首先将 $n-1$ 分解为 $2^s d$ 。在每次测试开始时, 先随机选一个介于 $[1, n-1]$ 的整数 a , 之后如果对所有的 $r \in [0, s-1]$, 若 $a^d \not\equiv 1 \pmod{n}$ 且 $a^{2^r d} \not\equiv -1 \pmod{n}$, 则 n 是合数。否则, n 有 3/4 的概率为素数, 随着增加测试的次数, 是素数的概率会越来越高。

```
def primality_testing(n, k):
    # 用miller_rabin算法对n进行k次检测
    if n%2==0:
        return False
    if n<2:
        return False
    d=n-1
    r=0
    while not (d&1): #d为偶数进入循环==>最后得到2^r*d
        r+=1
        d>>=1 #d除以2

    for _ in range(k):
        a=randint(1,n-1)
        x=pow(a,d,n)
        if x==1 or x==n-1:
            continue
        flag=0
        for _ in range(r-1):
            x=pow(x,2,n)
            if x==n-1:
                flag=1
                break
        if flag==0:
            return False

    return True
```

(2) 密钥获取函数

Hastad 证明如果采用不同的模 n ，相同的公钥 e ，则对 $e(e+1)/2$ 个线性相关的消息加密，系统就会受到威胁。一般选取 16 位以上素数(速度快且可防止攻击)。所以设置 e 时选择 16 位以上的素数

```
def GetKey(n):
    # 获取位数为n的p,q 并选取合适的e, 求出d
    while True:
        p=GetPrime(n)
        q=GetPrime(n)
        if p==q: # 取不相等的素数p,q
            continue
        N=p*q
        phi=(p-1)*(q-1)
        e=randint(65536,100000)#一般选取16位以上素数=>防止低指数攻击
        if gcd(e,phi)==1:
            d=ModInverse(e,phi)
            return e,N,d
```

(3)特点

没有使用快速幂自己编写 $\text{pow}()$ 函数，而是直接使用自带的 $\text{pow}()$ 函数，因为通过测试，快速幂写的 $\text{pow}()$ 函数并没有自带的快。

1.3 RSA-OAEP 填充模式

RSA 中各项参数应选得足够大来避免穷举攻击，所以密文长度不能太短，如果太短需要进行填充。OAEP 就是是最优非对称加密填充函数，它不仅能扩充密文长度，也能实现加盐的功能，使对同一密文同一公钥加密的结果不一样，使之不具有同态性，避免了一些构造攻击。

OAEP 主要分为三个步骤：

(1)长度检查：密文长度过长不需要填充，因为 PS 最小长度为 0，故要满足

$$k - 2 * hlen - 2 * l - mlen \geq 0$$

即

$$mlen \leq k - 2 * hlen - 2 * l$$

(2)EME-OAEP 编码：

主要有三个步骤：

①构造数据块 DB；

②构造随机字节串 seed；

③基于 DB 和 seed 通过 MGF 函数分别生成 maskedDB 和 maskedSeed，用于构造 EM。

编码步骤如下图所示：

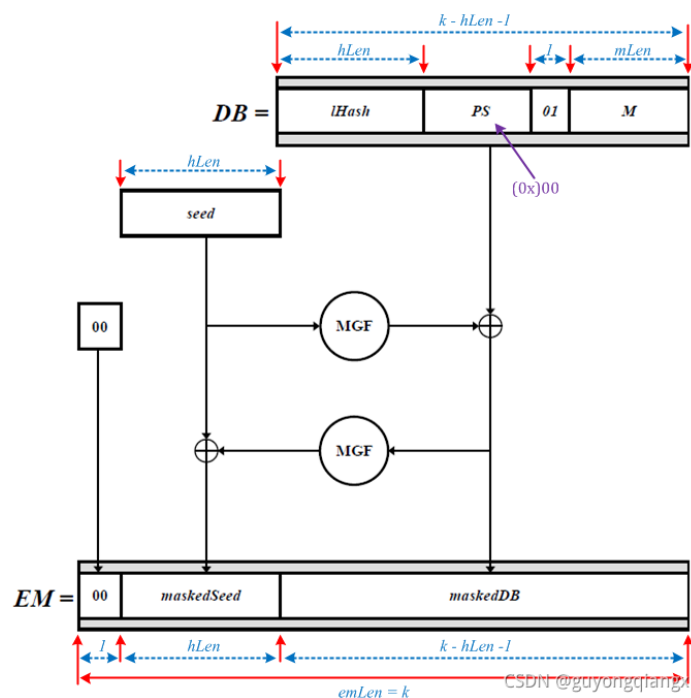


图 5.1 EME 编码示意图

(3)RSA 加密

1.4 OAEP 模式核心代码实现

开始写的是以 16 进制字符串为密文格式和 EM 格式，但在加解密过程中以 8bit（即 1 字节）进行的异或运算，而且如果只以整型数字形式 EM 前面的 00 容易丢失，所以以字节单位作为加解密中及加解密结果。

(1)RSA-OAEP 加密函数

```
def RSAOAPEnc(M: bytes, publicKey: Key):
    hLen = 20
    k = get_key_len(publicKey)
    mLen = len(M)

    # 长度检查
    assert mLen <= k - hLen - 2

    lHash = sha1(b'')
    hLen = len(lHash)

    # EME-OAEP 编码
    ps = b'\x00' * (k - mLen - 2 * hLen - 2)

    DB = lHash + ps + b'\x01' + M

    seed = os.urandom(hLen)
    dbMask = MGF1(seed, k - hLen - 1)
    maskedDB = xor(DB, dbMask)
    seedMask = MGF1(maskedDB, hLen)
    maskedSeed = xor(seed, seedMask)
    EM = b'\x00' + maskedSeed + maskedDB

    # RSA 加密
    m = os2ip(EM)
    c = RSAEP(publicKey, m)
    C = i2osp(c, k)

    return C
```

(2) RSA-OAEP 解密函数

```
def RSAOAEPDec(privateKey: Key, C: bytes):
    lHash = sha1(b'')

    # 使用 sha1 时的默认长度
    hLen = 20

    k = get_key_len(privateKey)
    assert len(C) == k

    c = os2ip(C)
    d, n = privateKey
    m = mypower(c, d, n)
    # print("m = " + str(m))
    EM = i2osp(m, k)

    #EME-OAEP 解码

    _, maskedSeed, maskedDB = EM[:1], EM[1:1 + hLen], EM[1 + hLen:]
    seedMask = MGF1(maskedDB, hLen)
    seed = xor(maskedSeed, seedMask)
    dbMask = MGF1(seed, k - hLen - 1)
    DB = xor(maskedDB, dbMask)

    _lHash = DB[:hLen]

    assert lHash == _lHash
    i = hLen
    while i < len(DB):
        if DB[i] == 0:
            i += 1
            continue
        elif DB[i] == 1:
            i += 1
            break
        else:
            raise Exception()
    M = DB[i:]
    return M
```

(3)特点

- ①在编写 OAEP 模式的时候，参考了 RSA 中 PSCK#1 V2.0 的部分，在阅读源码的时候学到了在输入参数的时候标注传入的类型的方式，在编写的时候更清晰方便(因为总是有各种的格式转换)
- ②以元组的方式，将公私钥 e(d),n 两个参数放在一起，格式更清晰，也更贴合 rsa 模块中的密钥格式。

1.5 正确性验证

待加密文本:

Zhouzixin is a handsome girl.If you like her, she also like you.

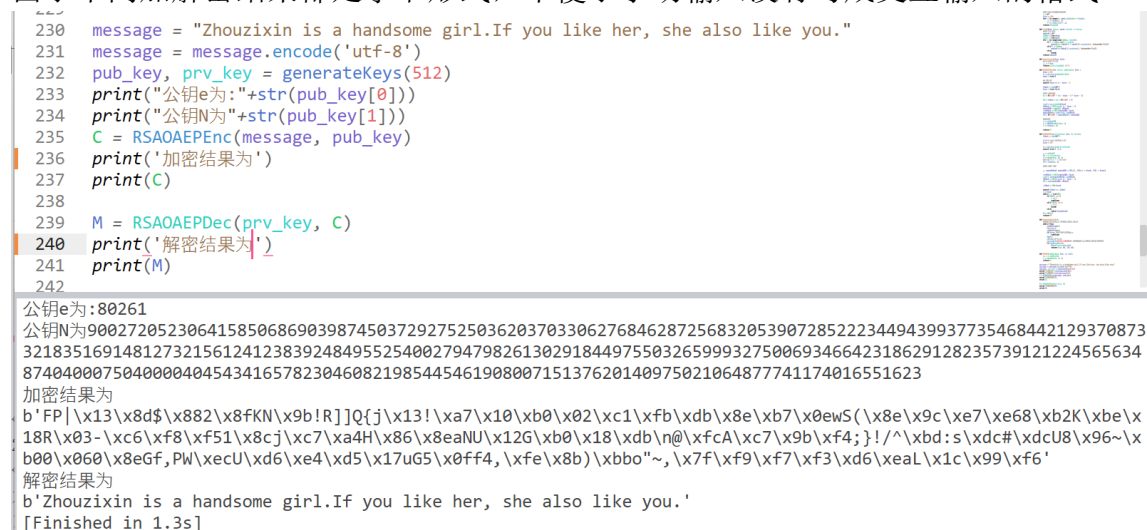
(1)不使用 oaep 填充模式

```
加密输入e,解密输入d,退出输入0
e
输入明文:
Zhouzixin is a handsome girl.If you like her, she also like you.
输入密钥位数:
512
公钥e为:79679
公钥n为:102820546526222320834160820222433381828303085811842472535313328965961174741386215236972767546706730120061509804827325982841700296023628352779227638604763104
86606212928948836386016746595166080983041235350896701163199131567275189832342540295006777534875527644958415535772867079223736536995869515725732968626197
私钥d为:1151196796596881656780525442990540568944174980675892857275760264022557968833432450975669129793819041328464053134309488710065488816479472450098958290640330550
63776790855261714297126666244081472295041102217364380667647439471803770444950854276717568240091357222987938744931099349418677811293141185499750556011379
密文为(10进制):9563861177275033015227746549341894828189701055135171693969662850644259515836416031991128254516356197928289301314668774968979644785605721563548184750240584114360519
40584114360519929589043429948472000206985705186529782238759722724819109310035640460197893664368063786239609633053417086652834011981137162405183411458354849108
加密输入e,解密输入d,退出输入0
d
输入密文(10进制):
9563861177275033015227746549341894828189701055135171693969662850644259515836416031991128254516356197928289301314668774968979644785605721563548184750240584114360519
929589043429948472000206985705186529782238759722724819109310035640460197893664368063786239609633053417086652834011981137162405183411458354849108
输入私钥d:
11511967965968816567805254429905405689441749806758928572757602640225579688334324509756691297938190413284640531343094887100654888164794724500989582906403305506377678
065261714297126666244081472295041102217364380667647439471803770444950854276717568240091357222987938744931099349418677811293141185499750556011379
输入公钥n:
1028205465262223208341608202224333818283030858118424725353133289659611747413862152369727675467067301200615098048273259828417002960236283527792276386047631048660621
2928948836386016746595166080983041235350896701163199131567275189832342540295006777534875527644958415535772867079223736536995869515725732968626197
密文为:b'Zhouzixin is a handsome girl.If you like her, she also like you.'
加密输入e,解密输入d,退出输入0
```

图 5.2 RSA 加解密结果

(2)使用 oaep 填充模式

由于中间加解密结果都是字节形式，不便于手动输入没有写成交互输入的模式



```

230 message = "Zhouzixin is a handsome girl.If you like her, she also like you."
231 message = message.encode('utf-8')
232 pub_key, prv_key = generateKeys(512)
233 print("公钥e为:"+str(pub_key[0]))
234 print("公钥N为:"+str(pub_key[1]))
235 C = RSAOAPEnc(message, pub_key)
236 print('加密结果为')
237 print(C)
238
239 M = RSAOAEPDec(prv_key, C)
240 print('解密结果为')
241 print(M)
242
公钥e为:80261
公钥N为9002720523064158506869039874503729275250362037033062768462872568320539072852223449439937735468442129370873
3218351691481273215612412383924849552540027947982613029184497550326599932750069346642318629128235739121224565634
874040007504000040454341657823046082198544546190800715137620140975021064877741174016551623
加密结果为
b'FP\x13\x8d$\x882\x8fKN\x9b!R]]Q{j\x13!\xa7\x10\xb0\x02\xc1\xfb\xdb\x8e\xb7\x0ewS(\x8e\x9c\xe7\xe68\xb2K\xbe\x
18R\x03-\xc6\xf8\xf51\x8c j\xc7\xa4H\x86\x8eaNU\x12G\xb0\x18\xdb\n@\xfca\x7\x9b\xf4; }!/^xbd:s\xdc#\xdcU8\x96~\x
b00\x060\x8eGf,PW\xecU\xd6\xe4\xd5\x17uG5\x0ff4,\xfe\x8b)\xbbo"~, \x7f\xf9\xf7\xf3\xd6\xeaL\x1c\x99\xf6'
解密结果为
b'Zhouzixin is a handsome girl.If you like her, she also like you.'
[Finished in 1.3s]

```

图 5.3 RSA-OAEP 加解密结果

2. 实验总结

本次实验过程中，用到了很多数论中的知识，例如快速幂的计算、费马小定理以 Miller-Rabin 素性检验，感受到了理论学到的数学知识用于解决现实问题的魅力。之后，还实现了最佳非对称填充模式 OAEP,为了实现该模式还进一步了解了 PSCK V2.0 的标准。整个实验下来，我对 RSA 公钥密码体制有了一个更全面、更深入的认识。

六.综合实验

1. 需求分析

应用情景：Alice 想通过公共信道给 Bob 传输一份秘密文件（文件非常大），又知道很多人和机构想得到这份文件。

分析：由于是传输的一份秘密文件，为了保证文件的机密性对文件进行 AES_CBC 模式的加密，但是在公共信道中传输，无法保证对称密钥传输的安全性，所以要对 AES 的密钥(加密密钥和初始向量)进行公钥加密。同时，为了保证文件传送的时候没有破损利用 md5 进行验证。

2. 功能流程图

Alice(发送端):

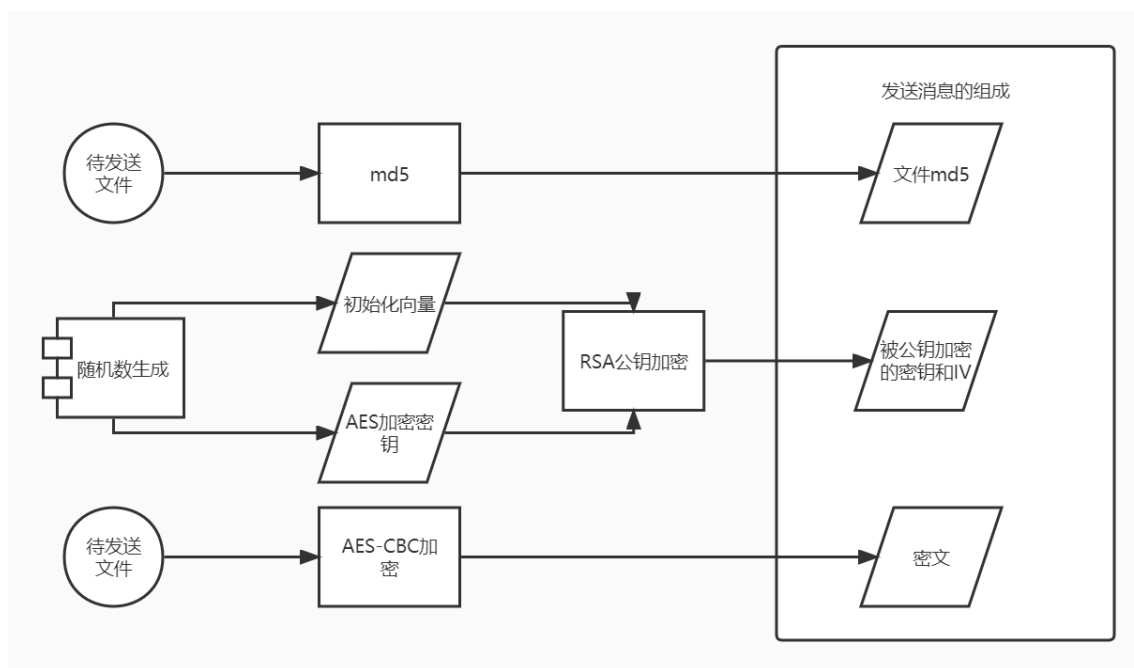


图 6.1 发送端流程示意图

Bob(接收端):

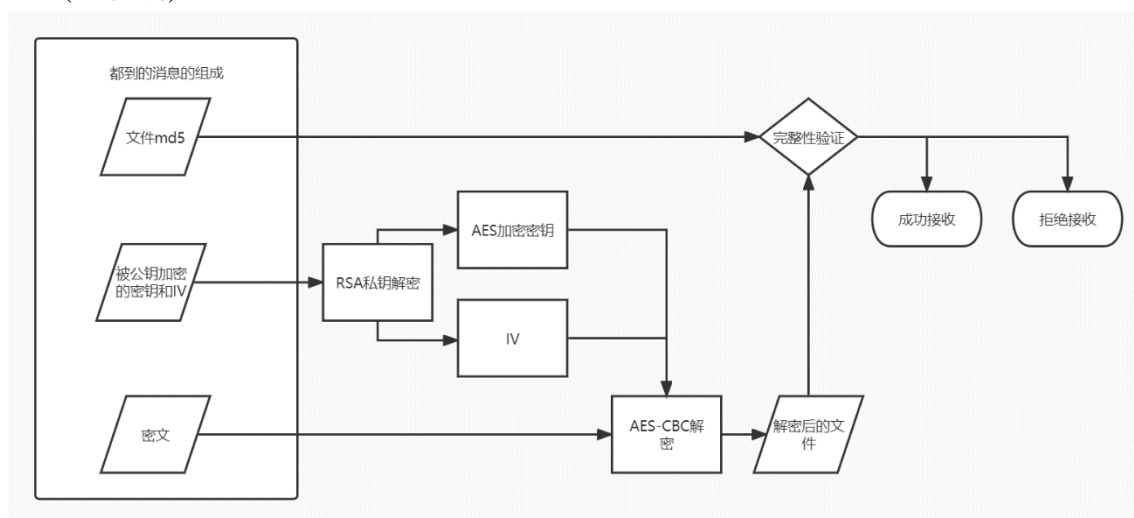


图 6.2 接收端流程示意图

3. 核心代码实现

在加密传输过程的两种思路，一种是服务端整个文件加密完成后再传输，客户端整个文件接收后再解密；一种是服务端对每个原文件分组加密传输，客户端对收到的每个原文件分组解密并写入。实验中，我一开始使用的是第一种方式，但是当文件较大时，由于是使用的自己编写的分组加密算法，加密效率较低、时间较长，会造成客户端长时间未收到响应而超时。于是实现了第二种方式，此方式每

个传输分组都使用了不同的 key 和 IV，安全性有所提高，且是以文件流的形式传输，即使文件很大也可以传输而不会超时，只不过时间会比较久。

(1)服务器端(发送方)文件加密发送函数

```
def get_file_stream(filename):
    print("\n*****")
    # 确定文件路径
    filepath = SERVER_WORKDIR + filename
    print("Send: " + filepath)

    if os.path.isfile(filepath):
        file_size = str(os.path.getsize(filepath))
        mainSocket.send(file_size.encode())
        while True:
            confirm = mainSocket.recv(10).decode()
            if confirm == "yes":
                break
        # 总共要传输的次数
        count = (int(file_size) // BUFSIZE) + 1

        # 获得原文件的md5
        file_digest = get_file_md5(filepath)
        send_size = 0
        f = open(filepath, 'rb')
        print("Start transferring...")
        # 循环发送文件
        while count:
            filedata = b64encode(f.read(BUFSIZE)).decode()
            key_encrypted, iv_encrypted, allfiles_encrypted = transfer_encrypt(filedata)
            # 发送aes-cbc加密后的内容(str)
            # print('allfiles_encrypted'+str(allfiles_encrypted))
            mainSocket.send(allfiles_encrypted.encode())
            while True:
                confirm = mainSocket.recv(10).decode()
                if confirm == "yes":
                    break
            # 发送RSA加密后的key(10进制)
            mainSocket.send(str(key_encrypted).encode())
            while True:
                confirm = mainSocket.recv(10).decode()
                if confirm == "yes":
                    break
            # 发送RSA加密后的iv
            mainSocket.send(str(iv_encrypted).encode())
            while True:
                confirm = mainSocket.recv(10).decode()
                if confirm == "yes":
                    break
            # time.sleep(0.2)
            # 计算已发送的大小
            send_size += BUFSIZE
            if send_size > int(file_size):
                send_size = file_size
            progress_bar(send_size, int(file_size), 10)
            count -= 1
        print("\nGet the file Successfully.")
        f.close()
        # 发送md5
        mainSocket.send(file_digest.encode())
        while True:
            confirm = mainSocket.recv(10).decode()
            if confirm == "yes":
                break
    else:
        print("[Error]: Can't find the file")
        mainSocket.send("0".encode())
    print("*****\n")
    return 0
```

(2)客户端(接收方)解密接收文件函数


```

def get_file_stream(filename):
    print("\n*****Ready to transfer the file*****")
    start = time.time()
    #与服务端建立TCP连接
    clientServer.settimeout(10000)
    filepath = CLIENT_WORKDIR + filename
    # 首先得到下载的文件大小(为了传输大文件,因此要分组接收)
    file_size = clientServer.recv(BUFFSIZE).decode()
    clientServer.send("yes".encode())
    file_size = int(file_size, 10)
    # 计算传输的次数
    count = (int(file_size) // BUFFSIZE_SENT) + 1
    if file_size > 0:
        f = open(filepath, 'wb')
        # 已接收的文件大小
        recv_size = 0
        print("Start transferring...")
        print("The file needs to be encrypted/decrypted, please wait...")
        # 循环接收文件
        while count:
            # 接收加密分组
            filedata_encrypted = clientServer.recv(BUFFSIZE).decode()
            clientServer.send("yes".encode())
            # 接收key
            key_encrypted = int(clientServer.recv(BUFFSIZE).decode())
            clientServer.send("yes".encode())
            # 接收iv
            iv_encrypted = int(clientServer.recv(BUFFSIZE).decode())
            clientServer.send("yes".encode())
            # 对分组进行解密
            filedata_decrypted = transfer_decrypt(filedata_encrypted, key_encrypted, iv_encrypted)
            f.write(b64decode(filedata_decrypted.encode()))
            # 更新下载进度
            recv_size += BUFFSIZE_SENT
            if recv_size > file_size:
                recv_size = file_size
            progress_bar(recv_size, file_size)
            count -= 1
        f.close()
        # 接收原文件md5值
        recv_digest = clientServer.recv(32).decode()
        clientServer.send("yes".encode())
        if recv_digest != get_file_md5(filepath):
            print("\n[Warning]: The file you received may be broken!")
        else:
            end = time.time()
            print("\nGet the file successfully!")
            print("Spend time: " + str(int(end - start)) + "s")
    print("*****\n")
    return 0

```

(3)特点:

在服务端与客户端的通信过程中,由于短时间内有多个消息要传输,所以可能造成客户端错误的接收到其他内容,从而造成后续的解密失败。对于这个问题找到的解决方法是客户端每收到一个消息,就回应一个“yes”,服务端确认收到后再发送下一个消息,这也虽然不会发生错误,但是明显是会减低通信的效率。

4. 实现效果

(1)查看可下载文件

客户端: 输入 list

```

Input one of the following command:
[+] list -----list the file
[+] get [filename] -----get the file by stream
[+] exit -----close socket and exit
[+] list

```

```

*****
The list of files:
study.pdf
xiyy.jpg
zhouzixin.txt
*****

```

服务器端：

(2)从服务端下载 xiyy.jpg(Alice 传图片文件给 Bob)

服务器端:

(3)从服务器下载大文件（Alice 传大文件给 Bob）

34



图 6.9 一个数据流示意图

6. 实验总结

通过本次实验，我将之前实验编写的算法 RSA,AES,MD5 结合再利用 socket 实现了文件的加密传输，并利用 wireshark 看到文件传输在信道中真实的数据格式，对密码学实际应用体系有了一点了解。但在最后对代码测试的过程中发现使用分组密码加密大文件的方式仍然比较慢，可能使用流密码效果会更好。本来想用 SM3 进行完整性检验，提高安全性，但实现中数据的检验总是有点问题，可能代码有点问题(比如因为填充多出来的字符没处理好)，最后还是使用了 MD5。