SHANDONG UNIVERSITY

密码工程第六次实验报告

姓名: 谢子洋
学院: 网络空间安全学院 (研究院)
专业: 网络空间安全
学号:202100460116

2023 年 11 月 29 日

目录

1 实验原理

1.1 RSA

RSA（Rivest-Shamir-Adleman）是一种公钥密码系统, 广泛用于安全的数据传输.

密钥生成: 选择大素数 p,q, 计算 $n=pq$, 选择 e, 计算 $d=e^{-1}mod\ \phi(n)$

加密: $c \leftarrow m^e\ mod\ n$

解密: $m \leftarrow c^d\ mod\ n$

1.2 CRT

中国剩余定理，又称孙子定理或中国余数定理，是数论中的一个关于一元线性同余方程组的定理，说明了一元线性同余方程组有解的准则以及求解方法。

给定同余方程组

$$\begin{cases} x \equiv a_1\ (mod\ m_1) \\ x \equiv a_2\ (mod\ m_2) \\ \cdots \\ x \equiv a_n\ (mod\ m_n) \end{cases} \tag{1.1}$$

中国剩余定理 (CRT) 说明, 假设整数 $m_1,m_2,\cdots,m_n$ 中任意两数互素, 则对任意整数 a_1,a_2,a_n, 方程组 (S) 有解, 通解如下构造.

设 $M=\prod_{i=1}^{n} m_i$. 计算 $M_i=M/m_i, \forall i \in \{1,2,\cdots,n\}$. 计算 $t_i=M_i^{-1}\ mod\ m_i$.

则方程组 (S) 的通解形式为:

$$x=\sum_{i=1}^{n} a_i t_i M_i + kM = a_1t_1M_1 + a_2t_2M_2 + \cdots + a_nt_nM_n + kM \tag{1.2}$$

1.3 CRT 加速 RSA 解密算法

1.3.1 算法

A. 密钥生成步骤.

使用 CRT 加速 RSA 解密需要在密钥生成阶段预计算一些额外参数.

$$dp = e^{-1}mod\,(p-1)$$

$$dq = e^{-1}mod\,(q-1)$$

$$qInv = q^{-1}mod\ p$$

密钥生成伪代码如下:

Algorithm 1 RSA Key Init

Input: p,q,e,d
Output: $dq,dp,qInv$
1: $dp \leftarrow e^{-1} mod\,(p-1)$
2: $dq \leftarrow e^{-1} mod\,(q-1)$
3: $qInv \leftarrow q^{-1} mod\ p$

B. RSA 加密步骤.

加密步骤与原算法相同.

C. RSA 解密步骤.

CRT 加速 RSA 解密将直接计算大整数的模幂转化成计算同余方程组, 大大减小了计算的时间复杂度.

$$\begin{aligned} m1 &= c^{dp} mod\ p \\ m2 &= c^{dq} mod\ q \\ h &= qInv \cdot ((m1-m2) mod\ p)\ mod\ p \\ m &= m2 + hq \end{aligned} \tag{1.3}$$

伪代码如下:

Algorithm 2 RSA Decry By CRT

Input: $c,dq,dp,q,p,qInv$
Output: m
1: $m1 \leftarrow c^d p\ mod\ p$
2: $m2 \leftarrow c^d q\ mod\ q$
3: $temp \leftarrow (m1-m2) mod\ p$
4: $h \leftarrow qInv * temp\ mod\ p$
5: $m \leftarrow m2 + h * q$

1.3.2 正确性证明

因为

$$m1 = m\ mod\ p = c^d\ mod\ p = c^{d\ mod \varphi(p)}\ mod\ p$$

$$m2 = m\ mod\ q = c^d\ mod\ q = c^{d\ mod \varphi(q)}\ mod\ q$$

因此有同余方程组

$$\begin{cases} m = m1\ mod\ p \\ m = m2\ mod\ q \end{cases}$$

利用中国剩余定理 CRT 计算该同余方程组, 得到结果:

$$\begin{aligned} m &= (q \cdot qInv \cdot m_1 + p \cdot pInv \cdot m_2)\ mod\ n \\ &= m_2 + q \cdot (qInv \cdot ((m_1 - m_2) mod\ p))\ mod p) \\ &= m \end{aligned}$$

2 实验过程及代码实现

利用 CRT 加速 RSA 解密需要在密钥生成步骤额外计算参数.

```
class RSA():
  def randomPara(self):
    q=number.getPrime(length)
    p=number.getPrime(length)
    self.__q=q
    self.__p=p
    self.__n=p*q
    self.__phi_n=(p-1)*(q-1)
    e=number.getPrime(length)
    while(gmpy2.gcd(e,self.__phi_n)!=1):e=number.getPrime(length)
    d=gmpy2.invert(e,self.__phi_n)
    self.__e=e
    self.__d=d
    self.__dp=gmpy2.invert(self.__e,(self.__p-1))
    self.__dq=gmpy2.invert(self.__e,(self.__q-1))
    self.__qInv=gmpy2.invert(self.__q,self.__p)
    self.__havePK=1
    self.__haveSK=1
```

消息 m 经过加密后得到密文 c, 加密过程与 RSA 原算法一致.

```
class RSA:
  def encry(self,Plaintext):
    if self.__havePK==0: return None
    m=Plaintext
    return gmpy2.powmod(m,self.__e,self.__n)
```

解密方收到密文 c 后可使用 CRT 对其进行加速解密.

```
class RSA:
  # CRT解密
  def decryptByCRT(self,c):
      if self.__haveSK==0 or c==None: return None
      m1=gmpy2.powmod(c,self.__dp,self.__p)
      m2=gmpy2.powmod(c,self.__dq,self.__q)
      temp=gmpy2.mod((m1-m2),self.__p)
      h=gmpy2.mod((self.__qInv*temp),self.__p)
      m=m2+h*self.__q
      return m
```

3 实验结果

本文使用 python 实现了 RSA 的参数生成、加密、原解密和 CRT 加速解密算法, 分别测试了 RSA 的原始解密算法和 CRT 加速解密的时间开销, 并计算了两者的效率比对比两者效率差距.

对得到的数据使用 matlab 软件处理并绘图, 得到结果如下:

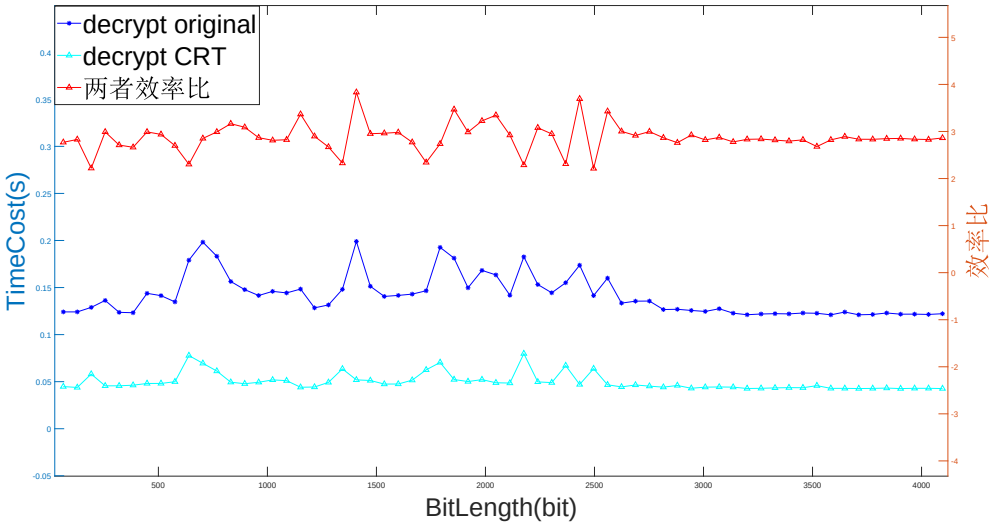

图 1: CRT decrypt benchmark

可观察到 CRT 加速 RSA 解密时间开销远小于原算法, 在效率上约为原算法的三倍. 由此可知,CRT 能大幅加速 RSA 的解密步骤, 作用显著.

参考文献

[1] https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[2] https://en.wikipedia.org/wiki/Chinese_remainder_theorem.

A Code

A.1 CRT RAS In Python

```
from Crypto.Util import number
import time
import random
length=4096
import gmpy2
def transToCode(plaintext):
    #print("start to trans to code")
    code=0
    for i in plaintext:
        temp=ord(i)
        code*=128
        code+=temp
    return code

def transFromCode(code):
    #print("start to trans from code")
    plaintext=""
    while(code!=0):
        temp=chr(gmpy2.mod(code,128))
        code=code>>7
        plaintext+=temp
    return plaintext[::-1]

class RSA():
    #p q 为不等素数
    def __init__(self):
        self.__q=0
        self.__p=0
        self.__n=0
        self.__phi_n=0
        self.__e=0
        self.__d=0
        self.__dp=0
        self.__dq=0
        self.__qInv=0
        self.__havePK=0
        self.__haveSK=0

    def randomPara(self):
        # q=random_prime(2**length-1,False,2**(length-1))
        # p=random_prime(2**length-1,False,2**(length-1))
        q=number.getPrime(length)
        p=number.getPrime(length)
        self.__q=q
        self.__p=p
        self.__n=p*q
        self.__phi_n=(p-1)*(q-1)
        e=number.getPrime(length)
        while(gmpy2.gcd(e,self.__phi_n)!=1):e=number.getPrime(length)
        d=gmpy2.invert(e,self.__phi_n)
        self.__e=e
        self.__d=d
```

```
53          self.__dp=gmpy2.invert(self.__e,(self.__p-1))
54          self.__dq=gmpy2.invert(self.__e,(self.__q-1))
55          self.__qInv=gmpy2.invert(self.__q,self.__p)
56          self.__havePK=1
57          self.__haveSK=1
58
59      def setPara(self,p,q,e,d):
60          self.__q=q
61          self.__p=p
62          self.__n=p*q
63          self.__phi_n=(p-1)*(q-1)
64          self.__e=e
65          self.__d=d
66          self.__dp=gmpy2.invert(self.__e,(self.__p-1))
67          self.__dq=gmpy2.invert(self.__e,(self.__q-1))
68          self.__qInv=gmpy2.invert(self.__q,self.__p)
69          self.__havePK=1
70          self.__haveSK=1
71
72      def receivePublicKey(self,set):
73          self.__n=set[0]
74          self.__e=set[1]
75          self.__havePK=1
76
77
78      #向外界发送公钥
79      def getPublicKey(self):
80          return [ self.__n , self.__e]
81      # 加密
82      def encrypt(self,Plaintext):
83          if self.__havePK==0: return None
84          m=Plaintext
85          return gmpy2.powmod(m,self.__e,self.__n)
86      # 直接解密
87      def decrypt(self,c):
88          if self.__haveSK==0 or c==None:
89              return None
90          m=gmpy2.powmod(c,self.__d,self.__n)
91          return m
92      # CRT解密
93      def decryptByCRT(self,c):
94          if self.__haveSK==0 or c==None:
95              return None
96          m1=gmpy2.powmod(c,self.__dp,self.__p)
97          m2=gmpy2.powmod(c,self.__dq,self.__q)
98          temp=gmpy2.mod((m1-m2),self.__p)
99          h=gmpy2.mod((self.__qInv*temp),self.__p)
100         m=m2+h*self.__q
101         return m
102
103 def test():
104     Alice=RSA()
105     Bob=RSA()
106     m=13789378936125037
107     # 1. 初始化参数
108     Alice.randomPara()
109     # 2. 公开公钥,加密
110     PK=Alice.getPublicKey()
111     Bob.receivePublicKey(PK)
```

```
    c=Bob.encrypt(m)
    print(f"cipher: {c}")
    # 3. 私钥解密
    m1=Alice.decrypt(c)
    m2=Alice.decryptByCRT(c)
    print(f"plain1:{m1}\nplain2(CRT): {m2}")
#test()

def benchmark():
    Alice=RSA()
    Alice.randomPara()
    loopTimes=30
    step=64
    for bitLen in range(step,length+step,step):
        m=random.randint(2**bitLen,2**(bitLen+1)-1)
        c=Alice.encrypt(m)
        st=time.time()
        for i in range(loopTimes):
            m1=Alice.decrypt(c)
        et=time.time()
        print(f"{bitLen} {(et-st)/loopTimes} ",end='')
        st=time.time()
        for i in range(loopTimes):
            m2=Alice.decryptByCRT(c)
        et=time.time()
        print(f" {(et-st)/loopTimes} ")

benchmark()
```