密码工程实验报告

实验名称 RSA-CRT

目录

1 实验目的

本实验的目的是使用中国剩余定理来加速 RSA 算法的实现.RSA 算法是重要的公钥密码学算法，它依赖于因子分解困难问题，中国剩余定理又称孙子定理，是数论中的一个求解一元线性方程组的定理，通过将 CRT 和 RSA 进行结合，能够减小 RSA 求解过程中模数计算的复杂度，提升 RSA 算法的实现速度.

2 实验原理

2.1 Chinese Reminder Theorem

中国剩余定理是计算如下的一元线性方程组

$$x = a_1 \bmod n_1$$
$$x = a_2 \bmod n_2$$
$$\cdots$$
$$x = a_k \bmod n_k$$

1. 首先第一步计算 $N = n_{12} \dots n_k$
2. 对每一个 $i = 1, 2, \dots, k$ 计算

$$y_i = \frac{N}{n_i} = n_1 n_2 \dots n_{i-1} n_{i+1} n_k$$

3. 对每一个 $i = 1, 2, \dots, k$ 计算 $z_i = y_i^{-1} \bmod n_i$, 这里可以使用扩展欧几里得算法进行求解，当然若 z_i 存在 n_i 之间应该两两互素.
4. 最后计算出解为 $x = \sum_{i=1}^{k} a_i y_i z_i$.

Garner's Algorithm 也是一种计算一元线性同余方程组的算法，它是通过将 a 转化为 a_i 之间的乘积进行解决，下面是 a 的表示形式，通过不断的进行模运算求解其中的系数 x_1, x_{2k}, 这里我们不再赘述.

$$a = x_1 + x_2 p_1 + x_3 p_1 p_2 + \dots + x_k p_1 p_{2k-1}$$

2.2 RSA

RSA 是一种公钥算法，由密钥生成，加密和解密三部分组成.

Key generation:

- 选择两个不同的素数 p 和 q
- 计算 $n = pq$, 计算 $\phi(n) = (p-1)(q-1)$
- 选择一个和 $\phi(n)$ 互素的数 e 作为公钥
- 计算私钥 d 是 e 关于 $\phi(n)$ 的逆元，换句话说 $de \equiv 1 \pmod{\phi}(n)$
- 公钥：e 和 n 私钥：d

Encryption:

- m 是一个信息，也就是明文
- 计算密文 $c \equiv m^e \pmod{n}$

Decryption:

计算 $m \equiv c^d \pmod{n}$

2.3 RSA-CRT

由于解密方是密钥生成方，因此解密方可以使用 p 和 q 来减小模数的规模，通过中国剩余定理来加速解密运算.

1. 使用 p 和 q 预计算下面的值，这里假设 $p > q$

- $d_P = e^{-1} \bmod (p-1) = d \bmod (p-1)$
- $d_Q = e^{-1} \bmod (q-1) = d \bmod (q-1)$
- $q_{Inv} = q^{-1} \bmod p$

2. 根据欧拉定理，$c^d \bmod p = c^{d \bmod \phi(p)} \bmod p = c^{d \bmod (p-1)} \bmod p$, 因此我们可以使用下面的方式进行解密.

- $m_1 = c^{d_P} \bmod p = m \bmod p$
- $m_2 = c^{d_Q} \bmod q = m \bmod q$

3. 根据 Garner's Algorithm, 可以解决上述的一元线性方程组将 m 写成 $m = x_1 + x_2 q$, 其中 $x_1 = m_2$.

- $x_2 = q_{Inv} \cdot (m_1 - m_2)$
- $m = m_2 + x_2 \cdot q$

3 算法实现

3.1 扩展欧几里得算法求逆元

在 RSA 计算过程中，需要求解逆元运算，这里我们采用扩展欧几里得算法求解逆元，扩展欧几里得算法就是在得到整数 a, b 的最大公因子后，还希望得到整数 x, y; 使得 $ax + by = gcd(a, b)$.

- 对于整数 $a > b$，显然 $b = 0$ 时，$gcd(a, 0) = a$; 此时 $x = 1, y = 0$.
- 设 $ax_1 + by_1 = gcd(a, b)$.
- 有 $bx_2 + (a\%b)y_2 = gcd(b, a\%b)$
- 由于 $gcd(a, b) = gcd(b, a\%b)$, 那么 $ax_1 + by_1 = bx_2 + (a\%b)y_2$
- 即 $ax_1 + by_1 = bx_2 + (a - \lfloor(a/b)\rfloor)y_2$
- 也就是 $ax_1 + by_1 == ay_2 + b(x_2 - \lfloor(a/b)\rfloor y_2)$, 这样就可以由 x_2, y_2 的值推算知道 x_1, y_1 的值
- 根据上式可以写出等式 $x_1 = y_2; y_1 = x2 - \lfloor(a/b)\rfloor y_2$
- 由于上式是递归的，最终根据欧几里得算法 $b = 0$, 也就是最终 $x = 1, y = 0$, 那么就可以回推到一开始的值

对于求逆元的情况，也就是 $ax = 1 \mod p$, 我们可以写成 $ax + kp = 1$, 这样就可以求出最后的 x 就是逆元. 下面是迭代形式的代码，当然我们也可以写成递归形式:

```
def inverse_mod(a, p):
old_s, s = 1, 0
old_t, t = 0, 1
old_r, r = a, p
if p == 0:
    return 1, 0, a
else:
    while r != 0:
        q = old_r // r
        old_r, r = r, old_r - q * r
        old_s, s = s, old_s - q * s
        old_t, t = t, old_t - q * t
return (old_s%p+p)%p
```

3.2 快速幂算法

在求解过程中，还需要用到很多的模幂运算，这里可以使用快速幂算法进行解决. 快速幂算法从低位开始逐位的查看指数位，如果指数位是 1，就将底数乘到最终的结果上，这里模乘运算还可以使用 Montgomery 模乘进行优化，相关细节在代码注释中给出：

```
def power_mod(a,k,N):
    res = 1
    #A = A%N
    a = Barett(a,N)
    #预先进行一步模数运算
    while k:
    #对指数位进行移位，最终移位到0停止，也就是指数位的BIT数
        if k&1:
            res = montgomery_mult(res,a,N)
            #RES = (RES*A)%N
        #如果有效位是1，那么就乘一个A
```

```
        a = montgomery_mult(a,a,N)
        #A = (A*A)%N
        #指数后移一个，那么等于扩大底数
        k = k >>1
    return res
```

3.3 改进快速幂算法

在课程中，老师提到上述的算法可能存在侧信道泄露的问题. 因为有的位需要乘底数，而有的位不需要乘底数，这是很危险的. 而下面这种算法在每一轮运算中都需要做两次乘法，因此不会造成功耗的明显变化.

Algorithm 9.5 Montgomery's ladder

INPUT: An element $x \in G$ and a positive integer $n = (n_{\ell-1} \ldots n_0)_2$.
OUTPUT: The element $x^n \in G$.

1. $x_1 \leftarrow x$ and $x_2 \leftarrow x^2$
2. **for** $i = \ell - 2$ **down to** 0 **do**
3.   **if** $n_i = 0$ **then**
4. $x_1 \leftarrow x_1^2$ and $x_2 \leftarrow x_1 \times x_2$
5.   **else**
6. $x_1 \leftarrow x_1 \times x_2$ and $x_2 \leftarrow x_2^2$
7. **return** x_1

图 1: Montgomery's ladder

这个算法和上面的算法类似，只不过通过 x_1, x_2 使每一轮都做两次乘法运算，如果 bit 位为 0，那么就先 x_1^2, 在将 $x2 * x_1$，其实就相当于直接给 x_2^2，而当 bit 为 1，就直接将 x_1 乘到 x_2 上，然后直接给 x_2^2，算法的具体实现如下，和上面一样，模乘运算同样可以使用蒙哥马利模乘进行:

```
def ladder(a,k,N):
res = 1
#x1 = A%N
```

```
    #x2 = A^2%N
    x1 = Barrett(a,N)
    x2 = montgomery_mult(a,a,N)
    while k:
        if k&1:
            #x1 = x1*x2%N
            #x2 = x2^2%N
            x1 = montgomery_mult(x1,x2,N)
            x2 = montgomery_mult(x2,x2,N)
        else:
            #x1 = x1*x1%N
            #x2 = x1*x2%N
            x1 = montgomery_mult(x1,x1,N)
            x2 = montgomery_mult(x2,x1,N)

        k = k>>1
    res = x1
    return res
```

3.4 密钥生成过程

在密钥生成过程中，加入实验原理中进行的预处理过程，提前计算 $dp, dq, qinv$ 并进行保留，相关的细节已经在代码注释中给出.

```
def Key():
#P = RANDOM_PRIME(2^1024-1,LBOUND = 2^1023)
#Q = RANDOM_PRIME(2^1024-1,LBOUND = 2^1023)
#上面是利用SAGEMATH随机生成的算法，下面为了实验方便我们使用确定的P和Q
n = p*q
#然后我们计算PI
phi = (p-1)*(q-1)
#下面随机在1-PHI之间选取与PHI互素的数E,不过在一般情况下我们取65537
'''
E = RANDINT(1,PHI)
```

```
    WHILE GCD(E,PHI)!=1:
        E = RANDINT(1,PHI)
    '''
    e = 65537
    #现在我们已经有了E, 下面计算E对PHI的模反指数D
    d  = inverse_mod(e,phi)
    #现在我们就有了公钥N和E, 还有私钥D
    #下面是计算用于CRT的私钥
    dp = inverse_mod(e,p-1)
    dq = inverse_mod(e,q-1)
    qinv = inverse_mod(q,p)
    return n,e,d,dp,dq,qinv
```

3.5 加密和解密过程

加密过程和原始的 RSA 加密过程一样，因为对于文件加密者来说，只受到了加密公钥 d.

```
    def Encrypt(m,e,n):
    #这是加密程序,ALICE利用公钥加密密文M发送给BOB
    public_text = power_mod(m,e,n)
    return public_text
```

在解密过程中，根据上述的实验原理，先利用 dp 和 dq 计算 m1 和 m2，然后再利用 Garner's Algorithm 计算 m. 当然这里面的模乘运算和模运算可以使用实验四实现的 Montgomery multiplication 和 Barett reduction.

```
def Decrypt(c,dp,dq,qinv):
    #这里有明文C, 私钥D
    #C1 = C%P
    #C2 = C%Q
    #C1 = BARRETT(C,P)
    #C2 = BARRETT(C,Q)
```

```
    c1 = amod(c,p)
    c2 = amod(c,q)
    m1 = power_mod(c1,dp,p)
    m2 = power_mod(c2,dq,q)
    h = montgomery_mult(qinv,m1-m2,p)
    #H = (QINV*(M1-M2))%P
    m = m2 +h*q
    return m
```

4 结果验证和对比

下面进行加解密结果正确性验证，可以通过下面的图片看出对明文进行加密，然后再使用 CRT 算法优化后的算法进行解密，可以看到解密结果和明文一致.

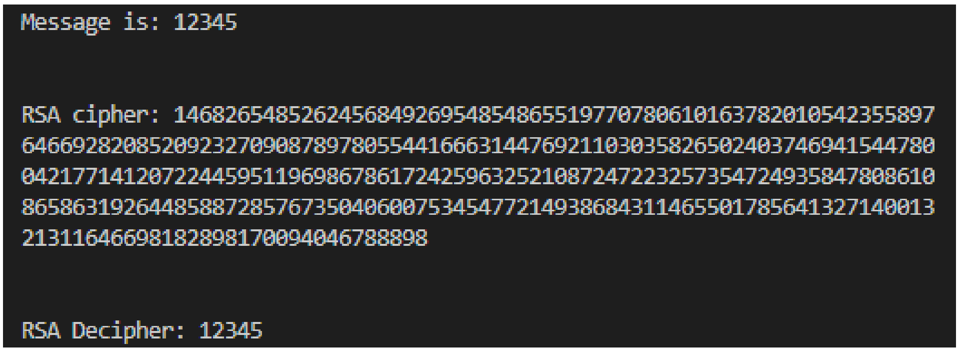

图 2: Verify

下面对 RSA-CRT 的效率进行简单的分析，我使用 jupyter notebook 对两种解密方式进行分析，可以看到采用 CRT 模式的情况下均值是 1.6ms，而不适用 CRT 优化均值是 4.4ms，提升近 3 倍.

```
In  [16]: %timeit Decrypt(cipher,dp,dq,qinv)
          1.6 ms ± 1.73 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In  [19]: %timeit Decrypt2(cipher,d,n)
          4.44 ms ± 28.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

图 3: Compare

5 分工和心得

在实验过程中进一步深入了解了 RSA 算法，并学习到了 RSA 算法和中国剩余定理的结合，通过实际测试体会到了 CRT 优化带来的效率提升. 在代码实现的过程中，学习了快速幂算法和扩展欧几里得算法，并使用代码进行了实现. 对扩展欧几里得算法的原理更加深入的理解，以前只知道如何使用，而这次深入了解的原理，收获很大.

参考文献

[1] https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

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

[3] https://asecuritysite.com/encryption/random3

[4] https://en.wikipedia.org/wiki/RSA_(cryptosystem)?wprov=srpw1_0