实验报告

【实验目的】

1. 掌握 RSA 算法原理及实现。
2. 了解常见的 RSA 攻击方法。

【实验环境】

1. 语言：C
2. 平台：clion 2021.2 版本

【实验内容】

一、RSA算法

1. **算法流程**

   - **伪代码：**

     1. **强素数生成算法伪代码：**

        强素数应满足下列条件：

        1 什么是强素数

        1984年Gordon J.提出了强素数的概念，强素数p应当满足以下4个条件：

        (1) p是一个很大的随机素数；

        (2) p-1必须有一个很大的素数因子r；

        (3) p+1必须有一个很大的素数因子；

        (4) r-1也必须有一个大的素数因子。

        该定义于1986年写入ISO-DP-9307[3]。

        其大致的算法流程如下：

算法步骤如下：

s1: 随机生成 128位二进制数 x, 确保首位与末位为 1;

s2: 采用 Rabin-Miller概率素性检测 x, 若通过则转 s3, 否则转 s1;

s3: 随机生成 128位二进制数 q1, 计算 r=x*q1+1;

s4: 采用 Rabin-Miller概率素性检测 r, 若通过则转 s5, 否则转 s3;

s5: 随机生成 256位二进制数 q2, 计算 p=r*q2+1;

s6: 采用 Rabin-Miller概率素性检测 p, 若通过转 s7, 否则转 s5;

s7: 对 p+1采用 (256) 10以内的素数进行分解，最终得到 m, 如果 m少于 64位转 s5;

s8: 输出 512位素数 p, 该素数满足强素数特性①②④，并以较大概率满足特性③。

伪代码如下：

Algorithm 1 get_strong_prime

```
    Input: bit_length
    Output: prime
get a prime p of 256bit
for Rabin_wit(pq_1) == false do
    get a random number q in range 2^256 to 2^257
    pq_1 = p*q+1
end for
for Rabin_wit(prime) == false do
    get a random number r in range 2^512 to 2^513
    result = pq_1 * r
end for
return prime
```

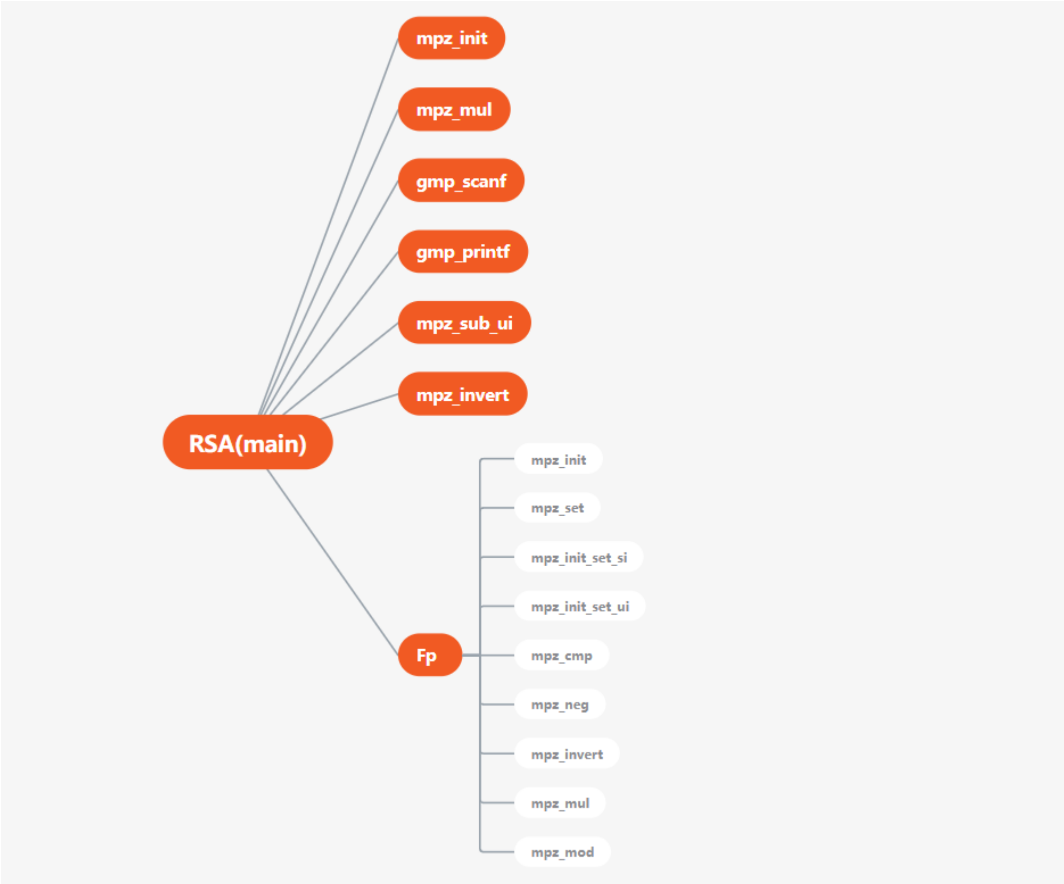
2. 加密指数e的生成：

生成了e之后可以相继确定d，在生成e，d时需要注意以下几点：

> 1. e不能过小，否则可以直接分解。
> 2. d不可过小，需要满足$d >= \frac{1}{3} \times N^{\frac{1}{4}}$否则可以通过连分式理论破解。

e的选择主要有两种，一种是直接选择e=65537， 这样得到的d也较大，另一种是先选择d使之满足第二点的要求，再通过求逆元得到e。

- **函数调用图：**



1. 在进行解密时，可以采用中国剩余定理加速。原理如下：

$$m \equiv c^d \, mod \, p * q$$

运算时，可以利用中国剩余将其分解为方程组

$$\begin{cases} m \equiv c^d \ mod \ p \\ m \equiv c^d \ mod \ q \end{cases}$$

缩小模数及幂次以加快运算速度，其中，令

$$\begin{aligned} d_1 &\equiv d \ mod \ p-1 \\ d_2 &\equiv d \ mod \ q-1 \\ c_1 &\equiv c \ mod \ p \\ c_2 &\equiv c \ mod \ q \\ m_1 &\equiv c_1^{d_1} \ mod \ p \\ m_2 &\equiv c_2^{d_2} \ mod \ q \\ q' &\equiv q^{-1} \ mod \ p \end{aligned}$$

那么最后方程的解为$m \equiv m_1 \times q \times q' + m_2 \times p \times p' \ mod \ N$

2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA.exe"
908476527945997372270310512257843446357046054844601178279553109663160871138675803986984662440780963693572545421019820171
 0287359721897030068646353331803383847658128301673191733319
658095039219456120108212222379832238548414272254622727412601986833530725509567656248224707048870541531999954162212518311
 795751129437784576275843041325259889627849392531691924887
292040952240292659762719148812063624453872427002299816698103758369418288974507215399795170913290346915374615405878230204
 2356274604585338978224507837300094680174067789478551346335
826718899719870938310221629552886186705042966870912238136248179774211463529775 0
1
5616976246200305158039112558279347117127190323140566029817119222816875304862873433160440188246954626180457052713652373354
6503116990250870468693721436449417595407928058472984986636283252068375379203054671193294002273552318509530075387982542250
3310814623494784422486899345631204592432795069494302263847732614570838788115433044853549920244063677931167782491853844496
197927972890261768738120358345754986526753 0
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA.exe"
3145146953629048607738364782752285594106365442120564670006583827002441998129206649536184601915044436208346
 336968257420236259053514044492545183009495889575175420573
2715103174948853754079516214181578000578870225651557171880455968607799840339723687098910246678506771888275
 596947416571314897563955799952947290635642290806606383147
8353071955346697473854286582418805429715488371674438543487903930263478574464018536690463223310462836209978
 964392087478519876793678011898365315127216008242280076273
6100059052757661792577723524531265362483490423134877375290816503122522707704131321378098718702241606115899
 042268595306050345349721365091890288913166008828457981678896914533049036894858915683951643003086886242119
 6573856333723979295424547857615780183996990677414987010216331640845717498130753216481826795665730197897 92
0
15207835352836962566843817308506642859032535621100422165482455033137859330220880
Process finished with exit code 0
```

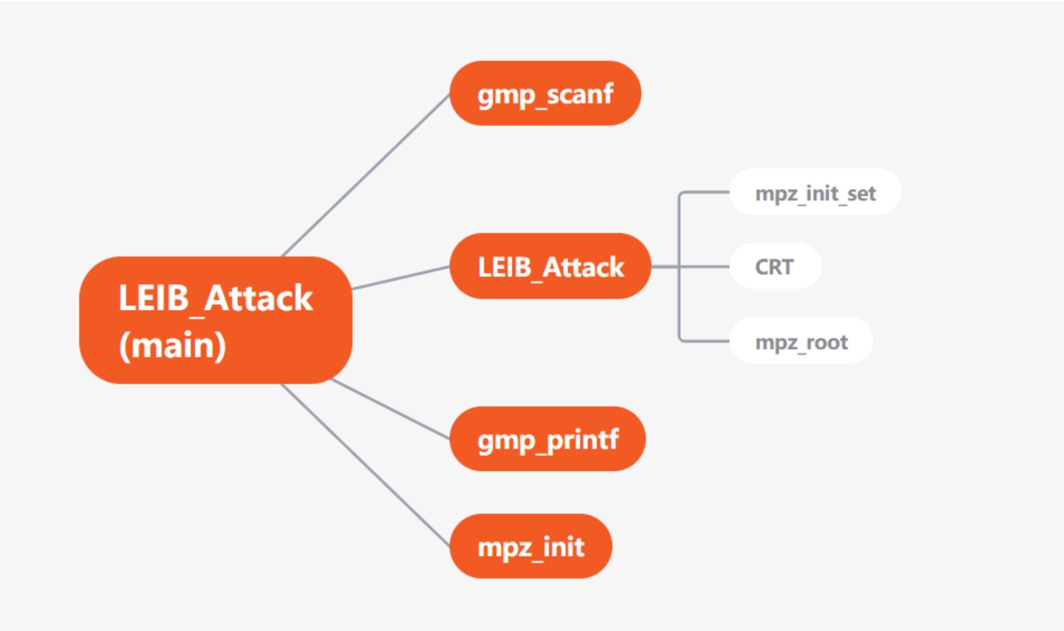
3. 讨论与思考：

> 1. 在本节1.1中已经给出了一些提高RSA安全性的措施，如：选择至少1024位的大素数作为p,q；保证p,q是强素数；保证p,q差值较大；保证e不能过小；保证$d >= \frac{1}{3} \times N^{\frac{1}{4}}$；对于一个模数N，不能用两组及以上的公钥加密同一消息，即：选定一个模数N后只生成一对e,d等
> 2. RSA的NP问题就是基于大数分解的困难性，选择大素数首先保证了密钥的安全性，其次更好满足了对于e，d的要求。

二、小指数广播攻击

1. 算法流程

- **函数调用图：**



- 加密指数过小，可以尝试对密文c直接进行开n(3,5等等)次根运算，尝试得到明文m。若开根后得不到明文，还可利用中国剩余定理将不同密文（至少两组）组成方程组求解m^n，再进行开根运算，若依然得不到明文m，则换不同的组合依次尝试，直到解出明文m。

2. 测试样例及结果截图：

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_LEIB_Attack.exe"
5
5
5237805360905343991463786249893320594810289324753993495779837563930262254405740864391991579649674
273965850672156409736682495098761512997606791440567791905799764746780701501738996354606916280080413
421315011511328883898332089846238925797584067360270125733993412172195328902540924044994647896112466
1100365290994923166639393499528364132472249752491751774902344901796554918237008377930110072185725283
171285941597524901843910825980559513046934101819574823613136850797805219220097914819624003143111120
430288594795771914142299703947415266517444985935340964254479388655746242605540787116246700560005663
815924494431443493097612335286538070451687236046520927137833404829655454884965688238114944123092821
1101144250801893701071836240764923036002195198985195844582149406798928263398174360668285254459036689
122527503747418467804784628662899125215658290182264993438817507537028003106176205781823457005237507
343005085795516801840536590431220991431272473875664500714831695953582369208778516763015132339540751
93836990443949686201568664981191507299551781949035
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_LEIB_Attack.exe"
3
3
380160393409916084581586482073964227679413819412971373380690696928015957687074293541296614386179401
1429713533445220937904437230043628669296730754094678194794827686080260930172169628375879375376024199
2493028086186569192712361957490111800395573100667030722097914553188141420312150830172172316578472120
2821647924905924718857862155692498016728304756100256662125417228066283228996656947866082027636681183
96579096900203989769961925472998438075513232694310979626502760119983872953688937489313960167023228
199180892410493658788014721774061164434334635121897210306985296717419316480186046667246141016771527
73210300988719884596236874756477117122193057746909
Process finished with exit code 0
```

3. **讨论与思考：**

   这种攻击手段提示我们加密时一定不可以选择很小的e，但其实也不能选择很大的e，这样的话会导致解密指数d很小。

三、共模攻击

1. **算法流程**

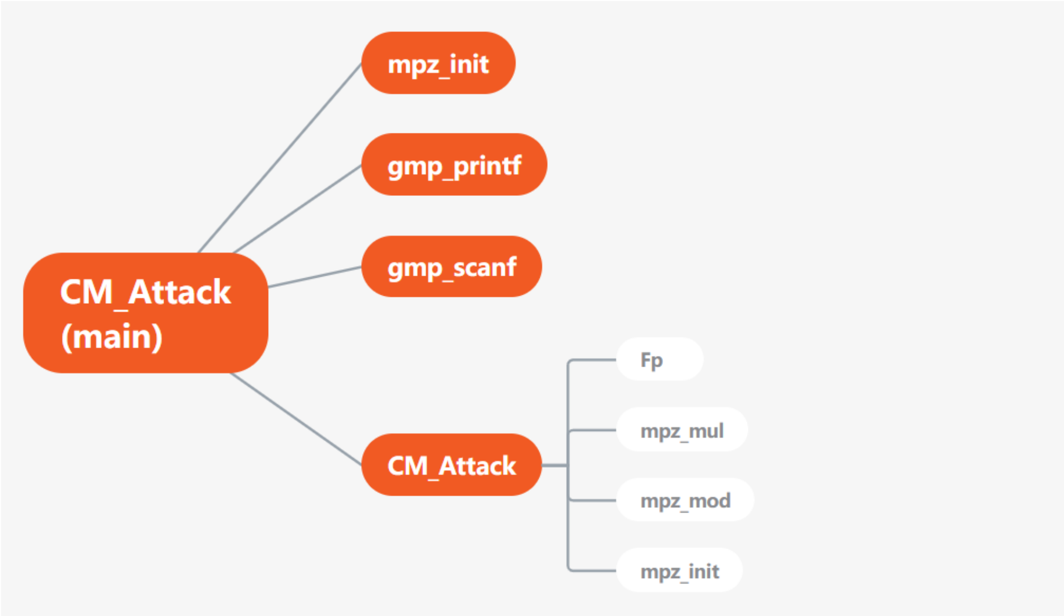
   - **原理：**

     若$(e_1, e_2) = 1$那么可以利用欧几里得算法得到$e_1 \times s_1 + e_2 \times s_2 = 1$所以有

     $$m \equiv m^{e_1 \times s_1 + e_2 \times s_2} \equiv c_1^u \times c_2^v \ mod\ N$$

     从而得到明文m。

   - **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_CM_Attack.exe"
346270541728252349966689606661927676577210124792353
595614202586059875656387022106783821468266573506693
154955311000579732437188648319131994792342638009003843285187270576012791286122494146996214447606784
807541138648096185361509171275485390610449558307527231087810431490878307690882387100577913160269 68
156901705893430563139409154971022074895105669127211169712652765533727765305030716855665951223742613
551916454676857405892718892291039989230059437594 59
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_CM_Attack.exe"
959559461963512608443126267894722577435595927353245
453561406060629441513517787631500095541985432999623
933487881511791159997776588419923755663371696036135458267758289418730881261511583913559820909490 56
195535073200740134718658864013777573590333481680107472823411070838887551713826888289490054914765135
284692435486579387446059868209966611187130207151707238183764764200229735175022665877552838862750631
5211122954991219407743790367561809234505247224529 5
Process finished with exit code 0
```

3. **讨论与思考：**

对于一个大整数N不能生成两个不同的公钥e1,e2，更不能用这两个不同的公钥加密同一个消息。

四、已知公私钥分解合数N（选做一）

1. **算法流程**

   - **原理：**

计算$k = e * d - 1$，选一个随机数计g算$g^k \equiv g^{e*d-1} \equiv 1 \ mod \ N$，k是偶数，可被分解为：$k = 2^t \times r$，且有

$$g^k - 1 \equiv (g^{\frac{k}{2}} - 1)(g^{\frac{k}{2}} + 1) \equiv 0 \ mod \ n$$

验证$g^{\frac{k}{2}} - 1$是否是n的因子，即$n \ mod \ g^{\frac{k}{2}} - 1$是否是0。若是，则$g^{\frac{k}{2}} - 1$为p,q其一；若不是，继续将$g^{\frac{k}{2}} - 1$平方差分解，也即将k除以2，验证$g^{\frac{k}{2^2}} - 1$是否是n的因子。以此类推。在k被分解出奇数r前，总能得到某个$g^{\frac{k}{2^l}} - 1$为n的因子，也即得到p,q。

- ## 伪代码：

  1. **伪代码：**

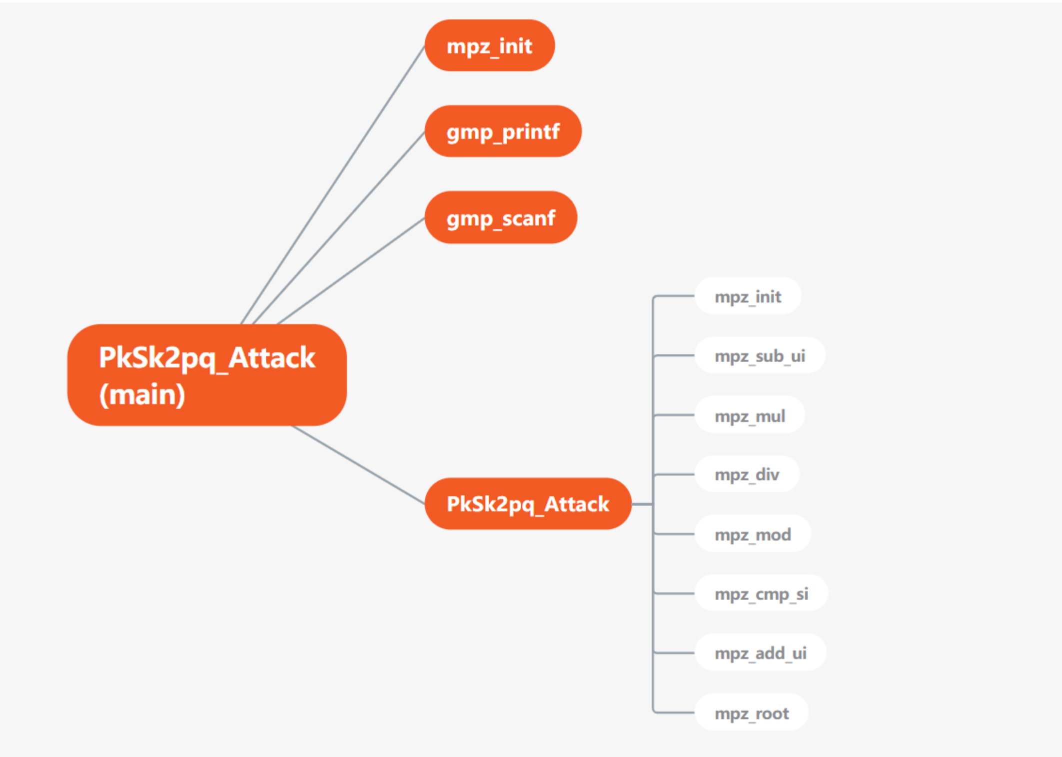
算法 2 已知e，d分解N

输入： e, d, n

输出： p, q

```
function RESOLVE(e, d, n)
    while 1 do
        k = e * d - 1
        g = random(2, n)
        while kmod2 == 0 do
            k = k//2
            temp = g^k(modn) - 1
            if gcd(temp, n) > 1 & temp! = 0 then
                p = gcd(temp, n)
                q = n//p
                if p > q then
                    p, q = q, p
                end if
                return p, q
            end if
        end while
    end while
end function
```

- ## 函数调用图：



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_PkSk2pq_Attack.exe"
185826930866337241052147166615123554610976232886325
129314958794825303057876604675231489563157347507079784638505001928383259840079245329593446794945053
142033935361876741177350358368393633644913604028211673524986056669737625139443380031850600678768257
8714971402650636317095011632046200556138984923633
16297693796066557615622499749924387348947252221329
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_PkSk2pq_Attack.exe"
112657465789831416610210431035939941338974980593875
3915187856228974120521846889823579330409190830576834603127066565836479984149201716155251645837996 51
5027015867400781650426521998372694333283151285970238644019163429662494214279384894836984901404266 07
1257604189079087936579030891936822157724056192025 3
3997295739832052816474082400583161071193295143061 9
Process finished with exit code 0
```

五、维纳攻击（选做三）

1. 算法流程

- ### 原理：

当d较小满足Wiener攻击的条件时，可利用连分数理论计算出$\frac{e}{n}$的渐进分数，其覆盖了$\frac{d}{k}$，在得到e,d,n,k等值后，便可轻易计算出$\phi(n)$，进而根据n的值建立二次方程p,q的值。

- ### 伪代码：

1. **求渐进分数的伪代码：**

算法 3 求渐进分数list

输入： x, y
输出： ans

```
function LISTF(x, y)
    while y do
        a.append(x//y)
        x, y = y, x mod y
    end while
    ans = []
    for i = 1 → len(a) do
        d, k = 0, 1
        for j = i → 0 do
            d, k = k, a[j] * k + d
        end for
        ans.append((d, k))
    end for
    return ans
end function
```

2. **维纳攻击伪代码：**

算法 4 维纳攻击

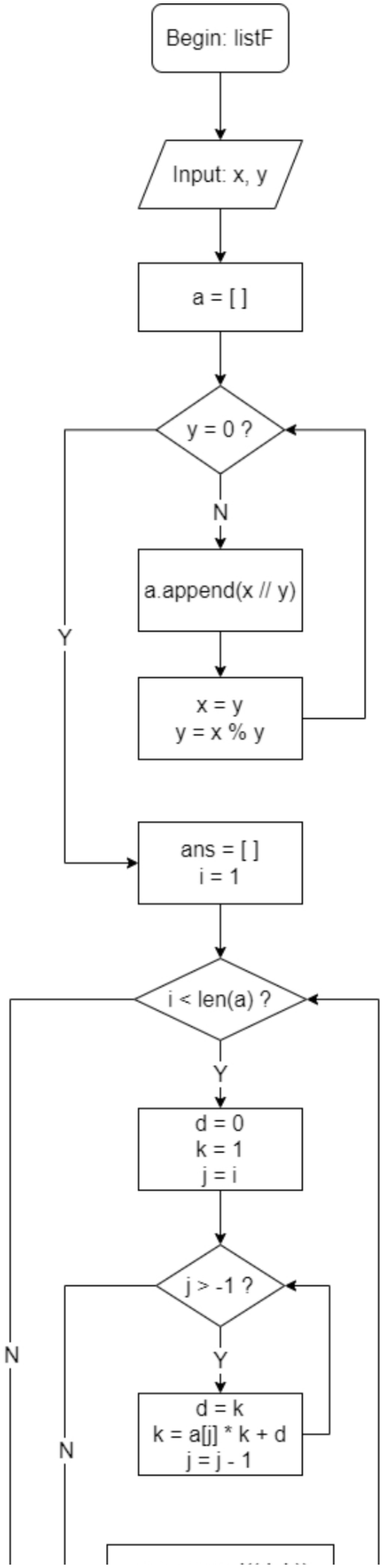
输入： e, n
输出： d, p, q

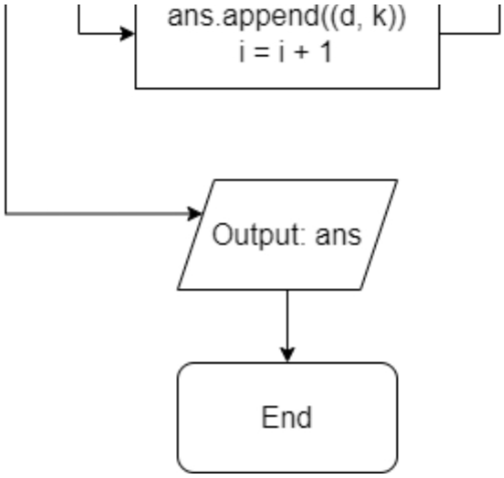
```
function WEINERATTACK(e, n)
    a = listF(e, n)
    for i = 0 → len(a) do
        d, k = a[i]
        if k == 0 then
            continue
        end if
        if (e * d − 1)modk! = 0 then
            continue
        end if
        phi = (e * d − 1)//k
        dlt = sqrt((n − phi + 1)^2 − 4 * n)
        px, qy = (n − phi + 1 + dlt)//2, (n − phi + 1 − dlt)//2
        if px * qy == n then
            p, q = int(px), int(qy)
            d = invmod(e, (p − 1)(q − 1))
            if p > q then
                p, q = q, p
            end if
            return d, p, q
        end if
    end for
end function
```

- ## 函数调用图：

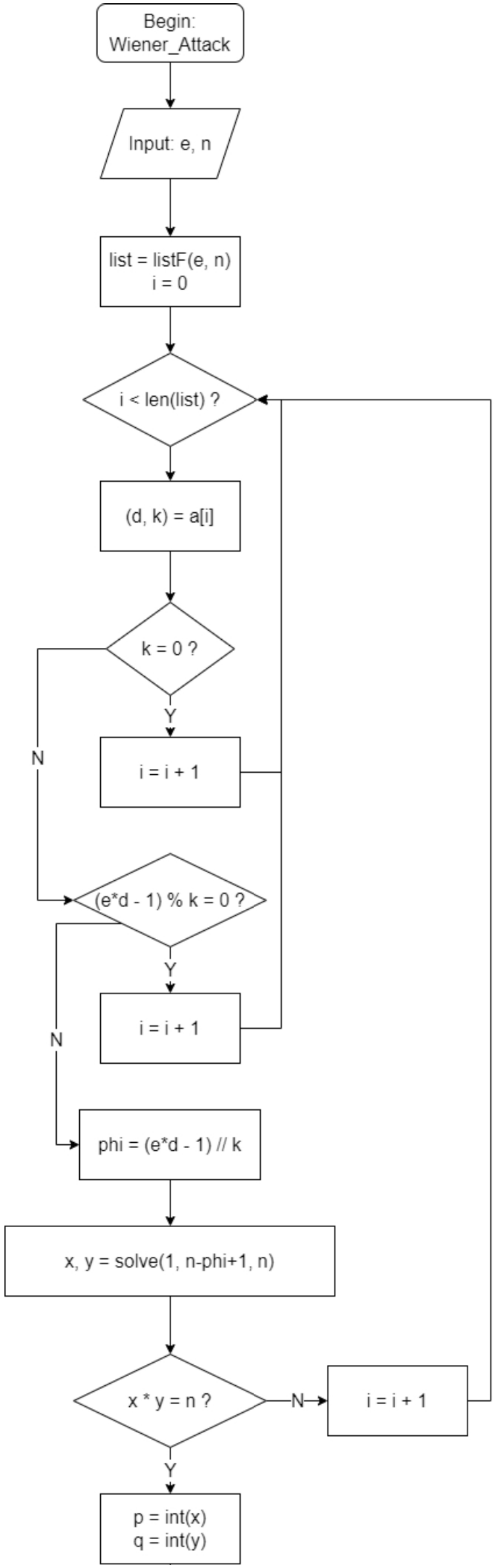
- ## 流程图：

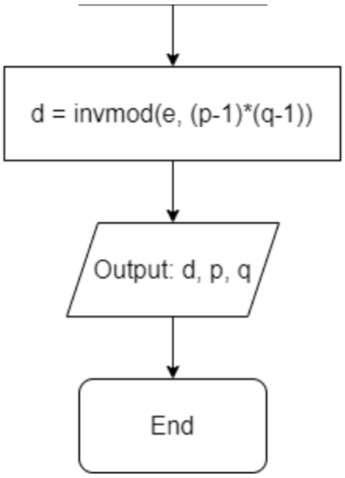
1. **渐进分数流程图：**


Begin: listF
Input: x, y
a = [ ]
y = 0 ?
N
Y
a.append(x // y)
x = y
y = x % y
ans = [ ]
i = 1
i < len(a) ?
Y
N
d = 0
k = 1
j = i
j > -1 ?
Y
N
d = k
k = a[j] * k + d
j = j - 1



2. **维纳攻击流程图：**


Begin:
Wiener_Attack
Input: e, n
list = listF(e, n)
i = 0
i < len(list) ?
(d, k) = a[i]
k = 0 ?
Y
N
i = i + 1
(e*d - 1) % k = 0 ?
Y
N
i = i + 1
phi = (e*d - 1) // k
x, y = solve(1, n-phi+1, n)
x * y = n ?
N
i = i + 1
Y
p = int(x)
q = int(y)



2. **测试样例及结果截图：**

   本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_Wiener_Attack.exe"
1810052545521859708126118363541669585649646237957696825637794517098768219433950302563813210599083466267405256198156197613033570939979757471975694
 983970887462503376766648906853211788576985486451614962365659123224750406105217368206069618852379246729211012323616299804098592531931417382273038
 292524770113399109123664571640214558945403930156109290457936291593460591651824249787565171683867125512347219794061
3848175304374239446374136728652210327196184921990571543190924427517351475552322243357299587707005052524192633432368462384104965242180681028180521 15
 690102102817202012835053827117309537789639047644474627523566337793717961356345587053810694828102445856600364001916903873697200345625192977636587
 409635379422334512156148878584144609837439564269731902888172531553099811749498006772906041020950968367021004116903
858519161386877541694989232057939656610614805396964282620189311960158818413634511485445944359054515659410802083095019917
260340383317538703418270723688879983709112952901823729633674862314469789611151537
448234061329252327411235839618501786194375528895706189031247788013373161412170343429905707933825906391643894783027692500
7593033646168761933570572534741651862486945516128473882069851085095134636496732119
3070485135032081312266360105609808975477811133561813202666605341872797216847987827223308370285829477
Process finished with exit code 0
```

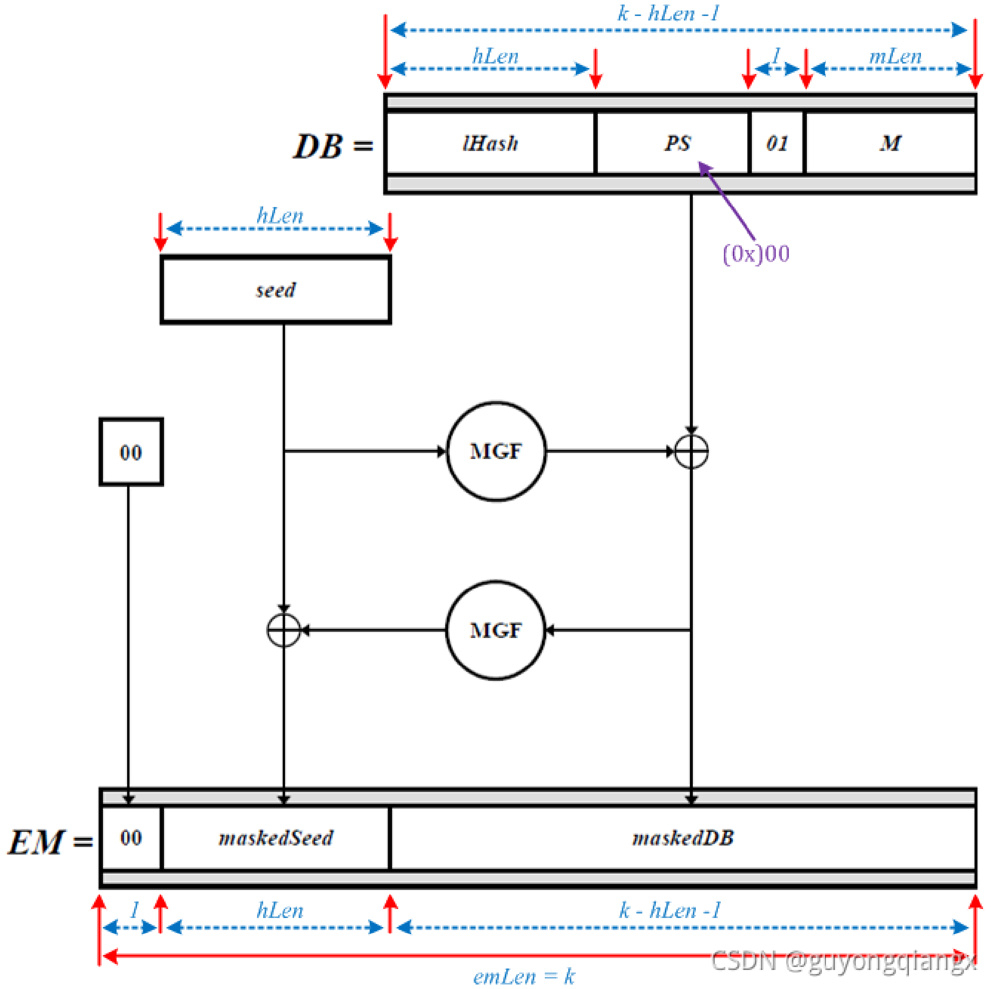
```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_Wiener_Attack.exe"
2841004786931616423276957124525054688917944103019064654346046433658550641019222526983275845249569553735533558141387844026055175364360090733723392
 644225226100100128772436304548891271600563586375997048719376594439856448714533455767284144224890757917397315472851386483077707751553125459287210
 946029495882371193 45
4684598872797817891888861888573017406548524570309663876064881031936564733341508945283407498306248145591559137207097347130203582813352382018491852 9
 228491868272791115552239820322717019726424382247300822166721103161425281082397081717818504915784333099640932939076970727415386493478948638991033
 400303478588677052 31
18489327396055733397216193236128138397765028288613793035021305599301380136673327250408422592244732819005905679957567952 9
74717041052102175277835219391448987
25336772790324258952117622504537139442881120269760383961991795601846585772802865528712760553670210656524156997774484665 8
33049279421936394718949688217533213
55376898524316912820937618623478740716277734256147429013087729366616259670633
Process finished with exit code 0
```

六、RSA-OAEP（选做二）

1. **算法流程**

   - **原理：**

     1. **加法填充原理：**



RSAES-OAEP-ENCRYPT $((n, e), M, L)$

可选项：　Hash　哈希函数（$hLen$ 表示散列函数输出的以八位组为计量单位的长度）

　　　　　MGF　掩模生成函数

输入：　(n, e)　接收方的 RSA 公钥（k 表示 RSA 合数模 n 的以八位组为计量单位的长度）

　　　　M　待加密的消息，是一个长度为 $mLen$ 的八位组串，其中 $mLen \le k - 2hLen - 2$

　　　　L　消息的可选附加标签：如果没有提供 L，那么 L 的默认值是空串

输出：　C　密文，一个长度为 k 的八位组串

出错提示：　“消息太长”；“标签太长”

假设：　RSA 公钥 (n, e) 是有效的

步骤：

1. 长度检查：

   a. 如果 L 的长度超出哈希函数的输入限制（SHA-1 的限制是 $2^{61}-1$ 个八位组），输出“标签太长”然后终止运算。

   b. 如果 $mLen > k-2hLen-2$，输出“消息太长”然后终止运算。

2. *EME-OAEP* 编码（见 **错误！未找到引用源。**）：

   a. 如果没有提供标签 L，则让 L 为空串。让 $lHash = \mathrm{Hash}(L)$，这是一个长度为 $hLen$ 的八位组串（见下面的注释）。

   b. 生成一个由 $k-mLen-2hLen-2$ 个零值八元组构成的串 PS。PS 的长度可能是零。

   c. 连接 $lHash$，PS，十六进制值为 0x01 的八元组和消息 M，形成一个长度为 $k-hLen-1$ 个八位组的数据块 DB：

   $$DB = lHash \,\|\, PS \,\|\, 0x01 \,\|\, M\ 。$$

   d. 生成一个长度为 $hLen$ 的随机八位组串 $seed$。

   e. 使 $dbMask = \mathrm{MGF}(seed, k-hLen-1)$

   f. 使 $maskedDB = DB \oplus dbMask$.

   g. 使 $seedMask = \mathrm{MGF}(maskedDB, hLen)$.

   h. 使 $maskedSeed = seed \oplus seedMask$.

   i. 连接一个十六进制值为 0x00 的八位组，$maskedSeed$ 和 $maskedDB$，形成一个长度为 k 个八位组的编码消息 EM

   $$EM = 0x00 \,\|\, maskedSeed \,\|\, maskedDB。$$

3. *RSA* 加密：

   a. 将编码消息 EM 转换成一个整数消息代表（见 4.2 部分）：

   $$m = \mathrm{OS2IP}(EM)\ 。$$

   b. 将 RSA 公钥(n, e) 和消息代表 m 代入 RSAEP 加密原语（5.1.1 部分），产生一个整数的密文代表 c：

   $$c = \mathrm{RSAEP}((n, e), m)\ 。$$

   c. 将密文代表 c 转换为一个长度为 k 个八元组的密文 C（见 4.1 部分）：

   $$C = \mathrm{I2OSP}(c, k)\ 。$$

4. 输出密文 C。

2. 解密原理：

RSAES-OAEP-DECRYPT (K, C, L)

选项：	Hash	散列函数哈希（$hLen$ 表示散列函数的输出的以八位组为计量单位的长度）
	MGF	掩模生成函数
输入：	K	接受方的 RSA 私钥（k 表示 RSA 合数模 n 的以八位组为计量单位的长度）
	C	待解密的密文，使一个长度为 k 的八位组串，其中 $k \ge 2hLen+2$
	L	可选标签，其与消息的联系将得到验证；如果没有提供 L 值，则 L 的默认值为空串。
输出：	M	消息，是一个长度为 $mLen$ 的八位组串，其中 $mLen \le k-2hLen-2$
错误提示：	“解密出错”	

步骤：

1. 长度检查：

   a. 如果 L 的长度大于散列函数的输入限制（SHA-1 的限制是 $2^{61}-1$ 个八位组），输出"解密出错"并中止运算。

   b. 如果密文 C 的长度不是 k 个八位组，则输出"解密出错"并中止运算。

   c. 如果 $k < 2hLen + 2$，则输出"解密出错"并中止运算。

2. *RSA* 解密：

   a. 将密文 C 转换成一个整数密文代表 c（见 0 部分）：

   $$c = \mathrm{OS2IP}\ (C)\ 。$$

   b. 将 RSA 私钥 K 和密文代表 c 代入 RSADP 解密原语（见 0 部分），从而产生一个整数消息代表 m：

   $$m = \mathrm{RSADP}\ (K, c)\ 。$$

   如果 RSADP 输出"密文代表超出范围"（意思是 $c \geq n$），则输出"解密出错"并且中止运算。

   c. 将消息代表 m 转换成一个长度为 k 个八位组的编码消息 EM（见 0 部分）：

   $$EM = \mathrm{I2OSP}\ (m, k)\ 。$$

3. *EME-OAEP* 编码：

   a. 如果未提供标签 L 的值，则使 L 的值为空串。使 $lHash = \mathrm{Hash}\ (L)$，这是一个长度为 $hLen$ 的八位组串（见 0 部分的注释）。

   b. 将编码消息 EM 分解为一个八位组 Y，一个长度为 $hLen$ 的八位组串 $maskedSeed$，以及一个长度为 $k - hLen - 1$ 的八位组串 $maskedDB$，使得

   $$EM = Y \,\|\, maskedSeed \,\|\, maskedDB\ 。$$

   c. 使 $seedMask = \mathrm{MGF}\ (maskedDB, hLen)$.

   d. 使 $seed = maskedSeed \oplus seedMask$.

   e. 使 $dbMask = \mathrm{MGF}\ (seed, k - hLen - 1)$.

   f. 使 $DB = maskedDB \oplus dbMask$.

   g. 将 DB 分解成一个长度为 $hLen$ 的八位组串 $lHash'$，一个（可能为空的）由十六进制值为 0x00 的八位组构成的填充 PS，以及一个消息 M，使得

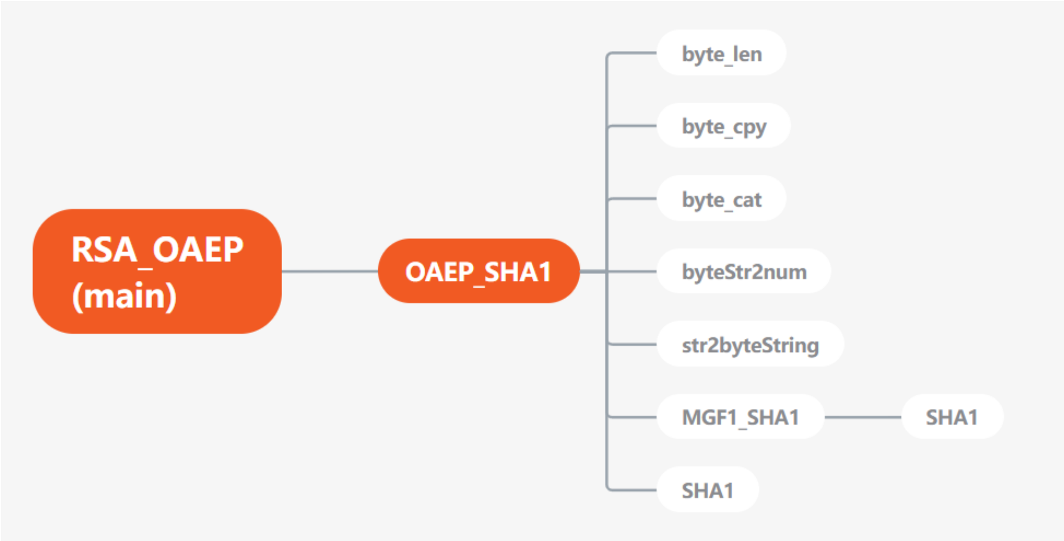
   $$DB = lHash' \,\|\, PS \,\|\, \mathrm{0x01} \,\|\, M\ .$$

   如果没有可以从 M 中分离出 PS 的十六进制值为 0x01 的八位组，如果 $lHash$ 没有等同的 $lHash'$，或者如果 Y 是非零的，则输出"解密出错"并中止运算。（见下面的注释）

4. 输出消息 M。

注释：必须确保对手无法在步骤 3.f 中分辨出不同的出错条件，防止对手了解关于编码消息 EM 的部分信息，无论是通过出错消息或是定时，或者更一般的。否则对手可能能够获得关于密文 C 的解密的有用信息，进而导致像 Manger 发现的攻击手法一样的选择密文攻击[36]。

- **函数调用图：**



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_OAEP.exe"
1
128
0x10001
0xc168fb2417c03f369904f83060870eb7ee92e609b6087d8345d927579c2495618fed5c08ab41673bc6d1597971c1652ffced57
 a51a449919844bf2dc57d1ae74d76b09870bd1c1d7bacbb762cc24d8d9c23995a3e08ee977197e1c90420ec01c3d4c5703667e9
0xab541d54c0f9d92c443e1835244e237f985cee56c6aefccd8c79764b1ded61d927b48d8fdf7ce8f694587988b88f9d0f37c11a
 7550d85ea514ab6045f78e063f0c1e3b7
0xfd8a1f243c7f38fd
0xaf80389753872f27c31dc6516607d755e11d21f4
Err
Process finished with exit code 0
```

```
 dce09cf94d89da60f
0x5cb545b00acadd8a12088ee58da7a25279e6b1b4347c986cee5f0710fe361864376c6e82648121e2e57b40b38f969f47c905211d6cb28fb88aaafcecb
 ca3b58b249ab2aff3d7cf8f2509bcea00f5ba43f9c6124a07fc5e938654d0d70d709b73ab1013a34d969c950e2ebce1279364a0df88a89e74b27fc9f8c
 5991835ff749a7b4098cbb7ca0e0f6a5255f7e3174d9dcfccc6fc8c7e834bbd14f2c58ac33094b277413b84620a5296ca9a2c8486d152ea3fb1025f2d8
 761649eb833b9083c09c8940a2f6c1981bc0839bbbbc608cb41e72d56671648d9bac0eca47240cd4ccadbd9c100f01913af27457a40dc25dad53c90c29
 4f68f034b77f583c8e58892d44113a4a73d565df68fc69e15fbfaec4643d22f14f376ea8f4edc3902aa68f7b39d42c0ba34b6eaaa0872ea1a739a40e26
 a1a9da48f0f1eb5c328eea541e6bcda9d85eac47f2185616fabc0025f25529b27ac36a880a7cae088942f489627fd2f51c13f5d9713d3d00ae5915da6a
 2e8b5603d7e3ade797fc92378115fadad98b9520370db7b96c0538b191afc82a2928e163b8b0852534f3e5905b547655707695b5b986db502e1737e1c8
 171e31cfa11d733e4
0x
0x88241d011b11d452f2a07054ac6ff68f559136fe9fa46e70d176e132e74ec124c871a71c2140da5bfc7b43f66fa1df3c7f43c9045f566980ebc528
46db39519dbddfd03a8b159618fcc09f242e15c9bb13182eddde444e5f86224a4689110b23b335a158cb5d01d75adeda690eee32cc947a1d952b5763
c9d39d7e24faa96811b55dd2717c408aefc13dcd8af8cccf51c88f803d6b5b397cae7dbd5088b4bd3858ad5e5566c33141c658f00c2190833f9f6185
0272f4a3b62f615b7adebedacb3aa2b01a324f01a3901c44e0c117c5d999b0195ab3b34e4308f55e53b4e8d6eaf502c2d8f549761bc3a3f84fdf39ef
bc45fca01ec598d71741a86ea37ecdc0105ee6718d1dd7b534517e51d9ee034b36a18fbd6a05b969f611b7d896fa979f5078faaf10db30f994a0b2e0
8b0d8a811a805fe03366892573cf20b1edace59b5722046174b4f2a5cd9dce5d156be92742fd90d4e9695847bbcacce59f1bde9f498a134a59a32a64
f59baef45580ca817f513e48a9718
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_OAEP.exe"
1
128
0x10001
0xc32a1dbd27a8b57617cde48c424ff5936d6415a679a019605deb0f1a4a3b29c73770d77fcb4f3ed36c4f2c94fb17e8c32feb0674b11d340c7734ffc4ca1
 6e6459022fe7b318e222b35435aaffcc430990acda67199035118ae140f80fc5b0dafaa3df9f9300a0316340403908278cfe3b9aca2aa6bf1
0xb96af050d20a2f68f1b1b68a90139ff3ef62c2c5527d41122bc6c2bfa59c300dea4791e2e9d9b74518042bc860
0x
0xf72f68a0abb2c333c489800626349e8ea7ecd4da
0x820d8cf1e57e85e3f4084da44ff0f08fdfdb6b0dbb3055ec56d8e113c5c816b54feae15cd18d71a3993a10d61db05bddcd2744d6e1385a3ef739bc
77590e0382246925643051b802761505cb6b3f0b4fe52feecba70b57b9f74d9b4c52e11fa89f7dcb2236689d56ddbe87724ae83c89c411af8eb653c4
d9519faf7d72dce495
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp7--RSA_OAEP.exe"
0
128
0x70c65007dcaf7ee0859e4e987b53a92c60a3bc3767b02b8595ddf706122ef2ab340c760bbe58619c07e1ca609367989602ab40a4facf2cc2148250b
 1989bc50a12fb092345693ab513d817f7f6925c6b4fff8d4c088ebf36c78eef654e95264b3955b63ceff8256f6d7f4590eef8806b81b2701
0x91b9c21e697dd8344cb9ea27b733c8998cf20d254557ed4279151fe1d00f4c3faff6ff85107ca708a09c0c6bc12917f43fc5349b912d940eb2e57f6
 fa92d0c52b78a546cfae785582888fc8646691409899e8259142e2eb7a9850111aaa86195fe125e367f3851588068af11f81e13d6591497b5
0x61a2ef593b028bfeec0f29f6fa02f21a9c86e97173e2b8bc56474abbb59bb725833bcacd65ad0c745c2357ae4fb9ef73301114ef1e3879023419eaf
 036dd060f0aae77b1e6262d942fd552f358af30b0bd89c241501cd0a1e24d44051c2418541173d925b665acb2f48a3ad45ea687a8fc21d7ea
0xbedb3324b469de7f
Ree
Process finished with exit code 0
```

七：思考题：

1. 考虑 RSA 算法在实际应用中提高安全性的措施；

   提高RSA安全性的几种方法 - 豆丁网 (docin.com)

   1）不可共用模，在一个系统中如果共用一个模数 n，则很容易遭受共用模攻击，降低系统安全性。

   2）明文熵要尽可能的大，使得第三方在得到密文的情况下，破解明文的概率降到最低。

   3）数字签名时，首先使用 HASH 函数对明文进行散列运算，保证签名的消息有足够的冗余度，防止篡改签名。

   对n, p, q, e的合理选取。

2. RSA 算法在生成密钥时为什么要选取大素数？请简要说明；

   非对称加密的关键是具有非对称功能,允许解密由非对称密钥加密的消息,而不允许找到另一个密钥.在RSA中,使用的函数基于素数的因子分解,但它不是唯一的选项(例如,椭圆曲线是另一个选项).

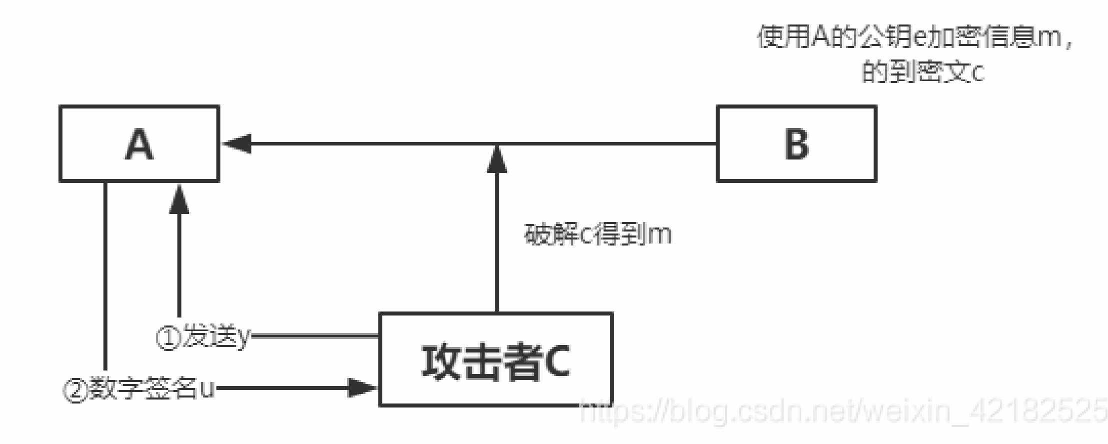
   因此,基本上您需要两个素数来生成RSA密钥对.如果您能够分解公钥并找到这些素数,那么您将能够找到私钥.**RSA的整体安全性基于以下事实:对大型复合数进行分解并不容易**,这就是为什么密钥长度高度改变RSA算法的鲁棒性的原因.

   每年都会有很多价格合理的计算器将大质数分解的竞赛.分解RSA密钥的最后一步是**在2009年通过分解768位密钥完成的**.这就是为什么现在应该使用至少2048位密钥的原因.

3. 阐述如何利用 RSA 算法的性质进行选择密文攻击。

   在使用RSA加密算法时，如果密钥使用不当的话也会有一定的危险，由于RSA既能作加密算法也能作数字签名算法，因此攻击者可以通过**选择密文攻击**的方式进行破解。

基本过程：



1. 攻击者C欲破解B发给A的密文c；
2. C选择 **r < n，计算 t = r-1 mod n**；
3. **x = re mod n, y = xc mod n**；
4. C把y发送给A，要求数字签名；
5. A对y进行数字签名 **u = yd mod n**；
6. 计算 **tu = r-1yd = r-1(x * c)d = r-1 r m = m mod n**，得到m的值。

求取乘法逆元方法和计算乘方取模方法和因子分解攻击里的一样。