華中科技大學

课 程 设 计 报 告

题目： 密码学课程设计

课程名称： 密码学课程设计

专业班级：

学　　号：

姓　　名： 黄浩岩

指导教师： 徐鹏

报告日期： 2020 年 10 月 17 日

教师评语：

分数：

网络空间安全学院

密码学课程设计任务书

课题内容：

（1）原始 SPN（教材上）算法的实现。
（2)对上述算法进行线性密码分析及差分密码分析(求出所有 32 比特密钥)。
（3）增强以上 SPN 的安全性（如增加分组的长度、密钥的长度、S 盒、轮数等)。
（4）对原始及增强的 SPN 进行随机性检测，对检测结果进行说明。
（5）生成 RSA 算法的参数（如 p、q、N、私钥、公钥等)。
（6）快速实现 RSA（对比模重复平方、蒙哥马利算法和中国剩余定理)。
（7）利用椭圆曲线密码算法、HASH 函数、压缩函数、对称加密算法实现一个类似 PGP 的文件加解密及完整性校验功能。
（8）利用 USBKEY 和数字证书实现简单的文件加密保护。
（9）构造彩虹表破解 HASH 函数。

课题任务要求：

（1） 掌握线性、差分分析的基本原理与方法。
（2） 体会位运算、预计算在算法快速实现中的作用。
（3） 可借助 OpenSSL、GMP、BIGINT、FLINT 等大数运算库的低层基本函数（加、减、乘、除、模)，实现过程中必须体现模重复平方、中国剩余定理和蒙哥马利算法的过程。内容(7)的算法可以直接调用 OpenSSL 或者其它密码库。
（4） 了解和掌握彩虹表构造的基本原理和方法，能够设计和实现约化函数（Reduction Function)，针对特定的 HASH 函数构造彩虹表，进行口令破解，了解彩虹表的弱点及防范方法。
（5） 独立完成课程设计内容，现场演示并讲解。
（6） 掌握 USBKEY 和证书的基本使用方法。
（7） 课程设计完成后一周内，提交课程设计报告。

主要参考文献（由指导教师选定）

（1） 密码学原理与实践（第三版）. DouglasR.Stinson 著，冯登国译，电子工业出版社，2009

（2） 应用密码学：协议算法与 C 源程序（第二版）. Bruce Schneier 著，吴世忠等译，机械工业出版社，2014

同组设计者 无

目　　录

1　SPN 实现

1.1 实验要求

内容：按照教材例 3.1 实现 SPN 加解密函数。
要求：（1）正确实现算法的加解密过程（2）快速实现算法

1.2 实验过程

1. 算法分析

SPN 实现的算法基本参照教材，如图 1.1 算法所示。

Algorithm: SPN

Input : x is plaintext, π_s, π_p, K is round key sequence
Output: y is ciphertext

$w \leftarrow x$;
for $r \leftarrow 1$ **to** $Nr-1$ **do**
  $u \leftarrow K[r] \oplus w$;
  for $i \leftarrow 1$ **to** m **do**
    $v \leftarrow \pi_s(u_{<i>})$;
  end
  $w \leftarrow (v_{\pi_p(1)}, ..., v_{\pi_p(lm)})$;
end
$u \leftarrow K[Nr-1] \oplus w$;
for $i \leftarrow 1$ **to** m **do**
  $v \leftarrow \pi_s(u_{<i>})$;
end
$y \leftarrow K[Nr] \oplus v$;
return y;

图 1.1 SPN 实现算法

其中，加密与解密使用相同 SPN 结构。

加密时使用密钥编排算法生成的 Nr+1 个轮秘钥$K = \{k^1, k^2, k^3, \dots, k^{Nr+1}\}$,使用 S 盒代换$\pi_s$。

解密时需要将加密使用的轮秘钥倒序，且中间的 Nr-1 个轮密钥进行逆置换，即$K = \{k^{Nr+1}, \pi_p^{-1}(k^{Nr}), \pi_p^{-1}(k^{Nr-1}), \dots, \pi_p^{-1}(k^2), k^1\}$，同时使用 S 盒的逆代换$\pi_s^{-1}$。

2.算法优化

（1）使用“快速读入”加快数据读取速度。

（2）题目中的 S 盒与 P 盒已经固定，在代换和置换操作时直接使用表达式进行“打表”而避免使用循环语句，可以提高计算速度。

1.3 实验结果分析

表 1.1 SPN 实现的 OJ 测试时间

ID	1	2	3	4	5	6	7
Time	1ms	2ms	0ms	6ms	30ms	257ms	1025ms

2　线性分析

2.1 实验要求

内容：根据已知明密文对分析原始 SPN 的密钥。
要求：
（1）实现教材所给算法。
（2）能根据所给 8000 对明密文对分析对应位置密钥。
（3）分析出所有 32 比特密钥。

2.2 实验过程

算法分析：

线性分析的算法思路参考教材，分析结果为获得最后一轮的 16 位子密钥，然后通过暴力枚举前 16 位从而获得 32 位正确密钥。

线性分析需要已知具有较大偏差绝对值的两个线性逼近。教材中提供了一条输入与第 5、7、8 位（记为 0x0b00）相关，输出与 6、8、14、16 位记为（0x0505）相关即与第 2 和第 4 个 S 盒相关的线性逼近，可用来计算出最后一轮子密钥的第 2 和第 4 个 16 进制位。

通过寻找，可找出几条可以覆盖到第 1 和第 3 个 S 盒（即能计算出最后一轮轮密钥第 1 和第 3 个 16 进制位）的具有较大偏差绝对值的线性逼近。经过筛选，找到了输入与 0x0900 相关输出与 0x5555 相关，输入与 0x0c00 相关输出与 0x5550 相关，输入与 0x0f00 相关输出与 0x5550 相关，输入与 0x0090 相关输出与 0xeeee 相关共 4 条线性逼近可以通过测试题。此处选择以输入与 0x0f00 相关，输出与 0x5550 相关的线性逼近进行分析。

需要注意的是，以上找出的线性逼近的输出不仅涉及最后一轮轮密钥的第 1 和第 3 个 16 进制位，还包括第 2 或第 4 个 16 进制位。需要将教材提供的第一条线性逼近计算所获得的最后一轮轮密钥的第 2 和第 4 个 16 进制位作为已知，代入到第二条线性逼近的计算中，所得的第 1 和 3 个 16 进制位的字符才更接近正确密钥。

此外，由于线性分析是概率分析，第一条线性逼近计算出的计数最大的第 2 位和第 4 位（16 进制）有一定可能不为正确轮密钥对应位的值，因此需要尝试其他计数较大的值。此处采用一个 16×16 的数组用来记录候选子密钥的所有第 2 位和第 4 位及其计数值，并按照计数值由大到小排序。当通过计数最大的值的候

选子密钥的 2、4 位，代入第二条线性逼近计算出最后一轮子密钥的 1、3 位而暴力枚举而不能计算出正确密钥时，则第一条链计数最大的并不为正确密钥，则依次从计数第二大继续尝试计算，直到找到正确密钥。

具体算法描述如图 2.1.

Algorithm: LinerAnalysis

Input : pc is set of plain ciphertext pairs, T is number of plain ciphertext pairs, π_s^{-1}
Output: $k16b$ is last 16 bits of correct key
// First liner chain
for $(L_2, L_4) \leftarrow (0,0)$ **to** (F, F) **do**
  $Count24[L_2, L_4] \leftarrow 0$;
end
foreach $(x, y) \in pc$ **do**
  for $(L_2, L_4) \leftarrow (0,0)$ **to** (F, F) **do**
    $v_{<2>} \leftarrow L_2 \oplus y_{<2>}, v_{<4>} \leftarrow L_4 \oplus y_{<4>}$;
    $u_{<2>} \leftarrow \pi_s^{-1}(v_{<2>}), u_{<4>} \leftarrow \pi_s^{-1}(v_{<4>})$;
    $z24 \leftarrow x_5 \oplus x_7 \oplus x_8 \oplus u_8 \oplus u_{14} \oplus u_{16}$;
    if $z == 0$ **then**
      $Count24[L_2, L_4] + +$;
    end
  end
end
$no \leftarrow 0$;
for $(L_2, L_4) \leftarrow (0,0)$ **to** (F, F) **do**
  $maxCount \leftarrow |Count24[L_2, L_4] - \frac{T}{2}|$;
  $keyInfo[no] \leftarrow (L_2, L_4, maxCount)$;
  $no + +$;
end
$sort(keyInfo, keyInfo + 256)$;
// Start with the maximum value of maxCount
foreach $(maxkey2, maxkey4, maxCount) \in keyInfo$ **do**
  // Second liner chain
  for $(L_1, L_3) \leftarrow (0,0)$ **to** (F, F) **do**
    $Count13[L_1, L_3] \leftarrow 0$;
  end
  foreach $(x, y) \in pc$ **do**
    $v_{<2>} \leftarrow maxkey2 \oplus y_{<2>}, u_{<2>} \leftarrow \pi_s^{-1}(v_{<2>})$;
    for $(L_1, L_3) \leftarrow (0,0)$ **to** (F, F) **do**
      $v_{<1>} \leftarrow L_1 \oplus y_{<1>}, v_{<3>} \leftarrow L_3 \oplus y_{<3>}$;
      $u_{<1>} \leftarrow \pi_s^{-1}(v_{<1>}), u_{<3>} \leftarrow \pi_s^{-1}(v_{<3>})$;
      $z13 \leftarrow x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus u_2 \oplus u_4 \oplus u_6 \oplus u_8 \oplus u_{10} \oplus u_{12}$;
      if $z13 == 0$ **then**
        $Count13[L_1, L_3] + +$;
      end
    end
  end
  $max13 \leftarrow -1$;
  for $(L_1, L_3) \leftarrow (0,0)$ **to** (F, F) **do**
    $Count13[L_1, L_3] \leftarrow |Count13 - \frac{T}{2}|$;
    if $Count13[L_1, L_3] > max13$ **then**
      $max13 \leftarrow Count13[L_1, L_3]$;
      $maxkey1 \leftarrow L_1, maxkey3 \leftarrow L_3$;
    end
  end
  // Force enumerates the first 16 bits
  $k16b \leftarrow (maxkey1 << 12)|(maxkey2 << 8)|(maxkey3 << 4)|maxkey4$;
  if $canForceEnum(k16b)$ **then**
    // can enum to get first 16bits of correct key
    return $k16b$;
  end
end
return 0;

图 2.1 线性分析算法

2.3 实验结果分析

表 2.1 线性分析的 OJ 测试时间

ID	1	2	3	4	5	6	7	8	9	10
Time	105ms	254ms	340ms	493ms	707ms	714ms	799ms	1078ms	1105ms	1236ms

3　差分分析

3.1 实验要求

内容：根据已知明密文对，选择明密文分析原始 SPN 的密钥。
要求：
（1）实现教材所给算法。
（2）能根据所给明密文对分析对应位置密钥。
（3）分析出所有 32 比特密钥。

3.2 实验过程

1.算法分析：

差分分析的算法思路参考教材，分析结果为获得最后一轮的 16 位子密钥，然后通过暴力枚举前 16 位从而获得 32 位正确密钥。

差分分析需要已知具有较大差分特征的两个差分链。教材中提供了一条输入 x’=0x0b00，输出$u^{4'}$=0x0606,即与第 2 和第 4 个 S 盒相关的差分特征高达 0.02637 的差分链，可用来计算出最后一轮轮密钥的第 2 和第 4 个 16 进制位。

通过寻找，可找出数条可以覆盖到第 1 和第 3 个 S 盒（即能计算出最后一轮轮密钥第 1 和第 3 个 16 进制位）的具有较大差分特征的差分链。经过筛选，找到了几条差分特征为 0.008789(已经比较大)且输入、输出的异或值较为规整的差分链，如输入 x’=0x00a0，输出$u^{4'}$=0x5055；输入 x’=0x0a00，输出$u^{4'}$=0x6066；输入 x’=0x00a0，输出$u^{4'}$=0x5055；输入 x’=0x00a0，输出$u^{4'}$=0x5055 等均可通过测试题。需要注意的是，第二条差分链的差分特征小于第一条链，因此需要更多的明密文对。

与线性分析相同，差分分析同样需要将教材提供的第一条差分链计算所获得的最后一轮轮密钥的第 2、4 个 16 进制位作为已知，代入到第二条差分链的计算中，所得的第 1、3 个 16 进制位的字符才更接近正确密钥。

具体算法描述如图 3.1.

2.算法优化

16 比特的密文并不以整体的形式存储到 1 个变量中,而是作为 4 个 16 进制字符存储到一个长度为 4 的数组中。这样可以在差分分析的计算中直接使用$y_{<1>}$、

$y_{<2>}$、$y_{<3>}$和$y_{<4>}$，而不用使用密文整体的值移位求得，可提高算法速度。

Algorithm: DifferentialAnalysis

Input : $PC24$ and $PC13$ are two 4-tuple (x, x^*, y, y^8) sets of two differential chains, $T24$ and $T13$ are size of PC24 and PC13, π_s^{-1},

Output: $k16b$ is last 16 bits of correct key

// First differential chain

for $(L_2, L_4) \leftarrow (0,0)$ **to** (F, F) **do**
  $Count24[L_2,\ L_4] \leftarrow 0$;
end
foreach $(x, x^*, y, y^*) \in PC24$ **do**
  if $y_{<1>} == y^*_{<1>}$ ***and*** $y_{<3>} == y^*_{<3>}$ **then**
    for $(L_2, L_4) \leftarrow (0,0)$ **to** (F, F) **do**
      $v_{<2>} \leftarrow L_2 \oplus y_{<2>},\ v_{<4>} \leftarrow L_4 \oplus y_{<4>}$;
      $u_{<2>} \leftarrow \pi_s^{-1}(v_{<2>}),\ u_{<4>} \leftarrow \pi_s^{-1}(v_{<4>})$;
      $(v_{<2>})^* \leftarrow L_2 \oplus (y_{<2>})^*,\ (v_{<4>})^* \leftarrow L_4 \oplus (y_{<4>})^*$;
      $(u_{<2>})^* \leftarrow \pi_s^{-1}((v_{<2>})^*),\ (u_{<4>})^* \leftarrow \pi_s^{-1}((v_{<4>})^*)$;
      $(u_{<2>})' = u_{<2>} \oplus (u_{<2>})^*,\ (u_{<4>})' = u_{<4>} \oplus (u_{<4>})^*$;
      if $(u_{<2>})' == 0110$ ***and*** $(u_{<4>})' == 0110$ **then**
        $Count24[L_2, L_4] + +$;
      end
    end
  end
end
$max24 \leftarrow -1$;
for $(L_2, L_4) \leftarrow (0,0)$ **to** $(F, F))$ **do**
  if $Count24[L_2, L_4] > max24$ **then**
    $max24 \leftarrow Count24[L_2, L_4]$; $maxkey2 \leftarrow L_2,\ maxkey4 \leftarrow L_4$;
  end
end

// Second differential chain

for $(L_1, L_3) \leftarrow (0,0)$ **to** (F, F) **do**
  $Count13[L_1, L_3] \leftarrow 0$;
end
foreach $(x, x^*, y, y^*) \in PC13$ **do**
  if $y_{<2>} == y^*_{<2>}$ **then**
    $v_{<4>} \leftarrow maxkey4 \oplus y_{<4>},\ u_{<4>} \leftarrow \pi_s^{-1}(v_{<4>})$;
    $(v_{<4>})^* \leftarrow maxkey4 \oplus (y_{<4>})^*,\ (u_{<4>})^* \leftarrow \pi_s^{-1}((v_{<4>})^*)$;
    $(u_{<2>})' = u_{<2>} \oplus (u_{<2>})^*$;
    if $u_{<4>} == 0101$ **then**
      for $(L_1, L_3) \leftarrow (0,0)$ **to** (F, F) **do**
        $v_{<1>} \leftarrow L_1 \oplus y_{<1>},\ v_{<3>} \leftarrow L_3 \oplus y_{<3>}$;
        $u_{<1>} \leftarrow \pi_s^{-1}(v_{<1>}),\ u_{<3>} \leftarrow \pi_s^{-1}(v_{<3>})$;
        $(v_{<1>})^* \leftarrow L_1 \oplus (y_{<1>})^*,\ (v_{<3>})^* \leftarrow L_3 \oplus (y_{<3>})^*$;
        $(u_{<1>})^* \leftarrow \pi_s^{-1}((v_{<1>})^*),\ (u_{<3>})^* \leftarrow \pi_s^{-1}((v_{<3>})^*)$;
        $(u_{<1>})' = u_{<1>} \oplus (u_{<1>})^*,\ (u_{<3>})' = u_{<3>} \oplus (u_{<3>})^*$;
        if $(u_{<1>})' == 0101$ ***and*** $(u_{<4>})' == 0110$ **then**
          $Count13[L_1,\ L_3] + +$;
        end
      end
    end
  end
end
$max13 \leftarrow -1$;
for $(L_1, L_3) \leftarrow (0,0)$ **to** $(F, F))$ **do**
  if $Count13[L_1, L_3] > max13$ **then**
    $max13 \leftarrow Count13[L_1, L_3]$; $maxkey1 \leftarrow L_1, maxkey3 \leftarrow L_3$;
  end
end
$k16b \leftarrow (maxkey1 << 12)|(maxkey2 << 8)|(maxkey3 << 4)|maxkey4$;
if $canForceEnum(k16b)$ **then**
  // can enum to get first 16bits of correct key
  return $k16b$;
end
return 0;

图 3.1 差分分析算法

3.3 实验结果分析

表 3.1 差分分析的 OJ 测试时间

ID	1	2	3	4	5	6	7
Time	7ms	2ms	8ms	22ms	122ms	250ms	1064ms

4　SPN 增强

4.1 实验要求

内容：对原始 SPN 进行改进。

要求：

（1）选择合适的密钥长度、分组长度、S 盒、P 置换、轮数。

（2）效率较高

（3）输出达到随机数检测标准

4.2 实验过程

算法分析：

保持 SPN 的基本结构不变，在以下方面进行了改进：

（1）SPN 分组长度即一次处理字节数增加至 64 位即 8 字节，加密轮数增至 8 轮。

（2）密钥：主密钥字节数增至 8 字节即 64 位，通过密钥编排算法得到 9 个子密钥。

（3）使用 8 个长度为 8 的 S 盒和 1 个长度为 64 的 P 盒。

（4） 采用 CBC 的分组密码工作模式进行加密，使用随机数生成器生成 64 位初始向量，每次都使用当前明文和上一次密文异或的结果进行加密。

具体算法描述如图 4.1.

Algorithm: SPN Plus with CBC

Input : key is 128bits key, π_s, π_p, $InputBytes$, $SpnBytes$

```
// generate subkeys and save to K
arrangeEncKeyPlus(key, K);
// randomly generate initial vector IV
IV ← randULL();
for i ← 0 to InputBytes/SpnBytes do
    fread(&x, SpnBytes, 1, stdin);
    // CBC mode
    x ← x ⊕ IV;
    // use SPNPlus to encrypt
    y ← SPNPlus(x, K, π_s);
    fwrite(&y, SpnBytes, 1, stdin);
    // update IV
    IV ← y;
end
```

图 4.1 差分分析算法

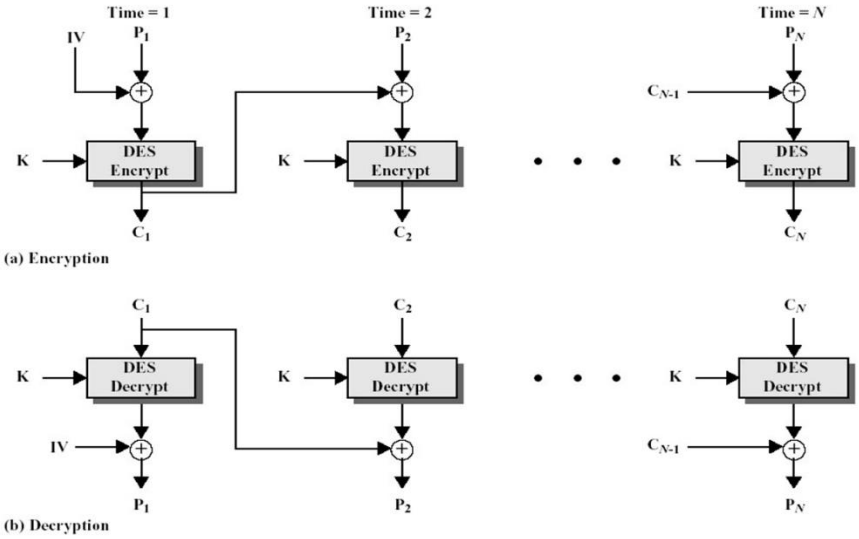

图 4.2 CBC 工作模式示意图

4.3 实验结果分析

表 4.1 SPN 增强的 OJ 测试时间

ID	1	2	3	4	5	6	7
Time	520ms	510ms	509ms	523ms	505ms	508ms	517ms

5　RSA 参数计算

5.1 实验要求

内容：求 RSA 参数 d。

要求：

（1）利用加法、减法、乘法、模运算等基本运算。

（2）自己实现求逆,gcd

（3）简单检查参数的合法性

5.2 实验过程

1.相关算法

（1）求最大公约数：使用辗转相除法，具体算法描述如图 5.1。

Algorithm: gcd

Input : a, b

Output: g is the greatest common divisor of a and b

```
r ← a;
while r ≠ 0 do
    r ← a%b, a ← b, b ← r;
end
g ← a;
return g;
```

图 5.1 求最大公约数算法

（2）模逆运算：利用扩展欧几里得算法求$a(mod\ b)$的逆元 c：先做辗转相除，当 a、b 互素时，最后一步得到的余数为 1，再从 1 触发对前面得到的所有除法算式进行变性，将余数用除数和被除数表示，最终可将 1 表示为 a 与 b 的一种线性组合，即 ax+by=1，从而 x 就是 a 模 b 的乘法逆元。具体算法描述如图 5.2。

Algorithm: modInv

Input : a, N

Output: $a^{-1}(modN)$, or 0 if no inverse element

```
x1 ← 1, x2 ← 0, x3 ← N, y1 ← 0, y2 ← 1, y3 ← a;
if a > N then
    a ← a%N;
end
while y3 ≠ 0 do
    q ← x3/y3;
    t1 ← x1 − q * y1, t2 ← x2 − q * y2, t3 ← x3 − q * y3;
    x1 ← y1, x2 ← y2, x3 ← y3;
    y1 ← t1, y2 ← t2, y3 ← t3;
end
if x3 == 1 then
    return x2 > 0?x2 : x2 + N;
end
return 0;
```

图 5.2 模逆运算

（3）素性检测：采用 Miller-Rabin 算法，具体算法描述如图 5.3。

Algorithm: Miller-Rabin

Input : n
Output: *true* or *false*: whether n is prime

$n-1=2^s t$, which $t\%2==1$;
$a=randInt()$, which $1 \le a \le n-1$;
$b \equiv a^t (mod\ n)$;
if $b \equiv 1 (mod\ n)$ **then**
  return *true*;
end
for $i \leftarrow 0$ **to** $s-1$ **do**
  if $b \equiv -1 (mod\ n)$ **then**
    return *true*;
  else
    $b \leftarrow b^2 (mod\ n)$;
  end
end
return *false*;

图 5.3 Miller-Rabin 算法

（4）模乘运算：使用平方乘算法，具体算法见下一节。

2.参数合法性要求

（1）p 和 q 均为素数

（2）$\varphi(N)=(p-1)(q-1)$与 e 互素，即$\gcd(\varphi(N),\ e)==1$.

（3）p 与 q 的差值不能过小，此处令$|p-q|>10000$。

（4）n 的素因子 p 和 p-1 只有小的因子，此处令$\gcd(p-1,\ q-1)<5000$.

5.3 实验结果分析

表 5.1 RSA 参数计算的 OJ 测试时间

ID	1	2	3	4	5
Time	7ms	15ms	26ms	171ms	3ms

6　模重复平方

6.1 实验要求

内容：正确计算$a^e(mod\ N)$。

要求：（1）利用加法、减法、乘法、模运算等基本运算。

（2）自己实现 expmod(a,e,n)求 RSA 参数 d。

6.2 实验过程

1.算法分析

使用平方-乘算法：$a^e(mod\ N)$计算公式为：

$$e = \sum_{i=0}^{l-1} e_i 2^i,\ e_i \in \{0,1\}, e_{i-1} = 1.$$

$$a^e = a^{\sum_{i=0}^{l-1} e_i 2^i} = ((\dots(1 \times a^{e_{l-1}})^2 \times \dots)^2 \times a^{e_1})^2 \times a^{e_0}$$

具体算法描述如图 6.1。

Algorithm: Square-Multiply

Input : a, e, N
Output: $a^e(mod\ N)$

```
1 z ← 1;
2 for i ← l − 1 downto 0 do
3 |   z ← z^2(mod N);
4 |   if e_i == 1 then
5 |   |   z ← (z × a)(mod N);
6 |   end
7 end
8 return z;
```

图 6.1 平方-乘算法

2.算法优化

平方-乘算法算法中，e 需要依次获取其二进制的高位到低位，此处不选择通过移位运算得到 e 的每一个二进制位，因为对于较大的数字 e 而言每次移位运算都比较耗时；而选择使用 gmp 库中的函数 mpz_get_str()来将 e 转换为二进制 01 组成的字符串，之后再使用下标读取其每一个二进制位，这样使得读取 e 每一位的速度得到有效提升。

6.3 实验结果分析

表 6.1 模重复平方的 0J 测试时间

ID	1	2	3	4	5	6
Time	158ms	236ms	395ms	478ms	346ms	487ms

7　中国剩余定理

7.1 实验要求

内容：正确计算$c^d(mod\ pq)$。

要求：

（1）利用 V 中的求逆运算从加密密钥 e 计算解密密钥 d。

（2）利用 VI 中实现的模幂运算和中国剩余定理计算$c^d(mod\ pq)$。

7.2 实验过程

算法分析：

对于 RSA 算法，解密方可以使用中国剩余定理加速解密：对于解密方而言，p、q 和 d 已知，因此可以事先计算好$p^{-1}(mod\ q)$ 、$q^{-1}(mod\ p)$、$d(mod\ p-1)$以及$d(mod\ q-1)$等值并存储，解密时便可直接使用，利用如下公式即可对密文c进行快速解密得到明文$m \equiv c^d(mod\ pq)$：

$$m_1 \equiv c^{d(mod\ p-1)}(mod\ p)$$

$$m_2 \equiv c^{d(mod\ q-1)}(mod\ q)$$

$$m \equiv m_1 q q^{-1}(mod\ p) + m_1 p p^{-1}(mod\ q)(mod\ pq)$$

7.3 实验结果分析

表 7.1 中国剩余定理的 0J 测试时间

ID	1	2	3	4	5	6	7	8	9	10
Time	7ms	18ms	36ms	68ms	115ms	138ms	248ms	302ms	417ms	984ms

8 Montgomery

8.1 实验要求

内容：正确计算$c^d(mod\ pq)$。

要求：

（1）实现 Montgomery 算法。

（2）Montgomery+中国剩余定理。

8.2 实验过程

1.算法分析：

Montgomery 算法：不用除法完成大整数模乘运算$ab(mod\ N)$。基本流程如下：

（1） $a \to A = Mont(a), b \to B = Mont(b)$

（2） $C = MontMult(A, B)$

（3） $ab \bmod N = MontInv(C)$

其中，Montgomery 变换和 Montgomery 乘法的算法具体描述如图 8.1 和 8.2。

Algorithm: Mont

Input : a, N
Output: $A \equiv aR(mod\ N)$

$d = 2^{32}$;
// $d = 2^{32}$ *or* 2^{64},which depends on the bits of computer
N is $k = 32n$ bits number, which $N\%2 == 1$;
$IN \equiv -N^{-1}(mod\ 2^{32})$;
$R = d^n > N$, which $gcd(R, N) == 1,\ a, b \in Z_N$;
$A \leftarrow (a << k)\%N$; // $A \equiv aR \equiv ad^n \equiv a2^{32n} \equiv a2^k(mod\ \ N)$
return A;

图 8.1 Montgomery 变换算法

Algorithm: MontMult

Input : A=Mont(a), B=Mont(b), N, $IN \equiv -N^{-1}(mod\ 2^{32})$
Output: $T \equiv ABR^{-1}(mod\ N)$

$T \leftarrow AB$;
for $i \leftarrow 1$ **to** n **do**
  $T = (0t_{2n-1}t_{2n-2}...t_1t_0)$;
  $T' \leftarrow T + N \times ((t_0 \times IN)mod\ 2^{32})$;
  $T \leftarrow T' >> 32$;
end
if $T > N$ **then**
  // $T = (0ct'_{n-1}t'_{n-2}...t'_0)$
  return $T - N$;
end
return T; // $T = (t'_{n-1}t'_{n-2}...t'_0)$

图 8.2 Montgomery 乘法算法

Montgomery 逆变换可用 MontMult 等价表示：

$$MontInv(a) = MontMult(a, 1)$$

将 Montgomery 算法与平方-乘算法相结合，可实现$a^e(mod\ N)$的快速模幂运算，算法具体描述如图 8.3.

Algorithm: Montgomery-SquareMult

Input : a, e, N
Output: $a^e(mod\ N)$

```
X ← Mont(1);
A ← Mont(a);
while e ≠ 0 do
    if e ≡ 1(mod 2) then
        X ← MontMult(X, A);
    end
    A ← MontMult(A, A);
    e ← e >> 1;
end
return MontInv(X);
```

图 8.3 Montgomery 快速模幂算法

2.算法优化

Montgomery 快速模幂算法中，需要多次调用 MontMult 函数，使用 gmp 库中用汇编代码编写的底层运算函数代替 Montmult 函数中大整数的普通运算函数，可以大幅度提高算法执行速度。

8.3 实验结果分析

表 8.1 Montgomery 的 OJ 测试时间

ID	1	2	3	4	5	6	7	8	9	10
Time	8ms	7ms	31ms	39ms	83ms	104ms	173ms	303ms	388ms	877ms

9 PKCS7

9.1 实验要求

PKCS#7 是 PKI 中用于消息加密的语法标准。可以用于给拥有公钥的用户发加密邮件、传送加密文件等。

现在已知一个可信的根 CA 的公钥证书，以及 PKCS#7 格式加密消息的接收者用户 B 的私钥。要求解开 PKCS#7 包装，获取明文消息。

9.2 实验过程

对于加密的消息，其中包括加密的明文部分和数字签名部分。首先通过接受者的私钥获取解密后的信息，然后通过已知的根 CA 证书获取发送者的证书链，再通过根 CA 的证书对所有签名进行验证，若签名均验证成功则明文消息是可信的未修改的，反之存在签名验证失败则明文消息有错误。

算法实现中使用了 GmSSL 库中提供的相关函数，如 PKCS7_dataDecode()用于解码 PKCS#7 的消息，sk_PKCS7_SIGNER_INFO_value 用于获取 PKCS#7 中签名者信息，PKCS7_dataVerify()用于签名验证等。

具体算法实现如图 9.1.

Algorithm: PKCS7

Input : $cacert$ is certificate of root CA, $pkeyB$ is private key of user B, $p7$ is PKCS#7 data

Output: P is plaintext, or "$Error$" if there is an error

```
// decode PKCS#7 to get the original data
p7Bio = PKCS7_dataDecode(p7, pkeyB);
sk = PKCS7_get_signer_info(p7); // get stack of signer info
foreach signInfo ∈ sk do
    // verify signature
    if PKCS7_dataVerify(p7, p7Bio, signInfo) ≤ 0 then
        return "Error";
    end
end
pLen = BIO_read(p7Bio, P);
P ← P[: pLen];
return P;
```

图 9.1 PKCS7 解密及签名验证算法

9.3 实验结果分析

表 9.1 PKCS7 的 OJ 测试时间

ID	1	2	3	4	5
Time	3ms	3ms	4ms	3ms	2ms

10 彩虹表

10.1 实验要求

已知 SHA1 彩虹表的 100 个 R 函数，现在有一些彩虹表的链头和链尾，每条链从链头开始，依次调用了 100000 次 SHA1 和 R 函数得到链尾。

要求从这些链中找到 SHA1 值对应的口令。

10.2 实验过程

1.算法分析

假设彩虹表链长上界为 n，即从链头开始依次调用 n-1 次 SHA1 和 k 个 R 函数得到链尾，总共可计算得到 n 个口令。

由待查询的 SHA1 值开始，分别从不同的 R 函数R_i开始依次使用 R 函数和 SHA1 函数计算口令。若 SHA1 值为 H 的口令是彩虹表中可计算的口令中的一个，则某次由R_k计算所得的口令应与彩虹表中某个链的链尾口令相同，再由该彩虹链从链头计算生成后续口令，若口令 H 在该条链中，则可以在链中找到与待查 SHA1 值相同的哈希值；若 SHA1 值为 H 的口令不在彩虹表中，则一定无法查询出其对应口令。具体算法描述如图 10.1。

Algorithm: FindSha1Pwd: find the password of sha1 by rainbow table

Input : $sha1$, m rainbow chains $(head, tail)$ of length n in $Rainbow$, k functions R
Output: pwd is the password of $sha1$, or $None$

```
for i ← 0 to k − 1 do
    for j ← 0 to n − 2 do
        idx ← (i + j)%k + 1;
        P ← R(H, idx);
        if idx == k then
            // compare with tail of chains
            foreach (head, tail) ∈ Rainbow do
                if P == tail then
                    // found P is the same as the chain tail
                    tmpP ← head;
                    no ← 1;
                    for l ← 1 to n do
                        // generate rainbow chain from head
                        tmpH ← SHA1(tmpP);
                        if sha1 == tmpH then
                            // found a hash value is the same as sha1
                            pwd ← tmpH;
                            return pwd;
                        end
                        P ← R(tmpH, no);
                        no ← no < 100?no + 1 : no;
                    end
                    break;
                end
            end
        end
        H ← SHA1(P);
    end
end
return None;
```

图 10.1 利用彩虹表寻找口令算法

10.3 实验结果分析

表 10.1 彩虹表的 0J 测试时间

ID	1	2	3	4	5
Time	7627ms	4391ms	22125ms	18624ms	19725ms

实验总结

本次课程设计对我而言收获颇丰。

课设中，我完成了从 SPN、线性分析到模重复平方、Montgomery 算法再到彩虹表等共 10 道与密码学相关的题目，巩固和加深了我对密码学中相关概念的理解与认识。同时这个过程中，需要将密码学中的知识从理论转换成了具体的代码，也锻炼了我的编程能力。比如在完成“差分分析”这一题目时，问题的难点之一在于在 SPN 中找到一条合适的差分特征较大的差分链，这一过程仅凭人工去找效率低下，因此我需要认真分析差分链的寻找规则，通过编写了一个可以找出所有覆盖给定 S 盒的差分链的程序，来解决找链的问题。

同时，在将课上的理论知识运用到真正实际的做题和解决问题时，还需要结合实际情况进一步的调整与改进。比如“线性分析”破解密钥时，不仅需要两条线性逼近，还需要将第一条链得出的结果带入第二条链，使得第二条链计算的结果更加准确；此外还要考虑到破解方法本身就是一个“概率”方法，计数最大值并不一定对应着正确密钥，还需要尝试多个计数较大的值，以便能够找到正确密码。再比如 Montgomery 算法，除了算法本身的实现以外，还需要考虑如何更好地结合 gmp 库的底层函数来优化算法，以实现更快速的计算。

除了密码学的相关知识本身，在完成课程设计的过程中使用了 gmp 库和 GmSSL 库，这个过程需要我通过相关文档进行库的编译安装、学习其函数、结构的用法，也提高了我查阅资料、自学的能力。

此外，由于本次课程设计使用“OJ”进行题目提交，这对我们的代码有了更高的要求，需要考虑题目的时间和空间限制。这要求我在编写程序时更加严谨缜密，比如需要考虑是否可以减少函数的调用、是否可以通过开辟数组增大空间开销以提高算法性能、循环语句是否可以换成简单的表达式、对于大批量数据读入是否可以用“快读”完成、对于同样的计算是否有更优的算法方案等等。在解决一个个测试用例时，这也让我认识到代码优化的重要性以及如何考虑代码的优化。同时，每个题目都有多个测试用例，需要考虑多种情况，也要求我在编写的程序对于一些条件的约束需要更加准确，提高了我的逻辑思维和问题分析能力。

总体而言，在课程设计解决多个题目的过程中，不仅加深了我对密码学课程中一些重点知识的理解，同时在用代码实现过程中也提高了我的编程能力，锻炼了我快速学习、查找资料、代码优化等多方面的能力，有很大收获。

密码学课程设计总评成绩

完成情况	课程报告		总评成绩
（80%）	格式规范 （10%）	内容完整 （10%）	100%