武汉大学国家网络安全学院

密 码 学 实 验 报 告

学　　号　2021302181156

姓　　名　赵伯侯

实验名称　公钥密码 RSA

指导教师　何琨

一、 实验名称: 公钥密码 RSA

二、 实验目的及要求:

2.1 实验目的

(1) 掌握公钥密码的概念和基本工作方式

(2) 掌握 RSA 密码、ElGamal 密码和椭圆曲线密码的原理与算法

(3) 了解 RSA 密码、ElGamal 密码和椭圆曲线密码的安全性

(4) 了解 RSA 密码、ElGamal 密码和椭圆曲线密码的应用

2.2 实验要求

(1) 掌握 RSA 密码的实现方案

(2) 掌握 ElGamal 密码的实现方案

(3) 掌握椭圆曲线密码的实现方案

(4) 了解公钥算法实现中的相关优化算法

三、 实验设备环境及要求:

Windows 操作系统，python 高级语言开发环境

四、 实验内容与步骤:

4.1 RSA 密码

4.1.1 RSA 初始化算法

在 RSA 算法开始加密之前，需要先计算公钥 <n,e> 和私钥 <p,q,d,$\varphi(n)$> 的值，该步骤的代码如下所示

```
def init(p=0, q=0, e=0):
    """计算初始参数"""
    p, q, e, n, d, fn = keyGenerate(1000, 10000, p, q, e)  # 密钥生成
    # 更改keyGenerate函数的两个参数，可以改变生成素数的位数大小。

    print("公钥<n,e>为:" + str(n) + " " + str(e))
    print("私钥<p,q,d,fn>为: " + str(p) + " " + str(q) + " " + str(d) + " " +
        str(fn))
    return e, n, d
```

代码 1: RSA 初始化

```
def keyGenerate(lower, upper, p, q, e):
    """生成密钥"""
    # 在p,q没有给出时自己生成
    if p == 0:
        p = findPrime(lower, upper)
        q = findPrime(lower, upper)

    # 求fn的值
    n = p * q
    fn = (p - 1) * (q - 1)
    # 求e的值,没有给出就自己生成
    if e == 0:
        e = selectE(fn)
```

```
    # 求d的值
    temp = e_gcd(e, fn)  # 欧几里得算法求逆元
    d = temp[1]
    if d < 0:  # 由于e和fn互素故一定存在逆元
        d = d + fn  # 保证d为正数

    return p, q, e, n, d, fn
```

代码 2: RSA 初始化

（1）首先随机地选择两个大素数 p 和 q，而且保密；

（2）计算 n=pq，将 n 公开；

（3）计算 $\varphi(n) = (p-1)(q-1)$, 将 $\varphi(n)$ 保密

（4）随机地选取一个正整数 e，满足 $1 < e < \varphi(n)$ 且 $(e, \varphi(n)) = 1$，将 e 公开；

（5）根据 $ed = 1 mod\, \varphi(n)$，求出 d，并对 d 保密；

4.1.2 RSA 加密算法

RSA 算法的加密程序如下所示。

```
def RSA_encode(M, e, n):
    """RSA加密"""
    c = fastExpMod(int(M), e, n)  # 加密  c为密文 m^e mod n
    return c
```

代码 3: RSA 加密

该程序直接进行运算 $C = M^e mod\, n$ 得到密文

4.1.3 RSA 解密算法

RSA 算法的解密程序如下所示。

```
def RSA_decode(c, d, n):
    """RSA解密"""
    M = fastExpMod(c, d, n)  # 解密 c^d mod n
    return M
```

代码 4: RSA 解密

该程序直接进行运算 $M = C^d mod\ n$ 得到明文

4.1.4 求逆算法

该算法要实现计算 $a^{-1} mod\ p$ 的计算

令 R1=p,R2=a, 计算

$R_1 = R_2 * Q_2 + R_3$

$R_2 = R_3 * Q_3 + R_4$

.....

$R_{n-1} = R_n * Q_n + 1$

然后令 $S_0 = 0, S_1 = 1$ 依次计算 $S_i = S_{i-2} - S_{i-1} * Q_i$

最终计算得到的 S_n 即为 a 的逆元 $a^{-1} mod\ p$

实现求逆算法的程序如下所示

```
def e_gcd(a, b):
    """计算模的逆元"""
    if b == 0:
        return a, 1, 0
    g, x, y = e_gcd(b, a % b)
    return g, y, x - a // b * y
```

代码 5: 求逆元算法

4.1.5 快速乘方运算

该算法通过模重复平方算法实现 $c = a^n mod\ m$ 的计算

将 n 写成 2 进制的形式为 $n = n_0 + n_1 \times 2 + \cdots + n_{k-1} \times 2^{k-1}$

因此 $c = a^n mod\ m$ 也就可以写成

$a^n = a^{n_0} \times (a^2)^{n_1} \times \cdots \times (a^{2^{k-2}})^{n_{k-2}} \times (a^{2^{k-1}})^{n_{k-1}} (mod\ m)$ 的形式

由此可以大幅简化计算。

快速乘方运算的实现函数如下所示

```
1  def fastExpMod(b, n, m):
2      '''
3      return : b^n mod m
4      '''
5      result = 1
6      while n != 0:
7          if (n & 1) == 1:  # 按位与&操作
8              result = (result * b) % m
9          b = (b * b) % m
10         n = n >> 1  # 位数右移>>操作
11     return result
```

代码 6: 求逆元算法

4.1.6 实验（1）

令 p=3,q=11,d=7,m=5, 手工或编程计算密文 C，程序代码如下所示

```
1  def test1():
2      p = 3
3      q = 11
4      d = 7
5      m = 5
```

```
    fn = (p - 1) * (q - 1)
    temp = e_gcd(d, fn)  # 欧几里得算法求逆元
    e = temp[1]
    e, n, d = init(p, q, e)
    secret = RSA_encode(m, e, n)
    print("p=3,q=11,d=7,m=5的加密结果为：" + str(secret))
```

代码 7: 测试算法 1

将对应的参数在初始化程序中进行修改然后计算 e 的值之后进行加密得到的结果如下图所示

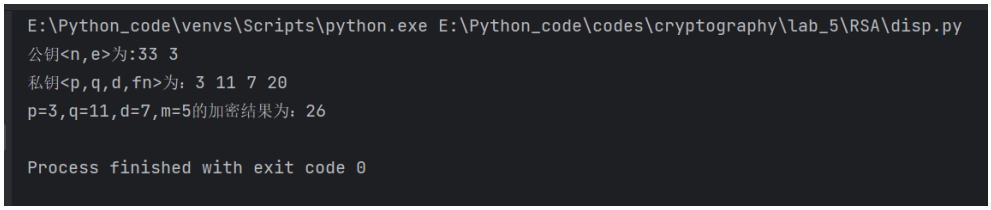

图 1: 测试算法 1 运行结果

4.1.7 实验（2）

设 RSA 密码的 e=3,n=33,C=9, 手工或编程计算明文 M 程序代码如下所示

```
def test2():
    fn = 2 * 10
    e = 3
    temp = e_gcd(e, fn)  # 欧几里得算法求逆元
    d = temp[1]
    if d < 0:  # 由于e和fn互素故一定存在逆元
        d = d + fn  # 保证d为正数
    result = RSA_decode(9, d, 33)
    print("e=3,n=33,C=9解密的明文为：" + str(result))
```

代码 8: 测试算法 2

已知 n 之后将 n 分解为 3*11, 可以得到 fn=2*10, 由此可以计算得到 d 的值，所以能够得到明文

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\RSA\RSA.py
e=3,n=33,C=9解密的明文为：27

Process finished with exit code 0
```

图 2: 测试算法 2 运行结果

4.1.8 实验（3）

令 p=17,q=11, e=7, 试计算 RSA 密码其余参数，进一步对于 m=88, 计算密文 C。程序代码如下所示

```
1  def test3():
2      p = 17
3      q = 11
4      e = 7
5      e, n, d = init(p, q, e)
6      m = 88
7      result = RSA_encode(m, e, n)
8      print("p=17,q=11, e=7,m=88时加密结果为" + str(result))
```

代码 9: 测试算法 3

首先调用 init 函数计算出 d，n 的值，然后再调用 RSA_encode 函数对明文 88 进行加密。其余参数和加密结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\RSA\disp.py
公钥<n,e>为:187 7
私钥<p,q,d,fn>为：17 11 23 160
p=17,q=11, e=7,m=88时加密结果为11

Process finished with exit code 0
```

图 3: 测试算法 3 运行结果

4.2 ELGamal 密码

4.2.1 密钥生成

用户随机地选取一个整数 d 作为自己的解密钥，然后计算 $y = a^d mod\, p$ 的值作为公开加密钥。

该过程的程序代码如下所示

```
def generate_key(p=0, a=0, d=0):
    """计算公钥y"""
    # 若没有给出p，需要自己生成p,a,d
    if p == 0:
        while True:
            q = sympy.randprime(10 ** 149, 10 ** 150 / 2 - 1)
            if sympy.isprime(q):
                p = 2 * q + 1
                if len(str(p)) == 150 and sympy.isprime(p):
                    break
        a = primitive_element(p, q)
        d = randint(2, p - 2)

    y = fastExpMod(a, d, p)
    return p, y, d, a
```

代码 10: 密钥生成

在程序中如果没有题前设定 p 的值，则需要自动生成一个 p，a 和随机数 d

4.2.2 加密

加密过程为：用户随机选定一个整数 k（1<k<p-1）, 首先计算 $U = y^k mod\ p$ 然后计算 $C_1 = \alpha^k mod\ p$ 与 $C_2 = UM\ mod\ p$ 将（C1,C2）作为加密得到的密文。

加密程序如下所示

```
def ELGamal_encode(M, p=0, a=0, d=0, k=0):
    y, d, p, a = generate_key(p, a, d)

    y = fastExpMod(a, d, p)
    print("计算得公钥y=" + str(y))
    print("    私钥d=" + str(d))
    if k == 0:
        k = randint(2, p - 1)
    U = fastExpMod(y, k, p)
    c1 = fastExpMod(a, k, p)
    UM = U * int(M)
    c2 = fastExpMod(UM, 1, p)

    return c1, c2, d, p
```

代码 11: ELGamal 加密

4.2.3 解密

解密过程为：首先计算 $V = C_1^d mod\ p$ 然后计算 $M = C_2 V^{-1} mod\ p$ 从而得到明文。

解密程序如下所示

```
def ELGamal_decode(c1, c2, d, p):
    V = fastExpMod(c1, d, p)
    V_1 = e_gcd(V, p)[1]
```

```
    C2V_1 = c2 * V_1
    M = fastExpMod(C2V_1, 1, p)
    return M
```

代码 12: ELGamal 解密

4.2.4 实验

设 p=19，m=17, 构造一个 ELGamal 密码，并用它对 m 加密

取 $\alpha = 2$, 密钥 d=765, 随机数 k=853 加密得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\ELGamal\ELGamal.py
明文为：17
计算得公钥y=18    私钥d=765
加密后的密文(c1,c2)为：14 2
解密后的明文为：17

Process finished with exit code 0
```

图 4: 测试算法 1 运行结果

4.2.5 实验（4）

设 p=5，m=3, 构造一个 ELGamal 密码，并用它对 m 加密取 $\alpha = 2$, 密钥 d=765, 随机数 k=853 加密得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\ELGamal\ELGamal.py
明文为：3
计算得公钥y=2    私钥d=765
加密后的密文(c1,c2)为：2 1
解密后的明文为：3

Process finished with exit code 0
```

图 5: 测试算法 2 运行结果

4.3 椭圆曲线密码

4.3.1 求椭圆曲线所有解点（实验 5）

编写求已知椭圆曲线和 p 的情况下所有的椭圆曲线的解点程序如下所示

```
def find_points_on_elliptic_curve(a, b, p):
    points = []
    for x in range(p):
        y_squared = (x ** 3 + a * x + b) % p
        for y in range(p):
            if (y * y) % p == y_squared:
                points.append((x, y))
    return points


def print_points_in_groups(points,  width=2):
    for i, point in enumerate(points):
        # 使用字符串格式化来确保对齐
        formatted_point = f"({point[0]:>{width}}, {point[1]:>{width}})"
        print(formatted_point, end=' ')
        if (i + 1) % 10 == 0:
            print()  # 每 group_size 个点后换行
    print()  # 在列表末尾换行


if __name__ == '__main__':
    a = 1  # 曲线参数 a
    b = 1  # 曲线参数 b
    p = 23  # 质数 p

```

```
26      points = find_points_on_elliptic_curve(a, b, p)
27      print("给出的椭圆曲线的所有解点为")
28      print_points_in_groups(points)
```

代码 13: ELGamal 解密

该程序使用穷举法首先遍历 x 的所有可能值，然后计算 y^2 接着，它再次遍历所有可能的 y 值，寻找使等式成立的 y。当找到这样的 y 时点（x,y）就是曲线上的一个解点。最后，程序返回曲线上的所有解点。

将给出的椭圆曲线和 p 的值带入程序得到的运行结果为:

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\SM2\椭圆曲线求解点.py
给出的椭圆曲线的所有解点为
( 0,  1) ( 0, 22) ( 1,  7) ( 1, 16) ( 3, 10) ( 3, 13) ( 4,  0) ( 5,  4) ( 5, 19) ( 6,  4)
( 6, 19) ( 7, 11) ( 7, 12) ( 9,  7) ( 9, 16) (11,  3) (11, 20) (12,  4) (12, 19) (13,  7)
(13, 16) (17,  3) (17, 20) (18,  3) (18, 20) (19,  5) (19, 18)

Process finished with exit code 0
```

图 6: 解点计算结果

4.3.2 SM2 加密

SM2 加密过程为：

（1）定义一个随机数 d 作为用户的私钥，将用户的公钥定义为椭圆曲线上的 P 点，P=dG，G（x,y）是基点

（2）用随机数发生器产生一个随机数 k

（3）计算椭圆曲线上的 C1 点 C1=kG=（x1,x2）

（4）计算椭圆曲线上的点 S=hP_b 的值，若 S 为无穷远点的话直接报错退出

（5）计算椭圆曲线点（x2,y2）=kP_b

（6）调用密钥派生函数 KDF 计算 t 的值，若计算出的 t 值为全 0 则返回（2）

（7）调用二进制位上的异或函数，计算 M 与 t 的抑或结果保存到 C2 中

（8）调用 SM3-Hash 密码杂凑函数根据 x2,M,y2 的拼接结果得到 C3 的值

（9）将 C1，C2,C3 拼接起来作为密文输出

SM2 加密算法的算法流程图如下图所示

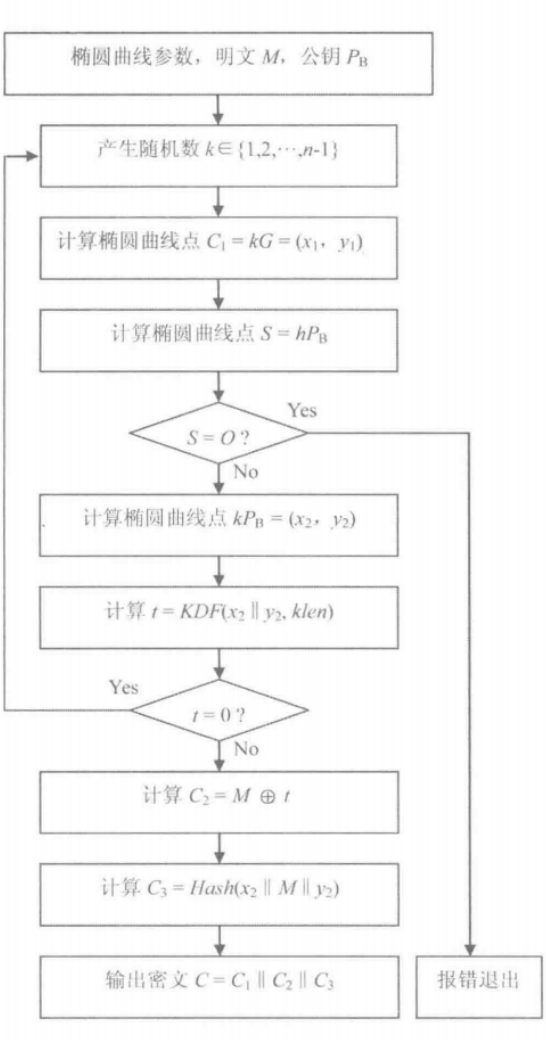

图 7: SM2 加密流程

实现该过程的 SM2 加密算法如下所示

```
1  def enSM2(message: str):
2      """SM2加密"""
3      plen = len(hex(p)[2:])
4      m = str_bit(message)
5      klen = len(m)
6
```

```
    while True:
        # k = 0
            x4C62EEFD6ECFC2B95B92FD6C3D9575148AFA17425546D49018E5388D49DD7B4F
        k = randint(1, n)
        while k == dB:
            k = randint(1, n)
        x2, y2 = mul_point(xB, yB, k, a, p)
        x2, y2 = '{:0256b}'.format(x2), '{:0256b}'.format(y2)
        t = kdf(x2 + y2, klen)
        if int(t, 2) != 0:
            break
    # C1计算
    x1, y1 = mul_point(gx, gy, k, a, p)
    x1 = int_hex(x1, plen)
    y1 = int_hex(y1, plen)

    c1 = '04' + x1 + y1
    c2 = xor_2(m, t, klen)
    c3 = Hash(hex(int(x2 + m + y2, 2))[2:])
    return c1, c2, c3
```

代码 14: SM2 加密

4.3.3 SM2 解密

SM2 解密过程为：

（1）验证 C1 是否满足椭圆曲线方程

（2）计算椭圆曲线点 S=hC1，若 S 为无穷远点则报错退出

（3）计算（x2,y2）=$d_B C_1$

（4）调用密钥派生函数 KDF 计算 t 的值，若计算出的 t 值为全 0 则报错

（5）调用 16 进制字符串异或函数计算 C2 与 t 按位异或的结果，得出明文

（6）调用 SM3-Hash 密码杂凑函数根据 x2, 计算得到的明文,y2 得出的结果与 C3 相比，若两者不相等则报错。反之得到的明文为正确的明文

SM2 解密算法的算法流程图如下图所示

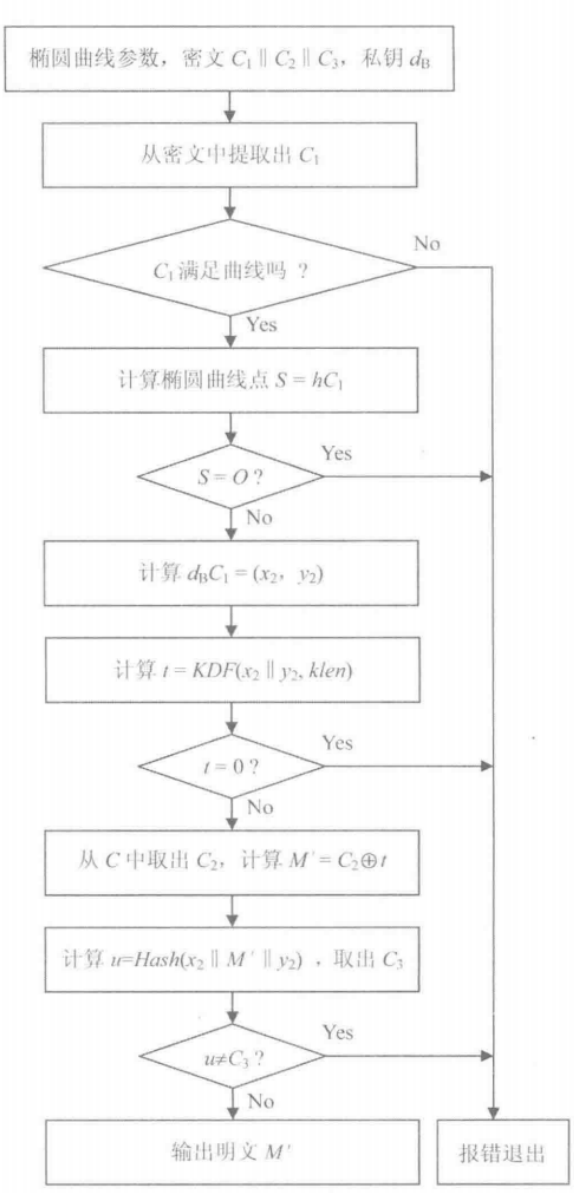

图 8: SM2 解密流程

实现该过程的 SM2 解密程序如下所示

```
1 def deSM2(c1, c2, c3, a, b, p):
2     c1 = c1[2:]
```

```
    x1, y1 = int(c1[:len(c1) // 2], 16), int(c1[len(c1) // 2:], 16)
    # 验证c1是否在椭圆曲线上
    if pow(y1, 2, p) != (pow(x1, 3, p) + a * x1 + b) % p:
        return False
    # 计算点S
    x2, y2 = mul_point(x1, y1, dB, a, p)
    x2, y2 = '{:0256b}'.format(x2), '{:0256b}'.format(y2)
    # 计算t，若t全0则退出
    klen = len(c2) * 4
    t = kdf(x2 + y2, klen)
    if int(t, 2) == 0:
        return False
    # 计算m‘
    m = xor_16(c2, t, klen)
    # hash计算u
    u = Hash(hex(int(x2 + m + y2, 2))[2:])
    if u != c3:
        return False

    return hex(int(m, 2))[2:]
```

代码 15: SM2 解密

4.3.4 KDF 密钥派生函数

首先以每 256 比特为一组，循环生成足够的哈希数据，直到达到所需的密钥长度 klen。

在每次循环中，使用 sm3hash 函数对 z 和 ct 的组合进行哈希处理。并将计数器 ct 转换为 32 位二进制形式，然后与 z 连接。

之后将这个二进制字符串转换为十六进制字符串，用作 sm3hash 的输入。然后

将哈希值加到 k 变量上，并将计数器 ct 加 1。

最后将之前生成的哈希字符串 k 转换为二进制形式，并确保其长度为 256 的倍数。返回所要求的位数

密钥派生函数的实现代码如下所示

```
def kdf(z, klen):
    """密钥派生函数"""
    ct = 1
    k = ''
    for _ in range(math.ceil(klen / 256)):
        k = k + Hash(hex(int(z + '{:032b}'.format(ct), 2))[2:])
        ct = ct + 1
    k = '0' * ((256 - (len(bin(int(k, 16))[2:]) % 256)) % 256) + bin(int(k, 16)
        )[2:]
    return k[:klen]
```

代码 16: KDF 密钥派生

4.3.5 SM3-Hash 密码杂凑函数

在 SM2 中所使用的 Hash 函数采用 SM3 中所使用的 Hash 函数，该函数首先将原始数据填充到适合的长度，以便进行处理。然后将填充后的数据分成多个固定大小的块。之后对每个数据块进行扩展，生成一系列新的数据。最后使用一系列复杂的数学运算，将扩展的数据压缩成一个固定长度的哈希值。

本次实验中所调用的 SM3-Hash 密码杂凑函数代码如下所示

```
iv = '7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e'



def t(j):
    if j < 16:
        return 0x79cc4519
```

```
    return 0x7a879d8a


def csl(x, k):
    """左循环移位"""
    x = '{:032b}'.format(x)
    k = k % 32
    x = x[k:] + x[:k]
    return int(x, 2)


def ff(x, y, z, j):
    """布尔函数"""
    if j < 16:
        return x ^ y ^ z
    return (x & y) | (y & z) | (z & x)


def gg(x, y, z, j):
    """布尔函数"""
    if j < 16:
        return x ^ y ^ z
    return (x & y) | (~x & z)


def p0(x):
    """置换函数"""
    return x ^ csl(x, 9) ^ csl(x, 17)

```

```
36
37  def p1(x):
38      """置换函数"""
39      return x ^ csl(x, 15) ^ csl(x, 23)
40
41
42  def fill(m):
43      """填充原始消息"""
44      l = len(m) * 4
45      m = m + '8'
46      k = 112 - (len(m) % 128)
47      m = m + '0' * k + '{:016x}'.format(l)
48      return m
49
50
51  def grouping(m):
52      """消息分组"""
53      n = len(m) // 128
54      b = []
55      for i in range(n):
56          b.append(m[i * 128:(i + 1) * 128])
57      return b
58
59
60  def extend(bi):
61      """消息扩展函数"""
62      w = []
63      for i in range(16):
64          w.append(int(bi[i * 8:(i + 1) * 8], 16))
```

```
    for j in range(16, 68):
        w.append(p1(w[j - 16] ^ w[j - 9] ^ csl(w[j - 3], 15)) ^ csl(w[j - 13],
            7) ^ w[j - 6])
    for j in range(68, 132):
        w.append(w[j - 68] ^ w[j - 64])
    return w


def cf(vi, bi):
    """压缩函数"""
    w = extend(bi)
    a, b, c, d, e, f, g, h = int(vi[0:8], 16), int(vi[8:16], 16), int(vi
        [16:24], 16), int(vi[24:32], 16), int(vi[32:40],
        vi[40:48], 16), int(vi[48:56], 16), int(vi[56:64], 16)
    for j in range(64):
        ss1 = csl((csl(a, 12) + e + csl(t(j), j)) % pow(2, 32), 7)
        ss2 = ss1 ^ csl(a, 12)
        tt1 = (ff(a, b, c, j) + d + ss2 + w[j + 68]) % pow(2, 32)
        tt2 = (gg(e, f, g, j) + h + ss1 + w[j]) % pow(2, 32)
        d = c
        c = csl(b, 9)
        b = a
        a = tt1
```

```
87          h = g
88          g = csl(f, 19)
89          f = e
90          e = p0(tt2)
91      abcdefgh = int('{:08x}'.format(a) + '{:08x}'.format(b) + '{:08x}'.format(c)
                + '{:08x}'.format(d) + '{:08x}'.format(
92          e) + '{:08x}'.format(f) + '{:08x}'.format(g) + '{:08x}'.format(h), 16)
93      return '{:064x}'.format(abcdefgh ^ int(vi, 16))
94
95
96  def iteration(b):
97      n = len(b)
98      v = iv
99      for i in range(n):
100         v = cf(v, b[i])
101     return v
102
103
104 def Hash(m):  # m为16进制串
105     m = fill(m)
106     b = grouping(m)
107     return iteration(b)
```

代码 17: 密码杂凑函数

五、 实验结果与处理

5.1 RSA 加密

去掉所有的预制初始值之后运行 RSA 加密程序得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\RSA\RSA.py
明文为:6882326879666683
公钥<n,e>为:41265823 8589
私钥<p,q,d,fn>为：6197 6659 41248165 41252968
RSA加密得到密文为:513127024343762338114753137372924213954654549 8
RSA解密得到明文为:6882326879666683

Process finished with exit code 0
```

图 9: RSA 加解密结果

5.2 ELGamal 加密

去掉所有的预制初始值之后运行 ELGamal 加密程序得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\ELGamal\ELGamal.py
明文为：1299
计算得公钥y=56067916282250437581198279177455077317113039734582312073044716382539052282236572820003860357040430668769989332085233405505558722709862418605832676798
    私钥d=832335815875880411983009094834424541809851104482975003576611642837426612438041443071536025010561364948538068298772195864351322053548374435207438939 9
加密后的密文(c1,c2)为：
2364042053078748539909341159997139652992136239581552444561114824249045644611233762699878467484760663122386117585233204566196191129720247808380225429 1
2599844246163758255241885267003704720616755622369681909868157194981709692239287464221888210461880066219658506766142578318600891321632494415235519958 21
解密后的明文为：1299

Process finished with exit code 0
```

图 10: ELGamal 加解密结果

5.3 SM2 加密

去掉所有的预制初始值之后运行 SM2 加密程序得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_5\SM2\SM2.py
待加密明文为：encryption standard
加密后的密文为：
047BCECD 23E2416C F514DFD7 538921A2 6D7909F7 B8FCE6E6 D1291360 6372808F
C13C050A 2BFCF621 03AFFE8D A913600E 59C22BAC 891F1BA5 F0EADFDE 708006FB
845E18B3 696E9519 EA3BEBEA 1BDD04F0 BBC38B6B 9A898DD8 A53BAAEB FD04976D
7D56387F 4F7BD407 A8DFC529 C8100BEA DE984CC6
解密后的明文为：encryption standard
```

图 11: SM2 加解密结果

六、 分析与讨论

在这次密码学实验中，我深入了解了公钥密码学的核心原理，特别是 RSA、ElGamal 和椭圆曲线密码学等算法。通过编程实现这些算法，我更加清楚地认识到了它们在保护网络通信安全中的重要性。在实验过程中，我遇到了一些编程和逻辑上的挑战，通过不断的尝试和调试，我提高了解决问题的能力。总的来说，这次实验不仅提升了我的技术技能，也加深了我对网络安全重要性的理解。

七、 教师评语