北京建筑大学

2019/2020 学年　　第2学期

课　程　设　计

课程名称　　密码学课程设计

设计题目　　数字签名系统

（工程名称）

系　　别　信息与计算科学

班　　级　　信171

学生姓名　李金哲 聂晓婧

学　　号　201707010119 201707010103

完成日期　2020年7月2日

成绩	
指导教师 （签名）	

“现代密码学课程设计”任务书

指导教师姓名	高雁飞	教研室	信息与计算科学		
课程设计题目	数字签名系统	人数	2	学时	2周

设计目的、任务和要求

设计目的：
本课程设计力求为学生提供一个理论联系实际的机会。通过实践，建立密码学理论的整体思想，锻炼编写、调试程序的能力，学习文档编写规范，培养独立学习、吸取他人经验、探索前沿知识的习惯。同时，课程设计可以充分弥补课堂教学及普通实验中知识深度与广度有限的缺陷，更好地帮助学生从全局角度把握课程体系。

设计任务：
（1）在深入理解基于RSA/ElGamal的数字签名算法的基础上，设计数字签名系统；
（2）要求输入信息（可以是汉字或英文，信息量要求不受限制），或者是文本文档。使用RSA/ElGamal算法进行数字签名，RSA/ElGamal中模数n的长度不低于100比特。
（3）提供良好地用户界面。
（4）要求提供所设计系统的报告及完整的软件。

设计的方法和步骤

第一步：进行系统设计；
第二步：代码编码；
第三步：对实现部分的软件功能或者模块进行测试
第四步：提交完整可执行软件，准备答辩；
第五步：答辩，演示软件，教师根据实际情况提出测试用例，学生作最后的修改和完善，教师对软件运行部分进行评分；
第六步：完成课程设计报告并提交。

设计工作计划

1. 本次课程设计的时间是2周，工作计划如下：
2. 任务书下达，理解选题，明确软件功能，分析和设计：1天；
3. 分析和设计报告的撰写：1天
4. 编码及调试：6天；
5. 报告完善：1天；
6. 成果提交和验收：1天。

主要参考资料

1. 陈鲁生、沈世镒《现代密码学》（2） 北京：科学出版社，2008年
2. 杨波编， 《现代密码学》， 北京：清华大学出版社，2015年。
3. 冯登国译，[加]斯廷森（Stinson,D_R_）著，密码学原理与实践（第3版），北京：电子工业出版社，2016年。
4. 张焕国，刘玉珍，《密码学引论（第二版）》，武汉：武汉大学出版社，2015。

教研室签字： **年 月 日**

学院签字： **年 月 日**

目　　录

摘　　要

计算机网络的产生把我们带进了一个信息化的社会。当前，计算机网络的安全问题日益突出，网络给我们带来便利的同时，网络安全的形式不容乐观，已严重威胁到人们正常的生活，甚至威胁到国家安全。网络安全的实质是信息安全，信息安全技术的核心之是数字签名技术。

如何在信息传输的过程中保证信息.的安全性和真实性，这是数字签名技术所要研究的重要问题。数字签名可以解决否认、伪造、算改、冒充问题。使用数字签名技术使得发送者发送的时候不能否认发送的报文签名，接受者不能伪造发送者的报文签名，即网络中的某一用户不能冒充另一用户。

RSA数字签名体制使用的是RSA公开密钥密码算法进行数字签名，RSA算法是目前公认的在理论和实际应用中最为成熟和完善的一种公钥密码体制，它是第一个既能用于数据加密也能用于数字签名的算法，是公钥密码体制的代表。

ELGamal数字签名方案作为目前最为重要的数字签名方案之一，极大地促进了现代密码学的发展。ElGamal数字签名与一般公钥密码体制签名的不同之处，是具有高安全性和实用性。

本文阐述了RSA和ELGamal数字签名算法，实现了RSA和ELGamal数字签名的签名与验证，设计了数字签名系统。

关键词：数字签名、信息安全、RSA、加密、ELGamal、签名、验证

1. 需求分析

1.1 设计背景

密码技术是信息安全的核心技术.公钥密码在信息安全中担负起密钥协商、数字签名、消息认证等重要角色，已成为最核心的密码.本文介绍了数字签名技术的基本功能、原理和实现条件，并实现了基于RSA 和 ELGamal的数字签名算法。

1.2 设计意义

数字签名技术满足网络安全的目标即身份真实性、信息机密性、信息完整性、服务可用性、不可否认性、系统可控性、系统易用性、可审查性等等。特别是其身份鉴别、数据完整性和不可抵赖性在电子商务、电子政务等应用领域中有很重要的作用。

1.3 概述

本系统分为两部分，分别实现RSA数字签名的签名与验证和ELGamal数字签名的签名与验证，可以对数据或者传输文件进行两个算法相对应的数字签名。

2. 系统设计

2.1 系统主要目标

（1）在深入理解基于RSA/ElGamal的数字签名算法的基础上，设计数字签名系统；

（2）要求输入文本文档，使用RSA算法进行数字签名。

要求输入信息或者图片，使用ElGamal算法进行数字签名。

RSA/ElGamal中模数n的长度不低于100比特。

（3）要求提供所设计系统的报告及完整的软件。

2.2 系统编写环境

本系统使用Python语言编写

下载：

Python 3.6版本以上

Pycharm （python编译器）

Microsoft Visual C++14.0 （下载pyunit-prime包之前,需要下载VC++14.0，否则pyunit-prime包安装不成功）

pyunit-prime （pycharm中存储大质数的包，用于生成随机大素数）

wxFormBuilder（用于设计界面）

wxPython（用于生成界面代码的包，在Pycharm中包下载）

2.3 系统运行

使用pyinstaller将python文件打包成exe程序

运行 签名系统.exe 时 ，不需要系统编写环境，系统可直接运行。

打包成exe方法： https://www.cnblogs.com/mini-monkey/p/11195309.html

3. 开发技术

3.1 Python语言

*P*ython 是一种跨平台的计算机程序设计语言。 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本（shell），随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

3.2 RSA数字签名

3.2.1 用RSA生成签名

在 *RSA* 中，被签名的消息、密钥以及最终生成的签名都是以数字形式表示的。在对文本进行签名时，需要事先对文本编码成数字。用 *RSA* 生成签名的过程可用下列公式来表述：

$$\text{签名} = \text{消息}^{D} \bmod N$$

这里所使用的 D 和N就是签名者的私钥。签名就是对消息的 D 次方求 mod N的结果，也就是说将消息和自己相乘 D 次，然后再除以N求余数，最后求得的余数就是签名。生成签名后，发送者就可以将消息和签名发送给接收者了。

3.2.2 用RSA验证签名

RSA 的签名验证过程可用下列公式来表述：

这里所使用的 E 和 N 就是签名者的公钥。接收者计算签名的 E 次方并求 $\bmod N$，得到“由签名求得的消息”，并将其与发送过来的“消息”内容进行对比，如果两者一致，则签名验证成功，否则签名验证失败。

3.3 ELGamal数字签名

（1） 生成乘法群 Z 中的一个生成元 g； p,q 公开。

（2） 随机选取整数 x，$1 \le \mathrm{x} \le p-2$，计算 $\mathrm{y} = g^{x} (\bmod p)$， y 是公开密钥，而 x 是保密密钥。

（3） 签名算法：设 $m \in Z$ 是待签名的消息，随机选取一个整数 k，$1 \le \mathrm{k} \le p-2$，且$(\mathrm{k}, p-1) = 1$，计算

$\mathrm{r} = g^{k} (\bmod p)$　　$\mathrm{s} = k^{-1(m-rx)} (\bmod p-1)$　则（m,r,s）为对消息的 m 数字签名。

（4） 验证算法：对方收到对消息 m 的数字签名（m,r,s）后，利用签名者的公开密钥 y,g,p 可对签名进行以下验证: $(\mathrm{y}^{\mathrm{r}})(\mathrm{r}^{\mathrm{s}}) = \mathrm{g}^{\mathrm{m}} (\bmod p)$ 如果上式成立，则接受该签名，否则拒绝该签名。

对 m 正确签名，那么有：

$$\begin{aligned}(y^{r})(r^{s}) (\bmod p) &= g^{(rx+sk)} (\bmod p) \\ &= \mathrm{g}^{(\mathrm{rx+m-rx})} (\bmod p) \\ &= \mathrm{g}^{\mathrm{m}} (\bmod p)\end{aligned}$$

3.4 ELGamal生成原根g定理

图论书中：

定理8 设 $m>1$， $\varphi(m)$ 的所有不同素因数是 $q_1,...,q_k$ ，则 g 是模 m 的一个原根的充分必要条件是：

$g^{\varphi(m)/q_i} \neq 1(\mathrm{mod}\, m)$， $i=1,...,k$

这里用例题解释：求模43的原根.

解：设 $m=43$，则

$$\varphi(m)=\varphi(43)=2\cdot 3\cdot 7，\ q_1=2，\ q_2=3，\ q_3=7$$

因此，

$$\varphi(m)/q_1=21，\ \varphi(m)/q_2=14，\ \varphi(m)/q_3=6$$

这样，只需验证： g^{21},g^{14},g^{6} 模 m 是否同余于1，对2,3，... 逐个验算：

$$2^2\equiv 4，\quad 2^4\equiv 16，\quad 2^6\equiv 64\equiv 21，\quad 2^7\equiv 21\cdot 2=-1，$$
$$2^{14}\equiv 1，\quad 3^2\equiv 9，\quad 3^4\equiv 81\equiv -5，\quad 3^6\equiv 9\cdot(-5)\equiv -2，$$
$$3^7\equiv -6，\quad 3^{14}\equiv(-6)^2\equiv 36，\quad 3^{21}\equiv(-6)\cdot 36\equiv -1(\mathrm{mod}\, 43)$$

因此，3是模43 的原根.

3.5 公钥密码算法的可行性分析

（1）利用公钥密码进行高字节文件的数字签名，原理就是将其分解成为多段较小字节数的部分，然后再分别进行数字签名，通过这样的方法就能够将较大的文件进行公钥密码的数字签名。但是由于个人电脑的性能不可能达到企业用户的级别，因此在个人对文件进行 公钥密码数字签名时还是应该考虑文件的大小，一般个人用户在数字签名时，最好不宜超过 10K，那样等待时间才不会过长。而如果强行利用低配置的电脑对较大文件进行数字签名，除了耗时过长的因素外，还会对数字签名后文件的安全性带来一些意外因素。

（2）在进行非对称加密操作应注意到如下几点：首先，公钥密码的相关运算中，幂模运算及素数运算功能模块耗时最长，因此在进行核心优化时应该首先对这两个功能模块进行优化操作。其次，在进行公钥密码数字签名操作时，对加密文件的读取及储存是消耗时间的主要环节，造成这一消耗的主要原因是文件存储的磁盘读写速度与内存读写速度不匹配所造成的，因此要对此环节进行优化就应当将绝大多的读写操作集中到内存中，再进行相关操作，同时在完成所有操作后，再一次性写入存储磁盘，这样就能减少不必要的磁盘读写操作，极大地减少了 公钥密码数字签名所耗费的时间，提高了工作效率。

4. 详细设计

4.1 系统数据结构

——| 加密系统
————| Gui
————————| ElGanml
————————————| de_date.py
————————————| en_data.py
————————————| de_result.py
————————————| en_result.py
————————————| key_result.py
————————————| Main.py
————————————| tip.py
————————| RSA
————————————| de_data.py
————————————| en_data.py
————————————| de_result.py
————————————| en_result.py
————————————| key_result.py
————————————| Main.py
————————| enter.py
————| model
————————| common.py
————————| ELGamal.py
————————| RSA.py
————| main.py

4.2 函数定义、接收参数、返回值及作用

4.2.1 model - RSA

（1） 函数：**def** get_two_prime(lenth1, lenth2)
接收参数： lenth1大素数1的获取长度 lenth2大素数2的获取长度
返回值：**return** prime1, prime2
prime1大素数1, prime2大素数2
作用：获取两个大素数

（2） 函数：**def** ext_gcd(a, b)
接收参数： a b
返回值：**return** x, y, gcd

gcd最大公因数

作用：欧几里得算法求最大公约数

（3） 函数：**def** get_key(len1, len2)

接收参数： len1大素数1的获取长度 len2大素数2的获取长度

返回值：**return** n, e, d

n公钥, e公钥，d私钥

作用：生成e的逆元d

（4） 函数：**def** data_encrypt(n,e,c_path=**None**,m=**None**)

接收参数：n e c_path path输入文档路径 m

返回值：**return** c_path,max_len

c_path，d私钥 max_len最高位长度

作用：实现对文档的RSA数字签名的签名

（5） 函数：**def** data_encrypt(n,e,c_path,m=**None**)

接收参数：n e c_path path输入文档路径 m

返回值：**return** c_path,max_len

c_path，d私钥 max_len最高位长度

作用：实现对数据的RSA数字签名的签名

（6） 函数：**def** data_decrypt(n, d, c_max_len, c_path, m_path=None)

接收参数：n d c_max_len c_path m_path

返回值：**return** m_path

return m_s.decode()

作用：实现RSA数字签名的验证

4.2.2 model - ELGamal

（1） 函数：**def** get_key(n=20)

接收参数： n生成的大素数的长

返回值：**return** p, i, fastExpMod(i, x, p), x

作用：获取公钥p,g,g^xmodp,和私钥x

（2） 函数：**def** exgcd(a, b, x=1, y=0)

接收参数： a b x y

返回值：**return** gcd,x, y

gcd最大公因数

作用：欧几里得算法求最大公因数

（3） 函数：**def** cal_inv(a, m)

接收参数： a m模的值

返回值：**return** (x % m + m) % m **if** gcd == 1 **else** -1

作用：求逆元

（4） 函数：**def** encrypt(m, p, g, y)

接收参数：m p g y:公开参数（gxmodp）

返回值：**return** c1,c2

作用：实现ELGamal的数字签名的签名算法

（5） 函数：**def** decrypt(c1, c2, x, p)

接收参数：c1密文 c2密文 x私钥 p公钥

返回值：**return** c1 * c2 % p

作用：实现ELGamal的数字签名的验证算法

（6） 函数：**def** data_encrypt(p, g, y, c_path, path=**None**, m_data=**None**)

接收参数：p g y c_path path m_data

返回值：**return** path1, path2, c1_max_len, c2_max_len

作用：实现对文档的ELGamal数字签名的验证

（7） 函数：**def** data_decrypt(p, x, path1, path2, c1_max_len, c2_max_len, path_m=**None**)

接收参数：p x path1 path2 c1_max_len c2_max_len

返回值：**return** m_s.decode()

作用：实现对数据的ELGamal数字签名的验证

4.2.3 model - common

（1） 函数：**def** fastExpMod(m, e, n)

接收参数：m底数 e幂数 n模数

返回值：**return** result

作用：模的重复平方法

（2） 函数：**def** list_split(items, n):

接收参数：items n

返回值：**return** [items[i:i + n] **for** i **in** range(0, len(items), n)]

作用： 划分区间 item划分成n个区间

（3） 函数：**def** makefile(path, content)

接收参数： path保存地址 content二进制数据列表

返回值：无

作用：以二进制编码，保存文件

（4） 函数：**def** get_data(path)

接收参数：path文件地址

返回值：**return** data_all

作用：读取文件的数据

（5） 函数：**def** standard_data(data_all, lenth)

接收参数：data_all lenth

返回值：**return** list_split(data_list, lenth)

作用：将数据标准化

（6） 函数：**def** recovery_data(res)

接收参数：res

返回值：**return** m

作用： 恢复数据

（7） 函数：**def** save_ctext(path, data_list, max_lenth)

接收参数：path签名存储地址 data_list签名列表 max_lenth单列签名最大长度

返回值：无

作用： 签名专属写入操作，要规范数据格式，将所有数据长度一致化

4.3 算法及对应的关键代码

（1）快速幂取模/模的重复平方法

```
def fastExpMod(m, e, n):
        result = 1
    while e != 0:
        if (e & 1) == 1:
            result = (result * m) % n
        e >>= 1
        m = (m * m) % n
    return result
```

（2）划分区间

```
def list_split(items, n):
    return [items[i:i + n] for i in range(0, len(items), n)]
```

将item划分成n个区间

（3）生成文件

```
def makefile(path, content):
    try:
        f = open(path, 'wb')
        for i in content:
            f.write(int(i).to_bytes(1,'little'))
        f.seek(0)
        f.close()
        print('文件已生成')
    except Exception as e:
        print('错误：',e)
        exit(0)
```

（4）读取文件

```
def get_data(path):
    f = open(path, 'rb')
    data_all = f.read()
    f.close()
    return data_all
```

（5）进行标准化

```
def standard_data(data_all, lenth):
    import math
```

```
    data_list = ''
    for data in data_all:
        if data == 0:
            long = 1
        else:
            long = int(math.log10(data)) + 1
        data_list += str(long * 10 ** 3 + data)
    if lenth < 4:
        lenth = 1
    else:
        lenth = int(lenth / 4)*4
    return list_split(data_list, lenth)
```

byte 最多为 256 个 即最高位数为 4

（6）签名专属写入操作

```
def save_ctext(path, data_list, max_lenth):
    m_s = ''
    for data in data_list:
    if len(str(data)) < max_lenth:
            m_s += (max_lenth - len(str(data))) * '0' + str(data)
        else:
            m_s += str(data)
    m_s = list_split(m_s, 2)
    makefile(path, m_s)
```

要规范数据格式，将所以数据长度一致化

（7）扩展欧几里得算法

```
def ext_gcd(a, b):
  if b == 0:
        return 1, 0, a
    else:
        x, y, gcd = ext_gcd(b, a % b)
        x, y = y, (x - (a // b) * y)
    return x, y, gcd
```

递归直至余数等于0(需多递归一层用来判断)辗转相除法反向推导每层a、b的因子使得gcd(a,b)=ax+by成立

（8）生成公开的n,e,私钥d

```
def get_key(len1, len2):
    p, q = get_two_prime(len1, len2)
    n = p * q
    yn = (p - 1) * (q - 1)
    import random
    while True:
        e = random.randint(2, yn - 1)
        res = ext_gcd(yn, e)
```

```
        if res[2] == 1:
            break
    if res[1] < 0:
        d = yn + res[1]
    else:
        d = res[1]
    return n, e, d
```

通过两个大素数p和q，求取n和φ(n)，随机选取2到φ(n)-1的整数e（1＜e＜φ(n)），令e和φ(n)的最大公因数为1（即e和φ(n)互素），求e的逆元$d=e^{-1} \bmod \varphi(n)$，d为私钥，（n，e）为公钥。

（9）RSA数字签名

```
def data_encrypt(n, e, c_path, path=None, m=None):
    if path:
        data_all = get_data(path)
    else:
        data_all = m.encode()
    import math
    data_list = standard_data(data_all, lenth=int(math.log10(n)) + 1)
    m_l = []
    max_len = 0
    for data in data_list:
        c = fastExpMod(int(data), e, n)
        if c == 0:
            lenth = 1
        else:
            lenth = int(math.log10(c)) + 1
        if lenth > max_len:
            max_len = lenth
        m_l.append(c)

    save_ctext(c_path, m_l, max_len)
    return c_path, max_len
```

（10）RSA签名验证

```
def data_decrypt(n, d, c_max_len, c_path, m_path=None):
    c = get_data(c_path)
    import math
    m_list = ''
    for m in c:
        if m == 0:
            m_list += '00'
        else:
```

```
            m_list += '0' * (1 - int(math.log10(m))) + str(m)
    c = m_list
    str_list = list_split(c, c_max_len)
    res = ''
    for data in str_list:
        m = fastExpMod(int(data), d, n)
        res += str(m)
    m = recovery_data(res)
    if m_path:
        makefile(m_path, m)
        return m_path
    else:
        m_s = bytes()
        for data in m:
            m_s += data.to_bytes(1, 'little')
        return m_s.decode()
```

（11）ELGamal的原根g

```
def get_key(n=20):
    import pyunit_prime
    p = pyunit_prime.get_large_prime_length(n)
    yn = p - 1
    f = 2
    x_list = []
    while yn != 1:
        print('yn：', yn)
        print('f：', f)
        print(pyunit_prime.is_prime(int(yn)))
        if pyunit_prime.is_prime(int(yn)):
            x_list.append(int(yn))
            break
        if pyunit_prime.is_prime(f):
            if yn % f == 0:
                x_list.append(f)
                while yn % f == 0:
                    yn = yn // f
            if yn == 1:
                break
            f += 1
        else:
```

```
            f += 1
    yn = p - 1
    print('yn：', yn)
    print('x list：', x_list)
    error_list = []
    import random
    while True:
        i = random.randint(2, p)
        if i in error_list:
            continue
        else:
            error_list.append(i)
        k = 0
        for j in x_list:
            if fastExpMod(i, int(yn // j), p) == 1:
                break
            else:
                k += 1
        if k == len(x_list):
            import random
            x = random.randint(1, p - 1)
            return p, i, fastExpMod(i, x, p), x
```

g是模m的一个原根的充分必要条件是：$g^{\varphi(m)/q_i} \equiv 1(\mathrm{mod}\, m)$

（12）求最大公因数

```
def exgcd(a, b, x=1, y=0):
    if b == 0:
        return a, x, y
  gcd, x, y = exgcd(b, a % b, x, y)
  x, y = y, x - a // b * y
    return gcd, x, y
```

（13）求逆元

```
def cal_inv(a, m):
        gcd, x, y = exgcd(a, m)
    return (x % m + m) % m if gcd == 1 else -1
```

（14）ELGamal数字签名

```
def data_encrypt(p, g, y, c_path, path=None, m_data=None):
    if path:
        data = get_data(path)
    else:
        data = m_data.encode()
```

```
    import math
    data_list = standard_data(data, lenth=int(math.log10(p))+1)
    c1_l = []
    c2_l = []
    c1_max_len = 0
    c2_max_len = 0
  for data in data_list:
        num = int(data)
        c1, c2 = encrypt(num, p, g, y)
        c1_l.append(c1)
        c2_l.append(c2)
        if c1 != 0:
            len1 = int(math.log10(c1)) + 1
        else:
            len1 = 1
        if c2 != 0:
            len2 = int(math.log10(c2)) + 1
        else:
            len2 = 1
        if len1 > c1_max_len:
            c1_max_len = int(math.log10(c1)) + 1
        if len2 > c2_max_len:
            c2_max_len = int(math.log10(c2)) + 1
    import os
    path1 = os.path.join(c_path, 'c1.txt')
    save_ctext(path1, c1_l, c1_max_len)
    path2 = os.path.join(c_path, 'c2.txt')
    save_ctext(path2, c2_l, c2_max_len)
    return path1, path2, c1_max_len, c2_max_len
```

（15）ELGamal签名验证

```
def data_decrypt(p, x, path1, path2, c1_max_len, c2_max_len, path_m=None):
      c1 = get_data(path1)
    import math
    m_list = ''
    for m in c1:
        if m == 0:
            m_list += '00'
        else:
            m_list += '0' * (1 - int(math.log10(m))) + str(m)
    c1 = m_list
```

```
c2 = get_data(path2)
import math
m_list = ''
for m in c2:
    if m == 0:
        m_list += '00'
    else:
        m_list += '0' * (1 - int(math.log10(m))) + str(m)
c2 = m_list
c1 = str(c1).replace("[", "").replace("]", "").replace(",", "").replace(" ", "")
c2 = str(c2).replace("[", "").replace("]", "").replace(",", "").replace(" ", "")
c1 = list_split(c1, c1_max_len)
c2 = list_split(c2, c2_max_len)
if len(c1) != len(c2):
    print('密文有错误')
    exit(0)
res = ''
for i in range(len(c1)):
    res += str(decrypt(int(c1[i]), int(c2[i]), x, p))
m = recovery_data(res)
if path_m:
    makefile(path_m, m)
    return path_m
else:
    m_s = bytes()
    for data in m:
        m_s += data.to_bytes(1, 'little')
    return m_s.decode()
```

4.4 界面

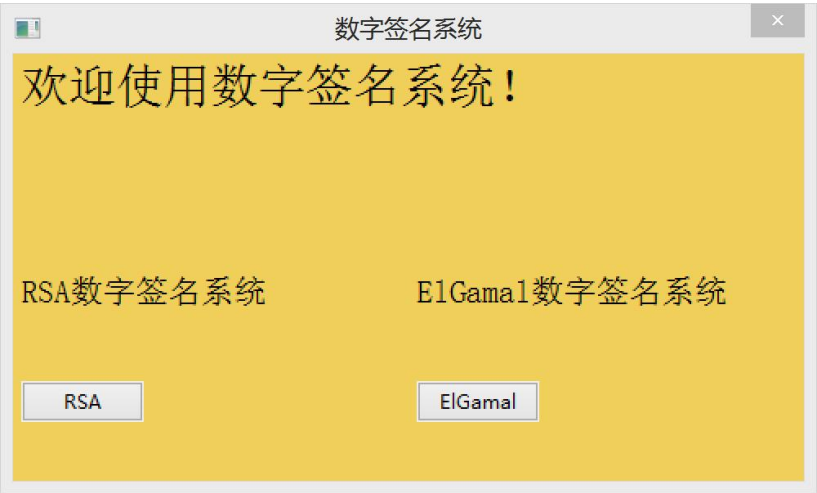

图4-4-1 数字签名系统界面

RSA按钮绑定的函数

def rsa_enter(self, event)

触发跳转到Gui.RSA.Main.py中

按钮绑定的函数

（1）**def** get_key(self, event):

myframe = key_result(**None**, p_len=int(self.p_len.GetValue()), q_len=int(self.q_len.GetValue()))

跳转到Gui.RSA.key_result页面传递参数为当前页面获得的p,q_len信息

（2）**def** en_data(self, event)

myframe = en_data(**None**, path=self.m_file.GetPath())

myframe = en_data(**None**, m=self.m_text.GetValue())

跳转到Gui.RSA.en_data页面传递参数为当前页面获得的path,m信息

（3）**def** de_data(self, event):

myframe = de_data(**None**, path=self.m_filePicker4.GetPath(), file=file)

跳转到Gui.RSA.de_data页面传递参数为当前页面获得的path,file信息

（4）**def** return_menu(self, event):

SiteFrame = MyFrame2(**None**)

跳转到Gui.enter页面

图4-4-2 RSA数字签名系统界面

生成密钥按钮绑定的函数

```
def __init__(self, parent, p_len, q_len):
    n, e, d = get_key(len1=p_len, len2=q_len)
```

触发跳转到Gui.RSA.key_result界面 并调用函数get.key获取n,e,d信息

图4-4-3 RSA生成密钥结果界面

签名操作按钮绑定的函数

```
def en_result(self, event)
```

触发跳转到Gui.RSA.en.data界面

图4-4-4 RSA签名界面

签名按钮绑定的函数

def __init__(self, parent, path=**None**, m=**None**)

触发跳转到 Gui.RSA.en_result 界面

图4-4-5 RSA签名结果界面

验证操作按钮绑定的函数

def de_result(self, event)

触发跳转到Gui.RSA.de_data界面中

图4-4-6 RSA验证操作界面

验证操作按钮绑定的函数

def __init__(self, parent, content)

触发跳转到 Gui.RSA.de.result 界面

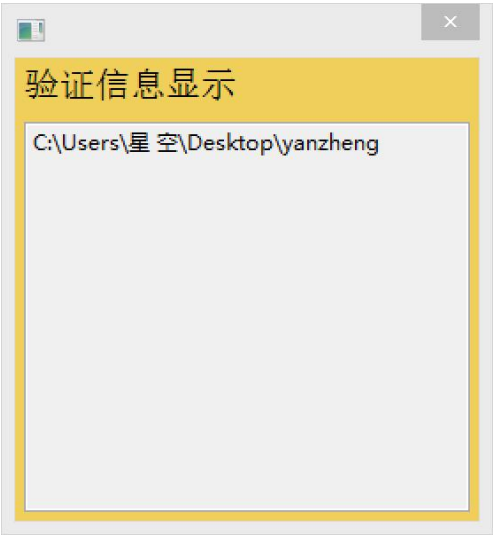

图4-4-7 RSA验证信息显示界面

ELGamal按钮绑定的函数

def elgamal_enter(self, event)

触发跳转到Gui.ElGamal.Main.py中

按钮绑定的函数

（1）**def** get_key(self, event)

myframe = tip(**None**, self.long.GetValue())

跳转到Gui.ELGamal.tip页面传递参数为当前页面获得的self.long.GetValue()信息

（2）**def** encrypt(self, event):

myframe = MyDialog2(**None**, path=self.file.GetPath())

myframe = MyDialog2(**None**, data=self.text.GetValue())

跳转到Gui.ElGamal.en_data页面传递参数为当前页面获得的path,data信息

（3）**def** decrypt(self, event)

myframe = MyFrame3(**None**, c1_path=self.c1_file.GetPath()

跳转到Gui.ELGamal.de_data页面传递参数为当前页面获得的path信息

（4）**def** return_menu(self, event):

SiteFrame = MyFrame2(**None**)

跳转到Gui.enter页面

图4-4-8 ELGamal数字签名界面

生成密钥按钮绑定的函数

def get_key(self, event)

触发跳转到Gui.ELGamal.tip界面

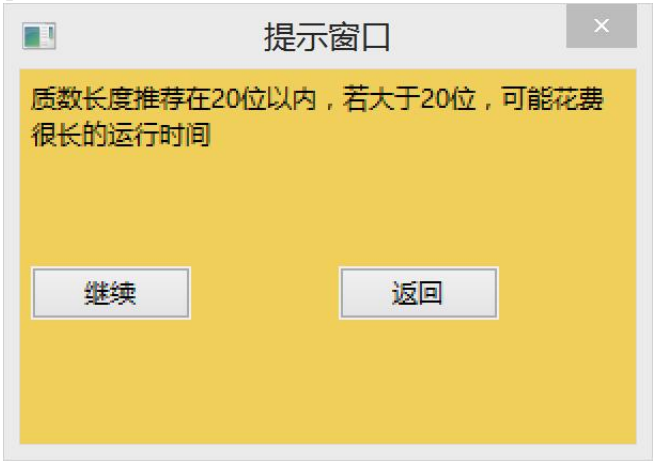

图4-4-9 ELGamal生成原根提示窗

继续按钮绑定的函数

def __init__(self, parent, lenth)

触发跳转到Gui.EGamal..key_result界面

图4-4-10 ELGamal生成密钥结果

签名操作按钮绑定的函数

def en_result(self, event)

触发跳转到Gui.ELGamal.en_data界面

图4-4-11 ELGamal签名界面

签名按钮绑定的函数

def do_on(self, event)

触发跳转到 Gui.ELGamal.en_result 界面

图4-4-12 ELGamal签名结果界面

验证操作按钮绑定的函数

def de_result(self, event)

触发跳转到Gui.ELGamal.de_data界面

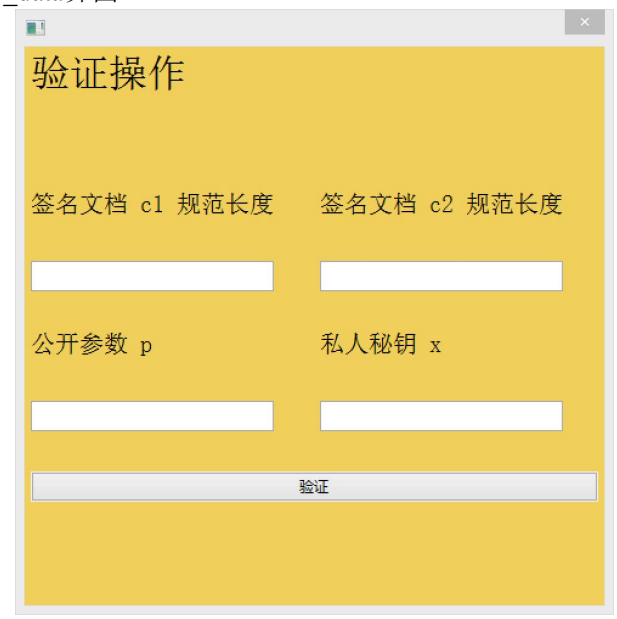

图4-4- 13 ELGamal验证操作界面

验证按钮绑定的函数

def __init__(self, parent, content)

触发跳转到 Gui.ELGamal.de_result 界面

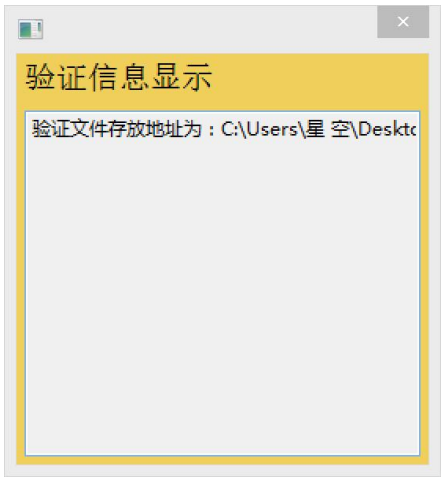

图4-4-14 ELGamal验证信息显示

5. 测试

5.1 测试内容

测试系统的签名结果和验证结果，通过不同用户输入相同内容，判断输出结果是否相同，来判定测试成功或失败。

5.2 测试步骤

步骤一：生成密钥n，e，d

打开数字签名系统-->进入RSA数字签名系统界面-->填写大质数p，q的长度-->点击生成密钥

图5-2-1 生成密钥结果

步骤二：填写签名模块进行签名操作（在这里我们选择数据签名作为测试）

在数据签名框中输入想要填写的数据，例：姓名+学号-->点击签名操作

图5-2-2 签名模块填写示例

-->进入签名界面，将生成的n,e输入，输入或浏览生成文件的保存地址，填写文档名称-->点击签名

图5-2-3 签名界面的填写

-->得到签名结果的保存地址和签名文档规范长度

图5-2-4 签名结果图

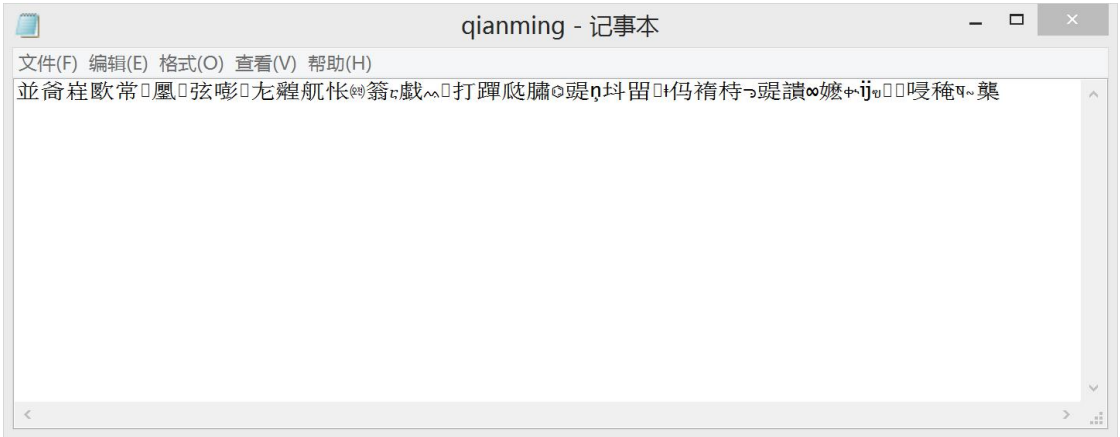

图5-2-5 生成的文档

步骤三：填写验证模块进行验证操作

点金浏览选中签名生成的文档qianming.txt-->输入规范长度，n，d，输入或点击浏览验证文件保存位置并填写文件名称-->点击操作验证

-->得到验证信息显示

图5-2-6 验证信息显示结果

5.3 测试结果

其他用户输入相同内容的时候可以输出相同的结果，即：将验证信息与签名模块输入信息对比，**结果一致，测试成功**！

6. 总结

本次课设我们组使用了python编程语言，实现了对输入的任意文本文档或数据，进行了RSA和ELGamal的数字签名。通过本学期的课上学习，我们已经初步了解了RSA和ELGamal算法的基本原理，并基于这个基础动手实际编写RSA和ELGamal数字签名算法的代码，在RSA的部分难点主要在于实现欧几里得算法上获得解密密钥d，并通过RSA数字签名算法对其进行签名与验证；在ELGamal的部分中难点在于计算原根g，在这部分主要学习了图论中求原根的定理及方法，其他原理与RSA基本相同。课设让我们更好的了解和掌握了数字签名的相关内容。通过用python对密码体制进行编程，使我们对非对称加密的数字签名过程有了更深入的理解。通过这个实验更是让我们获得了很多实际工作中所要具备的能力。

编译过程中出现的问题及解决：

1、 签名认证时，中文出现乱码

问题分析：在读取数据文件时没有规范读取文件的格式，最初想使用的 ASCII 码作为读取文档的规范格式，但是很明显 ASCII 码通常表示英文或者标点符号等，而中文需要自己的编码集，类似于 UTF-8、GBK 等。在不同的编码集的编码范围不同，因此其中的相同编码对应的汉字不同，导致最后解读出来的内容也存在着巨大的偏差，造成了数据最后存在有乱码的形式。

解决方法：通过查找相关资料，发现当读取数据为 bytes（即字节流）时，数据中已经具有了格式的规定，而且由于 bytes 的单位 byte（字节）具有明确的规定，大小范围为 0-255。故而转而使用 bytes 作为读写数据的单位。

2、 加密格式的规范

问题分析：当读取数据时，存在以 0 作为首个 byte 时，可能会因为数据区间的划分，导致的无法对首个 byte 进行操作签名，例如：原始数据为 0、10 的 bytes。假设我们按照每两个单元的 bytes 进行签名，首先在签名的过程中，由于需要将原始数据转为 int 类型，因此 0、10 的 bytes 会变成 10 的 bytes，首位的 0 被吞掉；其次在将 bytes 写回文本时无法确定具体时应当将 0、10 作一个 byte 写回，还是需要作为 2 个 byte 写回文件。

解决方法：将读取的数据按照类似 MD 拓展的形式进行填充，根据 byte 的大小的性质，可以确定 byte 最大长度为 3 位，因此将读入的每个数据拓展成为 4 位的数据，首位为有效位数，后 3 位为原始数据。如 0 会拓展成为 1000，20 会拓展成为 2020，132 会拓展成为 3123。进而对拓展后的数据进行签名操作

3、 签名的数据格式

问题分析：在生成的签名内容的过程中，由于原文的不同，会产生长度不一致的签名内容，但是因为签名文件的内容的长度存在着差异，会影响接下来在签名文件的读取的过程中无法确定应当对那一段签名文件进行验证，严重影响验证的效率。

解决方案：获取签名文件数据集中签名内容的最大长度，将最大长度作为规范长度，将所有签名的长度都规定为最大长度，位数不足的前面补 0 至位数相同。最后对新的签名文件数据集进行拆分写入签名文件中。

4、 被签名内容长度问题

问题分析：根据签名算法，最终的签名内容为，被签名的内容经过一系列的运算最后需要对 y(n)进行取模运算，但是如果被签名的内容自身转为 10 进制的数据流的具体值大于 y(n)的具体值，会导致在验证的过程中，由于需要对 y(n) 取模，生成的验证数据的具体值不可能大于 y(n)，从而导致验证文件和原始文件不一致。

解决方案：将原始被签名拆分成多个内容的长度小于 y(n)的长度的区间，由于被签名文件为规范后的 4 位数据，因此区间长度应当位 y(n)的长度整除 4 后减 1 在乘 4，该区间范围作为每次签名的数据范围。

参考文献

[1]陈鲁生、沈世镒《现代密码学》（2） 北京：科学出版社，2008.

[2]杨波编 《现代密码学》 北京：清华大学出版社，2015年.

[3]冯登国译 [加]斯廷森（Stinson,D_R_）著 密码学原理与实践（第3版） 北京：电子工业出版社，2016.

[4]张焕国 刘玉珍 《密码学引论（第二版）》 武汉：武汉大学出版社， 2015.

[5]刘红 《试论RSA技术的应用》价值工程 2013年. 238页

[6]施向东、董平 基于RSA 算法的一种新的加解密核设计[J]. 微计算机信息，2005.

[7]Burton Kalinski. Some Examples of the PKCS Standards[J].RSA Laboratories，1993.

[8]陈发来 中国科学技术大学数学系，数学实验—素数，2005.

[9]原民民 《基于ELGamal改进的数字签名算法研究》 淮海工学院学报 2011.

[10]韩韬、党天岳 《ELGamal加密算法在文件传输中的实现》 技术应用 2016.