



公钥密码学数学基础第二次实验报告

周家熠、王煜涵、潘子豪、刘一童

2024 年 10 月 17 日

目录

1 摘要	2
2 实验内容	3
2.1 题目 1: 凯撒密码	3
2.2 题目 2: 同余方程解身份证校验码	3
2.3 实验过程	3
2.3.1 凯撒密码	4
2.3.2 身份证校验码	7
3 实验小结	11
3.1 凯撒密码安全性分析	11
3.2 身份证校验码的作用	11

1 摘要

本次实验为公钥密码学数学基础实验课第二次实验，由周家熠、王煜涵、潘子豪、刘一童小组完成。实验内容依次完成两个实验。第一个为编程解决凯撒密码并打开第二个实验的文档；第二个实验为解同余方程解决身份证校验码的问题。于 10 月 11 日完成实验，17 日撰写实验报告。小组成员任务分别为：周家熠用 NTL 完成同余方程解校验码的问题，刘一童、潘子豪负责编程解凯撒密码问题，王煜涵负责优化代码，刘一童负责撰写实验报告。本文作者刘一童学号：202300460117，所属班级为 2023 级密码二班。本实验报告为用 overleaf 所含的 LaTeX 在线编译工具完成。

2 实验内容

2.1 题目 1：凯撒密码

(1) 下列密文采用凯撒密码的方式进行加密，请编程实验凯撒密码解密计算，求出系列密文所对应的明文。

DROZKCCGYBNSCDROCAEKBOYPOSQRDIYXO

(2) 请根据解密结果，打开说明附件，并完成附件中实验内容。

(3) 请将凯撒密码和附件中实验内容写入实验报告。

2.2 题目 2：同余方程解身份证校验码

已知有如下三个正确的身份证号码：

110101190001011009

236265196008162472

65256919760524176X

请编程实验身份证号码算法，求出第 15,16, 17 位置上的加权因子 x, y, z 。

附：计算校验码的方法：

(1) 计算 17 位数字本体码加权求和公示

$$S = \sum_{i=0}^{i=16} A_i * W_i$$

其中， A_i 表示第 i 位置上的身份证号码数字值， W_i 表示第 i 位置上的加权因子。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W_i	7	9	10	5	8	4	2	1	6	3	7	9	10	5	x	y	z

(2) 计算模：

$$Y \equiv S(mod11)$$

(3) 通过模得到对应的校验码：

Y	0	1	2	3	4	5	6	7	8	9	10
校验码	1	0	x	9	8	7	6	5	4	3	2

2.3 实验过程

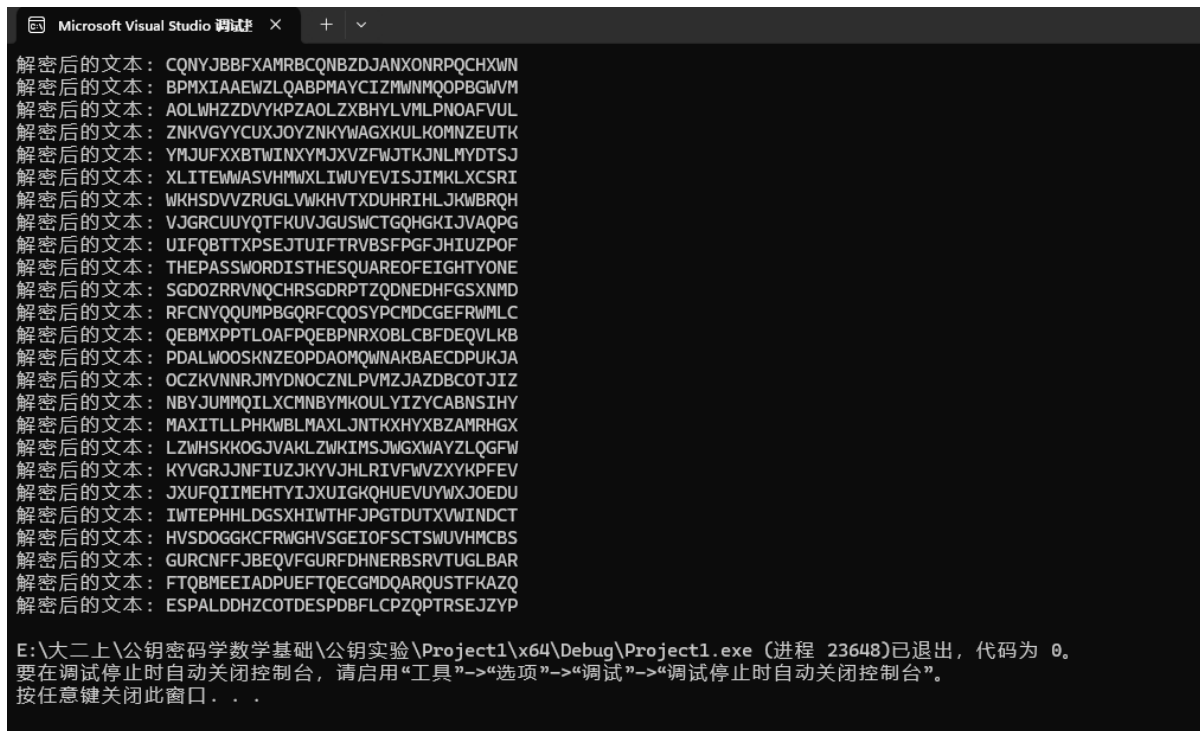
本次实验由两部分构成，下面会将实验具体内容以及相关代码附上。

2.3.1 凯撒密码

下面是完成凯撒密码部分所需的全部代码

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 std::string caesarDecrypt(const std::string& encryptedText, int shift)
5 {
6     std::string decryptedText = "";
7
8     // 遍历每一个字符
9     for (char c : encryptedText)
10    {
11        // 如果是大写字母
12        if (isupper(c))
13        {
14            decryptedText += char(int(c - shift - 65 + 26) % 26 + 65);
15        }
16        // 如果是小写字母
17        else if (islower(c))
18        {
19            decryptedText += char(int(c - shift - 97 + 26) % 26 + 97);
20        }
21        // 如果是非字母字符，保持不变
22        else {
23            decryptedText += c;
24        }
25    }
26
27    return decryptedText;
28 }
29 int main()
30 {
31     for (int shift = 1; shift <= 25; shift++)
32     {
33         std::string encryptedText = "DROZKCCGYBNSCDROCAEKBOYPOSQRDIYXO";
34         // 这是加密文本
35         std::string decryptedText = caesarDecrypt(encryptedText, shift);
36         std::cout << "解密后的文本: " << decryptedText << std::endl;
37     }
38     return 0;
39 }
```

该实验较为简单通过构造函数来实现，大致思路为将密文以字符串的形式读取，并且依据凯撒密码的定义对每个字母向后读取 i 位，进行循环分别取 i 不同的值，之后将其打印读取，找出有意义的结果即可完成解密。



```

解密后的文本: CQNYJBBFXAMRBCQNBZDJANXONRPQCHXWN
解密后的文本: BPMXIAAEWZLQABPMAYCIZMWNMQOPBGWVM
解密后的文本: AOLWHZZDVYKPAOLZXBHVLVMLPNOAFVUL
解密后的文本: ZNKVGYCUXJOYZNKYWAGXKULKOMNZEUTH
解密后的文本: YMJUFXBTWINXYMJXVZFWJTKJNLMYDTSJ
解密后的文本: XLITEWASVHMWXLWUYEVISJIMKLXCSRI
解密后的文本: WKHSDVZRUGLVWKHVTXDUHRIHLJKWBRQH
解密后的文本: VJGRUUYQTFKUVJGUSWCTGQHGKIJVAQPG
解密后的文本: UIFQBTXPSEJTUIFTRVBSFPGFJHIUZPOF
解密后的文本: THEPASSWORDISTHESQUAREOFEIGHTYONE
解密后的文本: SGDOZRRVNQCHRSGDRPTZQDNEDHFGSXNMD
解密后的文本: RFCNYQUMPBGQRCQOSYPCMDCGEFRWMLC
解密后的文本: QEBMXPPTLOAFPQEBPNRXOBLCBFDEQVLKB
解密后的文本: PDALWOOSKNZEOPDAOMQWNAKBAECDPUKJA
解密后的文本: OCZKVNNRJMYDNOCZNLPMVZJAZDBCOTJIZ
解密后的文本: NBYJUMQILXCMNBMYMKOULYZYCABNSIHY
解密后的文本: MAXITLLPHKWBLMAXLJNTKXHYXBZAMRHGX
解密后的文本: LZWHSKKOGJVAKLZWIKIMSJWGXYVZLQGFV
解密后的文本: KYVGRJJNFIUZJKYVJHLRIVFWVZXYPFEV
解密后的文本: JXUFQIIMEHTYIJXUIGKQHUEVUYWXJOEDU
解密后的文本: IWTEPHLDGSXHIWTHFJPGTDUTXVWINDCT
解密后的文本: HVSDOGKCFRWGHVSGEIOFSCTSWUVMCBBS
解密后的文本: GURCNFFJBEQVFGURFDHNERBSRVUGLBAR
解密后的文本: FTQBMEEIADPUEFTQECGMDQARQUSTFKAZQ
解密后的文本: ESPALDDHZCOTDESPDBFLCPZQPTRSEJZYP

E:\大二上\公钥密码学数学基础\公钥实验\Project1\x64\Debug\Project1.exe (进程 23648)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

```

图 1: 解密结果

发现有意义的字符串

THEPASSWORDISTHESQUAREOFEIGHTYONE

显然密码为 6561，解密成功。

解密后的文档:

身份证号算法实现

一、实验准备

公民身份号码是特征组合码，由十七位数字本体码和一位校验码组成。排列顺序从左至右依次为：六位数字地址码，八位数字出生日期码，三位数字顺序码和一位数字校验码。

校验码（身份证最后一位）是根据前面十七位数字码，按照 ISO7064:1983.MOD11-2 校验码计算出来的校验码。

校验码的计算方法：

(1) 计算十七位数字本体码加权求和公式

$$S = \sum_{i=0}^{16} A_i * W_i$$

中， A_i 表示第*i*位置上的身份证号码数字值， W_i 表示第*i*位置上的加权因子。

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W_i	7	9	10	5	8	4	2	1	6	3	7	9	10	5	x	y	z

(2) 计算模：

$$Y = S \bmod 11$$

(3) 通过模得到对应的校验码

Y	0	1	2	3	4	5	6	7	8	9	10
校验码	1	0	x	9	8	7	6	5	4	3	2

例如：某男性的身份证号码为【53010219200508011X】。我们看看这个身份证是不是符合计算规则的身份证。

首先我们得出前 17 位的乘积和【(5*7)+(3*9)+(0*10)+(1*5)+(0*8)+(2*4)+(1*2)+(9*1)+(2*6)+(0*3)+(0*7)+(5*9)+(0*10)+(8*5)+(0*x)+(1*y)+(1*z)】是 189，然后用 189 除以 11 得出的结果是 189÷11=17 余 2，也就是说其余数是 2。最后通过对应规则就可以知道余数 2 对应的检验码是 X。所以，可以判定这是一个正确的身份证号码。

二、实验内容

已知有如下三个正确的身份证号码：

110101190001011009

236265196008162472

65256919760524176X

请编程实现身份证号码算法，求出第 15、16、17 位置上的加权因子 x、y、z。

图 2: 解密后的文档

2.3.2 身份证校验码

以下为该部分的全部代码, 后面会给出代码的部分解释及其功能。

```
1  #include<iostream>
2  #include<string>
3  #include<NTL/ZZ.h>
4  #include<NTL/ZZ_p.h>
5  #include<NTL/mat_ZZ_p.h>
6  #include<NTL/vec_ZZ_p.h>
7  using namespace std;
8  using namespace NTL;
9  struct equation// 定义同余方程结构体
10 {
11     mat_ZZ_p matrix;//模p下方程的系数矩阵
12     vec_ZZ_p vector;//模p下方程右边的列向量
13     equation(int num_x,int num_equation)//初始化
14     {
15         matrix.SetDims(num_equation, num_x);
16         vector.SetLength(num_equation);
17     }
18 };
19 equation input()
20 {
21     string a = "110101190001011009";
22     string b = "236265196008162472";
23     string c = "65256919760524176X";
24     equation eq(3, 3);
25     eq.vector[0] = 3;//反向查表得到效验码对应的Y
26     eq.vector[1] = 10;//初步初始化列向量
27     eq.vector[2] = 2;
28     int arr[14] = { 7,9,10,5,8,4,2,1,6,3,7,9,10,5 };//前14位的权重
29     int sum = 0;
30     for (int i = 0; i < 14; i++)
31     {
32         sum += arr[i] * (a.at(i) - 48);
33     }//48是由char转为int的ascii码值差
34     eq.vector[0] -= sum;//将左边的常数移到右边来
35     sum = 0;
36     for (int i = 0; i < 14; i++)
37     {
38         sum += arr[i] * (b.at(i) - 48);
39     }
40     eq.vector[1] -= sum;
```



```
41     sum = 0;
42     for (int i = 0; i < 14; i++)
43     {
44         sum += arr[i] * (c.at(i) - 48);
45     }
46     eq.vector[2] -= sum;
47     eq.matrix[0][0] = a.at(14) - 48;
48     eq.matrix[0][1] = a.at(15) - 48;
49     eq.matrix[0][2] = a.at(16) - 48;
50     eq.matrix[1][0] = b.at(14) - 48;
51     eq.matrix[1][1] = b.at(15) - 48;
52     eq.matrix[1][2] = b.at(16) - 48;
53     eq.matrix[2][0] = c.at(14) - 48;
54     eq.matrix[2][1] = c.at(15) - 48;
55     eq.matrix[2][2] = c.at(16) - 48;
56     return eq;
57 }
58 void solve_equation(equation eq)
59 {
60     vec_ZZ_p solution; // 解向量
61     solution.SetLength(3);
62     ZZ_p d = determinant(eq.matrix); // d是系数矩阵的行列式
63     solve(d, eq.matrix, solution, eq.vector); // 解方程Ax=b(mod p)
64     cout << "x = " << solution[0] << endl;
65     cout << "y = " << solution[1] << endl;
66     cout << "z = " << solution[2] << endl;
67 }
68 int main()
69 {
70     ZZ_p::init(ZZ(11)); // 设定模数为11
71     solve_equation(input());
72     return 0;
73 }
```

(1)

```
1 struct equation
2 {
3     mat_ZZ_p matrix; // 模p下方程的系数矩阵
4     vec_ZZ_p vector; // 模p下方程右边的列向量
5     equation(int num_x, int num_equation) // 初始化
6     {
7         matrix.SetDims(num_equation, num_x);
8         vector.SetLength(num_equation);
```

```
9     }  
10 };
```

首先构造一个结构体 *equation* 用来表示需要的方程结构并对相关矩阵以及向量初始化。

(2)

```
1 equation input()  
2 {  
3     string a = "110101190001011009";  
4     string b = "236265196008162472";  
5     string c = "65256919760524176X";  
6     equation eq(3, 3);  
7     eq.vector[0] = 3; // 反向查表得到效验码对应的Y  
8     eq.vector[1] = 10; // 初步初始化列向量  
9     eq.vector[2] = 2;  
10    int arr[14] = { 7,9,10,5,8,4,2,1,6,3,7,9,10,5 }; // 前14位的权重  
11    int sum = 0;  
12    for (int i = 0; i < 14; i++)  
13    {  
14        sum += arr[i] * (a.at(i) - 48);  
15    } // 48是由char转为int的ascii码值差  
16    eq.vector[0] -= sum; // 将左边的常数移到右边来  
17    sum = 0;  
18    for (int i = 0; i < 14; i++)  
19    {  
20        sum += arr[i] * (b.at(i) - 48);  
21    }  
22    eq.vector[1] -= sum;  
23    sum = 0;  
24    for (int i = 0; i < 14; i++)  
25    {  
26        sum += arr[i] * (c.at(i) - 48);  
27    }  
28    eq.vector[2] -= sum;  
29    eq.matrix[0][0] = a.at(14) - 48;  
30    eq.matrix[0][1] = a.at(15) - 48;  
31    eq.matrix[0][2] = a.at(16) - 48;  
32    eq.matrix[1][0] = b.at(14) - 48;  
33    eq.matrix[1][1] = b.at(15) - 48;  
34    eq.matrix[1][2] = b.at(16) - 48;  
35    eq.matrix[2][0] = c.at(14) - 48;  
36    eq.matrix[2][1] = c.at(15) - 48;  
37    eq.matrix[2][2] = c.at(16) - 48;  
38    return eq;
```

39 }

该部分为对方程进行填充，声明了方程的未知量个数以及其各个系数含义。

(3)

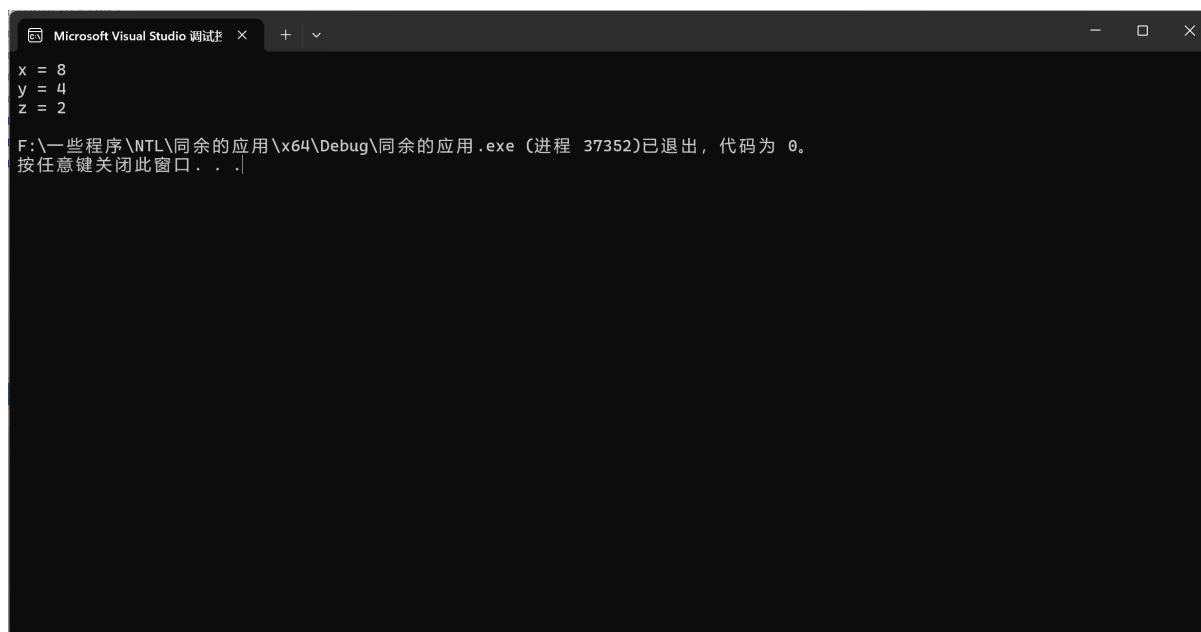
```
1 void solve_equation(equation eq)
2 {
3     vec_ZZ_p solution; // 解向量
4     solution.SetLength(3);
5     ZZ_p d = determinant(eq.matrix); // d是系数矩阵的行列式
6     solve(d, eq.matrix, solution, eq.vector); // 解方程 Ax=b(mod p)
7     cout << "x = " << solution[0] << endl;
8     cout << "y = " << solution[1] << endl;
9     cout << "z = " << solution[2] << endl;
10 }
```

然后运用 NTL 自带的 solve 函数解方程，特别地，解方程都是在模 11 的前提下进行。

(4)

```
1 int main()
2 {
3     ZZ_p::init(ZZ(11)); // 设定模数为 11
4     solve_equation(input());
5     return 0;
6 }
```

main 函数，设定模数为 11，运行上面定义的两个函数。以下为代码运行的结果：



```
Microsoft Visual Studio 调试  x + v
x = 8
y = 4
z = 2
F:\一些程序\NTL\同余的应用\x64\Debug\同余的应用.exe (进程 37352)已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

图 3: 运行结果

解得 x, y, z 的值分别为 8, 4, 2.

3 实验小结

3.1 凯撒密码安全性分析

凯撒密码 (Caesar cipher)，作为一种古老的加密技术，其历史可以追溯到古罗马时期。据说，尤利乌斯·凯撒 (Julius Caesar) 曾用此方法来保护他的军事通信。尽管这种方法在现代密码学中已不再使用，但它仍然是密码学教育中的一个重要案例，帮助我们理解加密技术的基本原理。

而其在现代密码学中不再使用的原因其实也是其安全性相较于其他更为先进的加密方式来说的欠缺。凯撒密码又被称为平移密码、凯撒平移，凯撒密码的安全性主要依赖于密钥的保密性。然而，由于其密钥空间非常有限（只有 26 种可能的偏移量），它很容易被破解。攻击者可以通过频率分析等统计方法来确定正确的偏移量。在英语中，某些字母（如 'E'）出现的频率较高，攻击者可以利用这些信息来猜测偏移量。

3.2 身份证校验码的作用

身份证校验码的实际意义在于确保身份证号码的**唯一性和准确性**，通过防止号码输入错误或伪造号码来提升身份证号码的安全性和可靠性。它有以下几个具体作用：

(1) 防止号码输入错误

在手动输入身份证号码时，容易出现误输入的情况。校验码通过加权计算和取模验证，可以检测出常见的输入错误，尤其是某一位数字输入错误或数字顺序错误时，系统会通过校验码的计算发现该错误。这在银行开户、购票、在线注册等需要身份证号码的场景中尤为重要。

(2) 防伪功能

校验码的存在使得伪造身份证号码变得更加困难。一个有效的身份证号码不仅需要符合号码结构的规则（如地区码、出生日期等），还必须通过校验码验证。因此，即使前 17 位号码伪造得看似合理，但如果校验码不正确，号码就会被判定为无效。这提升了身份证号码的防伪能力。

(3) 数据录入和核查的准确性

在大量身份证信息的管理和数据录入过程中，校验码有助于自动检测和过滤错误数据。政府、银行、企业等在处理公民或客户的身份证信息时，校验码为系统自动检测提供了基础，避免了大量错误或无效数据进入数据库。

(4) 信息验证中的自动化

各类线上或线下验证系统，如实名认证、网络购票、保险登记等，可以根据身份证号的校验码自动进行身份信息的初步筛查和验证，节省了人工审核的时间，提升了信息验证的效率。

(5) 保障公民隐私

虽然校验码的主要功能是验证身份证号的准确性，但它也间接帮助减少了使用虚假身份证号的风险，进而保护了公民的个人隐私。在需要身份证号码的交易或服务中，校验码确保了信息的真实性，防止冒名顶替的发生。

总的来说，身份证校验码不仅增强了身份证号码的安全性和防伪能力，还提高了数据处理和管理准确性，确保身份验证流程的高效性和可靠性。