# 《现代密码学》实验报告

| | |
|---|---|
| 实验名称：古典密码 | 实验时间：2024 年 10 月 7 日 |
| 学生姓名：黄集瑞 | 学号：22336090 |
| 学生班级：22 保密管理 | 成绩评定： |
| 由于本次古典密码实验包含两部分实验内 | 容，故全部完成在此篇报告中 |

## Assignment1、维吉尼亚密码分析

### 一、实验目的

通过实现维吉尼亚密码的密文分析，并且使用重合指数法找出密钥以及明文；可以帮助我们更好理解关于维吉尼亚密码的译码方法，有助于我们掌握多表代换加密算法的基本思想，提高我们对经典密码学算法的认知。

### 二、实验内容

- 用 C++实现维吉尼亚密码的密文分析
- 输入：给定密文（不改变明文格式），且密文和密钥的长度至少为 50:1。
- 输出：密钥以及所对应的明文

### 三、实验原理

维吉尼亚密码是一种多表代换密码，它基于凯撒密码的变体进行加密。它的核心思想是使用一个密钥来控制加密表的偏移，形成多个凯撒加密表。而我们要对其进行破解，首先就需要先求出其的密钥长度，这个也是破解该密码的关键；其次，在求出相对应的密钥长度后，我们便可以将密文进行对应的分组，相当于对多个单表代换进行操作，然后跟据重合指数法进一步的确认密钥的具体内容；最后，根据解码规则就可以将密文还原成明文了。

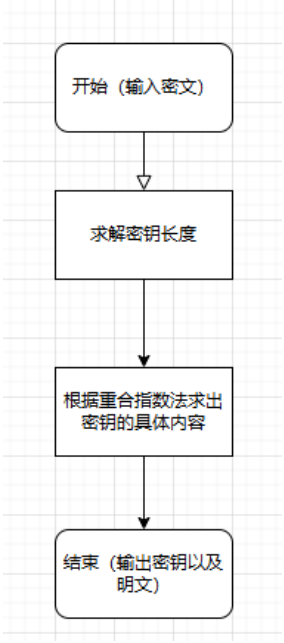


图 1. 具体实验流程

## 四、实验步骤（源代码）

- 密文输入：

```
string cyphertext, input;
while (cin >> input)
{
    cyphertext += input + ' ';
}
```

在尝试了多种输入方式后，发现只有这种方法能够处理好系统的输入。在此时，我并没有选择将输入的密文进行处理（诸如“统一大/小写化”、处理非字符内容），因为如果考虑到如果先对密文进行处理，那么到时候的复原将会是个比较大的问题，所以我直接保留了输入进行后续的处理。

- 获取密钥长度

```
// 课上提到密钥的长度可能会超过 30，这里长度设置为 35
double min_gap = 1;
int keylength = 0;
for (int i = 1; i < 36; i++)
{
    string *group = group_by(ct, i); // 得到分组后的结果
    double IC_total = 0;
    for (int j = 0; j < i; j++)
    {
        // 这里需要统计每个分组中每个字母出现的次数
        int packet = group[j].length();
        int frq[26] = {0};
```

```
            for (int k = 0; k < packet; k++)
            {
                // 统计每个字母出现的次数
                char tmp = group[j][k];
                if (isupper(tmp))
                {
                    frq[tmp - 'A']++;
                }
                else
                {
                    frq[tmp - 'a']++;
                }
            }
```

在求取密钥时，我是从密钥长度为 1 时开始测试，因为上课曾有提到最大的密钥长度仅会超过 30，所以就设置了 35 次来进行试验密钥长度，之后便是统计分组后的密文中各个字母所出现的次数。然后，根据该公式我们计算该每个分组的重合指数；最后，我们将所有分组的重合指数加起来来计算平均的重合指数。根据统计显示，有意义的文本的重合指数一般都应小于 0.065。

$$I_c(x)=\frac{\sum_{i=0}^{25}\binom{f_i}{2}}{\binom{n}{2}}=\frac{\sum_{i=0}^{25}f_i(f_i-1)}{n(n-1)}$$

```
            int sum = 0;
            //完全按照课本上的思路来进行计算
            for (int l = 0; l < 26; l++)
            {
                sum += frq[l] * (frq[l] - 1);
            }
            IC_total += sum * 1.0 / (packet * (packet - 1));
        }
        double avg_ic = IC_total / i;
        if (avg_ic >= BIC)        //其中 BIC 的值为 0.061
        {
            min_gap = abs(avg_ic - BIC);
            keylength = i;
            break;
        }
```

但是，我在判断时则是判断第一次大于我设置的 BIC 时就可以退出，此时返回的便是正确的密钥长度。虽然这个步骤可以说是为了通过样例而试验出来的，但是经过多次的失败后我也发现，其实密钥长度一般来说都是所求长度的公因子，所以说这个条件也能够使得密钥长度在比较小的时候就可以直接退出，也是有一定的合理性的。

- 求解密钥

用于我们在上面已经求解出了密钥长度，那么我们便可以得到对应的密钥分组，由于每个分组就对应一个密钥字母，所以我们需要对每个密钥分组都进行拟重合指数法，推测出最可能的密钥字母。对每一个分组，我们用‘A’-‘Z’去测试其的拟重合指数，并且将拟重合指数最高的字母当作密钥的字母。

```
        // 使用拟重合指数(IC)进行密钥推测
        double max_ic = 0.0;
        char key = 'A';

        // 尝试每一个可能的密钥字母（0 到 25 位移量）
        for (int g = 0; g < 26; g++)
        {
            double ic = 0.0;
            int total_count = 0;

            // 计算当前位移(g)下的拟重合指数
            for (int i = 0; i < 26; i++)
            {
                total_count += frq[i];
                ic += wf[i] * frq[(i + g) % 26]; //其中 wf 数组就是英文字母对应的频率表
            }

            ic /= total_count;

            // 更新最大拟重合指数，推测最可能的密钥字母
            if (ic > max_ic)
            {
                max_ic = ic;
                key = 'A' + g;
            }
        }
        keystring += key;
    }
```

- 译码输出

在得到了密钥长度以及具体的密钥内容后，我们便可以进行译码破解出对应的明文。对应每个字符，都根据密钥对应位置的字母计算偏移量，利用取模运算循环使用密钥中的字母，再减去密钥偏移量，通过取模运算确保结果落在合法范围内，最后输出相应内容即可。

```
    for (int i = 0; i < cyphertext.size(); i++)
    {
        if (isalpha(cyphertext[i]))
        {
```

```
            // 计算密钥字母的偏移量（将 key[count % keylength] 转为 0-25 的
数字）
            int key_offset = toupper(key[count % keylength]) - 'A';

            if (isupper(cyphertext[i]))
            {
                // 大写字母的解密公式
                cyphertext[i] = (cyphertext[i] - 'A' - key_offset +
26) % 26 + 'A';
            }
            else
            {
                // 小写字母的解密公式
                cyphertext[i] = (cyphertext[i] - 'a' - key_offset +
26) % 26 + 'a';
            }
            count++; // 只对字母进行计数
        }
    }
```

五、实验结果

```
K ccpkbgu fdp hq tyavinrrt mvg rkdnbv fd etdgi ltxrgu ddk o
tfmb pvge G ltg ck qracqc. Wdn awcrxiza kft lewrpt ycqkyvx ch
kft poncq qr hjv ajuwe tmcmspkq dy hjv dahct rls vskcgcz
qqdzxgsf rls wcwsjt bh. Afs iasp rjahk jrju mvg kmitz hfp
dispzlvl gwtf pl kkebdpg ceb shctj. Rwxb afs pezqn rwx cvycg
aonw ddk ackawbbi kft iovkcgg. Hjvl nhi ffsqes vyc lacnv rwbbi
repbb vf exos c dygzwp fd tkfqi. Ycw hjvl nhi qibtk hjv npist.
^Z
CRYPTO
I learned how to calculate the amount of paper needed for a room when I was at school. You multiply the square footage of the walls by the cubic contents of the floor and ceiling combined and double it.
You then allow half the total for openings such as windows and doors. Then you allow the other half for matching the pattern. Then you double the whole thing again to give a margin of error. And then you
 order the paper.
```

可以看到成功输出了对应密钥以及正确的明文内容。

六、实验总结

在本次实验中，通过对维吉尼亚密码的解密过程进行实现与验证，我更加深入理解了多表代换密码的基本原理和操作。在这当中，我遇到的最大的挑战就是一开始并不理解整个译码过程，但是在经过自己阅读课本以及请教同学后，最终也是成功理解了整个过程并成功完成实验。该实验也让我更好的理解了课堂上的知识，令我收益匪浅。

# Assignment2、仿射希尔密码分析

## 一、实验目的

通过实现仿射希尔密码的密文分析，可以帮助我们更好掌握矩阵运算在密码学中的应用，尤其是矩阵与向量的乘法关系，探索如何通过逆矩阵来实现解密，增强对线性代换密码的理解，锻炼我们对已知明文攻击的破解能力。

## 二、实验内容

- 用 C++实现仿射希尔密码的密文分析
- 输入：给定明文以及密文对，m 值。
- 输出：密钥矩阵的内容，以及 m 长度向量 b 的内容

## 三、实验原理

仿射希尔密码结合了**希尔密码**和**仿射密码**的加密思想，基于线性代数中的矩阵运算实现多字符加密，有效避免了实验 1 维吉利亚秘密容易遭到频率攻击的问题，而且本次实验的内容是仿射希尔密码，也就是在原先矩阵的操作上还引入了偏移量，增强了加密过程的复杂性。我们需要利用明文攻击的方法来对其进行破解，首先，由于矩阵的长度 m 是给定的，所以我们可以对密文进行分组重构得到两个方程 Y1=X1*L+b;Y2=X2*L+b;然后我们让两式相减这样就可以先忽略掉偏移量 b 的影响；接着，我们对处理好的方程进行求逆操作，得到完整的 key 矩阵后，带入原先的明密文对就可以求解出对应的偏移量 b，这样我们就完成了对该密码的破解。

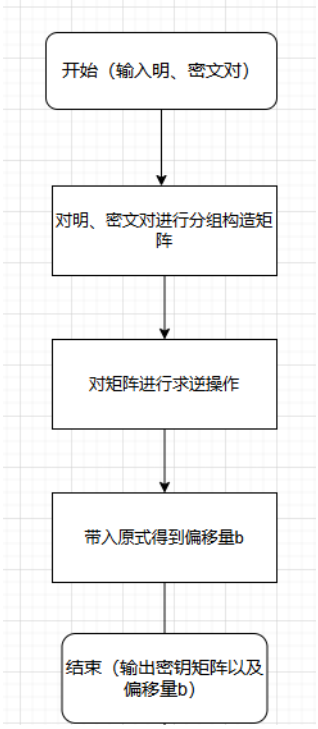


图 2 实验具体流程图

## 四、实验步骤（源代码）

- 对明、密文对进行分组

```
vector<vector<int>> group_by(string input1, int n)
{
    vector<vector<int>> group;
    int count = 0;
    vector<int> tmp;
    for (int i = 0; i < input1.length(); i++)
    {
        tmp.push_back((input1[i] - 'A') % 26);
        count++;
        // 这里默认如果不满足分组长度，则不进行添加
        if (count == n)
        {
            group.push_back(tmp);
            tmp.clear();
            count = 0;
        }
    }
    return group;
}
```

这一部分一开始其实只是按照顺序取出对应长度的数组然后直接拼接成 mxm 的矩阵，但是在后面矩阵求逆时发现有些矩阵的行列直接按照明、密文对顺序来组成的话是没有办法求出

逆矩阵的，所以我便对明、密文对进行初始的分组使其形成长度为 m 的向量，而不满足这个长度的话，就不将其纳入我们的初始分组。在处理完分组后，就要构建对应的 mxm 矩阵。

```
    int len = p.size();
    for (int i = len - 1; i > 0; --i)
    {
        int j = getRandomNumber(0, i);
        swap(p[i], p[j]);
        swap(c[i], c[j]);
    }
    vector<vector<int>> pm, cm, pm_, cm_;
    for (int i = 0; i < n; i++)
    {
        pm.push_back(p[i]);
        pm_.push_back(p[i + n]);
        cm.push_back(c[i]);
        cm_.push_back(c[i + n]);
    }
```

值得一提的是，在该函数中，当我得到了分好组的明、密文对时，我采取了生成随机数的方法来随机打乱原先的分组排序，这样是为了保障该矩阵最终计算出来是可逆。然后，将不同的分组分别塞入两个方程即上面提到的 X1、X2 中，为后面的相减做准备。

```
    // 让对应的数组进行相减
    vector<vector<int>> pm_result, cm_result;
    for (int i = 0; i < n; i++)
    {
        vector<int> temp1, temp2;
        for (int j = 0; j < n; j++)
        {
            temp1.push_back((pm[i][j] - pm_[i][j] + 26) % 26);
            temp2.push_back((cm[i][j] - cm_[i][j] + 26) % 26);
        }
        pm_result.push_back(temp1);
        cm_result.push_back(temp2);
    }//通过前面的这些步骤，我们便得到了相减的矩阵即 pm_result 以及 cm_result，
接着就可以进行求取矩阵逆的操作
```

- 矩阵逆操作

该部分我认为是本次实验的重点之一，在这部分中十分考验我们的矩阵操作，只有经过一系列的矩阵操作才可以得到最终的结果。求出矩阵的逆的方法有很多种，而我采用的是伴随矩阵求解逆矩阵的方法。

首先，我们需要对矩阵的行列式进行计算，这里我采用的是递归嵌套的方法。

```
// 递归计算行列式
int deta(const vector<vector<int>> &m, int n)
```

```
{ // 前面两个是一般情况，后面的则是进行递归操作
    if (n == 1)
    {
        return (m[0][0] % 26 + 26) % 26; // 1x1 矩阵的行列式
    }

    if (n == 2)
    {
        // 2x2 矩阵的行列式公式
        return ((m[0][0] * m[1][1] - m[0][1] * m[1][0]) % 26 + 26) % 26;
    }

    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        // 获取子矩阵
        vector<vector<int>> minorMatrix = get_sub_matrix(m, n, 0, i); //
去掉第 0 行和第 i 列

        // 递归计算子矩阵的行列式
        int cofactor = deta(minorMatrix, n - 1);

        // 符号为 (-1)^i，因此 alternation 依次为 1 和 -1
        int alternation = (i % 2 == 0) ? 1 : -1;

        // 计算当前项，并确保结果在 [0, 25] 范围内
        int currentTerm = (alternation * m[0][i] * cofactor) % 26;

        // 如果 currentTerm 为负数，加 26 再取模，保证非负
        if (currentTerm < 0)
        {
            currentTerm = (currentTerm + 26) % 26;
        }

        // 累加当前项
        sum = (sum + currentTerm) % 26;
    }

    // 确保 sum 非负
    return 26 - ((sum + 26) % 26);
}
```

其中，还编写了求出子矩阵的函数方法也就是 get_sub_matrix，可以通过该函数获取删掉指定行和列后的子矩阵。接着，就需要求出 deta 的模逆，在这里我使用了扩展欧几里得算

法来求出该模逆。

```
int modInverse(int a, int mod)
{
    a = a % mod;

    for (int x = 1; x < mod; x++)
    {
        if ((a * x) % mod == 1)
        {

            return x;
        }
    }
    return -1; // 如果没有逆元，返回 -1
}
```

可以注意到，如果行列式计算结果为 0 或者没有模逆的情况下，我们都是无法对我们相减后的矩阵进行逆运算以至于来求解我们的 key 矩阵：（Y2-Y1）=(X2-X1)*L 通过求解（X2-X1）在模 26 下的逆矩阵来解 L。这里便是我认为该实验的另外一个重点，老师在课上有提到过，一个矩阵如果当前的排列无法求出行列式的话，对其中的行列进行交换也是有很大概率可以求出行列式的。所以，我前面才使用分组形式以及引入随机数来对矩阵进行重排列，这样就可以保障矩阵一直在变换直到有逆之后才会退出运算。

```
    bool flag = true; //通过 flag 标志位的控制，让求矩阵逆的操作一直进行下去，直到该矩阵可以求出其逆为止
    while (flag)
    {
        int anwser = process(p, c, n);
        if (anwser == -1)
        {
            continue;
        }
        else
        {
            flag = false;
        }
    }
```

在得到 deta 的模逆之后，我们便可以计算出其的伴随矩阵然后在用行列式的模逆去相乘便可以得到最后的逆矩阵。

```
    // 4. 用行列式的逆乘以伴随矩阵的每个元素，并取模 26
    vector<vector<int>> inv(n, vector<int>(n));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
```

```
        {
            inv[i][j] = (adj[i][j] * det_inv) % 26;
            if (inv[i][j] < 0)
            {
                inv[i][j] += 26; // 确保非负
            }
        }
    }
    return inv;
}
```

- 求解 key 矩阵以及偏移量 b

求出逆矩阵之后，我们根据公式通过逆矩阵与密文矩阵进行运算后，就可以将原先的 key 矩阵求出来。

```
    // 计算密钥矩阵
    vector<vector<int>> key_matrix(n, vector<int>(n));

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < n; k++)
            {
                key_matrix[i][j] += pm_inv[i][k] * cm_result[k][j];
            }
            key_matrix[i][j] = (-(key_matrix[i][j] % 26) + 26) % 26;
        }
    }
```

得到 key 矩阵之后，我们只要将明文以及密文进行带入就可以直接得到偏移量 b 了，其中，我直接选取了明文矩阵的第一行带入便求出了偏移量 b。

```
    // 计算 key_matrix * pm[0]
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j < n; j++)
        {
            // 矩阵乘法 key_matrix 的第 i 行和 pm 的第 0 行的点积
            sum += pm[0][j] * key_matrix[j][i];
        }

        // 计算偏移量，先减去矩阵乘积的结果，再取模
        int diff = cm[0][t] - sum;

        // 保证结果为正数
```

```
        offset.push_back((diff % 26 + 26) % 26);

        t++;
    }
```

五、实验结果

```
3
ADISPLAYEDEQUATION
DSRMSIOPLXLJBZULLM
3 6 4
5 15 18
17 8 5
8 13 1
```

可以看到，至此实验成功完成。

六、实验总结

在本次实验中，我通过求解仿射希尔密码中的矩阵逆运算，成功加深了对线性代换密码和矩阵运算的理解。我也同时认识到矩阵运算在密码学中的重要性，尤其是在仿射希尔密码中，矩阵的逆运算决定了解密的可行性。本次实验并不同于课本上简单的希尔密码而是增加了偏移量，让整个实验富有挑战性。完成本实验后，我对古典密码的了解更进了一步，同时也提升了我的代码能力，锻炼了我的思维能力，让我受益匪浅。