Hill Cipher Program

By Mohamed Atef and Nihad Younis

Table Of Contents

Introduction

This program encrypts a given text using a 3x3 key using Hill cipher method. The program also decrypts an encrypted message provided the key is given.

Only alphabetical letters are encrypted and any other characters are omitted.

If the number of characters in the text is not a multiple of 3, the text is extended by wrapping around (1 or 2 letters).

Files for I/O

Provided with the program are the following 4 txt files:

- plainTxt.txt : source for encryption operation.
- encryptTxt.txt : target for encryption operation.
- decryptSrcTxt.txt : source for decryption operation.
- decryptTxt.txt : target for decryption operation.
-

The first two files (highlighted in green) are the files used when encrypting. The second two files (highlighted in purple) are used in decryption.

Program Guide

The program is self guiding as it prompts the user with messages each step indicating the required inputs. Nonetheless, here is a general guide on using it.

1. Place the text to be encrypted in the plainTxt.txt file, and the text to be decrypted in the decryptSrcTxt.txt file.
2. Run the program.

3. Choose encryption by writing ‘e’ at the prompt or decryption by writing ‘d’. If a different letter is entered, the user is given a notification message indicating an unknown input.
4. Upon selecting one of the two options, the user is prompted to write the key. The key is a 3x3 matrix and is entered in the following format:

   $$a_{00}\ a_{01}\ a_{02}\ a_{10}\ a_{11}\ a_{12}\ a_{20}\ a_{21}\ a_{22}$$

   to represent the following matrix:

   $$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

5. After entering the key, the selected operation is performed and the result is written in the corresponding target file.
6. The program then prompts the user to check if they want to restart it. If ‘r’ is written, the program restarts. Any other letter causes the program to terminate.

Figure 1 is an example screen of an interaction to perform an encryption process.

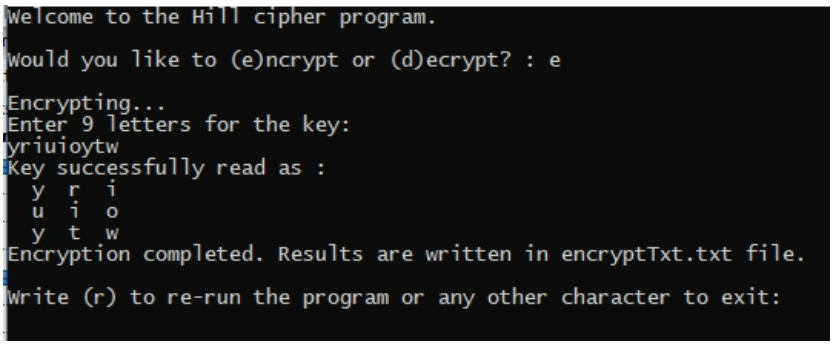

Figure 1

Program Operation and Explanation

Introduction:

The program main operation is to perform the following calculations:

$$C = E(K, P) = PK \bmod 26$$
$$P = D(K, C) = CK^{-1} \bmod 26 = PKK^{-1} = P$$

Where E(K, P) is the encryption function of the 3 letters P using the 3x3 key K. And D(K, C) is the function for decrypting the 3 letters C using the 3x3 key K.
Thus, to perform the operations, the program must be able to perform matrix operations.

An important note to consider is if the total number of letters to be encrypted is not a multiple of 3, what should be used to complete P (and C) to 3 letters? The two options considered are either to pad the end of each file by a specific latter ('a' for example), or use letters from the file by wrapping around. In this program, the latter solution is used.

Matrix Class:

To perform the matrix operations, a Matrix class is used. The class represents a 2 dimensional char matrix of specified rows and columns. Some of the features given by the class are:

- Matrix by Matrix multiplication using the * operator.
- Matrix by integer multiplication using the * operator.
- Matrix by integer modulo using the % operator.
- Determinant function to calculate the determinant of the Matrix.
- Get and Set functions to set specific elements of the Matrix.
- Multiple constructors for creating the Matrix.
- Destructor for preventing data leaks.

The detailed C++ implementation is found in Matrix.h file.

Modulo header:

To aid in modulo arithmetic, a file called "modulo.h" is included in the previously mentioned Matrix class and the soon to be mentioned Hill class. The header file is very simple and consists of one function and one preprocessor macro.

```
#define mod(a,b) (b+a%b)%b
int modInverse(int a, int m);
```

The function, modInverse, obtains the multiplicative inverse of a under modulo m. Its implementation is very simple and is found in "modulo.cpp" file.
The preprocessor macro is very important and is used in different occasions to replace the standard C++ modulo operator '%'. The reason is that the modulo operator in C++ produces wrong mathematical results when 'a' is negative in 'a%b'. In C++, *(-a)%b == -(a%b)*, which is not the desired result (review modulo arithmetic for more info). *Please note that using macros may result in hard to debug problems if not used carefully.*

The macro used obtains the correct mathematical result and is thus used whenever there is a chance a negative number is used in the mod operation.

Hill Class:

A class called hillCipher is used to implement the functions encrypt and decrypt.
The class definition is as follows:

```
class hillCipher
{
private:
public:
    string encrypt(string plain, string key);
    string decrypt(string crypt, string key);
};
```

The function encrypt takes a *string plain* for the text and a *string key* for the key. Function decrypt is very similar and is self explanatory.

Encrypt:

Before encrypting, we generate a key matrix using the input key string to convert from the format

$a_{00}\ a_{01}\ a_{02}\ a_{10}\ a_{11}\ a_{12}\ a_{20}\ a_{21}\ a_{22}$ to $\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$

The key is then checked for inverse condition to be met (has a non zero modulo determinant).
To encrypt a file, we loop through the file encrypting each 3 letters. In the following C++ code snippet, we use the modulo operator '%' to wrap around when encrypting. We finally return the encrypted string.

(!) Note that throughout this document, some code may wrap around to the next line which will cause issues if directly copied into a code editor (for example in single line comments and strings). This is only for preview purposes and in the actual code everything is well sorted

```
string hillCipher::encrypt(string plain, string key)
{
    //Generate matrix from the given key.
    Matrix keyM(3, 3);
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
```

```
            keyM.set(i, j, (key[3 * i + j] - 'a') % 26);

    //Test if the key is invertible.
    if (! (mod(keyM.determinant(), 26)) )
    {
        cout << "ERROR: Key is not invertible, aborting encryption (ret
urning a 0 string)..." << endl;
        return "\0";
    }

    string encrypt;

    //Loop through the plain text and encrypt each 3 characters.
    for (int i = 0; i < plain.length(); i += 3)
    {
        Matrix result(1, 3), letters(1, 3);

        //Modulo to wrap around the last letters in case the number of
letters is not a multiple of 3.
        letters.set(0, 0, plain[(i) % plain.length()] - 'a');
        letters.set(0, 1, plain[(i + 1) % plain.length()] - 'a');
        letters.set(0, 2, plain[(i + 2) % plain.length()] - 'a');

        result = letters * keyM;
        result = result % 26;

        //Add the result to the encrypt string.
        encrypt += toupper(result.get(0, 0) + 'a');
        encrypt += toupper(result.get(0, 1) + 'a');
        encrypt += toupper(result.get(0, 2) + 'a');
    }
    return encrypt;
}
```

Decrypt:

For decryption, the function is very similar. However, instead of multiplying the key to the input string, we multiply the multiplicative inverse of the key under modulo 26 with the input string.

The following code snippet previews the decrypt function.

```
string hillCipher::decrypt(string crypt, string key)
{
    //Generate matrix from the given key.
    Matrix keyM(3, 3);
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
            keyM.set(i, j, (key[3 * i + j] - 'a') % 26);

    //Test if the key is invertible.
    if (! (mod(keyM.determinant(), 26)) )
    {
        cout << "ERROR: Key is not invertible, aborting decryption (ret
urning a 0 string)..." << endl;
        return "\0";
    }

    //Obtain inverse of the matrix
    int b = modInverse(int(keyM.determinant()), 26);
    keyM = (keyM.adjoint() * b) % 26;

    //Decrypt using the inverse key matrix
    string decrypt;

    //Loop through the crypted text and decrypt each 3 characters.
    for (int i = 0; i < crypt.length(); i += 3)
    {
        Matrix result(1, 3), letters(1, 3);

        //Modulo to wrap around the last letters in case the number of
letters is not a multiple of 3.
        letters.set(0, 0, crypt[(i) % crypt.length()] - 'a');
        letters.set(0, 1, crypt[(i + 1) % crypt.length()] - 'a');
        letters.set(0, 2, crypt[(i + 2) % crypt.length()] - 'a');

        result = letters * keyM;
        result = result % 26;

        //Add the result to the decrypt string.
        decrypt += tolower(result.get(0, 0) + 'a');
        decrypt += tolower(result.get(0, 1) + 'a');
        decrypt += tolower(result.get(0, 2) + 'a');
    }
    return decrypt;
}
```

One should notice that after decrypting the file, the last one or two letters may resemble the first one or two letters. This occurs if wrap around is performed during encryption.

Last page