Hill Encryption: The Intersection of Linear Algebra and Basic Cryptography (A Java Programming Project)

Om Kashyap Avashia
Freshman, Computer Science (Hons.), B.S., Ira A. Fulton Schools of Engineering, Arizona State University
Calculus II For Engineers Honors Enrichment Project
Professor Iulia Inozemtseva
Spring 2024

Abstract

Strong encryption is the backbone of modern society. Every bit of personal information exchanged on the internet, without strong encryption, would land in the hands of malicious people, groups, or even states. The strongest modern encryption algorithm is called the RSA algorithm (named after its developers **R**ivest, **S**hamir, and **A**dleman), and, in very simple terms, it is based on the principle that it is easy to multiply large numbers, but factoring large numbers is very difficult[1].

Since the mathematical concepts involved in the RSA algorithm are out of the scope of this paper, I will be discussing an encryption algorithm – called the Hill Cipher – that is not used industrially but serves as an important milestone in our path to understanding RSA and encryption algorithms in general. Furthermore, this paper will include a Java-based program that encrypts plaintext messages using the Hill encryption algorithm.

[1] RSA Encryption | Brilliant Math & Science Wiki. https://brilliant.org/wiki/rsa-encryption/

Encrypting plaintext using the Hill Cipher

In a word, the Hill cipher converts a plaintext message into a numeric matrix, and then multiplies it with another matrix (a publicly available key) to encrypt the plaintext. My program will take a user-entered plaintext message, convert it into an ASCII[2]-based matrix, and encrypt 3 letters at a time (since the public key is a 3x3 matrix, it can encrypt only 3 letters at a time). For instance, if the message is "*abcdef*" my program will first convert it into two 1x3 matrices with each element being a character's ASCII-equivalent integer value.

This way, "*abc*" becomes $[97 \quad 98 \quad 99]$ and "*def*" becomes $[100 \quad 101 \quad 102]$, since the ASCII value of "a" is 97, "b" is 98, and so on.

Next, the public key *k* will be set to a 3x3 matrix $k = \begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix}$ and *k* will then be used to encrypt the message by multiplying the ascii-converted matrix with the key, and performing the **mod** operation.

The **mod** operation essentially results in the remainder of a number to the left of the **mod** operator when divided by the number to the right of the **mod** operator. For example,

$$52 \; mod \; 26 = 0$$

Or,

$$53 \; mod \; 26 = 1$$

The operation will help us keep the resultant (encrypted) matrix within the bounds of the ASCII-character values for lowercase alphabets, thus making it easier to encrypt and decrypt the message.

Using these ideas, we multiply the "abc" matrix with *k,*

$$\begin{bmatrix} 97 & 98 & 99 \end{bmatrix} \begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix} \mod 26 + 97$$

[2] ASCII stands for *American Standard Code for Information Interchange*. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@'

Which will give us the final encrypted matrix for “abc”, which we can call μ,

$$\mu = \begin{bmatrix} 97 & 98 & 99 \end{bmatrix} \begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix} \mod 26 + 97 = \begin{bmatrix} 590 & 300 & 787 \end{bmatrix} \mod 26 + 97$$

And thus, the final encrypted matrix for “abc” is:

$$\begin{bmatrix} 115 & 111 & 104 \end{bmatrix} = \mu$$

Using this algorithm, we can convert each matrix element into its corresponding letter-value using the ASCII table. Thus, *abc* → *soh* and similarly, *def* → *kxf* ⇒ *abcdef* becomes *sohkxf*.

Another message, such as “I love pancakes” will be encrypted to: yqdgbnqatqfango.

We have now encrypted a message using the Hill cipher! This encrypted text is also referred to as cipher text.

Decrypting the Hill Cipher

But how do we decrypt this? We have to reverse the entire process – and while I won't be covering the second part of the reversal, because that goes into number theory, I will be discussing the first: finding the inverse of our public key *k.*

Say $k^{-1} = \phi$, which is the inverse of our matrix. The formula for the inverse of *k* is,

$$k^{-1} = \phi = \frac{1}{\det(k)} \text{adj}(k)$$

Where ***det(k)*** is the determinant of the matrix *k* and ***adj(k)*** is the adjoint of the matrix *k*. Obviously, ***det(k)*** $\neq 0$ for *k* to be invertible.

Finding the determinant

To compute the determinant, we start with the first element in our matrix (here, it's 1 red text), and then multiply that with the difference in the cross products of the numbers in the matrix that do not belong to the same row or column as this first element (blue text).

$$\begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix}$$

And so we get our first expression,

$$\det(k) = 1[(1 \times 5) - (4 \times 1)] = 1$$

This process is repeated for the second and third elements in our first row. Every additional expression added to our determinant will alternate in sign (positive or negative). For instance, for a matrix *m,*

$$\det(m) = a - b + c - d$$

where a,b,c,d are the values derived using the process mentioned above.

Here's how we find the inverse of *k* step by step (notice the signs highlighted in yellow):

$$\begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix}$$

$$\det(k) = 1 - (-2)[(2 \times 5) - (3 \times 1)] = 1 - (-2)(7) = 15$$

Finally,

$$\begin{bmatrix} 1 & -2 & 2 \\ 2 & 1 & 1 \\ 3 & 4 & 5 \end{bmatrix}$$

$$\det(k) = 15 + 2[(2 \times 4) - (3 \times 1)] = 15 + 2(8 - 3) = 25$$

Therefore, $\det(k) = 25$.

Finding the adjoint

To define the adjoint of a matrix, we must first define it's *transpose*[3] and *cofactor.* The transpose is simple,

$$k^T = \begin{bmatrix} 1 & 2 & 3 \\ -2 & 1 & 4 \\ 2 & 1 & 5 \end{bmatrix}$$

For the cofactor, however, Consider the following matrix, where the first subscript number denotes the row number, and the second one denotes the column number. For instance, A_{12} represents a matrix element in the first row, second column of a matrix. Using this definition, we define *k* our 3x3 public key matrix, as:

$$k = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

[3] The transpose of a matrix is simple: each row becomes a column, and each column becomes a row. The transpose is denoted by a superscript *T*.

The formula for each element of the cofactor matrix of *k* is:

$$A_{mn} = (-1)^{m+n} \times \det(A_{mn})$$

Using this formula, we get the cofactor matrix:

$$\psi = \begin{bmatrix} 1 & -7 & 5 \\ 18 & -1 & -10 \\ -4 & 3 & 5 \end{bmatrix}$$

To calculate the adjoint, we must first find the cofactor matrix, then transpose it. Thus,

$$\psi^T = \begin{bmatrix} 1 & 18 & -4 \\ -7 & -1 & 3 \\ 5 & -10 & 5 \end{bmatrix}$$

Now all that's left to do is multiply the determinant and the adjoint. Therefore, our inverse matrix is:

$$k^{-1} = \frac{1}{25} \begin{bmatrix} 1 & 18 & -4 \\ -7 & -1 & 3 \\ 5 & -10 & 5 \end{bmatrix}$$

A short note on modulo operators and congruences

Now that we have found our inverse matrix, we must use mathematical tools from crypotgraphy and number theory to fully decrypt our encrypted text. Since we want all our messages to be within the 26-alphabet range we can't have an inverse matrix with non-integer entries. As is obvious from the inverse matrix above, we have non-integer entries.

Since we're dealing with matrices and a **mod** operator, using simple equations to decrypt our cipher text won't work. We need to use the concept of congruences instead. According to Khan Academy, for a statement $A \equiv B$ (read: A is congruent to B) a congruence is defined by: The values A and B are in the same **equivalence class.**

> *An equivalence class is the name given to a subset of some equivalence relation R which includes all elements equal to each other.*

Loosely put, a congruence between two values suggests that they are related to each other by means of some operation, and thus lie in the same set of equivalent values. For this specific example, the operation we can refer to is the **mod 26** operation.

Now that we know how congruences work, here's how we fix the issue of the non-integer entries in our inverse matrix: we need to replace the non-integer value (here, $\frac{1}{25}$) with a value or operation that is **congruent** to 25. Fortunately, since $25 \equiv 1 \ mod\ 26$, we can replace $\frac{1}{25}$ with $mod\ 26$. This, however, only works for values of **det(k)** which follow the relation $gcd(\det \mathrm{k}, 26) = 1$.

Now that we have this, we can rewrite our inverted key matrix like so:

$$k^{-1} = \begin{bmatrix} 1 & 18 & -4 \\ -7 & -1 & 3 \\ 5 & -10 & 5 \end{bmatrix} \mod 26$$

And multiply each of the ciphertext values to retrieve the plaintext ones. For any public-key matrix for a hill cipher using the English alphabet to be valid, it must follow $gcd(\det \mathrm{k}, 26) = 1$, without which, decryption won't be possible, since the inverted matrix will not have any integer values.

The Program

```
import java.util.ArrayList;
import java.util.Scanner;

public class encrypt
{
    public static void main(String[] args)
    {
        //scanner declaration
        Scanner sc = new Scanner(System.in);

        //asking for user input
        System.out.println("Enter a message to encrypt: ");
        String message = sc.nextLine();

        // key matrix
        int [][] key = {{1,0,0},{0,1,0},{0,0,1}};


        //encrypts message
        System.out.println("Encrypting...");
        String encryptedMessage = hillEncrypt(message);

        //display encrypted message

        System.out.println(encryptedMessage);

    }

    // return final encrypted string
    public static String hillEncrypt(String input)
    {
        // character to add in case the stirng without spaces is not a
perfect multiple of 3
        final String nullChar = "x"; // option-shift-\

        //remove spaces
        input = removeSpaces(input);
        int requiredLength = input.length();

        //checking if requiredLength is a multiple of three.
        // if not, add nullChar till it becomes a multiple of 3
        if(input.length()%3!=0)
        {
            int counter = 0;
            do
            {
                requiredLength++;
            }while (requiredLength%3!=0);

            int diff = requiredLength - input.length();
            do
            {
                input = input.concat(nullChar);
                counter++;
```

```
            }while (counter < diff);
        }


        String encrypted = "";

        //adding 3 characters at a time
        for(int i=0;i<requiredLength;i=i+3)
        {
            encrypted =
encrypted.concat(matrixEncrypt((input.substring(i,i+3))));
        }

        return encrypted;
    }

    // return encrypted substring
    public static String matrixEncrypt(String a)
    {

        //Declaring key matrix
        final int [][] Key = new int[3][3];

        //Initializing key matrix
        Key[0][0] = 1;
        Key[0][1] = -2;
        Key[0][2] = 2;
        Key[1][0] = 2;
        Key[1][1] = 1;
        Key[1][2] = 1;
        Key[2][0] = 3;
        Key[2][1] = 4;
        Key[2][2] = 5;

        //Storing ASCII Values

        int [] charValues = new int[3];
        charValues[0] = (int)a.charAt(0);
        charValues[1] = (int)a.charAt(1);
        charValues[2] = (int)a.charAt(2);


        int [] encryptedString = new int[3];

        //Encrypting (multiplies with key matrix)

        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                encryptedString[i] += charValues[j] * Key[j][i];
            }
        }

        //Converting to alphabet
        for(int i=0;i<3;i++)
        {
```

```
            encryptedString[i] %=26;
        }

        for(int i=0;i<3;i++)
        {
            encryptedString[i] +=97;
        }

        String encrypted="";
        for(int i=0;i<3;i++)
        {
            encrypted =
encrypted.concat(Character.toString((char)encryptedString[i]));
        }

        return encrypted;
    }

    //removes spaces
    public static String removeSpaces(String message)
    {
        // trims the string
        message=message.trim();

        //new string with no spaces
        String modifiedMessage="";

        //removes spaces
        for(int i =0;i<message.length();i++)
        {
            if(Character.isWhitespace(message.charAt(i)))
            {
                modifiedMessage =
modifiedMessage.concat(Character.toString(message.charAt(i+1)));
                i++;
            }
            else
                modifiedMessage =
modifiedMessage.concat(Character.toString(message.charAt(i)));
        }

        //returns the no spaces string
        return modifiedMessage;
    }
}
```

Test Cases/Sample Outputs:

1. This message will be encrypted
   Output: jsrmxmmnymhsvsafxwaxfrwylos

2. ASU is #1 in innovation
   Output: shoytgrfzfcmoytjfiiqe

Conclusion

While Hill and Caesar ciphers are fascinating in that they offer insight into earlier forms of cryptography, in the modern world, they are very easily breakable and thus never used for encrypting real-world data.

In the real world, complex (and elegant) algorithms like RSA-232 encryption are used to encrypt data. To put it in a few words, the RSA-232 encryption method has a large integer number (232 digits long) as a public key – and the code can only be broken (that is, the private keys can be discovered) only when this large integer is factorized *completely*. The two prime numbers making up this public key are what the private keys are

It is obviously impossible for a human brain to factorize integers that are 232 integers long, but even supercomputers would take (and I am not kidding) 300 trillion years to break, which is why it is effectively impossible to determine the two private keys (two prime numbers multiplying to give the 232-digit public key). This is why RSA is so effective.

For now, understanding these two forms of encryption is a step forward for me in the world of math-based cryptography, and I am grateful to have had an opportunity to do so.