

REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188		
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) 28-08-2012		2. REPORT TYPE Related Material		3. DATES COVERED (From - To) -	
4. TITLE AND SUBTITLE Study of the Hill Cipher Encryption/Decryption Algorithm			5a. CONTRACT NUMBER W911NF-11-1-0174		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER 206022		
6. AUTHORS Melvin Steven Hernández, Dr. Alfredo Cruz (Advisor)			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Polytechnic University of Puerto Rico 377 Ponce De Leon Hato Rey San Juan, PR 00918 -			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211			10. SPONSOR/MONITOR'S ACRONYM(S) ARO		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S) 58924-CS-REP.17		
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT The Hill Cypher was Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. As stated before, the Hill Cipher is an encryption algorithm that uses polygraphic substitution ciphers based on linear algebra concepts. Each letter is encoded as a number. In the case of the English alphabet, letters are usually represented by the following scheme: A =0, B=1, C=2 ... Z=25. The message that is to be encrypted/decrypted will be held in a block of n letters and multiplied by a					
15. SUBJECT TERMS Polygraphic cypher; encryption; decryption; ASCII table					
16. SECURITY CLASSIFICATION OF:		17. LIMITATION OF ABSTRACT		15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU	UU		Alfredo Cruz
					19b. TELEPHONE NUMBER 787-622-8000

Report Title

Study of the Hill Cipher Encryption/Decryption Algorithm

ABSTRACT

The Hill Cypher was Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. As stated before, the Hill Cipher is an encryption algorithm that uses polygraphic substitution ciphers based on linear algebra concepts. Each letter is encoded as a number. In the case of the English alphabet, letters are usually represented by the following scheme: A =0, B=1, C=2 ... Z=25. The message that is to be encrypted/decrypted will be held in a block of n letters and multiplied by a square matrix using modulo of the amount of letters in the alphabet in use. In this research project, the main goal is to recreate this cipher using the values of the ASCII table. Since the Hill Cipher works by assigning a numeric value to the characters that are to be used, using the ASCII table seems like a perfect fit for this application.

Study of the Hill Cipher Encryption/Decryption Algorithm

By

Melvin Steven Hernández Negrón

Under the Guidance of
Alfredo Cruz PhD.



Polytechnic University of Puerto Rico

April 2012

Introduction

The following report depicts the work employed in the research of the Hill Cipher encryption/decryption algorithm. But before any explanation regarding this subject is made, we must first step back a little to explain the foundations of this algorithm, which is in the field of Cryptography.

What is Cryptography?

It is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication. One of the algorithms inside of the field of study is the Hill Cipher.

The Hill Cipher

Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. As stated before, the Hill Cipher is an encryption algorithm that uses polygraphic substitution ciphers based on linear algebra concepts. Each letter is encoded as a number. In the case of the English alphabet, letters are usually represented by the following scheme: A =0, B=1, C=2 ... Z=25. The message that is to be encrypted/decrypted will be held in a block of n letters and multiplied by a square matrix using modulo of the amount of letters in the alphabet in use. In this research project, the main goal is to recreate this cipher using the values of the ASCII table. Since the Hill Cipher works by assigning a numeric value to the characters that are to be used, using the ASCII table seems like a perfect fit for this application.

What is Polygraphic Substitution?

In a polygraphic substitution cipher, plaintext letters are substituted in large groups, instead of substituting letters individually. Some examples of this type of substitutions are the Four-Square Cipher, the Playfair Cipher and of course, the Hill Cipher.

The Advantages of the Polygraphyc Substitution instead of individual substitution is that its frequency distribution is much flatter than that of individual characters, though not completely

flat in real languages; for example in the English language, “TH” is much more common than “XQ”. Also the larger number of symbols requires correspondingly more ciphertext to productively analyze letter frequencies.

The Hill Cipher Encryption Process

Each letter is first encoded as a number. The most common scheme used being:

$$A = 0, B = 1, \dots, Z = 25$$

Then, the message that is to be encrypted will be held in a block of n letters considered as a vector of n dimensions. It will then be multiplied by an $n \times n$ matrix known as the key matrix. Then the result will be converted with modulo 26 (in this case, since the alphabet has 26 letters). This will yield the cipher text.

Mathematical Process

Given the plain text message: “*paymoremoney*”

Encoding to the message to numbers yields:

$$P = 15, 0, 24, 12, 14, 17, 4, 12, 14, 13, 4, 24$$

Then for example using the key:

$$K = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

We’ll want to get the cipher text which the formula is:

$$C = PK \text{ mod } 26$$

Then since the key is a 3×3 matrix, then we can use the first three letters of the message and multiply them by the key matrix like so:

$$\begin{aligned} C &= (15, 0, 24) \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \\ &= (303, 303, 531) \text{ mod } 26 \\ &= (17, 17, 11) = RRL \end{aligned}$$

Repeating this process with obtain that the encrypted string is:

$$C = RRLMWBKASPDH \text{ or in numbers } C = 17, 17, 11, 12, 22, 1, 10, 0, 18, 15, 3, 7$$

The Hill Cipher Decryption Process

To decrypt, we hold the cipher text in a vector of n dimensions, like we did with the plain text. Then we multiply by the inverse of the key matrix. This in reality is not a regular inverse matrix. It is heavily dependant in the modulo being used. Then we will convert the resultant matrix with modulo 26. This yields the original message again.

Mathematical Process

Given the cipher text in numerical value:

$$C = 17, 17, 11, 12, 22, 1, 10, 0, 18, 15, 3, 7$$

We want to obtain the plain text which is:

$$P = CK^{-1} \text{ mod } 26$$

But we need to get the inverse of the key matrix. This process has some differences to the normal means of getting a matrix inverse. With the key we do the adjoint of each element and then transpose it and we obtain:

$$\begin{pmatrix} 300 & -313 & 267 \\ -357 & 313 & -252 \\ 6 & 0 & -51 \end{pmatrix}$$

Now what is different is the way the determinant is computed, we have that normally our determinant would be:

$$\det K = -939$$

But the Hill Cipher uses a modulo so the real determinant would have to be converted with modulo 26, to obtain its correct values, hence we have:

$$-939 \text{ mod } 26 = -939 - 26 \times \text{floor}\left(\frac{-939}{26}\right) = 23$$

Or simply:

$$(-939 \text{ mod } 26) + 26 = -3 + 26 = 23$$

Then the multiplicative inverse of the determinant must be computed which yields:

$$23^{-1} \text{ mod } 26 = 17$$

Because the modulo of the product between the determinant and the multiplicative inverse yields 1

$$(23 \times 17) \bmod 26 = 1$$

Then we multiply the adjoint matrix by the multiplicative inverse with the modulo and we obtain the inverse of the key matrix:

$$K^{-1} = 17 \begin{pmatrix} 300 & -313 & 267 \\ -357 & 313 & -252 \\ 6 & 0 & -51 \end{pmatrix} \bmod 26$$

$$K^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

And now is just a simple mean of multiplying the cipher text vectors with the inverse key matrix.

$$P = (17 \ 17 \ 11) \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} \bmod 26 = (15 \ 0 \ 24)$$

$$(15 \ 0 \ 24) = P \ A \ Y$$

Which ends up giving us our original message "paymoremoney".

Work with the Hill Cipher

The project consists of using the Hill Cipher but instead of using the English alphabet which consists of 26 characters, the ASCII chart will be used which consists of 128 characters. These include upper case letters, lower case letters numbers and special characters. This will increase the complexity of the encryption/decryption process. Ultimately the goal is to be able to encrypt and decrypt a large document which has ASCII-only characters.

For this test the following document has been used:

document.txt

To whom it may concern:

My name is Melvin Steven Hernandez Negron. I am a graduate student currently pursuing a master's degree in Software Engineering at the Polytechnic University of Puerto Rico. My plans for this year are to continue pursuing my master's degree in Software Engineering at the Polytechnic University of Puerto Rico and in April be able to take the Fundamental Engineering License Exam, this is because I want to better myself as much as possible, and become a professional at what I do. Depending on what happens on my master's degree and other aspects of my personal life I may decide to pursue a doctorate degree although I still am not completely certain. I hope that in studying my master's and working on this research, if I'm selected, I will have a better picture of what my next move will be, be it continue my doctorate or join the workforce.

I have a high sense of duty for my country and I wish to aid it and not be part of its problem. I want to be able to work on something that brings progress to this country, that means something, that helps mankind, not only myself. But for that I have the need to keep on learning as much as I can, so my knowledge is used for the greater good.

Receiving the support that this fellowship provides may very well help me pay for some of the costs of my studies so that I may be able to accomplish them. This fellowship may as well bring me the opportunity to start my thesis research since I am supposed to work for 20 hours per week if I am selected, and the opportunity to participate in a summer internship, something I have yet to do, but have been really interested in doing. Also this research would give me more work experience which is something I am looking for.

Obtaining this fellowship would help me to some extent load off some of the costs that my studies are placing me. If I were to receive this fellowship, it would motivate me because I would mean that someone feels that I have what it takes, that believes in me, that thinks I can make a difference. When I graduated from my Bachelor's degree of Science in Computer Engineering I ended up with a 3.39 Grade Point Average (GPA) out of a 4.0 scale which gave me the honor of being Cum Laude undergraduate student, it is certainly not the best, and I know that but I want to improve upon it and better myself along the way. I know that I can succeed and shape a better future not only for myself, but for those around me. Thank you for your time, attention and consideration.

This document is saved in a ".txt" file the it is stored in the root of the project, the program will access it alongside the key file. In this example the key file is:

key.txt

17 17 5

21 18 21
2 2 19

After the Encryption Process the program stores the encrypted message into another file, the result is as follows:

EncryptedDocument.txt

```
lpS3N__I|=aq$ __t|7_DB__x;_%',eU#hF;Tp,J__4oi:O{1_uF'|Ea2Kpf__9_TJL;[
@_?twB      cH?chw_+C_|_6
_1~7_->:j_k_XM_8U2u_u_a2_Fm_;b%v__^gMT,8\W_?>%VsP_Jb38M_8}Z1t_\u
_-i_C_f@vY]}}^W__u_\o_ :M_*_w!o#CavMiHh=)M}Z;_/qy#e_v_5UZ_kdyV
=]5Ee$M_Mi_B5Ui_/,>__Qq: #_Hop] 8
+Kwb> "?usj_L9T: lci__I_a
_Ly5YI_NK__m_%${P1v_uvCq_:Sztav_)Ki3T(t_{,6%n!j?uW_^gU_E
L[tN)Y6[5_E_B?[]*=]5qB1_<!sP_#Y_
I!i_I_a
_Lyo4r!W__[]BG$dt__k0.#hFtN)rO_,}hg b__&atN)_&G,qBiKJ_Z
G4=_Jj[#C/4#[5_
u?uWtN)_;!_7L4dF_U_d
y']d_.oBb9E"_]}_Ny~23_f#_b;Uu`\\|_E"_j2[j_] $u%[_j[]h_u_a2_Fm_;b%v_qN?"
BQ_NKt_
_Mn}!!| :M<_JO__f_x_N
GO 'Ss$ 323__wK@_&a:j_k_X_7L2__u_0oLK=_Wb>_"5q]m
R_gk_[])ne_.o9Yk#G_;!_R
U&_o_Vw!oj/,@`q_NeyJY~F[]&F/TtO_6I_G$Qq: #_Hop]
8_%_B_:f_d_Y^Uu`Vvbk0.Dny.[]_X_^~Gx__H__D<_4A_q]}\\!t_<ti
_U2u3_Oa2_"R_%"_lc}Kily_.N{_%'u
mo(N_0w\\!t_0_
u
\_ =aq
=4-r
N_rj[]h_8h3fU?ch_R10r_?us_NKA\_%d_[]^__}IiEi_U2uS_Jn_1!W_3SV.N
;_H\*;;,qo_VTun_*_qN?Vv[]bP.k3'_B?V_n%jYqN? @>#G_
\_I&_2R!8Xic
_(RG(dEC9Gg b__&atN)Y6[5_E_B?f_d_9tTtO_S_V_Ly~F[]8*
P__-R(RG{%\Ry__B?_V_)8Tun_*_t_E"[_]_tm_Ln?_a&O3/[]F[]$D_q-
%_ =vM \G}[]h_&#G_ =s5_#_BiKf4dt.9 @vy#e~F[]f_Oj2[_0w_NKp&%)
_B?.[]n_
FfJ_kdGm9YV_ =v4
{X_]_}dA_r__Yyb[]nW
=_Q`l,(O\~%7_0&y#e_NKwB__o
Oy3P_[~tx_R<>_XM_8t_\;["U_I!K!t_\5YI_V__/qi%Y_#we*u4dF_}$S$FSD_5?_+N-
HbH_XZUj
g5Ui_32Ex_m_LF_J}Ki_NK_t|
1L :M<w_+K_Hd
H~1:~F[]f_OSDm
C_7L(dE_&a[8n_;!_ZWk3'_NKC9Gn6w_/qi%Y_#we*uSDk
01Q4i%Y_h_M_8[___t_\3SV~.__$mS__1_e_T
j/.N{ 1_hZ(O\A_kd[#W_a_m;[@CcJU_I_? ,L0m
```

```
_9v\*;_6_ @H\}:#C/Oy3zK_
-_x_ oL,Y__;>E_yNn/B_:_NK_N
o"A)J o__&a_B_h_U\?ch_^gU2u_7oj;`_^ga2_
4M_1_m_Ln?_a&O_ )_j2[_0w:
B_&a2__PpB --I&?_9_Z+lDny%aE{_H_@)A_E_y_^g2__M_80PYy
Ht_\(O\A__kd[#Wl__t0`%b/O0/O0_-_A\_Xam!a
__7XCcnK~Zzs)Ih_S__V__Ly;[@5UEP_7M_8\*;lb>1
1j/,M_8t_\(O\g8K5h)E~_%PqH_4Lq-%[___&am_LF_Ju
m#Y_Z_9e9y_W_m_LF_J}Ki_NK_t|
1Lt_\5YIj[h_5r
P_%_lc_J?Q6Qq:;1_wE+g
bY_R_&aDny.[]_0w_V_/qi%Y_#we*u^~GBb9PqH_>+5Yz3__/_O0_9_mK~%7]_]l__
t0H__VvbE"_m_Ld5'W(OY*F[_k~F[]_Oj2[_0wly_&F/<\__A+[];~F[]_8*c_ -
K&_5*Hh]
u~F[]\_a&OJqf?_f7mK_e_7L&z_n<[7_-8   ""_{1_Gg2O,_"[])}
~(_7WX-Mc`7<o+_+6v_rs{%\3SV,Gz?
7_D_^g}4\W_a2_%VsP_Jb38M_8]_]_9k)}
5eYl1Jj[+_9P%S}p~=_pv_6
IgrU2T%E_IYF7W* --}Kiy_a>.6[_5_EA   &[#W8_:/_O0=]5qB1__My#e}Ki_n~_Lyf
[]:_C_|_6cQ"3s_[]
@=_m_5rI_g_@=aq(O\@_{X_X}:c<\_qB1l<_@?uW]_][]Zo_E"_nYf_O%@_6
_B?J_N_L?_Hg_u%jYqN?_(Z!E_Oy3_%_]_gmJ_3iVt_\Ccn:O<]_][]Zo_E"_]_]dA_y
>|S
)}
?uW_Gk_U2u3_Oa2_h6-1RS1QC#G_ =s5u_y#e_%_]_gPpB --Ex_t_\_?_yV
Tun_>+;1_^&_U_M_A'\*;$DU\}:v_o^/.}Z1Y*7/t_0PyB_:
=4_MHQ{/t_TJL
```

As we can see the message is completely unreadable, it may even appear as if this is not a ".txt" file, the only way to obtain the message again is if we have the proper key that has to be inversed and converted like the process showed. As to not be redundant the decrypted message will not be displayed here, since it is exactly the same as the original message, the program that has been annexed alongside this document can be used to test the feature, the decrypted message can be found on "DecryptedFile.txt".

Screen Captures

Here there are a few screenshots of the terminal window of the program displaying the encrypted and decrypted messages:

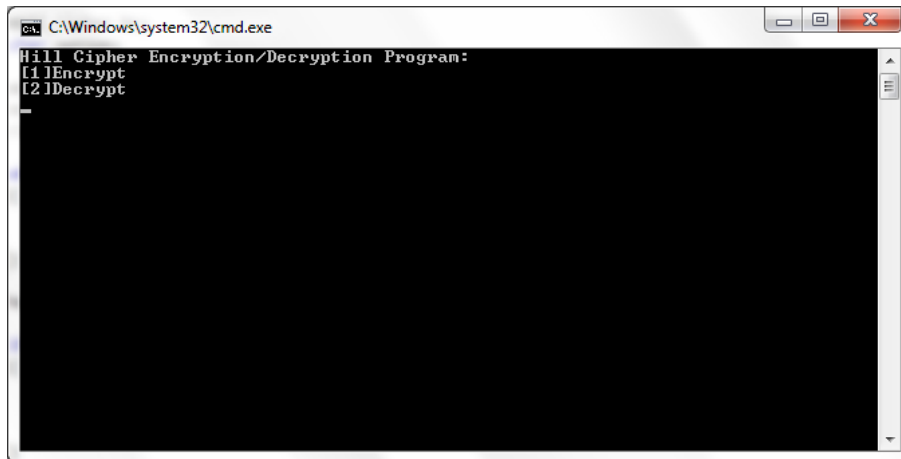


Image 1: Basic Menu of the Hill Cipher Program

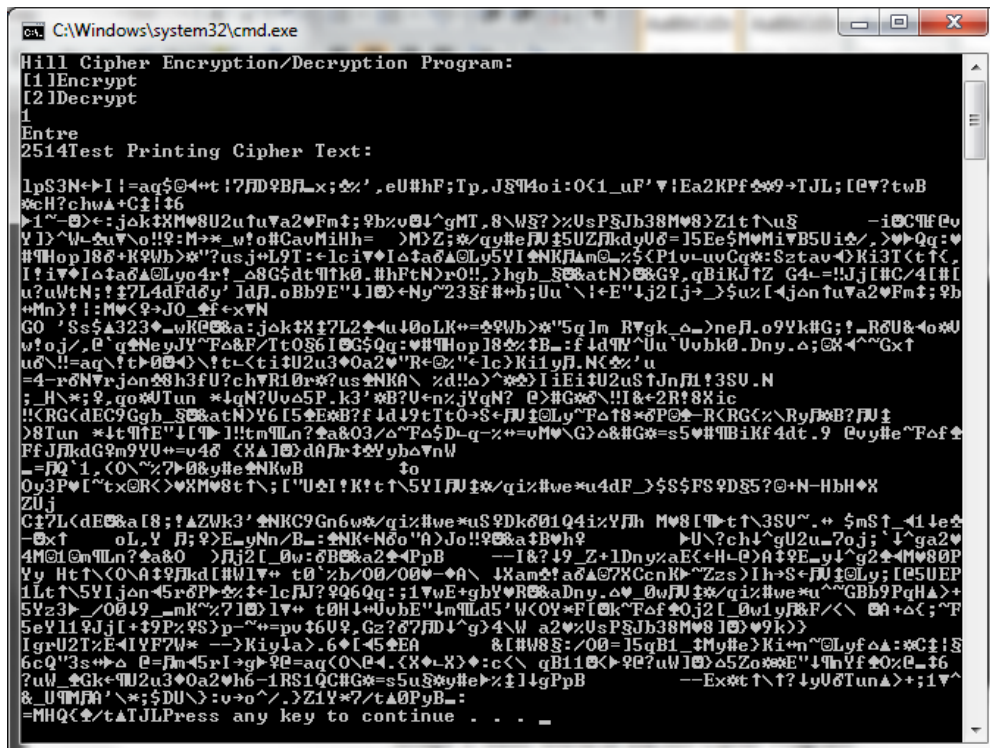
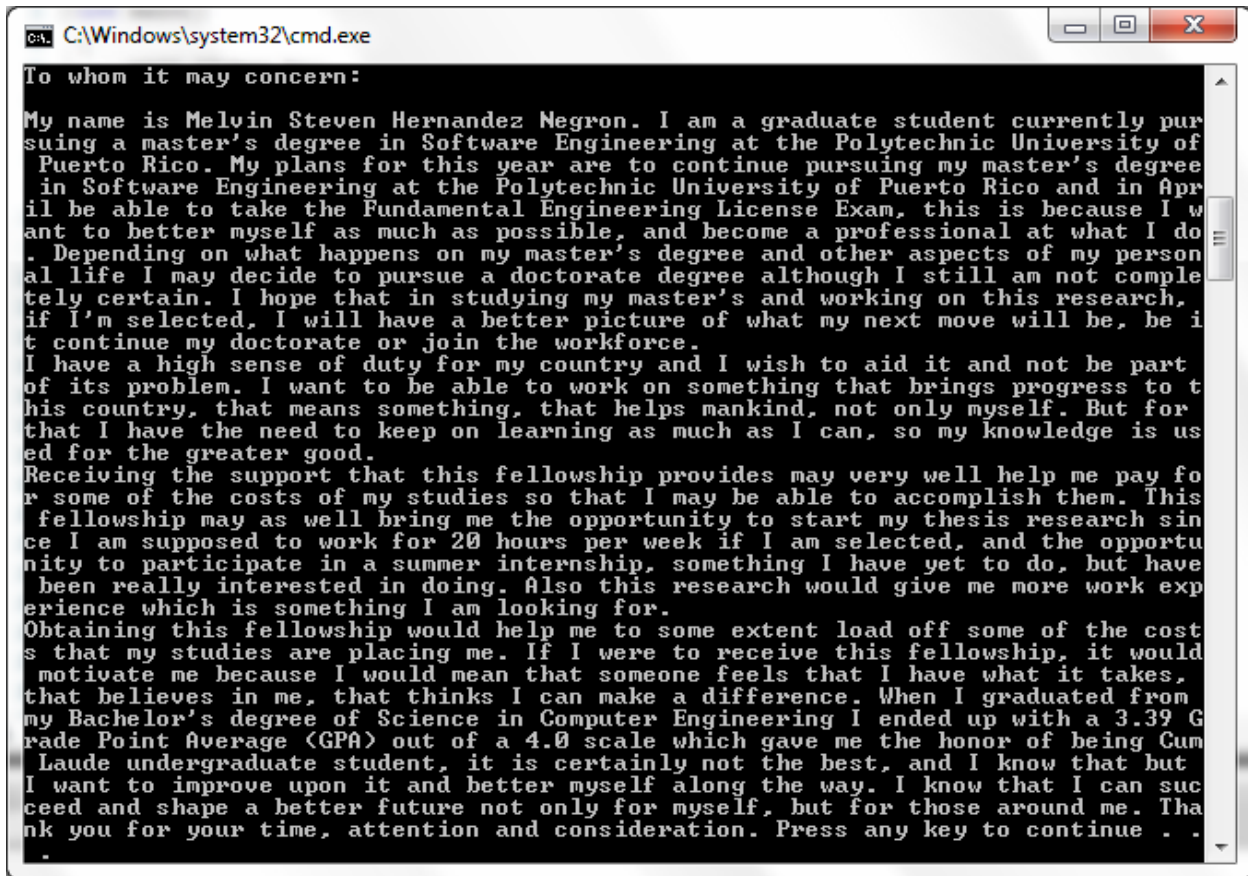


Image 2: After inputting the encrypt option in the program, it proceeds to look for the document.txt and encrypting it, displaying its encrypted message before storing it in another ".txt" file

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains a large block of text that has been decrypted. The text is a letter from Melvin Steven Hernandez Negron, a graduate student at the Polytechnic University of Puerto Rico. He discusses his plans to take the Fundamental Engineering License Exam, his interest in pursuing a doctorate, and his desire to contribute to his country and mankind. He also mentions receiving support for his studies and participating in a summer internship. The text ends with 'Thank you for your time, attention and consideration. Press any key to continue . . .'.

```
C:\Windows\system32\cmd.exe
To whom it may concern:

My name is Melvin Steven Hernandez Negron. I am a graduate student currently pursuing a master's degree in Software Engineering at the Polytechnic University of Puerto Rico. My plans for this year are to continue pursuing my master's degree in Software Engineering at the Polytechnic University of Puerto Rico and in April be able to take the Fundamental Engineering License Exam, this is because I want to better myself as much as possible, and become a professional at what I do. Depending on what happens on my master's degree and other aspects of my personal life I may decide to pursue a doctorate degree although I still am not completely certain. I hope that in studying my master's and working on this research, if I'm selected, I will have a better picture of what my next move will be, be it continue my doctorate or join the workforce. I have a high sense of duty for my country and I wish to aid it and not be part of its problem. I want to be able to work on something that brings progress to this country, that means something, that helps mankind, not only myself. But for that I have the need to keep on learning as much as I can, so my knowledge is used for the greater good.

Receiving the support that this fellowship provides may very well help me pay for some of the costs of my studies so that I may be able to accomplish them. This fellowship may as well bring me the opportunity to start my thesis research since I am supposed to work for 20 hours per week if I am selected, and the opportunity to participate in a summer internship, something I have yet to do, but have been really interested in doing. Also this research would give me more work experience which is something I am looking for.

Obtaining this fellowship would help me to some extent load off some of the costs that my studies are placing me. If I were to receive this fellowship, it would motivate me because I would mean that someone feels that I have what it takes, that believes in me, that thinks I can make a difference. When I graduated from my Bachelor's degree of Science in Computer Engineering I ended up with a 3.39 Grade Point Average (GPA) out of a 4.0 scale which gave me the honor of being Cum Laude undergraduate student, it is certainly not the best, and I know that but I want to improve upon it and better myself along the way. I know that I can succeed and shape a better future not only for myself, but for those around me. Thank you for your time, attention and consideration. Press any key to continue . . .
```

Image 3: After inputting the decrypt command in the program, it then proceeds to decrypt the message and displays it on the screen before storing it on the DecryptedFile.txt file

Future Work

One of the possible future works this project could be the parallelization of the whole process. This may prove really useful when encrypting/decrypting huge amounts of text, as with today's technology with dual-core, quad-core, and the increasing amount of cores not only Central Processing Units (CPUs), but Graphical Processing Units (GPUs) that are getting very popular lately, because this could remarkably accelerate the amount of time needed for such a task.

Code

Hill_Cipher.h

```
#pragma once
#ifndef HILLCIPHER_H
#define HILLCIPHER_H

#include "main.h"

class Hill_Cipher: public Memory_Allocation
{
private:
    int **cipherText;
    int **plainText;
    int **key;
    int **invKey;
    int size;
    int ptSize;
    int mod;

public:
    Hill_Cipher();

    /* Set Functions */
    void setPlainText(char *pT, long length); //void setPlainText(char *pT, long
length);
    void setKey(int **k, int size);
    void setCipherText(int *cT, long length);
    void setSize(int colLength);
    void setPtSize(int rowLength);
    void setMod (int modulo);

    /* Get Functions */
    int** getPlainText();
    int** getKey();
    int** getCipherText();
    int getSize();
    int getPtSize();
    int getMod();

    /*File Handling Functions*/
    void OpenPlainTextFile(char *fileName);
    void OpenCipherTextFile(char *fileName);
    void OpenKeyFile(char *fileName);
    void WritePlainTextFile();
    void WriteCipherFile();

    /* Hill Cipher Encrypt and Decrypt functions */
    void Encrypt();
    void Decrypt();

    ~Hill_Cipher();
};
#endif
```

Hill_Cipher.cpp

```
#include "main.h"

Hill_Cipher::Hill_Cipher()
{
    mod = 128; //Number of Ascii Characters
}

void Hill_Cipher::setPlainText(char *pT, long length)
{
    plainText = MxN_Matrix(ptSize, length);

    for(int i = 0; i < ptSize; i++)
    {
        for(int j = 0; j < length; j++)
        {
            plainText[i][j] = (int)pT[i];
        }
    }

    free(plainText);
}

void Hill_Cipher::setKey(int **k, int size)
{
    key = NxN_Matrix(size);

    for(int i = 0; i < size; i++)
    {
        for(int j = 0; j < size; j++)
        {
            key[i][j] = k[i][j];
        }
    }
}

void Hill_Cipher::setCipherText(int *cT, long length)
{
    cipherText = MxN_Matrix(ptSize, length);

    for(int i = 0; i < ptSize; i++)
    {
        for(int j = 0; j < length; j++)
        {
            cipherText[i][j] = cT[i];
        }
    }
}

void Hill_Cipher::setPtSize(int rowLength)
{
    ptSize = rowLength;
}

void Hill_Cipher::setSize(int collength)
{
    size = collength;
}
```

```

}

void Hill_Cipher::setMod(int modulo)
{
    mod = modulo;
}

int** Hill_Cipher::getPlainText()
{
    return (plainText);
}

int** Hill_Cipher::getKey()
{
    return (key);
}

int** Hill_Cipher::getCipherText()
{
    return (cipherText);
}

int Hill_Cipher::getSize()
{
    return(size);
}

int Hill_Cipher::getPtSize()
{
    return(ptSize);
}

int Hill_Cipher::getMod()
{
    return(mod);
}

void Hill_Cipher::OpenPlainTextFile(char *fileName)
{
    cout<<"Entre"<<endl;
    fstream text;
    char asciiChar;
    int fileLength = 0, counterRow = 0, counterCol = 0, counter = 0;
    text.open(fileName);

    if(!text)
    {
        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }

    while(!text.eof())
    {
        text.read((char*)&asciiChar, sizeof(char));

        if(!text.eof())
        {
            fileLength++;

```

```

    }
}

text.close();

cout<<fileLength;
ptSize = fileLength/size;
plainText = MxN_Matrix(ptSize,size);

text.open(fileName);

if(!text)
{
    cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
    exit(1);
}

while(!text.eof() && counter < fileLength)
{
    text.read((char*)&asciiChar, sizeof(char));
    plainText[counterRow][counterCol] = (int)asciiChar;

    if(!text.eof())
    {
        if(counterCol < size-1)
        {
            counterCol++;
        }
        else
        {
            counterRow++;
            counterCol = 0;
        }
    }

    counter++;
}

text.close();
}

void Hill_Cipher::OpenCipherTextFile(char *fileName)
{
    cout<<"Entre"<<endl;
    ifstream text;
    char asciiChar;
    int fileLength = 0, counterRow = 0, counterCol = 0, counter = 0;
    text.open(fileName);

    if(!text)
    {
        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }

    while(!text.eof())
    {

```



```

        text.read((char*)&asciiChar, sizeof(char));

        if(!text.eof())
        {
            fileLength++;
        }
    }

    text.close();

    cout<<fileLength;
    ptSize = fileLength/size;
    cipherText = MxN_Matrix(ptSize,size);

    text.open(fileName);

    if(!text)
    {
        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }

    while(!text.eof() && counter < fileLength)
    {
        text.read((char*)&asciiChar, sizeof(char));
        cipherText[counterRow][counterCol] = (int)asciiChar;

        if(!text.eof())
        {
            if(counterCol < size-1)
            {
                counterCol++;
            }
            else
            {
                counterRow++;
                counterCol = 0;
            }
        }

        counter++;
    }

    text.close();
}

void Hill_Cipher::OpenKeyFile(char *fileName)
{
    ifstream text;
    char verSpace;
    char asciiChar[20];
    int keySize = 1, counterRow = 0, counterCol = 0;
    text.open(fileName);

    //asciiChar = AssignCharMemory(9);
    if(!text)
    {

```

```

        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }

    while(!text.eof())
    {
        text.read((char*)&verSpace, sizeof(char));

        if(verSpace == ' ' || verSpace == '\n')
        {
            keySize++;
        }
    }

    keySize = sqrt((double)keySize);

    key = NxN_Matrix(keySize);
    size = keySize;

    text.close();
    text.open(fileName);

    while(text>>asciiChar)
    {
        key[counterRow][counterCol] = atoi(asciiChar);

        if(counterCol < keySize-1)
        {
            counterCol++;
        }
        else
        {
            counterRow++;
            counterCol = 0;
        }
    }
    text.close();
}

void Hill_Cipher::WritePlainTextFile()
{
    fstream text;

    text.open("DecryptedFile.txt", ios::in|ios::out);

    for(int i = 0; i <ptSize; i++)
    {
        for(int j = 0; j <size; j++)
        {
            text.write((char*)&plainText[i][j], sizeof(char));
        }
    }
    text.close();
}

void Hill_Cipher::WriteCipherFile()
{
    fstream text;

```

```

text.open("EncryptedDocument.txt", ios::in|ios::out);

    for(int i = 0; i < ptSize; i++)
    {
        for(int j = 0; j < size; j++)
        {
            text.write((char*)&cipherText[i][j], sizeof(char));
        }
    }
text.close();
}

void Hill_Cipher::Encrypt()
{
    Matrix_Algebra encrypt;

    cipherText = MxN_Matrix(ptSize,size);
    encrypt.MatrixMultiplication(plainText,key,cipherText,ptSize,size,size,size);

    cout<<"Test Printing Cipher Text:"<<endl<<endl;
    for(int i = 0; i < ptSize; i++)
    {
        for(int j = 0; j < size; j++)
        {
            cipherText[i][j] %= mod;
            cout<<(char)cipherText[i][j];
        }
    }
}

void Hill_Cipher::Decrypt()
{
    Matrix_Algebra decrypt;

    //deallocateNxN_Matrix(plainText,ptSize);
    plainText = MxN_Matrix(ptSize,size);
    invKey = NxN_Matrix(size);

    decrypt.InverseMatrix(key,invKey,size,mod);
    decrypt.MatrixMultiplication(cipherText,invKey,plainText,ptSize,size,size,size);

    cout<<"Test Printing Plain Text:"<<endl<<endl;
    for(int i = 0; i < ptSize; i++)
    {
        for(int j = 0; j < size; j++)
        {
            plainText[i][j] %= mod;
            cout<<(char)plainText[i][j];
        }
    }
}

Hill_Cipher::~Hill_Cipher()
{
    deallocateNxN_Matrix(cipherText,ptSize);
}

```

MemoryAllocation.h

```
#pragma once
#ifndef MATRIXALGEBRA_H
#define MATRIXALGEBRA_H

#include"main.h"

class Matrix_Algebra: public Memory_Allocation
{
public:
    Matrix_Algebra();

    void MatrixMultiplication(int **matrixA, int **matrixB, int **multResultant, int
rowsA, int columnsA, int rowsB, int columnsB);
    void MatrixCofactor(int **matrix, int**cofactorMatrix, int n);
    void MatrixTranspose(int **matrix,int **transposedMatrix, int n);
    void InverseMatrix(int **matrix, int **invResultant, int n, int modulo);
    int Determinant(int **matrix, int n);

    ~Matrix_Algebra();
};
#endif
```

MemoryAllocation.cpp

```
#include"main.h"

Memory_Allocation::Memory_Allocation()
{
}

char* Memory_Allocation::AssignCharMemory(long length)
{
    char *space;

    space = (char*)calloc(length,sizeof(char));

    return (space);
}

double* Memory_Allocation::AssignRealMemory(long quantity)
{
    double *memory;

    memory = (double*)calloc(quantity,sizeof(double));

    return (memory);
}

int* Memory_Allocation::AssignIntMemory(long quantity)
{
    int *memory;

    memory = (int*)calloc(quantity,sizeof(int));
}
```

```

    return (memory);
}

char** Memory_Allocation::CharMxN_Matrix(long rows, long columns)
{
    char **matrix;
    long i;

    matrix = (char**)calloc(rows,sizeof(char*));

    for(i=0;i<rows;i++)
    {
        matrix[i] = (char*)calloc(columns,sizeof(char));
    }

    return (matrix);
}

int** Memory_Allocation::MxN_Matrix(long rows, long columns)
{
    int **matrix;
    long i;

    matrix = (int**)calloc(rows,sizeof(int*));

    for(i = 0; i < rows; i++)
    {
        matrix[i] = (int*)calloc(columns, sizeof(int));
    }

    return (matrix);
}

double** Memory_Allocation::RealMxN_Matrix(long rows, long columns)
{
    double **matrix;
    long i;

    matrix = (double**)calloc(rows,sizeof(double*));

    for(i = 0; i < rows; i++)
    {
        matrix[i] = (double*)calloc(columns, sizeof(double));
    }

    return (matrix);
}

char** Memory_Allocation::CharNxN_Matrix(long rows_columns)
{
    char **matrix;

    matrix = CharMxN_Matrix(rows_columns, rows_columns);

    return (matrix);
}

int** Memory_Allocation::NxN_Matrix(long rows_columns)

```

```

{
    int **matrix;

    matrix = MxN_Matrix(rows_columns, rows_columns);

    return (matrix);
}

double** Memory_Allocation::RealNxN_Matrix(long rows_columns)
{
    double **matrix;

    matrix = RealMxN_Matrix(rows_columns, rows_columns);

    return (matrix);
}

void Memory_Allocation::delocateCharMemory(char *memory)
{
    free(memory);
}

void Memory_Allocation::delocateIntMemory(int *memory)
{
    free(memory);
}

void Memory_Allocation::delocateRealMemory(double *memory)
{
    free(memory);
}

void Memory_Allocation::delocateNxN_Matrix(int **matrix, long rows)
{
    for(int i = 0; i < rows; i++)
    {
        free(matrix[i]);
    }

    free(matrix);
}

void Memory_Allocation::delocateRealNxN_Matrix(double **matrix, long rows)
{
    for(int i = 0; i < rows; i++)
    {
        free(matrix[i]);
    }

    free(matrix);
}

void Memory_Allocation::delocateCharNxN_Matrix(char **matrix, long rows)
{
    for(int i = 0; i < rows; i++)
    {
        free(matrix[i]);
    }
}

```

```

    free(matrix);
}

Memory_Allocation::~Memory_Allocation()
{
}

```

MatrixAlgebra.h

```

#pragma once
#ifndef MATRIXALGEBRA_H
#define MATRIXALGEBRA_H

#include "main.h"

class Matrix_Algebra: public Memory_Allocation
{
private:
    //int ** multResultant;
    //int ** invResultant;

public:
    Matrix_Algebra();

    void MatrixMultiplication(int **matrixA, int **matrixB, int **multResultant, int
rowsA, int columnsA, int rowsB, int columnsB);
    void MatrixCofactor(int **matrix, int**cofactorMatrix, int n);
    void MatrixTranspose(int **matrix,int **transposedMatrix, int n);
    void InverseMatrix(int **matrix, int **invResultant, int n, int modulo);
    int Determinant(int **matrix, int n);

    ~Matrix_Algebra();
};
#endif

```

MatrixAlgebra.cpp

```

#include "main.h"

Matrix_Algebra::Matrix_Algebra()
{
}

void Matrix_Algebra::MatrixMultiplication(int **matrixA, int **matrixB, int
**multResultant, int rowsA, int columnsA, int rowsB, int columnsB)
{
    for(int i = 0; i < rowsA; i++)
    {
        for(int j = 0; j < columnsB ; j++)
        {

```

```

        multResultant[i][j] = 0;
    }
}

for(int i = 0; i < rowsA; i++)
{
    for(int j = 0; j < columnsB ; j++)
    {
        for(int k = 0; k < columnsB; k++)
        {
            multResultant[i][j] += matrixA[i][k]*matrixB[k][j];
        }
    }
}

}

void Matrix_Algebra::MatrixCofactor(int **matrix,int** cofactorMatrix, int n)
{
    int shiftrow;
    int shiftcol;
    int **adjointMatrix;

    adjointMatrix = NxN_Matrix(n-1);

    for ( int g = 0; g < n; g++ )
    {
        for ( int h = 0; h < n; h++ )
        {
            shiftrow = 0;
            shiftcol = 0;

            for (int i = 0; i < n; i++ )
            {
                for (int j = 0; j < n; j++ )
                {
                    if ( i != g && j != h )
                    {
                        adjointMatrix[shiftrow][shiftcol] =
matrix[i][j];

                        if ( shiftcol < n-2 )
                        {
                            shiftcol++;
                        }
                        else
                        {
                            shiftcol = 0;
                            shiftrow++;
                        }
                    }
                }
            }

            if((g+h) % 2 != 0)
            {
                cofactorMatrix[g][h] = Determinant( adjointMatrix, n-1 ) * -1;
            }
            else

```



```

        {
            cofactorMatrix[g][h] = Determinant( adjointMatrix, n-1 );
        }
    }
}

delocateNxN_Matrix(adjointMatrix, n-1);
}

void Matrix_Algebra::MatrixTranspose(int **matrix,int **transposedMatrix, int n)
{
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            transposedMatrix[i][j] = matrix[j][i];
        }
    }
}

void Matrix_Algebra::InverseMatrix(int **matrix,int **invResultant, int n, int modulo)
{
    int **cofactorMatrix;
    int **transposedMatrix;
    int det; //Determinant
    int modDet; //Modulo Determinant
    int multInv = 0; //Multiplicative Inverse of Modulo Determinant

    cofactorMatrix = NxN_Matrix(n);
    transposedMatrix = NxN_Matrix(n);

    MatrixCofactor(matrix,cofactorMatrix,n);
    MatrixTranspose(cofactorMatrix,transposedMatrix,n);

    det = Determinant(matrix,n);
    modDet = (det % modulo) + modulo;

    while((modDet*multInv) % modulo != 1)
    {
        multInv++;
    }

    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            invResultant[i][j] = (transposedMatrix[i][j] * multInv) % modulo;

            if(invResultant[i][j] < 0)
            {
                invResultant[i][j] += modulo;
            }
        }
    }

    delocateNxN_Matrix(cofactorMatrix, n);
    delocateNxN_Matrix(transposedMatrix, n);
}

```

```

}
int Matrix_Algebra::Determinant(int **matrix, int n)
{
    int det = 0;
    int pivot;
    int shift;
    int **reducedMatrix;

    /*If the square matrix is greater than 2x2 we need to reduce it*/
    if(n > 2)
    {
        reducedMatrix = NxN_Matrix(n-1);

        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < n-1; j++)
            {
                for(int k = 0; k < n-1; k++)
                {
                    if(k != i)
                    {
                        if(k == i+1)
                        {
                            shift = (k*2) - i;
                            reducedMatrix[j][k] = matrix[j+1][shift];
                        }
                        else
                        {
                            if(k > i+1)
                            {
                                shift = k + 1;
                                reducedMatrix[j][k] =
matrix[j+1][shift];
                            }
                            else
                            {
                                reducedMatrix[j][k] =
matrix[j+1][k];
                            }
                        }
                    }
                    else
                    {
                        shift = k + 1;
                        reducedMatrix[j][k] = matrix[j+1][shift];
                    }
                }
            }
        }

        /* if the column is even it is +, if not it is - */
        if(i % 2 == 0)
        {
            det += matrix[0][i] * Determinant(reducedMatrix, n-1);
        }
    }
}

```

```

        else
        {
            det += (matrix[0][i] * -1) * Determinant(reducedMatrix, n-1);
        }
    }

    delocateNxN_Matrix(reducedMatrix, n-1);
}

else
{
    /* [a b] determinant = ad - bc
       [c d] */

    pivot = (matrix[0][0] * matrix[1][1]) - (matrix[0][1] * matrix[1][0]);

    det = pivot;
}

return (det);
}

Matrix_Algebra::~Matrix_Algebra()
{
}

```

FileHandler.cpp

```

#include "main.h"

FileHandler::FileHandler()
{
}

void FileHandler::OpenFile(char *fileName)
{
    ifstream text;
    char asciiChar, bah;
    int fileLength = 0, counter = 0;
    char *plainText;
    text.open(fileName);

    if(!text)
    {
        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }

    while(!text.eof())
    {
        text.read((char*)&asciiChar, sizeof(char));

        if(!text.eof())
        {
            fileLength++;
        }
    }
}

```

```

text.close();
text.open(fileName);

plainText = AssignCharMemory(fileLength);
while(!text.eof())
{
    text.read((char*)&asciiChar, sizeof(char));
    plainText[counter] = asciiChar;

    if(!text.eof())
    {
        counter++;
    }
}

setPlainText(plainText,fileLength);

cout<<"\nPrinting plainText read from the file: \n";

for(int i = 0; i < fileLength; i++)
{
    cout<<plainText[i]<<" ";
}
//deallocateCharMemory(plainText);
text.close();
}

void FileHandler::OpenKeyFile(char *fileName)
{
    ifstream text;
    char asciiChar;
    int fileLength = 0, counter = 0;
    char *key;
    text.open(fileName);

    if(!text)
    {
        cout<<"\n\nUNABLE TO OPEN FILE!!\n\n";
        exit(1);
    }
}

FileHandler::~FileHandler()
{
}

```

main.cpp

```
#include "main.h"

int main()
{
    Hill_Cipher hc;
    char option;

    cout<<"Hill Cipher Encryption/Decryption Program:"<<endl;
    cout<<"[1]Encrypt\n[2]Decrypt"<<endl;
    cin>>option;

    while(!(option == '1' || option == '2'))
    {
        cout<<"\nIncorrect Option, enter again...\n";
        cin>>option;
    }

    hc.OpenKeyFile("key.txt");

    if(option == '1')
    {
        hc.OpenPlainTextFile("document.txt");
        hc.Encrypt();
        hc.WriteCipherFile();
    }

    else if(option == '2')
    {
        hc.OpenPlainTextFile("document.txt");
        hc.Encrypt();
        hc.WriteCipherFile();
        hc.Decrypt();
        hc.WritePlainTextFile();
    }
}
```