Instituto Politécnico Nacional

Escuela Superior de Cómputo

Session 3

Cryptography

Group: 3CM6

Students:
Nicolás Sayago Abigail
Naranjo Ferrara Guillermo

Teacher:
Díaz Santiago Sandra

February 27, 2019

Contents

1 Cryptanalysis of the Hill Cipher

First, we assume that the opponent has determined the value of **m** being used. Suppose he has at least **m** distinct plaintext-ciphertext pairs, say

$$x_j = (x_{1,j}, x_{2,j}, \ldots, x_{m,j})$$

and

$$y_j = (y_{1,j}, y_{2,j}, \ldots, y_{m,j})$$

for $1 \leq \mathrm{j} \leq m$, such that $y_j = e_K(x_j)$, $1 \leq \mathrm{j} \leq m$. If we define two m x m matrices $\mathbf{X} = (x_{i,j})$ and $\mathbf{Y} = (y_{i,j})$, then we have the matrix equation $\mathbf{X} = \mathbf{YK}$, where the m x m matrix K is the unknown key. Provided that the matrix **X** is invertible, the opponent can compute $\mathbf{K} = \mathbf{X}{-1}\mathbf{Y}$ and thereby break the system. (If **X** is not invertible, then it will be necessary to try other sets of m plaintext-ciphertext pairs.)

If the opponent does not know **m** and assuming that **m** is not too big, he could simply try $m = 2, 3, \ldots,$ until the key is found.

2 Programming Exercises

Design a program to implement the cryptanalysis of Hill cipher, considering the following requirements.

- Assume that to encipher it was used 3 x 3 matrix as a key, which is unknown for you.
- Your program must find the key and store it in a file. Also you must prove that the key is correct, this implies that given a ciphertext encrypted with this key, you must be able to decrypt it.

2.1 Main code

In this subsection, we will explain the main code. We assume that the size of the matrix is 3 x 3. To be able to implement the cryptanalysis of Hill cipher, we need the plaintext and ciphertex. The user give us these text by a file. We expect an entry like the following:

- plainText.in
- cipherText.in

To be able to read the texts from a file, we create a function, which concatenates everything found in the file. The function name is **leerArchivo()**.

We get a valid key, called **K**, with the function **obtencionLlave()**, and finally, we decipher the ciphertext with **K** so that the key is correct.

✓Code

```
int main() {
        char *path, *path2;
        string dir, cip_msj, msj;
        int **K;
        // Read text plain path
        getline(cin, dir);
        path = const_cast<char*>(dir.c_str());
        leerArchivo(path, &msj);

        // Read ciphertext path
        getline(cin, dir);
        path = const_cast<char*>(dir.c_str());
        leerArchivo(path, &cip_msj);

        // Get valid key
        K = obtencionLlave(msj, cip_msj);
        cout << decipher(K, cip_msj);
        return 0;
}
```

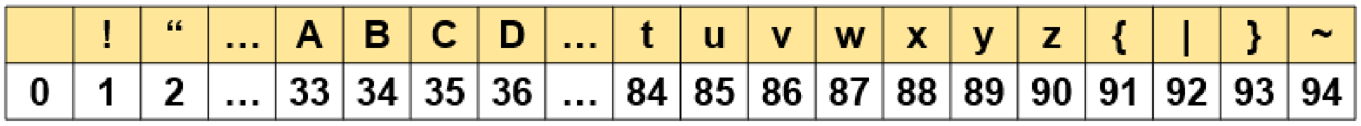
2.2 Get Key

We need 4 matrices:

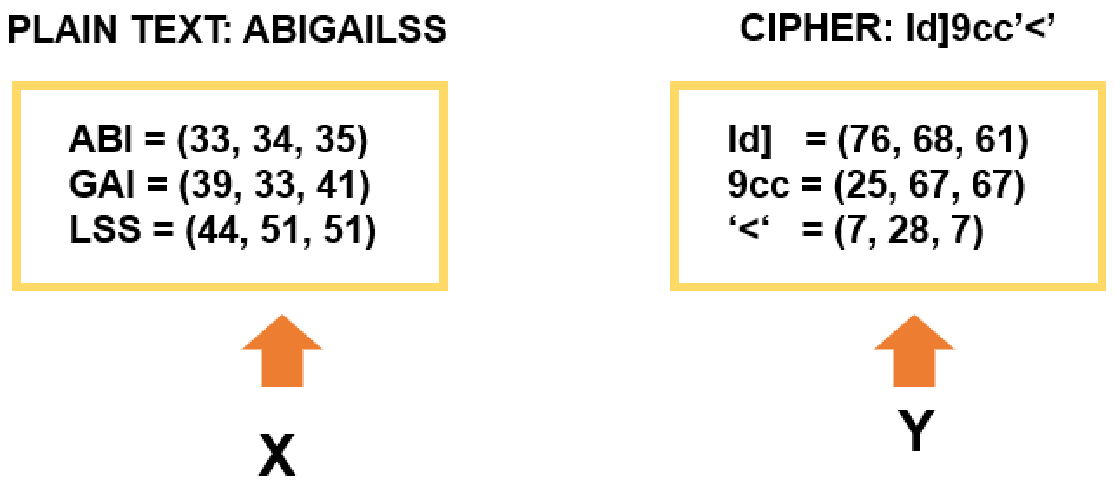
- Matrix **X** : Save the ASCII numbers of the plain text. It is a 3 x 3 matrix.
- Matrix **Y** : Save the ASCII numbers of the cipher text. It is a 3 x 3 matrix.
- Matrix **K**: Save the KEY. It is a 3 x 3 matrix.
- Matrix **Xinv**: Save the inverse matrix of the ASCII numbers of the plain text. It is a 3 x 3 matrix.

First, we have a cycle that is repeated till there's found a key that has inverse working in modulo 95. After that, **X** and **Y** are filled with the first 3 triads of the plaintext and ciphertext respectively.

Each one of the if-else is in charge of verify that the char specified belongs to the printable characters in ascii, in order to avoid the tab. If the char is tab, the next possition in the text is inserted into the matrix.



	!	“	…	A	B	C	D	…	t	u	v	w	x	y	z	{	|	}	~
0	1	2	…	33	34	35	36	…	84	85	86	87	88	89	90	91	92	93	94



We obtain the correct **X**, and we get its inverse.

$$\textbf{inverse}$$

$$\underbrace{\begin{bmatrix} 33, 34, 35 \\ 39, 33, 41 \\ 44, 51, 51 \end{bmatrix}}_{X^{-1}} = \begin{bmatrix} 24, 74, 87 \\ 50, 63, 47 \\ 69, 11, 81 \end{bmatrix}$$

Finally, we make a multiplication to obtain the **K**.

$$K = \underbrace{\begin{bmatrix} 24, 74, 87 \\ 50, 63, 47 \\ 69, 11, 81 \end{bmatrix}}_{X^{-1}} * \underbrace{\begin{bmatrix} 76, 68, 61 \\ 25, 67, 67 \\ 7, 28, 7 \end{bmatrix}}_{Y} = \begin{bmatrix} 8, 1, 1 \\ 4, 7, 0 \\ 6, 2, 3 \end{bmatrix}$$

✓Code

```
int **obtencionLlave(string msj, string cip_msj) {
        int cont3 = 0, ban, cont, cont2, aux=0;

        //Matrix corresponding to the plaintext
        int **X = (int**)malloc(sizeof(int*)*3);
        //Matrix corresponding to the ciphertext
        int **Y = (int**)malloc(sizeof(int*)*3);
        //Matrix corresponding to the key
        int **K = (int**)malloc(sizeof(int*)*3);
        int **Xinv = (int**)malloc(sizeof(int*)*3);

        for(int i = 0; i < 3; i++) {
                X[i] = (int*)malloc(sizeof(int)*3);
                Y[i] = (int*)malloc(sizeof(int)*3);
                K[i] = (int*)malloc(sizeof(int)*3);
                Xinv[i] = (int*)malloc(sizeof(int)*3);
        }

        // Is repeated till there's found a key that has inverse working in
         ↪  modulo 95
        do{
                // Fill X, Y with the first 3 triads of the plaintext and
                 ↪  ciphertext
                for(cont = 0; cont < 3; cont3+=3, cont++) {
                        /* Each one of the if-else is in charge of verify
                        that the char specified belogs to the
                        printable characters in ascii */
                        if(msj[cont3] >= 32 && msj[cont3] <= 126) {
                                X[cont][0] = (int)msj[cont3] - 32;
                                Y[cont][0] = (int)cip_msj[cont3] - 32;
                        }
                        else {
                                cont3++;
                                X[cont][0] = (int)msj[cont3] - 32;
                                Y[cont][0] = (int)cip_msj[cont3] - 32;
                        }

                        if(msj[cont3 + 1] >= 32 && msj[cont3 + 1] <= 126){
                                X[cont][1] = (int)msj[cont3 + 1] - 32;
                                Y[cont][1] = (int)cip_msj[cont3 + 1] - 32;
                        }
                        else {
                                cont3++;
                                X[cont][1] = (int)msj[cont3 + 1] - 32;
                                Y[cont][1] = (int)cip_msj[cont3 + 1] - 32;
                        }

                        if(msj[cont3 + 2] >= 32 && msj[cont3 + 2] <= 126){
                                X[cont][2] = (int)msj[cont3 + 2] - 32;
                                Y[cont][2] = (int)cip_msj[cont3 + 2] - 32;
                        }
                        else {
                                cont3++;
                                X[cont][2] = (int)msj[cont3 + 2] - 32;
                                Y[cont][2] = (int)cip_msj[cont3 + 2] - 32;
                        }
                }
                //The matrix X is inverted
                ban = inverseKey(X, Xinv);
        }while(ban != 0);

        // Multiplication to obtain K
        FILE *f=fopen("C:/Users/Dell/Documents/Crypto/Session3/key.txt",
         ↪  "w");
        for(cont = 0; cont < 3; cont++){
                fprintf(f, "(\t");
                for(cont3 = 0; cont3 < 3; cont3++){
                        for(cont2 = 0; cont2 < 3; cont2++){
                                //Obtains the coeficients Aij of the matrix
                                aux = aux + (Xinv[cont][cont2] *
                                 ↪  Y[cont2][cont3]);
                                if(cont2 == 2){
                                        //Calculate modulo 95 for each
                                         ↪  element in the matrix
                                        if(aux < 0)
                                                aux = 95 - (aux % 95);
                                        else
                                                aux = aux % 95;
                                        K[cont][cont3] = aux;
                                        aux = 0;
                                        fprintf(f, "%d\t", K[cont][cont3]);
                                }
                        }
                }
                fprintf(f, ")\n");
        }
        fclose(f);
        return K;
}
```

2.3 Get Inverse Key

This function is used to get the inverse key so we can decrypt the ciphertext. In essence, it's the same function of the last session, except that, instead of receive the key and return its inverse, now it receives the key, and the matrix of the inverse to fill instantly, to make possible that the function returns an int value.

This value derives from an if-else structure that's in charge of evaluate if the determinant of the K has inverse or not. If it has inverse, we proceed to do the same as the function in the last session returning a value of 0. If K doesn't have inverse we return a value of -1.

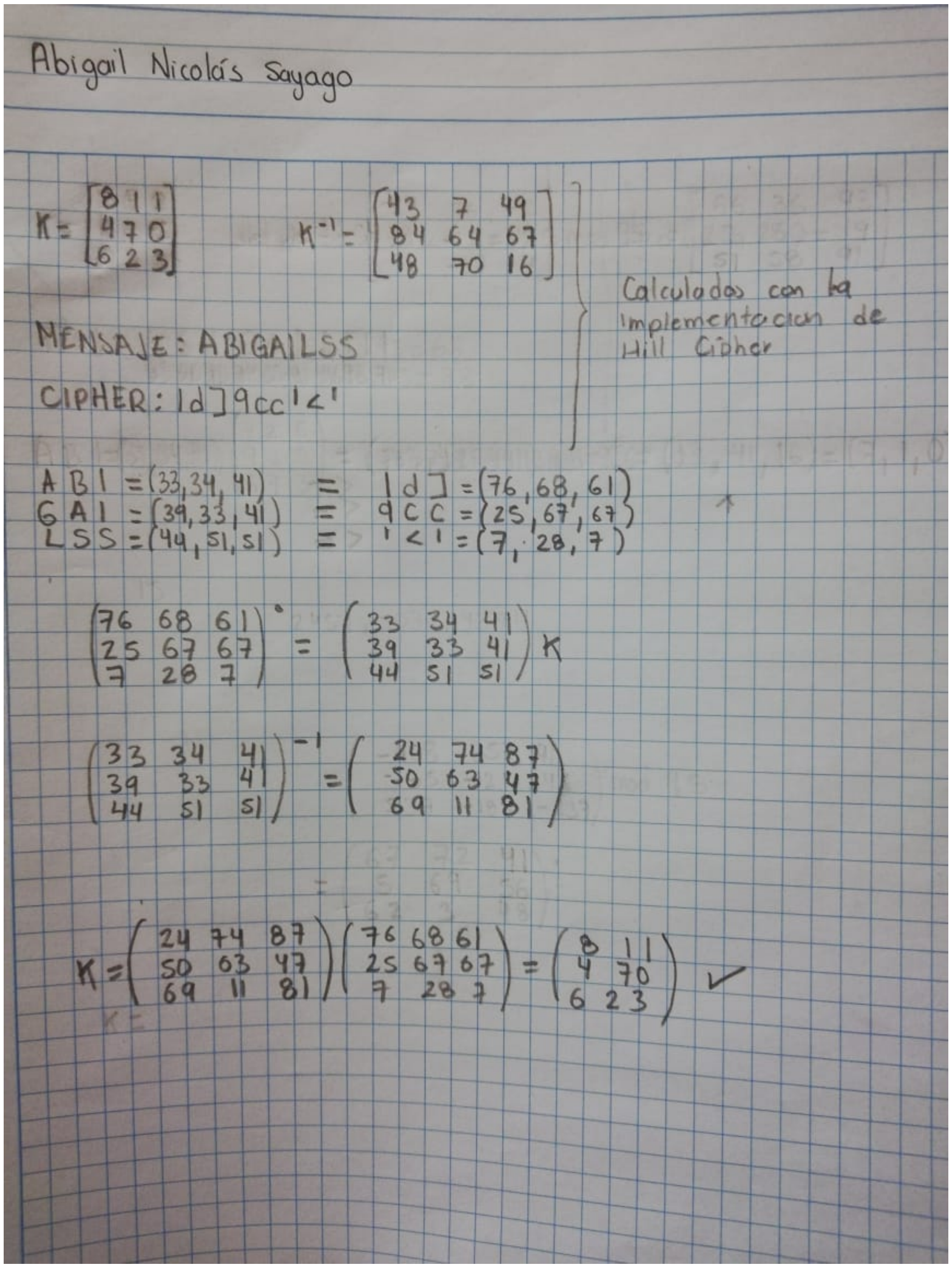
This value is essential for the proof of different triads in the function **obtencionLlave()** till the key is obtained.

✓Code

```
int inverseKey(int **K, int **K_inv) {
        int cofac[4];
        int cont, cont2, cont3, cont4, cont_cofac, det = 0, inv_det = 0,
         ↪  cofactor = 0, n = 95;
        //Obtention of the determinant modulo n
        det = matrixDeterminant(K);
        if(det < 0)
                det = n + (det % n);
        else
                det = det % n;
        //Ontention of the determinant inverse
        inv_det = inverse(n, det);
        //In case that the determinant doesn't have inverse, the function
         ↪  returns -1
        if(inv_det == -1)
                return -1;
        //In case that it has inverse
        else{
                // Getattached matrix
                // Two cycles to traverse all the elements of the matrix
                for(cont = 0; cont < 3; cont++) {
                        for(cont2 = 0; cont2 < 3; cont2++) {
                                // Last two cycles to obtain cofactor of the
                                 ↪  element analyzed
                                cont_cofac = 0;
                                for(cont3 = 0; cont3 < 3; cont3++){
                                        for(cont4 = 0; cont4 < 3; cont4++){
                                                if(cont != cont3 && cont2 !=
                                                 ↪  cont4){
                                                        cofac[cont_cofac] =
                                                         ↪  K[cont3][cont4];
                                                        cont_cofac++;
                                                }
                                        }
                                }
                                // Get cofactor
                                cofactor = cofac[0] * cofac [3] - cofac[1] *
                                 ↪  cofac[2];
                                // Evaluated if i+j is even or odd
                                if ((cont + cont2) % 2 != 0)
                                        cofactor = cofactor * (-1);
                                // Get modulo of the cofactor
```

```
                                        if(cofactor < 0)
                                                cofactor = n + (cofactor % n);
                                        else
                                                cofactor = cofactor % n;
                                        // Cofactor element is multiplied by the
                                         ↪  inverse of the determinan
                                        // The module is obtained
                                        cofactor = (cofactor * inv_det) % n;
                                        /* The matrix of cofactors is transposed to
                                         ↪  obtain attached
                                         (which will be the inverse of the key by
 ↪  the previous operation) */
                                        K_inv[cont2][cont] = cofactor;
                              }
                    }
                    return 0;
          }
}
```

3 Exercise



Abigail Nicolás Sayago

$$K = \begin{bmatrix} 8 & 1 & 1 \\ 4 & 7 & 0 \\ 6 & 2 & 3 \end{bmatrix} \qquad K^{-1} = \begin{bmatrix} 43 & 7 & 49 \\ 84 & 64 & 67 \\ 48 & 70 & 16 \end{bmatrix}$$

MENSAJE: ABIGAILSS

CIPHER: ld]9cc'<'

Calculados con la implementación de Hill Cipher

A B I = (33, 34, 41) = l d] = (76, 68, 61)
G A I = (39, 33, 41) = 9 c c = (25, 67, 67)
L S S = (44, 51, 51) = ' < ' = (7, 28, 7)

$$\begin{pmatrix} 76 & 68 & 61 \\ 25 & 67 & 67 \\ 7 & 28 & 7 \end{pmatrix} = \begin{pmatrix} 33 & 34 & 41 \\ 39 & 33 & 41 \\ 44 & 51 & 51 \end{pmatrix} K$$

$$\begin{pmatrix} 33 & 34 & 41 \\ 39 & 33 & 41 \\ 44 & 51 & 51 \end{pmatrix}^{-1} = \begin{pmatrix} 24 & 74 & 87 \\ 50 & 63 & 47 \\ 69 & 11 & 81 \end{pmatrix}$$

$$K = \begin{pmatrix} 24 & 74 & 87 \\ 50 & 63 & 47 \\ 69 & 11 & 81 \end{pmatrix} \begin{pmatrix} 76 & 68 & 61 \\ 25 & 67 & 67 \\ 7 & 28 & 7 \end{pmatrix} = \begin{pmatrix} 8 & 1 & 1 \\ 4 & 7 & 0 \\ 6 & 2 & 3 \end{pmatrix} \checkmark$$