RUHR-UNIVERSITÄT BOCHUM

Cryptanalysis of Spanish Civil War Ciphers

Jose Luis Benitez Moreno

Chair for Embedded Security – Prof. Dr.-Ing. Christof Paar
Advisor: Christof Paar
The current technological advances allow operations with much computational cost. Thanks to this, it is possible to decipher texts that were not possible at the time. During the Spanish Civil War, an encryption method was used to send coded messages, which at that time was really hard to break by the other band. The purpose of this thesis is try to break this kind of encryption, analyze it weaknesses, and provide a tool which can automatically decrypt an encrypted message which uses the same method of the Spanish Civil War.

Performing a previous frequency analysis on the telegrams and with the use of dictionaries, we will see that we are able to decrypt this kind of messages.
Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

______________________________

Jose Luis Benitez Moreno
# Contents

1 Introduction ...................................................... 1
   1.1 Motivation ......................................................... 1
   1.2 Related Work ....................................................... 1
   1.3 Contribution ....................................................... 2
   1.4 Organization of this Thesis ................................. 2

2 Background ...................................................... 3
   2.1 Definitions ......................................................... 3
   2.2 Caesar’s Cipher ...................................................... 3
   2.3 Monoalphabetic Cipher ............................................. 4
   2.4 Vignere’s Cipher .................................................... 5

3 Spanish Civil War Cipher ........................................... 7
   3.1 Definition ........................................................ 7
   3.2 Homophonic table ............................................... 8
   3.3 Keyword and Displacement ..................................... 9
   3.4 Example ............................................................ 10

4 Cryptanalysis of Monoalphabetic Ciphers ......................... 13
   4.1 Introduction ...................................................... 13
   4.2 Frequency analysis in Monoalphabetic cipher ............... 13
      4.2.1 Example .................................................. 15

5 Cryptanalysis of Spanish Civil War Cipher ....................... 17
   5.1 Previous work .................................................... 17
   5.2 Proposed method ................................................. 17
      5.2.1 Decryption without knowing the tables ............... 18
      5.2.2 Decryption knowing the tables ....................... 20
      5.2.3 Example .................................................. 21

6 Implementation .................................................. 23
   6.1 Previous work .................................................... 23
   6.2 Proposed Method ............................................... 23

7 Results .......................................................... 31
   7.1 Monoalphabetic cipher .......................................... 31
   7.2 Homophonic cipher ............................................... 32
1 Introduction

Nowadays we live in a world where the cryptography plays a very important role in our lives, such as in communication systems. Cryptography is a field in constant development due to the constant searches of system’s vulnerabilities. This searches leads to old encrypted systems which could not be broken in their time. For example during the war time, messages were encrypted using methods that we are now able to break. The services of information present in both sides in a war, give an extreme importance to the encryption. In our case, we deal with the encrypted messages sent during the Spanish Civil War (Spanish Civil War [SCW]).

The origin of this project is due that there are still many ciphertexts of the civil war, which have not been decrypted yet, hiding messages that could be considered invaluable at the historical level.

A method has been proposed, which allows to decrypt ciphertexts of the civil war time. The encryption system is based on homophone table substitution with a displacement of the alphabet. This is called ‘Cifrado por cinta movil’. This project propose tool that allows to use the current advances to decipher encrypted messages with this method. At the very end, the iteration with a person is necessary, due to this method is not perfect, and not every single letter is correctly decrypted.

The encryption method consists on replacing each letter of the text by a two-digit number. A given table provides in each case a correspondence between each letter and several numbers, so that the same letter can be replaced by more than one number, and the cryptanalysis become more complex.

1.1 Motivation

Nowadays, there are still messages of the SCW which have not been decrypted. Since we have sufficient resources to decrypt this messages, the investigation becomes more interesting.

1.2 Related Work

There are some works about the decryption of the SCW cipher. We are based on the project of Jose Miguel Soriano de la Camara ‘Criptoanalisis, mediante algoritmos genéticos, de textos cifrados en la Guerra Civil española con la técnica cinta movil’ [dLC11], which proposes a method based on genetic algorithm. Also we are based on project of Juan Jesus Ruiz ‘Code breaking of SCW communications’ [Rui12]
1.3 Contribution

The contribution of this project is to provide a tool which is able to decrypt certain messages encrypted with this method automatically, and also for other kind of encryptions.

1.4 Organization of this Thesis

First, we have a small introduction to monoalphabetic encrypted, which is the basis of encryption "cinta movil". Then we have a description of the "cinta movil" encryption, followed by an example. These issues are introductory, the following issues are the important part of this thesis: monoalphabetic encryption cryptanalysis and encryption of the SCW cryptanalysis. Finally we provide a tool which can decrypt monoalphabetic encrypted messages and messages encrypted by the "cinta movil" method, under some rules and limitations.
2 Background

2.1 Definitions

2.2 Caesar’s Cipher

The Caesar’s cipher is one of the first known ciphers historically. This encryption method consist in a shift of the latin alphabet the number of position provided by a key. For example, if we want to encode the message: 'ATTACK NOW' with a key of four, we will have to shift each letter four positions to the left in the latin alphabet.

Plain Alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Table 2.1: Encrypted alphabet using Caesar’s Cipher

So the encrypted text is: EXXEGO RSA

To decode this message, we only need to shift four positions to the right side, having the original message.

If an attacker want to decipher a Caesar’s cipher, there exists two possible methods:

- Decode the text with every possible key. This method is an exhaustive search, and it is called "brute force" method, and in it will need in the worst of the cases 26 iterations. An attacker can use brute force to try to decrypt every cipher. The success of this will depend on the keyspace of the cipher. The bigger the keyspace is, the worst success probability the attacker will have.

- Search for the frequency of the letters that appears more in the text. If we know that the message belongs to an english text, we can search for the most frequent letter of the encrypted text and compare to the repetition of characters table of the english language. The most frequent encrypted character should correspond to the most frequent character in the english language. If this assignation is correct, we know the key, and so the clear text.
2.3 Monoalphabetic Cipher

Since a Caesar’s cipher has two main weaknesses, a variant of this cipher was invented to avoid brute force attacks. This is the monoalphabetic cipher, which consist in an assignment of every character to another one located on a ciphered alphabet. Using the previous example, if we want to encrypt the message 'ATTACK NOW' and we have the next assignment:

Plain Alphabet:     A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Encrypted Alphabet: U P K F A V Q L G B W R M H C X S N I D Y T O J E Z

Table 2.2: Encrypted alphabet using Monoalphabetic cipher

Our ciphered message will be: 'UDDUKW HCO' Now, if an attacker wants to recover the original message, he could try to use the two previous methods:

- If the attacker wants to use a brute force analysis, he has to search in a keyspace of $27! \approx 10^{28}$ possible keys. So brute force is now a discarded method to the monoalphabetic cipher.

- Since the assignment of the clear character to the ciphered character is unique, the frequency of the most frequent clear character will be the same as the most frequent ciphered character. So monoalphabetic cipher is weak against frequency analysis.
2.4 Vignere’s Cipher

Until now we have seen a univocal correspondence between the non-ciphered character and the ciphered. This leads to a weakness in the frequency analysis sense. In order to avoid this, the Vignere’s cipher is introduced.[Wiki13g]

Vignere’s cipher picks every possible Caesar’s cipher (the 26 possible keys), and join every ciphered alphabet in a table called Vignere’s table:

![Figure 2.2: Vignere’s Table](image)

To encrypt a message, the sender choose a keyword to cipher, and then letter by letter does the assignment, adding the two values of the letters and performing the modulo operation:

\[(X_i + Z_i)_{mod 26}\]
For example, if an attacker once again wants to encrypt the message 'ATTACK NOW' with the keyword 'KEY':

Message: A T T A C K N O W
Keyword: K E Y K E Y K E Y
Encrypted Message: K D R K G I X S Z

Table 2.3: Encrypted Message using Vignere’s cipher

Since the keyword is chosen free, and the assignation of every character depends on the keyword, the frequency analysis is much harder to perform. However, a possible cryptanalysys can be performed if the attacker guess correctly the length of the key. If this happens, the attacker can break it treating the text as individual Caesar’s ciphers.
3 Spanish Civil War Cipher

3.1 Definition

The encryption method consists in replacing each letter of the text by a two-digit number. In order to do this, we use a table that gives us the correspondence between numbers and letters. This table is called “homophonic table”. The numbers are fixed in a column, and we can have a maximum of 5 numbers for each letter, so that, the same letter can be replaced by more than one number, and the cryptanalysis is more complex.

Alongside this, we have the strip cipher; it consists in usage of a disordered alphabet. It is called 'Cinta móvil'. For communications, both sides must agree in the key and the displacement used in order to have the disordered alphabet An example is shown in the figure[1].

![Figure 3.1: Example of homophonic table.](image)

The columns of numbers are called "homophones". Homophones are known as the multiple representations that we can use to encrypt a character, without following a certain pattern or function (numbers between 00 and 99 in this case). In the figure shown, we can see that there are two different rows with alphabet, one of them is ordered and the other one is disordered. The disordered alphabet is used to encrypt the text. In order to build it we need a keyword and a displacement (strip cipher). These elements are given.

As follows we see how this alphabet is built. We can see the relationship between numbers and letters in the next figure[1]:

![Figure 3.1](image)
To encrypt a plaintext with this method we need three elements:

- **Homophonic table**: Correspondence between characters and numbers.
- **Keyword**: First element used to build the disordered alphabet.
- **Displacement**: Once we have the disordered alphabet, we have to align with the ordered alphabet for correspondence with numbers.

Keyword and Displacement are used at the same time.

### 3.2 Homophonic table

This element is the most important one because with this it’s known which groups of numbers are assigned to a character. An important characteristic of this method is that the difference between the numbers for a character is at least a dozen. For example, for character A, we would have the numbers (08, 19, 30, 45) but not the numbers (15, 17, 30, 45) because 15 and 17 are in the same dozen. In addition, we choose randomly a number of the column for a character. If the character is repeated, every single value from this column can be selected, but we cannot repeat the same number in the next iteration. This is in order to avoid frequency attacks in decoding. How we can see in the next figure:
For example, all the numbers in the first column replace the character shown down, in this case X.

### 3.3 Keyword and Displacement

This element is the main mechanism to get the disordered alphabet. Once we have the key, we have to procedure to eliminate the repeated character, for example: If we have the key "AVIADOR", we have to eliminate the second "A", and then we have "AVIDOR". After that, we have to fill this word with the alphabet (in alphabetical order and with the letters that are not contained in "AVIDOR"). In the next table we can see how we do this:

<table>
<thead>
<tr>
<th>A</th>
<th>V</th>
<th>I</th>
<th>D</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>W</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Filled alphabet

Once it is done, we have to rearrange the new alphabet. In order to do this, we take new alphabet letters according to the length of "AVIDOR":

<table>
<thead>
<tr>
<th>A</th>
<th>V</th>
<th>I</th>
<th>D</th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>W</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Example of method to get disordered alphabet
Now we only have to concatenate each column in order. For instance, we take the first column, later the second one... until the end. After that, we have the disordered alphabet:

```
A B J P X V C K Q Y I E L S Z D F M T O G N U R H N W
```

Table 3.3: Disordered alphabet

Once it is done, the only thing that we have to do is to shift the alphabet to the position designed by the displacement. Normally, the displacement is given like "Clave C en T". It means that the group of numbers assigned to T (in ordered alphabet) corresponds now to C (in disordered alphabet). We have to shift the C character in disordered alphabet to the T character in the ordered alphabet. This displacement is given in the next table:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
S Z D F M T O G N U R H Ñ W A B J P X V C K Q Y I E L
```

Table 3.4: Encrypted alphabet using SCW alphabet

The shaded column correspond to the disordered alphabet, blank rows correspond to the ordered alphabet. Now we have the real correspondence between characters and numbers.

### 3.4 Example

In order to clarify the explanation, we see an example:

- **KEYWORD**: AVIADOR || **DISPLACEMENT**: "Clave C en T"
- **DISORDERED ALPHABET**: "SZDFMTOGNURHNWABJPXVCKQYIEL"
- **HOMOPHONIC TABLE**:
3.4 Example

Figure 3.4: homophonic table used for the example.

- MESSAGE: "DEFENDIDO"
- CIPHERED MESSAGE: "0327 3081 0731 1203 41"

The result is given in pairs of two numbers, as it was the most frequent during the Civil War.
4 Cryptanalysis of Monoalphabetic Ciphers

4.1 Introduction

In order to perform a monoalphabetic encryption, as we have seen in chapter two, every character from cipher alphabet corresponds to a character from plain alphabet. That includes a unique one-to-one relationship. The first method that we use to decrypt an encrypted message with this encryption is a brute force attack. As we read in [Wik13b], this method requires a lot of computational charge.

For example, for an alphabet of 27 letters, we would have to do $27! \approx 10^{28}$ iterations to decrypt the ciphertext, which can be considered impossible. This makes us studying the weaknesses of the different methods of encryption.

As we know, in monoalphabetic ciphers each encrypted character corresponds to one form the plain alphabet. In relatively long messages, each character will have a frequency which corresponds with the frequency of a character from the plain alphabet. So we can know which character corresponds each one analysing the frequencies: Frequency analysis[Wik13e].

4.2 Frequency analysis in Monoalphabetic cipher

The basic idea of frequency analysis is to count the frequency of each character in the ciphertext and associate it with an equal frequency of the plain alphabet. The objective of this project is to propose a method that works to decipher the telegrams emitted in the SCW. Telegrams have a relatively short length, so that, a frequency analysis will not give us a real frequency which encrypts each character. To analyse this, we make use of a plain text (about 500 characters) and we count the frequency of each character. Here it is an example:

*Rápida formación del Ejército Regular. El cumplimiento de esta tarea exige la absorción completa de los restos que aún subsisten de unidades militares espontáneas, milicias sindicales y de partido, en las unidades disciplinadas, orgánicas del ejército único ( ... ) Armamento e instrucción militar general y especial. Férrea disciplina revolucionaria de guerra. ( ... ) g) ( ... ) educar al pueblo en el odio y la intransigencia hasta el exterminio contra el fascismo nacional y extranjero que invade y arrasa con sus armas nuestro país y que traiciona la retaguardia, pero también contra sus agentes.* [Wik12]

If we count the frequency of each character and we compare them with the frequencies of Spanish alphabet in that time:
Here we can see that several characters do not coincide, and then we can not assign directly the decrypted character. What we have to do is to assign a group of possible characters to each encrypted character, these are called candidates. The question now is: How do we assign the candidates? To assign the candidates we are based on binomial and standard normal distributions to each character of the alphabet. This way, every candidate of the encrypted character has a probability of being the corresponding character. For example, supposing that we have the encrypted character ‘x’, the probability that x is actually the character E of the plaintext is (for binomial [Wiki13a]):

\[
P(x = E) = \binom{n}{x} (P_e)^x (1 - P_e)^{n-x}
\]  

(4.1)
4.2 Frequency analysis in Monoalphabetic cipher

Where $P_e$ is the probability of character 'E' ($\text{Freq}_E/100$) of the plain alphabet and $n$ is the number of characters we have. In the right side, $x$ is the number of repetitions of character 'x'. For normal distribution:

$$P(x = E) = \frac{1}{\sqrt{2\pi}\sigma_e} e^{-\frac{(x-\mu_e)^2}{2\sigma_e^2}}$$

(4.2)

Where $\sigma_e$ is the standard deviation for frequency of 'E', we take this as 1, $\mu_e$ is the median for frequency of 'E' and $x$ in the right side, is the frequency of the encrypted character. To get a standard normal distribution we have to remain the frequency of the correspondence character we want to calculate. Once we have done this with all the characters for all encrypted characters, we sort the probabilities from highest to lowest. This way, we have the 'best' candidates at top. If we only take the first 5 positions, we reduce the search space around 81% . After that, in order to get the best combination for candidates, we calculate the joint probability for the 5 most frequent characters in the ciphertext, since taking the first 5, we are deciphering about 50 % of the text without much computational cost. To calculate the joint probability does the following:

$$P = P_{c1}(S_i) \cdot P_{c2}(S_j) \cdot P_{c3}(S_k) \cdot P_{c4}(S_l) \cdot P_{c5}(S_m) \quad \text{for } i, j, k, l, m = 1, ..., 5$$

(4.3)

Where $P_{c,x}(S_j)$ is the probability of encrypted character $x$ to be the character $j$. Having 5 candidates for each encrypted character, we have in total to make $5^5 = 3125$ iterations, where many of them will be zero, because we can not do the product of two probabilities for the same candidate since it is a monoalphabetic cipher. In order to clarify all this, we see an example for two encrypted characters.

### 4.2.1 Example

Suppose we have encrypted characters $w$ (the plain character A) and $x$ (the plain character E) Each with a given frequency:

<table>
<thead>
<tr>
<th>Encrypted character</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>13%</td>
</tr>
<tr>
<td>x</td>
<td>12,4%</td>
</tr>
</tbody>
</table>

Table 4.2: Frequency of encrypted characters

We proceed to calculate the probabilities of each of them with all the letters of the alphabet; we have used binomial distribution for this example. After sort the probabilities and taking the first 5 candidates we have:

\footnote{In this example we are not based on any text.}
### Table 4.3: Probabilities encrypted characters

How we can observe, we obtain the same candidates, this is because the frequencies are near. The list of candidates are (in order to take):

- Encrypted character x: E A O N S
- Encrypted character w: E A O N S

Table 4.4: List of candidates of encrypted characters

Now we have to calculate the joint probability. The only condition we have to keep in mind is that we cannot make the product of probabilities for the same candidate, for example, the probability of being 'E' of character 'w' with the probability of 'E' the character 'x'. It makes no sense. Sorting these products we have (only show 5):

<table>
<thead>
<tr>
<th>Joint probability(%)</th>
<th>Probability 1º (%)</th>
<th>Probability 2º (%)</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.91%</td>
<td>33%</td>
<td>27%</td>
<td>w='A'</td>
</tr>
<tr>
<td>7.35%</td>
<td>35%</td>
<td>21%</td>
<td>w='E'</td>
</tr>
<tr>
<td>4.9%</td>
<td>35%</td>
<td>14%</td>
<td>w='E'</td>
</tr>
<tr>
<td>4.62%</td>
<td>33%</td>
<td>14%</td>
<td>w='A'</td>
</tr>
<tr>
<td>3.96%</td>
<td>33%</td>
<td>12%</td>
<td>w='A'</td>
</tr>
</tbody>
</table>

Table 4.5: Joint probability of encrypted characters

The best option is to take 'w' as 'A' and 'x' as 'E', which is the solution for us. We can extend this for more encrypted characters, exactly 5, getting more than 70% of success with around 210 iterations. To do all this analysis, we have based on Caesar Cipher. It can be seen in appendix B.
5 Cryptanalysis of Spanish Civil War Cipher

5.1 Previous work

To start with the cryptanalysis of the SCW Cipher, we are based on the project of José Miguel Soriano de la Cámara. Which makes a frequency analysis for the decryption of the encryption algorithm. Is based on the knowledge of the boards but not the key. Establishes an average of 4 homophones (numbers) per character, thus based on the idea that the numbers are repeated in a proportional manner, the frequency of each number, multiply by four. To assign a set of possible characters to numbers (candidate) uses a frequency range. That is, assigns the candidates if it is satisfied that:

\[ F_C \cdot (1 - M) \leq F_N \leq F_C \cdot (1 + M) \]  

(5.1)

Where \( F_C \) is the frequency of the character we take, \( F_N \) is the frequency of the number and \( M \) is the margen of frequency.

For example, if we have the number 20 with a frequency of 11% and we have de frequency of character \( C_i = 10, 1\% \) and using a margen of 20%:

\[ 10.1\% \cdot (0.8) \leq 11\% \leq 10.1\% \cdot (1.2) \]  

(5.2)

\[ 8.08 \leq 11\% \leq 12.12\% \]  

(5.3)

Here we can observe that the frequency of the number fulfil the condition, then, it is possible to assign the correspondence character. This method is based on establishing a range of frequencies, we want to try to optimize it. Not based on a range of frequencies, but on probabilities, we determine which are the best candidates for each homophone.

5.2 Proposed method

For the cryptanalysis of the telegrams of the SCW, we are based on statistical analysis. That is, we count how often each number is repeated and then can assign it to a character. As we have seen in the tables in Chapter 3, groups of numbers are assigned to a character. To calculate the frequency of a character, is enough for us to calculate the frequency of that group of numbers. But for that we need to have the tables. We analyze this encryption in two ways:

- Decryption without knowing the tables: It is difficult to obtain a frequency of every number near his character.
- Decryption knowing the tables: Encryption is like a monoalphabetic.
5.2.1 Decryption without knowing the tables

The main problem is that the frequency of each character is divided into several numbers (usually 3, 4 or 5) so that it is more difficult to calculate a real rate which would be a character. Also each number assigned to a character is not repeated in proportion, ie, if for example the letter 'E' is assigned to the numbers '2,32,57,70', 70 may be repeated once, and 57 4 times, so that the frequency is masked. An example of this:

<table>
<thead>
<tr>
<th></th>
<th>E (13%)</th>
<th>2 (5%)</th>
<th>20 (3%)</th>
<th>57 (4%)</th>
<th>70 (%1)</th>
<th>Total: 13%</th>
</tr>
</thead>
</table>

Table 5.1: Example of frequency for character E

How we can see, each number repeats randomly.

As we know, each character is usually divided into 3, 4 or 5 numbers, according to the tables we have. If we calculate how many times each character is divided into 3, 4 or 5, we get an average of 3.7. This means that the frequency of each number of the ciphertext, must be multiplied by 3.7 for a value close to the frequency of the character. We propose a mathematical programming model that implements this. We see it in the next chapter.

The length of telegrams of SCW are short, then the frequency analysis is a bit more complicated. In a short text, the character most repeated, can not match the most repeated character Spanish alphabet. With a large amount of ciphertexts is possible to find some strings of numbers or symbols that are repeated and therefore can form digrams, trigrams and the best words and phrases.

Due to the problem of disturbance frequency, we can not know with certainty the character corresponding to each number, which is necessary to assign a set of characters to each number, these character sets are called candidates. The idea is to assign the most efficient way possible, to do this we are based on probabilities, to tell us the probability of a character corresponds to a number. To calculate the probabilities, we make use of binomial and normal distributions, these follow a similar pattern. These distributions are based on the frequency of the letters in spanish alphabet, each character follows a binomial/normal distribution. The frequencies of Spanish alphabet used in this section are based on texts of that time, we have used to count several texts from Wik12, a graph is attached in appendix B:
For example, we take the number 2, which has a frequency (already with 3.7 factor) of 12.3% . To calculate the probability which this number is the character 'E', where character 'E' has a frequency of 13% \(^1\)

\[
P(x = E) = \binom{n}{x} (P_e)^x (1 - P_e)^{n-x}
\]  

(5.4)

Where \(n\) is the total number of characters, \(x\) is the number of repetitions of the number 2 and \(P_e\) is the probability of character 'E' (13% divided to 100). Once this is done for all the characters of the alphabet, we have all the probabilities. Only we have to sort

\(^1\)this is for binomial, for normal, see previous chapter.
these probabilities then we have first the best candidate (the best probability), later the second... If we only take the first five, we reduce the search space in 81.48%. With the first five candidates, we are taking the five nearer to the frequency of the number. For the number 2, we have the candidates "EAORS".

Once we have calculated all probabilities for all the numbers, we proceed to calculate the joint probability, this means, we calculate the product of probabilities of different numbers:

\[ P(x_A \sim y_B) = P(x = A)P(y = B) \]  

\( (5.5) \)

In this algorithm, we are based on only take the most frequent numbers to do the joint probability. When we have all the joint probabilities, we take the best values and then we have the best chances to decipher. The problem for the algorithm without the tables is that we have to take the 20 most frequent numbers to get almost 5 characters, because a character is encrypted with 3, 4, or 5 numbers. So it takes a lot of memory usage, to get an idea, if we calculate the joint probability of the 20 most frequent numbers, would have to make the order of \( 5^{20} \) multiplications (each number has five candidates with five different probabilities). So in this project we decide to decipher only the SCW cipher method considering that the fixed table columns are known. That is, we know what numbers go together but not which letters they correspond to.

### 5.2.2 Decryption knowing the tables

This method is based on everything explained in the previous section, the difference from the above is that we know what numbers go together and correspond to the same character. For simplicity, from now, the set of numbers that corresponds to the same character, we will call quartet. For frequency analysis, we do not have to multiply by a factor, we calculate the frequency by adding the frequencies of numbers of the quartet, this gives us the frequency of the encrypted character:

<table>
<thead>
<tr>
<th>Character E</th>
<th>2</th>
<th>20</th>
<th>57</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>13%</td>
<td>2%</td>
<td>4%</td>
<td>3%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 5.3: Example 2 of frequency for character E

Once we have all the frequencies, we calculate the probability with binomial and normal distributions (in next chapters we show the results for each one). After that, we can calculate the joint probability. For this case, this requires less memory usage because we do not have to take the 20 most frequent numbers, only the 5 most frequent quartets. With this, we only have to do around \( 5^5 = 3125 \) multiplications, each quartet has 5 candidates with 5 different probabilities sorted from higher to lower. When we have all the probabilities, only we have to sort them and take the best options. Here is given an example to clarify all this:

\( ^2 \)Because the 5 most frequent characters, take around 50% of text
5.2 Proposed method

5.2.3 Example

<table>
<thead>
<tr>
<th>Quartet1</th>
<th>Probability 1°</th>
<th>Probability 2°</th>
<th>Probability 3°</th>
<th>Probability 4°</th>
<th>Probability 5°</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.13 (E)</td>
<td>0.11 (A)</td>
<td>0.09 (O)</td>
<td>0.08 (R)</td>
<td>0.05 (S)</td>
</tr>
<tr>
<td>Quartet2</td>
<td>0.12 (A)</td>
<td>0.11 (E)</td>
<td>0.09 (R)</td>
<td>0.07 (S)</td>
<td>0.04 (O)</td>
</tr>
<tr>
<td>Quartet3</td>
<td>0.15 (I)</td>
<td>0.13 (R)</td>
<td>0.11 (S)</td>
<td>0.10 (N)</td>
<td>0.03 (L)</td>
</tr>
<tr>
<td>Quartet4</td>
<td>0.14 (O)</td>
<td>0.12 (A)</td>
<td>0.11 (L)</td>
<td>0.08 (R)</td>
<td>0.07 (S)</td>
</tr>
<tr>
<td>Quartet5</td>
<td>0.13 (S)</td>
<td>0.11 (N)</td>
<td>0.10 (L)</td>
<td>0.09 (D)</td>
<td>0.07 (P)</td>
</tr>
</tbody>
</table>

Table 5.4: Probabilities of the 5 most frequent quartets.

Here we have the table of the 5 best probabilities for each quartet, they are sorted highest to lowest, also we can see the candidate that corresponds for each probability. After this, we multiply all the probabilities. It means:

\[
\prod_{j=1}^{j=5} P_{ij} \quad \text{for } i = 1...5
\]

When we have calculated all the joint probabilities, we have to sort highest to lowest (so we have the best candidates in top positions). For the first position we have the first column of candidates shown above, this is logical, in addition, we do not consider the product where the candidates are the same, for example, the fist candidate of quartet1 with the second candidate of quartet2, this is an impossible combination, we take it like 0 probability, after this, here it is shown the main joint probabilities:

<table>
<thead>
<tr>
<th>Option</th>
<th>Joint Probability</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1° Option</td>
<td>4.25 \cdot 10^{-5}</td>
<td>[E,A,I,O,S]</td>
</tr>
<tr>
<td>2° Option</td>
<td>3.69 \cdot 10^{-5}</td>
<td>[E,A,O,R,S]</td>
</tr>
<tr>
<td>3° Option</td>
<td>3.60 \cdot 10^{-5}</td>
<td>[E,A,I,O,N]</td>
</tr>
<tr>
<td>4° Option</td>
<td>3.30 \cdot 10^{-5}</td>
<td>[A,E,I,O,S]</td>
</tr>
<tr>
<td>5° Option</td>
<td>2.86 \cdot 10^{-5}</td>
<td>[A,E,R,O,S]</td>
</tr>
</tbody>
</table>

Table 5.5: Joint probability

Where candidates means the character for each quartet. For example, in first option, we have the candidate "E" for the fist quartet, "A" for the second, "I" for the third, "O" for the fourth and "S" for the fifth. We now have 3125 possibilities to decrypt, but if we only take the best options, on the order of 500, for 300 characters in the text, we get about 70. If we get success for the 5 most frequent quartets, we have around 50% of text decrypted, how we have seen in the previous chapter. You can try for more characters, the only drawback is that you increase the number of iterations in powers of 5 (if we take the top 5 candidates).
6 Implementation

Following the statistical background and the description of the encryption methods, we want now to implement a tool to be able to break monoalphabetic and homophonic ciphers. We assume that a frequency analysis has been performed previously.

6.1 Previous work

In order to start developing a tool which provides us a readable text, we should have 5 characters previously decrypted. This can be done by the implementation of a frequency analysis and try to find out which letters corresponds to the 5 more frequent characters, which in case of the Spanish language are: 'A','E','O','R' and 'S'.

Once we have a text with this characters decrypted and the others encrypted, we can now work with this text. We have had in consideration two cases:

- Monoalphabetic encryption
- Homophonic encryption

In both of the cases, we will work with a text with the non-decrypted letters in the uppercase, and the decrypted letters in the downcase. Therefore, if we want to work with a homophonic encryption, we should do a preprocessing of the text. Furthermore, we will need to know the table used to encrypt the plaintext, and then assign the pairs of numbers belonging to the table to their characters in the table.

6.2 Proposed Method

Once we have our text prepared, we will have an unique relationship between an encrypted character and the decrypted character. In principle, every character has the possibility of being any other character. What we are going to do is to compare every word, depending on the length, with a dictionary of the known words of the Spanish language which are stored for their lengths: two, three, upon seven.

Every time that a word fits with one of the dictionary, we increase the number of times that a letter has a success being replaced by other letter and belongs to a word of the dictionary. Let’s see an example in order to clarify this:

If we had the plaintext: "La mesa", coded as: "Sa Jesa", we begin picking the encrypted word "Sa". Then we take a look with every possible word of length two, which ends in "a", and we will find two matches: "La" and "Fa". So now the probability of the
encrypted character 'S' being 'L' or 'F' increases one unit. The same with the "Jesa", which matches for example with "Mesa", "Pesa", or "Besa".

At the end of analyzing the whole text, we sort for every encrypted character the number of successes, having in the top the most likely character. Finally, when we are going to decrypt the text with this tool, we reach a very readable text, but not perfect. This is due to the appearance of success of the same letter in words of different lengths.

An analyst can easily correct this false positive, and break finally the entire encryption. Now, a flowchart of the tool is introduced:

![Figure 6.1: Tool flowchart](image)

In order to select the word, we read until we find an empty space between two coded
words. Then we call the dictionary functions.

Now we are comment the code belonging to every block of the flowchart and the main variables and structures that take place into the program:

The structures that are used are ‘QUARTETS’ and ‘CHANGE’. The first one is used only if the entry text is an homophonic ciphertext. We store the homophone table given with this structure and then replace the numbers for the cipher letters according to the table. The second one is used to count the number of successes of a cipher letter being another one, as described above.

```c
typedef struct quartets{
    int num[5];
    float freq;
    float prob[L];
    char ci;
}QUARTETS;

typedef struct change{
    char ci;
    char sol;
    char candidates[27];
    int success[27];
}CHANGE;
```

Listing 6.1: Structures

The preprocess function take place only if the number of arguments is bigger than two. This function store the homophic table and assign every number that find in the ciphertext to the character that corresponds in the homophonic table.

We have found several tables and keys that were used for the encryption in the book "Soldados sin Rostro". For example, we have selected a key that was commonly used at that time: ABANDO.

![Homophone table used with key ABANDO](image)

Figure 6.2: Homophone table used with key ABANDO

We have transcribed these tables to txt documents that continue the following format:
Then, knowing the format of this table, we are able to perform the assignment of number to character with the preprocess function showed below:

```c
void preprocess(int fh, int fo){
    fn = open("preprocesado.txt",O_CREAT|O_TRUNC|O_RDWR,0777);

    while(read(fh,&rd,sizeof(unsigned char))!=0){
        //We store the desired table
        if(rd>='A' && rd<='Z' || rd=='*') {
            if(rd=='A') {
                strncpy(&quartet[i].ci,&rd,1);
                read(fh,&rd,sizeof(char));
                i=0;
            }
            else {
                k=0;
                i++;
                strncpy(&quartet[i].ci,&rd,1);
                read(fh,&rd,sizeof(char));
            }
        }
        else {
            dec = atoi(&rd)*10;
            read(fh,&rd,sizeof(char));
            ind = dec + atoi(&rd);
            quartet[i].num[k]=ind;
            k++;
        }
    }

    while(read(fo,&rd,sizeof(char))!=0){ //COUNTING THE NUMBERS TWO BY TWO AND STORE
        if(rd >=48 && rd<=57){
            dec = atoi(&rd)*10;
            read(fh,&rd,sizeof(char));
            ind = dec + atoi(&rd);
            quartet[i].num[k]=ind;
            k++;
        }
    }

    while(read(fh,&rd,sizeof(char))!=0){ //COUNTING THE NUMBERS TWO BY TWO AND STORE
        if(rd >=48 && rd<=57){
            dec = atoi(&rd)*10;
            read(fh,&rd,sizeof(char));
            ind = dec + atoi(&rd);
            quartet[i].num[k]=ind;
            k++;
        }
    }
}
```

Figure 6.3: Example of txt document for table
6.2 Proposed Method

```
ind = dec + atoi(&rd); //We read two caracters, convert them into numbers and associate to
their real value
for(i=0;i<27;i++){
    while(quartet[i].num[length]!=0){
        length++;
    }
    for(j=0;j<length;j++){
        if(ind==quartet[i].num[j])
            write(fn,&quartet[i].ci,sizeof(char));
    }
    length=0;
}
else{
    write(fn,&rd,sizeof(char));
}
}
```

Listing 6.2: Preprocess function

After initialize the structures and preprocess the text (if necessary), we read every word of the ciphertext and select a dictionary in function of its length.

```
while(read(fo, &rd, sizeof(char))!=0){
    if(rd!=' '){
        longitud++; 
        cadena[i]=rd;
        i++;
    }else{
        switch (longitud) {
            case 2:
                dicci2(cadena,change);
                break;
            case 3:
                dicci3(cadena,change);
                break;
            case 4:
                dicci4(cadena,change);
                break;
            case 5:
                dicci5(cadena,change);
                break;
            case 6:
                dicci6(cadena,change);
                break;
            case 7:
                dicci7(cadena,change);
                break;
            default:
                ;
        }
        longitud=0;
        for(i=0;i<10;i++)
            cadena[i] = '\0';
        i=0;
    }
```
Then, we continue to compare the word with the dictionary. Here we must take into account that the word has two cases: an encrypted letter, or a plain letter. So the condition of a readed word of the text matching with a word of the dictionary, is that if we find a plain letter, they have to match in the same position. If this is fulfill, then we analyze the whole word, search when a match happens, and increase the number of successes in the structure. Here one dictionary function is showed:

```c
void dicci2(char* cadena, CHANGE* changes){
    //cad = chain readed from the text
    //dic = chain of the dictionary to be compared
    for(i=0;i<2;i++){
        cad[i] = *(cadena+i);
    }
    for(i=0;i<2;i++)
        dic[i]=\0;
    cad[2]=\0;
    fd = open("dicci2.txt",O_RDONLY);
    i=0;
    o=0;
    while(fin!=0){
        for(i=0;i<2;i++){
            if ((fin == 0) break;
            if (rd != ' ' ){
                dic[i] = rd;
                i=-1;
            }else{
                i=-1;
            }
        }dic[2]=\0;
        if(fin!=0){
            match=1;
            for(i=0;(cad[i]!=\0)&&match==1);i++){
                if (((cad[i]=='a')&&(cad[i]=='e')&&(cad[i]=='o')&&(cad[i]=='r')&&(cad[i]=='s')){
                    aux2 = dic[i];
                    aux1 = cad[i];
                }
            if((cad[i]=='a')||(cad[i]=='e')||(cad[i]=='o')||(cad[i]=='r')||(cad[i]=='s')){
                if(cad[i]==(dic[i]+32)){
                    match=1;
                }else{
                    match=0;
                }
            }else if (((cad[i]=='a')&&(cad[i]=='e')&&(cad[i]=='o')&&(cad[i]=='r')&&(cad[i]=='s'))&&((
                match=1;
            }
        }
    }
}
```
When there is no more words to be counted, we proceed to sort the candidates for every cipher letter:

```c
void exchange_position(CHANGE* changes, int i, int j, int pos)
{
    int tmp = changes[pos].success[i];
    char tmp2;
    strncpy(&tmp2, &changes[pos].candidates[i], 1);
    changes[pos].success[i] = changes[pos].success[j];
    strncpy(&changes[pos].candidates[i], &changes[pos].candidates[j], 1);
    changes[pos].success[j] = tmp;
    strncpy(&changes[pos].candidates[j], &tmp2, 1);
}
```

```c
void selection_sort(CHANGE* changes, int N, int pos)
{
    int i, j, k;
```
for( i=0;i<N;i++)
    for(k=i,j=i+1;j<N;j++)
        if(changes[pos].success[j]>changes[pos].success[k])
            k=j;
        if(k!= i)
            exchangeposition(changes,i,k,pos);
}

Listing 6.5: Sort function

Finally we do the substitution of the characters from the ciphertext, using the first value contained in the candidates structure:

while (read(fo,&rd,sizeof(char))!=0) {
    if(((rd!='a')&&(rd!='e')&&(rd!='o')&&(rd!='r')&&(rd!='s') )&& ((rd>='A') && (rd<= 'Z'))){
        for(i=0;i<27;i++){
            if(rd == change[i].ci)
                ci = change[i].candidates[0]; //The first candidate
        }
        write(fexit,&ci,sizeof(char));
    }
    else{
        write(fexit,&rd,sizeof(char));
    }
}

Listing 6.6: Substitution

The results will be discussed in the next chapter
7 Results

In this section we discuss the results that have been given by the implementation.

7.1 Monoalphabetic cipher

In the case of a monoalphabetic cipher, with the most frequent letters decrypted, we see a simple example:

Plaintext: "EN EL MUNDO EDITORIAL, CUANDO UNO ALCANZA EL EXITO CON UN LIBRO, LO INTELIGENTE ES ESCRIBIR ALGO EN LA MISMA LINEA UNA VEZ AL A*O DURANTE EL RESTO DE LA VIDA. LOS PAYASOS NO DEBERIAN TRATAR DE INTERPRETAR EL PAPEL DE HAMLET Y LAS ESTRELLAS DEL POP NO DEBERIAN COMPOSER SINFONIAS."

Ciphertext: "eR eP QYRhO eHMXorMaP, GYaRHo YRo aPGaRDa eP eBMXo GoR YR PMFVo, Po MRxePMKeRXe eW esGrHFMr aPKo eR Pa QMsQa PMRea YRa ZeD aP a*o HraERXe eP VesXo He Pa ZMHa. PoS TACasos Ro HeFeRMaR XVaXar He MRXeTReXar eP TaTeP He LaQPex C Pas esXrEpPaS HeP ToT Ro HeFeRMaR GoQToRer WMJoRMas."

We can notice that the downcase correspond to the correct decipher of the letters 'a','e','o','r','s', which have been obtained performing a previous frequency analysis.

When we obtain the exit after running the tool implemented, we see a very similar text, which an analyst who knows the language could easily decrypt.

eN eL CUNIo eIIoRiAL, CUAiNiO UNo aLCaNiA eL eTIo CoN UN LIcLo , Lo INLeLITeNLe eH esCrICiR aLo eN La CIsCa LiNeea UNa VeI aL a*o IraNNLe eL LesLo iE La VIIa. Los BaAasos No IeCeRiaN LLaLaR iE INleBReLar eL BaBeL iE CaCleL A Las esLreLLaS iEl BoB No IeCeRiAN CoCBoNeR HINaOHiAS.

We see how the characters 'U','N','L','C','I', etc, are usually decrypted correctly, becoming a readable text.
7.2 Homophonic cipher

Doing the same as the previous section, but preprocessing the text, we see an example:

Plaintext: "EN EL MUNDO EDITORIAL, CUANDO UNO ALCANZA EL EXITO
CON UN LIBRO, LO INTELIGENTE ES ESCRIBIR ALGO EN LA MISMA LINEA
UNA VEZ AL AÑO DURANTE EL RESTO DE LA VIDA. LOS PAYASOS NO
DEBERIAN TRATAR DE INTERPRETAR EL PAPEL DE HAMLET Y LAS
ESTRELLAS DEL POP NO DEBERIAN COMPOSER SINFONIAS."

Ciphertext: "e46 e65 84289675o e758266or313a65, 935a9624o
7971o a4068a4632a e90 e295791o 43o71 282o 65821072o, 40o
312041e655753e9615e es es939757103172 a6578o e20 40a
8431e59a 908271ea 7946a 62e32 a65 a22o 2428ra2066r e14
47e6e6o 75e 90a 378224a. 90os 87a18asos 46o 50e10er31a71
91ra15ar 75e 824691er8715ar e14 87a23e40 75e 42a8465e91
44 65as es41re4090as 50e14 23o23 96o 24e10er31a46
93o8487o71er s824689o9682as"

Preprocessed text: "eA e* BQAJo eJXPorXU*, TQAJo QAo a*TaAZa e*
 eSXPo ToA QA *XCo, *o XAPe*XOSeAPe es esTCXXC a*Oo eA *a BXsBa
 *XAea QAa IeZ a* aEo JQraAPr e* CesPo Je *a IXJa. *os HaWasos Ao
 JeGerXaA PraPar Je XAPerHrePar e* HaHe* Je RaB*eP V *as esPra**as
 Je* HoH Ao JeGerXaA ToBHoAer sXAMoAXas"

Taking into account that we have used the following table:

A20467196
B335984
C21477297
D346085
E22487398
F356186
G10367499
H234987
I113762
J245075
K12386388
L255176
M13396489
N265277
*14406590
Finally, if we run the tool over the preprocessed text, we obtain the next result:

```
eN e* CUNIo eIIlorIA*, CUaNio UNo a*CaNIa e* eTIlo CoN UN *ICMo, 
*o INLe*ITTeNLLe es esCMICIM a*Tol eN *a CIsCa *INea UNa VeI a* aDo
IUraNLr e* MesLo Ie *a VIIa. *os BaAasos No IeCerIaN LraLar Ie
INLerBreLar e* BaBe* Ie CaC*eL A *as esLre**as Ie* BoB No IeCerIaN
CoCBoNer sINaoNIas
```

Which once again, is a very readable text, which an analyst who knows the language can easily read and decrypt.
8 Conclusion

8.1 Summary

According to the results seen in the last chapter, if we have a telegram of the SCW based in homophones, where the 5 most frequent letters are decrypted (for example, performing a frequency analysis), we have seen that we can easily have the plaintext. The only drawback is that we need to know which table was used to encrypt this telegram. Furthermore, the presence of an analyst is needed, because the mechanism is not perfect.

The decoding method is based on a dictionary analysis and the length of the words. But how we saw in the example of chapter 3, the ciphertext usually does not provide this length of the words. As we said, this method requires the homophone table used to encrypt, so the strength of this method is improved as many tables we have.

8.2 Future Work

As future work, we propose to improve this method by different ways. We have seen that when we have counted the successes of the cipher letters for their candidates, we only pick the first one in the list. This can be improved looking at the combination of the candidates and the frecuency of appearence in the language to have better results.

It is also proposed as future work to investigate how can we use this method without knowing the length of the words of the ciphertext, due to usually we do not know this data.

Finally, different versions of this tool could be developed for different languages, using other dictionaries.
A Acronyms

SCW Spanish Civil War
B Code

B.1 Tool

- Main Code

```c
int main (int argc, char** argv)
{
    int fo,fh,aux;
    char rd,ci;
    char cadena[10];
    int longitud=0;
    int fexit,i,j;
    char A = 'A';
    char B = 'A';
    CHANGE change[27];

    fo=open(argv[1],O_RDONLY); //Open the entry text
    fexit = open("exit.txt",O_CREAT|O_TRUNC|O_RDWR,0777); //Create the exit

    if(argc < 2){
        printf("Usage: ./tool text.txt [homophonic table.txt]"�);
        exit(0);
    } else {
        fh = open(argv[2],O_RDONLY);
        preprocesa(fh,fo);
        close(fo);
        fo = open("preprocesado.txt",O_RDONLY);
    }

    printf("Welcome. Please, press any key to continue
");
    scanf("%d",&aux);
    printf("The tool is working now. This may take a few seconds...

");

    /*********INITIALIZATION**************/
    for(i=0;i<10;i++)
        cadena[i] = '\0';

    for(i=0;i<27;i++){
        if(i=14){
            change[i].ci = A;
        } else {
            change[i].ci='*';
            A--;
        }
    }

    for(j=0;j<27;j++)
        change[i].success[j]=0;
    if(j!=14){
```
change[i].candidates[j] = B;
} else {
    change[i].candidates[j] = '*';
    B--;
} } 
B++;
}
B = 'A';
A++;
}
i = 0;

/******** WORD SELECTION ********/
while(read(fo, &rd, sizeof(char)) != 0) {
    if(rd != ' '){
        longitud++;
        cadena[i] = rd;
        i++;
    } else {
        switch (longitud) {
        case 2: 
            dicci2(cadena, change);
            break;
        case 3:
            dicci3(cadena, change);
            break;
        case 4:
            dicci4(cadena, change);
            break;
        case 5:
            dicci5(cadena, change);
            break;
        case 6:
            dicci6(cadena, change);
            break;
        case 7:
            dicci7(cadena, change);
            break;
        default:
            ;
        }
        longitud = 0;
        for(i = 0; i < 10; i++)
            cadena[i] = '\0';
        i = 0;
    }
}
printf("Generating the text...
");
sleep(1);
/*****SORT OF THE CANDIDATES******/

for(i=0;i<27;i++)
selection_sort(change,27,i);

for(i=0;i<27;i++)
strncpy(&change[i].sol,&change[i].candidates[0],1);

lseek(fo,0,SEEK_SET);
i=0;

/*******SUBSTITUTION*******/

while (read(fo,&rd,sizeof(char))!=0) {

if(((rd!='a')&&(rd!='e')&&(rd!='o')&&(rd!='r')&&(rd!='s') )&& ((rd>='A') && (rd<= 'Z')))
{
for(i=0;i<27;i++)

if(rd == change[i].ci)
    ci = change[i].candidates[0]; //The first candidate

    write(fexit,&ci,sizeof(char));

}
else{
    write(fexit,&rd,sizeof(char));
}

printf("The exit was successfully generated.\n");
printf("Exiting...\n");
sleep(1);

return 0;

}

Listing B.1: Main program of the decoding tool

- Functions used by the main program of the tool

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
define L 27
```
void dicci2(char* cadena, CHANGE* changes) {
  int fd, i, j, k, m, o;
  char dic[2];
  char cad[2];
  char rd, aux1, aux2;
  int fin = 1;
  int match = 1;
  for (i = 0; i < 2; i++) {
    dic[i] = * (cadena + i);
  }
  for (i = 0; i < 2; i++)
    dic[2] = '\0';
  fd = open("dicci2.txt", O_RDONLY);
  i = 0;
  o = 0;
  while (fin != 0) {
    for (i = 0; i < 2; i++) {
      fin = read(fd, &rd, sizeof(char));
      if (fin == 0)
        break;
      if (rd != ' ')
        dic[i] = rd;
      else
        i = -1;
    }
    dic[2] = '\0';
    if (fin != 0) {
      match = 1;
      for (i = 0; (cad[i] != '\0') && (match == 1); i++) {
        if ((cad[i] == 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's')) {
          aux2 = dic[i];
          aux1 = cad[i];
        }
        if ((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's')) {
          if (aux1 == (dic[i] + 32)) {
            match = 1;
          } else {
            match = 0;
          }
        } else if (((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's')) &&
                   ((dic[i] == 'A') && (dic[i] == 'E') && (dic[i] == 'O') && (dic[i] == 'R') && (dic[i] == 'S'))) {
            match = 1;
        } else if (((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's'))
                   && ((dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') || (dic[i] == 'S')))
            match = 0;
      }
    } else if (match == 1) {
      ...
    } else {
      ...
    }
  }
}
for(m=0;m<2;m++){
    aux2 = dic[m];
    aux1 = cad[m];
    if((aux2 != 'A') && (aux2 != 'E') && (aux2 != 'O') && (aux2 != 'R') && (aux2 != 'S')){
        for(j=0;j<27;j++){
            if(aux1 == changes[j].ci){
                for(k=0;k<27;k++){
                    if(aux2 == changes[j].candidates[k]){
                        changes[j].success[k]++;
                    }
                }
            }
        }
    }
    for(i=0;i<2;i++)
        dic[i]='\0';
}
void dicci3(char* cadena, CHANGE* changes){
    int fd,i,j,k,m;
    char dic[3];
    char cad[3];
    char rd,aux1,aux2;
    int fin=1;
    int match=1;
    for(i=0;i<3;i++){
        cad[i] = *(cadena+i);
    }
    for(i=0;i<3;i++)
        dic[i]='\0';
    fd = open("dicci3.txt",O_RDONLY);
    i=0;
    while(fin!=0){
        for(i=0;i<3;i++)
            rd = dic[i];
    }
}
fin = read(fd, &rd, sizeof(char));

if (fin == 0)
    break;

if (rd != ' ')
    dic[i] = rd;

else{
    i = -1;
}
}
dic[3] = '\0';

if (fin != 0) {
    match = 1;

    for (i = 0; (cad[i] != '\0') && (match == 1); i++) {
        aux2 = dic[i];
        aux1 = cad[i];

        if ((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's')) {
            if ((cad[i] == 'A') || (dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') || (dic[i] == 'S')) {
                match = 1;
            }
        } else if (((cad[i] != 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's')) && (dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') || (dic[i] == 'S')) {
            match = 0;
        } else if (((cad[i] != 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's')) && (dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') || (dic[i] == 'S')) {
            match = 1;
        }
    }

    if (match == 1) {
        for (m = 0; m < 3; m++) {
            aux2 = dic[m];
            aux1 = cad[m];

            if ((aux2 != 'A') && (aux2 != 'E') && (aux2 != 'O') && (aux2 != 'R') && (aux2 != 'S')) {
                for (j = 0; j < 27; j++) {
                    if ((aux1) == changes[j].ci) {
                        for (k = 0; k < 27; k++) {
                            if (aux2 == changes[j].candidates[k]) {
                                changes[j].success[k]++;
                            }
                        }
                    }
                }
            }
        }
    }
    }

}
for(i=0; i<3; i++)
dic[i] = '\0';
}
}
}
}
for(i=0; i<4; i++)
dic[i] = '\0';
}
void dicci4(char* cadena, CHANGE* changes){
int fd, i, j, k, m;
char dic[4];
char cad[4];
char rd, aux1, aux2;
int fin=1;
int match=1;
for(i=0; i<4; i++){
cad[i] = *(cadena+i);
}
for(i=0; i<4; i++)
dic[i] = '\0';
cad[4] = '\0';
fd = open("dicci4.txt", O_RDONLY);
i=0;
while(fin!=0){
for(i=0; i<4; i++){
  fin = read(fd, &rd, sizeof(char));
  if (fin == 0)
    break;
  if (rd != ' ')
    dic[i] = rd;
}
else{
  i=-1;
}
}dic[4] = '\0';
if(fin!=0){
  match=1;
  for(i=0; (cad[i] != '\0') && (match==1); i++)
    if((cad[i] == 'a') && (cad[i+1] == 'e') && (cad[i+2] == 'o') && (cad[i+3] == 'r') && (cad[i+4] == 's')){
      aux2 = dic[i];
      aux1 = cad[i];

  }else{
    i=-1;
  }
}
if((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's')) {
  if(cad[i] == (dic[i] + 32)) {
    match = 1;
  } else {
    match = 0;
  }
}

} else if (((cad[i] != 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's'))
  match = 1;
}

} else if (((cad[i] != 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's'))
  && ((dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') || (dic[i] == 'S'))) {
  match = 0;
}

} else if ((match == 1) {

  for (m = 0; m < 4; m++) {
    aux2 = dic[m];
    aux1 = cad[m];

    if((aux2 != 'A') && (aux2 != 'E') && (aux2 != 'O') && (aux2 != 'R') && (aux2 != 'S')) {
      for (j = 0; j < 27; j++) {
        if((aux1) == changes[j].ci) {
          for (k = 0; k < 27; k++) {
            if(aux2 == changes[j].candidates[k]) {
              changes[j].success[k]++;
            }
          }
        }
      }
    }
  }

  for (i = 0; i < 4; i++) {
    dic[i] = '\0';
  }
}

close(fd);
```c
void dicci5(char* cadena, CHANGE* changes){
    int fd, i, j, k, m;
    char dic[5];
    char cad[5];
    char rd, aux1, aux2;
    int fin=1;
    int match=1;
    for(i=0; i<5; i++){
        cad[i] = *(cadena+i);
    }
    for(i=0; i<5; i++)
    dic[i] = '\0';
    for(i=0; i<5; i++)
    cad[5] = '\0';
    fd = open("dicci5.txt", O_RDONLY);
    i=0;
    while(fin!=0){
        for(i=0; i<5; i++){
            fin = read(fd, &rd, sizeof(char));
            if (fin == 0)
                break;
            if (rd != ' ')
                dic[i] = rd;
            else
                i=-1;
        }
        dic[5] = '\0';
        if(fin!=0){
            if (fin==0)
                break;
            if (rd != ' ')
                dic[i] = rd;
            else
                i=-1;
        }
    }
    if (fin!=0){
        match=1;
        for(i=0; (cad[i] != '\0') && (match==1); i++){
            if (((cad[i] == 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's'))
                || ((cad[i] == 'A') && (cad[i] != 'E') && (cad[i] != 'O') && (cad[i] != 'R') && (cad[i] != 'S')))
                aux2 = dic[i];
            else
                if (((cad[i] != 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's'))
                    || ((cad[i] == 'A') && (cad[i] != 'E') && (cad[i] != 'O') && (cad[i] != 'R') && (cad[i] != 'S')))
                    aux1 = cad[i];
            else
                if (((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's'))
                    || ((cad[i] == 'A') || (cad[i] == 'E') || (cad[i] == 'O') || (cad[i] == 'R') || (cad[i] == 'S')))
                    if (cad[i] == (dic[i]+32))
                        match=1;
                    else
                        match=0;
            }
        }
    }
    else if (((cad[i] == 'a') && (cad[i] != 'e') && (cad[i] != 'o') && (cad[i] != 'r') && (cad[i] != 's'))
            || ((cad[i] == 'A') && (cad[i] != 'E') && (cad[i] != 'O') && (cad[i] != 'R') && (cad[i] != 'S'))
        match=1;
    }
```
else if (((cad[i]!="a")&&((cad[i]!="e")&&((cad[i]!="o")&&((cad[i]!="r")&&((cad[i]!="s")
                     )&&((dic[i] == 'A') || (dic[i] == 'E') || (dic[i] == 'O') || (dic[i] == 'R') ||
                     (dic[i] == 'S')))#)
            }

if(match == 1){
    for(m=0;m<5;m++){
        aux2 = dic[m];
        aux1 = cad[m];
        if((aux2 != 'A') && (aux2 != 'E') && (aux2 != 'O') && (aux2 != 'R') && (aux2 != 'S')){
            for(j=0;j<27;j++){
                if((aux1) == changes[j].ci){
                    for(k=0;k<27;k++){
                        if(aux2 == changes[j].candidates[k]){#
                            changes[j].success[k]++;
                        }
                    }
                }
            }
        }
    }
    for(i=0;i<5;i++)
        dic[i]=\"0\";
}
}
}
}
void dicci6(char* cadena, CHANGE* changes){
    int fd,i,j,k,m ;
    char dic[6];
    char cad[6];
    char rd,aux1,aux2;
    int fin=1;
    int match=1;
    for(i=0;i<6;i++){#
        cad[i] = *(cadena+i);
    }
    for(i=0;i<6;i++)
        ...
dic[i] = '\0';
cad[i] = '\0';
fd = open("dicci6.txt",O_RDONLY);
i=0;
while(fin!=0){
  for(i=0;i<6;i++){
    fin = read(fd,&rd,sizeof(char));
    if (fin == 0)
      break;
    if (rd != ' ')
      dic[i] = rd;
    else
      i=-1;
  }
}
cad[6] = '\0';
if(fin!=0){
  match=1;
  for(i=0;i<6;i++){
    if((cad[i] == 'a') && (ic[i] == 'e') && (cad[i] == 'o') && (cad[i] == 'r') && (cad[i] == 's')){
      aux2 = dic[i];
      aux1 = cad[i];
    }
    if((cad[i] == 'a') || (cad[i] == 'e') || (cad[i] == 'o') || (cad[i] == 'r') || (cad[i] == 's')){
      match=0;
    }
  }
  if(match == 1){
    for(m=0;m<6;m++){
      aux2 = dic[m];
      aux1 = cad[m];
      //printf("%c %c",aux2,aux1);
if((aux2 != 'A') && (aux2 != 'E') && (aux2 != 'O') && (aux2 != 'R') && (aux2 != 'S')){
    for(j=0;j<27;j++) {
        if((aux1) == changes[j].ci){
            for(k=0;k<27;k++) {
                if(aux2 == changes[j].candidates[k]){  
                    changes[j].success[k]++;
                }
            }
        }
    }
    for(i=0;i<6;i++)
        dic[i] = \0;
}
close(fd);
}

void dicci7(char* cadena, CHANGE* changes){
    int fd,i,j,k,m;
    char dic[7];
    char cad[7];
    char rd,aux1,aux2;
    int fin=1;
    int match=1;
    for(i=0;i<7;i++) {
        cad[i] = *(cadena+i);
    }
    for(i=0;i<7;i++)
        dic[i] = \0;

cad[7] = \0;
    fd = open("dicci7.txt",O_RDONLY);
    i=0;
    while(fin!=0){
        for(i=0;i<7;i++){
            fin = read(fd, &rd, sizeof(char));
            if (fin == 0)
                break;
            if (rd ! = \0){
                dic[i] = rd;
            }
        }
    }
    for(i=0;i<7;i++)
        dic[i] = \0;
    close(fd);
}
```c
B.1 Tool

```
void exchangeposicion(CHANGE* changes, int i, int j, int pos){
  int tmp = changes[pos].success[i];
  char tmp2;
  strncpy(&tmp2,&changes[pos].candidates[i],1);
  changes[pos].success[i] = changes[pos].success[j];
  strncpy(&changes[pos].candidates[i],&changes[pos].candidates[j],1);
  changes[pos].success[j] = tmp;
  strncpy(&changes[pos].candidates[j],&tmp2,1);
}

void selection_sort(CHANGE* changes, int N, int pos){
  int i,j,k;
  for( i=0;i<N;i++){
    for(k=i,j=i+1;j<N;j++)
      if(changes[pos].success[j]>changes[pos].success[k])
        k=j;
    if(k!= i)
      exchangeposicion(changes,i,k,pos);
  }
}

void preprocesa(int fh,int fo){
  int dec,ind,k,i,j,length,fn;
  char rd;
  QUARTETS quartet[27];
  k=0;
  i=0;
  j=0;
  fn = open("preprocesado.txt",O_CREAT|O_TRUNC|O_RDWR,0777);
  while(read(fh,&rd,sizeof(unsigned char))!=0){
    //We store the desired table
    if((rd>='A' && rd<='Z') || rd=='*')
      if(rd=='A'){
        strncpy(&quartet[i].ci,&rd,1);
        read(fh,&rd,sizeof(char));
        i=0;
      }else{
        k=0;
      }
    else
      k=0;
  }
}
Listing B.2: Functions used by the main program of the tool

- Structures

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```c
#include <unistd.h>
#include <string.h>
#define L 27

typedef struct quartets{
    int num[5];
    float freq;
    float prob[L];
    char ci;
} QUARTETS;

typedef struct change{
    char ci;
    char sol;
    char candidates[L];
    int success[L];
} CHANGE;
```

Listing B.3: Structures
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Frequency of the characters for english alphabet [Wiki3d]</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Vignere’s Table</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of homophonic table</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Relationship between plaintext-ciphertext</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>homophonic table used in Civil war</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>homophonic table used for the example</td>
<td>11</td>
</tr>
<tr>
<td>6.1</td>
<td>Tool flowchart</td>
<td>24</td>
</tr>
<tr>
<td>6.2</td>
<td>Homophone table used with key ABANDO [FL08]</td>
<td>25</td>
</tr>
<tr>
<td>6.3</td>
<td>Example of txt document for table</td>
<td>26</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Encrypted alphabet using Caesar’s Cipher . . . . . . . . . . . . . . . . . . . 3
2.2 Encrypted alphabet using Monoalphabetic cipher . . . . . . . . . . . . . 4
2.3 Encrypted Message using Vignere’s cipher . . . . . . . . . . . . . . . . . . 6

3.1 Filled alphabet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.2 Example of method to get disordered alphabet . . . . . . . . . . . . . . 9
3.3 Disordered alphabet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
3.4 Encrypted alphabet using SCW alphabet . . . . . . . . . . . . . . . . . . 10

4.1 Comparison between Frequences of Spanish alphabet and text . . . . . . . 14
4.2 Frequency of encrypted characters . . . . . . . . . . . . . . . . . . . . . . 15
4.3 Probabilities encrypted characters . . . . . . . . . . . . . . . . . . . . . . 16
4.4 List of candidates of encrypted characters . . . . . . . . . . . . . . . . . . 16
4.5 Joint probability of encrypted characters . . . . . . . . . . . . . . . . . . 16

5.1 Example of frequency for character E . . . . . . . . . . . . . . . . . . . . . 18
5.2 Frequency of Spanish alphabet . . . . . . . . . . . . . . . . . . . . . . . . 19
5.3 Example 2 of frequency for character E . . . . . . . . . . . . . . . . . . . 20
5.4 Probabilities of the 5 most frequent quartets . . . . . . . . . . . . . . . . . 21
5.5 Joint probability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
# List of Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1  Structures</td>
<td>25</td>
</tr>
<tr>
<td>6.2  Preprocess function</td>
<td>26</td>
</tr>
<tr>
<td>6.3  Selection of the word and the dictionary to be used</td>
<td>27</td>
</tr>
<tr>
<td>6.4  Example of a dictionary function. Case of length 2</td>
<td>28</td>
</tr>
<tr>
<td>6.5  Sort function</td>
<td>29</td>
</tr>
<tr>
<td>6.6  Substitution</td>
<td>30</td>
</tr>
<tr>
<td>B.1  Main program of the decoding tool</td>
<td>39</td>
</tr>
<tr>
<td>B.2  Functions used by the main program of the tool</td>
<td>41</td>
</tr>
<tr>
<td>B.3  Structures</td>
<td>53</td>
</tr>
</tbody>
</table>
Bibliography


