SAPIENZA
UNIVERSITÀ DI ROMA

An Automatic Cryptanalysis Tool for Classical Ciphers

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di laurea in Informatica

Candidato
Luca Campese
1617747

Responsabile
Alessandro Mei

Corresponsabile
Thomas Erlebach

A/A 2015/2016

This page is intentionally left blank.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptography is the study of secret writing. There are many methods to encrypt data, with varying degree of security, which allow one to transform readable information into apparent nonsense.

Cryptanalysis concerns the breaking of these ciphers by exploiting the mathematical or statistical patterns of the encrypted message.

The aim of this project is to produce a stand-alone Java application which automatically recovers an encrypted message as accurately as possible, without user intervention and without knowing the cipher used for the encryption.

The ciphers which have been considered are the following classical ciphers:

- Caesar cipher
- Simple substitution cipher
- Vigenère cipher
- Hill cipher

I will illustrate how the above-mentioned ciphers operate in the next chapter. Then, the discussion will point to the cryptanalysis techniques used to break them.

After that, design and implementation details will be provided, followed by a concluding chapter which will summarise what has been done and what could be improved.

Chapter 2

Encryption methods

Before describing the inner workings of each encryption method used, it should be noted that classical ciphers operate on an alphabet of letters, whereas modern ciphers operate on bits and bytes.

The 26 capital letters of the English alphabet will be used across the examples. However, any finite set of symbols can be chosen instead.

Every character will thus be represented by a number, so that $A = 0$, $B = 1$, $\ldots$, $Y = 24$, $Z = 25$.

2.1 Caesar cipher

2.1.1 Description

The **Caesar cipher** is one of the oldest and simplest encryption methods. It is named after Julius Caesar who used it to exchange private information.

The key of this cipher is a number k between 0 and 25, which determines the shift to apply to each letter in the plaintext. To encrypt a message, each letter is in fact replaced by another letter k positions down the alphabet. Decryption is identical to encryption, but a shift of $-k$ is used instead.

This process can be easily described using modular arithmetic as follows:

$$E_k(x) = x + k \pmod{26}$$
$$D_k(x) = x - k \pmod{26}$$

2.1.2 Example

Using a shift of 3, each letter of the alphabet is mapped to another letter as follows:

```
Plaintext alphabet:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext alphabet: DEFGHIJKLMNOPQRSTUVWXYZABC
```

When encrypting, the sender replaces each letter x in the plaintext with the corresponding letter to which x is mapped.

```
Plaintext:  VENIVIDIVICI
Ciphertext: YHQLYLGLYLFL
```

When decrypting, the receiver applies the shift backwards. For example, the first letter of ciphertext is decrypted as follows:

$$\begin{aligned} D_3(24) &= 24 - 3 \pmod{26} \\ D_3(24) &= 21 \pmod{26} \end{aligned}$$

In fact, $V \mapsto 21$. The same process is repeated to decrypt the remaining ciphertext.

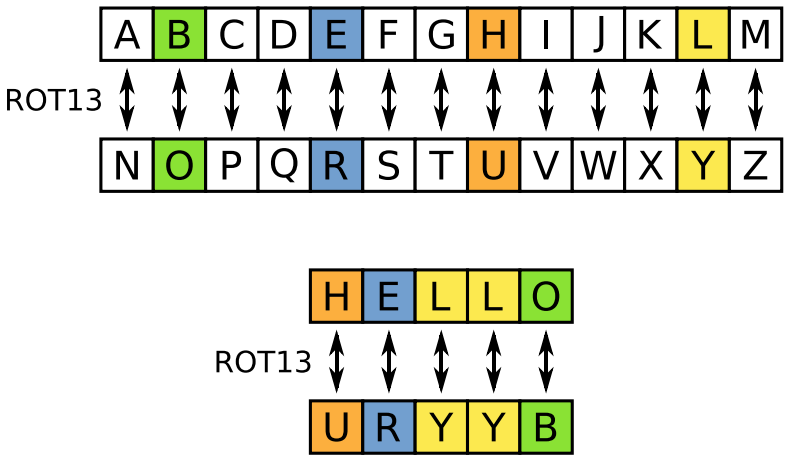

Figure 2.1: ROT13 is a Caesar cipher with a shift of 13. [1]

2.2 Simple substitution cipher

2.2.1 Description

The **simple substitution cipher** is, not surprisingly, a substitution cipher. The same cipher alphabet is used both to encrypt and decrypt a message, and defines to which letter every letter in the plaintext is mapped.

Contrary to the Caesar cipher, the cipher alphabet is not simply the result of shifting the alphabet. Indeed, every letter of the alphabet can be arbitrarily mapped to another one.

An easy way to create a cipher alphabet consists of first writing out a keyword, deleting repeated characters in it, and then writing all the remaining letters of the alphabet in the usual order. [2]

2.2.2 Example

If a sender and a receiver agreed upon the keyword *zebras*, they would use the following alphabet: [2]

```
Plaintext alphabet:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext alphabet: ZEBRASCDFGHIJKLMNOPQTUVWXY
```

Using the keyword above:

```
Plaintext:  ALLISDISCOVEREDFLEEATONCE
Ciphertext: ZIIFPRFPBLUAOARSIAAZQLKBA
```

2.3 Vigenère cipher

2.3.1 Description

The **Vigenère cipher** is a polyalphabetic substitution cipher, in which a set of monoalphabetic ciphers is used. A keyword determines which of these ciphers is to be applied to a particular position.

This not only means that different plaintext letters are shifted by different amounts, but also that the same plaintext letter can be enciphered to different ciphertext letters.

For this reason, the Vigenère cipher was believed to be unbreakable for three centuries, earning the name of *le chiffre indéchiffrable* (French for *the indecipherable cipher*). [3]

As in the case with the Caesar cipher, the encryption and decryption algorithms can be described using modular arithmetic.

$$C_i = E_K(M_i) = M_i + K_i \pmod{26}$$
$$M_i = D_K(C_i) = C_i - K_i \pmod{26}$$

where M_i is the i-th letter in the plaintext, C_i is the i-th letter in the ciphertext and K_i is the i-th letter of the keyword. If necessary, the keyword can be repeated until it becomes as long as the plaintext.

2.3.2 Example

Suppose that the keyword chosen is *lemon*. Then, a message is enciphered as follows:

```
Plaintext:  DEFENDTHECASTLE
Keyword:    LEMONLEMONLEMON
Ciphertext: OIRSAOXTSPLWFZR
```

It should be noted how the first E enciphers to I, whereas the second E enciphers to S.

A *Vigenère square* — shown in Figure 2.2 — can be used both to encipher and decipher a message, especially when these tasks are to be performed by hand.

2.4 Hill cipher

2.4.1 Description

The **Hill cipher** is a polygraphic substitution cipher which operates on groups of letters (two or more) in order to produce corresponding units of ciphertext.

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

Figure 2.2: The Vigenère square can be used to ease both the encryption and the decryption process. [4]

This is achieved using matrices and matrix multiplication, therefore a basic knowledge of matrices and number theory is required to fully understand this cipher.

Firstly, the message to be encrypted is split into blocks of n letters. If the length of the message is not a multiple of n, the message has to be padded with some extra letters. Thus, the letters in each block, represented as numbers, form an n-component vector which is multiplied by an invertible $n \times n$ matrix (modulo 26). The result of this operation, namely an n-component vector, is a unit of ciphertext.

Decryption differs from encryption in that each block of ciphertext is multiplied by the inverse of the matrix used to encrypt the message. Hence, if the key matrix chosen is not invertible modulo 26, then the message cannot be decrypted.

2.4.2 Example

The following example [5] shows how encryption and decryption are performed using a 2×2 key matrix.

Let

$$K = \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}$$

be the key matrix and suppose that the message to be encrypted is *HELP*. The latter is represented as follows:

$$HELP \rightarrow \begin{pmatrix} H \\ E \end{pmatrix}, \begin{pmatrix} L \\ P \end{pmatrix} \rightarrow \begin{pmatrix} 7 \\ 4 \end{pmatrix}, \begin{pmatrix} 11 \\ 15 \end{pmatrix}$$

To encrypt, we compute:

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 7 \\ 4 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 8 \end{pmatrix} \pmod{26}$$

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 11 \\ 15 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 19 \end{pmatrix} \pmod{26}$$

Hence:

$$\begin{pmatrix} 7 \\ 8 \end{pmatrix}, \begin{pmatrix} 0 \\ 19 \end{pmatrix} \rightarrow \begin{pmatrix} H \\ I \end{pmatrix}, \begin{pmatrix} A \\ T \end{pmatrix} \rightarrow HIAT$$

To decrypt, we first need to compute the inverse of K by using the formula:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = (ad - bc)^{-1} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{2.1}$$

where $(ad - bc)^{-1}$ is the multiplicative inverse of the determinant of K in $\mathbb{Z}_{26}$.

Therefore:

$$K^{-1} \equiv 9^{-1} \begin{pmatrix} 5 & 23 \\ 24 & 3 \end{pmatrix} \equiv 3 \begin{pmatrix} 5 & 23 \\ 24 & 3 \end{pmatrix} \equiv \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \pmod{26}$$

Now, we multiply K^{-1} by each of the two vectors obtained through the encryption process.

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 4 \end{pmatrix} \pmod{26}$$

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 19 \end{pmatrix} \equiv \begin{pmatrix} 11 \\ 15 \end{pmatrix} \pmod{26}$$

The result is the original message. In fact:

$$\begin{pmatrix} 7 \\ 4 \end{pmatrix}, \begin{pmatrix} 11 \\ 15 \end{pmatrix} \to \begin{pmatrix} H \\ E \end{pmatrix}, \begin{pmatrix} L \\ P \end{pmatrix} \to HELP$$

Chapter 3

Cryptanalysis methods

In this chapter, the methods used to break the aforementioned ciphers will be described, along with the statistical tools employed to perform the cryptanalysis.

3.1 Cryptanalysis of the Caesar cipher

A simple yet effective method to break the Caesar cipher is to try and decrypt the ciphertext using every possible key. Since the number of keys is small, it is easy to recognise which shift has been used by comparing all the decryption results.

This approach is known as **brute-force attack**, or **exhaustive key search**, and it is feasible in practise only when the number of values to be tested is reasonably small.

This attack, per se, is not particularly interesting, but leads to a fascinating problem which heavily depends on the type of data an encrypted message carries: how does one know when a decryption attempt has been successful?

Generally, encryption is used to conceal meaningful information. If the content of an encrypted message was just random data, then a ciphertext-only attack would be impossible.

Our goal is to find an automated solution which detects when a meaningful message has been found. In our case, a meaningful message is one containing English words.

This can be achieved exploiting some statistical patterns of the English language.

3.2 Chi-squared statistic

The frequency of letters in text has been studied by cryptanalysts for a long time. [6] In fact, in every language there are some letters appearing more frequently than others as well as some letters appearing just rarely.

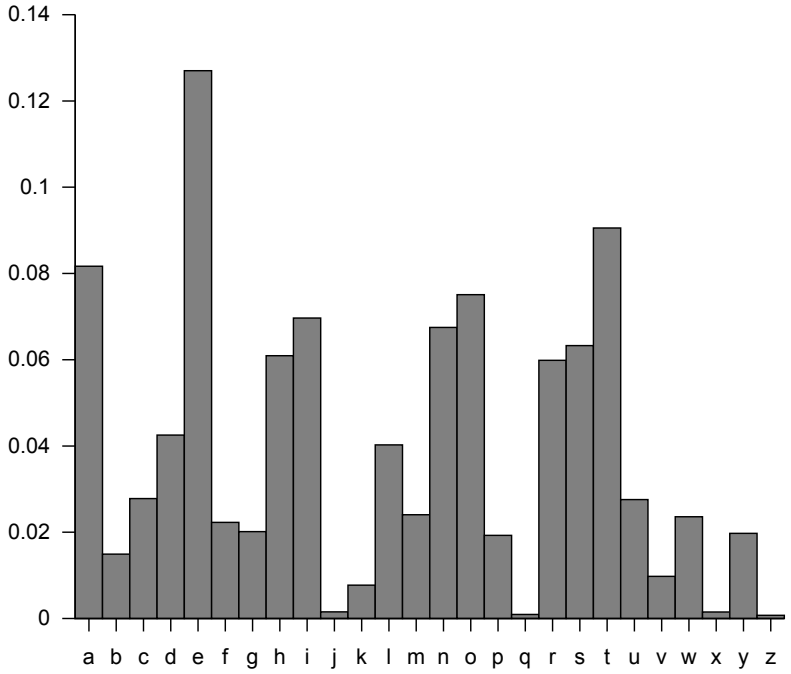

Figure 3.1: Frequencies of letters in the English language. [7]

The graph above shows some distinctive features of the English language: E and T are considerably more common than other letters, whereas Q and Z are the least frequent.

For this reason, if a message encrypted with a Caesar cipher has F and U as the most common letters, it would be a good guess to assume that a shift of 1 has been used.

More generally, a cryptanalyst would calculate the frequency distribution of the ciphertext letters, graph it, and then shift it until it matches up with the

expected distribution of those letters in the English language. This method is called **frequency analysis**.

The **chi-squared statistic** is a way for a computer to perform the procedure described above. It is calculated using the following formula: [8]

$$\chi^2(C, E) = \sum_{i=A}^{Z} \frac{(C_i - E_i)^2}{E_i}$$

where C_i is the count of the i-th letter and E_i is the expected count of the i-th letter.

The chi-squared statistic tells us how similar two categorical probability distributions are. This is expressed by a number, which is the result of the formula above: the closer it is to 0, the more similar the two distributions are.

The chi-squared statistic will be used in the following example to rank all the possible decryption keys of a Caesar cipher, and will suggest which one is most likely to have been used.

3.2.1 Solving a Caesar cipher

Before going through an example, it should be pointed out that, given a text of length n, we can estimate how many times each letter of the alphabet is expected to occur in that text. In fact, if p_i is the probability of choosing the i-th letter picking a character from any text at random, then one expects it to appear roughly $p_i \times n$ times in a text of length n.

This is important because the chi-squared statistic uses counts of letters, not their probabilities of occurring in the text.

In this example, taken from [8], we will try to recover the encrypted message:

```
AOLJHLZHYJPWOLYPZVULVMAOLLHYSPLZARUVDUHUKZPTWSLZAJPWOLY
ZPAPZHAFWLVMZBIZAPABAPVUJPWOLYPUDOPJOLHJOSLAALYPUAOLWSH
PUALEAPZZOPMALKHJLYAHPUUBTILYVMWSHJLZKVDUAOLHSWOHILA
```

Following the approach described previously, the chi-squared statistic will be calculated for each possible decryption. The lowest value thus obtained will correspond to the original message.

Shift	Plaintext	Chi-squared
0	AOLJHLZHYJPWOLYPZVULVMAOL...	1634.09
1	ZNKIGKYGXIOVNKXOYUTKULZNK...	3441.13
2	YMJHFJXFWHNUMJWNXTSJTKYMJ...	2973.71
3	XLIGEIWEVGMTLIVMWSRISJXLI...	1551.67
4	WKHFDHVDUFLSKHULVRQHRIWKH...	1199.40
5	VJGECGUCTEKRJGTKUQPGQHVJG...	1466.62
6	UIFDBFTBSDJQIFSJTPOFPGUIF...	1782.26
7	THECAESARCIPHERISONEOFTHE...	33.67
8	SGDBZDRZQBHOGDQHRNMDNESGD...	1747.07
9	RFCAYCQYPAGNFCPGQMLCMDRFC...	1386.62
10	QEBZXBPXOZFMEBOFPLKBLCQEB...	3423.96
11	PDAYWAOWNYELDANEOKJAKBPDA...	809.38
12	OCZXVZNVMXDKCZMDNJIZJAOCZ...	4646.96
13	NBYWUYMULWCJBYLCMIHYIZNBY...	724.11
14	MAXVTXLTKVBIAXKBLHGXHYMAX...	2159.43
15	LZWUSWKSJUAHZWJAKGFWGXLZW...	1787.26
16	KYVTRVJRITZGYVIZJFEVFWKYV...	3527.17
17	JXUSQUIQHSYFXUHYIEDUEVJXU...	2967.66
18	IWTRPTHPGRXEWTGXHDCTDUIWT...	1368.70
19	HVSQOSGOFQWDVSFWGCBSCTHVS...	929.17
20	GURPNRFNEPVCUREVFBARBSGUR...	461.19
21	FTQOMQEMDOUBTQDUEAZQARFTQ...	4395.68
22	ESPNLPDLCNTASPCTDZYPZQESP...	703.43
23	DROMKOCKBMSZROBSCYXOYPDRO...	1226.79
24	CQNLJNBJALRYQNARBXWNXOCQN...	1817.85
25	BPMKIMAIZKQXPMZQAWVMWNBPM...	2939.16

Table 3.1: This table shows the values of the chi-squared statistic calculated for every possible decryption.

The lowest value is obtained using a shift of 7. The suggested decryption is indeed the only one containing a meaningful message.

3.3 Cryptanalysis of the simple substitution cipher

The simple substitution cipher is more difficult to break compared to the Caesar cipher. The reason is that the number of keys, being 26!, is significantly higher, hence an exhaustive key search is infeasible. [9]

Nevertheless, the frequencies of the letters in the ciphertext are not hidden. In fact, given the nature of this cipher, the same plaintext letter always encrypts to the same other ciphertext letter. This allows one to guess a good key just using frequency analysis. However, decryption will not be accurate in most cases, especially when the ciphertext available is not long enough.

In order to overcome this issue, we will iteratively refine our key following an approach known as *random-restart hill climbing*.

Before discussing the algorithm itself, we first need a way to rate the goodness of a key, so that we can distinguish between a good key and a bad key.

3.4 Quadgram statistics

As said earlier, we need a way to tell whether a piece of text is likely to be English or just gibberish. It is desirable for such a method to be fast, since it will be used to rate a large number of decryption results. Counting quadgrams satisfies both requirements.

A **quadgram** is simply a block of four letters. For example, the word `ORANGE` contains the quadgrams `ORAN`, `RANG` and `ANGE`.

Some quadgrams appear more frequently than others. For instance, `YOUR` is much more commonly seen than `QKPC`. For this reason, multiplying the quadgram probabilities is an efficient means to determine whether any given text is likely to be English or not, and this can be done in $O(n)$, where n is the length of the text to be tested.

Given a reasonably large training corpus, which can be created combining a vast variety of English text sources, the probability of a quadgram can be

calculated by dividing the number of occurrences of that quadgram by the total number of quadgrams in the corpus.

Therefore, for the text `ORANGE`, the total probability is:

$$p(\texttt{ORANGE}) = p(\texttt{ORAN}) \times p(\texttt{RANG}) \times p(\texttt{ANGE})$$

Now, in order to prevent numerical underflows from happening when multiplying many small numbers, the logarithm of each probability is taken. Thus, the identity $\log(a * b) = \log(a) + \log(b)$ allows us to calculate the log probability as follows: [10]

$$\log(p(\texttt{ORANGE})) = \log(p(\texttt{ORAN})) + \log(p(\texttt{RANG})) + \log(p(\texttt{ANGE})) \tag{3.1}$$

A higher number resulting from 3.1 suggests that some given text is likely to be English. On the contrary, a lower number means that the latter is likely to have been obtained using an incorrect decryption key.

As in the case with frequency analysis, the method described above can fail to detect successful decryptions when texts deviate a great deal from standard frequencies. Furthermore, if there are less than a hundred letters available, this approach will probably not yield good results. [11, p. 56]

In conclusion, it is worth mentioning that in 1969 the French author Georges Perec wrote a 300-page novel without any word contaning the letter *e*. This book, entitled *La Disparition*, was then translated into English and other various languages, with the added constraint that the most frequently used letter in each respective alphabet was left out of the translations.

If the aforementioned book was to be encrypted using a substitution cipher, then a naive attempt to decrypt it with the help of any of the statistical tools previously described would certainly fail.

3.4.1 Solving a simple substitution cipher

Having found a function to score the goodness of a key, one can use it to allow a hill-climbing algorithm to iteratively refine an initial key. The latter is chosen using frequency analysis, namely assuming the most frequent ciphertext letter to correspond to plaintext E, the second most frequent ciphertext letter to correspond to plaintext T and so forth.

The pseudocode of the hill-climbing algorithm employed to break this cipher will now be given and then briefly discussed.

Algorithm 1 Random-restart hill climbing

```
1: currentKey ← initialGuess
2: bestKey ← currentKey
3: iteration ← 0
4: while iteration ≤ numberOfRandomRestarts do
5:     counter ← 0
6:     while counter ≤ numberOfLateralMoves do
7:         nextKey ← currentKey
8:         for all key ∈ Adj(currentKey) do
9:             if Score(key) ≥ Score(nextKey) then
10:                nextKey ← key
11:            end if
12:        end for
13:        if nextKey = currentKey then
14:            break
15:        end if
16:        if Score(nextKey) = Score(currentKey) then
17:            counter ← counter + 1
18:        else
19:            counter ← 0
20:        end if
21:        currentKey ← nextKey
22:    end while
23:    if Score(currentKey) > Score(bestKey) then
24:        bestKey ← currentKey
25:    end if
26:    currentKey ← RandomKey()
27:    iteration ← iteration + 1
28: end while
29: return bestKey
```

As previously said, *initialGuess* is simply a key generated using frequency analysis.

$Adj(currentKey)$ can be thought of as a set containing all the possible keys obtained by swapping two letters of $currentKey$. The key with the highest score among them will thus be chosen. The algorithm keeps doing so until no better key can be found.

With regard to random restarts, they have proved to be pretty useful when the ciphertext to decrypt is quite short. In such a case, the initial key is likely to be very inaccurate, because frequency analysis does not yield good results when there is not enough text available.

However, the algorithm runs significantly more slowly when random restarts are used. Consequently, its implementation will let the user decide whether to use them or not.

3.5 Cryptanalysis of the Vigenère cipher

As mentioned in Section 2.3, the Vigenère cipher was thought to be unbreakable for a long time. If long keys are employed, it can indeed be very difficult to break.

Although, once the length of the keyword is known, one can decrypt an encrypted message with ease.

Two statistical tools will be used for the cryptanalysis of this cipher: the *index of coincidence* will first be needed to guess the length of the key, after which we will make use of the *chi-squared statistic* to discover the key itself.

3.6 Index of coincidence

The **index of coincidence**, introduced by William F. Friedman in 1922 [12], is defined as the probability of picking two identical letters from a given text.

This can be calculated as follows:

$$I.C. = \frac{\sum_{i=A}^{Z} n_i(n_i - 1)}{N(N-1)}$$

where n_i is the frequency of the i-th letter of the alphabet and N is the length of the given text.

In fact, there are $\binom{n_A}{2}$ ways of choosing two A's from a text containing n_A occurrences of the letter A, and the same is true for all the other letters. Similarly, there are $\binom{N}{2}$ ways of choosing any pair of letters from a text of length N. Hence, the probability of picking the i-th letter twice is:

$$\frac{\binom{n_i}{2}}{\binom{N}{2}} = \frac{\frac{n_i(n_i-1)}{2}}{\frac{N(N-1)}{2}}$$

Common factors in the numerator and in the denominator cancel out. Thus, the fraction above can be simplified to:

$$\frac{\frac{n_i(n_i-1)}{2}}{\frac{N(N-1)}{2}} = \frac{n_i(n_i-1)}{N(N-1)}$$

If we sum all the probabilities for each letter of the alphabet, we obtain the formula previously given.

Since the index of coincidence is based on letter frequencies, the latter can be used to calculate its expected average value for any language. For example, the I.C. for English is equal to 0.067.

On the other hand, if the I.C. was to be calculated for a text containing random sequences of letters, then its value would be lower, because the characters in such a text are likely to be more uniformly distributed. This does not reflect the uneven distribution of letters in a natural language, causing the value of the I.C. to drop.

One of the interesting properties of the index of coincidence is that its value for a given text x does not change if x is encrypted using any monoalphabetic substitution cipher. The reason is that the individual probabilities of ciphertext letters will be a permutation of the individual probabilities of plaintext letters, therefore the probability of choosing any pair of identical letters at random will not be affected.

This property is fundamental in order to break a Vigenère cipher, as we will see in the next section.

3.6.1 Solving a Vigenère cipher

As said previously, the Vigenère cipher applies different shifts to the letters in the plaintext according to the chosen keyword. However, if one can assume the latter to be m-letter long, then $p_i, p_{m+i}, p_{2m+i}, \ldots, p_{n-m+i}$ are all enciphered using the same i-th key letter, hence using the same shift. For clarity, p is a plaintext letter, n is the length of the plaintext and $1 \leq i \leq m$.

Keeping this in mind, we can arrange the ciphertext into a certain number of columns, then compute the index of coincidence for each one of those and take the average. If the number of columns is equal to the length of the keyword, it is likely that the average I.C. will be close to the expected I.C. for the English language.

For example, consider the following encrypted message:

```
LPUQRHEEPRRMZBVYKFCTPXHSEJXUFROSRGVEXUBTMCTSEDMEHRCSZHU
PFMBXLRPCSSEHWARRAHUTRSHBOSABPPSDHJTGQGUPLMRCPIBSQTRFCG
SINCBVLQFFTWFSEHEEFRLHUBTMYFWGSEPBBAMOHHCIECENSZJRCWMHV
ZREWATXMBQHLMHVDXTSHDIATNMSAYGSSGUUEEXWPPAUHUZYFDVNXGFR
DSDQBYZQFFLXUCAD
```

To calculate the average I.C. assuming a four-letter keyword, we first split the message into four columns as follows:

Column 1	Column 2	Column 3	Column 4
L	P	U	Q
R	H	E	E
P	R	R	M
Z	B	V	Y
K	F	C	T
P	X	H	S
E	J	X	U
F	R	O	S
R	G	V	E
X	U	B	T
⋮	⋮	⋮	⋮

Then we compute the I.C. for each column thus obtained. Since the average resulted value is 0.042, it is not probable that the keyword is four-letter long.

This process is repeated for all the key lengths we want to test. In our example, we obtain the following values:

Key length	Average I.C.
2	0.044
3	0.046
4	0.042
5	0.069
6	0.047
7	0.043
8	0.039
9	0.048
10	0.076
11	0.044
12	0.044
13	0.041
14	0.040
15	0.085
16	0.038
17	0.041
18	0.045
19	0.044
20	0.079

Table 3.2: Indices of coincidence calculated for all key lengths between 2 and 20.

The table above clearly shows that when the key length is 5 (or a multiple of it), the average I.C. is close to its expected value.

Therefore, the message is split into five columns and frequency analysis is applied to each one of those. Specifically, the chi-squared statistic is used on each column to reveal a single letter of the keyword, in the same way it has been used to solve a Caesar cipher. After the keyword has been guessed, it is used to decrypt the whole message.

Deciphering the given ciphertext with the keyword `LEMON` produces the following plaintext:

```
ALICEWASBEGINNINGTOGETVERYTIREDOFSITTINGBYHERSISTERONTH
EBANKANDOFHAVINGNOTHINGTODOONCEORTWICESHEHADPEEPEDINTOT
HEBOOKHERSISTERWASREADINGBUTITHADNOPICTURESORCONVERSATI
ONSINITANDWHATISTHEUSEOFABOOKTHOUGHTALICEWITHOUTPICTURE
SORCONVERSATIONS
```

The content of the message — being the first paragraph of *Alice's Adventures in Wonderland* — is discovered!

3.7 Cryptanalysis of the Hill cipher

The Hill cipher appears more complicated to cryptanalyse compared to the ciphers seen before. In fact, *frequency analysis* is not very effective against this cipher, especially when the size of the matrix used as key increases.

The reason is that matrix multiplication provides *diffusion*, term introduced by Claude Shannon in 1949 to describe one of the properties a good cryptosystem should have [13]. Informally, diffusion is related to the way changes in the plaintext affect the ciphertext.

In the case of the Hill cipher, this means that when a letter in the plaintext changes, then the whole block to which that letter belongs changes as well. This shows a higher level of diffusion in comparison to the ciphers seen so far: for example, when a Vigenère cipher is used, changing a single plaintext letter results in just the corresponding ciphertext letter changing.

However, the Hill cipher is easily breakable by means of a *known-plaintext attack*. In fact, if an opponent knows n blocks of letters to occur in the plaintext, where n is the size of the key matrix, he or she can just solve a linear system in order to discover the decryption key, therefore getting to know the content of the encrypted message.

The following example, taken from [14], shows how this can be done using a technique called *crib dragging*.

In our case, the message to be recovered is the following:

```
FUPCMTGZKYUKBQFJHUKTZKKIXTTA
```

The attacker knows that `OFTHE` — which we will refer to as the *crib* — is part of the plaintext, but he or she does not know its exact position.

Nevertheless, given that the crib must appear somewhere in the plaintext, one of the following possibilities is true:

FU	PC	MT	GZ	KY	UK	BQ	FJ	HU	KT	ZK	KI	XT	TA
OF	TH	E.	..	..	..	..	..	..	..	..	..	..	..
.O	FT	HE	..	..	..	..	..	..	..	..	..	..	..
..	OF	TH	E.	..	..	..	..	..	..	..	..	..	..
..	.O	FT	HE	..	..	..	..	..	..	..	..	..	..
..	..	OF	TH	E.	..	..	..	..	..	..	..	..	..
..	..	.O	FT	HE	..	..	..	..	..	..	..	..	..

Table 3.3: This table shows how *crib dragging* is performed.

As we can see in the table above, which does not show every possible case, the crib is dragged along the ciphertext. Under the assumption that the crib begins at the second position, PC deciphers to FT and MT deciphers to HE.

If D denotes the decryption matrix the attacker is trying to discover, then:

$$D\begin{pmatrix}P\\C\end{pmatrix} = \begin{pmatrix}F\\T\end{pmatrix} \rightarrow D\begin{pmatrix}15\\2\end{pmatrix} = \begin{pmatrix}5\\19\end{pmatrix} \pmod{26}$$

$$D\begin{pmatrix}M\\T\end{pmatrix} = \begin{pmatrix}H\\E\end{pmatrix} \rightarrow D\begin{pmatrix}12\\19\end{pmatrix} = \begin{pmatrix}7\\4\end{pmatrix} \pmod{26}$$

Now, D can be obtained as follows:

$$D\begin{pmatrix}15 & 12\\2 & 19\end{pmatrix} = \begin{pmatrix}5 & 7\\19 & 4\end{pmatrix} \pmod{26}$$

Therefore:

$$D = \begin{pmatrix}5 & 7\\19 & 4\end{pmatrix}\begin{pmatrix}15 & 12\\2 & 19\end{pmatrix}^{-1} \pmod{26}$$

The inverse of the matrix above can be calculated using Equation 2.1 given previously. Hence:

$$\begin{pmatrix}15 & 12\\2 & 19\end{pmatrix}^{-1} = 1^{-1} \times \begin{pmatrix}19 & -12\\-2 & 15\end{pmatrix} \equiv \begin{pmatrix}19 & 14\\24 & 15\end{pmatrix} \pmod{26}$$

At this point, D can be easily determined. However, it should be noted that not every matrix is invertible. In such a case, one can only continue dragging the crib in the hope that it occurs again further in the text.

$$D = \begin{pmatrix}5 & 7\\19 & 4\end{pmatrix}\begin{pmatrix}19 & 14\\24 & 15\end{pmatrix} = \begin{pmatrix}263 & 175\\457 & 326\end{pmatrix} \equiv \begin{pmatrix}3 & 19\\15 & 4\end{pmatrix} \pmod{26}$$

If the attacker tries to decrypt the message using D, the result would be:

```
FRFTHEZYSSQYVFETLVBAFVACONFZ
```

This attempt has evidently failed, since the result is not intelligible. This leads the attacker to believe the initial assumption of the crib beginning at the second position to be wrong.

Thus, the attacker repeats the described procedure for every possible starting position. In fact, if the crib was aligned at the eighteenth position, the decryption matrix would be:

$$D = \begin{pmatrix} 17 & 5 \\ 18 & 23 \end{pmatrix}$$

Using the matrix above as key, the message decrypts to:

```
DEFENDTHEEASTWALLOFTHECASTLE
```

The attacker can now stop dragging the crib, since he or she knows that this attempt has been successful.

However, the implementation of this procedure will not stop until reaching the last possible starting position. In fact, every decryption result will be rated using quadgram statistics, as described in Section 3.4, in order to decide which one is most likely to be meaningful.

In conclusion, it is worth noting that, when using a 2×2 key matrix, the length of the crib should be 5. In fact, if the crib was just 4-letter long, then it would have to be aligned at the beginning of a block in order for this method to work. Likewise, 11 consecutive letters are necessary in case a 3×3 key matrix is being used.

Chapter 4

Software design

This chapter provides insight into the design of the application. Class diagrams will be used to illustrate how the domain logic has been modelled, followed by a brief description of the main architectural pattern employed.

4.1 Domain logic

The **domain logic** is mainly encapsulated into three interfaces, which are shown in the following class diagrams along with some of the classes implementing them.

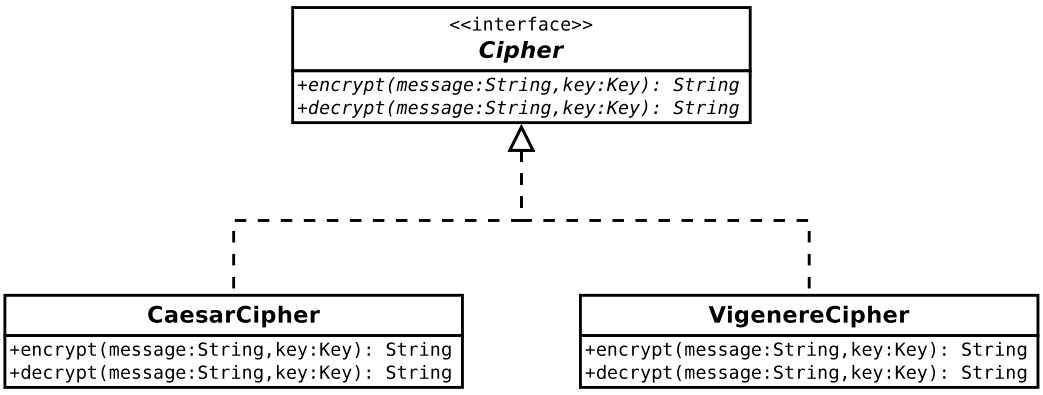

Figure 4.1: Class diagram showing two classes implementing the `Cipher` interface.

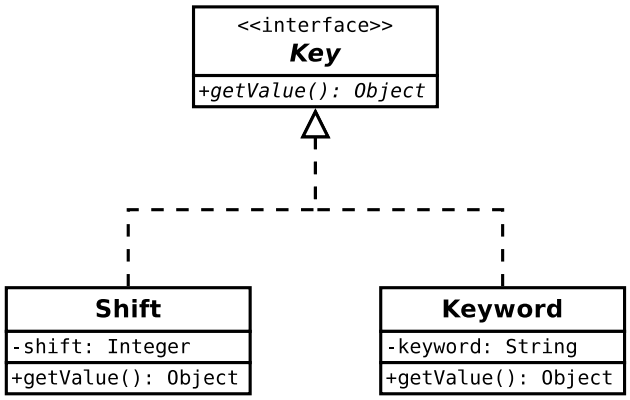

Figure 4.2: Class diagram showing two classes implementing the `Key` interface.

The `Cipher` and `Key` interfaces define the methods to encrypt and decrypt an input message, whereas the `Breaker` interface defines a single method whose function is to try to recover an encrypted message.

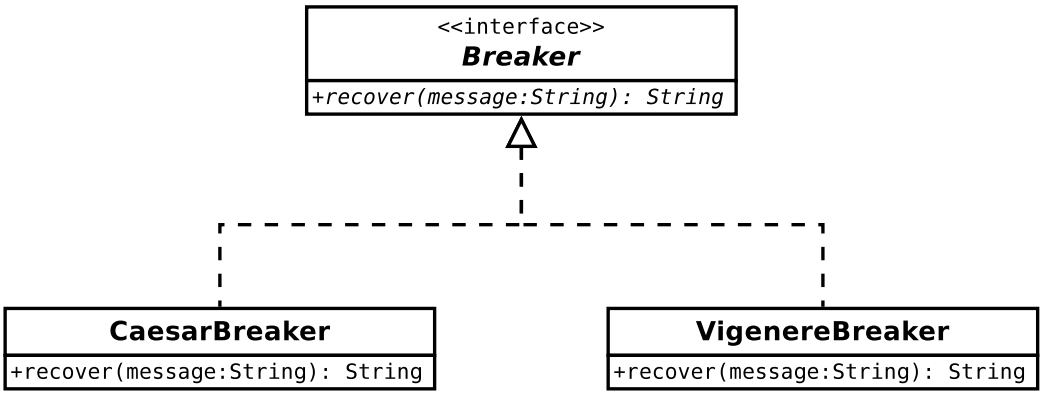

Figure 4.3: Class diagram showing two classes implementing the `Breaker` interface.

In order to add a new cipher, its encryption and decryption algorithms need to be implemented. Furthermore, the definition of a new key is often necessary, since different ciphers' keys do not normally share the same structure.

At the moment, the ciphers available are the ones described in the previous sections. However, the application can be easily extended by adding further ciphers as explained above.

Finally, the `TextStatistics` class should be mentioned, as it offers the methods necessary to gain all the information needed to perform the cryptanalysis.

This is achieved by delegating smaller tasks to a variety of text analysers, and then returning the final result to the caller.

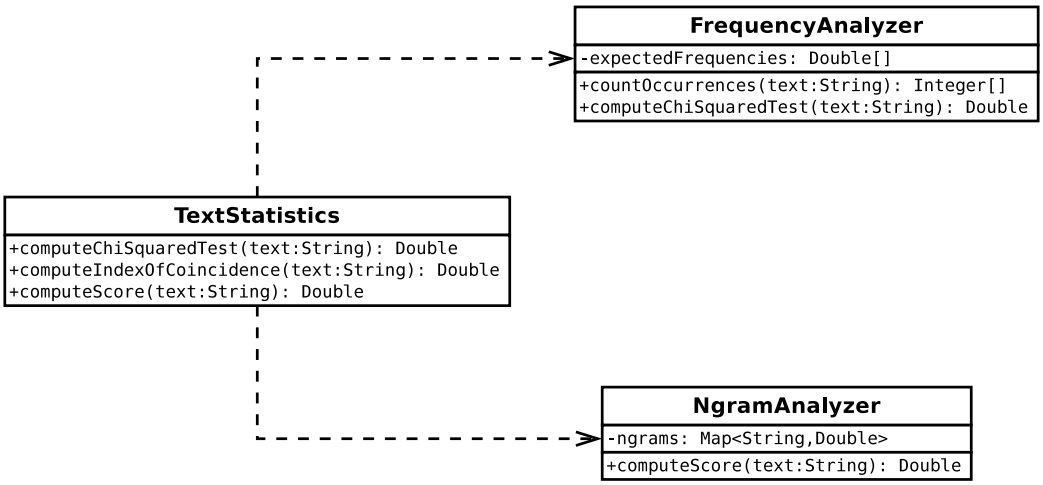

Figure 4.4: Class diagram showing dependency between the `TextStatistics` class and two text analysers.

A *facade pattern* has been used to provide a simpler interface to the callers, which do not interact directly with the text analysers.

4.2 Model-view-controller

When dealing with graphical user interfaces, it is often desirable to separate the code implementing the domain logic from the code handling the interaction with the user.

Model-view-controller is an architectural pattern which, if properly followed, enables one to fulfil this requirement.

In our case, the *model* consists of the domain objects implementing the cryptographic and cryptanalysis methods, the *view* comprises the XML files describing the appearance of the user interface, and finally the *controller* is simply the code being run when the user interacts with the application.

The next chapter will give more detailed information regarding how this pattern has been implemented.

Chapter 5

Java implementation

This chapter will give information regarding the implementation details of the software developed. When needed, code samples and screenshots will be also provided.

5.1 Project hierarchy

The source code has been organised into Java packages, in order to group related classes together.

The following packages have thus been created:

- `breakers`
- `ciphers`
- `exceptions`
- `gui`
- `gui.controllers`
- `keys`
- `math`
- `utils`

In addition, the images and resources used by the application have been stored respectively in the `images` and `resources` folders. Finally, `config.properties` has been used to save a few configuration settings.

5.1.1 The `ciphers` package

The `ciphers` package contains all the implemented ciphers. As shown in Figure 4.1, every cipher encrypts a message using a key, which is represented by the `Key` interface.

However, different ciphers need different keys, therefore a cast to the appropriate type returned by the `getValue()` method defined in the `Key` interface will be necessary both to encrypt and decrypt a message.

The implementation of the encryption method of the Vigenère cipher will now be given as an example.

```
@Override
public String encrypt(String message, Key key) {
    String keyword = (String) key.getValue();

    StringBuilder encrypted = new StringBuilder();

    int m, k;

    for (int i = 0, j = 0; i < message.length(); i++) {
        m = alphabet.getIndex(message.charAt(i));
        k = alphabet.getIndex(keyword.charAt(j));

        encrypted.append(alphabet.getChar(m + k));

        j = (j + 1) % keyword.length();
    }

    return encrypted.toString();
}
```

The encrypted message is created using a `StringBuilder` object rather than a `String` object. This avoids to allocate a new string at each iteration in the `for` loop.

Also, `alphabet` is an instance of the `Alphabet` class containing the 26 uppercase letters of the English alphabet. It is used to conveniently convert a character into a number and vice versa.

5.1.2 The `keys` package

The `keys` package contains a key for each cipher implemented. Keys are simply used to store a value which is returned when the `getValue()` method

is invoked.

For example, the `Shift` class represents the key used by a Caesar cipher. Its implementation is given below.

```
package keys;

public class Shift implements Key {

    private int shift;

    public Shift(int shift) {
        this.shift = shift;
    }

    @Override
    public Object getValue() {
        return new Integer(shift);
    }

}
```

5.1.3 The `breakers` package

The `breakers` package contains all the classes implementing the `Breaker` interface. For each cipher, a corresponding breaker whose only purpose is to recover a message encrypted with that cipher has been created.

For example, a `VigenereBreaker` object is meant to recover messages encrypted using a `VigenereCipher` object, whereas a `HillBreaker` object is meant to recover messages encrypted using a `HillCipher` object, and so forth.

The source code for the `recover(String message)` method of these classes can be quite lengthy, hence it has been omitted. However, the means by which cryptanalysis has been performed have been explained in Chapter 3.

5.2 Graphical user interface

The **graphical user interface** has been developed using JavaFX, which has been included in Java SE as a native library with the aim of replacing the older Swing library.

The software produced is therefore cross-platform, since it just needs Java SE to be installed.

JavaFX enables one to enforce the use of the MVC pattern — described in Section 4.2 — by providing the structure to build a user interface separated from the domain logic. This is achieved through the use of FXML and Java. The former is an XML-based language used to design the appearance of the application, whereas the GUI's behaviour is defined by the Java controllers included in the `gui.controllers` package.

The two main views of the application are shown below.

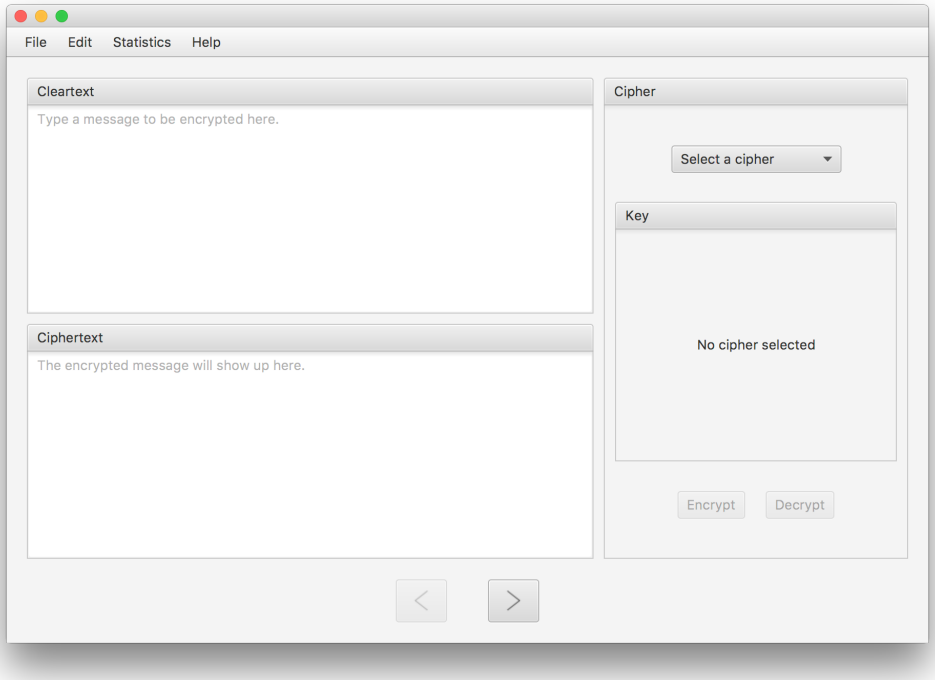

Figure 5.1: This view lets the user encrypt and decrypt messages.

The first view lets the user encrypt and decrypt messages using the selected cipher. The latter can be chosen using the drop-down list which can be seen in the top-right corner.

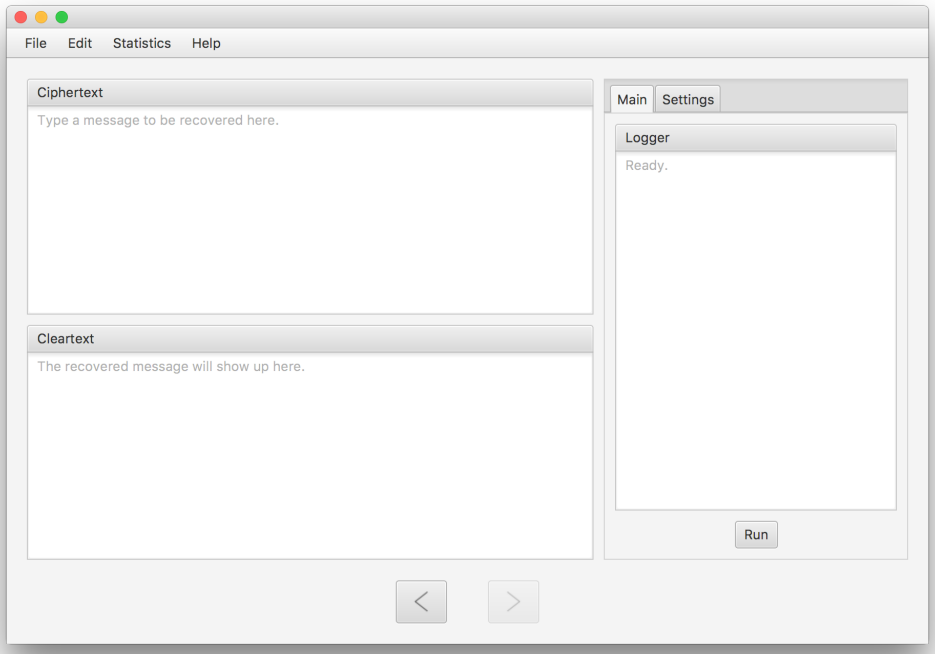

Figure 5.2: This view lets the user run the cryptanalysis process.

The second view includes the controls necessary to run the cryptanalysis process. Also, the *Settings* tab contains some options — not shown in Figure 5.2 — which modify the implemented algorithms, for instance letting the user enable or disable certain `Breaker` objects or choose whether to use a known-plaintext attack or not.

Finally, it is possible to display a bar chart to compare the letter frequencies of the plaintext with the letter frequencies of the ciphertext.

For example, consider the plaintext below:

```
ALICEWASBEGINNINGTOGETVERYTIREDOFSITTINGBYHERSISTERONTH
EBANKANDOFHAVINGNOTHINGTODOONCEORTWICESHEHADPEEPEDINTOT
HEBOOKHERSISTERWASREADINGBUTITHADNOPICTURESORCONVERSATI
ONSINITANDWHATISTHEUSEOFABOOKTHOUGHTALICEWITHOUTPICTURE
SORCONVERSATIONS
```

If this plaintext was to be encrypted with a Vigenère cipher using `LEMON` as keyword, then the bar charts in Figure 5.3 would be plotted.

Figure 5.3: Bar charts displaying the letter frequencies of the plaintext and of the ciphertext.

5.3 Workflow of the application

To conclude this chapter, the workflow of the application will be presented in a stepwise manner, outlining how the entire cryptanalysis process has been implemented.

The class handling this process is the `BreakerController` class, which coordinates all the implemented breakers and returns the best result once they have finished running.

Thus, when the code in the `run()` method of the `BreakerController` class is executed, the following steps are performed:

1. Compute the index of coincidence of the entered ciphertext.
2. If it is close to its expected value for English, assume that a monoalpha-

betic cipher has been used. Otherwise, assume that a polyalphabetic cipher has been used.

3. Wait until each breaker has finished running.
4. Choose the best decryption result using quadgram statistics.
5. Show the result chosen in the previous step.

In order to choose the best decryption result, all the strings returned by the breakers are first stored in a `List<String>`. Then, the latter is passed to a method — whose source code is given below — which returns the string obtaining the highest score.

```
public static String findBestCandidate
        (List<String> candidates) {

    String bestCandidate = null;
    double maxScore = Double.NEGATIVE_INFINITY;

    for (String candidate : candidates) {
        double score = ngramAnalyzer.computeScore(candidate);

        if (score > maxScore) {
            // Update maximum score
            maxScore = score;

            // Store the best string found so far
            bestCandidate = candidate;
        }
    }

    return bestCandidate;
}
```

The method above is defined in the `TextStatistics` class, which also offers other utility methods used to calculate the statistics described in Chapter 3.

Chapter 6

Conclusion

This dissertation has shown how classical ciphers can be broken even when one only knows a sufficient amount of ciphertext.

Despite the fact that some of these ciphers can offer a higher degree of security when certain keys are being employed, they are no longer used when confidentiality has to be guaranteed. Indeed, modern ciphers resist much stronger attacks than the ones that have been presented, and are therefore chosen when developing real-world applications.

On the other hand, some of the main concepts behind cryptanalysis have been explained and demonstrated in practise through the development of an automatic tool implementing them.

Future work will be aimed at adding support for multiple languages. This can be achieved by feeding the software with statistical data related to the various languages to be supported.

In conclusion, analysing the weaknesses of outdated ciphers can be an effective means for cryptographers to design new ciphers that stand up to the rapidly changing world of technology, especially in a society where information has become an increasingly valuable asset to protect.

References

[1] Wikipedia, *ROT13*, https://en.wikipedia.org/wiki/File:ROT13_table_with_example.svg, 2007.

[2] Wikipedia, *Simple substitution*, https://en.wikipedia.org/wiki/Substitution_cipher#Simple_substitution, 2016.

[3] Wikipedia, *Vigenère cipher*, https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher, 2016.

[4] Wikipedia, *Vigenère square*, https://en.wikipedia.org/wiki/File:Vigen%C3%A8re_square_shading.svg, 2011.

[5] Wikipedia, *Hill cipher*, https://en.wikipedia.org/wiki/Hill_cipher#Example, 2016.

[6] Wikipedia, *Letter frequency*, https://en.wikipedia.org/wiki/Letter_frequency, 2016.

[7] Wikipedia, *English letter frequency*, https://en.wikipedia.org/wiki/File:English_letter_frequency_(alphabetic).svg, 2010.

[8] Practical Cryptography, *Chi-squared statistic*, http://practicalcryptography.com/cryptanalysis/text-characterisation/chi-squared-statistic/, 2012.

[9] Amrapali Dhavare, Richard M. Low, Mark Stamp, *Efficient Cryptanalysis of Homophonic Substitution Ciphers*, http://www.cs.sjsu.edu/~stamp/RUA/homophonic.pdf

[10] Practical Cryptography, *Quadgram Statistics as a Fitness Measure*, http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/, 2012.

[11] Simon Singh, *The Code Book*, Fourth Estate, 1999.

[12] William F. Friedman, *The Index of Coincidence and its Applications in Cryptography*, Riverbank Laboratories, 1922.

[13] Claude Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal, 1949.

[14] Practical Cryptography, *Cryptanalysis of the Hill Cipher*, `http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-hill-cipher/`, 2012.