# *Using Frequency Analysis to Decrypt Monoalphabetic Ciphers*

*Omar Rodríguez Laborde*
*Master in Computer Science*
*Dr. Jeffrey Duffany, Ph.D.*
*Electrical & Computer Engineering and Computer Science Department*
*Polytechnic University of Puerto Rico*

***Abstract*** *— Over the past few years' technology has taken over society. Everyone uses technology. It makes everything easier, such as paying bills, buying food or any stuff that you use in your daily basis. People tend to store all the information on their online accounts such as; credit card numbers, address, personal information and others, all this without knowing how easy or hard is to a malicious person get into all this information. In order to understand this you have to think as an hacker, that's the only way you can help yourself and help others to understand how this people are getting thru all the security measures that companies are trying to use. Companies' security is only 50% of the equation, the other 50% is you as a user and how well you are using the tools that this company give you. Some of the hacker technique are: Frequency Analysis, Brute Force, phishing, etc. Frequency Analysis is a fundamental cryptanalytic technique that can be applied to any monoalphabetic cipher. The proposed program uses combined techniques of monogram, bigrams, trigram, frequency analysis, keywords rules and dictionary to decrypt monoalphabetic ciphers.*

***Key Terms*** *— Ciphers, Cryptanalytic, Decrypt, Monoalphabetic.*

## INTRODUCTION

A monoalphabetic cipher [1], uses the same substitution across the entire message. For example, if you are able to decrypt the letter T and now you know that the letter T is D, this will not change for the entire message. Some examples of monoalphabetic ciphers [1] are: Caesar cipher, atbash cipher, keyword cipher, pigpen cipher, etc. All this type of ciphers can be cracked by using frequency analysis, and trial and error.

Frequency Analysis [2] is the study of the frequency of letters or groups of letters in a cipher text. For each language there is a different frequency order from the most to the less used letters of that language. N-grams [3] are constantly used in frequency analysis [2]. An n-gram is contiguous sequence of n items from a given sequence of text or speech. With this data you can also create most used bigrams and trigrams. For instance given a section of the English language E, T, A, O, I, N, S are the most commons letters, TH, HE, IN, ER, AN, RE, ND are the most common bigrams and THE, AND, ING, HER, HAT, HIS are the most common trigrams.

Using this information you can create a more easy approach at the moment where you need to decrypt a monoalphabetic cipher [1] without knowing what cypher was used to encrypt the message or what key was applied to encrypt the message.

Some of the challenges of this project are how to make a substitution and identify that those substitutions are correct, how to combine all the data to increase the chance of make a correct letter swap, identify words after swapping letters, parse the message correctly and confirm that the message make sense and confirm that you successfully decrypted the message. The program I created in this project uses monograms, bigrams, trigrams, brute force, English dictionary of 100,000+ words, in order to decrypt [4] the message without any user input.

## HISTORY

Al-Kindi is credited with developing a method whereby variations in the frequency of the occurrence of letters could be analyzed and exploited to break ciphers. This was around AD 800, since then frequency analysis [2] have been used in rotor machines, World War I, World War II, etc.

In the present there is different websites and applications that have the options to do a frequency analysis [2] to a text, but none of them have the ability to decrypt the message without a user feedbacks. So basically those website can help a user to do letter swaps and show the new text but doesn't have the capabilities to know without any user input if the text is correct or if is needed to do more letters swaps in order to completely decrypt the message.

All this different applications in our world right now tend to help but this program tries to automatize the frequency analysis [2] in order that doesn't required the user interaction that needs all the others.

## PURPOSE

The only ways where you can break any of the different monoalphabetic ciphers [1] using the same approach are brute force [5] or frequency analysis [2]. If you use a brute force to break a monoalphabetic cipher [1] would be 26 letters per mutating 26 times equals 4.03291461126606E+26 tries. On a computer this amount of tries would take years to execute. So what we got left is the frequency analysis [2].

This application takes frequency analysis [2] and facilitate the decryption of any monoalphabetic cipher [1] automatically. Just need to provide the text as showed in Figure 1 and the program do the rest.

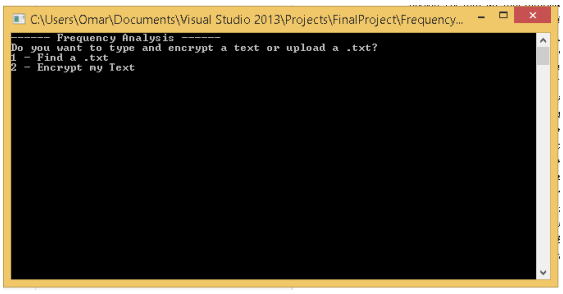



**Figure 1**
**Main Page – File Path Input**

Using this application you have the advantage of having an automated program breaking a cipher [1] without needing to wait years to obtain the decrypted code using brute force or without having to provide any data input to the application.

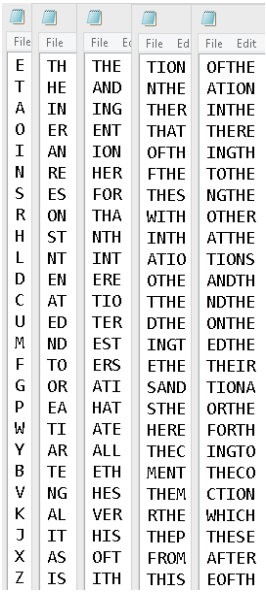
## SCHEME

The application will run using the input text or the input file by the user and will allow the user to do different options with this encrypted text.

In the folder where the application is started, some data files will be required Referred to Figure 2. Those files are:

- List of most used monograms
- List of most used bigrams
- List of most used trigrams
- List of most used quadgrams
- List of most used quintgrams
- English Dictionary
- List of most Used English words

The application will start taking the input text and changing all letters to lowercase, deleting any spaces between words, and also deleting any special characters from the text.



| | | | | |
|---|---|---|---|---|
| E | TH | THE | TION | OFTHE |
| T | HE | AND | NTHE | ATION |
| A | IN | ING | THER | INTHE |
| O | ER | ENT | THAT | THERE |
| I | AN | ION | OFTH | INGTH |
| N | RE | HER | FTHE | TOTHE |
| S | ES | FOR | THES | NGTHE |
| R | ON | THA | WITH | OTHER |
| H | ST | NTH | INTH | ATTHE |
| L | NT | INT | ATIO | TIONS |
| D | EN | ERE | OTHE | ANDTH |
| C | AT | TIO | TTHE | NDTHE |
| U | ED | TER | DTHE | ONTHE |
| M | ND | EST | INGT | EDTHE |
| F | TO | ERS | ETHE | THEIR |
| G | OR | ATI | SAND | TIONA |
| P | EA | HAT | STHE | ORTHE |
| W | TI | ATE | HERE | FORTH |
| Y | AR | ALL | THEC | INGTO |
| B | TE | ETH | MENT | THECO |
| V | NG | HES | THEM | CTION |
| K | AL | VER | RTHE | WHICH |
| J | IT | HIS | THEP | THESE |
| X | AS | OFT | FROM | AFTER |
| Z | IS | ITH | THIS | EOFTH |

**Figure 2**
**N-grams Files [3]**

There is a function that will replace letters in the encrypted text, each time that the program replace a letter in the text the new letter will be uppercase in order to the application remember which letters have been already swapped, then the

application will do a statistical approach in order to select which one of all the letters combinations fit best and makes more sense out of the unencrypted text. Then another function will try to find words of 4 letters plus in the selected combinations in order to confirm that this is the right combination in order to break the unencrypted text.

## TECHNOLOGIES

For this application two programming languages where used in order to create a balance between coding speed and performance. Those programming languages are:

- Python
- C# [6]

C# [6] provides a tight integration with the rest of .NET framework and other Microsoft products. The integrated development environment used was Visual Studio 2015, this IDE is perfect for the C# [6] environment and a library was used in order to be able to integrate Python with C# [6], this library is IronPython, this allow to use subroutines in python and receive the results on C# [6]. IronPython is an implementation of Python language looking for integrate python with .NET Framework and Mono.

One of IronPython's key advantages is in its function as an extensibility layer to application frameworks written in a .NET language. It is relatively simple to integrate an IronPython interpreter into an existing .NET application framework. LINQ extends the C# [6] language by the addition of query expressions such like SQL statements, this option facilitate the extract and data process from arrays and enumerable classes. Refer to figure 3 and figure 4 to see Python integration in C# [6] and LINQ uses inside C# [6].

In order to be able to search inside those big files I had to do some performance test in order to provide what was the best way to search patterns on the files. And I found that Regular Expressions was the way to go. Regular Expressions is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs. The regular expressions also helped to create dynamic functions for different kind of uses such as get any size of n-grams [3] from a text, get different size of words in the dictionary, and find words with a certain prefix, suffix or both. Figure 4 have an example of the regex uses in C# [6].

```
using IronPython.Hosting;
using Microsoft.Scripting.Hosting;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FrequencyAnalysis.Library
{
    public static class PythonRef
    {
        public static dynamic CreatePythomClass(string ProjectContainerPath, string PyFilePath, string ClassName)
        {
            ScriptEngine engine = Python.CreateEngine();
            engine.SetSearchPaths(new List<string>() { ProjectContainerPath });
            ScriptSource source = engine.CreateScriptSourceFromFile(PyFilePath);
            ScriptScope scope = engine.CreateScope();
            source.Execute(scope);
            dynamic CreatedFunctions = scope.GetVariable(ClassName);
            return CreatedFunctions();
        }

        /*****************************************************Classes Python Here**************************

        public static string generateSpacesBetweenWords<TModel>(dynamic FunctionName, TModel Model)
        {
            return FunctionName.generateSpacesBetweenWords(Model);
        }

        public static void getDesencryptProbabilities<TModel>(dynamic FunctionName, TModel Model)
        {
            FunctionName.getDesencryptProbabilities(Model);
        }

        public static string getDesencryptText<TModel>(dynamic FunctionName, TModel Model)
        {
            return FunctionName.getDesencryptText(Model);
        }
    }
}
```

**Figure 3**
**Python Integration in C# [6]**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using IronPython.Hosting;
using IronPython.Runtime;
using IronPython;
using Microsoft.Scripting.Hosting;
using Microsoft.Scripting;
using FrequencyAnalysis.Library.Models;

namespace FrequencyAnalysis.Library
{
    public static class Analysis
    {
        public static IEnumerable<KeyValuePair<string,int>> getRepeatedLetters(string text,int times)
        {
            List<KeyValuePair<string, int>> result = new List<KeyValuePair<string, int>>();

            foreach(char c in PreSets.Alphabet)
            {
                string toFInd = "";
                for (int x = 0; x < times; x++)
                {
                    toFInd = toFInd + c.ToString();
                }
                result.Add(new KeyValuePair<string,int>(toFInd,Regex.Matches(text, toFInd).Count));
            }
            return result.Where(x=>x.Value>0).OrderByDescending(x=>x.Value);
        }
```

**Figure 4**
**LINQ and Regex Integration in C# [6]**

## REQUIREMENTS AND OBJECTIVES

This application will provided the user's ability to break any monoalphabetic cipher [1] while the user can work in a different task. This application objective to fulfill are:

- Automation of Frequency Analysis [2] decryption
- Balance performance and optimization in order to provide a quality application

As a requirement the application should be able to manage any type of encrypted text data with lowercase, uppercase, special characters, space between words or letters and without any space. The application should be able to manage English text. The application should be able to run in a standard grade PC. The application should be able to manage any type of monoalphabetic encryption.

## CONSIDERATIONS

The project accomplish all the requirements mention before. There was some tasks that wasn't easy to accomplish such as:

- Maintain Performance – in this type of projects always have to maintain a balance between how accurate the program is and how is the performance. Sometimes you have to choose between being more accurate and maybe take a little longer in order to provide the results or give a faster result with some probability that the answer is not accurate as expected.
- Parsing of the sentences - encrypted text does not contain any space or special characters, so basically at the end of the result you have to set the spaces between words, and have to make sure that it make sense.
- Selection of best combination – this was a really big challenge, in order to being able to choose which was the best combination I had to create a function to find words inside the whole plain text counting the words amount I had to eliminate all the words that contain 4 letters or less in order to take out of the equation some combinations that contains false positive.

In addition, the application contain few others functions in order to facilitate the user ability to find in the encrypted text any kind of n-gram, missed letters, and doubled letters. This functions are standards in any frequency analysis [2] application. Including the Requirements list, the project design and the development of the tool, we need to take into consideration the results of this application and prove that the needs for the user a fulfilled and over expectations. In figure 6 you will be able to see the main menu of the program. We will see the functionality of the program since the program is started and the user is asked to input a text manually to encrypt it or upload a .txt already encrypted.

The menu of this application have been done simple enough to any kind of user being able to make use of all the applications advantages without any training in it.

Coming up next, the result of this project will be presented. Providing better view of the tool which makes this work and it processes.

## RESULTS

This application will be available for the use of researchers, students and any professional on the computer security field. Everyone will find this application appealing because is not time consuming and it works as intended. The application consist of different kind of options. Some of the options are:

- Upload .txt or Data Entry – how data will be entered to the program.
- Get Letters Count – do the normal frequency analysis of the cipher text.
- Get Ngrams – Create a list of n-grams with the user entry amount of the cipher text.
- Get missing letters – Create a list of missing letters from the cipher text.
- Get repeated letters – Create a list of double letters in the cipher text.
- Decrypt Message – Decrypts the cipher text.

- Exit – close the application.

The following figures show the results for the different options in the application.

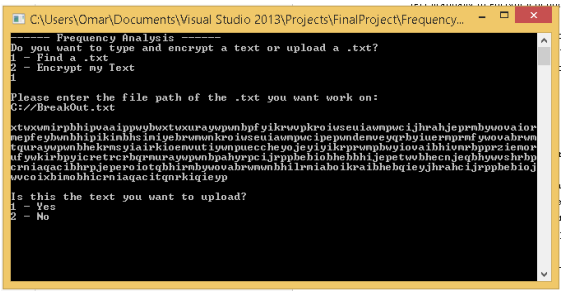



**Figure 5**
**.txt File Upload**

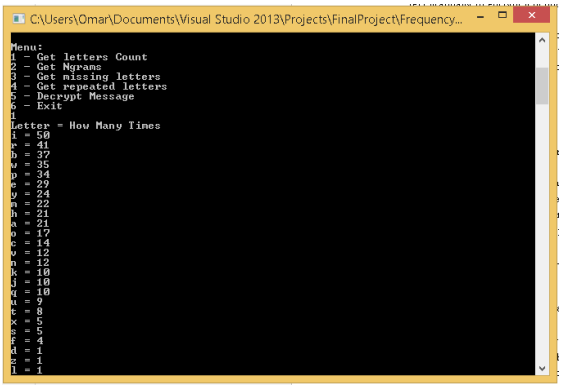



**Figure 6**
**Main Menu - Get Letters Count**

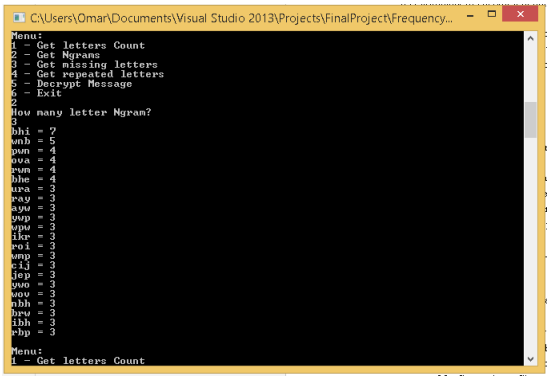



**Figure 7**
**Main Menu - 3-letters N-gram**

In the figure 10 you can take a look of how the program selects what is the best combination for the unencrypted text, then he separates each words to make sentences with more logic. All the words are compare to a dictionary to make sure each one of the words make sense, and there is nothing without unencrypt.

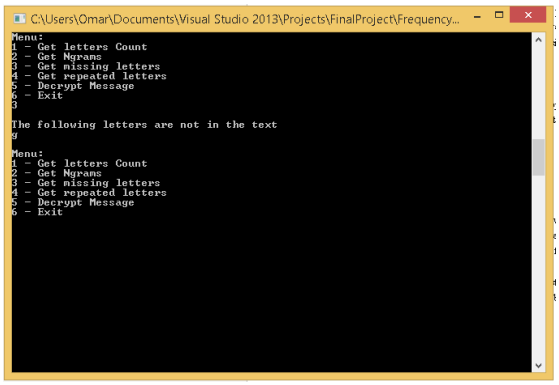



**Figure 8**
**Main Menu - Missing Letters**

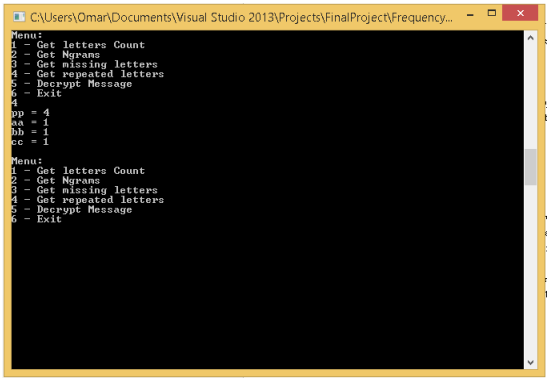



**Figure 9**
**Main Menu - Repeated Letters**


**Figure 10**
**Main Menu – Decrypted Message**

## FUTURE WORK

Some implementations that can be done to the program for the future are:

- Adding more languages to the application. Right now it only support English language.
- Create a Function in order to detect the sentence without needing to loop for the best combinations. This will also help the performance of the program.

- Create a Web Page interface in order to provide a more User Friendly Program.
- Create a second validation of the data result in order to provide a more accurate data.

With this changes, the application ensures a solid future.

## CONCLUSION

This application was able to decrypt monoalphabetic ciphers in more than 75% of the cases. The estimate time for any cipher size would be between 55minutes to 75 minutes. This tool is able to decrypt any type of ciphers without any issues. The ram data needed to run the application is 2GB.

This proves that any monoalphabetic cipher are unlikely to provide any real protection for confidential data.

## ACKNOWLEDGEMENTS

This project would like to acknowledge Dr. Jeffrey Duffany for his contribution of ideas and guidance during the project and the knowledge provided in the different courses.

## REFERENCES

[1] D. Rodriguez-Clark (2013). Monoalphabetic Substitution Ciphers in Crypto Corner [Online]. Available: http://crypto.interactive-maths.com/monoalphabetic-substitution-ciphers.html.

[2] M. Markowitz (2004). Frequency Analysis in Wikipedia, Wikimedia Foundation [Online]. Available: https://en.wikipedia.org/wiki/Frequency_analysis.

[3] M. Davies (2014). N-grams: Based on 520 million word COCA corpus, in Ngrams Data. [Online]. Available: http://www.ngrams.info/.

[4] M. Rouse (2008). What is data encryption/decryption IC? - definition from SearchSecurity. [Online]. Available: http://searchsecurity.techtarget.com/definition/data-encryption-decryption-IC.

[5] M. Crypto (2004). Brute-force attack, in Wikipedia, Wikimedia Foundation [Online]. Available: https://en.wikipedia.org/wiki/Brute-force_attack.

[6] M. Corp (2003). C# Reference from Microsoft Corp. [Online]. Available: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx.