

**Computer Science and Engineering**

**Cryptanalysis of Permutation Ciphers**

**Applied Cryptography**

Team Members: **Ajay Shenoy**

**George Lin**

**Shearyar Shamim Khan**

**Table of Contents**

**Part 1: Programming Report | 3**

1.1 Team Members and Specification Changes | 3

1.2 Cryptanalysis Approach Used in the Program | 5

**Part 2: Extra Credit: A survey on permutation ciphers | 8**

**References/** **Works Cited | 10**

**Part 1:**

**Programming Report**

**1.1 Team Members and Specification Changes**

**Team Members:**

**Ajay Shenoy**

* Wrote sections 1.1 and half of 1.2 of the report
* Wrote algorithm and decryption schemes for dictionary one

**George Lin**

* Helped with algorithms and decryption schemes through pair programming for dictionary 2

**Shearyar Shamim Khan**

* Created the project program and handled the coding aspects
* Wrote algorithms and decryption schemes for dictionary 2
* Wrote half of 1.2 of the report

**No modifications were made** with respect to the following specifications outlined in the project:

This cryptanalysis project consists of a software implementation of an algorithm that tries to decrypt an L-symbol challenge ciphertext computed using a permutation cipher. Informally speaking, your program's goal is to find the plaintext used to compute this ciphertext within a reasonable amount of time. Specifically, your program should print on screen something like "Enter the ciphertext:", obtain the ciphertext from stdin, apply some cryptanalysis strategy and output on screen something like "My plaintext guess is:" followed by the plaintext found by your strategy. In doing that, your program is allowed access to:

1. The ciphertext (to be taken as input from stdin)
2. A plaintext dictionary (to be posted on top of this web page), containing a number q of plaintexts, each one obtained as a sequence of space-separated words from the English dictionary
3. Partial knowledge of the encryption algorithm used (to be described below).

Your program is not allowed access to:

1. The key used by the permutation cipher.
2. Part of the encryption scheme (to be detailed below).

The plaintext is a space-separated sequence of words from the English dictionary (the sentence may not be meaningful). The key is a map from each English alphabet (lower-case) letter to a list of numbers randomly chosen between 0 and 102, where the length of this list is the (rounded) letter’s frequency in English text, as defined in the table below. The ciphertext is a space-separated sequence of encryptions of words, where each word is encrypted as a comma-separated list of numbers between 0 and 102, and these numbers are computed using the table below.

|  |  |  |
| --- | --- | --- |
| English letters | Average frequency | Key values (randomly chosen distinct numbers between 0 and 102) |
| a | 8 | k(a,1),…,k(a,8) |
| b | 1 | k(b,1) |
| c | 3 | k(c,1),…,k(c,3) |
| d | 4 | k(d,1),…,k(d,4) |
| e | 13 | k(e,1),…,k(e,13) |
| f | 2 | … |
| g | 2 |  |
| h | 6 |  |
| i | 7 |  |
| j | 1 |  |
| k | 1 |  |
| l | 4 |  |
| m | 2 |  |
| n | 7 |  |
| o | 8 |  |
| p | 2 |  |
| q | 1 |  |
| r | 6 |  |
| s | 6 |  |
| t | 9 |  |
| u | 3 |  |
| v | 1 |  |
| w | 2 |  |
| x | 1 |  |
| y | 2 |  |
| z | 1 |  |

The permutation cipher works as follows. It takes as input a plaintext from a message space and a key randomly chosen from a key space and returns a ciphertext.

* The message space is the set {<space>,a,..,z}^L. In other words the message m can be written as m[1]...m[L], where each m[i] is in {(space>,a,..,z}
* The ciphertext c can be written as c[1],...,c[L], where each c[i] is in {<space>,0,..,102}. To avoid ambiguities, cyphertext symbols are separated by a comma.
* The key space is the set of random maps from {0,..,26} to a permutation of all numbers in {0,…,102}, grouped in 26 lists, each list having length determined by column 2 of the table below.
* The encryption algorithm works as follows. A space in the plaintext is mapped to a space in the ciphertext. For each message character m[j], the algorithm finds m[j] in column 1 of the table below, and returns one of the keys in column 3 of the same row. The computation of which key is returned by the algorithm is based on a scheduling algorithm which is intentionally left unknown and is a deterministic algorithm (that is, it does not use new random bits) that may depend on j, L and the length of the list on that row.
* The decryption algorithm does the inverse process. It maps space to a space in the plaintext. On any ciphertext character different from a space, it finds the ciphertext character in column 3 of the table, and returns the column 1 plaintext letter that is on the same row.

For instance, assume k(b,1)=23, k(c,1)=11, k(c,2)=98, k(c,3)=5, k(g,1)=34, k(g,2)=56. Then the plaintext “cbcb gbgg gcb” may be encrypted as “98,23,5,23 34,23,56,34 34,11,23”.

We are currently choosing L=500, and a plaintext dictionary with q=5 plaintexts.

Your program will be scored based on two tests.

In the first test, your program will be run many times, each time on a new ciphertext, computed using the above encryption scheme and a plaintext randomly chosen from the plaintext dictionary, with a different scheduling algorithm. On the first execution, the scheduling algorithm will compute “j mod length(list)” and use this result to select the element of that position in the list. On the other executions, the scheduling algorithms will be more and more complex variations of this one.

In the second test, your program will be run a few times, each time on a new ciphertext, computed using a plaintext obtained as a space-separate sequence of words that are randomly chosen from the set of all English words (as in the attachment english\_words at the top of this page) and the above encryption scheme, with a different scheduling algorithm.

**1.2 Cryptanalysis Approach Used in the Program**

**What we know:**

* The message space is the set {< space >, a, .., z}N and the cipher text space is the set {< space >, a, .., z}M. Specifically the message m can be written as m[1],...,m[N], where each m[i] is in {<space>,a,..,z}, and the cipher text c can be written as c[1],...,c[M], where each c[i] is in {<space>,1,..,102}

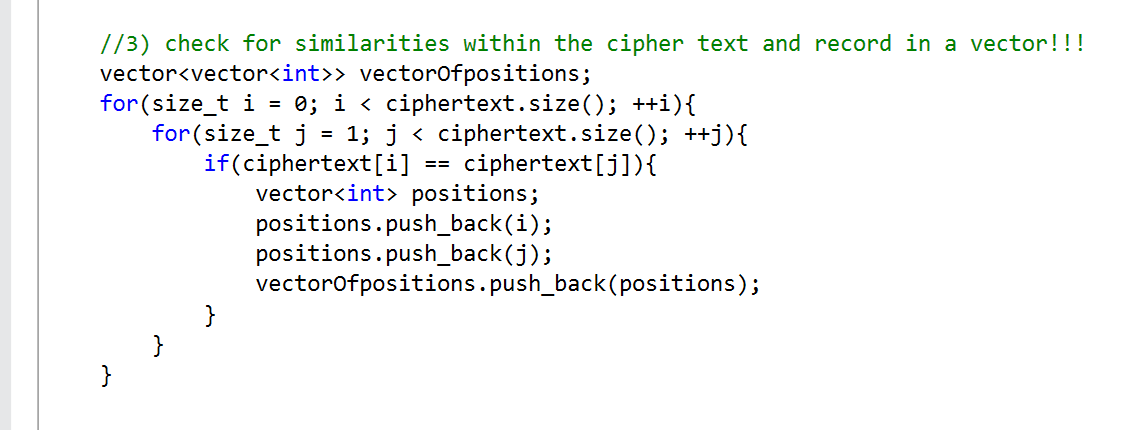
**Strategy for Dictionary 1:**

Since each letter can be mapped to a corresponding integer value or multiple integer values, the integer’s values in the cipher text can be compared against itself for repeating value. The position of the repeating values are noted, and then compared to the plain text.

For Dictionary 1, the plaintext is 500 characters. Therefore, all that is needed to be done is to do a direct comparison between the cipher text the plaintext. In other words, dict1[0] will be compared with cipher text, dict1[1] will be compared with cipher text, etc.

The comparison is done by checking the positions of similar characters in cipher text with the same positions of the plain text. If the plain text has the repeating values in similar positions then there is a chance that the chose plain text is the message that was encrypted the algorithm. If there are multiple plain texts with similar matchings, then the plain texts with the highest number of matching letters is the chose plain text is the message that was encrypted the algorithm.

The code that handles the comparison between similar letters in the cipher text can be seen here:



The code that handles the comparison between the plain text and the cipher text can be seen here:



**Strategy for Dictionary 2:**

In this part we were unsuccessful in deciphering a cipher text of random words from the dictionary. Our method of attack was to use frequency analysis on the characters. We first generated several plaintextsconsisting of random words from the dictionary and each of the plaintexts had 500 characters. We counted the number of times each character occurred in the plain text. For example, “a” occurred on average 32 times, “b” occurred 7 times, and so on for each of the 26 letters. We then divided these numbers by the respective frequencies given to us in the table.

For example, since a has eight numbers assigned to it we divided 32 by 8 and assumed each of the numbers would occur on average 4 times in the cipher text since the numbers were chosen randomly. We do the same for the other 25 letters and find the number of times each number would occur. Finally we divided each of these numbers by 500 to get their decimal value. Thus each of the eight numbers in “a” was assigned a value of 4/500 or .008.

After we completed this part we sorted the words in the dictionary by their word length and placed it in a vector. Then we read the encrypted text in and began the deciphering algorithm. Basically the idea was that we would read the first encrypted word in say, for example: 18,25,72. We would find the frequency of each of these numbers in the cipher text, which, for the sake of argument, were: 0.02, 0.05, and 0.1. We square each of these frequencies and add them together to get a number.

The next step was to search through each of the three letter words in the dictionary and take the dot product of the frequencies <.02,.05.,0.1> and the frequencies of the three letter words to get a number lets call x. So say we had the word the and the frequency of t we earlier found was .08, the frequency of h was .05, and the frequency of e was .03. We would take the dot product of <.02,.05,.01> and <.08,.05,.03> and lets say its .008. We then would find the absolute value of the difference x and .008 and keep a record of it. Using this method we would find the smallest difference and say that was the word that we were looking for.

However this method did not work out. We were only able to decipher a few words. The reason we believe this method did not work is because our sample size was too small. Perhaps 500 characters was too small for the Law of Large Numbers to work properly work and converge to the average frequencies we calculated. In the end we realized that maybe if we had more characters our solution might have worked.

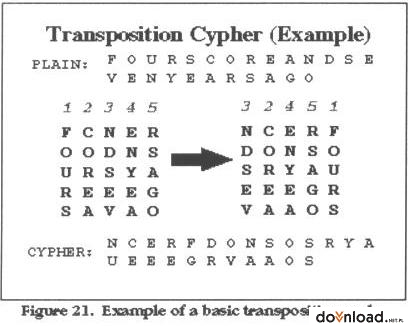
**Part 2:**

**Extra Credit: A survey on permutation ciphers**

Permutation ciphers, also known as transposition ciphers, are ciphers that, as their name suggests, permute the message as a means to encrypt. In the case of a plain text we permute the characters in such a way that it is hard to decipher the meaning. This method has been used for hundreds of years and is important in modern encryption.

For example, the rail fence cipher was a form of permutation cipher that was used by the ancient Greeks in wars. The messenger would write the message on a ribbon of a certain length, wrap it around a cylinder, and then write the letters off the cylinder creating a cipher text. The receiver would then decipher the cipher text by using a cylinder of the same size. However, the fatal flaw in this encryption is that the key space is too small and so through an exhaustive search one could find a cylinder of the right size and easily decrypt the message (Hilton).

Another popular cipher was the columnar cipher. In fact the rail fence cipher was really just a simple columnar cipher. The columnar cipher worked by choosing a column size of a certain length and writing the plain text character by character until it hit the last row which also had a size the length of the column. In that case one would continue the message on the second line and so on until the message was done. Then each column would get a number from one to the number of columns assigned randomly and from there we would create the cipher text. For example in Figure 1 seen below we have five column and five rows:



The message is: “Four scores and seven years ago” and is first written downwards along the rows before going to the next column. Then the columns are assigned random numbers from 1 to 5 (Transposition).. Finally the words would be read from left to right as shown in the figure. The columnar cipher was a very simple idea but was powerful when combined with other ciphers such as a substitution cipher. In fact during World War I and World War II double permutation ciphers like a double columnar cipher would be used to encrypt messages (Hilton).

Though these permutation ciphers were formidable against a cipher text attack they could be broken easily with a known plaintext attack. One could find a pattern in the encryption given the plain text and cipher text. One could also use frequency analysis since the frequency of the letters remains constant under the permutation. By shifting some of the characters one could find hints of the message in a regular permutation cipher such as a rail fence cipher. However later encryptions, such as the double columnar cipher, were impervious to these types of anagramming attacks.

Today we have developed computers and algorithms can analyze these permutation ciphers. While these permutation ciphers have been broken in the present they are an important part in more complex algorithms that are still used today.

**Works Cited**

Hilton, Ordway, and Laurence Dwight Smith. "Cryptography, the Science of Secret Writing." *Journal of Criminal Law and Criminology (1931-1951)* 34.6 (1945): 420. Web. 13 Mar. 2016.

"Transposition Cipher." *Crack The Codes*. Web. 13 Mar. 2016.