

**Computer Science and Engineering**

**Cryptanalysis of Permutation Ciphers**

**Applied Cryptography**

Team Members: **Ajay Shenoy**

**George Lin**

**Shearyar Shamim Khan**

**Table of Contents**

**1 Introduction | 6**

1.1 Purpose | 6

1.2 Scope | 6

1.3 Identification | 6

1.4 Document Summary | 6

1.5 System Overview | 6

1.6 Document Overview | 7

**2 Reference Documents | 7**

**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
* Helped with algorithms and decryption schemes through pair

**George Lin**

* Helped with algorithms and decryption schemes through pair

**Shearyar Shamim Khan**

* Created the project program and handled the coding aspects
* 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}

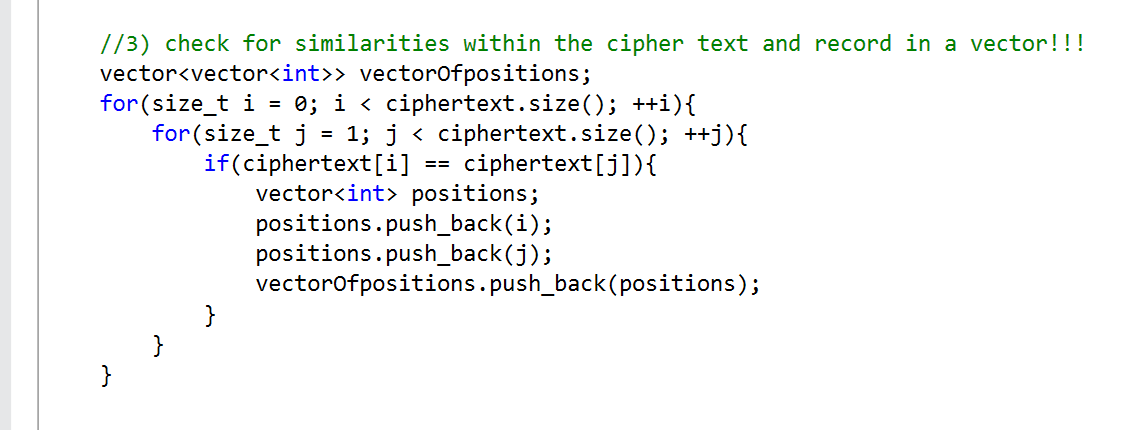
**Strategy for Dictionary 1:**

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:**