Fall 2018: Applied Cryptography Project 1

CRYPTANALYSIS OF A CLASS OF CIPHERS BASED ON {ALGORITHM METHOD USED}

Authors:
Susanna Xu
Chuhan Jin
Suyash Soumya

Professor:
Giovanni Di Crescenzo

Oct 13, 2018

TABLE OF CONTENTS

1. INTRODUCTION

1. Purpose

The purpose of this cryptanalysis project is to clearly lay out the software implementation of an algorithm that tries to decrypt an L-symbol challenge ciphertext computed using a specific cipher.

The program's goal will be to find the plaintext used to compute this ciphertext within a reasonable amount of time. It will also print on screen "Enter the ciphertext:" to obtain the ciphertext from stdin upon which it will apply some cryptanalysis strategy and output on screen "My plaintext guess is:" followed by the plaintext found.

To achieve this goal the program is allowed access to:

1. The ciphertext (to be taken as input from stdin)
2. A plaintext dictionary which contains a number q of plaintexts, each one obtained as a sequence of space-separated words from the English dictionary
3. Encryption algorithm used

The team consists of Susanna Xu, Chuhan Jin and Suyash Soumya. Collectively the team discussed the various methodologies to go about solving the task at hand. In this document we list an informal explanation of the cryptoanalysis approach used in our program followed by a more detailed explanation. See README.md for more information on running our decryption program and reference files used.

2. ENCRYPTION SCHEME

2.1 Algorithm

The plaintext is a space-separated sequence of words from the English dictionary (the sentence may not be meaningful, for sake of simplicity). The key is a map from each English alphabet (lower-case) letter (including <space>) to a list of numbers randomly chosen between 0 and 105, where the length of this list is the (rounded) letter’s frequency in English text. The ciphertext looks like a sequence of comma-separated numbers between 0 and 105, obtained as a sequence of encryptions of words, where each word is encrypted as a comma-separated list of numbers between 0 and 105, and these numbers are computed using the table below.

English letters	Average frequency	Key values (randomly chosen distinct numbers between 0 and 105)
<space>	19	k(<space>,1),…,k(<space>,19)
a	7	k(a,1),…,k(a,7)
b	1	k(b,1)
c	2	k(c,1),k(c,2)
d	4	k(d,1),…,k(d,4)
e	10	k(e,1),…,k(e,10)
f	2	…
g	2	
h	5	
i	6	
j	1	
k	1	
l	3	
m	2	
n	6	
o	6	
p	2	
q	1	
r	5	
s	5	
t	7	
u	2	
v	1	
w	2	
x	1	
y	2	
z	1	

2.2 Cipher Computation

The 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] belongs to set {(space>,a,..,z)}

- The ciphertext c can be written as c[1],...,c[L], where each c[i] belongs to set {0,..,105}. 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,…,105}, grouped in 27 lists, each list having length determined by column 2 of the table above.

- The encryption algorithm works as follows.

  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. On input a ciphertext character, it finds the ciphertext character in column 3 of the table, and returns the column 1 plaintext letter (possibly including <space>) that is on the same row.

For instance, assume k(<space>,1)=76,...,k(<space>,19)=94,

k(b,1)=23, k(c,1)=11, k(c,2)=98, k(g,1)=34, k(g,2)=56.

Then the plaintext “cbcb gbgg gcb” may be encrypted as “98,23,11,23,79,34,23,56,34,82,34,11,23”.

3. Test 1

In the first test, the program is 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 this test we will choose L=500, and a plaintext dictionary with q=5 plaintexts.

Pseudo-Code for Test 1

```
Each plaintext’s score initiated to 0
For each plaintext
        For each character in that plaintext
                Create a pattern array
                Loop through plaintext
                        If position in plaintext has character c
                                Position at pattern array is 1
                        Else
                                Position at pattern array is 0
                Loop through plaintext
                        If the position in plaintext is the character c
                                Find value of the same position in ciphertext
                                If (value not in the uniqcipher vector (vector
                                contains the number of unique cipher number
                                in certain place)
                                        add value to uniqcipher vector
                If (num of values in uniqcipher vector is less than or equal
                to the frequency)
                        increment that plaintext’s score by one
Max score initially set to 0
For each plaintext’s score
        If (plaintext score is greater than max score)
                Max score is set to plaintext score
                Encrypted plaintext is set to this plaintext
//The encrypted plaintext is the answer
```

Informal description of code for Test 1

We set all the plaintexts' score to zero. We create a pattern array for each character in each plaintext that shows whether that character exists in that position of the plaintext. We use this pattern array by getting the positions in which such a character exists and find the value in that respective position in the ciphertext. We will add all thaose values into an uniqcipher vector. Then we compare the length of the uniqcipher vector of that character in that plaintext to the frequency of that character, as long as the former is less than or equal to the latter, we will increment that plaintext's score by one. After doing that for each character in each plaintext, we can compare all the scores. The plaintext with the highest score is the answer (the plaintext that has been encrypted)

4. Test 2

In the second test, the 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 a subset of the set of all English words (specifically, a few words taken from the attachment english_words.txt) and the above encryption scheme, with a different scheduling algorithm. In this test we consider L=500.

For this test we used the provided English word textfile and reordered them in terms of the of the sum of the word's letters' frequencies and outputted that as dict.txt. This way we can test words that are lower overall (in terms of sum of the word's letters' frequencies), possibly increasing chances of finding the correct words and thus our guess key map faster.

Pseudo-Code for Test 2

```
Construct the limitation frequency table for all 27 letters(include whitespace)
Take the Input cipher[500]
//Recursively guessing the first word from highest score word of the dict.txt
Recursive(){
        for highest score word to lowest score word{

        length = plainword.length()
        Assign first cipher[length] to corresponding letters, and only save unique cipher.
        if the frequency(how many unique numbers) > the frequency provided, this guess failed, move to next word.
        if the cipher cipherlength <=0 then we might find out the correct answer, out put it.
        cipher length = cipher length -length

        Recursive();
        }
}
```

Informal description of code for Test 2

We test each word to see if it could possibly be encrypted into the given ciphertext. For each character of the word, we append the cipher value in the same position in our guess map's value's vector (plaintext char to vector of cipher value is the key to value respectively) If the length of the vector of a plaintext char is less than equal to the frequency of that value, we can continue on with this guess map and our guess plaintext string. Otherwise, we will have to clear all the values in each of the map's key's value's vector and our guess plaintext string to start all over. However, if our guessed string and guess map seem to fit the ciphertext we continue by testing other words. If we see a cipher value in the ciphertext that we added to our map, we will assume that that cipher value is the plaintext associated to it in the map. We will also use that to check and to decide which words we should check. When we use our guess map to guess the plaintext characters and it doesn't match any word in our dict.txt file, we will also need to start all over.