## Adeen Ayub

## Aphichaya Piyapinansook

## Rony Xavier

## CS6903 Project 1

Cryptanalysis based on Unit Frequency Characters in the key

This team consists of Adeen Ayub, Aphichaya Piyapinansook, and Rony Xavier. We are submitting one cryptanalysis approach as follows:

## Cryptanalysis APPROACH for test1

We learned that the method used by this encryption scheme is called a homophonic substitution cipher, where a plaintext letter can be encrypted into different amount of numbers or letters. It is stronger than Caesar cipher and mono-alphabetic cipher because frequency analysis, based on frequency of English alphabets, can’t be performed. The cipher overcomes this weakness by assigning as many numbers to a letter as its average frequency. Therefore, if a letter appears more, it can get mapped to more numbers. However, as a result of this, it also means that the letters that rarely appear will only get mapped to one number.

In addition, the pigeonhole principle also comes into play. When there are more objects than containers, then certainly at least one container have to contain more than one object. Because the message and cipher lengths (i.e. 500) is more than the total cipher numbers (i.e. 105) to which the plaintext can get mapped, some of the ciphertexts is bound to get repeated. This is how we can compare the ciphertext with each of the plaintexts and see where in the given ciphertext the repeated numbers correspond to the characters in the message.

The cryptanalysis approach we used in our program utilizes the fact that seven of the alphabets ( 'b', 'j', 'k', 'v', 'q', 'x', 'z' ) have the average frequency of one, which means that their corresponding ciphertext will be same and repeated. For example, the letter j has an average frequency of 1 and the key value would be k(j,1), for any randomly chosen key, Its ciphertext will always be identical.

We pre-process the known plaintexts to map out occurrences of all the unit frequency characters. In other words, we collected and saved all the known unit frequency character positions that help us determine which one of the five plaintexts is given as ciphertexts.

The decryption scheme will work regardless of the scheduling algorithm used in the encryption scheme. The Scheduling algorithm does not play a crucial role in this decryption scheme due to the weakness that several alphabets only have an average frequency of 1. This results in those letters being mapped to the same number.

## IMPLEMENTATION for test1

As stated above, the decryption scheme preprocesses the known plaintexts to map the occurrences of all the unit frequency characters using the method GENERATE\_UNIT\_FREQ\_CHAR\_POSITIONS.

**procedure** GENERATE\_UNIT\_FREQ\_CHAR\_POSITIONS

PLAINTEXT\_LIST 🡨 List of known plaintext strings

UNIT\_FREQUENCY\_CHARS 🡨 List of unit frequency chars { 'b', 'j', 'k', 'v', 'q', 'x', 'z' }

**for** each string *plaintext* in the vector PLAINTEXT\_LIST **do**

**for** each character *char* in vector UNIT\_FREQUENCY\_CHARS **do**

**for** each character *letter* in string *plaintext* **do**

**if** *char == letter* **do**

insert into vector *positionlist* index of *letter* in string *plaintext*

**end if**

**end for**

insert into map *unitFreqCharPositions* vector *positionlist* at key *char*

clear vector *positionlist*

**end for**

insert into vector *unitFreqCharPositionsList* map *unitFreqCharPositions*

clear map *unitFreqCharPositions*

**end for**

**return** *unitFreqCharPositionsList*

**end procedure**

The ciphertext is received from the user from STDIN as series of integers delimited by commas. The method DECRYPT\_CIPHERTEXT shown below performs the decryption and returns an index which gives the guessed plaintext or returns ‘-1’ indicating failure to decrypt.

**procedure** DECRYPT\_CIPHERTEXT

CIPHERTEXT 🡨 vector of integers that form the provided ciphertext

*unitFreqCharPositionsList* 🡨 GENERATE\_UNIT\_FREQ\_CHAR\_POSITIONS ()

**for** each map *unitFreqCharPositions* in vector *unitFreqCharPositionsList* **do**

**for** each key *char* in map *unitFreqCharPositions* **do**

vector*positions* **🡨** *unitFreqCharPositions* value at key *char*

**for** eachinteger *position*in vector *positions* **do**

insert into vector *list* values of CIPHERTEXT at index *position*

**end for**

**if** length of *list >* 1 **AND** all elements of *list* are equal

/\* return index which is the index of guessed plaintext \*/

**return** index of *unitFreqCharPositions* in *unitFreqCharPositionsList*

**end if**

clear vector *list*

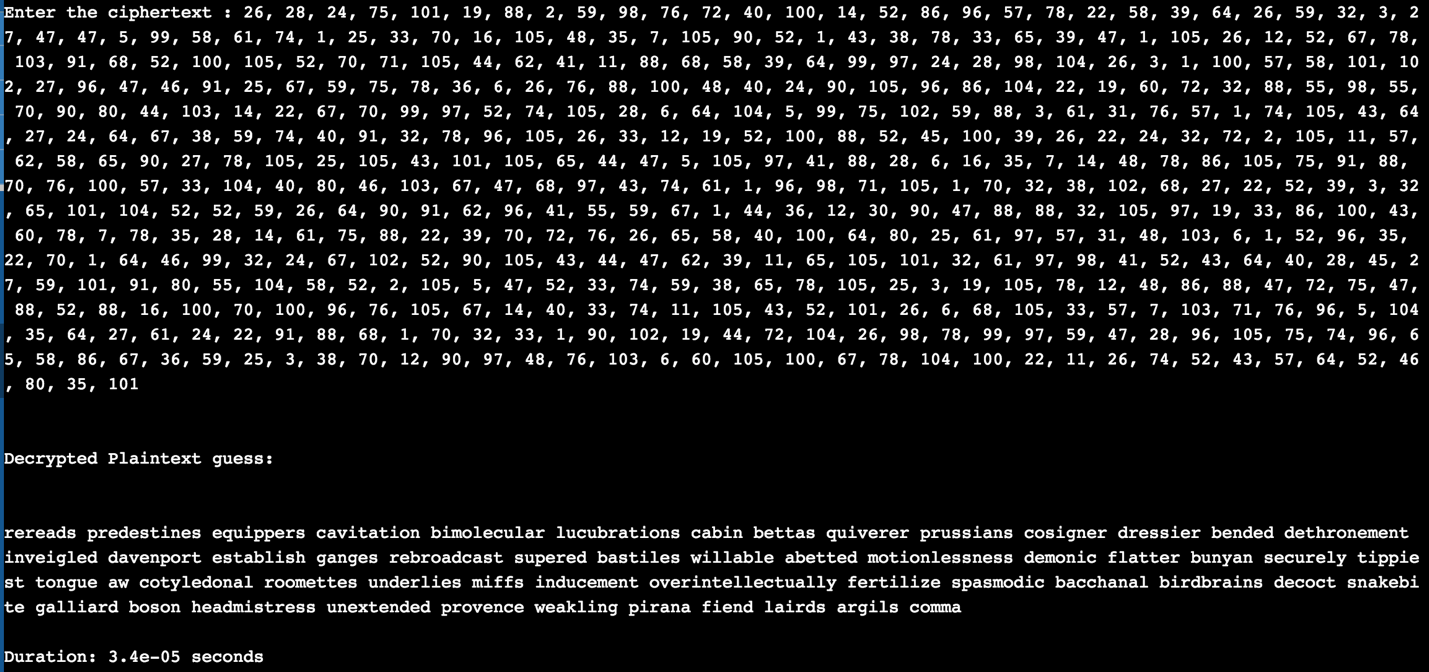
**end for**

**end for**

**return** -1 /\* indicating failure \*/

**end procedure**

## Trial run for test1:



## Conclusion

The decryption scheme could effectively decrypt ciphertexts of known plaintexts thus the encryption scheme fails the *indistinguishability* notion. The proposed scheme does not address test2.