Raj Ramrup, Andy Tang, Baochen Wang

Applied Cryptography Project 1 Report

Our solution for the project only works for test 1. Due to time constraints we were not able to get a working solution for test 2 and as a result we will not be discussing test 2 in this report. Our group members worked on roughly equal amounts of work. Andy Tang worked on the foundation of the decryption code and modified the encryption code to choose a random plain text. Baochen Wang worked on the initial encryption code and worked on test 2. Raj Ramrup made the final changes to decryption code and modified the encryption code to have a random key every time.

The cipher we are trying to decrypt has taken one important piece of information away from us. It has masked any frequency analysis we could perform on the cipher text. By replacing each character with a number from a set of random numbers equal in length to its frequency it has ensured that every number that occurs in the cipher text should appear equally likely. This is assuming that the scheduler used by encryption algorithm is random or pseudo-random.

With frequency analysis out, we are left to look at other things. An important piece of information that is not masked is position. We also know that every letter has a disjoint set of numbers from the set of integers from 0 to 105. With these two pieces of information we can decrypt test 1. We can find numbers that occur more than once and mark their position. We don't know what letter they are but since we know that every character is represented by a disjoint set those two numbers represent the same letter. We know that the same letter occurs at two set locations. If we check the plain text, which we have access to in test 1, we can find which plain text has matching characters in those same two positions. Doing this will eliminate some of the plain text from the pool. We can keep repeating this until there is only one plain text left and we have an answer.

The reason this should work is that the plain texts are different enough in that they rarely have matching characters in the same position across the plain-texts. The message length is 500 and the numbers they can use are from 1 to 105. Due to the pigeonhole principle, we are assured to have at least 4.7 repeated numbers. This should be enough to determine which plain-text is the right one.

I will now describe the decryption algorithm. We will take the cipher text in by standard input and put each number into a spot in the cipher text array. By array we mean a dynamic array. This part of this needs to be a dynamic array because we do not know how many times the number will occur while we are still iterating over the input. The cipher text is given to us as a comma separated list but we need to get just the numbers and put that into our array. We will use the function split_cipher for this. It will make an array of integers where each position holds that number of the cipher text

We will then iterate over the cipher text array looking at the number at each position. We need have the positions that each cipher text number occurs stored in an array. For this purpose we have a map called positions. Its key will be the cipher text number and its value with be an array of its positions in the cipher text. The function map_positions will create and fill the map.

Next we will iterate over the cipher text map looking at the values of every key. If the array of a key has multiple things in it then that cipher text number occurs multiple times. Then we will check all of positions against all of the valid plain texts which are stored as strings(c strings to be specific).

For example if in the map the value was an array which contained the values 34 and 57 and key was 5 then we would look as positions 34 and 57 to see if the characters match there in all the plain texts. If they do not they are eliminated. This will continue until they cipher text has been completely iterated over or until only one plain text remains in the possible solutions.

We will also be writing a short encryption program to test the decryption program that we hope will count as extra credit. We will start the counter 105 and pick a random number between 0 and 26. If that numbers list isn't full then will append that number to the list and decrease the counter. If it is full then we will re-roll. We keep doing this until the counter gets to 0. This means that each numbers list has their average frequency amount of numbers and they are all disjoint.