PROJECT 1

-ABHISHEK RAMACHANDRAN
-NIKHIL REDDY
-THOMAS ALEXANDER SELLIE-LUND

INTRODUCTION

PERMUTATION CIPHER

Permutation cipher also called the Transposition cipher is an encryption method where the cipher-text is basically a re-ordered sequence of the plain-text. The re-ordering is done per a random permutation that serves as the key.

Here is an example of a permutation cipher

Suppose Alice wants to send the plaintext message: "*He walked up and down the passage two or three times*" to Bob.

Alex and Bob agree on a random permutation of length 6 as the key:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 1 & 6 & 2 & 5 \end{pmatrix}$$

The encryption works by splitting the plaintext message into blocks of size equal to the key length: "*hewalk edupan ddownt hepass agetwo orthre etimes"* and then apply the permutation on each of the block to obtain the ciphertext "*WLEHKAUADENPONDDTWPSEHSAEWGAOTTRROEHIETESM*". (here, the 1st letter in the plaintext is the 4th letter in the cipher-text, the 2nd letter in the plaintext is the 3rd letter in the cipher-text etc.)

The decryption of the cipher-text generated using permutation cipher is an inverse function.

There are different forms of permutation ciphers such as – Rail fence cipher, latin square cipher etc.

However, permutation ciphers easily succumb to chosen-plaintext attacks and is therefore often used in combination with other methods.

The encryption scheme used in this project to produce the challenge cipher-text is a combination of permutation-substitution cipher.

HOMOPHONIC SUBSTITUTION CIPHER

In this encryption method, each letter in the plaintext can be mapped to multiple cipher substitutes. As a result of this, there may be more than one possible substitution for a given plaintext character. However, each cipher-text symbol can represent only one plaintext character, so the decryption is unique. This cipher makes cryptanalysis based on frequency analysis hard.

Here is a sample key table that represents homophonic substitution cipher:

Alphabets	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher Equivalents	24	2	5	9	8	3	18	22	25	4	28	20	17	31	16	29	23	19	6	26	14	15	21	16	27	12
	33		1		11			32												30						
	7				10																					
					13																					

The alphabet A has three possible cipher-texts: 24,33,7; the alphabet B has one possible cipher-text: 2 etc.

PROBLEM STATEMENT

This project, which requires us to decrypt a challenge cipher-text, uses the concept of homophonic substitution cipher to create the key table. In this table, each character is assigned to one or more numbers between 0-105(key space) depending on the frequency in English language. The frequency of each character is determined by some previously conducted heuristic analysis. The key space is the sum of all the frequencies. This method is also termed fractionation.

Since each character is mapped to many cipher equivalents, the encryption method deploys a variant of permutation cipher technique to decide on which cipher-text equivalent for a given plaintext is chosen for encryption. One variation of permutation cipher used in the project uses a deterministic scheduling algorithm to calculate the value of "j mod length(list)" where j corresponds to the position of the alphabet and length(list) is the number of key mappings for that character at position j. The value returned by this algorithm is used to determine which cipher-text equivalent for the alphabet is used for the encryption.

CRYPTANALYSIS APPROACHES

There are many techniques by which a given homophonic substitution/ permutation cipher can be broken. A homophonic cipher uses a one to many mapping from message space to key space as opposed to a simple substitution cipher. Hence, there may be more than one possible substitution (in our case, a number) for a given plaintext letter. However, each cipher number can represent only one plaintext letter, so the decryption is unique even though the encryption isn't.

Some of the techniques used in breaking homophonic ciphers are:

1. Exhaustive key search method: This method performs the cryptanalysis by brute forcing the given dictionary and trying to find the right encryption scheme which is used to encrypt the English letters. The key space and the message space is known. The cipher text is taken as input, and all the numbers are assigned to the letters using certain rules that try to fill up the encryption scheme and thereby, guess all the plaintext letters.

The rules for assigning the numbers to the letters may vary depending on how efficient the algorithm needs to be and how the encryption map is filled up. Initially, the assignment is done to the numbers based on the input cipher text, but when a collision occurs or if any rule of the encryption scheme is violated, then, it re assigns the cipher texts and does so, recursively to fill up the entire encryption scheme or till all the words of the dictionary are used up. Later, when the correct encryption scheme/key is found, decryption becomes very simple.

2. Hill climbing technique: In the case of such one to many mapping ciphers, the key space is very large for brute force/ exhaustive key search methods. To increase the efficiency, the hill climbing technique is used which incrementally improves the solution in every iteration. It is a heuristic technique, in which the algorithm tries to find a certain combinatorial structure using heuristics. A heuristic is an educated guess and a heuristic search uses a series of such guesses to find a reasonable solution to a given combinatorial problem. This tends to be faster than the brute force method.

Breaking such ciphers which have multiple cipher substitutes for a given plaintext can be tough if the number of homophones are high. In hill climbing technique, in addition to finding which letter maps to which number, we also determine how many numbers each plaintext can become. This is handled by having 2 layers of nested hill climbing; an outer layer to determine the number of symbols each letter maps to, and then an inner layer to determine the exact mapping.

3. Anagramming: The art of breaking permutation ciphers by sliding pieces of cipher text around and looking for sections
that look like English words, constitutes the anagramming cryptanalysis.

Eg:
R-----A-----D-----I-----U-----M-----C-----A-----M-----E

could represent a text

M-----A-----D-----A-----M-----C-----U-----R-----I-----E

Once such anagrams are found, they can reveal information about transposition patterns.

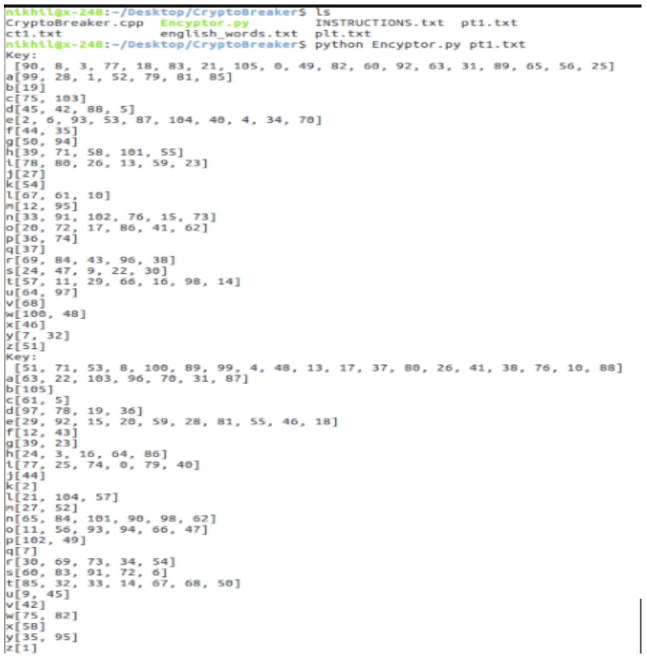
Genetic Algorithms
Another approach to cracking permutation ciphers is using a genetic algorithm that combines the anagramming with pattern matching to help get plaintext from gibberish.

PROPOSED SOLUTION

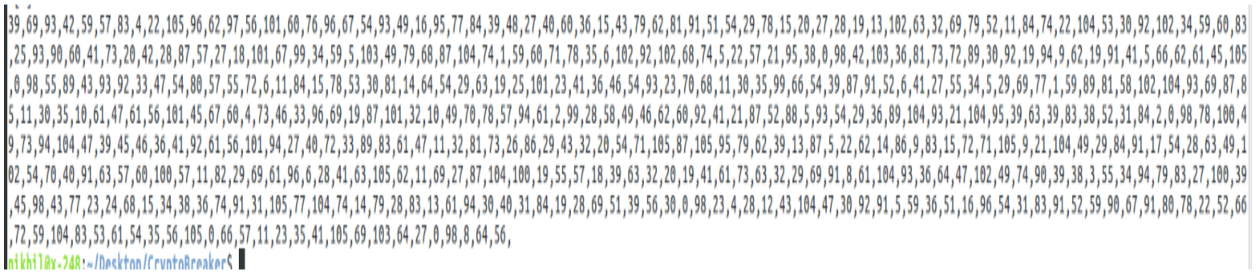
ENCRYPTION IMPLEMENTATION

For testing purposes, we developed an encryption algorithm. The encryption algorithm generated a key table by randomly assigning values from the key space [0-105] to the message space {<space>,[a-z]} per the frequency heuristics.

Using the knowledge of the scheduling algorithm provided in the problem statement, the encryption scheme computed cipher-text equivalents to randomly selected words from the plaintext dictionary.



```
nikhil@x-248:~/Desktop/CryptoBreaker$ ls
CryptoBreaker.cpp  Encyptor.py          INSTRUCTIONS.txt  pt1.txt
ct1.txt            english_words.txt  plt.txt
nikhil@x-248:~/Desktop/CryptoBreaker$ python Encyptor.py pt1.txt
Key:
 [90, 8, 3, 77, 18, 83, 21, 105, 0, 49, 82, 60, 92, 63, 31, 89, 65, 56, 25]
a[99, 28, 1, 52, 79, 81, 85]
b[19]
c[75, 103]
d[45, 42, 88, 5]
e[2, 6, 93, 53, 87, 104, 40, 4, 34, 70]
f[44, 35]
g[50, 94]
h[39, 71, 58, 101, 55]
i[78, 80, 26, 13, 59, 23]
j[27]
k[54]
l[67, 61, 10]
m[12, 95]
n[33, 91, 102, 76, 15, 73]
o[20, 72, 17, 86, 41, 62]
p[36, 74]
q[37]
r[69, 84, 43, 96, 38]
s[24, 47, 9, 22, 30]
t[57, 11, 29, 66, 16, 98, 14]
u[64, 97]
v[68]
w[100, 48]
x[46]
y[7, 32]
z[51]
Key:
 [51, 71, 53, 8, 100, 89, 99, 4, 48, 13, 17, 37, 80, 26, 41, 38, 76, 10, 88]
a[63, 22, 103, 96, 70, 31, 87]
b[105]
c[61, 5]
d[97, 78, 19, 36]
e[29, 92, 15, 20, 59, 28, 81, 55, 46, 18]
f[12, 43]
g[39, 23]
h[24, 3, 16, 64, 86]
i[77, 25, 74, 0, 79, 40]
j[44]
k[2]
l[21, 104, 57]
m[27, 52]
n[65, 84, 101, 90, 98, 62]
o[11, 56, 93, 94, 66, 47]
p[102, 49]
q[7]
r[30, 69, 73, 34, 54]
s[60, 83, 91, 72, 6]
t[85, 32, 33, 14, 67, 68, 50]
u[9, 45]
v[42]
w[75, 82]
x[58]
y[35, 95]
z[1]
```



```
39,69,93,42,59,57,83,4,22,105,96,62,97,56,101,60,76,96,67,54,93,49,16,95,77,84,39,48,27,40,60,36,15,43,79,62,81,91,51,54,29,78,15,20,27,28,19,13,102,63,32,69,79,52,11,84,74,22,104,53,30,92,102,34,59,60,83
,25,93,90,60,41,73,20,42,28,87,57,27,18,101,67,99,34,59,5,103,49,79,68,87,104,74,1,59,60,71,78,35,6,102,92,102,68,74,5,22,57,21,95,38,0,98,42,103,36,81,73,72,89,30,92,19,94,9,62,19,91,41,5,66,62,61,45,105
,0,98,55,89,43,93,92,33,47,54,80,57,55,72,6,11,84,15,78,53,30,81,14,64,54,29,63,19,25,101,23,41,36,46,54,93,23,70,68,11,30,35,99,66,54,39,87,91,52,6,41,27,55,34,5,29,69,77,1,59,89,81,58,102,104,93,69,87,8
5,11,30,35,10,61,47,61,56,101,45,67,60,4,73,46,33,96,69,19,87,101,32,10,49,70,78,57,94,61,2,99,28,58,49,46,62,60,92,41,21,87,52,88,5,93,54,29,36,89,104,93,21,104,95,39,63,39,83,38,52,31,84,2,0,98,78,100,4
9,73,94,104,47,39,45,46,36,41,92,61,56,101,94,27,40,72,33,89,83,61,47,11,32,81,73,26,86,29,43,32,20,54,71,105,87,105,95,79,62,39,13,87,5,22,62,14,86,9,83,15,72,71,105,9,21,104,49,29,84,91,17,54,28,63,49,1
02,54,70,40,91,63,57,60,100,57,11,82,29,69,61,96,6,28,41,63,105,62,11,69,27,87,104,100,19,55,57,18,39,63,32,20,19,41,61,73,63,32,29,69,91,8,61,104,93,36,64,47,102,49,74,90,39,38,3,55,34,94,79,83,27,100,39
,45,98,43,77,23,24,68,15,34,38,36,74,91,31,105,77,104,74,14,79,28,83,13,61,94,30,40,31,84,19,28,69,51,39,56,30,0,98,23,4,28,12,43,104,47,30,92,91,5,59,36,51,16,96,54,31,83,91,52,59,90,67,91,80,78,22,52,66
,72,59,104,83,53,61,54,35,56,105,0,66,57,11,23,35,41,105,69,103,64,27,0,98,8,64,56,
nikhil@x-248:~/Desktop/CryptoBreaker$
```

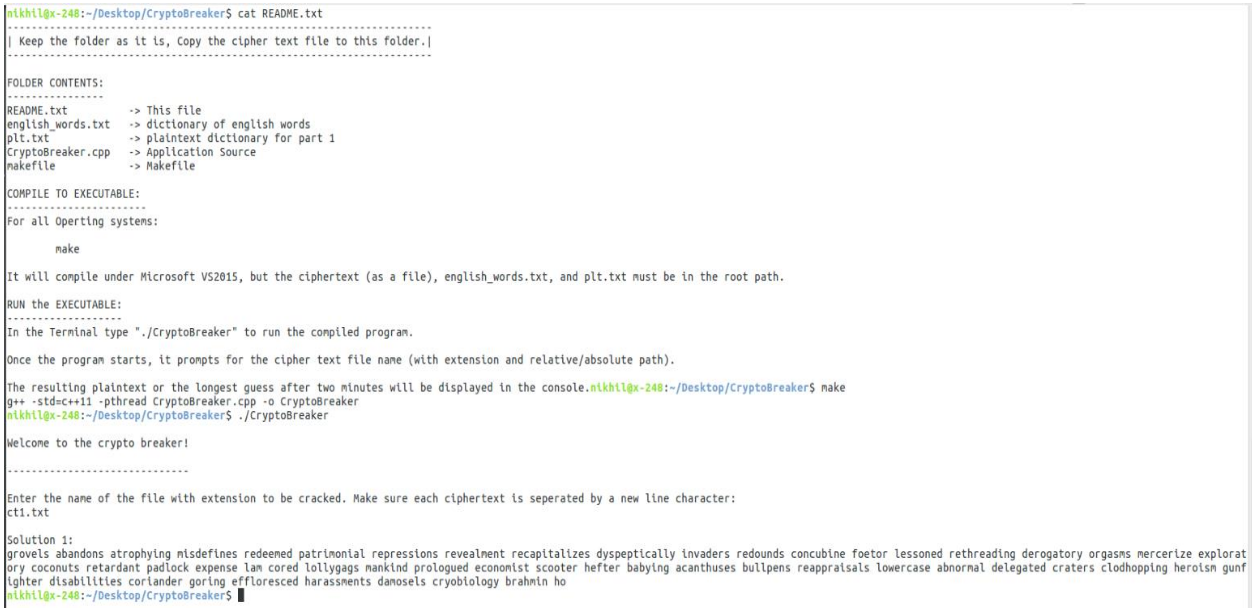
DECRYPTION IMPLEMENTATION

In our effort to decrypt the challenge cipher-text, we implement the Known Message Attack followed by an exhaustive plaintext search approach in case the Known Message attack function is unable to break the ciphertext.

In this section, we have outlined the steps that our decryption algorithm takes to output the plaintext message corresponding to the input cipher-text.

1. Get the ciphertext.
2. Load the Known Plaintexts one after the other.
3. Store the positions of occurrences of the characters 'b' in a vector. We chose 'b' and 'x' as they were the unit frequency characters that tend to appear the most in the plaintext dictionary.
4. Analyze the cipher text to check if the cipher text bits at the above stored positions are equal. Set flag to 0 if not equal at any two of these positions.
5. Repeat Steps 3 and 4 for character 'x'.
6. Return corresponding plaintext if both flags are equal to 1.
7. Repeat this for all 5 known plaintexts.
8. If no plaintext matches, we switch to the second processing section.

9. In the pre-processing phase, we load the dictionary of English words into a vector using load_dictionary().
10. Pass the dictionary along with the ciphertext is passed into "attack()".
11. After this phase, the system attempts to find the key by trying multiple plaintexts. This is accomplished by building up the plaintext word by word and checking if the resultant plaintext string is consistent with the ciphertext for every added word. This significantly reduces the number of plaintexts from a straightforward plaintext search by eliminating redundant searches, although it still takes a long time to compute, on the order of two to three days.
12. This, in theory, could be reduced significantly by the use of massive parallelization, for instance, by offloading the work to a dedicated graphics card, as the performance analytics available were from a single 5.25 gflops thread on an intel core-i5 machine, and modern high performance consumer graphics cards are capable of managing 11 Teraflops of computational horsepower.
13. To manage the time limit effectively, the system creates another thread directly after input is processed, which sleeps for two minutes. This thread changes a global value, alerting all other threads to return the "best" guess. This is then displayed to the user as the "best" guess.



```
nikhil@x-248:~/Desktop/CryptoBreaker$ cat README.txt
--------------------------------------------------------------------
| Keep the folder as it is, Copy the cipher text file to this folder.|
--------------------------------------------------------------------

FOLDER CONTENTS:
----------------
README.txt          -> This file
english_words.txt   -> dictionary of english words
plt.txt             -> plaintext dictionary for part 1
CryptoBreaker.cpp   -> Application Source
makefile            -> Makefile

COMPILE TO EXECUTABLE:
-----------------------
For all Operting systems:

        make

It will compile under Microsoft VS2015, but the ciphertext (as a file), english_words.txt, and plt.txt must be in the root path.

RUN the EXECUTABLE:
-------------------
In the Terminal type "./CryptoBreaker" to run the compiled program.

Once the program starts, it prompts for the cipher text file name (with extension and relative/absolute path).

The resulting plaintext or the longest guess after two minutes will be displayed in the console.nikhil@x-248:~/Desktop/CryptoBreaker$ make
g++ -std=c++11 -pthread CryptoBreaker.cpp -o CryptoBreaker
nikhil@x-248:~/Desktop/CryptoBreaker$ ./CryptoBreaker

Welcome to the crypto breaker!

------------------------------

Enter the name of the file with extension to be cracked. Make sure each ciphertext is seperated by a new line character:
ct1.txt

Solution 1:
grovels abandons atrophying misdefines redeemed patrimonial repressions revealment recapitalizes dyspeptically invaders redounds concubine foetor lessoned rethreading derogatory orgasms mercerize explorat
ory coconuts retardant padlock expense lam cored lollygags mankind prologued economist scooter hefter babying acanthuses bullpens reappraisals lowercase abnormal delegated craters clodhopping heroism gunf
ighter disabilities coriander goring effloresced harassments damosels cryobiology brahmin ho
nikhil@x-248:~/Desktop/CryptoBreaker$ █
```

BIBLIOGRAPHY

http://www.simonsingh.net/The_Black_Chamber/transposition.html

http://www.cs.sjsu.edu/~stamp/RUA/homophonic.pdf

http://crypto.interactive-maths.com/homophonic-substitution.html