COMPUTER SECURITY: PROJECT 1

**TEXT FILE 1**:

**PASSWORD**: Diddledidodah; **LENGTH**: 14; **DECRYPTED MESSAGE**:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">00

<!-- InstanceBegin

template="none.dwt" codeOutsideHTMLIsLocked="false" -->

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

<!-- InstanceBeginEditable name="Doc\_title" -->

<meta name="description" content="Website of the Mathematics, Statistics and Computer Science department, a part of The Helen Way Klingler College of Arts and Sciences at Marquette University">

<title>Mathematics, Statistics and Computer Science | Marquette University</title>

<style type="text/css">

**EXPLAINATION**: Used Friedman’s approach to solve this encoded stream cipher. First, we read the file into a byte array (text file is formatted in binary). Second, we looped through our guessed key length values (3 – 26). The key length values were used to shift a version of the original cipher text and directly compare the values at the indexed location to the unaltered cipher text. By doing this, we could count the number of coincidences that occurred for the current key length. The number of coincidences were returned into a dictionary that referenced the number of occurrences found for each shift length. This dictionary was then sorted from ascending to descending, leaving the top index as the max number of occurrences – this was our key length. After finding the key length we were then able to count the frequency of each letter in the position of the key lengths index (len(file) % keylength). These results were also stored inside of a dictionary and sorted to find the most frequent letters in the indexed position. After the max frequencies were found, we xor’ed the (x) amount of letters in key with the most commonly used symbol in the English dictionary “ “ (white space). From this point we used the ord() ascii keys and repeatedly xor’ed the encrypted cipher with the repeated key to obtain the original plain text.

**TEXT FILE 2:**

**PASSWORD:** Peaches; **LENGTH:** 7; **DECRYPTED MESSAGE:**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<!-- InstanceBegin

template="none.dwt" codeOutsideHTMLIsLocked="false" -->

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

<!-- InstanceBeginEditable name="Doc\_title" -->

<meta name="description" content="Website of the Mathematics, Statistics and Computer Science department, a part of The Helen Way Klingler College of Arts and Sciences at Marquette University">

<title>Mathematics, Statistics and Computer Science | Marquette University</title>

**EXPLAINATION**: Used Freidman’s approach to solve this encoded stream cipher. First, we read the file into a byte array (text file is formatted in binary). Second, we looped through our guessed key length values (3 – 26). The key length values were used to shift a version of the original cipher text and directly compare the values at the indexed location to the unaltered cipher text. By doing this, we could count the number of coincidences that occurred for the current key length. The number of coincidences were returned into a dictionary that referenced the number of occurrences found for each shift length. This dictionary was then sorted from ascending to descending, leaving the top index as the max number of occurrences – this was our key length. After finding the key length we were then able to count the frequency of each letter in the position of the key lengths index (len(file) % keylength). These results were also stored inside of a dictionary and sorted to find the most frequent letters in the indexed position. After the max frequencies were found, we xor’ed the (x) amount of letters in key with the most commonly used symbol in the English dictionary “ “ (white space). From this point we used the ord() ascii keys and repeatedly xor’ed the encrypted cipher with the repeated key to obtain the original plain text.

Due to inconsistences in the coincidence cases, Freidman’s method did not decipher the message 100% accurately. After running the python code in ‘text file 2’, the original key length was 14 (twice as long), and partially accurate { P e a ( h e s }. If we wanted to develop a more accurate decoder, one that needed less human intervention, we might try to implement a gini purity calculation to correct the output.

**TEXT FILE 3:**

**PASSWORD:** Shifted (+9) Using the capital letter “I” **DECRYPTED MESSAGE:**

AMENDMENTICONGRESSSHALLMAKENOLAWRESPECTINGANESTABLISHMENTOFRELIGIONORPROHIBITINGTHEFREEEXERCISETHERE

**EXPLAINATION:** The Caesar Cipher was easy to solve for. After opening the text file, we read each letter into an array list. This array list was then used to test each decryption test. The test was to shift each letter a fixed number of times, print the newly decrypted text, and determine if the plain text made sense. We cycled through each letter of the alphabet to find the correct shift length, starting with ‘A’ and finishing with ‘Z’. After looping and printing the results, we were able to find the correct letter that decrypted the cipher text.

**TEXT FILE 4: ……………………. USES RANDOMLY GENERATED STREAM TO ENCODE PLAIN TEXT FILE**

**TEXT FILE 5:**

**PASSWORD:** Shifted (+2) Using the capital letter “B” **DECRYPTED MESSAGE:**

AMANSTOODUPONARAILROADBRIDGEINNORTHERNALABAMALOOKINGDOWNINTOTHESWIFTWATERTWENTYFEETBELOWTHEMANSHANDS

**EXPLAINATION:** The Caesar Cipher was easy to solve for. After opening the text file, we read each letter into an array list. This array list was then used to test each decryption test. The test was to shift each letter a fixed number of times, print the newly decrypted text, and determine if the plain text made sense. We cycled through each letter of the alphabet to find the correct shift length, starting with ‘A’ and finishing with ‘Z’. After looping and printing the results, we were able to find the correct letter that decrypted the cipher text.

**­­**