Computer Networks and Security

88-447 Summer

Dr. H. Wu

Lab 01

Jason Choquette

104337378

Assigned ciphertext:

PYNVZVATNALZREVGSBEGUNGCNFGFREIVPRNAQJVGUZBERZRAGVBABSSENAPRBSORYTVHZBSTYBELBSU
BABHENAQBSFHPUXVAQERQGUVATFGURLUNQRZOENPRQRNPUBGUREURNEGVYLNAQGURPBAIREFNG
VBAUNQRAQRQNFGBJUNGVGUNQNYYORRANOBHGYVRHGRANAGQHOBFPJNFFGVYYVAGURQNEXOHG
GBUVZUNQORRAQRYRTNGRQGURQHGLBSFRRVATBSSZCBVEBGOLGURGNHEHFRKCERFFNAQURJNFPNEE
LVATVGBHGJVGUNYYGURMRNYNAQNEQBHEORSVGGVATNLBHATBSSVPREJVGUNCEBZVFVATPNERREN
URNQBSUVZGBQNLVFFHAQNLFNVQYVRHGRANAGQHOBFPGBZBEEBJZBAQNLRIRAVATLBHJVYYORVAFG
NZOBHYVGJNFABGGURSVEFGGVZRURUNQZNQRGUVFBOFREINGV

Plaintext:

CLAIMINGANYMERITFORTHATPASTSERVICEANDWITHMOREMENTIONOFFRANCEOFBELGIUMOFGLORYO
FHONOURANDOFSUCHKINDREDTHINGSTHEYHADEMBRACEDEACHOTHERHEARTILYANDTHECONVERSATI
ONHADENDEDASTOWHATITHADALLBEENABOUTLIEUTENANTDUBOSCWASSTILLINTHEDARKBUTTOHIMH
ADBEENDELEGATEDTHEDUTYOFSEEINGOFFMPOIROTBYTHETAURUSEXPRESSANDHEWASCARRYINGITOU
TWITHALLTHEZEALANDARDOURBEFITTINGAYOUNGOFFICERWITHAPROMISINGCAREERAHEADOFHIMTO
DAYISSUNDAYSAIDLIEUTENANTDUBOSCTOMORROWMONDAYEVENINGYOUWILLBEINSTAMBOULITWAS
NOTTHEFIRSTTIMEHEHADMADETHISOBSERVATI

Decryption key:

13

Inner products:

W . A_0	21.3940500
W . A_1	20.3372900
W . A_2	20.7106100
W . A_3	18.9292700
W . A_4	17.4564800
W . A_5	17.4692300
W . A_6	19.3691800
W . A_7	18.0969300
W . A_8	16.8883000
W . A_9	20.8375100
W . A_{10}	16.9824900
W . A_{11}	15.5040600
W . A_{12}	20.1253600
W . A_{13}	31.9370300
W . A_{14}	19.7760400
W . A_{15}	15.5997800
W . A_{16}	17.2887400
W . A_{17}	21.7488200
W . A_{18}	17.3868500
W . A_{19}	18.8310700
W . A_{20}	19.6774100
W . A_{21}	16.3495500
W . A_{22}	16.5221000
W . A_{23}	19.3944000
W . A_{24}	22.5239200
W . A_{25}	19.0085300

Source code:

```
/*
PROJECT         : Lab 1
FILE            : main.c
AUTHOR          : Jason Choquette, ID 104 337 378
LAST MODIFIED   : May 20, 2017
DESCRIPTION     : This program decrypts a given ciphertext using a simple
                  shift cipher algorithm and frequency analysis (english alphabet).
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>


#define ALPHABET_LENGTH 26
double A[ALPHABET_LENGTH]; // letter frequencies

// function prototypes
char     encrypt(char, int);
char   * decrypt(char*, int);
double * innerProduct(int[], int *);


int main()
{
    // english alphabet letter frequncies
    A[0] = 0.08167;
    A[1] = 0.01492;
    A[2] = 0.02782;
    A[3] = 0.04253;
    A[4] = 0.12702;
    A[5] = 0.02228;
    A[6] = 0.02015;
    A[7] = 0.06094;
    A[8] = 0.06996;
    A[9] = 0.00153;
    A[10] = 0.00772;
    A[11] = 0.04025;
    A[12] = 0.02406;
    A[13] = 0.06749;
    A[14] = 0.07507;
    A[15] = 0.01929;
    A[16] = 0.00095;
    A[17] = 0.05987;
    A[18] = 0.06327;
    A[19] = 0.09056;
    A[20] = 0.02758;
    A[21] = 0.00978;
    A[22] = 0.02360;
    A[23] = 0.00150;
    A[24] = 0.01974;
    A[25] = 0.00074;

    // given text to decrypt
    char * ciphertext = 
      "PYNVZVATNALZREVGSBEGUNGCNFGFREIVPRNAQJVGUZBERZRAGVBABSSENAPRBSORYTVHZBSTYB
      ELBSUBABHENAQBSFHPUXVAQERQGUVATFGURLUNQRZOENPRQRNPUBGUREURNEGVYLNAQGURPBAIR
      EFNGVBAUNQRAQRQNFGBJUNGVGUNQNYYORRANOBHGYVRHGRANAGQHOBFPJNFFGVYYVAGURQNEXOH
      GGBUVZUNQORRAQRYRTNGRQGURQHGLBSFRRVATBSSZCBVEBGOLGURGNHEHFRKCERFFNAQURJNFPN
      EELVATVGBHGJVGUNYYGURMRNYNAQNEQBHEORSVGGVATNLBHATBSSVPREJVGUNCEBZVFVATPNERR
      ENURNQBSUVZGBQNLVFFHAQNLFNVQYVRHGRANAGQHOBFPGBZBEEBJZBAQNLRIRAVATLBHJVYYORV
      AFGNZOBHYVGJNFABGGURSVEFGGVZRURUNQZNQRGUVFBOFREINGV";

    int ciphertext_length = strlen(ciphertext);
    int W[ALPHABET_LENGTH] = { 0 };
    int key = 0;

    // count occurences
    // Note 'A' = 65 in ascii code. So to put 'A' in position 0, subtract 65.
    for (int i = 0; i < ciphertext_length; i++)
        W[(int)ciphertext[i] - 65] += 1; // using -65 in order to fit ascii 
          characters in array positions 0-25. i.e., 'L' = 76 in ascii. W[76] 
          woild throw error...

    // compute inner product W . Ai, and find the key
    double * inner_product = innerProduct(W, &key);

    printf("%s\n\n\n", decrypt(ciphertext, key));

    getchar();
    return 0;
}





/*******************************************************************************
FUNCTION        : encrypt

DESCRIPTION     : This function encrypts a character based on a key as the 
                  parameter.

INPUT           : Type          : char
                : Description   : The character to encrypt.

                : Type          : int
                : Description   : The shift key.

OUTPUT          : Type          : char
                : Description   : The key-shifted chracter
*******************************************************************************/
```

```c
char encrypt(char ch, int key)
{
    if (!isalpha(ch)) return ch;
    char offset = isupper(ch) ? 'A' : 'a';
    return (char)((((ch + key) - offset) % 26) + offset); // shift cipher
}





/*******************************************************************************
FUNCTION        : decrypt

DESCRIPTION     : This function decrypts each character of the ciphertext using the
                  given key parameter.

INPUT           : Type          : char *
                : Description   : The ciphertext.

                : Type          : int
                : Description   : The shift key.

OUTPUT          : Type          : char *
                : Description   : The decrypted text
*******************************************************************************/
char * decrypt(char * text, int key)
{
    int text_length = strlen(text);
    char * plaintext = (char*)malloc(text_length + 1);

    for (int i = 0; i < text_length; i++)
        plaintext[i] = encrypt(text[i], key);

    plaintext[text_length] = '\0'; // add null termination character

    return plaintext;
}





/*******************************************************************************
FUNCTION        : innerProduct

DESCRIPTION     : This function uses the frequencies of letters expected
                  in an english message that has been Caeser-shifted i
                  letters to the left by a 26-dimensional vector, A.

                  One of these vectors should agree fairly closely with the
                  frequencies of letters we see in our ciphertext.
                  Which vector that is tells us the shift amount for our sampling,
                  and the first letter of our keyword.

                  To find the vector in the previous list above that most closely
                  matches the vector u, we recall that the dot product of two
                  vectors is connected to the angle θ between those two vectors
                  in the following way:

                  W.A=|W||A|cosθ

                  If we want to find the two vectors W and Ai that most closely
                  match, we want to find the two vectors with the smallest
                  angle between them.

                  Noting that smaller angles produce larger cosine values and
                  also noting that the magnitude of the denominator is the same
                  for every vi as the same 26 numbers are involved each time
                  (just in different orders), we can simply seek the two vectors W
                  and Ai whose dot product is largest.


INPUT           : Type          : int[]
                : Description   : The letter frequencies of the ciphertext.

                : Type          : int *
                : Description   : A reference to the encryption key.

OUTPUT          : Type          : double *
                : Description   : The array of innerproducts.
***************************************************************************/
double * innerProduct(int W[], int * key)
{
    double inner_product[ALPHABET_LENGTH] = { 0 };
    double sum = 0;
    int j;


    for (int i = 0; i < ALPHABET_LENGTH; i++)
    {
        for (j = 0; j < ALPHABET_LENGTH; j++)
            sum += W[j] * A[(j + i) % ALPHABET_LENGTH]; // shift the frequency array

        inner_product[i] = sum;

        // find the largest innerproduct. This will be the key.
        if (inner_product[*key] < inner_product[i]) *key = i;

        // reset counter and sum
```

```c
        j = 0;
        sum = 0;
    }

    return inner_product;
}


```