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:

$W \cdot A_0$	0.0427881
$W \cdot A_1$	0.0406746
$W \cdot A_2$	0.0414212
$W \cdot A_3$	0.0378585
$W \cdot A_4$	0.0349130
$W \cdot A_5$	0.0349385
$W \cdot A_6$	0.0387384
$W \cdot A_7$	0.0361939
$W \cdot A_8$	0.0337766
$W \cdot A_9$	0.0416750
$W \cdot A_{10}$	0.0339650
$W \cdot A_{11}$	0.0310081
$W \cdot A_{12}$	0.0402507
$W \cdot A_{13}$	0.0638741
$W \cdot A_{14}$	0.0395521
$W \cdot A_{15}$	0.0311996
$W \cdot A_{16}$	0.0345775
$W \cdot A_{17}$	0.0434976
$W \cdot A_{18}$	0.0347737
$W \cdot A_{19}$	0.0376621
$W \cdot A_{20}$	0.0393548
$W \cdot A_{21}$	0.0326991
$W \cdot A_{22}$	0.0330442
$W \cdot A_{23}$	0.0387888
$W \cdot A_{24}$	0.0450478
$W \cdot A_{25}$	0.0380171

Source code:

```c
/*
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 ciphertext_length;

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;
```

```
52      A[24] = 0.01974;
53      A[25] = 0.00074;
54
55      // given text to decrypt
56      char * ciphertext =
          "PYNVZVATNALZREVGSBEGUNGCNFGFREIVPRNAQJVGUZBERZRAGVBABSSENAPRBSORYTVHZBSTYB
          ELBSUBABHENAQBSFHPUXVAQERQGUVATFGURLUNQRZOENPRQRNPUBGUREURNEGVYLNAQGURPBAIR
          EFNGVBAUNQRAQRQNFGBJUNGVGUNQNYYORRANOBHGYVRHGRANAGQHOBFPJNFFGVYYVAGURQNEXOH
          GGBUVZUNQORRAQRYRTNGRQGURQHGLBSFRRVATBSSZCBVEBGOLGURGNHEHFRKCERFFNAQURJNFPN
          EELVATVGBHGJVGUNYYGURMRNYNAQNEQBHEORSVGGVATNLBHATBSSVPREJVGUNCEBZVFVATPNERR
          ENURNQBSUVZGBQNLVFFHAQNLFNVQYVRHGRANAGQHOBFPGBZBEEBJZBAQNLRIRAVATLBHJVYYORV
          AFGNZOBHYVGJNFABGGURSVEFGGVZRURUNQZNQRGUVFBOFREINGV";
57
58      ciphertext_length = strlen(ciphertext);
59      int W[ALPHABET_LENGTH] = { 0 };
60      int key = 0;
61
62      printf("\n\nCiphertext: \n\n%s\n\n", ciphertext);
63
64      // count occurences
65      // Note 'A' = 65 in ascii code. So to put 'A' in position 0, subtract 65.
66      for (int i = 0; i < ciphertext_length; i++)
67          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...
68
69      // compute inner product W . Ai, and find the key
70      double * inner_product = innerProduct(W, &key);
71
72      printf("\n\nPlaintext:\n\n%s\n\n\n", decrypt(ciphertext, key));
73
74      getchar();
75      return 0;
76  }
77
78
79
80
81
82  /*****************************************************************************
83  FUNCTION        : encrypt
84
85  DESCRIPTION     : This function encrypts a character based on a key as the
86                    parameter.
87
88  INPUT           : Type          : char
89                  : Description   : The character to encrypt.
90
91                  : Type          : int
92                  : Description   : The shift key.
93
94  OUTPUT          : Type          : char
```

```
95                  : Description   : The key-shifted chracter
96  ********************************************************************************/
97  char encrypt(char ch, int key)
98  {
99      if (!isalpha(ch)) return ch;
100     char offset = isupper(ch) ? 'A' : 'a';
101     return (char)((((ch + key) - offset) % 26) + offset); // shift cipher
102 }
103
104
105
106
107
108 /********************************************************************************
109 FUNCTION        : decrypt
110
111 DESCRIPTION     : This function decrypts each character of the ciphertext using
      the
112                   given key parameter.
113
114 INPUT           : Type          : char *
115                 : Description   : The ciphertext.
116
117                 : Type          : int
118                 : Description   : The shift key.
119
120 OUTPUT          : Type          : char *
121                 : Description   : The decrypted text
122 ********************************************************************************/
123 char * decrypt(char * text, int key)
124 {
125     int text_length = strlen(text);
126     char * plaintext = (char*)malloc(text_length + 1);
127
128     for (int i = 0; i < text_length; i++)
129         plaintext[i] = encrypt(text[i], key);
130
131     plaintext[text_length] = '\0'; // add null termination character
132
133     return plaintext;
134 }
135
136
137
138
139
140 /********************************************************************************
141 FUNCTION        : innerProduct
142
143 DESCRIPTION     : This function uses the frequencies of letters expected
144                   in an english message that has been Caeser-shifted i
145                   letters to the left by a 26-dimensional vector, A.
```

```
146
147                     One of these vectors should agree fairly closely with the
148                     frequencies of letters we see in our ciphertext.
149                     Which vector that is tells us the shift amount for our
                          sampling,
150                     and the first letter of our keyword.
151
152                     To find the vector in the previous list above that most closely
153                     matches the vector u, we recall that the dot product of two
154                     vectors is connected to the angle θ between those two vectors
155                     in the following way:
156
157                     W.A=|W||A|cosθ
158
159                     If we want to find the two vectors W and Ai that most closely
160                     match, we want to find the two vectors with the smallest
161                     angle between them.
162
163                     Noting that smaller angles produce larger cosine values and
164                     also noting that the magnitude of the denominator is the same
165                     for every vi as the same 26 numbers are involved each time
166                     (just in different orders), we can simply seek the two vectors
                          W
167                     and Ai whose dot product is largest.
168
169
170 INPUT              : Type          : int[]
171                     : Description   : The letter frequencies of the ciphertext.
172
173                     : Type          : int *
174                     : Description   : A reference to the encryption key.
175
176 OUTPUT             : Type          : double *
177                     : Description   : The array of innerproducts.
178 ****************************************************************************/
179 double * innerProduct(int W[], int * key)
180 {
181     double inner_product[ALPHABET_LENGTH] = { 0 };
182     double sum = 0;
183     int j;
184
185
186     for (int i = 0; i < ALPHABET_LENGTH; i++)
187     {
188         for (j = 0; j < ALPHABET_LENGTH; j++)
189             sum += W[j] * A[(j + i) % ALPHABET_LENGTH]; // shift the frequency
                    array
190
191         inner_product[i] = sum/ ciphertext_length;
192
193         // find the largest innerproduct. This will be the key.
194         if (inner_product[*key] < inner_product[i]) *key = i;
```

```
195
196         // reset counter and sum
197         j = 0;
198         sum = 0;
199     }
200
201     printf("Inner products: \n\n");
202     for (size_t i = 0; i < 26; i++)
203         printf("W[%d]  =  %f\n",i+1, inner_product[i]);
204
205     return inner_product;
206 }
207
208
```