Assignment 2

cpe 456 Fall 2012

A sekret ceases tew be a sekret if it iz once confided—it iz like a dollar bill, once broken, it iz never a dollar again.
— Josh Billings: His Sayings (1865)

Due by 11:59:59pm, Friday, Oct 26th This assignment is to be done individually.

Background: Attacking Vigenère

In order to attack the Vigenère cipher, it is necessary to disentangle the various alphabets, and to do this it is necessary to determine the key length.

Once the key length has been determined it is possible to attack individual alphabets using frequency analysis. Once an alphabet is known, letter frequency and n-gram frequency can be used to attack neighboring alphabets.

Index of Coincidence

The index of coincidence[1] is defined to be the probability that two letters chosen at random from a given ciphertext are the same. If F_c is the frequency of cipher character c—the number of them encountered in the ciphertext—and N is the total number of ciphertext characters, the index of coincidence can be calculated as:

$$IC = \frac{1}{N(N-1)} \sum_{i=0}^{25} F_i(F_i - 1)$$

The expected value of IC for a cipher of length N with key length d generated from English plaintext can be computed by:

$$\left(\frac{1}{d}\right) \frac{N-d}{N-1}(0.066) + \left(\frac{d-1}{d}\right) \frac{N}{N-1}(0.038)$$

Kasiski

Kasiski determined that repeated substrings in the ciphertext are likely to result from places where the same plaintext lined up at the same place in the key. (Otherwise, such coincidences are highly unlikely.) If this is true, the distance between repeated substrings will be a multiple of the key length.

To determine possible key lengths from repeated substrings, take all the distances you find between repeats, and look for common divisors. For example, if two strings appear 12 characters apart, key lengths of 1, 2, 3, 4, 6, and 12 are possible, because the prime factors of 12 are 1, 2, 2, and 3. If another pair of strings appears 16 characters apart (factors 1, 2, 2, 2, and 2) we know that the key length must be either 1, 2, or 4.

This is not foolproof. If there is a repeated substring in the key, there can be false positives, but it's a place to start.

[1] See .

Programs: `kasiski`, `ftable` and `ic`

This assignment requires you to write three more programs, this time to assist in cracking the Vigenère cipher. Once you have written the programs, you will use them (and your brain) to attempt to crack a Vigenère-enciphered text.

The only program of any significant complexity here is `kasiski`.

Cryptanalysis: `kasiski`

- As command line options, `kasiski` optional input and output filenames. If either filename is not given, stdin or stdout is used (as appropriate). If the "`-m` *num*" option is present, `kasiski` should only report substrings with a minimum length of *num* characters. (*num* defaults to 3). You may implement a "`-v`" option for verbosity if you like.

```
usage:  kasiski [ -v ] [ -m length ] [ infile [ outfile ] ]
```

- `kasiski` only finds **non-overlapping** repeats, since overlapping repeats are guaranteed to be red herrings.
- The string comparison is done only on alphabetic characters (A–Z).
- Non alphabetic characters are simply ignored. (Completely. That is they aren't counted in distances, either).
- The string comparison is not case sensitive.
- You may assume that the ciphertext to be analyzed is of "reasonable" size. That is, hundreds or thousands of characters would be possible, but not millions, and the ciphertext can be held entirely in memory. (This does **not** mean that you can place an arbitrary limit on it, however.)
- efficiency: just don't do anything dumb (e.g. if you didn't find any matching substrings of length 10, you're not going to find any of length 11, either.)
- Output:
  - The output will be preceded by a header as shown below
  - Each line contains:
    - length of the matched string,
    - the number of times that string was found,
    - the substring itself, and
    - the list of starting locations for that string. For every occurrence other than the first one, the distance from the previous one in parentheses.
  - Lines are sorted by length, then by count, then alphabetically
  - See the example below.

Cryptanalysis: `ic`

Write a quick program `ic` that computes the expected value of the index of coincidence for a given ciphertext length and given key lengths:

Details:

- As command line options, `ic` takes:
  - a mandatory message length, N
  - one or more key lengths, l for which to compute the index.

```
usage:  ic N l [ l2 [...]  ]
```

- Output: a table of the distances and expected values for the index of coincidence.
  - The table is preceded by a header line including the message length (as shown below)
  - Each line gives the key length, right justified in a field of width four, and the expected Index of Coincidence to four places of precision.

Cryptanalysis: modified `ftable`

Extend your ftable program from Asgn1 to compute the Index of Coincidence in addition to the other output. The only difference in the specification below is the last item of the output.

Details:

- As command line options, `ftable` takes:
  - an optional "`-v`" flag to increase verbosity;
  - an optional "`-s` n" option to cause `ftable` to skip the first n characters of the input (defaults to 0);
  - an optional "`-p` n" (period) option to cause `ftable` to only count every nth character (defaults to 1); and
  - optional input and output filenames.

  If filename(s) are not given, stdin or stdout are used (as appropriate).

```
usage:  ftable [ -v ] [ -s num ] [ -p num ] [ infile [ outfile ] ]
```

- Only frequencies for alphabetic characters ("A–Z") are computed.
- All other characters are ignored. (i.e., they do not count for skip or period, either.)
- Lowercase letters are mapped to uppercase.
- When complete, the output will be:
  - The total number of characters tabulated, reported as "Total chars: n", where n is the number,
  - A line for each letter of the alphabet, containing the following space-separated fields:
    - The character, followed by a colon,
    - The number of those characters seen, right justified in a field of 9 characters,
    - The percentage of the total characters, aligned in parentheses to two digits of precision,
    - A histogram containing one asterisk per percentage point or fraction thereof (the integer ceiling), and
  - A blank line, followed by the index of coincidence reported as "Index of Coincidence: n", where n is the number expressed with 4 digits of precision.

Cryptanalysis: Using the tools

On the CSL machines, in the directory `~pn-cs456/Given/Asgn2/Inputs` will be individual input files for each student in the class, named *logname*`.easy.in` and *logname*`.hard.in`, where *logname* is your username. Use `kasiski`, `ftable`, and `ic` to work out a probable key length, then, using frequencies and what you know about the relative likelihood of various digram, trigram, and word-length combinations, work out keys for your individual inputs.

As the names suggest, one message should be easier than the other.

There is an example cryptanalysis on page 7.

Feel free to discuss cryptanalytic *technique* with anybody, but do your analysis yourself.

Tricks and Tools

Library help

int fseek(3)	Stdio version of `lseek(2)`. Useful for repositioning streams.
`double ceil(3)`	Math library function for computing ceilings. Remember to link with the math library (`-lm`)
`int printf(3)`	Useful for creating formatted output.

Table 1: Some potentially useful library functions

Some potentialy useful functions are given in table 1.

0.0.1 Deriving Keys

Sometimes it's useful to be able to figure out the key that will transform a given plaintext to a given ciphertext. The way to do this is to use the enciphering tool, `vig`, backwards. That is, if we want "eee" to map to "zrq", do:

```
% vig -d eee
zrq
VNM
%
```

And we see it's true

```
% vig vnm
eee
ZRQ
%
```

Language

You can use C, C++, or Java, so long as:

1. The Makefile (below) builds the program appropriately, and
2. If you choose Java, you include scripts named `ftable`, `kasiski` and `ic` that run your programs with the appropriate arguments when called by those names. (I don't want to have to guess whether to run "`kasiski`" or "`java kasiski`".)

Coding Standards and Make

See the pages on coding standards and make on the cpe 456 class web page.

What to turn in

Submit via `handin` in the CSL to the `asgn2` directory of the `pn-cs456` account:

- your well-documented source files.
- A makefile (called `Makefile`) that will build your programs with "`make all`" or just "`make`".
- Two files, named `Key.easy` and `Key.hard`, containing the keys to your personal test inputs. These keys should decrypt the test input file when given the commands "vig -d *key.easy logname*.in.easy" and "vig -d *key.hard logname*.in.hard"
- A README file that contains:
  - Your name.
  - Any special instructions for running your program.
  - Any other thing you want me to know while I am grading it.

  The README file should be **plain text,** i.e, **not a Word document**, and should be named "README", all capitals with no extension.

Sample runs

Below are some sample runs of `kasiski`, `ic` and `ftable`.

I will also place executable versions on the CSL machines in `~pn-cs456/demos` so you can run it yourself.

```
% ./kasiski README
Length   Count  Word    Location (distance)
======   =====  ====    =========
     4       2  EVER    46 53 (7)
     3       2   ERT    35 48 (13)
     3       2   EVE    46 53 (7)
     3       2   ION    28 58 (30)
     3       2   THE    33 41 (8)
     3       2   VER    47 54 (7)
% ftable README
Total chars: 61
A:         4 (  6.56%) *******
B:         1 (  1.64%) **
C:         2 (  3.28%) ****
D:         0 (  0.00%)
E:        10 ( 16.39%) ****************
F:         1 (  1.64%) **
G:         0 (  0.00%)
H:         4 (  6.56%) *******
I:         6 (  9.84%) **********
J:         0 (  0.00%)
K:         0 (  0.00%)
L:         2 (  3.28%) ****
```

```
M:         2 (  3.28%) ****
N:         4 (  6.56%) *******
O:         3 (  4.92%) *****
P:         1 (  1.64%) **
Q:         0 (  0.00%)
R:         7 ( 11.48%) ***********
S:         3 (  4.92%) *****
T:         8 ( 13.11%) **************
U:         1 (  1.64%) **
V:         2 (  3.28%) ****
W:         0 (  0.00%)
X:         0 (  0.00%)
Y:         0 (  0.00%)
Z:         0 (  0.00%)

Index of Coincidence: 0.0749
% ic 1000 1 2 3 4 5 6 7 8 9 10 1000
Key  Expected IC (N=1000)
---- ------------------
   1 0.0660
   2 0.0520
   3 0.0473
   4 0.0450
   5 0.0436
   6 0.0426
   7 0.0420
   8 0.0415
   9 0.0411
  10 0.0408
1000 0.0380
```

Example Cryptanalysis

This is a demonstration of the process of cryptanalyzing a ciphertext enciphered with Vigenère using the tools developed in this assignment. The given ciphertext is:

```
CHREEVOAHMAERATBIAXXWTNXBEEOPHBSBQMQEQERBW
RVXUOAKXAOSXXWEAHBWGJMMQMNKGRFVGXWTRZXWIAK
LXFPSKAUTEMNDCMGTSXMXBTUIADNGMGPSRELXNJELX
VRVPRTULHDNQWTWDTYGBPHXTFALJHASVBFXNGLLCHR
ZBWELEKMSJIKNBHWRJGNMGJSGLXFEYPHAGNRBIEQJT
AMRVLCRREMNDGLXRRIMGNSNRWCHRQHAEYEVTAQEBBI
PEEWEVKAKOEWADREMXMTBHHCHRTKDNVRZCHRCLQOHP
WQAIIWXNRMGWOIFFKEE
```

Finding the key length

The first thing to do here is to try and determine the number of alphabets. First, let's run the Kasiski substring finder on it:

```
% kasiski cipher
Length   Count  Word    Location (distance)
======   =====  ====    =========
     4       2  EMND    93 218 (125)
     3       5   CHR    0 165 (165) 235 (70) 275 (40) 285 (10)
     3       2   ELX    118 123 (5)
     3       2   EMN    93 218 (125)
     3       2   GLX    192 222 (30)
     3       2   LXF    84 193 (109)
     3       2   MND    94 219 (125)
     3       2   REM    217 266 (49)
     3       2   XWT    19 74 (55)
     3       2   XXW    18 53 (35)
```

Now, extract the distances and look for common factors. This can easily be done by hand, but the following pipeline is effective for the lazy[2]:

```
% kasiski cipher | tr ' ' '\012' | grep '(' |\
         tr -d '()' | grep -v distance | sort -n | uniq
5
10
30
35
40
49
55
70
109
125
165
```

[2]Although typing the command took more typing than typing the distances, so maybe not.

```
% factor ‘!!‘
factor ‘kasiski cipher | tr ’ ’ ’\012’ | grep ’(’ |\
          tr -d ’()’ | grep -v distance | sort -n | uniq‘
5: 5
10: 2 5
30: 2 3 5
35: 5 7
40: 2 2 2 5
49: 7 7
55: 5 11
70: 2 5 7
109: 109
125: 5 5 5
165: 3 5 11
%
```

Looking at these factors, the most common factor is 5. It looks like there are some coincidental overlaps (e.g., the one 109 spaces apart), but on the whole, it looks like we're looking at 5. To confirm, let's look at the index of coincidence.

First, what are we expecting for this sample:

```
% wc -c cipher
322 cipher
% ic 322 1 2 3 4 5 6 7 8 9 1000
Key  Expected IC (N=322)
---- ------------------
   1 0.0660
   2 0.0520
   3 0.0473
   4 0.0449
   5 0.0435
   6 0.0426
   7 0.0419
   8 0.0414
   9 0.0410
1000 0.0379
%
```

The number of characters returned by `wc` isn't going to be exactly right, but it's close enough for our purposes. Now, look at the index of coincidence for the ciphertext itself. A run of `ftable` over the ciphertext is shown in Figure 1. The index shown, 0.0449, falls right on the entry for key length of four in our table. This is close to five, so we might be on the right track. If nothing interesting comes of five, we should remember this and come back and look at four.

```
% ftable cipher
Total chars: 313
A:        19 (  6.07%) *******
B:        15 (  4.79%) *****
C:         8 (  2.56%) ***
D:         7 (  2.24%) ***
E:        26 (  8.31%) *********
F:         7 (  2.24%) ***
G:        15 (  4.79%) *****
H:        17 (  5.43%) ******
I:        10 (  3.19%) ****
J:         7 (  2.24%) ***
K:        10 (  3.19%) ****
L:        12 (  3.83%) ****
M:        17 (  5.43%) ******
N:        15 (  4.79%) *****
O:         7 (  2.24%) ***
P:         8 (  2.56%) ***
Q:        10 (  3.19%) ****
R:        24 (  7.67%) ********
S:         9 (  2.88%) ***
T:        14 (  4.47%) *****
U:         4 (  1.28%) **
V:        10 (  3.19%) ****
W:        16 (  5.11%) ******
X:        20 (  6.39%) *******
Y:         3 (  0.96%) *
Z:         3 (  0.96%) *

Index of Coincidence: 0.0449
```

Figure 1: Running `ftable` on vit2.txt

Finding the key itself

Assuming that the key length is five, our next step is to find out what it is. My approach is to look at each individual alphabet and try to guess the appropriate key letter based on letter frequency. We'll start by assuming that the highest-frequency letter in each alphabet is *e*. If this doesn't help, we'll try, *t*, *o*, *a*, *n*, etc., in turn.

First, the frequency tables. Figure 2 shows the frequency tables for each alphabet. If we guess that the highest-frequency letter of each alphabet is *e*, that gives us the following mappings:

Alphabet	Mapping	Key
0	$e \mapsto w$	s
1	$e \mapsto e$	a
2	$e \mapsto r$	n
3	$e \mapsto e$	a
4	$e \mapsto x$	t

"Sanat" doesn't look like a terribly promising key, but let's try decrypting with it and see if any patterns jump out:

```
% vig -d sanat cipher
KHEELDONHTIEEAAJINXEETAXIMEBPOJSOQTYEDEYJW
EVECONKEIOFXEEENHIEGWMTYMAKNZFIGEETEZEEINK
SFFCSRIUGETVDPMNBSKMEJTHIHLNTMNXSEESFNWESF
VEVWZTHLOLNDWAEDGYNJPUXANAYJOISIBMFNTLSKHE
ZIEEYERUSWIRVBUWYRGAMNRSTLENELPOIGARIQEDJA
IMEVSKREETVDTLEZRVMNVSARDKHEQOIELECBADEIJI
CELEEIKHSORWHLRRMEUTOHOKHETRLNIRGKHECSYOUP
DYAVIDFNEMNEOVFMSER
```

Ok, so this isn't the correct key, surely, but look at the end of the text, "NEOVFMSER". This looks to me a little like "november" Let's see what would have to change to make it "november"[3]. Let's start changing the key to make it work out. If we re-write the ciphertext so that each letter is subscripted with the number of its alphabet (as in Figure 3) we see that for the *S* in "$\mathtt{N}_4\mathtt{E}_0\mathtt{O}_1\mathtt{V}_2\mathtt{F}_3\mathtt{M}_4\mathtt{S}_0\mathtt{E}_1\mathtt{R}_2$" to become a *B* we'll have to change the key for alphabet 0 from *S* to *J*, making the key "janat".

```
% vig -d janat cipher
THEELMONHTREEAASINXENTAXIVEBPOSSOQTHEDEYSW
EVELONKEROFXENENHINGWMTHMAKNIFIGENTEZENINK
SOFCSRRUGETEDPMNKSKMESTHIHUNTMNGSEESONWESO
VEVWITHLOUNDWANDGYNSPUXAWAYJORSIBMONTLSTHE
ZINEYERDSWIREBUWYAGAMNASTLEWELPORGARIZEDJA
RMEVSTREETEDTLEIRVMNESARDTHEQORELECKADEISI
CELNEIKHBORWHURRMEDTOHOTHETRUNIRGTHECSHOUP
DHAVIDONEMNNOVFMBER
```

Even though the intuition was wrong, this is looking more and more like language. Let's complete the transformation of the end of the text. This requires changing alphabet three so that it maps *f* to *e*, making the key "janyt":

[3]Of course, my intuition was flawed—there are nine letters in that string and only eight in "november"—but one never knows what the subconscious has picked up. Intuition is important here.

```
% ftable -p 5 -s 0 cipher
Total chars: 63
A:         7 ( 11.11%) ***********
B:         6 (  9.52%) **********
C:         6 (  9.52%) **********
D:         4 (  6.35%) *******
E:         1 (  1.59%) **
F:         2 (  3.17%) ****
G:         0 (  0.00%)
H:         0 (  0.00%)
I:         1 (  1.59%) **
J:         2 (  3.17%) ****
K:         2 (  3.17%) ****
L:         0 (  0.00%)
M:         2 (  3.17%) ****
N:         4 (  6.35%) *******
O:         0 (  0.00%)
P:         1 (  1.59%) **
Q:         4 (  6.35%) *******
R:         3 (  4.76%) *****
S:         0 (  0.00%)
T:         2 (  3.17%) ****
U:         1 (  1.59%) **
V:         1 (  1.59%) **
W:         9 ( 14.29%) ***************
X:         5 (  7.94%) ********
Y:         0 (  0.00%)
Z:         0 (  0.00%)

Index of Coincidence: 0.0630
```

```
%  ftable -p 5 -s 1  cipher
Total chars: 63
A:         3 (  4.76%) *****
B:         1 (  1.59%) **
C:         0 (  0.00%)
D:         3 (  4.76%) *****
E:        10 ( 15.87%) ****************
F:         2 (  3.17%) ****
G:         3 (  4.76%) *****
H:         5 (  7.94%) ********
I:         3 (  4.76%) *****
J:         0 (  0.00%)
K:         0 (  0.00%)
L:         0 (  0.00%)
M:         2 (  3.17%) ****
N:         6 (  9.52%) **********
O:         6 (  9.52%) **********
P:         1 (  1.59%) **
Q:         0 (  0.00%)
R:         3 (  4.76%) *****
S:         7 ( 11.11%) ***********
T:         5 (  7.94%) ********
U:         1 (  1.59%) **
V:         1 (  1.59%) **
W:         1 (  1.59%) **
X:         0 (  0.00%)
Y:         0 (  0.00%)
Z:         0 (  0.00%)

Index of Coincidence: 0.0681
```

```
% ftable -p 5 -s 2 cipher
Total chars: 63
A:         5 (  7.94%) ********
B:         2 (  3.17%) ****
C:         1 (  1.59%) **
D:         0 (  0.00%)
E:         3 (  4.76%) *****
F:         0 (  0.00%)
G:         4 (  6.35%) *******
H:         3 (  4.76%) *****
I:         3 (  4.76%) *****
J:         3 (  4.76%) *****
K:         0 (  0.00%)
L:         2 (  3.17%) ****
M:         0 (  0.00%)
N:         5 (  7.94%) ********
O:         1 (  1.59%) **
P:         2 (  3.17%) ****
Q:         4 (  6.35%) *******
R:        13 ( 20.63%) *********************
S:         1 (  1.59%) **
T:         2 (  3.17%) ****
U:         2 (  3.17%) ****
V:         4 (  6.35%) *******
W:         0 (  0.00%)
X:         1 (  1.59%) **
Y:         2 (  3.17%) ****
Z:         0 (  0.00%)

Index of Coincidence: 0.0686
```

```
% ftable -p 5 -s 3 cipher
Total chars: 62
A:         1 (  1.61%) **
B:         1 (  1.61%) **
C:         1 (  1.61%) **
D:         0 (  0.00%)
E:        10 ( 16.13%) ****************
F:         1 (  1.61%) **
G:         1 (  1.61%) **
H:         3 (  4.84%) *****
I:         3 (  4.84%) *****
J:         2 (  3.23%) ****
K:         4 (  6.45%) *******
L:         4 (  6.45%) *******
M:         8 ( 12.90%) ************
N:         0 (  0.00%)
O:         0 (  0.00%)
P:         3 (  4.84%) *****
Q:         2 (  3.23%) ****
R:         3 (  4.84%) *****
S:         1 (  1.61%) **
T:         1 (  1.61%) **
U:         0 (  0.00%)
V:         3 (  4.84%) *****
W:         3 (  4.84%) *****
X:         4 (  6.45%) *******
Y:         1 (  1.61%) **
Z:         2 (  3.23%) ****

Index of Coincidence: 0.0592
```

```
% ftable -p 5 -s 4 cipher
Total chars: 62
A:         3 (  4.84%) *****
B:         5 (  8.06%) *********
C:         0 (  0.00%)
D:         0 (  0.00%)
E:         2 (  3.23%) ****
F:         2 (  3.23%) ****
G:         7 ( 11.29%) ************
H:         6 (  9.68%) **********
I:         0 (  0.00%)
J:         0 (  0.00%)
K:         4 (  6.45%) *******
L:         6 (  9.68%) **********
M:         5 (  8.06%) *********
N:         0 (  0.00%)
O:         0 (  0.00%)
P:         1 (  1.61%) **
Q:         0 (  0.00%)
R:         2 (  3.23%) ****
S:         0 (  0.00%)
T:         4 (  6.45%) *******
U:         0 (  0.00%)
V:         1 (  1.61%) **
W:         3 (  4.84%) *****
X:        10 ( 16.13%) ****************
Y:         0 (  0.00%)
Z:         1 (  1.61%) **

Index of Coincidence: 0.0724
% exit
```

Figure 2: The frequency tables for individual alphabets

```
% vig -d janyt cipher
THEGLMONJTREECASINZENTAZIVEBROSSOSTHEDGYSW
EXELONMEROFZENENJINGWOTHMAMNIFIIENTEBENINM
SOFCURRUGGTEDPONKSKOESTHKHUNTONGSEGSONWGSO
VEXWITHNOUNDYANDGANSPUZAWAYLORSIDMONTNSTHE
BINEYGRDSWKREBUYYAGAONASTNEWELRORGATIZEDLA
RMEXSTREGTEDTNEIRVONESATDTHESORELGCKADGISI
CGLNEIMHBORYHURROEDTOJOTHEVRUNITGTHEESHOUR
DHAVKDONEONNOVHMBER
```

Hmm, something's wrong with this. We're definitely on the right track with most of our key, though, because we have short words like "the" and "tree" present in our decrypted text. Let's try one more leap of intuition: the tentative key, "janyt" looks very much like the name "janet":

```
% vig -d janet cipher
THEALMONDTREEWASINTENTATIVEBLOSSOMTHEDAYSW
ERELONGEROFTENENDINGWITHMAGNIFICENTEVENING
SOFCORRUGATEDPINKSKIESTHEHUNTINGSEASONWASO
VERWITHHOUNDSANDGUNSPUTAWAYFORSIXMONTHSTHE
VINEYARDSWEREBUSYAGAINASTHEWELLORGANIZEDFA
RMERSTREATEDTHEIRVINESANDTHEMORELACKADAISI
CALNEIGHBORSHURRIEDTODOTHEPRUNINGTHEYSHOUL
DHAVEDONEINNOVBMBER
```

And there it is! We were mislead by a typo made by the code clerk. It appears that the plaintext for the last word was not "november", but "novbmber." This is unfortunate in that it could have led us away from the proper solution.

All that's left is a little cleaning up:

> The almond tree was in tentative blossom. The days were longer, often ending with magnificent evenings of corrugated pink skies. The hunting season was over with hounds and guns put away for six months. The vineyards were busy again as the well organized farmers treated their vines and the more lackadaisical neighbors hurried to do the pruning they should have done in November.

$C_0H_1R_2E_3E_4V_0O_1A_2H_3M_4A_0E_1R_2A_3T_4B_0I_1A_2X_3X_4W_0T_1N_2X_3B_4E_0E_1O_2P_3H_4B_0S_1B_2Q_3M_4Q_0E_1Q_2E_3R_4B_0W_1$
$R_2V_3X_4U_0O_1A_2K_3X_4A_0O_1S_2X_3X_4W_0E_1A_2H_3B_4W_0G_1J_2M_3M_4Q_0M_1N_2K_3G_4R_0F_1V_2G_3X_4W_0T_1R_2Z_3X_4W_0I_1A_2K_3$
$L_4X_0F_1P_2S_3K_4A_0U_1T_2E_3M_4N_0D_1C_2M_3G_4T_0S_1X_2M_3X_4B_0T_1U_2I_3A_4D_0N_1G_2M_3G_4P_0S_1R_2E_3L_4X_0N_1J_2E_3L_4X_0$
$V_1R_2V_3P_4R_0T_1U_2L_3H_4D_0N_1Q_2W_3T_4W_0D_1T_2Y_3G_4B_0P_1H_2X_3T_4F_0A_1L_2J_3H_4A_0S_1V_2B_3F_4X_0N_1G_2L_3L_4C_0H_1R_2$
$Z_3B_4W_0E_1L_2E_3K_4M_0S_1J_2I_3K_4N_0B_1H_2W_3R_4J_0G_1N_2M_3G_4J_0S_1G_2L_3X_4F_0E_1Y_2P_3H_4A_0G_1N_2R_3B_4I_0E_1Q_2J_3T_4$
$A_0M_1R_2V_3L_4C_0R_1R_2E_3M_4N_0D_1G_2L_3X_4R_0R_1I_2M_3G_4N_0S_1N_2R_3W_4C_0H_1R_2Q_3H_4A_0E_1Y_2E_3V_4T_0A_1Q_2E_3B_4B_0I_1$
$P_2E_3E_4W_0E_1V_2K_3A_4K_0O_1E_2W_3A_4D_0R_1E_2M_3X_4M_0T_1B_2H_3H_4C_0H_1R_2T_3K_4D_0N_1V_2R_3Z_4C_0H_1R_2C_3L_4Q_0O_1H_2P_3$
$W_4Q_0A_1I_2I_3W_4X_0N_1R_2M_3G_4W_0O_1I_2F_3F_4K_0E_1E_2$

Figure 3: The numbered ciphertext