

# Notes on Polyalphabetic Encryption and Decryption

Wenhan Gao

April 14, 2022

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Encryption</b>	<b>2</b>
2.1	Keyword Transpose Encoding . . . . .	2
2.2	Example: Keyword Transpose Encoding by the First Keyword . . . . .	3
2.3	Vigenère Encoding on the Keyword Transpose Encoded Text . . . . .	4
<b>3</b>	<b>Cryptanalysis and Decryption</b>	<b>5</b>
3.1	Index of Coincidence . . . . .	6
3.1.1	Computer Programming Assisted Implementation . . . . .	7
3.2	Frequency Strips . . . . .	7
3.2.1	Computer Programming Assisted Implementation . . . . .	8
3.3	Mono-alphabetic Decipher: Trigram Analysis . . . . .	9
3.3.1	Computer Programming Assisted Implementation . . . . .	10

## 1 Overview

A poly-alphabetic cipher is a cipher based on multiple substitution alphabets. A well-known example of poly-alphabetic ciphers is the Enigma Machine that was widely used by Nazi Germany during World War II. Oftentimes, we encrypt a message for secure communication. This article introduces a poly-alphabetic cipher whose encryption consists of two stages: Keyword Transpose Encoding at the first stage and Vigenère Cipher at the second stage. Each stage has an encoding keyword, and the receiver who knows the encryption scheme and both keywords can easily decode the message to human-readable texts. The author of this article first learned this cipher in AMS 303 Graph Theory course taught by Prof. Alan Tucker at Stony Brook University.

## 2 Encryption

**Definition 2.1.** A human-readable message is called the **plaintext**, and an encrypted message that is not human-readable is called the **ciphertext**.

In the encryption process, we wish to encode a plaintext into a ciphertext. There are two stages in this process: Mono-alphabetic Substitution based on Keyword Transpose Encoding at the first stage and Vigenère Cipher at the second stage. Without loss of generality, we assume that the text consists of only uppercase English letters(ASCII 65 to 90); in other words, spaces between words, punctuation marks, etc. are ignored.

### 2.1 Keyword Transpose Encoding

**Definition 2.2.** Let  $\alpha$  be a fixed alphabet. A **mono-alphabetic substitution key** is a bijection  $f : \alpha \rightarrow \alpha$ .

**Example 2.1.** Given a fixed English alphabet ABCD, a mono-alphabetic substitution key can be represented as BCDA.

Plaintext Alphabet: ABCD  
Substitution Alphabet: BCDA

**Notation 2.1.** Let  $\alpha$  be a fixed alphabet.  $\alpha^*$  denotes the set of all strings consisting of letters in  $\alpha$ .

**Definition 2.3.** Let  $\alpha$  be a fixed alphabet. Let  $f$  be a mono-alphabetic substitution key of  $\alpha$ . A **mono-alphabetic substitution cipher** is the induced monoid homomorphism  $\bar{f}$  on  $\alpha^*$ .

**Example 2.2.** Given a fixed English alphabet ABCD, and a mono-alphabetic substitution key BCDA. The corresponding ciphertext of the plaintext BBCDABACD is CCDABCBDA.

Plaintext: BBCDABACD  
Ciphertext: CCDABCBDA

Now, given a plaintext, we can encode this plaintext by mono-alphabetic substitution cipher. While sending the message, we want to avoid including the substitution key directly as it is easy to figure out that it is a mono-alphabetic substitution cipher. We deploy a scheme that uses a given keyword(the first keyword) to create a substitution key.

**Definition 2.4.** Given a keyword  $k$ , the **encoding key** of  $k$  is an ordered list of distinct letters in the given keyword with repeated letters omitted.

**Example 2.3.** Given a keyword BROOK, the encoding key is BROK.

**Definition 2.5.** Let  $\alpha$  be a ordered alphabet. Let  $\bar{k} \in \alpha^*$  be the encoding key of a given keyword. The **keyword encoding array** is an ragged array of letters whose first row is  $\bar{k}$  and each successive row is constructed by listing the rest of letters in  $\alpha$  such that each successive row has the same length(same number of elements) as  $\bar{k}$  except for the last row.

**Example 2.4.** Let the ordered alphabet be the English Alphabet and the keyword be BROOK. The corresponding Keyword Encoding Array is:

B	R	O	K
A	C	D	E
F	G	H	I
J	L	M	N
P	Q	S	T
U	V	W	X
Y	Z		

**Definition 2.6.** Given a keyword, the **encoding sequence** is obtained by listing letters, column by column, in the keyword's encoding array. The word transpose comes from this part.

**Example 2.5.** Given the Keyword Encoding in Example 2.4, the encoding sequence is:

**BAFJPUYRCGLQVZODHMSWKEINTX**

Notice that this can represent a mono-alphabetic substitution key of the English Alphabet.

## 2.2 Example: Keyword Transpose Encoding by the First Keyword

Now, with above mathematical constructions in Section 2.1, we can encode an English text message into a ciphertext with a given keyword through keyword transpose encoding.

**Definition 2.7.** Given a plaintext  $p$  and a keyword  $k$ . The keyword transpose encoding of  $p$  is  $\bar{f}(p)$ , where  $\bar{f}$  is the monoid homomorphism induced from the substitution key represented by the encoding sequence of  $k$ .

**Example 2.6.** Suppose we are given the plaintext:

*i am a math graduate from the State University of New York at  
Stony Brook where I studied both applied mathematics and pure  
mathematics i am interested in Machine Learning Discrete Math  
Graph Theory and many other topics in applied mathematics and  
Computer Science i am looking for an internship opportunity in  
the field of Machine Learning especially Deep Learning and Rec-  
ommender Systems i am also open to work in Computer Vision  
Robotics and Software Development*

, as assumed in the beginning, the plaintext consists of only uppercase English letter; the spaces and capitalization here are only for readers' comprehension. Now, suppose we are also given the keyword BROOK, then the encoding sequence is:

**BAFJPUYRCGLQVZODHMSWKEINTX**

; we can use it as the substitution key to encode the plaintext:

Plaintext Alphabet: *ABCDEFGHIJKLMNOPQRSTUVWXYZ*  
 Substitution Alphabet: *BAFJPUYRCGLQVZODHMSWKEINTX*

resulting ciphertext(again, ignore spaces):

*C BV B VBWR YMBJKBWP UMOV WRP SWBWP KZCEPMSCWT OU  
 ZPI TOML BW SWOZT AMOOL IRPMP C SWKJCPJ AOWR BDDQCPJ  
 VBWRPVBWCFS BZJ DKMP VBWRPVBWCFS C BV CZWPMPSWPJ CZ  
 VBFRCZP QPBMZCZY JCSFMPWP VBWR YMBDR WRPOMT BZJ VBZT  
 OWRPM WODCFS CZ BDDQCPJ VBWRPVBWCFS BZJ FOVDKWPM  
 SFCPZFP C BV QOOLCZY UOM BZ CZWPMZSRC D ODDOMWKZCWT  
 CZ WRP UCPQJ OU VBFRCZP QPBMZCZY PSDPFCBQQT JPPD  
 QPBMZCZY BZJ MPFOVVPZJPM STSWPVS C BV BQSO ODPZ  
 WO IOML CZ FOVDKWPM ECSCOZ MOAOWCFS BZJ SOUWIBMP  
 JPEPQODVPZW*

## 2.3 Vigenère Encoding on the Keyword Transpose Encoded Text

To start with Vigenère Encoding, we introduce the integer representation of a given alphabet to make this part more intelligible, efficient, and computer-programming-friendly.

**Definition 2.8.** Given an ordered alphabet  $\alpha$  with  $n$  distinct letters, the **integer representation** of a letter  $a$  in  $\alpha$  is  $h(a)$ , where  $h$  is a bijection from  $\alpha$  to the first  $n$  non-negative integers such that integer  $z$  is mapped to  $(z + 1)$ -th letter in the ordered alphabet. Given an integer  $z$ , the associated letter is  $h^{-1}(z)$ .

**Example 2.7.** The integer representation of the English alphabet:

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

**Definition 2.9.** Given an alphabet  $\alpha$ , a text  $p$  and a keyword  $k$  with  $p, k \in \alpha^*$ , the **Vigenère Encoding** is:

$$c_i = p_i + k_{i \bmod L} \bmod N$$

, where  $L$  is the length of keyword,  $N$  is the length of the given alphabet,  $c_i$  and  $p_i$  are integer representation of the  $i$ -th letter in the Vigenère encoded ciphertext and original text respectively, and  $k_{i \bmod L}$  is the  $(i \bmod L)$ -th letter in the keyword.

**Example 2.8.** Given a keyword *STONY*(18 19 14 13 24 ) and the text:

*C BV B VBWR YMBJKBWP UMOV WRP SWBWP KZCEPMSCWT OU  
 ZPI TOML BW SWOZT AMOOL*

whose integer representation is:

*2 1 21 1 21 1 22 17 24 12 1 9 10 1 22 15 20 12 14 21 22 17 15  
 18 22 1 22 15 10 25 2 4 15 12 18 2 22 19 14 20 25 15 8 19 14  
 12 11 1 22 18 22 14 25 19 0 12 14 14 11*

the Vigenère Encoded ciphertext is:

*U UJ O TTPF LKTCYOUH NABT OKD FUTPD XXUXDZQUPH BS  
RIW GMEE PJ QOHNG YEHCY*

whose integer representation is:

*20 20 9 14 19 19 15 5 11 10 19 2 24 14 20 7 13 0 1 19 14 10 3  
5 20 19 15 3 23 23 20 23 3 25 16 20 15 7 1 18 17 8 22 6 12 4 4  
15 9 16 14 7 13 6 24 4 7 2 24*

**Definition 2.10.** Given an ordered alphabet  $\alpha$ , a plaintext  $p \in \alpha^*$ , the first keyword  $k_1$ , and the second keyword  $k_2$ , the **polyalphabetic encryption** of  $p$  is obtained by first applying keyword transpose encoding by  $k_1$  on  $p$ , and then applying Vigenère encoding by  $k_2$  on the keyword transpose encoded ciphertext.

**Example 2.9.** The polyalphabetic encryption of the plaintext in Example 2.6 with respect to the first keyword BROOK and the second keyword STONY yields (again ignore spaces):

*U UJ O TTPF LKTCYOUH NABT OKD FUTPD XXUXDZQUPH BS RIW GMEE  
PJ QOHNG YEHCY GJIAC A KPYWAHC OBUJ URQOUIX IZOKDIZOVTF  
ZRC RXKH OPJPHOPJAXL Q OT USKCKHLKCH US JODJVNC OHUAMARR  
XPQXFDJN NUKE WEURE UJICZR TSX IZRM CJPHE KBBUYG PX TWRDAH  
JOUJIJOUUYG OXB YCIBCPDZ QXVMDH V PI OGHZPXQ NCZ ZR  
VNJNESGEAV HRQMEPYMAOM QM UJI IPNIC CH TTYFPXH JDOKRVNL  
NKWDSATJEG HHIR DNTFNPXQ UNW KHYCITHSXCK KMGJNNL Q OT  
TJGB MVIN JM AHAY AR YCIBCPDZ CULQBX EHOBUEYG OXB LCHUAUAC  
HHXDDMVODMU*

### 3 Cryptanalysis and Decryption

Decryption with known keywords is fairly simple; one may decode just by reversing operations done in the encryption process; that is to say, one should apply the second keyword to reverse the Vigenère encoding first and then apply the first keyword to reverse the transpose encoding. This section will introduce cryptanalysis of the poly-alphabetic encryption and methods to decode without knowing the keywords. Without loss of generality, we, again, assume that the ciphertext consists of only uppercase English letters (ASCII 65 to 90); in other words, spaces between words, punctuation marks, etc. are ignored. We also assume that the second keyword is of length 4 or 5 for simplicity. Most importantly, we also assume that the plaintext is in modern English with the standard English alphabet as the intended audience of this article probably understands English. The cryptanalysis introduced in this article may be extended to keywords of any length and any language. The general steps to crack down this poly-alphabetic cipher are as follows:

- 1. Find the length of the second keyword by Index of Coincidence Analysis
- 2. Find the second keyword by frequency strips
- 3. Reverse Vigenère cipher by the second keyword

- 4. Find the first keyword by trigram analysis
- 5. Reverse Keyword Transpose Encoding by the first keyword

### 3.1 Index of Coincidence

**Definition 3.1.** Given an alphabet  $\alpha$  and a text  $t \in \alpha^*$ . The **index of coincidence(IC)** of  $t$  is the probability that two randomly selected letters in  $t$  are the same.

**Example 3.1.** Given an English text so that the alphabet is the English alphabet.

$$\begin{aligned}
 IC &= P(\text{two randomly chosen letters are the same}) \\
 &= P[(A_1 \cap A_2) \cup (B_1 \cap B_2) \cup \dots \cup (Z_1 \cap Z_2)] \\
 &= P(A_1 \cap A_2) + P(B_1 \cap B_2) + \dots + P(Z_1 \cap Z_2) \\
 &= \left(\frac{n_0}{n}\right)\left(\frac{n_0-1}{n-1}\right) + \left(\frac{n_1}{n}\right)\left(\frac{n_1-1}{n-1}\right) + \dots + \left(\frac{n_{25}}{n}\right)\left(\frac{n_{25}-1}{n-1}\right) \\
 &= \frac{1}{n(n-1)} \sum_{i=0}^{25} n_i(n_i-1)
 \end{aligned}$$

, where  $A_1$  denotes the event that the first randomly chose letter is  $A$ ,  $A_2$  denotes the event that the second randomly chose letter is also  $A$ ,  $n$  is the length of the given text, and  $n_i$  denotes the number of occurrence of the letter whose integer representation is  $i$  in the given text; in other words,  $n_0$  is the number of  $A$ s in the given text and  $n_1$  is the number of  $B$ s in the given text, etc..

**Example 3.2.** The index of an English ciphertext: *U UJ O TTPF LKTCYOUH NABT OKD FUTPD XXUXDZQUPH BS RIW GMEE PJ QOHNG YEH CY* is 0.03682. The index of an English plaintext: *I AM A MATH GRADUATE FROM THE STATE UNIVERSITY OF NEW YORK AT STONY BROOK* is 0.0602. One may verify these numbers by counting the letter frequencies and applying above formula for  $IC$ .

According to [Friedman(1987)], the index of coincidence of the English language is approximately 0.0686 so is an affine ciphertext, the index of coincidence of an English Vigenère ciphertext is approximately 0.046, and the index of coincidence of a random text of the English alphabet is approximately 0.038466. We can use this to find the length of the Vigenère keyword(the second keyword). Suppose the keyword length is  $L$ , we may break the ciphertext into  $L$  subsequences  $s_1$  to  $s_L$  such that the  $(i+jL)$ -th letter of ciphertext is in  $s_j$  for  $j = 1, 2, \dots$ . For example, if the keyword length is 4, then  $s_1$  contains the 1st, 5th, 9th, 13th,..., letters in the ciphertext. If the supposed keyword length  $L$  is in fact the correct keyword length, then each subsequence is homomorphic to that of the plaintext; thus, each subsequence should have an index of coincidence of approximately 0.0686. If the supposed keyword length  $L$  is incorrect, then each subsequence is homomorphic to that of an Vigenère ciphertext; thus, each subsequence should have an index of coincidence of approximately 0.046.

**Example 3.3.** To find the second keyword length of a Vigenère ciphertext: U UJ O TTPF LKTCYOUH NABT OKD FUTPD XXUXDZQUPH BS RIW GMEE PJ QOHNG YEHCY, as we assumed, the keyword length is either 4 or 5, we guess the keyword length is 4 first. Now, we calculate the IC of each subsequence and make statistical inference on the IC of ciphertext, and the result is  $IC_4 \approx 0.02454 \mp 0.05$ . Next, we guess the keyword length is 5, and then similarly, we get  $IC_5 \approx 0.06545 \mp 0.028$ . As  $IC_5$  is closer to 0.0686, we may draw the conclusion that the Vigenère keyword is of length 5.

In practice, given a long ciphertext, we may just calculate the IC of the first subsequence, and this value should be close the IC of the ciphertext.

**Example 3.4.** The the ciphertext in Example 2.9 has an  $IC_5$  of 0.08225 and an  $IC_4$  of 0.04334 using only the first subsequence. We may conclude that the second keyword is of length 5.

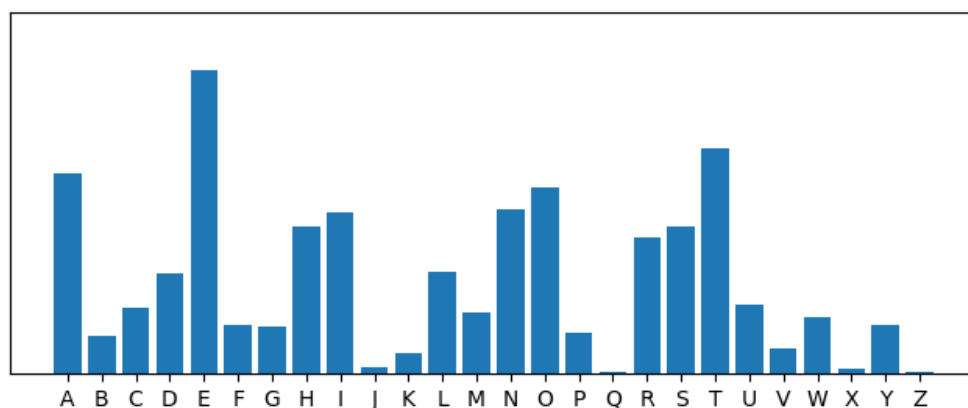
### 3.1.1 Computer Programming Assisted Implementation

Calculating IC and finding the keyword length is fairly an easy coding task. Thus, we will not discuss any detail for this part.

## 3.2 Frequency Strips

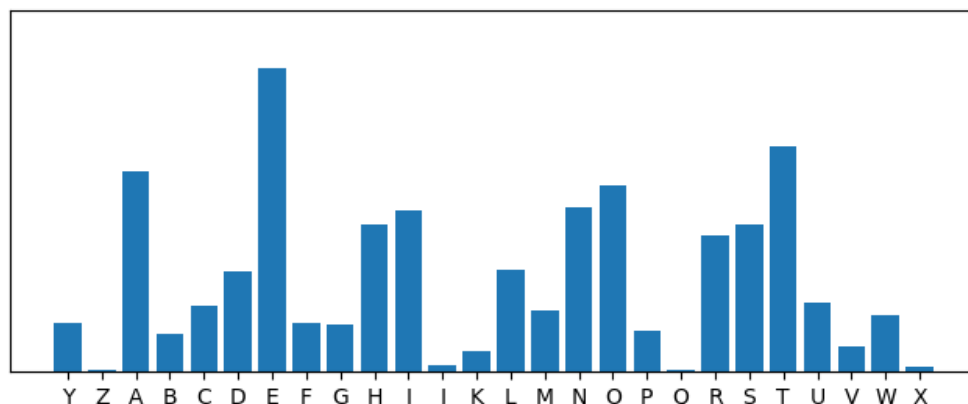
**Example 3.5.** The relative **frequency strip** of the English Language is shown below:

Figure 1: Relative Frequency Strip of the standard English Language



As noted in Section 3.1, with the right keyword length, all subsequences are homomorphic to that of the plaintext. Thus, the relative letter frequency distribution of each subsequence should be similar to that of the English language. If the Vigenère ciphertext is obtained from a plaintext, then we may try to match the frequency of letters of each subsequence to that of the English language to guess each letter in the keyword.

Figure 2: Frequency Strip of a Subsequence



**Example 3.6.** Suppose the first subsequence of a given Vigenère ciphertext has the below frequency strip: It is easy to see that we can get a frequency strip that is very similar to this by shifting the frequency strip of the English language to the right by 2, then we can guess the key letter is C whose integer representation is 2,

As we have the second keyword length, we may break the ciphertext into several subsequences. However, the text that is encoded with Vigenère cipher is not the English plaintext as we encode it with the keyword transpose encoding first. Thus, matching frequency strip of subsequences to that of the English language is not going to help. However, as all subsequences have frequencies similar to each other. We can align subsequences all together, and in the aligned frequency strips, there should be a column of letters forming an English word that is the second keyword.

### 3.2.1 Computer Programming Assisted Implementation

We may find the keyword by aligning frequency strips; however, so long as the frequency alignment reflects the right keyword, it is okay to use any column of letters as the "keyword" in the decryption process. That is to say, if STONY(19 20 15 14 25) is the keyword, using TUPOZ(18 19 14 13 24) to decode will work as well. This is because any column of letters can result in a homomorphism that can be cracked by trigram analysis. To phrase this in a less mathematical way, any column of letters can lead to a ciphertext that can be decrypted by trigram analysis. Note that if you were to decrypt by hand, finding if there is a column of English keyword is important as it indicates whether your alignment is right. With computers, one way we may try is to brutal force through every possible guess, and there will be  $26^{L-1}$ , where  $L$  is the keyword length, possible choices. As we need to perform mono-alphabetic decipher with every possible "keyword" here, it can be computationally expensive as the complexity grows exponentially with the keyword length. I will introduce a way to give a ranking to possible choices, and we can just try a certain number of top-ranked choices and hope that one of them works.



Alignment of frequencies is obvious to our human eyes. However, computers don't see it. To implement alignment of frequency strips, we may define a "difference measuring function" that takes an "guess shift" and outputs a measure indicating the overall degree of alignment. To illustrate, let's say we have an English keyword that is of length 5; the function takes a 4-tuple of letters representing the shifts in frequency strips 2 to 5(leave the first one fixed); for example, (2,10,13,17) means the shifting the second strip to the right by 2, the third to the right by 10, etc., and this also corresponds to the "keyword" being ACKNR. Then the function outputs a number indicating the extend of "difference" in frequency distributions; the higher the "difference" is, the less aligned frequency strips are. Finally, we take a certain number of highest-ranked(with lowest difference) 4-tuples for trials in the mono-alphabetic decryption phase. Here, the difference measuring function I use is mathematized as follows:

For each column in the shifted frequency strips, the difference  $J_c$ :

$$J_c = \begin{cases} 0 & \text{if } \mu_c < \frac{\text{ciphertext length}}{1000} \\ \frac{\sum_{i=1}^L |C_i - \mu_c|}{\mu_c} & \text{otherwise} \end{cases}$$

, where  $L$  is the keyword length(in each column, there are  $L$  entries),  $C_i$  is the  $i$ -th entry in the column and  $\mu_c$  is the mean of all entries in this column. When  $\mu_c$  is less than a threshold, meaning that all frequencies in this column are all small, we conveniently set  $J_c = 0$ . The ciphertext length divided by 1000 would be a reasonable threshold. Overall, this measuring function catches the variance in frequency in the column, and the denominator  $u_c$  balances the difference between high frequency letters such as E, A, and R and low frequency letters such as B, Y, and F because  $C_i$  tends to be larger for high frequency letters. One can view this process as **mean normalization**.

The overall difference measurement:

$$J = 0.9^{N_\mu} \cdot \sum_{c=1}^{L_\alpha} J_c$$

, where  $L_\alpha$  is the number of letters in the alphabet, e.g. 26 for the English alphabet, and  $N_\mu$  is the count of  $\mu_c$  that is less than the threshold. We "reward" the shift if in the column all frequencies are small meaning that we probably matched lower frequency letters such as X, Q, J, and Z. We do this because oftentimes, in a given plaintext, especially when it is relatively short, the letter frequency distribution of high frequency letters does not necessarily match but lower frequency letter do.

Pseudocode in Algorithm 3.1:

### 3.3 Mono-alphabetic Decipher: Trigram Analysis

Guessing with the help of trigrams is the main tool in this portion of decryption. This is explained in details in the Postlude of the Applied Combinatorics textbook [Tucker(2012)].

**Definition 3.2.** *Given a fixed alphabet  $\alpha$ , a  $n$ -gram is a sequence of  $n$  letters; in other words, an  $n$ -gram is an element in  $\alpha^n$ .*

---

**Algorithm 3.1:** Frequency Strip Alignment Ranking

---

**Result:**  $k$  most possible keywords to the Vigenère encoding for trial

**Require:** ciphertext

- 1 Get the keyword length  $L$  by index of coincidence analysis;
  - 2 Generate  $L$  frequency strips ;
  - 3 `guess_shifts` := all possible tuples of numbers representing shifts;
  - 4 Sort `guess_shifts` by their difference measures in ascending order;
  - 5 **return** First  $k$  `guess_shifts`;
- 

**Example 3.7.** *Given the English alphabet, THE, AND, THA, and HER are examples of trigrams(3-grams).*

Decoding by hand is tough, one may list all the trigrams in the ciphertext by making a trigraph table and try to find representations of THE and THAT. Some common English trigrams are THE, AND, ING, ENT, ION, HER, FOR, THA, NTH, INT, ERE. Also, making a transposed keyword table may help if and only if you have the correct second keyword. Details are in [Tucker(2012)].

### 3.3.1 Computer Programming Assisted Implementation

Computers can try a huge amount of substitution keys much faster than humans. A general idea is to have computers try different substitution keys and determine if the text after substitution is English plaintext. Of course, computers don't understand English without some complicated NLP applications. We just let the program try a huge amount of different substitution keys and return the most promising substitution. We give each substitution a rating by:

$$R(\xi) = \sum_1^{n_t} f(T_i)$$

, where  $\xi$  is a substitution,  $T_i$  is the  $i$ -th trigrams in the text substitute back to "plaintext" by  $\xi$ , and  $n_t$  is the total number of trigrams. Here  $f$  is a function that takes a trigram and returns the relative frequency of this trigram in the English language. For example,  $f(THE) = 0.01814036094632001$  and  $f(AND) = 0.007252281087500791$ . These numbers is generated by the data from [Lyons(2022)] and is implemented as a hashmap in the program. Now, if a substitution key has a higher rating, we say that it is a more promising substitution. So given a ciphertext(after Vigenère decryption), the pseudocode of mono-alphabetic decipher is in Algorithm 3.2.

There are one optimization that can be made in this algorithm, for the first few initial trials, we initialize the most frequent trigram in the ciphertext to be one of those most common trigrams in English.

In the previous step, we have a few possible second "keywords"(shifts), we may use a similar scheme to update the best rating and the best keywords.

---

**Algorithm 3.2:** Monoalphabetic Decipher

---

**Result:** Most promising substitution and its rating

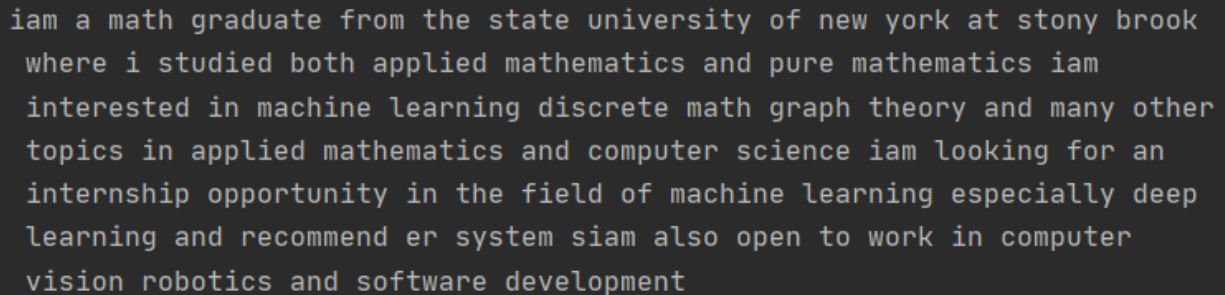
**Require:** ciphertext,  $N$ : number of initial trials,  $K$ : number of swaps in each trial, English trigram frequency hashmap

```
1 Initialize best rating = 0, i = 0, j = 0, best substitution = empty string;
2 repeat
3   i+ = 1
4   Randomly generate a substitution key and calculate rating
5   if rating > best rating then
6     best rating := rating ;
7     best substitution = current substitution
8     repeat
9       j+ = 1
10      Randomly Swap 2 letters in the substitution key
11      if rating > best rating then
12        best rating := rating ;
13        best substitution = current substitution
14    until j = N;
15 until i = N;
16 return best rating; best substitution;
```

---

When I run the program to decrypt the ciphertext in Example 2.9, the program yields

Figure 3: Sample Result



```
iam a math graduate from the state university of new york at stony brook
where i studied both applied mathematics and pure mathematics iam
interested in machine learning discrete math graph theory and many other
topics in applied mathematics and computer science iam looking for an
internship opportunity in the field of machine learning especially deep
learning and recommend er system siam also open to work in computer
vision robotics and software development
```

I also used a program to segment words which did a decent job, but not highly accurate, one can also use a more sophisticated NLP program to do this.

## References

[Friedman(1987)] William Friedman. *The index of coincidence and its applications in cryptanalysis*. Aegean Park Press, Laguna Hills, Calif, 1987. ISBN 978-0894121371.

- [Tucker(2012)] Alan Tucker. *Applied Combinatorics*. John Wiley & Sons, Chichester, England, 6 edition, January 2012.
- [Lyons(2022)] James Lyons. Crypto, 2022. URL <http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/>.