

Principles of Information Security

Assignment 1

Password Cracking

Anonymous

February 26, 2025



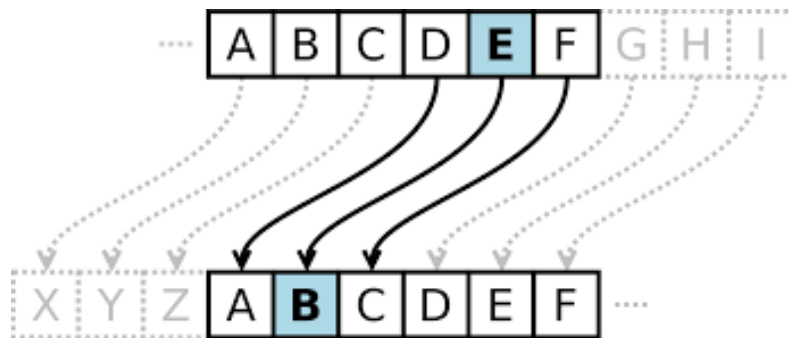
Contents

1	Caesar Cipher	3
1.1	Introduction	3
1.2	Plaintext and Key	3
1.3	Cryptanalysis Process	4
1.4	Source Code (C++)	4
2	Vigenère Square	5
2.1	Introduction	5
2.2	Plaintext and Key	5
2.3	Cryptanalysis Process	6
2.3.1	Determining Key Length by Coincidence Index Method	6
2.3.2	Determine the Relative Offset by Mutual Coincidence Index Method	7
2.3.3	Determine the Key	9
2.4	Source Code (C++)	10
3	Unknown Cipher	14
3.1	Plaintext and Key	14
3.2	Cryptanalysis Process	14
3.3	Source Code (Python)	14

1 Caesar Cipher

1.1 Introduction

The Caesar cipher is one of the oldest and simplest encryption techniques, attributed to Julius Caesar. In this cipher, each letter in the plaintext is shifted by a fixed number of positions down or up the alphabet. For instance, a shift of 3 would transform the letter *A* into *D*, *B* into *E*, and so on. Although it is easy to implement and understand, the Caesar cipher is vulnerable to simple cryptographic attacks, such as frequency analysis, and can be easily broken with modern computational tools. However, it still serves as a valuable introduction to the concepts of encryption and decryption.



1.2 Plaintext and Key

- Cipher text: FBUQUIUDSHOFJOEKHDQCUMYJXJXUIQCUAUOQDTKFBEQTJEBUQHDYDWYDPZK
- Key: $26 - 10 = 16$
- Plain text: Please encrypt your name with the same key and upload to learning in ZJU

```
Possible plain text with offset = 1: GCVRJVVETIPGKPFILIERDVNZKYKYVJRDBVBPVREULGCFRUKFCVRIEZEXZEQAL
Possible plain text with offset = 2: HDWSKWWFUJQHLQGMJFSEWOALZLZWKEWCWQSFVMDGSLVLDGWSJFAFYAFRBM
Possible plain text with offset = 3: IEXTLXXGVKIRMRHNGKTFXPBMAMAXLTFDXRTGWNIEHTWMHEXTKGBGBGSCN
Possible plain text with offset = 4: JFYUMYYHMLSJNSIOLHUGYQCNBNBYMUGVEYSUHXOJFIUXNIFYULHCHACHTDO
Possible plain text with offset = 5: KGZVNZIXMTKOTJPMIVHZRDOCOCZNVHZFZTVIYPKGJVYJGZVMIDIBDIUEP
Possible plain text with offset = 6: LHAWOAAJYNULPUKQNJWIASEPDDAOWIAGAUWJZQLHKWZPKHAWNJEJCEJVFQ
Possible plain text with offset = 7: MIBXPBBKZOVNQVLROKXJBTFQEQEBPXJBHBVXKARMILXAQLIBXOKFKDFKWR
Possible plain text with offset = 8: NJCYQCCLAPWNRWMSPLYKUGRFRFCQYKICWYLSNBJMYBRMJCYPLGLEGLXHS
Possible plain text with offset = 9: OKDZRDDMBQXOSXNTQMZLDVHSGSGDRZLDJDXZMCTOKNZCSNKKDZQMFMFMYIT
Possible plain text with offset = 10: PLEASEENCRYPTYOURNAMEWITHTHESAMEKEYANDUPLOADTOLEARNINGINZJU
Possible plain text with offset = 11: QMFBTFFODSZQUZPVSOBNFXJUIUIFTBNFLFZBOEVQMPBEUPMFBSOJOHJOAKV
Possible plain text with offset = 12: RNGCUGGPETARVAQWTPCOGYKVJVJGUCOGMGACPFWRNQCFFVQNGCTPKPIKPLW
Possible plain text with offset = 13: SOHDVHHQFUBSWBRXUQDPHZLWVKHVDPHNBDQGXSDRGWROHDDUQLQJLQCMX
Possible plain text with offset = 14: TPIEWIIRGVCCTXCSYVREQIAMXLXLIWEQIOICERHYTPSEHXSPIEVRMRKMDNY
Possible plain text with offset = 15: UQJFXJJSWHDUYDTZWSFRJBNYMYMJXFRJRPJDFSIZUQTFIYTFQJFWSNSLNSOZ
Possible plain text with offset = 16: VRGKYKTIIXEVZEUATGSKCOZNNKYGSKQEGTJAVRUGJZURKXGTOTMOTFPA
Possible plain text with offset = 17: WSLHZLLUJYFWAFVBYUHTLDPAAOALZHTLRLFHUKBWSVHKAVSLHYUPUNPUGQB
Possible plain text with offset = 18: XTMIAMVMKZGXBGCZVIUEQBPPMAIUMSMGIVLCTWILBWTIMIZVQVQVHRC
Possible plain text with offset = 19: YUNJBNNWL AHYCHDAWJVNFRCCQNB JVNTNHJWMDYUXJMCXUNJAWRWRPWRISD
Possible plain text with offset = 20: ZVOKCOOXMIBIZDIYEBXKWOQSDRDRCKWOUOIKXNEZVYKNDYVOKBXSQXSJTE
Possible plain text with offset = 21: AWPLDPPYNCJAEJZFCYLXPHTESESPDLXPVPJLYOFAWZLOEZWL CYTYRTYKUF
Possible plain text with offset = 22: BXQMEQQZODKBFKAGDZMYQIUFTFTQENYQWQKMPGBXAMPFAXQMDZUSZLVG
Possible plain text with offset = 23: CYRNFRRAPELCGLBHEANZRJVUGURFNZRRLNAQHCYBNQGBYRNEAVATVAMWH
Possible plain text with offset = 24: DZSOGSSBQFMDHMCIFBOASKWHVHVSOGAASYSMOBRIDZCORHCZSOFBWBUBNXXI
Possible plain text with offset = 25: EATPHTTCRGNEINDJGCPBTLXIWIWTHPBTZTNPCSJEADPSIDATPGCXVCXCOYJ

Press any key to continue . . .
```

As is shown in the figure above, $\text{offset} = -10$ successfully decrypts the cipher text to plain text, while other incorrect keys give meaningless results.

1.3 Cryptanalysis Process

The cryptanalysis process for the Caesar cipher is straightforward due to the limited number of possible shifts. Here is a step-by-step description of the algorithm:

1. **Try all possible shifts:** Since the Caesar cipher only uses shifts between 1 and 25, we can attempt all of them. Each shift corresponds to a possible decryption.
2. **Shift the ciphertext:** For each possible shift, the ciphertext is modified by shifting each letter in the ciphertext in the opposite direction of the encryption.
3. **Check for meaningful text:** After applying a shift, the result is checked to see if it forms readable text in the language (for example, English). This can be done manually or by using automated language detection methods.
4. **Select the correct shift:** The correct shift is identified when the decrypted text makes sense or matches known patterns of the plaintext. (Requires basic English capability)

The following pseudocode outlines the process of attempting all shifts and checking the results:

1. Let `cipher_text = "FBUQIUUDSHOFJOEKHDQC.....EBUQHDYDWYDPZK"`
2. For each offset from 1 to 25:
 - a. Create a copy of `cipher_text`, called `temp_string`
 - b. For each character `chr` in `temp_string`:
 - i. If `chr + offset <= 'Z'`, shift `chr` by `-offset`
 - ii. If `chr - offset < 'A'`, shift `chr` by `26 - offset` (wrap around)
 - c. Print the decrypted `temp_string` for the current offset
 - d. Check if `temp_string` is meaningful
3. The offset that results in readable text is the correct shift

1.4 Source Code (C++)

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
```

```

6     string cipher_text = "
          FBUQUIUDSHOFJOEKHDQCUMYJXXJXUIQCUAUOQDTKFBEQTJEBUQHDYDWYDPZK
          ";
7     int offset;
8
9     for (offset = 1; offset <= 25; ++offset) {
10         string temp_string = cipher_text;
11         for(auto& chr : temp_string) {
12             if (chr + offset <= 'Z') {
13                 chr += offset;
14             } else {
15                 chr = chr + offset - 26;
16             }
17         }
18
19         cout << "Possible plain text with offset=" << offset
              << ": " << temp_string << endl;
20     }
21 }

```

2 Vigenère Square

2.1 Introduction

The Vigenère cipher, named after the French cryptographer Blaise de Vigenère, improves upon the Caesar cipher by using a keyword to determine the shifting pattern. Instead of shifting all letters by the same amount, the Vigenère cipher uses different shifts for each letter in the plaintext, based on the letters of the key. This makes it significantly more secure than the Caesar cipher, as it introduces a polyalphabetic encryption scheme that resists basic frequency analysis. Despite its strength, the Vigenère cipher can still be broken with techniques such as the Kasiski examination and the Friedman test if the key is short or known to the attacker.

2.2 Plaintext and Key

- Cipher text: ktbueluegvitnthuexmonveggmrcgxptlyhhjaogchoemqchpdnetxupbqntieti
abpsmaoncnwvoutiugtagmmqsxtvxaoniiogtagmbpsmtuvvihpstpdcvcrxhokvhtawswquun
ewcgxptlcrxtevtubviewcnwwsxfnsnptswtagakvoyyak
- Key: cat
- Plain text: it is essential to seek out enemy agents who have come to conduct

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

espionage against you and to bribe them to serve you give them instructions and care for them thus doubled agents are recruited and used sun tzu the art of war (after reasonable parsing)

2.3 Cryptanalysis Process

Unlike Caesar cipher, if we simply use exhaustive enumeration for Vigenère Square, there will be 26^N candidates (in this case 26^{183}), which is impossible to calculate. So algorithm with lower time complexity need to be used to find the key and plain text quickly.

The cryptanalysis of the Vigenère cipher consists of three major steps: determining the key length, determining the relative offsets, and determining the key itself. Below we describe each step in detail, accompanied by the corresponding C++ programs that implement each algorithm.

2.3.1 Determining Key Length by Coincidence Index Method

The first step in breaking the Vigenère cipher is to determine the length of the key. The Coincidence Index (CI) method is commonly used for this purpose. The CI measures the likelihood of two characters in a text being the same, which can help in estimating the key

length by analyzing the repeating patterns of the ciphertext.

1. **Divide the ciphertext into groups:** The ciphertext is divided into groups based on the suspected key length. Each group corresponds to the same position in the key. For example, for a key length of 3, the ciphertext is divided into three groups, each containing characters at positions 0, 3, 6, 9, etc.
2. **Calculate the Coincidence Index for each group:** For each group, calculate the frequency of each letter and use this to compute the CI. The formula for CI is:

$$CI = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n - 1)}$$

where f_i is the frequency of the i -th letter in the group and n is the length of the group.

3. **Compare the CI values for different key lengths:** The key length with the highest average CI is likely to be the correct key length, about 0.065. While incorrect key length gives CI about 0.031-0.045.

The result of the algorithm is shown below:

```
Testing with key length = 2: p = 0.0494505 0.0468864 , with average = 0.0481685
Testing with key length = 3: p = 0.0639344 0.0781421 0.0639344 , with average = 0.0686703
Testing with key length = 4: p = 0.0376812 0.0444444 0.0521739 0.0444444 , with average = 0.044686
Testing with key length = 5: p = 0.042042 0.045045 0.0345345 0.0460317 0.0507937 , with average = 0.0436894
Testing with key length = 6: p = 0.0709677 0.0817204 0.0516129 0.0643678 0.0712644 0.0758621 , with average = 0.0692992
Testing with key length = 7: p = 0.045584 0.0307692 0.04 0.08 0.0430769 0.0307692 0.0430769 , with average = 0.0447538
Testing with key length = 8: p = 0.0434783 0.0355731 0.0474308 0.0434783 0.0632411 0.0790514 0.0474308 0.038961 , with average = 0.0498306
Press any key to continue . . . |
```

Obviously, key length = 3 and 6 gives highest CI that is close to 0.065. Since 6 is a multiple of 3, the key length is therefore 3.

2.3.2 Determine the Relative Offset by Mutual Coincidence Index Method

Once the key length is determined, the next step is to calculate the relative offsets between the groups. This can be done using the Mutual Coincidence Index (MIC) method, which calculates the similarity between different groups by shifting one group relative to another and measuring their overlap.

1. **Divide the ciphertext into groups based on the key length:** The ciphertext is divided into k groups (where k is the key length). Each group corresponds to a specific position in the key.
2. **Calculate the Mutual Coincidence Index (MIC) for each pair of groups:** For each pair of groups, calculate the MIC for different shifts. The MIC is calculated as:

$$MIC(i, j, k) = \sum_{l=0}^{25} \text{freq}_i(l) \times \text{freq}_j((l + k) \bmod 26)$$

where $\text{freq}_i(l)$ and $\text{freq}_j(l)$ are the frequencies of letter l in groups i and j , respectively.

3. **Determine the shift with the highest MIC:** For particular i and j , the shift k with the highest MIC indicates the most likely offset between the two groups. This offset corresponds to the relative shift between two letters of the key.

Below is the pseudocode description for the algorithm:

```

1: procedure DETERMINERELATIVEOFFSETS(cipher_text, key_length)
2:   Divide the cipher_text into key_length groups
3:   Initialize an empty 3D array MIC[0..key_length-1][0..key_length-1][0..25]
4:   for each pair of groups (i, j), where  $i < j$  do
5:     Initialize  $\text{max\_MIC} \leftarrow 0$ 
6:     Initialize  $\text{max\_k} \leftarrow 0$ 
7:     for each possible shift  $k$  from 0 to 25 do
8:       Compute the Mutual Coincidence Index (MIC) for the pair of groups (i, j)
       using the formula:

```

$$MIC(i, j, k) = \sum_{l=0}^{25} \text{freq}_i(l) \times \text{freq}_j((l + k) \bmod 26)$$

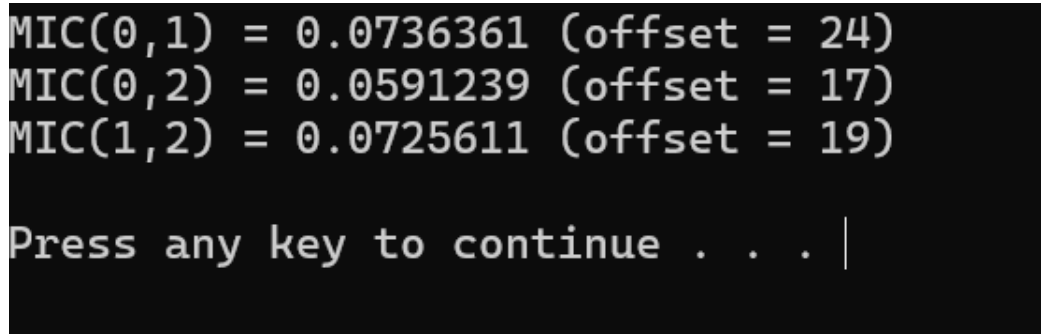
where $\text{freq}_i(l)$ and $\text{freq}_j(l)$ are the frequencies of letter l in groups i and j , respectively.

```

9:       if MIC(i, j, k) > max_MIC then
10:         Update  $\text{max\_MIC} \leftarrow MIC(i, j, k)$ 
11:         Update  $\text{max\_k} \leftarrow k$ 
12:       end if
13:     end for
14:     Output  $MIC(i, j)$  and the corresponding shift  $\text{max\_k}$ 
15:   end for
16:   The relative offsets between the groups are identified by the shift  $k$  with the highest
   MIC for each pair of groups.
17: end procedure

```

Here is the running result of the algorithm:



```

MIC(0,1) = 0.0736361 (offset = 24)
MIC(0,2) = 0.0591239 (offset = 17)
MIC(1,2) = 0.0725611 (offset = 19)

Press any key to continue . . . |

```


Thus, the offset between the first and second characters is 24, while the offset between the second and third characters is 19. For instance, assume the first character is 'a', the second should be 'y' and the third should be 'r'.

2.3.3 Determine the Key

After determining the relative offsets between the groups, the final step is to deduce the key. The key can be reconstructed by combining the determined offsets and aligning them with the expected shifts of the alphabet. There are only 26 candidates, so enumeration is feasible to get the final solution.

1. **Construct an initial key guess:** Based on the offsets obtained from the MIC step, form an initial guess of the key. The offsets correspond to shifts of individual letters in the key.
2. **Test all possible shifts for each key position:** For each letter in the key, test all possible shifts (from 0 to 25) and decrypt the ciphertext with each candidate key.
3. **Evaluate the decrypted text:** For each key guess, check if the resulting plaintext is meaningful. The correct key will produce readable text.

Below is the screenshot from the terminal output:

```
Test Key: ayr Plain Text: lkvkuuugpvkcnvuggnqwgvgpgoacigpvuyjgjcxeqogvgeqpfvewugrkqpcicgckcpvuaqwcpfvqdtkdgvjgovugtxgaqwikxgvjgokpuvtwevkpucpfectghqtvjgovjwufqndgfcigpvuctgtg
etwkvgfcfpfugfumpvbwvjgctvqhct
Test Key: bzs Plain Text: jujtfttfoujbmuptfflpvufofnzbhfoutxipibwfdpnfpdpoevduftqjpobhfbbjbtuzpvboeupscjcfuifnuptsfwfpzhvjwfuifnotusvdujpotboedsbfgspuifnuivtpevmfcbhfoutbsfsf
dsvjufbeboevtfetvouaavufbsupgxbz
Test Key: cat Plain Text: ktsassentiahtoseekoutenemvagentsmohavecometoconductespionageagainstyouandtobribethemtoserveyougive them instructions and care for them this doubled agents are
suited and used suitz theartofwar
Test Key: dbu Plain Text: hshdrdrdmshzksnrdjdntsdmdlxzfdmsrvngzudbnldsnbmctbsdrohnmzfzdzfzhmsrxntzmcnsaqhadsdglnsrdqudntxfhudsgdlhmsrqtbsnhmzmczbqdenqsgdlsgrcntakdczfdmsrzqdd
bqthsdcmzctdrctmsytsdgsznevzq
Test Key: ecv Plain Text: grgqcqqlrgyjrqqccimsrclckwyeclrqfmyftcamkcrmanlbsarcqngmlcecyegglrwmvslbrmgzpczrcfckrmqpcctmsegctrcfckglqrpsargmlqylbaypcdmprfckrfsqbsmzsjbyecrlqaypcpc
apsgrcbylbsqcbqslrxsrfcypmduyp
Test Key: fdm Plain Text: ftdpppbkqfxiqlpbhblrqbkjvxdbkptelxsbzljblzlkazrqbpmflkxdbdxdfkpgvlrxkaqlyofybbqbjqlpbosbvlrdfsbbqbjfjkpqrzqlfklpxkaxzobcLoqebjqrpalryibaxdbkqpxob
zorfqbaxkarphaprkawrqebxqlctxo
Test Key: gex Plain Text: epeaoaoajpewhpkooagkpaajaiuwcajposdkdwaraykiapkykzqypaolekjcawcwejpoukqwjzpkxnexapdaipkoanrauqkqerapdaiejpqnqyepkewjowjzywnabknpdapdqozkqxhawcajpowwana
yngepazwjzqaozqjpvqpdawnpkbswn
Test Key: hfy Plain Text: dodnznziodvgojnzjzfpjoizhtvzionrcjcvqzjhzojxiyypoxznkdjivzbvbdinotjpviojwmdwzoczhoinzmqtzjpbdqzoczhdinompdxindjinvixvmzajmoczhocpnyjpwgzyvzbzionvmz
xmpdozyvzpnznypiooupczvmojarm
Test Key: igz Plain Text: enemymmyhneufinmyeionhygsuayhnmqibupuyigyniwhxownymjcihuayuauchmnsiohxnivlcynbygnimylpsioacpynbygchmnlowncihmuhxwlyzlinbygnbmoxiovfxyuayhnmuly
ylcoonyuhxoaemyahntonbylnizql
Test Key: jha Plain Text: bmbllxlxgmbtemhxxdhnmqxftrtzmplahatoxvfhxmhvhgwnvmlxibhgtzxtzbtglmrhtgmhukbuxmaxfmlhkkoxrhnbboxmaxfbglmknvmbhgljgwvtkxyhkmaxfmanlwhnuextzxmgtkxx
vknbmxtgwnlxlngsmnmaxtkmhytk
Test Key: kib Plain Text: alakwkkwflasdglkwmcmglfwesqyfwflkcozgszwnugewlguflmwulmhagfsywsyafklqgmsfvlgatjatlzwlwglkwjnwqgmyanlwzweafkljmlagfksfvusjwsgjzwlzmkvgmdwsvwflksjw
ujnalwsvfwmkvkmflmlzwsjlgxosj
Test Key: lij Plain Text: zksjvjvjkzrcfkfjvvbflkvevdprxekjnyfymvtdfkvtfeultkvjgzferxvrzrzejkpflreukfsizsvkyvdkfjvimvflxzmvyvdzejkiltkzfejreutrivwfkvydkyljuflscvurxvekjrivi
tilzkvureaujvujlekqlkyvrlkfmrzi
Test Key: mld Plain Text: vjyjuuiudjyqbjeiuaekjducoquwdjinxexqlusecujesedtskjuifyedqmwqygdijoeqkdjjerhyrujxucjeiuhluoekwylujxucydijhksjyediqdtsqhuvehjxucjxitekrburtqwdjiqhu
shkyjutqdtikutidkpkjxuhqjwemgh
Test Key: nle Plain Text: xixthhtcixpaidhttdjtictbnptcihlwdmpktrdbtidrdsjrithehexdcpvtpvxchindjpcsdqgxtiwtbidhtgktdnvxktiwtbchigjrixdchpcsrpgtgdgiwtbiwhsdjatspvtcihpgtg
rgjxitspsjhtshjciojwtpgidulpg
Test Key: omf Plain Text: nhpsggssbhwozhcgssycihsbsamoushbgkvcvojsqaschecqbriqhsdwbousouowbghmciobrhcpfwpshsvahcgsfjsmciuwjshvsambghfiqhwcgbogrbqofstcfhsahvigrpcipsroushgofsf
zfihhsrobisgsgibhnihszofhtkof
Test Key: png Plain Text: vgvfrffragvnygbfrrxbhgrarzlntraqfjubunirpbzrbgpbahqpgpfvcbvbantrmtvafglbhnaagboevorgurzgbfreirlbhtvirgurzvafgehpvgbafnaqpnrsbegurzgufghoyzqntragnrer
behvgrqnaqhfrqfhagmhgurnegbsjne
Test Key: qoh Plain Text: ufueqeeqzfmuxfaeqwagqzqyqmsqzfeitahqoaqyfaaoazpgogfgebuaazmsmsuzefkagmzpfandunqftqyfaeqdhqagsuhqftqyuzefdgofuazemzpondqradftqyftgepagnxqmsqzfeindq
odgufqpmzpggeqpegzflgftandfaimd
Test Key: rpi Plain Text: tetdpddpyetlwezdpvpzfepypxjlrpyedhszlgpnzpeznzyofnepdatzylrlrltydejzflyoetznmpespezdpcgpjzfrtgespxtydecfnetzdylyonlcpqzcespesfdozfmpolrpyedlcp
neftepolyofdpodfyekflespzezhlc
Test Key: sqj Plain Text: xdsccccoxdskvdycoouyedoxwikqoxdcgryrkfomywodymxnendocszxkqkqksxcdiyeikndylbslodrowdycobfoieqsfodrowxscdbemdsyckxnmkboypbdrowdrecnyelvonkqoxdckbob
mbesdonkxneconcedjedrokbdyogkb
Test Key: trk Plain Text: rcrbnbbnrcrjucbnntdcnwnvhjpnwcbfqxjenlvnvcxlwmclcnbyrxwnjpnjrbwchxdjwmcckarkncqncvxbnaenhxdrpencqnrwbacdlcrxwbjwmljanoxacqncvqdbmxdkunmjpncwbjanan
ladrcnmjmdnmbdbwdicqnjacxofja
Test Key: usl Plain Text: qbgamaamvqitbwamswcbnmvguionvbaepwidmkwmbkwvlckbmaxqvwionioiqvabgwcivlbwjzqjmbpmubwanzdmgwoqdmqmuqavabzckbqwaivlkizmnwzbpmbpcalwctjtlionvbaizm
zqcbmlvlcanlaevbhcbpizbwmeiz
Test Key: vtm Plain Text: papzllzluaphsavzllrvbalultfhnluazdovohcljvtlavjvukbjalzwpuvhnlnhnpuzafvbkukaviypilaoltavzlyclfvbnpclaoltupazybjapvuzhukjlylmvyaoltaobzkbvislkhlnluazhyly
jybpalkhukbzlkzbuagbaohlyavndhy
Test Key: wun Plain Text: ozyokykyktzogrzykkquazktksegmtkzycnucngbkiuskziutjaizkyvoutgmkgmotyzeuagtjzuhxohkznkszykxbkeuanobkznksotyzaizoutygtjigxkluxznksznayjuahrkjgmktzygkxk
ixaozkjgtjykyatzfzknkgzclcgx
Test Key: xvo Plain Text: nynxjxxjsynfaytzjptzyjsjrdfljsyxbmtmfajhtrjyhtsizhyjxuntsfjflfnsxydzfsiytgwngjymjrytxjwajdtzlnajymjrnsxwzhytnxsfsihfwjktwmyjrmxiztgqjifljsyxfwj
wnzyjlfisizjixzsyzejmfwlbtw
Test Key: ywp Plain Text: nxmimwirmexpswiosyxiriquekirxwalslezigsqixsgrhyxiwmsrekiekemwxcserhxsfwfxflixswivzicsykmzixliqmrwvvgxmsrwerhegijsvxlqlywhsyfpihekirxwvivi
gyvmxiheryhiwrydxliexvsaev
Test Key: zqx Plain Text: lulvhvhwldowrvhhrxwhqhpbdjhqmwzkrkdyhrfphwrfrqgxfwhslrqddjhdjldqwbxrdqgurelewkhkpwrvuhybrxjlyhwhkhlplvwxwflrqlqddgdudhruwkhkpwvgrxeohgdjhwquduh
```

As is shown above, key = "cat" gives meaningful and correct result: it is essential to ...

2.4 Source Code (C++)

First part (Determining Key Length by Coincidence Index Method):

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7      string cipher_text = "ktbueluegv...kvoyyak";
8
9      vector<int> key_lengths = {2, 3, 4, 5, 6, 7, 8};
10
11     for(auto key_len : key_lengths) {
12         vector<string> texts(key_len, "");
13         vector<double> prob = {};
14
15         for (int i = 0; i < cipher_text.length(); i++) {
16             texts[i % key_len] += cipher_text[i];
17         }
18
19         for (auto text : texts) {
20             vector<int> frequency(26, 0);
21
22             for (auto chr : text) {
23                 frequency[chr - 'a']++;
24             }
25
26             double probability = 0;
27
28             for (auto freq : frequency) {
29                 probability += freq * (freq - 1);
30             }
31
32             int n = text.length();
33             probability = probability / (n * (n - 1));
34             prob.push_back(probability);
35         }
36
37         cout << "Testing with key length = " << key_len << ": p = ";
```

```

38     double average = 0;
39     for(auto p : prob) {
40         cout << p << " ";
41         average += p;
42     }
43
44     cout << ", with average=" << average / key_len << endl
45         ;
46 }

```

Second part (Determine the Relative Offset by Mutual Coincidence Index Method):

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  double calculate_MIC(vector<string>& C, int i, int j, int k);
7
8  int main() {
9      string cipher_text = "ktbueluegvit...gakvoyyak";
10     vector<string> C(3, "");
11
12     for (int i = 0; i < cipher_text.length(); i++) {
13         C[i % 3] += cipher_text[i];
14     }
15
16     vector<vector<vector<double>>> MIC(3, vector<vector<double>
17         >>(3, vector<double>(26, 0)));
18
19     double max_MIC;
20     int max_k;
21     for (int i = 0; i < 3; i++) {
22         for (int j = 0; j < 3; j++) {
23             if (i >= j)
24                 continue;
25
26             max_MIC = 0;
27             max_k = 0;
28
29             for (int k = 0; k < 26; k++) {

```

```

29         MIC[i][j][k] = calculate_MIC(C, i, j, k);
30         if (MIC[i][j][k] > max_MIC) {
31             max_MIC = MIC[i][j][k];
32             max_k = k;
33         }
34     }
35
36     cout << "MIC(" << i << "," << j << ")_=" << max_MIC
37         << "_(offset=" << max_k << ")" << endl;
38 }
39 }
40
41 double calculate_MIC(vector<string>& C, int i, int j, int k) {
42     double MIC = 0;
43
44     string Si = C[i];
45     string Sj = C[j];
46
47     vector<double> freq_i(26, 0), freq_j(26, 0);
48
49     for (int l = 0; l < Si.length(); l++) {
50         freq_i[Si[l] - 'a']++;
51     }
52
53     for (int l = 0; l < Sj.length(); l++) {
54         freq_j[Sj[l] - 'a']++;
55     }
56
57     for (int i = 0; i < 26; i++) {
58         freq_i[i] = freq_i[i] * 1.0 / Si.length();
59         freq_j[i] = freq_j[i] * 1.0 / Sj.length();
60     }
61
62     for(int l = 0; l < 26; l++) {
63         int index = (l + k) % 26;
64         MIC += freq_i[l] * freq_j[index];
65     }
66
67     return MIC;
68 }

```

Third part (Determining the Key):

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  int main() {
7      string cipher_text = "ktbueluegvitn...akvoyyak";
8
9      int offset_0_1 = 2, offset_1_2 = 7;
10     string init_key = "ayr";
11     string test_key, plain_text;
12
13     for (int i = 0; i < 26;i++) {
14         test_key = init_key;
15         for (int j = 0; j < 3;j++) {
16             test_key[j] = (test_key[j] - 'a' + i) % 26 + 'a';
17         }
18
19         plain_text = cipher_text;
20         for (int j = 0;j<cipher_text.length();j++) {
21             char c = test_key[j % 3];
22             plain_text[j] = cipher_text[j] - c + 'a';
23             if(plain_text[j] > 'z') {
24                 plain_text[j] -= 26;
25             }
26             if(plain_text[j] < 'a') {
27                 plain_text[j] += 26;
28             }
29         }
30
31         cout << "Test_Key:_" << test_key << " _Plain_Text:_" <<
            plain_text << endl;
32     }
33 }
```

3 Unknown Cipher

3.1 Plaintext and Key

- Cipher text: MAL TIRRUEZF CR MAL RKZYIOL EX MAL OIY UAE RICF "MAL ACWALRM DYEUPFLWL CR ME DYEU MAIM UL IZL RKZZEKYFLF GH OHRMLZH"
- Key: A dictionary shown below ('M'→'T', 'A'→'H', 'L'→'E'...)
- Plain text: THE PASSWORD IS THE SURNAME OF THE MAN WHO SAID "THE HIGHEST KNOWLEDGE IS TO KNOW THAT WE ARE SURROUNDED BY MYSTERY"

3.2 Cryptanalysis Process

The unknown cipher is different from the Caesar and Vigenere cipher, since we have no idea of what kind of cipher it is. However, a significant feature of this cipher is that **words are separated by space**, so each word must correspond to an existing English word, and the encryption method should be character replacement.

To begin with, firstly I noticed that MAL appears 3 times in the first sentence. The only possible 3-character word in English is THE, so some corresponding relationship can be guessed: 'M'→'T', 'A'→'H', 'L'→'E'.

Then notice that there is a word MAIM. After replacing known characters we have TH?T. The only possible word is THAT, so 'I' corresponds to 'A'. Similarly word ME gives T?. Guess that TO may be the correct answer, so 'E'→'O'.

Next I focus on the word UAE (after replacing is ?HO), obviously WHO is the answer, so 'U'→'W'. Same for IZL we can know 'Z'→'R'.

To continue with, there is a word EX, which is O? after replacing existing. There are 2 possible words OF and OR, since 'R' for 'Z' is determined, 'X' can only be 'F'.

Next, focusing on only one word can not solve any characters. So I focus on RICF+CR. A common two-character word that appears at the middle of sentence is IS and a common word that appears before a quote is SAID, so I tried 'C'→'I' and 'R'→'S', and it worked! Verify the guess on RICF we can get 'F' corresponds to 'D', thus RICF is SAID.

Finally, having many corresponding relationships, we can solve the remaining unknown characters using unknown words, like PASSWORD for TIRRUEZF...

3.3 Source Code (Python)

```
1 cipher_text = "MAL_TIRRUEZF_CR_MAL_RKZYIOL_EX_MAL_OIY_UAE_RICF_"
    MAL ACWALRM DYEUPFLWL CR ME DYEU MAIM UL IZL RKZZEKYFLF GH
    OHRMLZH"
2
3 key_dict = {
4     'M': 'T',
```

```

5      'A': 'H',
6      'L': 'E',
7      'I': 'A',
8      'E': 'O',
9      'U': 'W',
10     'Z': 'R',
11     'X': 'F',
12     'R': 'S',
13     'C': 'I',
14     'F': 'D',
15     'T': 'P',
16     'W': 'G',
17     'K': 'U',
18     'Y': 'N',
19     'O': 'M',
20     'D': 'K',
21     'P': 'L',
22     'H': 'Y'
23 }
24
25 plain_text = ""
26 for char in cipher_text:
27     if char in key_dict.keys():
28         plain_text += key_dict[char]
29     else:
30         plain_text += char
31
32 print(plain_text)

```