Principles of Information Security

Assignment 1

Password Cracking

Anonymous

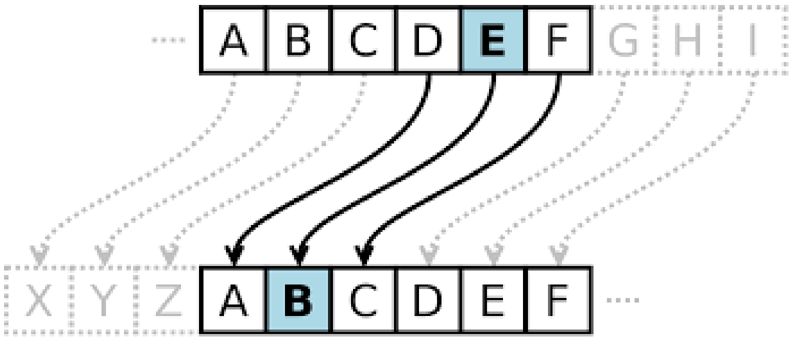
February 26, 2025

Contents

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: `FBUQIUUDSHOFJOEKHDQCUMYJXJXUIQCUAUOQDTKFBEQTJEBUQHDYDWYDPZK`
- 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: GCVRJVVETIPGKPFLIERDVNZKYKYVJRDVBVPREULGCFRUKFCVRIEZEXZEQAL
Possible plain text with offset = 2: HDWSKWWFUJQHLQGMJFSEWOALZLZWKSEWCWQSFVMHDGSVLGDWSJFAFYAFRBM
Possible plain text with offset = 3: IEXTLXXGVKRIMRHNKGTFXPBMAMAXLTFXDXRTGWNIEHTWMHEXTKGBGZBGSCN
Possible plain text with offset = 4: JFYUMYYHWLSJNSIOLHUGYQCNBNBYMUGYEYSUHXOJFIUXNIFYULHCHACHTDO
Possible plain text with offset = 5: KGZVNZZIXMTKOTJPMIVHZRDOCOCZNVHZFZTVIYPKGJVYOJGZVMIDIBDIUEP
Possible plain text with offset = 6: LHAWOAAJYNULPUKQNJWIASEPDPDAOWIAGAUWJZQLHKWZPKHAWNJEJCEJVFQ
Possible plain text with offset = 7: MIBXPBBKZOVMQVLROKXJBTFQEQEBPXJBHBVXKARMILXAQLIBXOKFKDFKWGR
Possible plain text with offset = 8: NJCYQCCLAPWNRWMSPLYKCUGRFRFCQYKCICWYLBSNJMYBRMJCYPLGLEGLXHS
Possible plain text with offset = 9: OKDZRDDMBQXOSXNTQMZLDVHSGSGDRZLDJDXZMCTOKNZCSNKDZQMHMFHMYIT
Possible plain text with offset = 10: PLEASEENCRYPTYOURNAMEWITHTHESAMEKEYANDUPLOADTOLEARNINGINZJU
Possible plain text with offset = 11: QMFBTFFODSZQUZPVSOBNFXJUIUIFTBNFLFZBOEVQMPBEUPMFBSOJOHJOAKV
Possible plain text with offset = 12: RNGCUGGPETARVAQWTPCOGYKVJVJGUCOGMGACPFWRNQCFVQNGCTPKPIKPBLW
Possible plain text with offset = 13: SOHDVHHQFUBSWBRXUQDPHZLWKWKHVDPHNHBDQGXSORDGWROHDUQLQJLQCMX
Possible plain text with offset = 14: TPIEWIIRGVCTXCSYVREQIAMXLXLIWEQIOICERHYTPSEHXSPIEVRMRKMRDNY
Possible plain text with offset = 15: UQJFXJJSHWDUYDTZWSFRJBNYMYMJXFRJPJDFSIZUQTFIYTQJFWSNSLNSEOZ
Possible plain text with offset = 16: VRKGYKKTIXEVZEUAXTGSKCOZNZNKYGSKQKEGTJAVRUGJZURKGXTOTMOTFPA
Possible plain text with offset = 17: WSLHZLLUJYFWAFVBYUHTLDPAOAOLZHTLRLFHUKBWSVHKAVSLHYUPUNPUGQB
Possible plain text with offset = 18: XTMIAMMVKZGXBGWCZVIUMEQBPBPMAIUMSMGIVLCXTWILBWTMIZVQVOQVHRC
Possible plain text with offset = 19: YUNJBNNWLAHYCHXDAWJVNFRCQCQNBJVNTNHJWMDYUXJMCXUNJAWRWPRWISD
Possible plain text with offset = 20: ZVOKCOOXMBIZDIYEBXKWOGSDRDROCKWOUOIKXNEZVYKNDYVOKBXSXQSXJTE
Possible plain text with offset = 21: AWPLDPPYNCJAEJZFCYLXPHTESESPDLXPVPJLYOFAWZLOEZWPLCYTYRTYKUF
Possible plain text with offset = 22: BXQMEQQZODKBFKAGDZMYQIUFTFTQEMYQWQKMZPGBXAMPFAXQMDZUZSUZLVG
Possible plain text with offset = 23: CYRNFRRAPELCGLBHEANZRJVGUGURFNZRXRLNAQHCYBNQGBYRNEAVATVAMWH
Possible plain text with offset = 24: DZSOGSSBQFMDHMCIFBOASKWHVHVSGOASYSMOBRIDZCORHCZSOFBWBUWBNXI
Possible plain text with offset = 25: EATPHTTCRGNEINDJGCPBTLXIWIWTHPBTZTNPCSJEADPSIDATPGCXCVXCOYJ

Press any key to continue . . .
```

As is shown in the figure above, 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++)

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
```

```
    string cipher_text = "FBUQIUUDSHOFJOEKHDQCUMYJXJXUIQCUAUOQDTKFBEQTJEBUQHDYDWYDPZK";
    int offset;

    for (offset = 1; offset <= 25; ++offset) {
        string temp_string = cipher_text;
        for(auto& chr : temp_string) {
            if (chr + offset <= 'Z') {
                chr += offset;
            } else {
                chr = chr + offset - 26;
            }
        }

        cout << "Possible␣plain␣text␣with␣offset␣=␣" << offset << ":␣" << temp_string << endl;
    }
}
```

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 abpsmaoncnwvoutiugtagmmqsxtvxaoniiogtagmbpsmtuvvihpstpdvcrxhokvhxotawswquun ewcgxptlcrxtevtubvewcnwwsxfsnptswtagakvoyyak`
- Key: **cat**
- Plain text: `it is essential to seek out enemy agents who have come to conduct`

--PLAINTEXT--

KEY	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
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:

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

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:

---

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

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

where $\text{freq}_i(l)$ and $\text{freq}_j(l)$ are the frequencies of letter l in groups i and j, respectively.
      if MIC(i, j, k) > max_MIC **then**
        Update $max\_MIC \leftarrow MIC(i,j,k)$
        Update $max\_k \leftarrow k$
      end if
    end for
    Output $MIC(i,j)$ and the corresponding shift max_k
  end for
  The relative offsets between the groups are identified by the shift k with the highest MIC for each pair of groups.
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

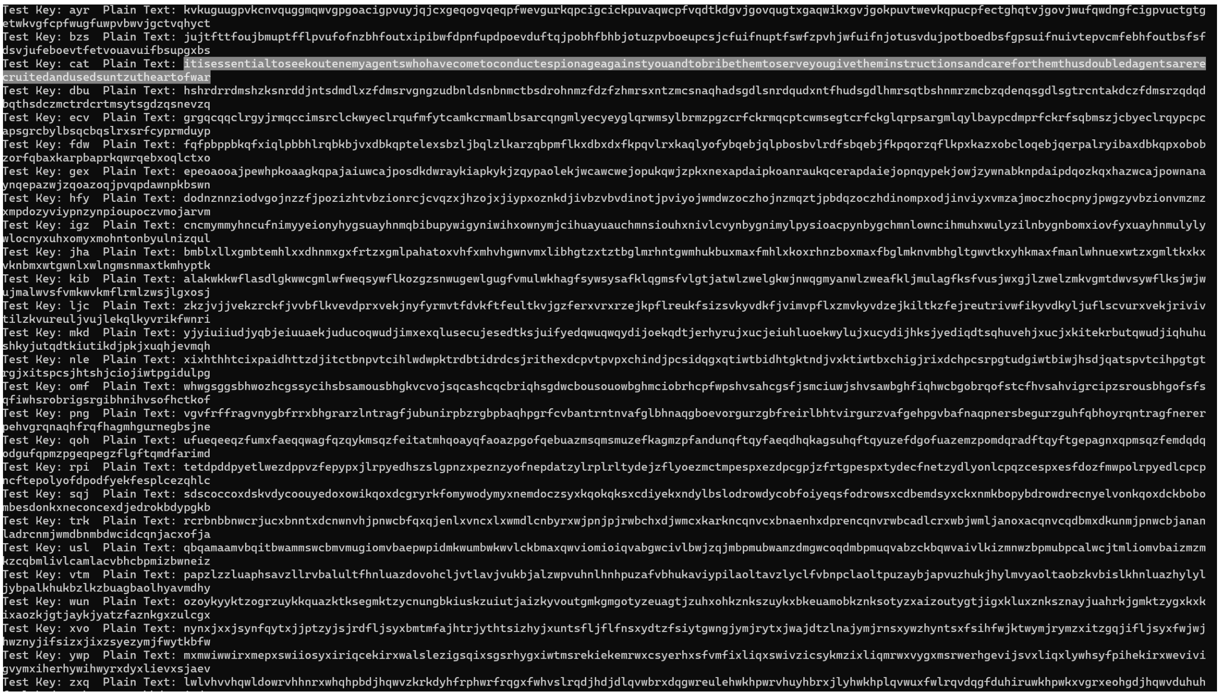
After determining the relative offsets between the groups, the final step is to deduce the key. The key can be reconstructed by combining the determined offsets and aligning them with the expected shifts of the alphabet. There are only 26 candidates, so enumeration is feasible to get the final solution.

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:



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):

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
    string cipher_text = "ktbueluegv...kvoyyak";

    vector<int> key_lengths = {2, 3, 4, 5, 6, 7, 8};

    for(auto key_len : key_lengths) {
        vector<string> texts(key_len, "");
        vector<double> prob = {};

        for (int i = 0; i < cipher_text.length(); i++) {
            texts[i % key_len] += cipher_text[i];
        }

        for (auto text : texts) {
            vector<int> frequency(26, 0);

            for (auto chr : text) {
                frequency[chr - 'a']++;
            }

            double probability = 0;

            for (auto freq : frequency) {
                probability += freq * (freq - 1);
            }

            int n = text.length();
            probability = probability / (n * (n - 1));
            prob.push_back(probability);
        }

        cout << "Testing␣with␣key␣length␣=␣" << key_len << ":␣p␣=␣";
```

```cpp
        double average = 0;
        for(auto p : prob) {
            cout << p << "␣";
            average += p;
        }

        cout << ",␣with␣average␣=␣" << average / key_len << endl;
    }
}
```

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

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

double calculate_MIC(vector<string>& C, int i, int j, int k);

int main() {
    string cipher_text = "ktbueluegvit...gakvoyyak";
    vector<string> C(3, "");

    for (int i = 0; i < cipher_text.length(); i++) {
        C[i % 3] += cipher_text[i];
    }

    vector<vector<vector<double>>> MIC(3, vector<vector<double>>(3, vector<double>(26, 0)));

    double max_MIC;
    int max_k;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (i >= j)
                continue;

            max_MIC = 0;
            max_k = 0;

            for (int k = 0; k < 26; k++) {
```

```
                MIC[i][j][k] = calculate_MIC(C, i, j, k);
                if (MIC[i][j][k] > max_MIC) {
                    max_MIC = MIC[i][j][k];
                    max_k = k;
                }
            }

            cout << "MIC(" << i << "," << j << ") = " << max_MIC
                << " (offset = " << max_k << ")" << endl;
        }
    }
}

double calculate_MIC(vector<string>& C, int i, int j, int k) {
    double MIC = 0;

    string Si = C[i];
    string Sj = C[j];

    vector<double> freq_i(26, 0), freq_j(26, 0);

    for (int l = 0; l < Si.length(); l++) {
        freq_i[Si[l] - 'a']++;
    }

    for (int l = 0; l < Sj.length(); l++) {
        freq_j[Sj[l] - 'a']++;
    }

    for (int i = 0; i < 26; i++) {
        freq_i[i] = freq_i[i] * 1.0 / Si.length();
        freq_j[i] = freq_j[i] * 1.0 / Sj.length();
    }

    for(int l = 0; l < 26; l++) {
        int index = (l + k) % 26;
        MIC += freq_i[l] * freq_j[index];
    }

    return MIC;
}
```

Third part (Determining the Key):

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
    string cipher_text = "ktbueluegvitn...akvoyyak";

    int offset_0_1 = 2, offset_1_2 = 7;
    string init_key = "ayr";
    string test_key, plain_text;

    for (int i = 0; i < 26;i++) {
        test_key = init_key;
        for (int j = 0; j < 3;j++) {
            test_key[j] = (test_key[j] - 'a' + i) % 26 + 'a';
        }

        plain_text = cipher_text;
        for (int j = 0;j<cipher_text.length();j++) {
            char c = test_key[j % 3];
            plain_text[j] = cipher_text[j] - c + 'a';
            if(plain_text[j] > 'z') {
                plain_text[j] -= 26;
            }
            if(plain_text[j] < 'a') {
                plain_text[j] += 26;
            }
        }

        cout << "Test Key: " << test_key << "  Plain Text: " << plain_text << endl;
    }
}
```

3 Unknown Cipher

3.1 Plaintext and Key

- Cipher text: `MAL TIRRUEZF CR MAL RKZYIOL EX MAL OIY UAE RICF "MAL ACWALRM DYEUPLFWL 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 K NOWLEDGE IS TO KNOW THAT WE ARE SURROUNDED GY 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)

```
cipher_text = "MAL␣TIRRUEZF␣CR␣MAL␣RKZYIOL␣EX␣MAL␣OIY␣UAE␣RICF␣"MAL ACWALRM DYEUPLFWL CR ME DYEU MAIM UL IZL RKZZEKYFLF GH OHRMLZH""

key_dict = {
    'M': 'T',
```

```
    'A': 'H',
    'L': 'E',
    'I': 'A',
    'E': 'O',
    'U': 'W',
    'Z': 'R',
    'X': 'F',
    'R': 'S',
    'C': 'I',
    'F': 'D',
    'T': 'P',
    'W': 'G',
    'K': 'U',
    'Y': 'N',
    'O': 'M',
    'D': 'K',
    'P': 'L',
    'H': 'Y'
}

plain_text = ""
for char in cipher_text:
    if char in key_dict.keys():
        plain_text += key_dict[char]
    else:
        plain_text += char

print(plain_text)
```