

Cambridge University Engineering Department Part IB Computing Project - Vigenere Decipher

Matt Galloway - Pembroke College
UID: mjpg91

February 27, 2005

Abstract

A task was set to decipher a long string which was known to have been coded using a Vigenere cipher but of unknown key. The key length is known to be between 5 and 9 characters long and is an English word.

To solve the task a C++ program was written which finds the key and deciphered message and then outputs these to the console. The program ran successfully and the key was found to be **CRUNCH** and the message a passage from Shakespeare.

Contents

1	Problem Statement	3
2	Approach	4
2.1	Initial Design	4
2.2	Implementation	4
2.3	Improvements	5
3	Running the program	6
4	Conclusions & Comments	6
A	Program Flow Chart	7
B	Source Code	8

This document has been prepared using LaTeX with the following packages:
Fullpage, Fancyhdr, Graphicx, Verbatim, Listings

1 Problem Statement

The following was taken direct from the CMI Tutor page and describes the problem I was faced with.

The following piece of text has been encrypted using a Vigenere Cipher. The codeword is not known, except that it is an English word between 5 and 9 letters long.

Write a program in C++ which will decrypt the message, and work out the codeword.

```
UYUYNPEFGCCYGKBRGAQRMHOTGIMQCFVYIHCYVDIEGSQMYAHPUGBTLVVGCGYCKYEQ
BIYQVPKUUIFJHMVNUGKCIFVPNDLXFQMORSNPKULGZGYUCYNULJRNUCSNKIBUOQINN
FHVVBOLVZGRVVQYIGVOGVSQRMJVUIGUUYCAGZCEXBHAGECFJPUXIYFJQDJYGEKFFH
QKTOVXNPKGMYEAMCZLSTVOWUVTZQDYGKTGUYPNPPVMOAJJRHPGVTEUGWYGJWUCUIZ
HTEVWIMRWUVICZOLFSOGVOAVNRTUCCMHOTGIMUCSNEIGHHFVHBTSQJYCQZUVMFKVP
FZGJHVWUVTAJFOBYZVEIEUOCCFQGHVYVECNVYIHYHPUYEUAKEBVUZJRXRYOGECAGA
GIHNSKEYFVVVZGRVOQLAEQDUKMBNVPXUFOLPTUADYGRNUGVTVSRUJCEMRGZQCIAI
SKMYFVOKJUAFAJZMTCGJFVHLVFNUGL
```

2 Approach

2.1 Initial Design

When I first looked at the problem I decided it would be a good idea to work out my overall strategy. I first decided how reusable my program would be and as the task only required one message to be deciphered I decided that I would put the ciphered message directly into the source code and also set the limits on the length of the key in the source code.

I knew that I would be representing the characters by numbers at some point so I had to have a numbering scheme. I decided to use A=0, B=1 ... Z=25 where there is no distinction between upper & lower case letters and punctuation & white space is ignored.

Now I had to decide how the program would go about cracking the key and deciphering the message. I decided that the actual problem here is finding the key as deciphering the message once the key has been found is simple. So my program would somehow try combinations of keys until it found one which made sense. I decided to use the statistical data of typical frequencies of letters used in the English language to compare subsets of the letters in the ciphered message. From this statistical analysis I could devise a scoring system to rate deciphering attempts. I then drew a flow chart (see Appendix A) which outlines the overall strategy I decided upon.

So the first function I would need is one which an array (or at least pointer to an array) and an integer representing the rotations to do on each element of the array are passed and then a score for that array is returned which will be a float. This function is very useful because it can be used to get the score on any array I choose to give it. For example a subset of the ciphered string or a complete deciphered string.

In the desire for reusability of code I decided to also have a function to find the best key for a given key length. This function should return nothing (i.e. void) but a pointer to an array passed to it would be filled with the rotations required to form the key.

I also decided to include a decrypt function which is passed two pointers to arrays; one a `char` array and one an `int` array. It is also passed the keylength and then decipheres the ciphered string and stores the result in the char array it was passed.

A key part of my strategy was to have the complete program automated as I wanted no human input. Therefore my program should call the frequency scoring function each time it decipheres the message using a certain key and then should pick the key which gave the lowest deciphered score.

I also decided I would quite often be converting from ascii to integers and rotating an integer value, I decided that I would put the code for these two tasks into functions. So overall I had the following prototypes:

```
int  ascii2int(char);
float freqscore(char*,int);
char rotate(char, int);
void getkey(int*,int);
void decrypt(char*,int*,int);
```

2.2 Implementation

Please refer to my C++ source code (see Appendix B)

Now that I had the general strategy and function prototypes worked out I had to write the functions and tie them all together in the main function. I started by choosing to include the `iostream` and `cmath` libraries for their obvious reasons. I have omitted the `using namespace std` declaration and instead opted for the explicit separate `using` statements because including the complete `std` namespace defeats the point of C++ namespaces.

I used `#define` to set the pre-processor variables `MIN_KEY_LENGTH` & `MAX_KEY_LENGTH` and I set the variable `static char ciphered[]` outside `int main()` so that it was in the scope of the whole program. This is desirable so that any function can access the ciphered string without having to pass it each time.

The functions `freqscore`, `getkey` and `decrypt` do the main work of the program. `freqscore` rotates each character in the array it is passed by an amount which is also passed to the function and then it calculates a score by adding up the differences between the frequencies of each letter in the array and the expected values. `getkey` forms the best key for a given key length and utilises the `freqscore` function to do this. It finds the best rotation for each letter of the key by finding the rotation which gives the best score from `freqscore`. A subset is taken of the ciphered message by getting the elements; 0, 0+keylength, 0+2*keylength, etc for the first character of the key, and then the elements; 1, 1+keylength, 1+2*keylength, etc for the second character of the key until this has been done for all characters of the key. `decrypt` takes the ciphered array and decipheres it using the key passed to it.

The other two functions, `ascii2int` and `rotate` are self explanatory and are to make the code easier to read and hence debug.

Inside `int main()` is the drawing together of the complete decipher program. This code simply loops through the various key lengths and then decides on the best key and outputs the deciphered message.

This is how I implemented my design and I didn't make any major changes to the initial design.

2.3 Improvements

As this task only required the deciphering of one message I put the ciphered message and key length range variables in the source code. However I would consider altering the code to take in the ciphered message and key length range from either a file whose name is passed to the program or from direct piping to the program. These modifications would be relatively simple to implement and would allow the program to be used on more messages.

3 Running the program

The following text shows the *exact* output from my compiled and run C++ program. As you will be able to see, the keyword is CRUNCH and the deciphered message is a passage from Shakespeare.

```
=====
mjpg91's Vigenere Decipher
=====

=====
Original message
=====
UYUYNPEFGCCYKBRGAQRMHOTGIMQCFVYIHCYVDIEGSQMYAHPUGBTLVVGCGYCKYEQ
BIYQVPKUUIFJHMVNUGKCIFVPNDLXFQMORSNPKULGZGYUCYNULJRNUCSNKIBUOQINN
FHVVBOLVZGRVVQYIGVOGVSRQMJVUIGUUYCAGZCEXBHAGECFJPUXIYFJQDJYGEKFFH
KQTOVXNPKGMYEAMCZLSTVOWVVTZQDYGKTGUYPNPPVMOAJJRHPGVTEUGWYGJWUCUIZ
HTEVWIMRWUVICZOLFSOGVOAVNRTUCCMHOTGIMUCSNEIGHHFVHBTSQJYCQZUVMFKVP
FZGJHVWUVTAFJOFYZVEIEUOCCFQGHVYVECNVYIHYHPUYEUAKEBVUZJRXRYOGECAGA
GIHNNSKEYFVVVZGRVOQLAEQDUKMBNVPXUFOLPTUADYGRNUGVTVSRUJCEMRGZQCI AI
SKMYFVOKJUAF AJZMTKCGJFVHLVFNUGL
=====

=====
Keylength: 5, keyword: RRNCN
Keylength: 6, keyword: CRUNCH
Keylength: 7, keyword: NVCRRNV
Keylength: 8, keyword: CUHCRCR
Keylength: 9, keyword: CCINRHNRU
I have chosen the key: CRUNCH
=====

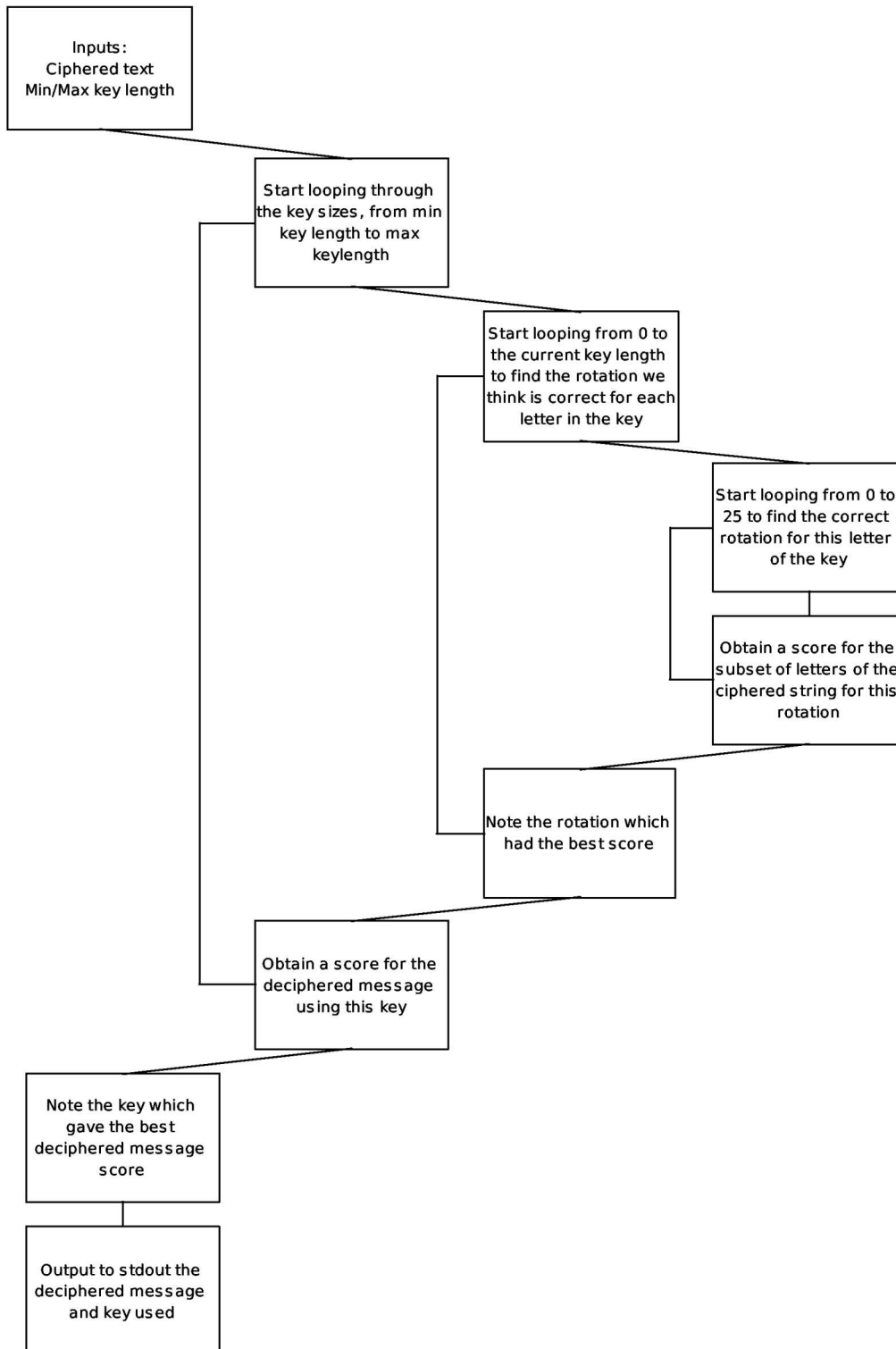
=====
Deciphered message
=====
SHALLICOMPARETHEETOASUMMERSDAYTHOUARTMORELOVELYANDMORETEMPERATERO
UGHWINDSDOSHAKETHEDARLINGBUDSOFMAYANDSUMMERSLEASEHATHALLTOOSHORTA
DATESOMETIMETOOTHOTTHEEYEOFHEAVENSHINESANDOFTENISHISGOLDCOMPLEXION
DIMMEDANDEVERYFAIRFROMFAIRSOMETIMEDECLINESBYCHANCEORNATURESCHANGI
NGCOURSEUNTRIMMEDBUTTHYETERNALSUMMERSHALLNOTFADENORLOSEPOSSESSION
OFTHATFAIRTHOUOWSTNORSHALLDEATHBRAGTHOUWANDERSTINHISHADEWHENINET
ERNALLINESTOTIMETHOUGROWSTSOLONGASMENCANBREATHEOREYESCANSEESOLONG
LIVESTHISANDTHISGIVESLIFETOTHEE
=====
```

I also ran my program on a message which I made up and ciphered. The message was 150 characters long and the key was 5 characters long. I found that I could also decipher this which was good to see.

4 Conclusions & Comments

From this task I have discovered what a Vigenere cipher is and how to decipher one. But I have also learned that computers can be very useful in cryptography and that with the exponential increase in computing power which we are seeing there is no doubt that encryption techniques will have to become more and more robust in order to avoid decryption by the wrong person.

A Program Flow Chart



B Source Code

```
#include <iostream>
using std::cout;
using std::endl;

#define MIN_KEY_LENGTH 5
#define MAX_KEY_LENGTH 9

float proper_freqs[26]={8.167, 1.492, 2.782, 4.253, 12.702, 2.228, 2.015, 6.094,
                      6.996, 0.153, 0.772, 4.025, 2.406, 6.749, 7.507, 1.929,
                      0.095, 5.987, 6.327, 9.056, 2.758, 0.978, 2.360, 0.150,
                      1.974, 0.074};

char ciphred []="UYUYNPEFGCCYKBRGAQRMHOTGIMQCFVYIHCYVDIEGSQMYAHPUGBTLVVGCGYCKY
EQBIYQVPKUUIFJHMVNUGKCIFVPNDLXFQMORSNPKULGZGYUCYNULJRNUSNKIBUOQINNPHVMBOLVZGRV
VQYIGVOGVSRQMJVUIGUUYCAGZCEXBHAGECFJPUXIYFJQDJYGEKFKTKTOVXNPKGMYEAMCZLSTVOWUVTZQ
DYGKTGUYPNPPVMOAJJRHPGVTEUGWYGJWUCUIZHTEVWIMRWUVICZOLFSOGVOAVNRTUCCMHOTGIMUCSNEI
GHHFVHBSQJYCQZUVMFKVPFZGJHVWUVTAJFOBYZVEIEUOCCFQGHVYVECNVYIHYHPUYEUAKEBVUZJRXRY
OGECAGAGIHNSKEYFVVVZGRVOQLAEQDUKMBNVPXUFOLPTUADYGRNUGVTVSRUJCEMRGZQCIASIKMYFVOK
JUAF AJZMTKCGJFVHLVFNUGL";
int length=strlen(ciphred);

int  ascii2int(char);
float freqscore(char*,int);
char rotate(char, int);
void  getkey(int*,int);
void  decrypt(char*,int*,int);

int main() {
    /* Initialisation stage */
    cout << "=====" << endl;
    cout << "mjpg91's Vigenere Decipher" << endl;
    cout << "=====" << endl << endl;

    cout << "=====" << endl;
    cout << "Original message" << endl;
    cout << "=====" << endl;
    for(int i=0; i<length; i++) {
        cout << ciphred[i];
    }
    cout << endl;
    cout << "=====" << endl << endl;

    /* Finding the keyword stage */
    char texts[MAX_KEY_LENGTH+1][length];
    int  rots [MAX_KEY_LENGTH+1][MAX_KEY_LENGTH+1];
    float bestscore=0.0;
    int  bestkey=0;
    cout << "=====" << endl;
    for(int i=MIN_KEY_LENGTH; i<=MAX_KEY_LENGTH; i++) {
        int  thisrots[i];
        char thistexts[length];
        float thisscore=0.0;
        getkey(thisrots, i);
        decrypt(thistexts, thisrots, i);
        for(int j=0; j<i; j++) rots[i][j]=thisrots[j];
        for(int j=0; j<length; j++) texts[i][j]=thistexts[j];
    }
}
```

```

// Get the score for the ciphered message decrypted with the current key
thisscore=freqscore(thistexts ,0);

// Track the best score
if(bestscore==0.0) {
    bestscore=thisscore;
    bestkey=i;
}
else if(thisscore<bestscore) {
    bestscore=thisscore;
    bestkey=i;
}

cout << "Keylength:_" << i << ",_keyword:_" ;
for(int j=0; j<i; j++) cout << (char)(thisrots [j]+65);
cout << endl;
}
// Show which key was chosen
cout << "I_have_chosen_the_key:_" ;
for(int i=0; i<bestkey; i++) cout << (char)(rots [bestkey][i]+65);
cout << endl;
cout << "======" << endl << endl;

/* Show decrypted message stage */
cout << "======" << endl;
cout << "Deciphered_message" << endl;
cout << "======" << endl;
for(int i=0; i<length; i++) cout << texts [bestkey][i];
cout << endl;
cout << "======" << endl;

return 0;
}

/* ascii2int returns the value of the ascii letter it is passed. A=0, B=1 ...
*/
int ascii2int(char ascii) {
    return (int)ascii -65;
}

/* freqscore(char*, int) returns a 'score' for the rotation defined of the
text in text by rot rotations depending on how well it matches the
frequencies of the letters in the English language */
float freqscore(char * text, int rot) {
    float score=0.0;
    int len=strlen(text);
    float freqs [26];

    for(int i=0; i<26; i++) freqs [i]=0.0;

    for(int i=0; i<len; i++) freqs [ascii2int(rotate(text [i], rot))]+=1;
    for(int i=0; i<26; i++) score+=fabs(proper_freqs [i]-((freqs [i]/(float)len)*100));

    return score;
}

/* rotate(char, int) returns the character, in, rotated rot times */
char rotate(char in, int rot) {
    return (char)((((int)in -65)+26-rot)%26+65);
}

```

```

}

/* getkey(int*, int) returns the most likely key for the specified keylength */
void getkey(int * rots, int keylength) {
    // Find the rotation for each letter of the key
    for(int i=0; i<keylength; i++) {
        float bestscore=0.0, thisscore=0.0;
        int bestrot=0;
        // Try each possible rotation
        for(int j=0; j<26; j++) {
            int l=0, m=0;
            for(int k=i; k<length; k+=keylength) m++;
            char subset[m];
            for(int k=i; k<length; k+=keylength) {
                subset[l]=ciphered[k];
                l++;
            }

            thisscore=freqscore(subset, j);
            if(bestscore==0.0) {
                bestscore=thisscore;
                bestrot=j;
            }
            else if(thisscore<bestscore) {
                bestscore=thisscore;
                bestrot=j;
            }
        }
        rots[i]=bestrot;
    }
}

/* decrypt(char*, int*, int) decrypts the cipher using the key given in key
and puts the result in the correct place in out */
void decrypt(char * out, int * key, int keylength) {
    char text[length];
    for(int i=0; i<keylength; i++) {
        for(int j=i; j<length; j+=keylength) {
            out[j]=rotate(ciphered[j], key[i]);
        }
    }
    text[length]='\0';
}

```