**CS241 Lab 2**

# Improved Caesar Cipher for Better Security

Caesar cipher is an old encryption technique in which each letter in the original text is replaced by a letter at some fixed number of positions down the alphabet. Here, the fixed number of positions is referred to as the key for the cipher. For example, if the key is 4, then the mapping between the plain text and cipher text alphabets will be as follows:

```
Plain Text:     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
Cipher Text:    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
```

Caesar cipher can be automatically broken by using the known letter frequencies of English or simply trying out all of the 25 keys. A simple yet effective way of improvement is to use several numbers, instead of one, as the key. With multiple numbers as the key, different shifts can be made at different positions in the plain text. For example, we could shift the first input character by 5, the second by 14, the third by 17, and the fourth by 10. Then we repeat the pattern, shifting the fifth input character by 5, the sixth by 14, and so on, until we run out of characters in the plain text. Apparently, the more numbers the key has, the more difficult is to break the cipher.

In the above example, the four numbers are 5, 14, 17, and 10, which are the numerical equivalents of the string ``FORK'' in a 26-letter alphabet consisting of the letters A-Z. In this lab, you will modify a Caesar cipher program that uses one number as its key and make it use four numbers, given as a string of four characters, like “FORK”, as its key.

Specifically, what you need to do is as follows. First, modify the given program to allow a string of four letters, instead of one integer number, as a command line argument for the key. Then, convert the input string into a group of four integer numbers and store them in an integer array. Finally, declare an integer variable, say n, to keep track of the number of characters to be processed, and use n % 4 to determine the index of the number to be used for encryption/decryption. In addition, find a way to make your program scalable; that is, allowing the key to be extended to any length. You may replace 4 in the code with the length of the input string and create a dynamically allocated array to hold the corresponding integer numbers for the key.

Specifically, what you need to do is as follows:

- In your working directory, use the command below to download the source code **cipher.c** from the instructor’s web page to your current working directory.

  wget http://cis.gvsu.edu/~wangx/teaching/cis241/cipher.c

- Also, download the files **makefile** and **data.txt** from the instructor’s web page. View the source code to find what it does.
- Run the make command to build an executable and run it automatically. Check and see if it works as expected.

  ```
  make
  make test
  diff -s data.txt data.bak
  ```

- Open **cipher.c** in a text editor, read the code carefully (make sure you fully understand it), and modify it as described above.
- Open **makefile** in a text editor and modify it accordingly.
- Run the make command to build an executable and run it automatically. Check and see if it works as expected.

Show the source code of your program and the **makefile** to your instructor when you have done all the above successfully.

**The following may be helpful:**

- The function strlen(str) in <string.h> library will return the length of string str. For example:

```
len = strlen (str); // if str = "FORKAB", then len = 6
```

- The statement key_arr = malloc (len * sizeof (int)) will allocate the space of len integers and assign the starting address to key_arr, where key_arr is a pointer of integer. For example:

```
int * key_arr; // declare an int pointer

key_arr = (int *) malloc (len * sizeof(int)); // allocate space for an int array of len elements
```