Mehran Sahami
CS 106A

Handout #26
October 22, 2007

Strings and Ciphers

Based on a handout by Eric Roberts.

Cryptography, derived from the Greek word *κρυπτοσ* meaning *hidden,* is the science of creating and decoding secret messages whose meaning cannot be understood by others who might intercept the message. In the language of cryptography, the message you are trying to send is called the **plaintext**; the message that you actually send is called the **ciphertext**. Unless your adversaries know the secret of the encoding system, which is usually embodied in some privileged piece of information called a **key**, intercepting the ciphertext should not enable them to discover the original plaintext version of the message. On the other hand, the intended recipient, who is in possession of the key, can easily translate the ciphertext back into its plaintext counterpart.

Caesar ciphers

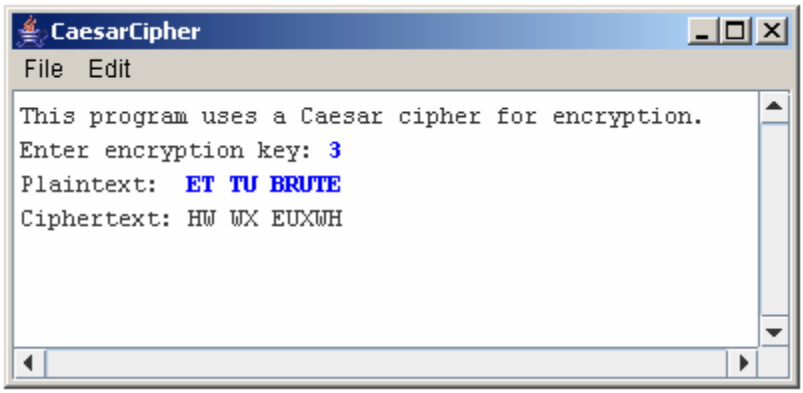
One of the earliest documented uses of ciphers is by Julius Caesar. In his *De Vita Caesarum,* the Roman historian Suetonius describes Caesar's encryption system like this:

> If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely **D**, for **A**, and so with the others.

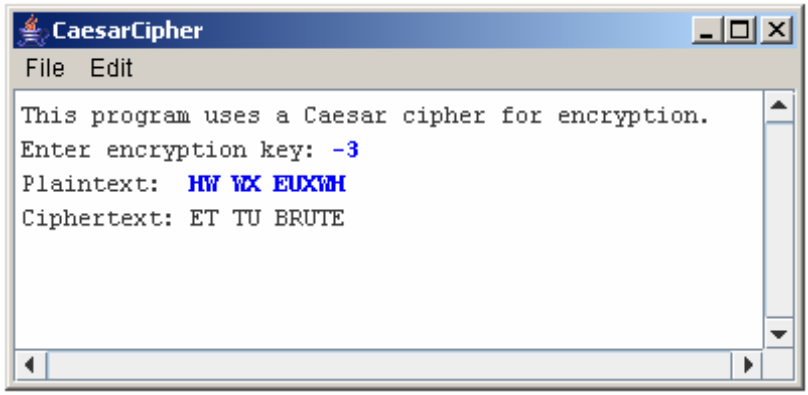
Even today, the technique of encoding a message by shifting letters a certain distance in the alphabet is called a **Caesar cipher**. According to the passage from Suetonius, each letter is shifted three letters ahead in the alphabet. For example, if Caesar had had time to translate the final words Shakespeare gives him, **ET TU BRUTE** would have come out as **HW WX EUXWH**, because **E** gets moved three letters ahead to **H**, **T** gets moved three to **W**, and so on. Letters that get advanced past the end of the alphabet wrap around back to the beginning, so that **X** would become **A**, **Y** would become **B**, and **Z** would become **C**.

Caesar ciphers have been used in modern times as well. The "secret decoder rings" that used to be given away as premiums in cereal boxes were typically based on the Caesar cipher principle. In early electronic bulletin board systems, users often disguised the content of postings by employing a mode called **ROT13**, in which all letters were cycled forward 13 positions in the alphabet. And the fact that the name of the **HAL** computer in Arthur C. Clarke's *2001* is a one-step Caesar cipher of **IBM** has caused a certain amount of speculation over the years.

Let's consider writing a simple program that encodes or decodes a message using a Caesar cipher. The program needs to read a numeric key and a plaintext message from the user and then display the ciphertext message that results when each of the original letters is shifted the number of letter positions given by the key. A sample run of the program might look like the example on the following page.



For the Caesar cipher, decryption does not require a separate program as long as the implementation is able to accept a negative key, as follows:



Letter-substitution ciphers

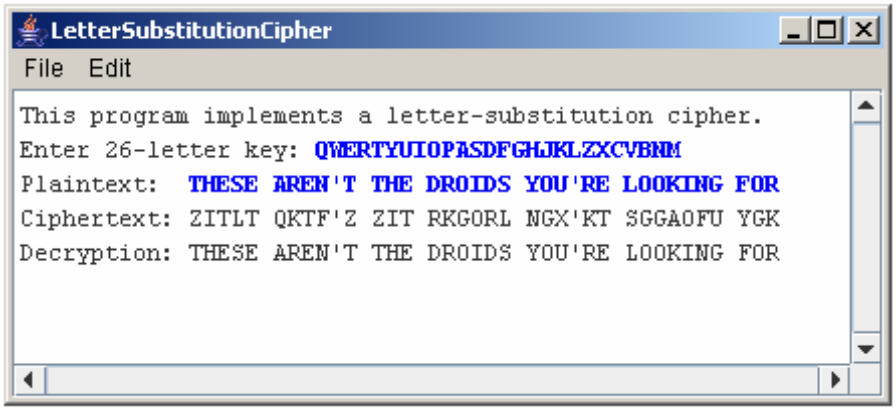
Although they are certainly simple, Caesar ciphers are also extremely easy to break. There are, after all, only 25 nontrivial Caesar ciphers for English text. If you want to break a Caesar cipher, all you have to do is try each of the 25 possibilities and see which one translates the ciphertext message into something readable. A somewhat better scheme is to allow each letter in the plaintext message to be represented by an arbitrary letter instead of one a fixed distance from the original. In this case, the key for the encoding operation is a translation table that shows what each of the 26 plaintext letters becomes in the ciphertext. Such a coding scheme is called a **letter-substitution cipher.** The key in such a cipher can be represented as a 26-character string, which shows the mapping for each character, as shown in the following example:

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

Letter-substitution ciphers have been used for many, many years. In the 15th century, an Arabic encyclopedia included a section on cryptography describing various methods for creating ciphers as well as techniques for breaking them. More importantly, this same manuscript includes the first instance of a cipher in which several different coded symbols can stand for the same plaintext character. Codes in which each plaintext letter

maps into a single ciphertext equivalent are called **monoalphabetic ciphers.** More complex codes in which the representation for a character changes over the course of the encryption process are called **polyalphabetic ciphers**.

The second problem to consider is to write a program that implements this more general letter-substitution cipher. The program should ask the user to enter a 26-letter key and the plaintext message. It should then display the ciphertext and, to ensure that the encryption is working, what you get if you decrypt the ciphertext using the same key:



Caesar ciphers solution

```
/*
 * File: CaesarCipher.java
 * -----------------------
 * This program translates a line of text into its Caesar cipher
 * form.
 */

import acm.program.*;

public class CaesarCipher extends ConsoleProgram {

   public void run() {
      println("This program uses a Caesar cipher for encryption.");
      int key = readInt("Enter encryption key: ");
      String plaintext = readLine("Plaintext:  ");
      String ciphertext = encryptCaesar(plaintext, key);
      println("Ciphertext: " + ciphertext);
   }


   /*
    * Encrypts a string by adding the value of key to each character.
    * The first line makes sure that key is always positive by
    * converting negative keys to the equivalent positive shift.
    */
   private String encryptCaesar(String str, int key) {
      if (key < 0) {
         key = 26 - (-key % 26);
      }
      String result = "";
      for (int i = 0; i < str.length(); i++) {
         char ch = encryptCharacter(str.charAt(i), key);
         result += ch;
      }
      return result;
   }


   /*
    * Encrypts a single character using the key given.  This
    * method assumes the key is non-negative.  Non-letter
    * characters are returned unchanged.
    */
   private char encryptCharacter(char ch, int key) {
      if (Character.isLetter(ch)) {
         ch = (char) ('A'
               + (Character.toUpperCase(ch) - 'A' + key) % 26);
      }
      return ch;
   }
}
```

Letter-substitution ciphers solution

```
/*
 * File: LetterSubstitutionCipher.java
 * -----------------------------------
 * This program translates a line of text using a letter-substitution
 * cipher.
 */

import acm.program.*;

public class LetterSubstitutionCipher extends ConsoleProgram {

   public void run() {
      println("This program implements a letter-substitution cipher.");
      String key = readKey();
      String plaintext = readLine("Plaintext:  ");
      String ciphertext = encrypt(plaintext, key);
      String decryption = decrypt(ciphertext, key);
      println("Ciphertext: " + ciphertext);
      println("Decryption: " + decryption);
   }


   /**
    * Reads an encryption key from the user and checks it for
    * legality (see isLegalKey below).  If the user enters an
    * illegal key, the user is asked to reenter a valid one.
    */
   private String readKey() {
      String key = readLine("Enter 26-letter key: ");
      while (!isLegalKey(key)) {
         println("That key is not legal.");
         key = readLine("Reenter key: ");
      }
      return key;
   }


   /**
    * Checks to see whether a key is legal, which means it meets
    * the following conditions:
    * (1) It is 26 characters long,
    * (2) It contains only uppercase letters, and
    * (3) It has no duplicated letters.
    */
   private boolean isLegalKey(String key) {
      if (key.length() != 26) return false;
      for (int i = 0; i < 26; i++) {
         char ch = key.charAt(i);
         if (!Character.isUpperCase(ch)) return false;
         for (int j = i + 1; j < 26; j++) {
            if (ch == key.charAt(j)) return false;
         }
      }
      return true;
   }
```

```
   /**
    * Encrypts a string according to the key.
    */
   private String encrypt(String str, String key) {
      String result = "";
      for (int i = 0; i < str.length(); i++) {
         char ch = encryptCharacter(str.charAt(i), key);
         result += ch;
      }
      return result;
   }


   /**
    * Encrypts a single character according to the key.
    */
   private char encryptCharacter(char ch, String key) {
      if (Character.isLetter(ch)) {
         ch = key.charAt(Character.toUpperCase(ch) - 'A');
      }
      return ch;
   }


   /**
    * Decrypts a string according to the key.
    */
   private String decrypt(String str, String key) {
      String result = "";
      for (int i = 0; i < str.length(); i++) {
         char ch = decryptCharacter(str.charAt(i), key);
         result += ch;
      }
      return result;
   }


   /**
    * Decrypts a single character according to the key.
    */
   private char decryptCharacter(char ch, String key) {
      int index = key.indexOf(ch);
      if (index != -1) {
         ch = (char) ('A' + index);
      }
      return ch;
   }
}
```