Lab 2 - Cryptography

Luciano Giannini, Ryan Pepin, Mike Emmett
LC 207: Mathematical Experiments in Computer Science
(Dated 10/16/2019)

Contents

Introduction

World War I was the first of two huge conflicts that caused the superpowers of the world to be at each other's throats. After World War I ended, Europe was left with the most damage and struggled to recover during the following two decades. Because of this vulnerability nations like Germany, Japan and Italy were taken over by militaristic leaders like Adolf Hitler. Hitler's goal for world domination forced the entire globe to reluctantly take part in the second great war of that time period. The United States was the last of all of the major superpowers in the world to join the conflict, and this was only in response to the Japanese bombing of Pearl Harbor. This war, like many others, cost millions of people around the world their lives, but because of the war behind the war many more were saved. During World War II the British were very focused on intercepting and decoding German transmissions regarding matters such as transport locations, bombing targets, ambush plans, and German defenses. During this lab, we have been exposed to how the Germans cleverly encoded their transmissions, and how the British were able to decode these hidden messages and save millions of lives.

While exploring the work of the British science division we learned about the life and works of Alan Turing, and how the Allied forces might have lost the war if it hadn't been for his efforts. We explored his works by watching two different films, Breaking the Code, and The Imitation Game, that explained how his efforts impacted the war. We learn about how his invention helped to crack the uncrackable encryption generated by the Enigma machine used by the Germans. During these films, we were also introduced to the idea of Cryptography and how far back in history this concept goes. During the film The Imitation Game, we are shown a scene of how Alan's teacher cannot read a note he is passing because he used Cryptography to

During this lab, we were tasked with using a key world to encrypt a message by shifting each letter and then decrypting it just as easily. We were tasked with creating programs that both demonstrated and used Euclid's Algorithm and Greatest Common Divisors. These ideas were used in RSA cryptography and helped us to demonstrate this type of cryptography in a real-world scenario.

Program 1: A Cycle of Shifts

Objective: Write a program that takes a text and a codeword, and produces the encrypted text.

We are going to need to use the % (mod) operation and an if-statement or two to make sure you wrap around appropriately.

FIG 1.1: Encrypted Class

```
public class encrypted
{
   private String code;
   private String text;
   private String encoded;
   private String decoded;

   public encrypted()
   {
      code = "";
      text = "";
      encoded = "";
      decoded = "";
   }

   public static void main(String[]args)
   {
      encrypted e = new encrypted();
      e.jumble();
   }
```

We created a class to hold the values that were going to be encrypted. So we have the code word, the text to be encrypted and the encoded text. Within the main method, we create the class and call the method jumble().

FIG 1.2: Jumble Method

```
public void jumble()
{
   int Cnum = 0, Ctemp = 0, Tnum = 0, Ttemp = 0, Enum = 0, Etemp = 0, Dnum = 0, Dtemp = 0;
   int Clength = 0, Tlength = 0, count = 0;
   Scanner input = new Scanner(System.in);
   System.out.println("Enter the codeword: ");
   code = input.nextLine();
   System.out.println("Enter the desired text to encrypt");
   text = input.nextLine();
   Clength = code.length();
   Tlength = text.length();
   for(int i = 0; i < Tlength; i++)
   {
      Tnum = text.charAt(i) + 0; //gets ASII code 97 --> 122
      Ttemp = Tnum - 97; //my numbers 0 --> 25
      if(count < Clength)
      {
         Cnum = code.charAt(count) + 0; //gets ASII code 97 --> 122
         Ctemp = Cnum - 97; //my numbers 0 --> 25
         count++;
      }
      else
      {
         count = 0;
         Cnum = code.charAt(count) + 0; //gets ASII code 97 --> 122
         Ctemp = Cnum - 97; //my numbers 0 --> 25
         count++;
      }

      Etemp = (Ttemp + Ctemp)%26;
      encoded = encoded + (char)(Etemp + 97);
     // Dtemp = (Etemp - Ctemp + 26)%26;
     //decoded = decoded + (char)(Dtemp + 97);
   }
   System.out.println("encrypted: " + encoded);
```

We start this method by creating a lot of different integer variables that will keep track of the location in the codeword and the text from the user to be encoded. We then need to ask the user for the codeword and the desired text to encrypt. We then record the length of the codeword and the text. We need to loop through the length of the text so we create a loop starting from zero the text length. Inside the loop, we want the ASCII value of the first letter of the text and then subtract by 97 to get a value from zero to 25. We need to check if the count is smaller than the length of the code length if it is then we get the ASCII value of the first letter of the codeword and subtract 97 also. To get the encoded value we add the two numbers together and module by 26 so we only get numbers between 0 and 25. Once we have this we add 97 and cast it to a character. We do this until the loop is finished.

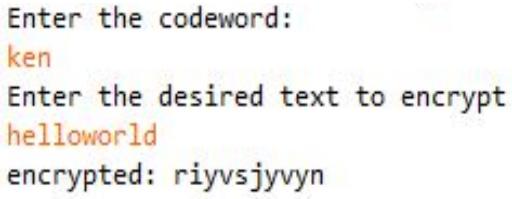

FIG 1.3: Output

Program 2: Decode A Cycle of Shifts

Objective: Take an encrypted message and, using the codeword, decrypt the message to reveal the original passage.

FIG 2.1 decoded.java

```
public static void main(String[]args)
{
   String code = "";
   String text = "";
   String encoded = "";
   String decoded = "";
   Scanner input = new Scanner(System.in);
   System.out.println("Enter the codeword: ");
   code = input.nextLine();
   System.out.println("Enter the desired text to decrypt");
   text = input.nextLine();
   int Cnum = 0, Ctemp = 0, Tnum = 0, Ttemp = 0, Enum = 0, Etemp = 0, Dnum = 0, Dtemp = 0;
   int Clength = 0, Tlength = 0, count = 0;
   Clength = code.length();
   Tlength = text.length();
   for(int i = 0; i < Tlength; i++)
   {
      Tnum = text.charAt(i) + 0; //gets ASII code 97 --> 122
      Ttemp = Tnum - 97; //my numbers 0 --> 25

      if(count < Clength)
      {
         Cnum = code.charAt(count) + 0; //gets ASII code 97 --> 122
         Ctemp = Cnum - 97; //my numbers 0 --> 25
         count++;
      }
      else
      {
         count = 0;
         Cnum = code.charAt(count) + 0; //gets ASII code 97 --> 122
         Ctemp = Cnum - 97; //my numbers 0 --> 25
         count++;
      }
      Dtemp = (Ttemp - Ctemp + 26)%26;
      decoded = decoded + (char)(Dtemp + 97);
   }
   System.out.println(decoded);
}
```

FIG 2.2 Result

```
Enter the codeword:
ken
Enter the desired text to decrypt
riyvsjyvyn
helloworld
```

In order to decrypt the text, we take in a codeword and a passage of encrypted text. We then take each individual encoded letter (ETemp) and subtract its ASCII value from that of the corresponding letter from the codeword (CTemp). We then add 26 to the result to make sure that the value is positive and mod (%) this number by 26 so the program wraps around correctly. This value is stored in DTemp and is added to the rest of the decoded text by adding 97 to DTemp (since lowercase numbers start at ASCII value 97), changing this value from an ASCII number to the corresponding character using (char), and adding it to the rest of the decoded letters (decoded).

Program 3: Index of Coincidence

Objective: Take an encrypted text and rotate the letters by one through n spaces, with n representing the length of the text. Then to find the Index of Coincidence, we take the characters from the rotated text that occur in the same place as the encrypted text and divide it by the total length of the text.

FIG 3.1: Methods

```
public static String rotate(String text, int rotations)
{
  String rotated= "";
  for (int i = 0;i<text.length();i++)
  {
     rotated += text.charAt(rotations);
    if (rotations==text.length()-1)
      rotations = 0;
    else
      rotations++;
  }
  return rotated;
}

public static double findIndex(String text,String rotated)
{
  double amount = 0.0;
  for (int i = 0;i<text.length()-1;i++)
  {
    if (text.charAt(i)==rotated.charAt(i))
      amount++;
  }
  int temp = (int)(((amount/text.length())*10000));
  double yurr = ((double)temp/100);
  return (yurr);
}
}
```

FIG 3.2: First 21 results

Rotation	Index of coincidence
1	6.06
2	1.68
3	5.05
4	4.37
5	4.04
6	4.04
7	2.69
8	2.69
9	6.39
10	3.36
11	5.05
12	9.09
13	3.03
14	4.04
15	4.04
16	5.38
17	4.71
18	9.42
19	4.37
20	4.71
21	7.4

FIG 3.3: Main Method

```
public static void main(String[]args)
{
  //Text is already encrypted
  String text = "gieorbcxekrtovfdsyyzrishurbgxuovhvifkrqcsqym
  // NON ENCRYPTED TEXT: String text = "werenostrangerstolove
  System.out.println("Rotation    Index of coincidence");
  for (int i = 1;i<text.length();i++)
  {
    String rotated = rotate(text,i);
    int index = findIndex(text,rotated);
    System.out.println(i+"           "+index);
  }
```

Original Text: First 296 characters of “Never Gonna Give You Up” by Rick Astley

Encrypted Text:

"gieorbcxekrtovfdsyyzrishurbgxuovhvifkrqcsqymnpyyvgbwqvdqrxxfglndmzdlvxovxkbpcbeabepq
xxtoxgrmfpvbweaisgrieqylsnhcxjkrakxrvplyyuyavwjropvxktyxgkqnuilyyhxhrbwgkrqxiiovtyrakkv
filyyhzrrfieqsaxeyoxlyyqyaaxiiovtyrakvhxeeyyaneanhrciedcberrfieqsaxezkorishmvlxiiovtyrakwni
kbyhoiiaozrbkbxrndiyveysinxhuevgish”

Codeword: ken

Analysis: The length of the codeword can be found from my results by adding up as many multiples of any number as possible and averaging them. When you find the average of the multiples of a number, it should be very close to 6.6% if that number is the length of the codeword. When averaging the first 7 multiples of 3, our data gives an index of coincidence of 6.49%, meaning that 3 is likely the length of the codeword.

Multiples of (average first 21 rotations):

(2, 4.88%) (3, 6.49%) (4, 5.25%) (5, 4.04%) (6, 7.52%) (7, 4.71%) (8, 4.04%) (9, 7.9%)

Program 4: Finding Greatest Common Divisor with Every Number

Objective: Write a program to find the greatest common divisor by trying each number from the lower number and working downwards towards 1.

```
public static void main (String [] args)

{
   Scanner input = new Scanner(System.in);
   System.out.println("Please enter two number to find the greatest common divisor");
   long one = input.nextLong();
   long two = input.nextLong();
   int result = 0;
   if(one >= two)
   {
      result = newGCD(one, two);
   }
   else
   {
      result = newGCD(two,one);
   }
   System.out.println(result);
}
```

FIG 4.1: Main Method

First, we have to ask for the two numbers to find the greatest common divisor. I had an if statement next to see which number was larger than the other because you are comparing both at the same time. As seen in FIG 4.2. Which using this if the smaller number is equal to 0 the GCD is the larger number, although if the smaller number is equal to 1 then the GCD is also 1.

```
public static int newGCD(long a, long b)
{
   if(b == 0)
   {
      return (int)a;
   }
   if(b == 1)
   {
      return 1;
   }
   int gcd = 0;
   for(int i = 1; i<= a; i++)
   {
      if(a%i == 0 && b%i ==0)
      {
         gcd = i;
      }
   }
   return gcd;
}
```

FIG 4.2 GCD Method

The for loop will go through each number that is between 1 and the larger number, which is a in this case. If both of the numbers module i are both 0 then we save the value of i to be the new GCD. If there is a larger value that is found the gcd will change to that value until the for loop is done. Then it returns the gcd.

Program 5: Finding Greatest Common Divisor with Prime Factors

Objective: Write a program to find the greatest common divisor by listing the prime factors for each number, and taking the intersection of the two lists.

```
static boolean prime[] = new boolean[10000];
public static void main(String[]args)
{

   for(int i = 0;i<prime.length;i++)
      prime[i]=true;

 // 0 and 1 are not prime numbers
   prime[0] = false;
   prime[1] = false;
   for (int p = 2; p * p < 10000; p++) {

     // If prime[p] is not changed, then it is a prime
      if (prime[p] == true) {

         // Update all multiples of p as non-prime
         for (int i = p * p; i < 10000; i += p)
            prime[i] = false;
      }
   }
```

FIG 5.1: Creating an Array of Prime Numbers

To see if a number is prime or not we created an array that will go through each multiple of a number which will mark it as a nonprime number.

```
Scanner input = new Scanner(System.in);
System.out.println("Please enter two number to find the greatest common divisor");
int one = input.nextInt();
int two = input.nextInt();
int result = 0;
Stack<Integer> stack1 = new Stack<Integer>();
Stack<Integer> stack2 = new Stack<Integer>();
common_prime(one,stack1);
common_prime(two,stack2);
int gcd = CheckPrimes(stack1,stack2);
System.out.println("The GCD of "+one+ " and " + two + " is " + gcd);
```

FIG 5.2 Main Method

Likewise in problem 4 we ask for the two numbers to find the greatest common divisor. I then went on and created two stacks for each number that I want to check the prime factors of. Then call the common_prime method for each passing in the number and a corresponding stack. Once the common_prime method is executed to find the GCD of the two numbers I created another method that checks the intersection of the two stacks.

FIG 5.3: Common Prime Method

```
static void common_prime(int a, Stack stack)
{
 // Find the prime divisors of the gcd
   for (int i = 2; i <= (a); i++)
   {

     // If i is prime and a divisor of gcd
      if (prime[i] && a % i == 0)
      {
         stack.push(i);
         int c = a/i;
         if(c!=1)
         {
            common_prime(c,stack);
            break;
         }
      }
   }
```

What this method does it checks throughout the prime array that we created in FIG 5.1 and the corresponding number module i. If they are both true then we push the value of i onto the stack. Then we want to find the next prime factors of the value passed through the method divided by i which was already a prime factor of the starting value passed. In FIG 5.3 if the value of c is not 1 then we recursively call the common_prime method with the new value of c until we have found all prime factors of the number.

FIG 5.4: Checks the Intersection of the Two Stacks

```
public static int CheckPrimes(Stack stack1, Stack stack2)
{
   int gcd = 1;
   while(!stack1.empty()&&!stack2.empty())
   {
      int one= 0;
      int two = 0;
      one = (int)stack1.pop();
      two = (int)stack2.pop();
      System.out.println(one + "\t\t"+ two);
      if(one == two)
      {
         gcd = gcd*one;
      }
   }
   return gcd;
}
```

Passing the two stacks that we created in FIG 5.3 to the CheckPrimes method, will pop the two stacks and print the values on each stack and then compare the data of the two. If two pops of the stack are the same then the gcd is the existing gcd times one same values. While the stacks are not empty, the loop will continue multiplying the same values together. Which once the while loop stops it will return the gcd of the numbers.

Problem 6: Euclid's Algorithm Recursive and Iterative

Objective: Write a recursive program to implement Euclid's algorithm. Write an iterative program to implement Euclid's algorithm.

```
public static int newGCD(long a, long b)
{
   if(b == 0)
   {
      return (int)a;
   }
   if(b == 1)
   {
      return 1;
   }
   return newGCD(b,a%b);
}
```

FIG 6.1: Euclid's Algorithm Recursively

The setup for this program is the same in FIG 4.1 but there is a recursive call at the end. Which in this method it will check to see if the smaller number is 0 or 1, which will return the larger number or 1 respectively. If the program doesn't go into any of the if statements the recursive call will move the smaller number to be the new larger number and the smaller number is the large number module the small number. This will eventually cause one of the if statements to return a or 1.

```
Scanner input = new Scanner(System.in);
System.out.println("Please enter two number to find the greatest common divisor");
int one = input.nextInt();
int two = input.nextInt();
int result = 0;
if(one >= two)
{
result = one;
   while(one!=0 && two!=0)
   {
      result = one;
      int a = one;
      int b = two;
      two = a%b;
      one = b;
   }
}
else
{
 result = two;
   while(one!=0 && two!=0)
   {

      result = one;
      int a = one;
      int b = two;
      one = b%a;
      two = a;
   }
   System.out.println(result);
```

FIG 6.2: Euclid's Algorithm Iteratively

The part to focus on is where the if statement starts. This is where the iterative program is mainly seen. We want to determine which number is larger than the other. Which if the first number is larger than second, then we have started at the largest value making the result equal to the largest number. Then we approach the while loop. Which determines if any of the two numbers is equal to zero. If one of them is, print the result. If not go into the loop. We then make the result to the larger number again. Then we want to create two new variables a and b. a will hold the value of one and b will hold the value of two. Using the same method from Euclid's algorithm we want to follow the same rules of gcd(a,b) = gcd(b,a%b). So we make one equal to a%b and two equal a, which then we continue to run this system along will making the result equal one. Until one of the values equals to 0. Which because we initialize result to 0 before going into the if statement and while loop, we are able to see what the last value result was saved to. Furthermore giving us our gcd of the two numbers.

```
 ----jGRASP exec: java GCDTimes
The time for GCD that goes through all numbers is 5809 milliseconds
The time for GCD Recursive is 0 milliseconds
The time for GCD Iterative is 0 milliseconds
The time for GCD using prime factors is 31 milliseconds

 ----jGRASP: operation complete.
```

FIG 6.3: GCD Times

int num1 = 676865576;

int num2 = 1221321;

The two starting number that we are going to use to determine which GCD method are faster is num1, and num2. The first program that we time is Program number 4. From FIG 4.2 we see that the for loop goes through each and every number to find the common divisors of the two inputs. This is the slowest out of all four programs and takes 5809 milliseconds to run and get the gcd of num1 and num2. The next two programs have relatively the same time to give an answer. The first program is finding the GCD recursively and the second is finding the GCD Iteratively. Although both finish in less than 0 milliseconds. The recursive algorithm should work faster than the iterative program. Finally we find the time it takes to find the GCD using the prime factors of the two numbers. This program is from problem 5, which gives us a time of 31 milliseconds.

Problem 7: Finding X and Y Such That aX + bY = gcd(a,b)

Objective: Find the two numbers (x and y) so that ax + by = gcd(a,b) using a recursive program.

FIG 7.1: Recursive Method

```
Scanner input = new Scanner(System.in);
System.out.println("Please enter two numbers (a,b) to find (x,y)");
int one = input.nextInt();
int two = input.nextInt();
Triple result = new Triple();
if(one >= two)
{
result = ExGCD(one, two);
}
else
{
result = ExGCD(two,one);
}
System.out.println(result.g+" "+result.x+" "+result.y);
}

public static Triple ExGCD(int a, int b)
{
  if (b == 0)
  {
  Triple temp = new Triple(a,1,0);
  return temp;
  }
  Triple z = ExGCD(b,a%b);
  Triple temp = new Triple(z.g,z.y,-z.y*(a/b)+z.x);
  return temp;
}
```

FIG 7.2: Triple Class

```
public class Triple
{
  int x,y,g;
  public Triple()
  {
    x = 0;
    y = 0;
    g = 0;
  }
  public Triple(int g,int x,int y)
  {
    this.x = x;
    this.y = y;
    this.g = g;
  }
}
```

FIG 7.3: Result

```
Please enter two numbers (a,b) to find (x,y)
113
47
1 5 -12
>
```

In order to return the gcd, x, and y values, We needed to create a class that holds three variables called Triple. The program takes the two numbers and finds the gcd, which is stored in g, the same way as in problem 6. The y- and x-values are found using Euclid's Extended Algorithm and the function from our in-class notes (f(x) = -y(a/b)+x).

Problem 8: Encoding using RSA

Objective: Write a program to encode numbers using RSA given the public key (e, m).

To do we are going to have to incorporate the fast modular exponentiation algorithm (Egyptian-style) which will help us to encode numbers.

```
public static void main(String[]args)
{  Scanner input = new Scanner(System.in);
   System.out.println("Please enter the private key e, them m");
   long exp = input.nextLong();
   long mod = input.nextLong();
   System.out.println("Please enter the number to encode");
   long base = input.nextLong();
   long result = key(exp, mod,base);
   System.out.println("Encoded number "+result);
}
```

FIG 8.1: Main Method

First, we ask for the public key, then the number the user wants to encode. Which we made them all long variables so the input could be a large number. Then we call the method key, which uses all three inputs in its method.

FIG 8.2: Key Method

```
public static long key(long e, long m, long w)
{

   if(e == 1)
   {
      return w%m;
   }
   else if(e % 2 ==0)
   {
      return ((long)Math.pow((key(e/2,m,w)),2))%m;
   }
   else
   {
      return (w*key(e-1,m,w))%m;
   }
```

This method is based on the fast modular exponentiation algorithm. Which tells us that the encoded number is equal to $w^e\%m$, w is the input for number, e and m are from the public key. The value of this expression will be an encoded number for the input value. This algorithm relies heavily on the exponent of the number. Which if the exponent is equal to 1 then the program will just return the value of $w\%m$. If the exponent is not equal to 1 then we need to check if the number is even or odd. So we have an if statement which sees if there is a remainder from the exponent module 2, and if it is 0 then we have calculated the value of $w^{e/2}\%m$ squared. Now for odd numbers, we want the number to be even so we subtract 1 and find the value of $w^{e-1}\%m$ times w. We do this until e is equal to 1 and return the value of $w\%m$.

Problem 9: Finding the Private Key and Decoding Using RSA

Objective: Write a program that factors the public key *pq*, into *p* and *q*, computes (*p*-1)(*q*-1), computes the inverse to find the private key, and decodes a message.

For this program, we have to use many of the previous programs, which will allow us to find p-1 and q-1 and find the positive x or y value seen in FIG 7.1 which we can use to find the private key, then decode a message.

```
static String[] list;
static long[] result;
static int count;
public static void main(String[]args)
{
   Scanner input = new Scanner(System.in);
   System.out.println("Please enter the public key, first p*q then e");
   long pq = input.nextLong();
   int e = input.nextInt();
   Double factorpq = factors(pq);
   long newP = (factorpq.x-1);
   long newQ = (factorpq.y-1);
   long newN = newP*newQ;
   Triple triple = new Triple();
   GCDxy p = new GCDxy();
   if(newN > e)
   {
      triple = p.ExGCD(newN,e);
   }
   else
      triple = p.ExGCD(e,newN);
   long newD;
   if(triple.x >   0)
   {
      newD = ((triple.x)+newN)%(newP*newQ);
   }
   else newD = ((triple.y)+newN)%(newP*newQ);

   decryption(newD,pq);
}//end main
```

FIG 9.1: Main Method

```
public static Double factors(long num)
{
   Double factors = new Double();
   for(long i = 1; i < num; i++)
   {
      if(num%i == 0)
      {
         long factor = num/i;
         factors = new Double(i,factor);
      }
   }//end forloop
   return factors;
}//end factors
```

FIG 9.2: Factors Method

```
public class Double
{

   long x,y;

   public Double()
   {
      x = 0;
      y = 0;

   }
   public Double(long x,long y)
   {
      this.x = x;
      this.y = y;

   }
}
```

FIG 9.3: Double Class

We first ask for the public key from the user. We want to find what (p-1)(q-1) equals so we are able to use it to find the positive value of x or y. We created a small program that would give us the largest factors of a given number. Look at FIG 8.2 we pass through a number and create an object Double, which in FIG 8.3 holds two values which are the two factors of a number. Then we need to check all of the values that are factors of the number passed into the method. We start with a for loop that goes from 1 to the value of the number. Then we have an if statement that tells us if the number has a remainder of 0, which lets us know whatever i is, it is a factor of the number. So we then save the factors object to these two new values. We do this until the loop is done and then return the Object Double factors.

Once we have the factors of pq then we are able to find the value of p-1 and q-1 and the value of (p-1)(q-1). Having the value of (p-1)(q-1) and the e value of the public key we can calculate the values of x and y i.e. ax + by = gcd(a,b). Using the method from FIG 7.1 we can find x and y. The Triple class is the same as the Double class we created but one extra value that is being stored within the object.

FIG 9.4: Cont.

```
public static Triple ExGCD(long a,long b)
{
   if (b == 0)
   {
      Triple temp = new Triple(a,1,0);
      return temp;
   }
   Triple z = ExGCD(b,a%b);
   Triple temp = new Triple(z.g,z.y,-z.y*(a/b)+z.x);
   return temp;
}
```

Once this method returns we need to determine which value is positive because that is the value that will be helpful in decryption. Once we see which is positive then we need to add the value of (p-1)(q-1) to it. Then module by (p-1)(q-1). This will give us the last part of the private key to start decryption.

```
public static void decryption(long num, long pq)
{
   count  =0;
   Scanner input = new Scanner (System.in);
   result = new long [100];
   RSA decode = new RSA();
   System.out.println("Please enter the message");
   String phrase1 = input.nextLine();
   splitPhrase(phrase1);
   for(int i = 0; i<list.length;i++)
   {
      if(list[i] != null)
      {
         result[i] = decode.key(num,pq,(Long.parseLong(list[i])));
         count++;
      }
   }//end for loop
   for(int i = 0; i<count;i++)
   {
      result[i] = (result[i]%26);
   }
   for(int i = 0; i<count;i++)
   System.out.print(result[i] + " ");
}// end decryption
```

FIG 9.4: Decryption Method

We pass the private key to the decryption method and ask the user for the encrypted message. The message is inputted with a comma separating each part i.e.: 128,1040,129,1144,788,735,570,875. We want to split this message into parts which allow us to decode each part of the message. In addition, we use the Key Method from program 8, FIG 8.2 to find the decrypted value of each number. We pass the private key and the number to the key method and save the data to an array of long called results. Then we mod every number in the result array by 26 to get the final alphabetical value. Then we print this out.

```
Please enter the public key, first p*q then e
1147
7
Please enter the message
128,1040,129,1144,788,735,570,875

2 3 5 7 11 13 17 19
```

FIG 9.5: Results

Cryptographic Decoding Challenges

Challenge 1:

```
Enter the codeword:
toto
Enter the desired text to decrypt
Bqhierpvbzxoporhasacnflqhbyskfbbzkbhahasyzhkxflqhbliehbbzkbhahaskobbpwmvmvxhacnuahlwwpxhawgybbbqhierustbhhaskzbbvcebbtbcgzrvtrtpkobb
Icouldwhileawaythehoursconferringwiththeflowersconsultingwiththerainwiththethoughtsidbethinkinicouldbeanotherlincolnifionlyhadabrain
```

Challenge 2:

```
Enter the codeword:
woody
Enter the desired text to decrypt
boqwmnwooqssfhqkasrlagowraadwrkaonciohkjkcyhvywhllc
factorialsweresomeonesattempttomakemathlookexciting
```

Challenge 3:

```
Please enter the public key, first p*q then e
21971
10555
Please enter the message
16912,19531,20676,16912,6613,2348,17835,15770,15770
4 11 12 4 17 5 20 3 3
ELMERFUDD
```

Challenge 4:

```
Please enter the public key, first p*q then e
118513313
5555551
Please enter the message
80217189,107242213,96490860,79543571,25953566
51 36 5 31 17
```

Enrichment Essay

Both The Imitation Game and Breaking the Code were two rather enjoyable films. The Imitation Game was the more modern of the two, but Breaking the Code was more realistic in the way it presented the story in regards to Turing's life. Both of these two films exposed us to the life and work of Alan Turing and how he lead British code breakers in their efforts to crack the German Enigma.

During Breaking the Code there are a lot more scenes regarding Turing's sexuality, and how his actions got him into trouble with the police. In this film, we witness more interactions between Turing and his mother, and how their relationship grows throughout his life. Another relationship we see more of is Alan and Pat and how, in this version, she pronounces her love for him and he immediately tells her about his homosexuality. Throughout this film, Alan's sexuality plays a more present role, and we do not witness in detail how he works with his team to break Enigma. The only characters we witness him working with are Pat and his government handler, and only in the form of them discussing the problems at hand. In The Imitation Game, we see an interpretation of how Turing worked with his team to break Enigma

In the Imitation Game Turing is presented as a Mathematical genius that does not work well with others and is very full of himself. They make this part him present by flashing back and forth between his childhood school life, and his job interview with Commander Denniston. In this interpretation of Turing's life, we watch as he and his team overcome the problems and obstacles they are faced with in order to break Enigma. During this film, Turing's personal life regarding his relationship with his mother and his homosexuality is less prominent. We still hear about his sexuality and how it affects his life in the end, and how it plays a part in his relationship with Pat. In this film, we are also shown more of Pat in comparison to Breaking the Code. In this film, she gains her position by completing Alan's test before any man is able to, but despite the fact she is smarter than most she is still discriminated against because she is a woman.

Yes, The Imitation Game was a more Hollywood style movie, but in its own way, it allowed us, the viewers, to see how Alan's creation allowed the British to defeat the Germans. It also exposed us to the fact that despite the fact that they cracked Enigma they had to allow some of the German attacks to be carried out in order to keep the Germans in the dark about the fact that they cracked their code.

Movie Quiz Questions

1. In The Imitation Game Alan Turing names his computer Christopher after his teenage boyfriend

2. During The Imitation Game “Pat” beats all of the other men by completing Turing’s test the quickest and earns her cryptography job against all odds.

3. Turing chains his coffee cup to the radiator in Breaking the Code, and this action causes people to become uncomfortable with this action.

4. Turing points out that the ends of the pine cone remind him of Fibonacci numbers and begins to explain his thinking.

5. Later on in the film, we learn that Turing’s government handler was also a homosexual and did not tell Turing or even hint at it.

6. Turing does not actually marry Pat in The Imitation Games. He breaks her heart and tells her he was only using her for getting Christopher to work.

7. Christopher, unfortunately, dies during tuberculosis right after Turing grows feelings for him.

8. We meet Turing's mother in both of the movies when he brings Christopher home, but only in Breaking the Code does she appear again when meeting Pat

What We Liked About This Lab

This was a very interesting and enjoyable lab for our group to work through. The films that we screened were very enjoyable and it was really interesting to witness how mathematics and cryptography played a major role in the turn out of World War II. The concept of cryptography is very intriguing and creating a program to shift text so we can encode some message and then decode it was one of the more enjoyable concepts. It was also really fascinating to witness how the concepts and theories that we have learned in both Discrete Mathematics and Data Structures have been present and used throughout history. The idea of RSA intertwining fast exponentiation and Euclid's Algorithm was really interesting.

What We Did Not Like About This Lab

There is only one part of this lab that troubled us, which was problem 9. We couldn't figure out the exact way to convert the numbers that we decoded into the correct letters. But after playing around with the module value and the value we added to the integer to get the correct character we finally were able to see some words forming. Of course the challenges we difficult, hence they are challenges, but after finding the code word for number 1 and 2, which took some time to find, we were able to get the original text. Most of these codes were challenging but overall this lab was the most interesting so far.

What We Learned From This Lab

As a group, we learned a lot from this lab. Learning how a simple encode and decode program works, and the extreme of using that same logic to do RSA encryption/decryption is amazing. Euclid's algorithm and the fast modular exponentiation algorithms allow us to find the encoded output or the decoded output to a problem. While decoding we have to have a public key so we can find the private key. In addition, the private key is what is used to decode the encoded message. It was really interesting to witness how these theories have been used throughout history, and how they have evolved into useful tools for tasks like safely encrypting messages in emails and allowing the credit card information to be shared securely.