**Final Project:**

**(Simplified) El Gamal Encryption System**

The purpose of the project is to create a program in Java that can decrypt a ciphertext encrypted with a simplified version of El Gamal (with p limited to less than 100000). The system is built based on basic Java compiler with standard libraries and is not included Crypto or BigInteger libraries. The program consisted of three main components: The input processor, the ciphertext solver and the plaintext converter; with each of them contain multiple modules.

**The Input Processor**

Modules included:

* Method: getPositions(File file)
  + This method take in input as a .txt file that contained multiple pairs of C1,C2 under the format: (C1,C2); and store them as an array list of “positions” or points.
* Class: Point()
  + Class created for the purpose of storing values of C1, C2

**The Ciphertext Solver**

Modules included:

* Method: modExp(int a, int x, int p)
  + Perform modular exponentiation using the form a^x (mod p)
* Method: extendedEuclidean(int a, int p)
  + Perform extended Euclidean algorithm on a mod p: Used by the modInverse function to find the modular multiplicative inverse
* Method: modInverse(int a, int p) (used extendedEuclidean method)
  + Obtains the coefficients of ax+py=gcd(a,p). Assumes a is the base and p is the modular value, calculates the inverse for a.
* Method: findKey( int alpha, int beta, int p) (used modExp method)
  + Solves the discrete log problem. The key is found when beta = alpha^secretKey (mod p)
* Method: modpow(int value, int power, int mod)
  + Custom method works similar to modPow() in BigInteger library. It takes in a value, raise the value to the defined power then take mod of the result.
* Method: findMessage(int p, int x, int C1, int C2) (used modpow method)
  + Perform decryption on the cipher text with x being the key knows from findKey and p, C1, C2 are known variables.

**The Plaintext Converter:**

* Method: matchWord(int a)
  + This method take in an integer a and match such integer with the alphabet (with 0 matched with A) and return a String of such letter.
* Method: findWord(int message) (used matchWord)
  + This method takes in a decrypted result of pair of C1, C2 and print out group of three letters that matched with them. The process is as followed:
    - Let a= 26^2\*x + 26\*y + 1\*z be the decrypted integer and x, y, z the position of the letters that need to be matched
    - a mod 26 = z
    - Thus y = (a-z)/26 mod 26
    - Substitute z and y to the original function to find x
    - Run x, y, z to matchWord and print them out

**Outstanding method:** decrypt(List<Point> cipher, int p, int secret)

* This method acts as a loop that loop through the entire stored arrayList of pairs of C1, C2 to decrypt and print out the plaintext.

The Code:

public class Point {

private int x;

private int y;

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public Point() {

}

}

public class ElGamal {

public static void main(String[] args) throws IOException {

int p = 31847;

int alpha = 5;

int beta = 18074;

int secret = findKey(alpha, beta, p);

File cipher = new File("D:\\Documents\\math314\\ElGamal\\src\\elgamal\\Cipher.txt");

List<Point> listPoint = new ArrayList<Point>();

listPoint = getPositions(cipher);

decrypt(listPoint, p, secret);

}

/\* Performs modular exponentiation

\* Uses the form a^x (mod p) \*/

public static int modExp(int a, int x, int p) {

int r = 1;

while (x != 0) {

if (x % 2 == 1) {

r = (r \* a) % p;

}

x = x / 2;

a = (a \* a) % p;

}

return r;

}

/\* Performs the extended euclidean algorithm on a % p

\* Used by the modInverse function to find the modular multiplicative inverse \*/

public static int[] extendedEuclidean(int a, int p) {

int x = 0;

int y = 1;

int prev\_x = 1;

int prev\_y = 0;

while (p != 0) {

int quotient = a / p;

int temp = a;

a = p;

p = temp % p;

temp = x;

x = prev\_x - quotient \* x;

prev\_x = temp;

temp = y;

y = prev\_y - quotient \* y;

prev\_y = temp;

}

int res[] = {prev\_x, prev\_y, a};

return res;

}

/\* Obtains the coefficients of ax+py=gcd(a,p)

\* Assumes a is the base, and p is the modular value

\* Calculates the inverse for a \*/

public static int modInverse(int a, int p) {

int res[] = extendedEuclidean(a, p);

return (res[0] + p) % p;

}

/\* Solves the discrete log problem

\* The secretKey is found when beta == alpha^secretKey (mod p) \*/

public static int findKey(int alpha, int beta, int p) {

int secretKey = 1;

while (true) {

if (beta == modExp(alpha, secretKey, p)) {

return secretKey;

}

secretKey++;

}

}

/\* Find Message using Pair C1,C2\*/

public static int findMessage(int p, int x, int C1, int C2) {

// Calculating the C1 / C2 Formula

int p1x = 0;

p1x = p - x;

p1x = p1x - 1;

C1 = modpow(C1, p1x, p);

C2 = C2 % p;

int message;

message = C1 \* C2;

message = message % p;

return message;

}

/\* modPow method for int \*/

public static int modpow(int value, int power, int mod) {

int e = 1;

for (int i = 0; i < power; i++) {

e = ((e \* value) % mod);

}

return e;

}

/\*Translating result to text\*/

public static void findWord(int message) {

int x, y, z;

z = message % 26;

y = ((message - z) / 26) % 26;

x = ((message - z) - (y \* 26)) / (26 \* 26);

System.out.print(matchWord(x) + matchWord(y) + matchWord(z));

}

/\*Matching numbers with letters\*/

public static String matchWord(int a) {

String x = "A";

switch (a) {

case 0:

x = "A";

break;

case 1:

x = "B";

break;

case 2:

x = "C";

break;

case 3:

x = "D";

break;

case 4:

x = "E";

break;

case 5:

x = "F";

break;

case 6:

x = "G";

break;

case 7:

x = "H";

break;

case 8:

x = "I";

break;

case 9:

x = "J";

break;

case 10:

x = "K";

break;

case 11:

x = "L";

break;

case 12:

x = "M";

break;

case 13:

x = "N";

break;

case 14:

x = "O";

break;

case 15:

x = "P";

break;

case 16:

x = "Q";

break;

case 17:

x = "R";

break;

case 18:

x = "S";

break;

case 19:

x = "T";

break;

case 20:

x = "U";

break;

case 21:

x = "V";

break;

case 22:

x = "W";

break;

case 23:

x = "X";

break;

case 24:

x = "Y";

break;

case 25:

x = "Z";

break;

}

return x;

}

/\* Reads the cipher text from the file and separates the pairs into a list \*/

private static List<Point> getPositions(final File file)

throws FileNotFoundException, IOException {

if (file == null || !file.canRead()) {

throw new IllegalArgumentException("file not readable: " + file);

}

final Scanner s = new Scanner(file);

final List<Point> positions = new ArrayList<Point>();

while (s.hasNext()) {

String line[] = s.nextLine().replace('(', ' ').replace(')', ' ').split(",");

positions.add(new Point(parseInt(line[0].trim()), parseInt(line[1].trim())));

}

return positions;

}

/\* Decrypts the cipher text once given p, the secret key, and the cipher text pair \*/

public static void decrypt(List<Point> cipher, int p, int secret) {

for (int i = 0; i < cipher.size(); i++) {

findWord(findMessage(p, secret, cipher.get(i).getX(), cipher.get(i).getY()));

}

}

}

The Plaintext: SHESTANDSUPINTHEGARDENWHERESHEHASBEENWORKINGANDLOOKSINTOTHEDISTANCESHEHASSENSEDACHANGEINTHEWEATHERTHEREISANOTHERGUSTOFWINDABUCKLEOFNOISEINTHEAIRANDTHETALLCYPRESSESSWAYSHETURNSANDMOVESUPHILLTOWARDSTHEHOUSECLIMBINGOVERALOWWALLFEELINGTHEFIRSTDROPSOFRAINONHERBAREARMSSHECROSSESTHELOGGIAANDQUICKLYENTERSTHEHOUSE