|  |
| --- |
| Университет итмо |
| Лабораторная работа №3 по дисциплине «Информационная безопасность» |
| «Шифрование открытого текста на основе эллиптических кривых» |
| Вариант 21  Группа Р3402 |
| **Выполнили:**  **Лалетина Екатерина Александровна**  **Орлова Кристина Александровна** |
| **Преподаватель: Маркина Татьяна Анатольевна** |

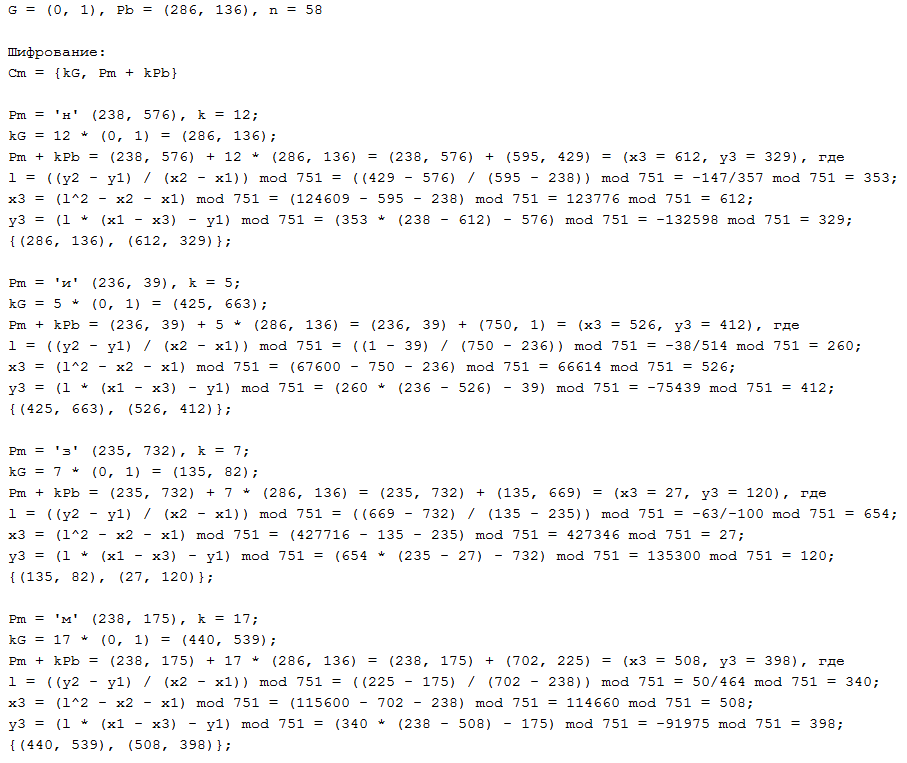
|  |
| --- |
| *05.04.20* |

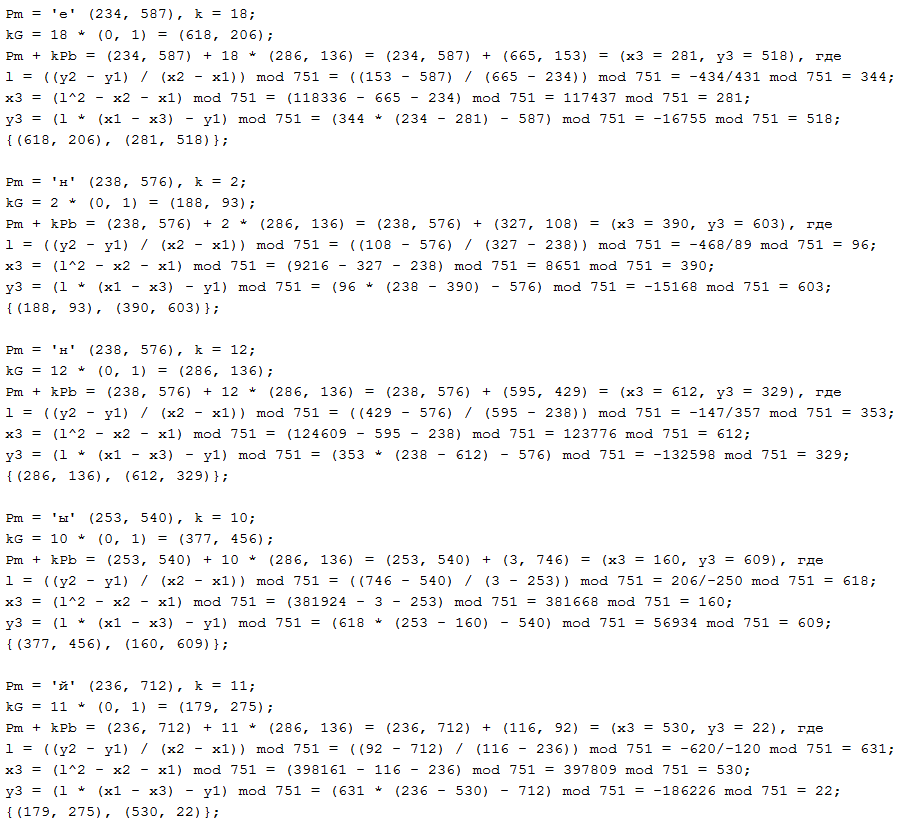
**Цель работы:** зашифровать открытый текст, используя алфавит, приведенный в [4], в подразделе «Задачи к лабораторным работам по криптографии на эллиптических кривых (используется кривая E751(-1,1) − и генерирующая точка G = (0, 1))»

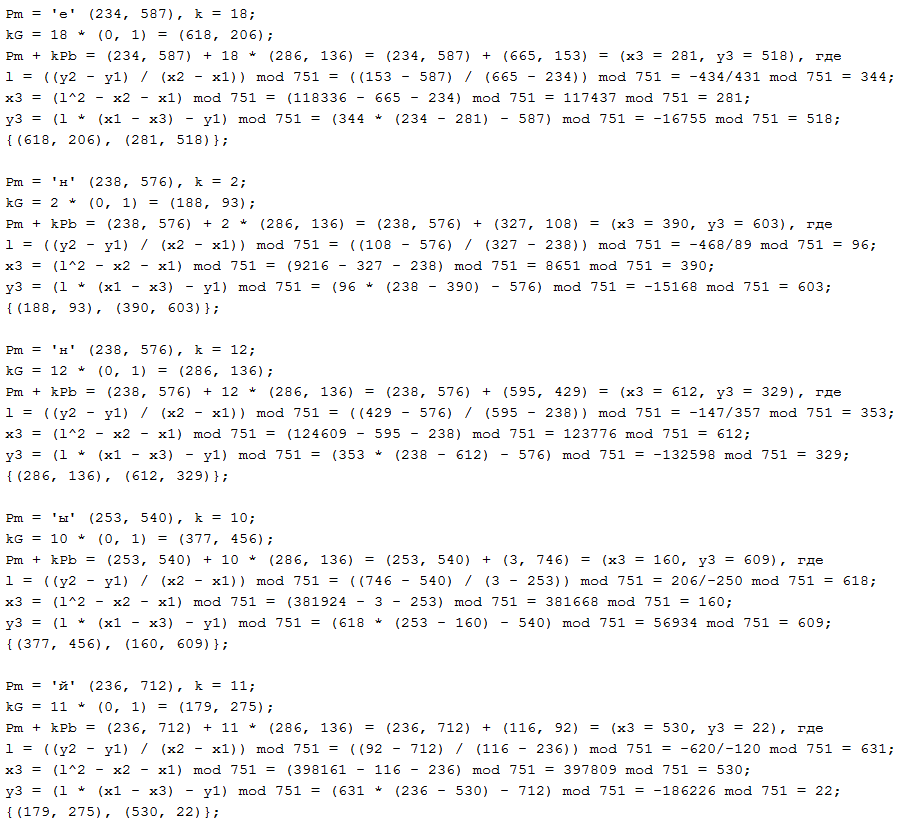
**Задание**

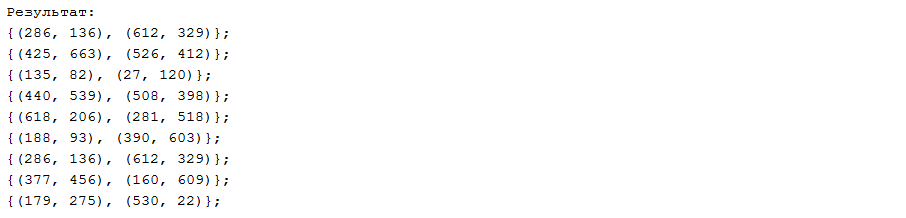


**Результаты работы программы**

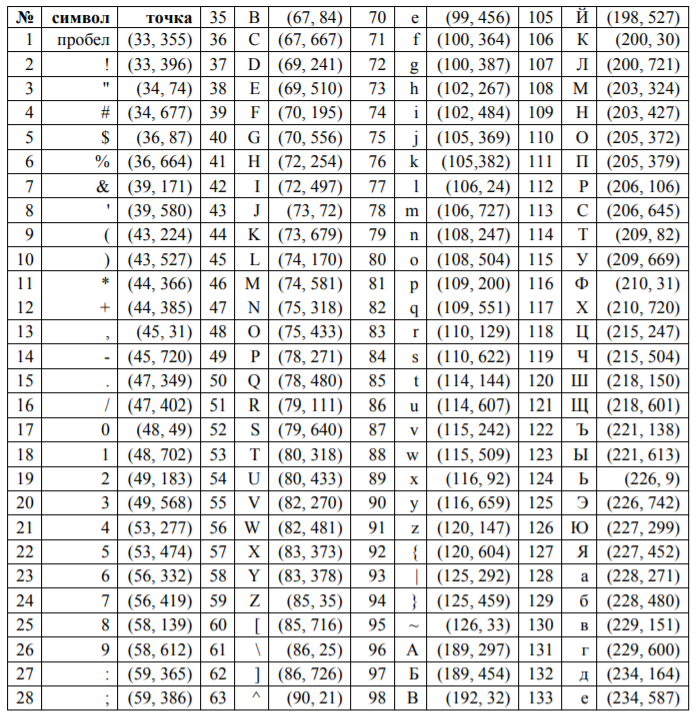


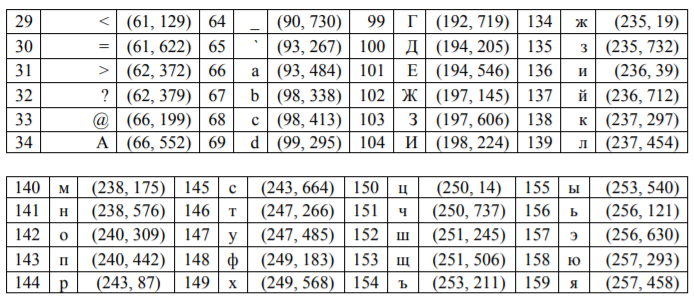






**Алфавит**





**Листинг разработанной программы**

**Класс Main.java:**

**package** Main;  
  
**import static** Structure.Alphabet.\*;  
**import static** java.lang.Math.*abs*;  
**import static** Main.Log.\*;  
  
**import** Structure.Point;  
  
**public class** Main {  
 **public static final int *a*** = -1;  
 **static final int *secretKey*** = 58;  
 **static final** Point ***openKey*** = **new** Point(286, 136);  
 **static final** Point ***G*** = **new** Point(0, 1);  
 **static int** *module* = 751;  
  
 **public static void** main(String[] args) {  
 **final** String plainText = **"низменный"**;  
 **final int**[] k = {12, 5, 7, 17, 18, 2, 12, 10, 11};  
 **final** Point[][] cipherText = {  
 {**new** Point(16, 416), **new** Point(93, 484)},  
 {**new** Point(489, 468), **new** Point(531, 397)},  
 {**new** Point(188, 93), **new** Point(654, 102)},  
 {**new** Point(489, 468), **new** Point(218, 150)},  
 {**new** Point(16, 416), **new** Point(530, 729)},  
 {**new** Point(425, 663), **new** Point(295, 219)},  
 {**new** Point(725, 195), **new** Point(742, 299)},  
 {**new** Point(188, 93), **new** Point(367, 360)},  
 {**new** Point(188, 93), **new** Point(235, 732)},  
 {**new** Point(618, 206), **new** Point(251, 245)},  
 {**new** Point(425, 663), **new** Point(688, 10)}  
 };  
  
 *logInitialParams*();  
 *encode*(plainText, k);  
 *decode*(cipherText);  
 }  
  
 */\*\*  
 \* Поиск математически корректного остатка от деления  
 \* - целого числа  
 \* - дроби  
 \* - отрицательного числа  
 \*  
 \** ***@param numerator*** *- числитель  
 \** ***@param denominator*** *- знаменатель (если не дробь, ставить в 1)  
 \** ***@return*** *возвращает остаток от деления  
 \*/* **public static int** mod(**int** numerator, **int** denominator) {  
 **final boolean** sign = (**double**) numerator / denominator < 0;  
 **long** mod;  
  
 numerator = *abs*(numerator);  
 denominator = *abs*(denominator);  
 *module* = *abs*(*module*);  
  
 **if** (denominator == 1) {  
 mod = numerator % *module*;  
 } **else** {  
 mod = (numerator \* *pow\_mod*(denominator, *module* - 2)) % *module*;  
 }  
  
 **if** (sign) {  
 mod = (mod == 0) ? mod : *module* - mod;  
 }  
  
 **return** (**int**) mod;  
 }  
  
 */\*\*  
 \* Шифрование  
 \* Выводит результат на стандартный поток вывода  
 \*  
 \** ***@param plainText*** *- открытый текст  
 \** ***@param k*** *- массив случайных чисел для символов plainText  
 \*/* **private static void** encode(String plainText, **int**[] k) {  
 Point[][] cipherText = **new** Point[plainText.length()][2];  
  
 *logEncodingFormula*();  
  
 **for** (**int** i = 0; i < plainText.length(); i++) {  
 **final** String symbol = String.*valueOf*(plainText.charAt(i));  
 **final** Point point = *getPoint*(symbol);  
 **final** Point temp;  
  
 cipherText[i][0] = ***G***.multiply(k[i]);  
 temp = ***openKey***.multiply(k[i]);  
 cipherText[i][1] = point.add(temp);  
  
 *logEncodingTact*(symbol, point, cipherText[i][0], temp, cipherText[i][1], k[i]);  
 }  
  
 *logEncodingResult*();  
 }  
  
 */\*\*  
 \* Расшифрование  
 \* Выводит результат на стандартный поток вывода  
 \*  
 \** ***@param cipherText*** *- шифртекст  
 \*/* **private static void** decode(Point[][] cipherText) {  
 Point[] plainText = **new** Point[cipherText.**length**];  
  
 *logDecodingFormula*();  
  
 **for** (**int** i = 0; i < plainText.**length**; i++) {  
 **final** Point temp = cipherText[i][0].multiply(***secretKey***);  
 String symbol;  
  
 plainText[i] = cipherText[i][1].subtract(temp);  
 symbol = *getSymbol*(plainText[i]);  
  
 *logDecodingTact*(cipherText[i][1], cipherText[i][0], temp, plainText[i], symbol);  
 }  
  
 *logDecodingResult*();  
 }  
  
 */\*\*  
 \* Возведение в степень по модулю  
 \* Код взят с https://brestprog.by/topics/modulo/  
 \*  
 \** ***@param base*** *- то, что возводим в степень  
 \** ***@param currentModule*** *- текущий модуль (начинается с module - 2)  
 \** ***@return*** *возвращает результат возведения числа в степень по модулю  
 \*/* **private static long** pow\_mod(**long** base, **long** currentModule) {  
 **if** (currentModule == 1) {  
 **return** base;  
 }  
  
 **if** (currentModule % 2 == 0) {  
 **final long** t = *pow\_mod*(base, currentModule / 2);  
 **return** t \* t % *module*;  
 } **else** {  
 **return** *pow\_mod*(base, currentModule - 1) \* base % *module*;  
 }  
 }  
}

**Класс Point.java:**

**package** Structure;  
  
**import static** java.lang.Math.*pow*;  
**import static** Main.Log.*setBufParams*;  
**import static** Main.Main.*mod*;  
**import static** Main.Main.***a***;  
  
*/\*\*  
 \* Точка эллиптической кривой  
 \*/***public class** Point {  
 **public int x**;  
 **public int y**;  
  
 **public** Point(**int** x, **int** y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
  
 */\*\*  
 \* Умножение точки эллиптической кривой  
 \*  
 \** ***@param k*** *- коэффициент умножения  
 \** ***@return*** *возвращает точку, являющуюся произведением данной точки на k  
 \*/* **public** Point multiply(**int** k) {  
 **int** i = 2;  
 Point point3 = **this**;  
  
 **do** {  
 point3 = point3.redouble();  
 i \*= 2;  
 } **while** (i <= k);  
  
 i /= 2;  
  
 **while** (i < k) {  
 point3 = add(point3);  
 i++;  
 }  
  
 **return** point3;  
 }  
  
 */\*\*  
 \* Вычитание точек эллиптической кривой  
 \*  
 \** ***@param point*** *- точка, которую необходимо вычесть  
 \** ***@return*** *возвращает точку, являющуюся результатом разности точек  
 \*/* **public** Point subtract(Point point) {  
 point.**y** = -point.**y**;  
 **return** add(point);  
 }  
  
 */\*\*  
 \* Сложение точек эллиптической кривой  
 \*  
 \** ***@param point*** *- точка, которую необходимо добавить  
 \** ***@return*** *возвращает точку, являющуюся суммой точек  
 \*/* **public** Point add(Point point) {  
 **final int** numerator = point.**y** - **y**;  
 **final int** denominator = point.**x** - **x**;  
 **final int** lambda = *mod*(numerator, denominator);  
 **final int** x3 = *mod*((**int**) *pow*(lambda, 2) - **x** - point.**x**, 1);  
 **final int** y3 = *mod*(lambda \* (**x** - x3) - **y**, 1);  
  
 *setBufParams*(lambda, x3, y3);  
  
 **return new** Point(x3, y3);  
 }  
  
 */\*\*  
 \** ***@return*** *возвращает строковое представление точки  
 \*/* **public** String toString() {  
 **return "("** + **x** + **", "** + **y** + **")"**;  
 }  
  
 */\*\*  
 \* Сравнение точек на эквивалентность  
 \*  
 \** ***@param point*** *- точка, с которой необходимо выполнить сравнение  
 \** ***@return*** *возвращает true, если координаты точек равны, иначе - false  
 \*/* **boolean** equals(Point point) {  
 **return x** == point.**x** && **y** == point.**y**;  
 }  
   
 */\*\*  
 \* Удвоение точки эллиптической кривой  
 \*  
 \** ***@return*** *возвращает точку, являющуюся произведением данной точки на 2  
 \*/* **private** Point redouble() {  
 **final int** numerator = 3 \* (**int**) *pow*(**x**, 2) + ***a***;  
 **final int** denominator = 2 \* **y**;  
 **final int** lambda = *mod*(numerator, denominator);  
 **final int** x3 = *mod*((**int**) *pow*(lambda, 2) - 2 \* **x**, 1);  
 **final int** y3 = *mod*(lambda \* (**x** - x3) - **y**, 1);  
  
 **return new** Point(x3, y3);  
 }  
}

**Класс Log.java:**

**package** Main;  
  
**import** Structure.Point;  
  
**import** java.io.PrintStream;  
**import** java.io.UnsupportedEncodingException;  
  
**import static** Main.Main.\*;  
**import static** java.lang.Math.*pow*;  
  
*/\*\*  
 \* Вывод на стандартный поток вывода  
 \*/***public class** Log {  
 **private static final boolean *isOutputVerbose*** = **true**; *// Если стоит true, вывод будет с подробными расчетами* **private static** PrintStream *ps*; *// Необходим для вывода кириллицы в кодировке UTF-8* **private static** StringBuilder *result* = **new** StringBuilder(**"\nРезультат:\n"**);  
 **private static int** *lambda*;  
 **private static int** *lambdaSquare*;  
 **private static int** *x3*;  
 **private static int** *y3*;  
  
 **static** {  
 **try** {  
 *ps* = **new** PrintStream(System.***out***, **true**, **"UTF-8"**);  
 } **catch** (UnsupportedEncodingException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Установка параметров временного буфера из класса Point, полученных в результате вычислений  
 \*  
 \** ***@param lambdaFromPoint*** *- lambda, необходимая для расчета x3  
 \** ***@param x3FromPoint*** *- x3, абсцисса результата вычислений  
 \** ***@param y3FromPoint*** *- y3, ордината результата вычислений  
 \*/* **public static void** setBufParams(**int** lambdaFromPoint, **int** x3FromPoint, **int** y3FromPoint) {  
 *lambda* = lambdaFromPoint;  
 *lambdaSquare* = (**int**) *pow*(*lambda*, 2);  
 *x3* = x3FromPoint;  
 *y3* = y3FromPoint;  
 }  
  
 */\*\*  
 \* Вывод постоянных параметров системы  
 \*/* **static void** logInitialParams() {  
 System.***out***.printf(**"G = %s, Pb = %s, n = %d\n"**, ***G***.toString(), ***openKey***.toString(), ***secretKey***);  
 }  
  
 */\*\*  
 \* Вывод формулы шифрования  
 \*/* **static void** logEncodingFormula() {  
 *ps*.println(**"\nШифрование:\nCm = {kG, Pm + kPb}"**);  
 }  
  
 */\*\*  
 \* Подробный вывод расчетов шифрования одного символа открытого текста  
 \*  
 \** ***@param symbol*** *- символ, который необходимо зашифровать  
 \** ***@param point*** *- точка, соответствующая символу в алфавите  
 \** ***@param kg*** *- первая точка шифртекста  
 \** ***@param kPb*** *- результат умножения k на открытый ключ  
 \** ***@param PmkPb*** *- вторая точка шифртекста  
 \** ***@param k*** *- случайное число  
 \*/* **static void** logEncodingTact(String symbol, Point point, Point kg, Point kPb, Point PmkPb, **int** k) {  
 **final** String tactResult = String.*format*(**"{%s, %s};\n"**, kg.toString(), PmkPb.toString());  
  
 *result*.append(tactResult);  
  
 **if** (!***isOutputVerbose***) {  
 **return**;  
 }  
  
 *ps*.printf(**"\nPm = '%s' %s, k = %d;\n"**, symbol, point.toString(), k);  
 *ps*.printf(**"kG = %d \* %s = %s;\n"**, k, ***G***.toString(), kg.toString());  
 *ps*.printf(**"Pm + kPb = %s + %d \* %s = %s + %s = (x3 = %d, y3 = %d), где\n"**, point.toString(), k, ***openKey***.toString(),  
 point.toString(), kPb.toString(), PmkPb.**x**, PmkPb.**y**);  
 *ps*.printf(**"l = ((y2 - y1) / (x2 - x1)) mod %d = ((%d - %d) / (%d - %d)) mod %d = %d/%d mod %d = %d;\n"**,  
 *module*, kPb.**y**, point.**y**, kPb.**x**, point.**x**, *module*, kPb.**y** - point.**y**, kPb.**x** - point.**x**, *module*, *lambda*);  
 *ps*.printf(**"x3 = (l^2 - x2 - x1) mod %d = (%d - %d - %d) mod %d = %d mod %d = %d;\n"**, *module*, *lambdaSquare*, kPb.**x**,  
 point.**x**, *module*, *lambdaSquare* - kPb.**x** - point.**x**, *module*, *x3*);  
 *ps*.printf(**"y3 = (l \* (x1 - x3) - y1) mod %d = (%d \* (%d - %d) - %d) mod %d = %d mod %d = %d;\n"**, *module*, *lambda*,  
 point.**x**, *x3*, point.**y**, *module*, *lambda* \* (point.**x** - *x3*) - point.**y**, *module*, *y3*);  
 *ps*.print(tactResult);  
 }  
  
 */\*\*  
 \* Вывод шифртекста  
 \*/* **static void** logEncodingResult() {  
 *ps*.print(*result*);  
 *result* = **new** StringBuilder(**"\nРезультат:\n"**);  
 }  
  
 */\*\*  
 \* Вывод формулы расшифрования  
 \*/* **static void** logDecodingFormula() {  
 *ps*.println(**"\nРасшифрование:\nPm = (Pm + kPb) - nkG"**);  
 }  
  
 */\*\*  
 \* Подробный вывод расчетов расшифрования шифртекста для одного символа открытого текста  
 \*  
 \** ***@param PmkPb*** *- вторая точка шифртекста  
 \** ***@param kg*** *- первая точка шифртекста  
 \** ***@param minusNkG*** *- обратная точка для nkG  
 \** ***@param sum*** *- точка, являющаяся результатом расшифрования  
 \** ***@param symbol*** *- символ, соответствующий результатирующей точке в алфавите  
 \*/* **static void** logDecodingTact(Point PmkPb, Point kg, Point minusNkG, Point sum, String symbol) {  
 **final** Point nkG = **new** Point(minusNkG.**x**, -minusNkG.**y**);  
  
 *result*.append(symbol);  
  
 **if** (!***isOutputVerbose***) {  
 **return**;  
 }  
  
 *ps*.printf(**"\nkG = %s, Pm + kPb = %s;\n"**, kg.toString(), PmkPb.toString());  
 *ps*.printf(**"Pm = %s - %d \* %s = %s - %s = %s + %s = (x3 = %d, y3 = %d) = '%s', где\n"**, PmkPb.toString(), ***secretKey***,  
 kg.toString(), PmkPb.toString(), nkG.toString(), PmkPb.toString(), minusNkG.toString(), sum.**x**, sum.**y**, symbol);  
 *ps*.printf(**"l = ((y2 - y1) / (x2 - x1)) mod %d = ((%d - %d) / (%d - %d)) mod %d = %d/%d mod %d = %d;\n"**,  
 *module*, minusNkG.**y**, PmkPb.**y**, minusNkG.**x**, PmkPb.**x**, *module*, minusNkG.**y** - PmkPb.**y**, minusNkG.**x** - PmkPb.**x**, *module*, *lambda*);  
 *ps*.printf(**"x3 = (l^2 - x2 - x1) mod %d = (%d - %d - %d) mod %d = %d mod %d = %d;\n"**, *module*, *lambdaSquare*, minusNkG.**x**,  
 PmkPb.**x**, *module*, *lambdaSquare* - minusNkG.**x** - PmkPb.**x**, *module*, *x3*);  
 *ps*.printf(**"y3 = (l \* (x1 - x3) - y1) mod %d = (%d \* (%d - %d) - %d) mod %d = %d mod %d = %d;\n"**, *module*, *lambda*,  
 PmkPb.**x**, *x3*, PmkPb.**y**, *module*, *lambda* \* (PmkPb.**x** - *x3*) - PmkPb.**y**, *module*, *y3*);  
 *ps*.printf(**"'%s' %s\n"**, symbol, sum.toString());  
 }  
  
 */\*\*  
 \* Вывод открытого текста  
 \*/* **static void** logDecodingResult() {  
 *ps*.println(*result*);  
 *result* = **new** StringBuilder(**"\nРезультат:\n"**);  
 }  
}

**Класс Alphabet.java:**

**package** Structure;  
  
**import** java.util.Map;  
**import** java.util.HashMap;  
  
*/\*\*  
 \* Алфавит символов, в котором каждому символу сопоставлена точка эллиптической кривой  
 \*/***public class** Alphabet {  
 **private static final** Map<String, Point> ***alphabet*** = **new** HashMap<>();  
  
 **static** {  
 ***alphabet***.put(**" "**, **new** Point(33, 355));  
 ***alphabet***.put(**"!"**, **new** Point(33, 396));  
 ***alphabet***.put(**"\""**, **new** Point(34, 74));  
 ***…***  
 ***alphabet***.put(**"э"**, **new** Point(256, 630));  
 ***alphabet***.put(**"ю"**, **new** Point(257, 293));  
 ***alphabet***.put(**"я"**, **new** Point(257, 458));  
 }  
  
 */\*\*  
 \* Поиск символа в алфавите по соответствующей ему точке  
 \*  
 \** ***@param point*** *- точка, по которой идет поиск  
 \** ***@return*** *возвращает найденный символ  
 \*/* **public static** String getSymbol(Point point) {  
 StringBuilder symbol = **new** StringBuilder();  
  
 ***alphabet***.forEach((key, value) -> {  
 **if** (value.equals(point)) {  
 **if** (key.equals(**"И"**)) {  
 symbol.append(key.toLowerCase()); *// Исправление ошибки, связанной с UTF-8, для "И"* } **else** {  
 symbol.append(key);  
 }  
 }  
 });  
  
 **return** symbol.toString();  
 }  
  
 */\*\*  
 \* Поиск точки в алфавите по соответствующему ей символу  
 \*  
 \** ***@param symbol*** *- символ, по которому идет поиск  
 \** ***@return*** *возвращает найденную точку  
 \*/* **public static** Point getPoint(String symbol) {  
 **return *alphabet***.get(symbol);  
 }  
}