ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования

Национальный исследовательский университет
«Высшая школа экономики»

Московский институт электроники и математики им. А. Н. Тихонова

Кафедра компьютерной безопасности

Отчёт
по курсовой работе по дисциплине
“Программирование алгоритмов защиты информации”

Выполнил студент гр. СКБ-171
Юрасов Никита Андреевич

Москва 2020

Содержание

1 Введение

Этот отчет является результатом выполнения работы построения эллиптической кривой в форме Якоби и вычисления кратной точки на этой кривой.

Задание:

1. Необходимо:
   (a) Построить/выбрать точку P на кривой
   (b) Выбрать случайное значение k.
   (c) Реализовать операцию вычисления кратной точки $Q = [k]P$.
   (d) Провести тестирование программы
2. Для проведения тестирования необходимо
   (a) Проверить, что результирующая точка Q лежит на кривой.
   (b) Проверить, что $[q]P = \mathcal{O}$, где q — порядок группы точек.
   (c) Проверить, что $[q+1]P = P$ и $[q-1]P = -P$.
   (d) Для двух случайных k_1, k_2 проверить, что

$$[k_1]P + [k_2]P = [k_1 + k_2]P$$

2 Теория

Эллиптическая кривая в форме Якоби имеет следующий вид:

$$Y^2 = eX^4 - 2dX^2Z^2 + Z^4,$$

где параметры e и d – некоторые коэффициенты.
Точка $(X : Y : Z)$ – точка на эллиптической кривой, заданная в проективных координатах.
Параметры e, d и координаты $X, Y, Z \in F_p$, где p – простое и $p > 3$.
Найти параметры e и d можно из соотношений $e = \frac{-(3\theta^2+4a)}{16}$, $d = \frac{3\theta}{4}$, где θ являтся координатой точки второго порядка $(\theta, 0)$, принадлежащей кривой в краткой форме Вейерштрасса.

Кривая в краткой форме Вейерштрасса имеет вид:

$$y^2 \equiv x^3 - ax + b \mod p, \tag{1}$$

где a, b – параметры кривой, (x, y) – точки на заданной кривой; $a, b, x, y \in F_p$ и $4a^3 + 27b^2 \neq 0 \mod p$.

Кривая в форме искривленной Эдвардса имеет вид:

$$eu^2 + v^2 \equiv 1 + 2du^2v \mod p,$$

где $e, d, u, v \in F_p$, p – простое и $p > 3$; $ed(e - d) \neq 0 \mod p$.
Важно заметить, что параметры кривой Якоби и кривой Эдвардса являются **различными**.

Переход от координат кривой в форме искривленной Эдвардса к координатам краткой Вейерштрасса осуществляется по следующим формулам:

$$(u, v) \to (x, y) = \left(\frac{s(1+v)}{1-v} + t, \frac{s(1+v)}{(1-v)u}\right),$$

$$(x, y) \to (u, v) = \left(\frac{x-t}{y}, \frac{x-t-s}{x-t+s}\right)$$

Для нахождения координаты точки второго порядка $(\theta, 0)$ – θ, нужно подставить значение $y \mapsto 0$ в уравнение (1). Тогда решение уравнения будет равно θ.

Чтобы найти параметры кривой в форме квадратики Якоби, необходимо воспользоваться переходами к ней от краткой формы Вейерштрасса:

$$\begin{cases} (\theta, 0) & \longmapsto (0 : -1 : 1) \\ (x, y) & \longmapsto (2(x-\theta) : (2x+\theta)(x-\theta)^2 - y^2 : y) \\ \mathcal{O} & \longmapsto (0 : 1 : 1) \end{cases}$$

Определение 2.1. *Нейтральный элемент – такая точка $\mathcal{O}$, что выполняются следующие свойства:*

1. $\mathcal{O} + \mathcal{O} = \mathcal{O}$
2. $\mathcal{O} + P = P + \mathcal{O} = P$, *где P – точка на эллиптической кривой.*

Для эллиптической кривой в форме квадратики Якоби нейтральный элемент равен $(0 : 1 : 1)$.

Определение 2.2. *Обратным элементом к точке* $(X : Y : Z)$ *является* $(-X : Y : Z)$.

Определение 2.3. *Порядком точки* P *называется такое минимальное число* q, *что* $[q]P = 0$, *а также выполнняется следующее:*

1. $[q+1]P = P$
2. $[q-1]P = -P$

2.1 Арифметические операции

Можно определить две операции для элементов, принадлежащих аддитивной абелевой группе: сложение двух различных точек и удвоение одной точки. Для кривых в форме квадратики Якоби удвоение является операцией сложение точки с такой же.

Сложение
Формулы сложение двух точек $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$:
$X_3 = X_1Z_1Y_2 + Y_1X_2Z_2$
$Y_3 = (Z_1^2Z_2^2 + eX_1^2X_2^2)(Y_1Y_2 - 2dX_1X_2Z_1Z_2) + 2eX_1X_2Z_1Z_2(X_1^2Z_2^2 + Z_1^2X_2^2)$
$Z_3 = Z_1^2Z_2^2 - eX_1^2X_2^2$

<u>Алгоритм сложения:</u>
$T_1 \leftarrow X1$
$T_2 \leftarrow Y1$
$T_3 \leftarrow Z1$
$T_4 \leftarrow X2$
$T_5 \leftarrow Y2$
$T_6 \leftarrow Z2$
$T_7 \leftarrow T_1 \cdot T_3$
$T_7 \leftarrow T_2 + T_7$
$T8 \leftarrow T_4 \cdot T_6$
$T8 \leftarrow T_5 + T8$
$T_2 \leftarrow T_2 \cdot T_5$
$T_7 \leftarrow T_7 \cdot T8$
$T_7 \leftarrow T_7 - T_2$
$T_5 \leftarrow T_1 \cdot T_4$
$T_1 \leftarrow T_1 + T_3$
$T8 \leftarrow T_3 \cdot T_6$
$T_4 \leftarrow T_4 + T_6$
$T_6 \leftarrow T_5 \cdot T8$
$T_7 \leftarrow T_7 - T_6$
$T_1 \leftarrow T_1 \cdot T_4$
$T_1 \leftarrow T_1 - T_5$
$T_1 \leftarrow T_1 - T8$
$T_3 \leftarrow T_1 \cdot T_1$
$T_6 \leftarrow T_6 + T_6$

$T_3 \leftarrow T_3 - T_6$
$T_4 \leftarrow e \cdot T_6$
$T_3 \leftarrow T_3 \cdot T_4$
$T_4 \leftarrow d \cdot T_6$
$T_2 \leftarrow T_2 - T_4$
$T_4 \leftarrow T8 \cdot T8$
$T8 \leftarrow T_5 \cdot T_5$
$T8 \leftarrow e \cdot T8$
$T_5 \leftarrow T_4 + T8$
$T_2 \leftarrow T_2 \cdot T_5$
$T_2 \leftarrow T_2 + T_3$
$T_5 \leftarrow T_4 - T8$
$X3 \leftarrow T_7$
$Y3 \leftarrow T_2$
$Z3 \leftarrow T$

Нахождение кратной точки

Замена переменных для перехода от проективных координат к аффинным:

$$\begin{cases} x = \frac{X}{Z} \\ y = \frac{Y}{Z^2} \end{cases}$$

Пусть $P = (x, y)$ – точка на кривой, тогда $[k]P = \underbrace{P + P + ... + P}_{k \text{ раз}}$ – кратная точка, $k \in \mathbb{Z}$ и $0 \le k < q$.

Самый эффективный способ вычисления кратной точки это алгоритм "Лесенка Монтгомери".

Algorithm 1 Лесенка Монтгомери

```
получить двоичное представление k = (k_{n-1}, ..., k_0) = \sum_{i=0}^{n-1} k_i 2^i
определить Q = O, R = P
for i ← n − 1 to 0 do
    if k_i = 0 then
        вычислить R = R + Q и Q = [2]Q;
    end if
    if k_i = 1 then
        вычислить Q = Q + R и R = [2]R;
    end if
end for
определить в качестве результата Q
```

3 Использование библиотеки GMP. Описание функций.

В качестве основной библиотеки длинной арифметики использовалась библиотека GMP. Официальный сайт библиотеки: https://gmplib.org, где можно скачать исходные программы, инструкцию по установке и документацию. В этом разделе будут описаны функции, основные типы и структуры данных, которые были использованы в работе.

Все определения описаны в заголовочном файле <`gmpxx.h`>.

Основной тип данных, в котором может храниться большое число: `mpz_t`.

explicit `mpz_class::mpz_class`*(const char *s, int base = 0)* – функция создающая в памяти большое число. Первым параметром принимается срока, состоящая из последовательности цифр (большое число), второй параметр – система счисления указанного числа.

mpz_t `mpz_class::get_mpz_t`*()* – функция, определенная для типа `mpz_t`, вызывающая соответствующий объект библиотеки `GMP C`.

void `mpz_neg`*(mpz_t rop, const mpz_t op)* – устанавливает в качестве параметра *-op* параметр *rop*.

int `mpz_invert`*(mpz_t rop, const mpz_t op1, const mpz_t op2)* – вычисляет обратный элемент *op1* по модулю *op2* и записывает результат в *rop*.

int `mpz_sgn`*(const mpz_t op)* – вычисляет знак *op*; возвращает +1 если $op > 0$, 0, если $op = 0$ и -1, если $op < 0$.

mp_bitcnt_t `mpz_scan1`*(const mpz_t op, mp_bitcnt_t starting_bit)* – считывает *op* в двоичном виде, начиная с *starting_bit* в направлении наиболее значимых бит. Возвращает индекс первого попавшегося бита равного 1.

int `mpz_tstbit`*(const mpz_t op, mp_bitcnt_t bit_index)* – возвращает бит в *op*, стоящий на месте *bit_index*.

void `mpz_powm`*(mpz_t rop, const mpz_t base, const mpz_t exp, const mpz_t mod)* – сохраняет в *rop* результат возведения *base* в степень *exp* по модулю *mod*.

тип `gmp_randstate_t` – тип переменной для создания псевдослучайных чисел, используя библиотеку `GMP`, содержит в себе выбранный алгоритм и текущее состояние.

void `gmp_randinit_mt`*(gmp_randstate_t state)* – инициализирует *state* для алгоритма Вихрь Мерсенна.

void `mpz_urandomm`*(mpz_t rop, gmp_randstate_t state, const mpz_t n)* – генерирует равномерно распределенную дискретную случайную величину (целое число) в интервале $[0, n)$.

void `gmp_randclear`*(gmp_randstate_t state)* – очищает память, выделенную под *state*.

4 Сборка проекта

Ниже приведен код файла `CMakeList.txt`

```
make_minimum_required(VERSION 3.16)

roject(EllipticCurve LANGUAGES CXX)

et(CMAKE_CXX_STANDARD 11)

ind_package(PkgConfig REQUIRED)
kg_check_modules(GMP REQUIRED IMPORTED_TARGET gmpxx)

dd_executable(EllipticCurve main.cpp curve.cpp curve.h)

arget_link_libraries(EllipticCurve PkgConfig::GMP)
```

Важно отметить, что для компиляции и запуска программы, необходимо использовать стандарт **C++11**.
Чтобы скомпилировать проект, выполните следующие команды в командной строке из папки с проектом:

```
mkdir build
cd build
cmake ..
make
./EllipticCurve
```

5 Исходные коды проекта

Ссылка на репозиторий GitHub: https://github.com/NikitaYurasov/EllipticCurve

5.1 main.cpp

```
#include "curve.h"

#include <iostream>

void Testing() {
        Param prm;
        std::cout << "----------------------------------------------------------"
        "-------------------------------------------------------\n";
        std::cout << "Параметры из стандарта id-tc26-gost-3410-2012-256-ParamSetA:\n";
        std::cout << "p = " << prm.p << '\n'
        << "a = " << prm.a << '\n'
        << "x_base = " << prm.x_base << '\n'
        << "y_base = " << prm.y_base << '\n'
        << "q = " << prm.q << "\n\n";
        std::cout << "Предрасчитанный параметр theta = " << prm.theta << "\n";
        std::cout << "-----------------------------------------------------------"
        "------------------------------------------------------\n\n";

        std::cout << "-----------------------------------------------------------"
        "------------------------------------------------------\n";
        std::cout << "Параметры Квадрики Якоби:\n";
        JacobiCurve curve(prm);
        std::cout << "e = " << curve.e << '\n'
        << "d = " << curve.d << '\n'
        << "X_base = " << curve.X << '\n'
        << "Y_base = " << curve.Y << '\n'
        << "Z_base = " << curve.Z << "\n";
        std::cout << "-----------------------------------------------------------"
        "------------------------------------------------------\n\n";


        std::cout << "-------------------------------------------------------------"
        "----------------------------------------------------\n";
        std::cout << "ТЕСТ: ПРОВЕРКА ПРИНАДЛЕЖНОСТИ НЕЙТРАЛЬНОГО ЭЛЕМЕНТА\n";
        JacobiPoint E(0, 1, 1);
        std::cout << "Нейтральный элемент E:\n"
        << " - в проективных координатах:\n";
        ProjectiveRepr(E);
        std::cout << " - в аффинных координатах:\n";
        AffineRepr(E, curve);
        std::cout << "Ответ: ";
        if (CheckPoint(E, curve))
        std::cout << "Точка E находится на кривой\n";
        else
        std::cout << "Точка E не находится на кривой\n";
        std::cout << "--------------------------------------------------------------"
        "---------------------------------------------------\n\n";

        std::cout << "-----------------------------------------------------------"
        "------------------------------------------------------\n";
        std::cout << "ТЕСТ 2:\n";
        JacobiPoint P_base;
        P_base.X = curve.X;
```

```
    P_base.Y = curve.Y;
    P_base.Z = curve.Z;

    std::cout << "Порождающий элемент в аффинных координатах:\n";
    AffineRepr(P_base, curve);

    std::cout << "Ответ: ";
    if (CheckPoint(P_base, curve))
    std::cout << "Точка находится на кривой\n";
    else
    std::cout << "Точка не находится на кривой\n";
    std::cout << "--------------------------------------------------------"
    "----------------------------------------------------------\n\n";

    std::cout << "---------------------------------------------------------"
    "---------------------------------------------------------\n";
    std::cout << "ТЕСТ 3: ";
    std::cout << "E+P_base = P_base?\n";
    JacobiPoint P1(2, 2, 2);
    AddPoints(E, P_base, P1, curve);

    std::cout << "Ответ: ";
    if (CheckEqualPoints(P_base, P1, curve))
    std::cout << "E+P == P\n";
    else
    std::cout << "E+P != P\n";
    std::cout << "-------------------------------------------------------"
    "-----------------------------------------------------------\n\n";

    std::cout << "-------------------------------------------------------"
    "-----------------------------------------------------------\n";
    std::cout << "ТЕСТ 4:\n";
    std::cout << "Принадлежит ли точка P2=(5:1:4) кривой\n";
    std::cout << "P2 в аффинных:\n";
    JacobiPoint P2(5, 1, 4);
    AffineRepr(P2, curve);
    std::cout << "Ответ: ";
    if (CheckPoint(P2, curve))
    std::cout << "точка P2 находится на кривой\n";
    else
    std::cout << "точка P2 не находится на кривой\n";
    std::cout << "----------------------------------------------------"
    "--------------------------------------------------------------\n\n";

    std::cout << "------------------------------------------------------"
    "------------------------------------------------------------\n";
    std::cout << "ТЕСТ 5: ";
    std::cout << "qP = E?\n";
    JacobiPoint resPoint(1, 1, 1);
    kPowPoint(resPoint, P_base, curve, prm.q);
    std::cout << "нейтральный элемент в аффинных координатах:\n";
    AffineRepr(E, curve);
    std::cout << "qP в аффинных координатах:\n";
    AffineRepr(resPoint, curve);
    std::cout << "-------------------------------------------------------"
    "-----------------------------------------------------------\n\n";

    std::cout << "----------------------------------------------------------"
    "--------------------------------------------------------\n";
    std::cout << "ТЕСТ 6: ";
```

```
        std::cout << "[q+1]P = P и [q-1] = -P\n";
        mpz_class degree = prm.q + mpz_class(1);
        kPowPoint(resPoint, P_base, curve, degree);
        std::cout << "[q+1]P:\n";
        AffineRepr(resPoint, curve);
        std::cout << "P:\n";
        AffineRepr(P_base, curve);

        std::cout << "Ответ: ";
        if((CheckEqualPoints(resPoint, P_base, curve)))
        std::cout << "[q+1]P == P\n";
        else
        std::cout << "[q+1]P != P\n";

        degree = prm.q - mpz_class(1);
        kPowPoint(resPoint, P_base, curve, degree);
        std::cout << "[q-1]P:\n";
        AffineRepr(resPoint, curve);
        std::cout << "-P:\n";
        JacobiPoint negP;
        GetNegativePoint(negP, P_base);
        AffineRepr(negP, curve);

        std::cout << "Ответ: ";
        if (CheckEqualPoints(resPoint, negP, curve))
        std::cout << "[q-1]P == -P\n";
        else
        std::cout << "[q-1]P != -P\n";
        std::cout << "------------------------------------------------------------"
        "------------------------------------------------------\n\n";

        std::cout << "------------------------------------------------------------"
        "------------------------------------------------------\n";
        std::cout << "ТЕСТ 7: ";
        std::cout << "Вычисление [k]P при k = 100; P = P_base?\n";
        degree = 100;
        kPowPoint(resPoint, P_base, curve, degree);
        AffineRepr(resPoint, curve);

        std::cout << "Ответ: ";
        if (CheckPoint(resPoint, curve))
        std::cout << "точка [k]P находится на кривой\n";
        else
        std::cout << "точка [k]P не находится на кривой\n";
        std::cout << "-----------------------------------------------------------"
        "-------------------------------------------------------\n\n";

        std::cout << "------------------------------------------------------------"
        "------------------------------------------------------\n";
        std::cout << "Тест 8:\n";
        std::cout << "Случайное k в диапазоне 0 <= k < q\n";
        mpz_class k;
        gmp_randstate_t rnd_state;
        gmp_randinit_mt(rnd_state);
        mpz_urandomm(k.get_mpz_t(), rnd_state, prm.q.get_mpz_t());
        std::cout << "k: " << k << '\n';
        std::cout << "[k]P в аффинных координатах:\n";
        kPowPoint(resPoint, P_base, curve, k);
        AffineRepr(resPoint, curve);
```

```
        std::cout << "Ответ: ";
        if (CheckPoint(resPoint, curve))
        std::cout << "точка [k]P находится на кривой\n";
        else
        std::cout << "точка [k]P не находится на кривой\n";
        std::cout << "---------------------------------------------------------"
        "---------------------------------------------------------\n\n";

        std::cout << "----------------------------------------------------------"
        "--------------------------------------------------------\n";
        std::cout << "Тест 9: ";
        std::cout << "[k1]P + [k2]P = [k1 + k2]P?\n";
        mpz_class k1;
        mpz_class k2;
        std::cout << "Случайные k1 и k2\n";
        mpz_class maxrand("100000000000000000");
        mpz_urandomm(k1.get_mpz_t(), rnd_state, maxrand.get_mpz_t());
        mpz_urandomm(k2.get_mpz_t(), rnd_state, maxrand.get_mpz_t());
        std::cout << "k1 = " << k1 << '\n'
        << "k2 = " << k2 << '\n';
        k = k1 + k2;
        std::cout << "k = k1 + k2 = " << k << '\n';
        JacobiPoint res1(0, 1, 1);
        JacobiPoint res2(0, 1, 1);
        JacobiPoint res3(0, 1, 1);
        kPowPoint(res1, P_base, curve, k1);
        kPowPoint(res2, P_base, curve, k2);
        kPowPoint(res3, P_base, curve, k);
        AddPoints(res1, res2, resPoint, curve);

        std::cout << "Ответ: ";
        if (CheckEqualPoints(resPoint, res3, curve))
        std::cout << "[k1]P + [k2]P == [k1 + k2]P\n";
        else
        std::cout << "[k1]P + [k2]P != [k1 + k2]P\n";

        std::cout << "Ответ: ";
        if (CheckPoint(res3, curve))
        std::cout << "точка [k]P находится на кривой\n";
        else
        std::cout << "точка [k]P не находится на кривой\n";

        gmp_randclear(rnd_state);
}

int main() {
        Testing();
        return 0;
}
```

5.2 curve.h

Параметры по умолчанию взяты из стандарта Р 50.1.114-2016:

- $p = 115792089237316195423570985008687907853269984665640564039457584007913129639319_{10}$ – характеристика простого поля, над которым определяется эллиптическая кривая.

- $q = 28948022309329048855892746252171976963338560298092253442512153408785530358887_{10}$ – порядок подгруппы простого порядка группы точек эллиптической кривой.
- $a = 87789765485885808793369751294406841171614589925193456909855962166505018127157_{10}$ – параметр a кривой в краткой форме Вейерштрасса.
- $x = 65987350182584560790308640619586834712105545126269759365406768962453298326056_{10}$ – координата x точки P, которая является порождающим элементом.
- $y = 22855189202984962870421402504110399293152235382908105741749987405721320435292_{10}$ – координата y точки P, которая является порождающим элементом.
- $\theta = 454069018412434321972378083527459607666454479745512801572100703902391945898_{10}$ – параметр θ, который был предрасчитан по формуле (1)

Все эти параметры определены в заголовочном файле как строки, чтобы в последствии было можно их использовать в библиотеке GMP.

```
#ifndef NULL
#define NULL (void*)0
#endif

#ifndef CURVE_H
#define CURVE_H

#include <gmpxx.h>
#include <string>

// id-tc26-gost-3410-2012-256-ParamSetA:
#define p_str "115792089237316195423570985008687907853269984665640564039457584007913129639319"

#define a_str "87789765485885808793369751294406841171614589925193456909855962166505018127157"
#define x_base_str "65987350182584560790308640619586834712105545126269759365406768962453298326056"
#define y_base_str "22855189202984962870421402504110399293152235382908105741749987405721320435292"
#define q_str "28948022309329048855892746252171976963338560298092253442512153408785530358887"

#define theta_str "454069018412434321972378083527459607666454479745512801572100703902391945898"

/**
* Структура хранения параметров стандарта
*/
struct Param {
        Param();

        mpz_class p;
        mpz_class a;
        mpz_class x_base;
        mpz_class y_base;
        mpz_class q;
        mpz_class theta;
};

/**
* Структура хранения параметров эллиптической кривой в форме квадратики Якоби
*/
struct JacobiCurve {
        JacobiCurve(const Param &param);

```

```
        mpz_class Y = 0;
        mpz_class X = 0;
        mpz_class e = 0;
        mpz_class d = 0;
        mpz_class Z = 0;
        mpz_class p = 0;
};

/**
* Структура для хранения параметров (координат) точки
*/
struct JacobiPoint {
        JacobiPoint(const std::string &x, const std::string &y, const std::string &z);

        JacobiPoint(int x, int y, int z);

        JacobiPoint(mpz_class x, mpz_class y, mpz_class z);

        JacobiPoint() = default;

        mpz_class X;
        mpz_class Y;
        mpz_class Z;
};

/**
* Складывает две точки P1 и P2. Результат заносится в переменную (точку) P_res
* @param P1: ссылка на точку №1 для сложения
* @param P2: ссылка на точку №2 для сложения
* @param P_res: ссылка на точку, в которую будет записан результат
* @param curve: структура кривой типа JacobiCurve, в которой хранятся параметры текущей кривой
*/
void AddPoints(const JacobiPoint &P1, const JacobiPoint &P2, JacobiPoint &P_res,
                const JacobiCurve &curve);

/**
* Возведение точки P в степень degree. Используется алгоритм <<Лесенка Монтгомери>>
* @param kP: ссылка на точку, в которую будет записан результат
* @param P: ссылка на точку, которая будет возводиться в степень
* @param curve: структура, хранящая текущие параметры кривой
* @param degree: значение степени
*/
void kPowPoint(JacobiPoint &kP, const JacobiPoint &P, const JacobiCurve &curve,
                const mpz_class &degree);

/**
* Переводит точку из проективных координат в аффинные
* @param affine_repr: ссылка на точку, в которую будет записан результат в аффинных координатах
* @param P: ссылка на точку в проективных координатах
* @param curve: структура, хранящая текущие параметры кривой
*/
void AffineCast(JacobiPoint &affine_repr, const JacobiPoint &P, const JacobiCurve &curve);

/**
* Выводит на экран координаты точки в аффинном представлении
* @param point: ссылка на точку
* @param curve: структура, хранящая текущие параметры кривой
*/
void AffineRepr(const JacobiPoint &point, const JacobiCurve &curve);

```

```
/**
* Выводит на экран координаты точки в проективном представлении
* @param P: ссылка на точку
*/
void ProjectiveRepr(const JacobiPoint &P);

/**
* Проверяет, лежит ли точка на кривой.
* Возвращает 1, если точка лежит на кривой, 0 -- в противном случае
* @param P: ссылка на точку
* @param curve: структура, хранящая текущие параметры кривой
* @return int
*/
int CheckPoint(const JacobiPoint &P, const JacobiCurve &curve);

/**
* Проверяет равны ли точки друг другу.
* Возвращает 1, если равны; 0 -- в противном случае
* @param P1 : ссылка на точку №1
* @param P2 : ссылка на точку №2
* @param curve : структура, хранящая текущие параметры кривой
* @return int
*/
int CheckEqualPoints(const JacobiPoint &P1, const JacobiPoint &P2, const JacobiCurve &curve);

/**
* Записываем в res -point
* @param res : ссылка на точку, в которую будет записан результат
* @param point : ссыка на точку, которую необходимо преставить в отрицательном виде
*/
void GetNegativePoint(JacobiPoint &res, const JacobiPoint &point);

#endif // CURVE_H
```

5.3 curve.cpp

```
#include "curve.h"

#include <iostream>
#include <limits>
#include <utility>

Param::Param()
        : p(p_str),
        a(a_str),
        x_base(x_base_str),
        y_base(y_base_str),
        q(q_str),
        theta(theta_str)
        {}

JacobiCurve::JacobiCurve(const Param &param) {
        this->p = param.p;

        this->e = (mpz_class(3) * param.theta * param.theta) % this->p;
        this->e += mpz_class(4) * param.a;
        mpz_neg(this->e.get_mpz_t(), this->e.get_mpz_t());
        mpz_class invert_16 = 16;
```

```cpp
        mpz_invert(invert_16.get_mpz_t(), invert_16.get_mpz_t(), this->p.get_mpz_t());
        this->e *= invert_16;
        this->e %= this->p;
        if (mpz_sgn(this->e.get_mpz_t()) == -1)
        this->e += this->p;

        this->d = mpz_class(3) * param.theta;
        mpz_class invert_4 = 4;
        mpz_invert(invert_4.get_mpz_t(), invert_4.get_mpz_t(), this->p.get_mpz_t());
        this->d *= invert_4;
        this->d %= this->p;

        mpz_class x_base_minux_theta = param.x_base - param.theta;
        this->X = (mpz_class(2) * x_base_minux_theta) % this->p;
        if (mpz_sgn(this->X.get_mpz_t()) == -1)
        this->X += this->p;

        this->Y = (2 * param.x_base + param.theta) % this->p;
        this->Y *= x_base_minux_theta;
        this->Y %= this->p;
        this->Y *= x_base_minux_theta;
        this->Y %= this->p;
        mpz_class y_base_sqr = (param.y_base * param.y_base) % this->p;
        this->Y -= y_base_sqr;
        this->Y %= this->p;
        if (mpz_sgn(this->Y.get_mpz_t()) == -1)
        this->Y += this->p;

        this->Z = param.y_base;
}

JacobiPoint::JacobiPoint(const std::string &x, const std::string &y, const std::string &z)
: X(x), Y(y), Z(z) {}

JacobiPoint::JacobiPoint(int x, int y, int z)
: X(x), Y(y), Z(z) {}

JacobiPoint::JacobiPoint(mpz_class x, mpz_class y, mpz_class z)
: X(std::move(x)), Y(std::move(y)), Z(std::move(z)) {}

void AddPoints(const JacobiPoint &P1, const JacobiPoint &P2, JacobiPoint &P_res,
                const JacobiCurve &curve) {
        mpz_class T1 = P1.X;
        mpz_class T2 = P1.Y;
        mpz_class T3 = P1.Z;
        mpz_class T4 = P2.X;
        mpz_class T5 = P2.Y;
        mpz_class T6 = P2.Z;
        mpz_class T7;
        mpz_class T8;

        T7 = (T1 * T3) % curve.p;
        T7 = (T7 + T2) % curve.p;
        T8 = (T4 * T6) % curve.p;
        T8 = (T8 + T5) % curve.p;
        T2 = (T2 * T5) % curve.p;
        T7 = (T7 * T8) % curve.p;
        T7 = (T7 - T2) % curve.p;
        T5 = (T1 * T4) % curve.p;
        T1 = (T1 + T3) % curve.p;
```

```
        T8 = (T3 * T6) % curve.p;
        T4 = (T4 + T6) % curve.p;
        T6 = (T5 * T8) % curve.p;
        T7 = (T7 - T6) % curve.p;
        T1 = (T1 * T4) % curve.p;
        T1 = (T1 - T5) % curve.p;
        T1 = (T1 - T8) % curve.p;
        T3 = (T1 * T1) % curve.p;
        T6 = (T6 + T6) % curve.p;
        T3 = (T3 - T6) % curve.p;
        T4 = (curve.e * T6) % curve.p;
        T3 = (T3 * T4) % curve.p;
        T4 = (curve.d * T6) % curve.p;
        T2 = (T2 - T4) % curve.p;
        T4 = (T8 * T8) % curve.p;
        T8 = (T5 * T5) % curve.p;
        T8 = (curve.e * T8) % curve.p;
        T5 = (T4 + T8) % curve.p;
        T2 = (T2 * T5) % curve.p;
        T2 = (T2 + T3) % curve.p;
        T5 = (T4 - T8) % curve.p;

        T7 %= curve.p;
        T2 %= curve.p;
        T5 %= curve.p;
        if (mpz_sgn(T7.get_mpz_t()) == -1)
        T7 += curve.p;
        if (mpz_sgn(T2.get_mpz_t()) == -1)
        T2 += curve.p;
        if (mpz_sgn(T5.get_mpz_t()) == -1)
        T5 += curve.p;

        P_res.X = T7;
        P_res.Y = T2;
        P_res.Z = T5;
}

void kPowPoint(JacobiPoint &kP, const JacobiPoint &P, const JacobiCurve &curve,
                const mpz_class &degree) {
        mp_bitcnt_t bit_count;
        for (mp_bitcnt_t i = 0; i != std::numeric_limits<mp_bitcnt_t>::max();
                        i = mpz_scan1(degree.get_mpz_t(), i + 1)) {
                bit_count = i;
        }
        ++bit_count;

        JacobiPoint R = P;
        JacobiPoint Q(0, 1, 1);

        for (int i = bit_count; i > 0; --i) {
                if (mpz_tstbit(degree.get_mpz_t(), i - 1)) {
                        AddPoints(Q, R, Q, curve);
                        AddPoints(R, R, R, curve);
                } else {
                        AddPoints(R, Q, R, curve);
                        AddPoints(Q, Q, Q, curve);
                }
        }

        kP = Q;
```

```
}

void AffineCast(JacobiPoint &affine_repr, const JacobiPoint &P, const JacobiCurve &curve) {
        mpz_class x;
        mpz_class y;
        mpz_class z;

        mpz_class z_inverted;

        mpz_invert(z_inverted.get_mpz_t(), P.Z.get_mpz_t(), curve.p.get_mpz_t());
        x = (z_inverted * P.X) % curve.p;
        if (mpz_sgn(x.get_mpz_t()) == -1)
        x += curve.p;

        y = (z_inverted * z_inverted) % curve.p;
        y = (y * P.Y) % curve.p;
        if (mpz_sgn(y.get_mpz_t()) == -1)
        y += curve.p;

        z = 0;

        affine_repr = {std::move(x), std::move(y), std::move(z)};
}

void AffineRepr(const JacobiPoint &point, const JacobiCurve &curve) {
        JacobiPoint affine_point(0, 1, 1);

        AffineCast(affine_point, point, curve);
        std::cout << "x = " << affine_point.X << "\ny = " << affine_point.Y << "\n\n";
}

void ProjectiveRepr(const JacobiPoint &P) {
        std::cout << "X = " << P.X << "\nY = " << P.Y << "\nZ = " << P.Z << "\n\n";
}

int CheckPoint(const JacobiPoint &P, const JacobiCurve &curve) {
        mpz_class left;
        mpz_class right;
        mpz_class buf1;
        mpz_class buf2 = 4;

        left = (P.Y * P.Y) % curve.p;
        mpz_powm(right.get_mpz_t(), P.X.get_mpz_t(), buf2.get_mpz_t(), curve.p.get_mpz_t());
        right = (right * curve.e) % curve.p;
        mpz_powm(buf2.get_mpz_t(), P.Z.get_mpz_t(), buf2.get_mpz_t(), curve.p.get_mpz_t());
        right += buf2;
        buf2 = (P.X * P.Z) % curve.p;
        buf2 = (buf2 * buf2) % curve.p;
        buf2 = (curve.d * buf2) % curve.p;
        buf2 += buf2;
        right -= buf2;
        buf1 = (left - right) % curve.p;

        int ans = (buf1 == 0);

        return ans;
}

int CheckEqualPoints(const JacobiPoint &P1, const JacobiPoint &P2, const JacobiCurve &curve) {
        JacobiPoint affineP1(0, 1, 1);
```

```cpp
        JacobiPoint affineP2(0, 1, 1);
        AffineCast(affineP1, P1, curve);
        AffineCast(affineP2, P2, curve);

        int ans;
        if (affineP1.X == affineP2.X && affineP1.Y == affineP2.Y)
        ans = 1;
        else
        ans = 0;

        return ans;
}

void GetNegativePoint(JacobiPoint &res, const JacobiPoint &point) {
        res = point;
        mpz_neg(res.X.get_mpz_t(), res.X.get_mpz_t());
}
```

6 Вывод программы

———————————————————————————————————

Параметры из стандарта id-tc26-gost-3410-2012-256-ParamSetA:
p = 115792089237316195423570985008687907853269984665640564039457584007913129639319
a = 87789765485885808793369751294406841171614589925193456909855962166505018127157
x_base = 65987350182584560790308640619586834712105545126269759365406768962453298326056
y_base = 22855189202984962870421402504110399293152235382908105741749987405721320435292
q = 28948022309329048855892746252171976963338560298092253442512153408785530358887

Предрасчитанный параметр theta = 45406901841243432197237808352745960766645447974551280157
———————————————————————————————————

———————————————————————————————————

Параметры Квадрики Якоби:
e = 21881292613901449512659201470451780075363042554712173057987834765447108787084
d = 58236596382467423453264776066989548632384833192629416620907867531883358779083
X_base = 15274473091028057513101540063430842355608196627407929088211752509188683120997
Y_base = 70639478069546534592066422814913955506998300889114271757947051176576672450210
Z_base = 22855189202984962870421402504110399293152235382908105741749987405721320435292
———————————————————————————————————

———————————————————————————————————

ТЕСТ: ПРОВЕРКА ПРИНАДЛЕЖНОСТИ НЕЙТРАЛЬНОГО ЭЛЕМЕНТА
Нейтральный элемент E:
– в проективных координатах:
X = 0
Y = 1
Z = 1

- в аффинных координатах:
x = 0
y = 1

Ответ: Точка E находится на кривой

ТЕСТ 2:
Порождающий элемент в аффинных координатах:
x = 26
y = 32588803023257230788452318859724590706198019287541469357859214741485052675122

Ответ: Точка находится на кривой

ТЕСТ 3: E+P_base = P_base?
Ответ: E+P == P

ТЕСТ 4:
Принадлежит ли точка P2=(5:1:4) кривой
P2 в аффинных:
x = 28948022309329048855892746252171976963317496166410141009864396001978282409831
y = 65133050195990359925758679067386948167464366374422817272194891004451135422117

Ответ: точка P2 не находится на кривой

ТЕСТ 5: qP = E?
нейтральный элемент в аффинных координатах:
x = 0
y = 1

qP в аффинных координатах:
x = 0
y = 1

ТЕСТ 6: [q+1]P = P и [q-1] = -P
$[q+1]$ P:
x = 26
y = 32588803023257230788452318859724590706198019287541469357859214741485052675122

P:
x = 26
y = 32588803023257230788452318859724590706198019287541469357859214741485052675122

Ответ: [q+1]P == P

$[q-1]$P:
x = 115792089237316195423570985008687907853269984665640564039457584007913129639293
y = 32588803023257230788452318859724590706198019287541469357859214741485052675122

-P:
x = 115792089237316195423570985008687907853269984665640564039457584007913129639293
y = 32588803023257230788452318859724590706198019287541469357859214741485052675122

Ответ: [q-1]P == -P

ТЕСТ 7: Вычисление [k]P при k = 100; P = P_base?
x = 46114831014247229923266331647927557586696495636126505757008735063481431609683
y = 38376220474406473655225685664497454497247526062573712862044892681609942213050

Ответ: точка [k]P находится на кривой

Тест 8:
Случайное k в диапозоне 0 <= k < q
k: 11283119821468158366191662829437219657438451067251718398758163510548403484771
$[k]$P в аффинных координатах:
x = 101490730742528333557806746127586592797289596879569381848136131261935915744108
y = 9851758315897559305814150804814137881662431171517574207703160923019686155389

Ответ: точка [k]P находится на кривой

Тест 9: [k1]P + [k2]P = [k1 + k2]P?
Случайные k1 и k2
k1 = 69631175459917429
k2 = 4314297476529749
k = k1 + k2 = 73945472936447178
Ответ: [k1]P + [k2]P == [k1 + k2]P
Ответ: точка [k]P находится на кривой

Process finished with exit code 0