

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 группы СКБ 172
Синицын Константин Алексеевич

Москва 2020

Содержание

| | | |
|----------|---|-----------|
| 1 | Введение | 2 |
| 2 | Теоретическая часть | 3 |
| 2.1 | Квадрика Якоби | 3 |
| 2.2 | Арифметические операции | 4 |
| 2.2.1 | Сложение | 4 |
| 2.2.2 | Нахождение кратной точки | 5 |
| 3 | Работа с библиотекой libtommath. Описание функций. | 7 |
| 4 | Описание основных функций и структур | 9 |
| 4.1 | Основные структуры | 9 |
| 4.2 | Основные функции | 10 |
| 4.2.1 | Функции, обеспечивающие арифметику по модулю числа | 10 |
| 4.2.2 | Функции инициализации основных структур | 11 |
| 4.2.3 | Функции освобождения памяти, использующейся для хранения основных структур | 11 |
| 4.2.4 | Функции для вывода координат точки на экран | 11 |
| 4.2.5 | Основные функции | 12 |
| 5 | Тестирование | 12 |
| 5.1 | Установка и запуск реализации | 12 |
| 5.2 | Исходные данные | 13 |
| 5.3 | Результаты | 14 |
| 5.3.1 | Проверка на утечки памяти | 14 |
| 5.3.2 | Тестирование реализации | 15 |
| 6 | Использованная литература | 18 |

1 Введение

Данный отчет является результатом выполнения работы по созданию программной реализации алгоритма вычисления кратной точки на эллиптической кривой в форме квадрики Якоби.

Задание:

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

2 Теоретическая часть

2.1 Квадрика Якоби

Эллиптическая кривая в форме квадрики Якоби имеет следующий вид:

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

где e, d - некоторые коэффициенты, $(X : Y : Z)$ - точка на данной кривой, заданная в проективных координатах; $e, d, X, Y, Z \in F_p$, где p - простое и $p > 3$.

Кривая в краткой форме Вейерштрасса (афинная форма) имеет следующий вид:

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

где a, b - параметры кривой в краткой форме Вейерштрасса, (x, y) - точка на данной кривой, заданная в афинных координатах; $a, b, x, y \in F_p$ и $4a^3 + 27b^2 \neq 0 \pmod{p}$.

Для перехода от проективных координат $(X : Y : Z)$ к афинным координатам (x, y) можно воспользоваться следующими формулами:

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

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

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

Здесь θ - координата точки второго порядка $(\theta, 0)$, принадлежащей кривой в краткой форме Вейерштрасса. Зная значение θ , можно найти значения параметров e и d согласно формулам $e = \frac{-(3\theta^2 + 4a)}{16}$, $d = \frac{3\theta}{4}$.

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

Определение 3. Порядком точки P называется такое минимальное число q , что $[q]P = \mathcal{O}$, а также выполняется следующее:

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

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

2.2 Арифметические операции

Поскольку точки данной кривой принадлежат аддитивной абелевой группе, для них можно определить операции *сложения* двух различных точек и *удвоения* одной точки. Для кривой в форме квадратики Якоби удвоение является операцией сложения точки с самой собой, поэтому можно обойтись только операцией сложения.

2.2.1 Сложение

Для кривой в форме квадратики Якоби формулы сложения двух точек $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ выглядят следующим образом:

$$\begin{cases} X_3 = X_1 Z_1 Y_2 + Y_1 X_2 Z_2 \\ Y_3 = (Z_1^2 Z_2^2 + e X_1^2 X_2^2)(Y_1 Y_2 - 2d X_1 X_2 Z_1 Z_2) + 2e X_1 X_2 Z_1 Z_2 (X_1^2 Z_2^2 + Z_1^2 X_2^2) \\ Z_3 = Z_1^2 Z_2^2 - e X_1^2 X_2^2 \end{cases}$$

Алгоритм сложения двух точек:

$$\overline{T_1 \leftarrow X_1; T_2 \leftarrow Y_1; T_3 \leftarrow Z_1; T_4 \leftarrow X_2; T_5 \leftarrow Y_2; T_6 \leftarrow Z_2}$$

$$\begin{array}{ll}
T_7 \leftarrow T_1 \cdot T_3 & (= X_1 Z_1) \\
T_7 \leftarrow T_2 + T_7 & (= X_1 Z_1 + Y_1) \\
T_8 \leftarrow T_4 \cdot T_6 & (= X_2 Z_2) \\
T_8 \leftarrow T_5 + T_8 & (= X_2 Z_2 + Y_2) \\
T_2 \leftarrow T_2 \cdot T_5 & (= Y_1 Y_2) \\
T_7 \leftarrow T_7 \cdot T_8 & (= X_3 + Y_1 Y_2 + X_1 X_2 Z_1 Z_2) \\
T_7 \leftarrow T_7 - T_2 & (= X_3 + X_1 X_2 Z_1 Z_2) \\
T_5 \leftarrow T_1 \cdot T_4 & (= X_1 X_2) \\
T_1 \leftarrow T_1 + T_3 & (= X_1 + Z_1) \\
T_8 \leftarrow T_3 \cdot T_6 & (= Z_1 Z_2) \\
T_4 \leftarrow T_4 + T_6 & (= X_2 + Z_2) \\
T_6 \leftarrow T_5 \cdot T_8 & (= X_1 X_2 Z_1 Z_2) \\
T_7 \leftarrow T_7 - T_6 & (= X_3) \\
T_1 \leftarrow T_1 \cdot T_4 & (= X_1 Z_2 + X_2 Z_1 + X_1 X_2 + Z_1 Z_2) \\
T_1 \leftarrow T_1 - T_5 & (= X_1 Z_2 + X_2 Z_1 + Z_1 Z_2) \\
T_1 \leftarrow T_1 - T_8 & (= X_1 Z_2 + X_2 Z_1) \\
T_3 \leftarrow T_1 \cdot T_1 & (= X_1^2 Z_2^2 + X_2^2 Z_1^2 + 2X_1 X_2 Z_1 Z_2) \\
T_6 \leftarrow T_6 + T_6 & (= 2X_1 X_2 Z_1 Z_2) \\
T_3 \leftarrow T_3 - T_6 & (= X_1^2 Z_2^2 + X_2^2 Z_1^2) \\
T_4 \leftarrow e \cdot T_6 & (= 2eX_1 X_2 Z_1 Z_2) \\
T_3 \leftarrow T_3 \cdot T_4 & (= 2eX_1 X_2 Z_1 Z_2 (X_1^2 Z_2^2 + X_2^2 Z_1^2)) \\
T_4 \leftarrow d \cdot T_6 & (= 2dX_1 X_2 Z_1 Z_2) \\
T_2 \leftarrow T_2 - T_4 & (= Y_1 Y_2 - 2dX_1 X_2 Z_1 Z_2) \\
T_4 \leftarrow T_8 \cdot T_8 & (= Z_1^2 Z_2^2) \\
T_8 \leftarrow T_5 \cdot T_5 & (= X_1^2 X_2^2) \\
T_8 \leftarrow e \cdot T_8 & (= eX_1^2 X_2^2) \\
T_5 \leftarrow T_4 + T_8 & (= Z_1^2 Z_2^2 + eX_1^2 X_2^2) \\
T_2 \leftarrow T_2 \cdot T_5 & (= (Z_1^2 Z_2^2 + eX_1^2 X_2^2)(Y_1 Y_2 - 2dX_1 X_2 Z_1 Z_2)) \\
T_2 \leftarrow T_2 + T_3 & (= Y_3) \\
T_5 \leftarrow T_4 - T_8 & (= Z_3) \\
X_3 \leftarrow T_7; Y_3 \leftarrow T_2; Z_3 \leftarrow T_5 &
\end{array}$$

2.2.2 Нахождение кратной точки

Определение 4. Пусть P - точка на кривой, тогда $[k]P = \underbrace{P + P + \dots + P}_k$

- кратная точка, $k \in \mathbb{Z}, 0 \leq k < q$.

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

Algorithm 1 Лесенка Монтгомери

```
1: получить двоичное представление  $k = (k_{n-1}, \dots, k_0) = \sum_{i=0}^{n-1} k_i 2^i$ 
2: определить  $Q = \mathcal{O}, R = P$ 
3: for  $i \leftarrow n - 1$  to  $0$  do
4:   if  $k_i = 0$  then
5:     вычислить  $R = R + Q$  и  $Q = [2]Q$ ;
6:   end if
7:   if  $k_i = 1$  then
8:     вычислить  $Q = Q + R$  и  $R = [2]R$ ;
9:   end if
10: end for
11: определить в качестве результата  $Q$ 
```

3 Работа с библиотекой libtommath. Описание функций.

Основным типом данных в данной библиотеке является тип `mp_int`, предназначенный для хранения больших целых чисел. При написании реализации были использованы следующие функции библиотеки:

`int mp_init(mp_int * a)` - инициализирует структуру `mp_int a` и выделяет память для хранения большого числа.

`int mp_init_multi(mp_int * mp, ..., NULL)` - инициализирует несколько структур `mp_int` и выделяет память для хранения больших чисел.

`int mp_init_set(mp_int * a, mp_digit b)` - выделяет память и инициализирует `mp_int a` однозначным числом `mp_digit b`. В качестве аргумента также можно передавать небольшие числа (не более `short int`).

`int mp_init_copy(mp_int * a, mp_int * b)` - выделяет память и инициализирует `mp_int a` копией значения `mp_int b`.

`void mp_clear(mp_int * a)` - освобождает память, используемую для хранения структуры `mp_int a`.

`int mp_clear_multi(mp_int * mp, ..., NULL)` - освобождает память, используемые для хранения каждой из структур `mp_int`.

`int mp_cmp(mp_int * a, mp_int * b)` - производит знаковое сравнение чисел `mp_int a` и `mp_int b`. Возвращает `MP_EQ` в случае равенства, `MP_GT`, если $a > b$ и `MP_LT`, если $a < b$.

`int mp_add(mp_int * a, mp_int * b, mp_int * c)` - записывает в `c` результат суммы $a + b$.

`int mp_sub(mp_int * a, mp_int * b, mp_int * c)` - записывает в `c` результат разности $a - b$.

`int mp_neg(mp_int * a, mp_int * b)` - записывает в `b` значение `a` с про-

тивоположным знаком $(-a)$.

`int mp_div(mp_int * a, mp_int * b, mp_int * c, mp_int * d)` - делит с остатком a на b и записывает частное в c , остаток в d ($a = bc + d$). В случае ненужности $mp_int * c$ или $mp_int * d$ может быть заменено на `NULL` (для получения, соответственно, только частного или только остатка).

`int mp_mul(mp_int * a, mp_int * b, mp_int * c)` - помещает результат умножения a на b в c .

`int mp_exptmod(mp_int * G, mp_int * X, mp_int * P, mp_int * Y)` - возводит G в степень X и помещает результат по модулю P в Y ($Y \equiv GX(modP)$).

`int mp_to_radix(const mp_int * a, char * str, size_t maxlen, size_t * written, int radix)` - помещает mp_int a в строковом виде в `char str`. В `radix` указывается основание системы счисления (от 2 до 64), в `size_t maxlen` - максимальный размер, который может занять число в `char str`, в `size_t written` помещается размер реально записанного в `char str` числа.

`int mp_radix_size(mp_int * a, int radix, int *size)` - помещает в `size` целое число - размер, необходимый для помещения mp_int a в `char`. В `radix` указывается основание системы счисления (от 2 до 64).

`int mp_read_radix(mp_int * a, char *str, int radix)` - считывает число в строковом виде из `char str` и помещает в mp_int a . В `radix` указывается основание системы счисления (от 2 до 64).

`int mp_prime_random(mp_int * a, int t, int size, int bbs, ltm_prime_callback cb, void * dat)` - генерирует случайное простое число, превышающее 256^{size} и помещает его в mp_int a . При этом число должно пройти t тестов. В аргументах `int bbs`, `ltm_prime_callback cb`, `void * dat` указываются дополнительные флаги (не используется). Данная функция используется в программе для генерации случайного числа, поскольку библиотека `Libtommath` не содержит функций для генерации случайных чисел.

4 Описание основных функций и структур

Полные исходные коды можно найти в репозитории по ссылке <https://github.com/kasinit syn/JacobiQuadric>. Ниже представлено описание основных функций и структур.

4.1 Основные структуры

Структура **Point** - точка в проективных координатах (X, Y, Z) . Каждая из координат имеет тип **mp_int**.

```
struct Point
{
    mp_int X;
    mp_int Y;
    mp_int Z;
};
```

Структура **Parameters** содержит параметры, определенные стандартом (подробное описание параметров см. Исходные данные)

```
struct Parameters
{
    mp_int p;
    mp_int q;

    mp_int a;
    mp_int b;

    mp_int x_base;
    mp_int y_base;

    mp_int theta;
};
```

Структура `JacobiQuadric` содержит параметры квадрики Якоби: p - характеристика поля, e, d - параметры кривой, X, Y, Z - координаты порождающего элемента.

```
struct JacobiQuadric
{
    mp_int p;
    mp_int e;
    mp_int d;

    mp_int X;
    mp_int Y;
    mp_int Z;
};
```

4.2 Основные функции

4.2.1 Функции, обеспечивающие арифметику по модулю числа

В виду отсутствия в библиотеке `Libtommath` необходимых для математики эллиптических кривых функций, осуществляющих арифметику по модулю числа, в ходе выполнения данной работы были написаны соответствующие функции. Функции стилистически оформлены как функции библиотеки.

void mp_add_mod(mp_int * a, mp_int * b, mp_int * c, mp_int * mod)
- записывает в `c` результат суммы $a + b$ по модулю числа p .

void mp_sub_mod(mp_int * a, mp_int * b, mp_int * c, mp_int * mod)
- записывает в `c` результат разности $a - b$ по модулю числа p .

void mp_mul_mod(mp_int * a, mp_int * b, mp_int * c, mp_int * mod)
- записывает в `c` результат произведения $a * b$ по модулю числа p .

void mp_neg_mod(mp_int * a, mp_int * b, mp_int * mod) - записывает в `b` значение a с противоположным знаком ($-a$) по модулю числа p .

4.2.2 Функции инициализации основных структур

void InitPoint(struct Point * P, mp_int * x, mp_int * y, mp_int * z) - инициализация точки P с проективными координатами x, y, z.

void InitParameters(struct Parameters * Param, mp_int * p, mp_int * q, mp_int * a, mp_int * b, mp_int * x_base, mp_int * y_base, mp_int * theta) - инициализация структуры Param с параметрами.

void InitJacobiQuadric(struct JacobiQuadric * JQ, struct Parameters * Param) - инициализация структуры JQ с параметрами кривой.

4.2.3 Функции освобождения памяти, использующейся для хранения основных структур

void ClearPoint(struct Point * P) - освобождение памяти, использовавшейся для хранения точки в структуре P.

void ClearParameters(struct Parameters * Param) - освобождение памяти, использовавшейся для хранения параметров в структуре Param.

void ClearJacobiQuadric(struct JacobiQuadric * JQ) - освобождение памяти, использовавшейся для хранения параметров кривой Якоби в структуре JQ.

4.2.4 Функции для вывода координат точки на экран

void PrintPoint(struct Point * P) - вывести значения проективных координат точки P на экран.

void PrintPointAffine(struct Point * P, struct JacobiQuadric * JQ)
- вывести значения аффинных координат точки P на экран. Структура JQ
- данная кривая.

4.2.5 Основные функции

bool IsPointOnCurve(struct Point * P, struct JacobiQuadric * JQ)
- проверка находится ли данная точка P на данной кривой JQ. Возвращает true, если точка находится на кривой и false в противном случае.

void Addition(struct Point * P1, struct Point * P2, struct Point * P3, struct JacobiQuadric * JQ) - сложение двух точек $P1 + P2$, результат записывается в третью точку P3. Структура JQ - данная кривая.

bool ArePointsEqual(struct Point * P1, struct Point * P2, struct JacobiQuadric * JQ) - проверка равенства двух точек P1 и P2 на кривой JQ.

void MontgomeryLadder(struct Point * P, mp_int * k, struct Point * Q, struct JacobiQuadric * JQ) - реализация алгоритма "лесенка Монте-гомери где P - точка на кривой, k - степень точки, Q - результирующая точка $Q = [k]P$, JQ - данная кривая.

5 Тестирование

5.1 Установка и запуск реализации

Для установки библиотеки Libtommath следует воспользоваться официальной инструкцией по установке, приведенной в документации к библиотеке <https://github.com/libtom/libtommath>.

Для установки и запуска реализации следует выполнить следующие команды в командной строке из папки с проектом:

1. cmake CMakeLists.txt
2. make
3. ./JacodiQuadric

5.2 Исходные данные

Параметры для проверки работоспособности реализации и правильности ее выполнения были взяты из документа «*Рекомендации по стандартизации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов. Р 50.1.114 – 2016*». Для проверки из предложенных наборов параметров был взят набор параметров **id-tc26-gost-3410-2012-256-paramSetA**, содержащий следующие значения:

$p = 115792089237316195423570985008687907853269984665640564039457584007913129639319_{10}$
 $a = 87789765485885808793369751294406841171614589925193456909855962166505018127157_{10}$
 $b = 18713751737015403763890503457318596560459867796169830279162511461744901002515_{10}$
 $q = 28948022309329048855892746252171976963338560298092253442512153408785530358887_{10}$
 $x = 65987350182584560790308640619586834712105545126269759365406768962453298326056_{10}$
 $y = 22855189202984962870421402504110399293152235382908105741749987405721320435292_{10}$

Где:

p - характеристика простого поля, над которым определяются эллиптическая кривая;

a , b - параметры эллиптической кривой в форме Вейерштрасса (параметр b в реализации не используется);

q - порядок циклической подгруппы группы точек эллиптической кривой;

x , y - координаты порождающего элемента в краткой форме Вейерштрасса (в реализации обозначаются как x_base , y_base).

В документе значения параметров приведены в шестнадцатиричной системе счисления. Для перевода параметров в десятичную систему использовалась программа Wolfram Mathematica. Также, с помощью данной программы были вычислена x -координата точки $(\theta, 0)$ второго порядка в форме Вейерштрасса (значение θ):

$\theta = 454069018412434321972378083527459607666454479745512801572100703902391945898_{10}$

Ниже приведен фрагмент кода Wolfram Mathematica и результат выполнения вычислений:

```
In[1]:= p = Interpreter["HexInteger"] [{"00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97"}]
Interpreter

Out[1]= 115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 129 639 319

In[2]:= a = Interpreter["HexInteger"] [{"00C2173F1513981673AF4892C23035A27CE25E20138F95AA33B22C656F277E7335"}]
Interpreter

Out[2]= 87 789 765 485 885 808 793 369 751 294 406 841 171 614 589 925 193 456 909 855 962 166 505 018 127 157

In[3]:= b = Interpreter["HexInteger"] [{"295F9BAE7428ED9CC20E7C359A9D041A22FCCD9108E17BF7BA9337A6F8AE9513"}]
Interpreter

Out[3]= 18 713 751 737 015 403 763 890 503 457 318 596 560 459 867 796 169 830 279 162 511 461 744 901 002 515

In[7]:= q = Interpreter["HexInteger"] [{"40000000000000000000000000000000FD8CDDFC87B6635C115AF556C360C67"}]
Interpreter

Out[7]= 28 948 022 309 329 048 855 892 746 252 171 976 963 338 560 298 092 253 442 512 153 408 785 530 358 887

In[4]:= xbase = Interpreter["HexInteger"] [{"0091E38443A5E82C0D880923425712B2BB658B9196932E02C7882582FE742DAA28"}]
Interpreter

Out[4]= 65 987 350 182 584 560 790 308 640 619 586 834 712 105 545 126 269 759 365 406 768 962 453 298 326 056

In[5]:= ybase = Interpreter["HexInteger"] [{"32879423AB1A0375895786C4BB46E9565FDE0B5344766740AF268ADB32322E5C"}]
Interpreter

Out[5]= 22 855 189 202 984 962 870 421 402 504 110 399 293 152 235 382 908 105 741 749 987 405 721 320 435 292
```

```
In[8]:= Reduce[x^3 + a * x + b == 0, x, Modulus -> p]
[привести] [модуль]

Out[8]:= x == 454 069 018 412 434 321 972 378 083 527 459 607 666 454 479 745 512 801 572 100 703 902 391 945 898
```

5.3.1 Проверка на утечки памяти

Команда: `valgrind --leak-check=full ./JacobiQuadric`

```

==1756== HEAP SUMMARY:
==1756== in use at exit: 0 bytes in 0 blocks
==1756== total heap usage: 489,445 allocs, 489,445 frees, 55,656,115 bytes
allocated
==1756==
==1756== All heap blocks were freed – no leaks are possible
==1756==
==1756== For counts of detected and suppressed errors, rerun with: -v
==1756== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0

```

from 0)

5.3.2 Тестирование реализации

Затем было проведено тестирование самой реализации:

Команда: ./JacobiQuadric

Вывод:

Посчитанные параметры квадрики Якоби:

$d = 58236596382467423453264776066989548632384833192629416620907867531883358779083$

$e = 21881292613901449512659201470451780075363042554712173057987834765447108787084$

$X_base = 15274473091028057513101540063430842355608196627407929088211752509188683120997$

$Y_base = 70639478069546534592066422814913955506998300889114271757947051176576672450210$

$Z_base = 22855189202984962870421402504110399293152235382908105741749987405721320435292$

ТЕСТ 1: ПРОВЕРКА ПРИНАДЛЕЖНОСТИ НЕЙТРАЛЬНОГО ЭЛЕМЕНТА

Точка в проективных координатах:

$X = 0$

$Y = 1$

$Z = 1$

Точка в аффинных координатах:

$x = 0$

$y = 1$

Нейтральный элемент E принадлежит кривой

ТЕСТ 2: ПОРОЖДАЮЩИЙ ЭЛЕМЕНТ В АФФИННЫХ КООРДИНАТАХ

Точка в аффинных координатах:

$x = 26$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Порождающий элемент P_base принадлежит кривой

ТЕСТ 3: $E + P_base = P_base$?

Точка в проективных координатах:

$X = 46006328807261240983066223367402584890441940889817085927906907131943988570069$

$Y = 55579995469928774719018320141672621810816086559469103791209555426079733213871$

$Z = 50758435016066899640090271479344983510631222008148588090843858893038015946253$

Точка в аффинных координатах:

$x = 26$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Точка $E + P_base$ принадлежит кривой

Точки $E + P_base$ и P_base равны

ТЕСТ 4: Принадлежит ли точка $P2 = (5 : 1 : 4)$ кривой

Точка в аффинных координатах:

$x = 28948022309329048855892746252171976963317496166410141009864396001978282409831$

$y = 65133050195990359925758679067386948167464366374422817272194891004451135422117$

Точка $(5 : 1 : 4)$ не принадлежит кривой

ТЕСТ 5: $[q]P = E$?

Точка в проективных координатах:

$X = 0$

$Y = 40178936660781546849967672518756049964755243454469907618687078886735047096265$

$Z = 57742264586188110633053061231704778952292277256767593810644170837071465729572$

Точка в аффинных координатах:

$x = 0$

$y = 1$

Точки $[q]P$ и E равны

ТЕСТ 6: $[q+1]P = P$ и $[q-1]P = -P$

Точка в аффинных координатах:

$x = 115792089237316195423570985008687907853269984665640564039457584007913129639293$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Точка в аффинных координатах:

$x = 115792089237316195423570985008687907853269984665640564039457584007913129639293$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Точки $[q-1]P$ и $-P$ равны

Точка в аффинных координатах:

$x = 26$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Точка в аффинных координатах:

$x = 26$

$y = 32588803023257230788452318859724590706198019287541469357859214741485052675122$

Точки $[q+1]P$ и P равны

ТЕСТ 7: Вычисление $[k]P$ при $k = 100$

Точка в аффинных координатах:

$x = 46114831014247229923266331647927557586696495636126505757008735063481431609683$

$y = 38376220474406473655225685664497454497247526062573712862044892681609942213050$

Точка $[k]P$ принадлежит кривой

ТЕСТ 8: Вычисление $[k]P$ для случайного k из диапазона $[0, q)$

$k = 991954433999604731829632709224396598341591234772024487906631$

Точка в аффинных координатах:

$x = 50779116323969119300621785808242934425388155432437577476919529444328576423118$

$y = 94020197051731514972631394841409410785510879144286959132168853193003725895704$

Точка $[k]P$ принадлежит кривой

ТЕСТ 9: $[k_1]P + [k_2]P = [k_1 + k_2]P$?

$k_1 = 1084845348725810821418535502021$

$k_2 = 795405475617922960716810407137$

$k_1 + k_2 = 1880250824343733782135345909158$

Точка в аффинных координатах:

$x = 36783066602330481256214373320726812578572207207168637666900660686517300314330$

$y = 52106396355070439400592651537488559251130145451034852674912273346313496501149$

Точка $[k_1]P$ принадлежит кривой

Точка в аффинных координатах:

$x = 23653286548373740116138831789119419465516319104618009133532289868355943583259$

$y = 6521473322108346065594065622514635457973368003972073332546242861921339483508$

Точка $[k_2]P$ принадлежит кривой

Точка в аффинных координатах:

$x = 100174933671734223955453094649162785325397815042489168097357339866005748107089$

$y = 84966962613761404393860727171805411782744711102320690988699985888828907160639$

Точка $[k_1 + k_2]P$ принадлежит кривой

Точки $[k_1]P + [k_2]P$ и $[k_1 + k_2]P$ равны

ВСЕ ТЕСТЫ УСПЕШНО ПРОЙДЕНЫ

6 Используемая литература

1. Нестеренко А. Ю. – Курс лекций «Методы программной реализации СКЗИ»;
2. O. Billet, M. Joye. – «The Jacobi model of an elliptic curve and side-channel analysis, proceedings of AAEECC-15, Lecture Notes in Computer Science» <https://eprint.iacr.org/2002/125.pdf>;
3. «Рекомендации по стандартизации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов. Р 50.1.114 – 2016»;
4. Документация библиотеки libtommath «LibTomMath User Manual» <https://github.com/libtom/libtommath>.