Федеральное государственное автономное образовательное учреждение высшего профессионального образования
"Национальный исследовательский университет "Высшая школа экономики"

Московский институт электроники и математики им А.Н.Тихонова

Департамент прикладной математики

Кафедра компьютерной безопасности

ОТЧЕТ

по курсовой работе

по дисциплине «Программирование алгоритмов защиты информации»

Программная реализация алгоритма возведения в степень точки на эллиптической кривой в форме пересечения Якоби (с использованием библиотеки libakrypt)

Выполнил:
студент группы СКБ-171
Гаварина Светлана

Москва, 2020

1 Теоретическая часть

1.1 Эллиптические кривые

Эллиптической кривой E над полем K называется кривая, определяемая уравнением:

$$E(K) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

где $\{a_1, a_2, a_3, a_4, a_5, a_6\} \in K$.

Пусть $\mathbb{F}_p$ является простым полем, и для $a,\ b \in \mathbb{F}$ выполняется неравенство $4a^3 + 27b^2 \not\equiv 0(mod\, p)$. Тогда эллиптическая кривая $E(a, b)(\mathbb{F}_p)$, определенная параметрами $a,\ b \in \mathbb{F}$, состоит из набора решений или точек $P = (x, y)$, где $x, y \in \mathbb{F}$ уравнения:

$$E(a, b)(\mathbb{F}_p) : y^2 \equiv x^3 + ax + b(mod\, p),$$

вместе с особой точкой $\mathcal{O}$, именуемой точкой в бесконечности. Уравнение $y^2 \equiv x^3 + ax + b(mod\, p)$ называется определяющим уравнением $E(\mathbb{F}_p)$.

Координаты точки $P = (x, y)$ называются *афинными координатами*.

Кратная точка $Q = [k]P = \underbrace{P + \cdots + P}_{k}$, где $P = (x, y)$ — точка, принадлежащая кривой $E(a, b)(\mathbb{F}_p)$.

Нейтральный элемент (точка) $\mathcal{O}$ — элемент, удовлетворяющий следующим условиям:

1. $\mathcal{O} + \mathcal{O} = \mathcal{O}$
2. $\mathcal{O} + (x, y) = (x, y) + \mathcal{O} = (x, y)$

1.2 Пересечение Якоби

Эллиптическая кривая в форме пересечения Якоби имеет параметр a и координаты Якоби (s, c, d), удовлетворяющие следующему уравнению:

$$\begin{cases} s^2 + c^2 = 1\ (mod\ p) \\ as^2 + d^2 = 1\ (mod\ p) \end{cases}$$

Нейтральная точка имеет вид $\mathcal{O} = (0, 1, 0)$.

Для увеличения производительности вычислений необходимо перейти от афинных координат к проективным.

Изначально дана точка в афинных координатах: $P = (x, y)$.

Уравнение кривой в форме Лежандра имеет вид:

$$E(a,b)(\mathbb{F}_p) : y^2 = x(x+1)(x+j)$$

В нашем случае $j = 3$, то есть уравнение после раскрытия скобок примет вид:

$$E(a,b)(\mathbb{F}_p) : y^2 = x^3 + 4x^2 + 3x$$

Пересечение Якоби является пересечением двух квадрик. Спроектируем координаты следующим образом:

$$(x,y) \rightarrowtail (x_0, x_1, x_2, x_3) = (-2y, x^2 - j, x^2 + 2jx + j, x^2 + 2x + j)$$

Нейтральная точка будет иметь вид: $\mathcal{O} = (0, 1, 0) \rightarrowtail (0, 1, 1, 1)$.

В проективных координатах справедлива следующая система уравнений:

$$\begin{cases} x_0^2 + x_1^2 = x_3^2 \ (mod\ p) \\ k^2 x_0^2 + x_2^2 = x_3^2 \ (mod\ p) \end{cases}$$

Сумма двух точек (a_0, a_1, a_2, a_3) и (b_0, b_1, b_2, b_3) вычисляется по формуле:
$c_0 = a_3 b_1 \cdot a_0 b_2 + a_2 b_0 \cdot a_1 b_3$
$c_1 = a_3 b_1 \cdot a_1 b_3 - a_2 b_0 \cdot a_0 b_2$
$c_2 = a_3 a_2 \cdot b_3 b_2 - k^2 \cdot a_0 a_1 \cdot b_0 b_1$
$c_3 = (a_3 b_1)^2 + (a_2 b_0)^2$

Дублирование точки (a_0, a_1, a_2, a_3) происходит по формуле:
$c_0 = 2 a_1 a_3 \cdot a_2 a_0$
$c_1 = (a_1 a_3)^2 - (a_2 a_3)^2 + (a_1 a_2)^2$
$c_2 = (a_2 a_3)^2 - (a_1 a_3)^2 + (a_1 a_2)^2$
$c_3 = (a_2 a_3)^2 + (a_1 a_3)^2 - (a_1 a_2)^2$

2 Библиотека libakrypt

Код на github: https://github.com/axelkenzo/libakrypt-0.x.

Работа с большими числами (вычетами) в библиотеке организована через с-массивы. Например, для вычетов по модулю 2^{256} определяется тип:
`typedef ak_uint64 ak_mpzn256[ ak_mpzn256_size ];`
, где `ak_uint64` – это псевдоним типа `unsigned long long int`,
`ak_mpzn256_size` — определение константы 4.

То есть вычет по модулю 2^{256} представляется в шестнадцатеричной системе в виде четырех 64-битных блоков.

Используемые в программе функции:

Присвоение вычету значения, записанного строкой шестнадцатеричных символов:
`int ak_mpzn_set_hexstr(ak_uint64 *, const size_t , const char* );`

Преобразование вычета в строку шестнадцатеричных символов:
`const char *ak_mpzn_to_hexstr( ak_uint64 *, const size_t );`

Присвоение значения вычета x вычету z:
`void ak_mpzn_set( ak_uint64 *, ak_uint64 *, const size_t );`

Присвоение вычету беззнакового целого значения:
`void ak_mpzn_set_ui( ak_uint64 *, const size_t, const ak_uint64 );`

Сложение двух вычетов в представлении Монтгомери:
`void ak_mpzn_add_montgomery( ak_uint64*, ak_uint64*, ak_uint64*, ak_uint64*, const size_t );`

Вычитание вычетов:
`ak_uint64 ak_mpzn_sub( ak_uint64 *, ak_uint64 *, ak_uint64 *, const size_t );`

Сравнение двух вычетов: `int ak_mpzn_cmp( ak_uint64 *, ak_uint64 *, const size_t );`

3 Программа

Код на github: https://github.com/Ctrl-Admiral/ISAP-works.

Реализован класс `ProjecticCurve`, описывающий точку в проективных координатах. Координаты x_0, x_1, x_2, x_3 являются private-членами. В конструктор могут передаваться как строки шестнадцатеричных символов, так и значения типа `ak_mpzn256` (передаются через указатель на тип `ak_uint64`).

Для этого класса перегружены оператор вывода « и оператор сравнения ==.

Реализован класс эллиптической кривой, хранящий следующие ее характеристики:

- size — размер параметров кривой;
- p — модуль эллиптической кривой;
- q — порядок подгруппы, порождаемой образующей точкой P;
- P — точка, лежащая на кривой;
- k2 — параметр кривой, представленной в проективных координатах. Используется в функции, которая определяет, лежит ли точка на кривой и в функции сложения точек.

Список реализованных функций:

Сложение двух точек:

```
ProjecticPoint add_points(ProjecticPoint& p1, ProjecticPoint& p2);
```

Удвоения точки:

```
void double_point(ProjecticPoint& p);
```

Проверка, лежит ли точка на кривой:

```
bool point_is_ok(ProjecticPoint& point);
```

Вычисления кратной точки:

```
ProjecticPoint point_pow(ProjecticPoint& p, ak_uint64* k,
const std::size_t& size);
```

Умножение двух вычетов:

```
void mul256(ak_uint64* res, ak_uint64* lhs, ak_uint64* rhs,
ak_uint64* p);
```

Вычитание вычетов по модулю:
```
void mpzn_sub_mod(ak_uint64* res, ak_uint64* lhs, ak_uint64* rhs,
ak_uint64* p, const std::size_t& size);
```

Расширенный алгоритм Евклида для нахождения v из уравнения $ru - pv == 1$:
```
void eucl_for_mpzn256(ak_uint64* res, ak_uint64* num,
ak_uint64* p)
```

Алгоритм перевода координат из проективных координат обратно в афинные:
```
ProjecticPoint to_affine(ProjecticPoint& p);
```

Также были внесены небольшие изменения в код библиотеки libakrypt для совместимости с C++ и из-за установки библиотеки как субмодуля:

1) Во все заголовочные файлы добавлено
```
#ifdef __cplusplus
extern "C"{
#endif
...
#ifdef __cplusplus
} // конец extern "C"
#endif
```
2) В CMakeLists.txt исправлено `CMAKE_SOURCE_DIR` на `CMAKE_CURRENT_LIST_DIR`

3) В libakrypt.h.in закомментирована часть:
```
/*#ifndef __STDC_VERSION__
#define inline
int snprintf(char *str, size_t size, const char *format, ... );
#endif*/
```

Список литературы

[1] Olivier Billet, Marc Joye (2003). "The Jacobi Model of an Elliptic Curve and Side-Channel Analysis".

[2] P.-Y. Liardet and N.P. Smart. "Preventing SPA/DPA in ECC Systems Using the Jacobi Form".

[3] hyperelliptic.org/EFD/g1p/auto-jintersect.html

[4] Лекции по курсу "Программирование алгоритмов защиты информации". Автор А.Ю.Нестеренко

Приложение

Расчеты для проективных координат и коэффициента k^2.

Вычисления проводились при помощи среды Wolfram Mathematica.

In[47]:= x =
53 120 966 775 614 406 564 057 078 211 824 306 529 450 596 665 534 024 993 770 026 409 585 664 726 263;
y =
25 507 288 279 082 307 145 042 908 627 955 212 830 305 255 163 313 245 582 388 962 269 818 176 148 595;

p =
115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 111 864
739;
q =
115 792 089 237 316 195 423 570 985 008 687 907 852 844 625 500 764 779 880 598 257 003 673 390 569
304;
j = 3;

In[11]:= x0 = Mod[-2 y, p];
остаток от деления
x1 = Mod[x^2 - j, p];
остаток от деления
x2 = Mod[x^2 + 2 j * x + j, p];
остаток от деления
x3 = Mod[x^2 + 2 x + j, p];
остаток от деления

Уравнение кривой в форме Вейерштрасса (из sage math)

y^2=x^3+115792089237316195423570985008687907853269984665640564039457584007913111861715*x+34560

In[28]:= roots = Solve[x_^3 +
решить уравнения
115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 111
861 715 * x_ + 34 560 == 0, x_, Modulus → p];
модуль

r1 = r /. roots[[1]]
r2 = r /. roots[[2]]
r3 = r /. roots[[3]]

Out[29]= 12

Out[30]= 48

Out[31]= 115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 111 864 679

Система уравнений:

$x_0^2 + x_1^2 = x_3^2$

$k^2 x_0^2 + x_2^2 = x_3^2$

Найдем k^2 :

In[36]:= Solve[k * x0^2 + x2^2 == x3^2, k, Modulus → p]
решить уравнения модуль

Out[36]= {{k →
115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 111
864 737}}

In[65]:= k =
115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 111 864
737;

Переведем в 16 систему координаты и константы:

In[66]:=
```
BaseForm[x0, 16]
```
запись в системе с осн

```
BaseForm[x1, 16]
```
запись в системе с осн

```
BaseForm[x2, 16]
```
запись в системе с осн

```
BaseForm[x3, 16]
```
запись в системе с осн

```
BaseForm[k, 16]
```
запись в системе с осн

```
BaseForm[q, 16]
```
запись в системе с осн

Out[66]//BaseForm=

$8f36c5dc8eb2baaa1fa5c137222983954b043986054271093fab450319609cbd_{16}$

Out[67]//BaseForm=

$54ac0537298374ed23a4b1ec9e5b97672d17712b0628aab6736be7ae5681e4ec_{16}$

Out[68]//BaseForm=

$15546e40d62036ccf0ff01eb997b05195ec5f0d4e82156489a71b1a3236411d3_{16}$

Out[69]//BaseForm=

$3f8ed2e50db7b58d12c2cc96f210bbf7e8519bb8fc26393c806dd5aa9acd493d_{16}$

Out[70]//BaseForm=

$fffffffffffffffffffffffffffffffffffffffffffffffffffffffffef0c5a1_{16}$

Out[71]//BaseForm=

$fffffffffffffffffffffffffffffffebffece026ebc90c86e63e3dd47025758_{16}$