ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: ПО для топологического анализа данных

(промежуточный, этап 1)

Выполнил:
Студент группы БПМИ225 _____ С.А.Корняков
Подпись И.О.Фамилия

29.01.2025
Дата

Принял:
Руководитель проекта Качан Олег Николаевич
Имя, Отчество, Фамилия

-
Должность, ученое звание

Лаборатория ИИ Сбербанка
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки _____ 2025 _____ _____
Оценка (по 10-ти бальной шкале) Подпись

Москва 2025

Содержание

Аннотация

Цель работы заключается в написании пошагового руководства по криптографии на эллиптических кривых. Реализованы объекты длинной арифметики, полей и эллиптических кривых. Изучены и имплементированы алгоритмы шифрования и дешифрования, электронной цифровой подписи, подсчёт количества точек на эллиптической кривой, быстрого умножения и деления длинных чисел. Протестированы объекты и алгоритмы по скорости, сравнивая с готовыми решениями. Написано руководство, которое параллельно с имплементацией объясняет и рассказывает, что и зачем было реализовано.

Ключевые слова: эллиптические кривые, шифрование и дешифрование, криптография, ECDSA, ECC, длинная арифметика, FFT, C++, конечные поля, оптимизация, Scoof's algorithm

1 Введение

Современная криптография с нынешними вычислительными мощностями требует значительных ухищрений в шифровке сообщений, и шифрование с помощью эллиптических кривых - один из мощнейших инструментов. Но доступных и полных объяснений от начала до конца по шифрованию на них ничтожно мало, поэтому я решил сделать руководство для людей, которые хотят ознакомиться с данным видом криптографии.

Задача проекта заключается в написании руководства по имплементации данного вида криптографии на личном компьютере, сведя количество используемых сторонних библиотек к минимуму, чтобы лучше понять каждый аспект работы шифрования и дешифрования.

В поставленные задачи входило:

Изучение теории

Изучена теория, стоящая за эллиптическими кривыми, виды шифрований и цифровых подписей на эллиптических кривых, быстрое преобразование Фурье, алгоритмы подсчёта точек на эллиптической кривой.

Реализация необходимых объектов

Имплементированы на языке программирования C++ различные объекты для постройки нужного окружения для шифрования:

1. Длинные целые числа и арифметика на них
   (a) Наивная реализация
   (b) Применение алгоритмов из [**?**]
   (c) Использование FFT и других алгоритмов из [**?**] для деления и умножения
2. Поля и арифметика на их элементах
3. Эллиптические кривые, точки на эллиптических кривых в 6 разных видах координат, арифметика на точках [**?**]

Имплементация алгоритмов

Написаны алгоритмы на полученных объектах:

1. Вычисление количества точек на эллиптической кривой [**?**]
2. Алгоритмы шифрования и дешифровки сообщений [**?**]
3. Алгоритм построения и проверки цифровой подписи [**?**]

Оптимизация написанного кода

Рассмотрены оптимизации, которые следуют из

1. Разных видов координат точек на эллиптических кривых[**?**]
2. Специализации имплементации объектов за счёт одного вида эллиптической кривой или поля[**?**]
3. Предположений о входных данных алгоритмов[**?**]

Тестирование библиотеки

Проведены стресс-тесты и тесты на корректность, измерено время работы без оптимизаций, с оптимизациями и на разных видах координат точки на эллиптической кривой, сделать выводы о лучшей компоновке и реализации. Для тестирования использовалось готовое решение по тестированию проектов [**?**].

Изложение результатов

В отчёте изложена теория, необходимая для понимания рассматриваемых алгоритмов, а также архитектура и дизайн написанной библиотеки.

Написание руководства

Написано сопроводительное руководство к реализованным объектам и алгоритмам, которое пошагово объясняет основную теорию для шифрования, зачем и почему создавать объекты и методы, какие способы оптимизации существуют.

2 Требования к проекту

2.1 Функциональные требования

Написанная в руководстве библиотека на C++ предоставляет доступ к объектам, нужным для шифрования, и готовым алгоритмам шифрования, дешифрования, цифровой подписи. Она обладает следующими объектами и методами:

2.1.1 Длинная арифметика

Для того чтобы работать с беззнаковыми числами, которые не помещаются во встроенные целочисленные типы, создан шаблонный класс беззнаковых длинных чисел с фиксированным количеством бит. У него есть все операции, которые можно делать с встроенными беззнаковыми числами и поведение аналогично им же. Есть способы конвертации из строки и обратно для удобной работы с большими числами.

Пример использования: .

```
#include "uint.h"

using ECG::uint_t;

int main(void) {
    uint_t<512> a = "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
    uint_t<512> b = 1;

    uint_t<512> result = a + b;
    std::string str = result.convert_to<std::string>();
    assert(str == "0x100000000000000000000000000000000000000000");

    result = a ^ b;
    std::string str = result.convert_to<std::string>();
    assert(str == "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE");

    result = a * b;
    std::string str = result.convert_to<std::string>();
    assert(str == "0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF");

    result = "0b1";
    size_t value = result.convert_to<size_t>();
    assert(value == 1);
}
```

2.1.2 Конечное Поле

Для работы в конечных полях реализованы классы простого поля и элемента простого поля. Поле конструируется от одного параметра - простого числа в беззнаковом числе длинной арифметики. Он умеет создавать элементы поля по беззнаковому числу, а также может предоставить свой порядок.

Класс элемента простого поля, который представляет элемент простого поля. Его нельзя сконструировать без поля. У него есть все операции, которые можно делать с элементами поля, а также возвращение значения элемента.

Пример использования: .

```
#include "field.h"

using namespace ECG;

int main(void) {
    uint p = "1000000007";
    Field F(p);
    FieldElement a = F(10);
    FieldElement b = F(uint("105"));

    FieldElement result = a - b;
    std::string str = result.value().convert_to<std::string>();
    assert(str == "999999912");

    result = b.inverse();
    std::string str = result.value().convert_to<std::string>();
    assert(str == "209523811");
}
```

2.1.3 Эллиптическая кривая

Для работы с точками на эллиптических кривых реализованы классы эллиптической кривой и точек на ней в разных видах координат. Эллиптическая кривая конструируется от трёх параметров: коэффициентов a и b в нормальной форме Вейерштрасса уравнения эллиптической кривой и простого поля, над которым построена эллиптическая кривая и в котором лежат элементы a и b. Объект данного класса умеет конструировать точки по координате x, автоматически проверяя, лежит ли на данной кривой точка с такой координатой, а также по координатам x и y, аналогично проверяя принадлежность к кривой. Есть метод для подсчёта количества точек на эллиптической кривой, а также метод генерации случайной точки на кривой.

Шаблонный по виду координат класс точки на эллиптической кривой. Его нельзя сконструировать без эллиптической кривой. Всего имплементированы 6 видов координат:

- Нормальные координаты
- Проективные координаты
- Координаты Якоби
- Модифицированные координаты Якоби
- Координаты Якоби-Чудновского
- Упрощённые координаты Якоби-Чудновского

У него есть все операции, которые можно делать с точками на эллиптической кривой, проверка на то, что это точка $\mathcal{O}$, а также операция умножения на беззнаковое число (т.е. сложение с собой столько раз)

Пример использования: .

```
#include "EllipticCurve/elliptic-curve.h"

using namespace ECG;
```

```
int main(void) {
    Field F("1000000007");
    FieldElement a =
    F(uint("0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc"));
    FieldElement b=
    F(uint("0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b"));

    EllipticCurve E(a, b, F);
    EllipticCurvePoint<CoordinatesType::Jacobi> p1 =
    E.point_with_x_equal_to<CoordinatesType::Jacobi>(uint("9124719249999124")).value();
    std::optional<EllipticCurvePoint<CoordinatesType::Jacobi>> p2_ =
    E.point_with_x_equal_to<CoordinatesType::Jacobi>(uint("129641236"));

    EllipticCurvePoint<CoordinatesType::Jacobi> p2 = p2_.value();

    EllipticCuverPoint<CoordinatesType::Jacobi> result =
                              (p1 + p2) * uint("9912371247");
    uint number_of_points = EC.points_number();
}
```

2.1.4 Шифрование

Для того чтобы шифровать сообщения и подписывать их, реализованы следующие классы:

1. Класс шифрования Эль-Гамаля с помощью эллиптический кривых. Конструируется от эллиптической кривой, точки на этой кривой (возможно, её порядка) и функций отображения сообщения на эллиптическую кривую и обратно. Имеет методы: шифрования от сообщения и секретного ключа в пару точек на эллиптической кривой, дешифровки сообщения от пары точек и секретного ключа.

2. Класс электронной цифровой подписи ECDSA. Конструируется от основных параметров для шифрования на эллиптической кривой. Также поддерживает создание от порядка поля и уровня безопасности. Имеет методы: генерации ключевой пары, генерации цифровой подписи от сообщения, проверки цифровой подписи по сообщению и подписи.

2.2 Нефункциональные требования

2.2.1 Описание программы и используемых библиотек

1. Программа написана на языке C++, используется версия языка C++20.

2. Для открытия, сборки и тестирования проекта используется Microsoft Visual Studio 2022.

3. Программа компилируется с помощью MSVC.

4. Используется пакетный менеджер Nuget, интегрированный в Microsoft Visual Studio 2022.

5. Проект использует систему поддержки версий git вместе с github.

6. Нет требований к используемой RAM или HDD.

7. Для форматирования используется clang format.

8. Используется clang-tidy для унификации именований и статической диагностики программы.

2.2.2 Используемые библиотеки

1. Библиотека [?] в качестве примера корректной и быстрой длинной арифметики для тестирования.

2. Библиотека [?] для криптографически-безопасной генерации случайных чисел.

3. Библиотека [**?**] для хеширования с помощью SHA3.

4. В качестве фреймворка для тестирования программы используется интегрированный в Microsoft Visual Studio 2022 Nuget пакет [**?**].

2.2.3 Политика обработки ошибок

1. Обработка std::exception отключена в угоду оптимизации выполнения программы.

2. Неправильный вид чисел в C-style строках при конструировании uint не обрабатывается и является UB.

3. Если пользователь складывает элементы из разных полей или точки с разных эллиптических кривых, то это никак не обрабатывается и является UB.

4. Вставлены asserты в DEBUG версии программы для выявления ошибок имплементации методов и алгоритмов.

5. Используется std::optional для обработки несуществования выходных данных алгоритмов и методов.

Подытоживая, программа предполагает, что использующий её пользователь вводит верные данные и следует ограничениям используемых алгоритмов.

3 Теория и описание алгоритмов

Для ознакомления с основными понятиями см. *A*.

Опишем алгоритмы по порядку:

3.1 Алгоритмы длинной арифметики

Длинная арифметика основана на системе счисления с основанием 2^{32} или 2^{64}. Для сложение и вычитания, и битовых сдвигов алгоритмы тривиальны. Рассмотрим умножение и деление:

3.1.1 Быстрое преобразование Фурье

Для того чтобы реализовать быстрое умножение и деление длинных чисел, которые представлены в виде массива целых чисел, используются алгоритмы FFT и iFFT с некоторыми поправками на то, что перемножаются числа с основанием, а не полиномы. Для начала опишем их:

Главная задача - перемножить два полинома. Пусть $A, B \in P_d[x]$ и $C(x) := A(x) \cdot B(x)$. Сначала сделаем такое наблюдение:

Лемма 1. $\forall A \in P_d[x]$ *однозначно определяется* $d+1$ *точкой, где точка - это пара координат в* $\mathbb{C}$.

Таким образом, любой полином $P(x) = p_d x^d + p_{d-1}x^{d-1} + \cdots + p_1 x + p_0$ представим в двух видах:

1. В коэффициентном: $[p_0, p_1, \ldots, _d]$

2. В виде точек: $(x_0, P(x_0), (x_2, P(x_2), \ldots, (x_{d+1}, P(x_{d+1}))))$, назовём такое представление "в значениях"

Пусть мы представили A в виде $((a_0, b_0), \ldots, (a_d, b_d))$, а B в виде $((a_0, c_0), \ldots, (a_d, b_d))$. Тогда, перемножив эти вектора покомпонентно, получим представление в значениях для $C(x)$ по однозначности представления: $((a_0, b_0 \cdot c_0), \ldots, (a_d, b_d \cdot c_0))$ - перемножили два полинома за линейное время: $O(d)$.

Получается, чтобы умножить два полинома степени d, надо перевести их в представление в значениях, перемножить покомпонентно точки и перевести обратно в коэффициентное представление. Эти переводы между представлениями и являются FFT и iFFT соответственно.

FFT: Заметим такой факт: если $f(x) = x^2$, то чтобы вычислить $f(x)$ в 8 точках, достаточно вычислить $f(x)$ в 4 положительных точках - тогда значения в отрицательных будут такими же. Аналогично, если $f(x) = x^2$, то чтобы вычислить $f(x)$ в 8 точках, достаточно вычислить $f(x)$ в 4 положительных точках - тогда значения в отрицательных будут противоположными.

Учитывая предыдущий факт, разделим наш полином $A(x)$ на мономы чётных степеней и нечётных степеней, например:

$$A(x) = 1 + 3x - 5x^2 + 7x^3 + x^4 - x^5 - 10x^6 = (1 - 5x^2 + x^4 - 10x^6) + (3x + 7x^3 - x^5)$$

$$= (1 - 5x^2 + x^4 - 10x^6) + x(3 + 7x^2 - x^4) =: P_e(x^2) + xP_o(x^2)$$

где $P_e(x) = 1 - 5x + x^2 - 10x^3, P_o(x) = 3 + 7x - x^2$, т.е. их степень в два раза меньше.

Хотим вычислить $A(x) := p_{d-1}x^{d-1} + p_{d-1}x^{d-1} + \cdots + p_1x + p_0$ в точках $\pm x_1, \pm x_2, \ldots, \pm x_{d/2}$. Заметим, что

$$\begin{cases} P(x_i) = P_e(x_i^2) + x_iP_o(x_i^2) \\ P(-x_i) = P_e(x_i^2) - x_iP_o(x_i^2) \end{cases}$$

Значит P_e и P_o надо вычислить только в $x_1^2, \ldots, x_{d/2}^2$. Заметим, что сложность уменьшилась: надо вычислить полиномы меньшего размера в $d/2$ точка, т.е. всего в d точках. Теоретическая сложность рекурсивного вычисления $O(n\log(n))$ против наивного перемножения за $O(n^2)$, но есть проблема - точки не противоположны друг другу, поэтому ту же идею нельзя повторять.

Решение проблемы - использовать $\sqrt[d]{1} \in \mathbb{C}$. Стандартным алгоритм использует $d := 2^n$, т.е. степень двойки, хотя есть вариации, которые обходят это ограничения, но они намного сложнее, поэтому ограничимся степенью 2. Все эти d точек образуют правильный многоугольник с углом поворота между вершинами $\frac{2\pi}{d}$. Тогда первый, отличный от 0, корень из 1 будет равен $w := 1 \cdot e^{i \cdot \frac{2\pi}{d}} = e^{\frac{2\pi i}{d}}$. Тогда искомые корни: $1, w^1, w^2, \ldots, w^{d-1}$, т.к. $w^{j+d/2} = -w^j$.

Заметим, что при возведении в квадрат мы получим $\sqrt[d/2]{1}$, где $\frac{d}{2} = 2^{n-1}$, т.е. аналогичную ситуацию.

Итого: Дан $P(x) : (p_0, p_1, \ldots, p_{n-1})$.

1. Вычислим $w := e^{\frac{2\pi i}{n}}$
2. Теперь нужно вычислить $P(x)$ в $w^0, w^1, \ldots, w^{n-1}$:
3. Если $n = 1$, то возвращаем $(1, P(1))$ иначе:
4. Разделим $P(x)$ на $P_e(x) : (p_0, p_2, \ldots, p_{n-2})$ и $P_o(x) : (p_1, p_3, \ldots, p_{n-1})$
   и вычислим их в точках $(w^0, w^2, \ldots, w^{2n-2} = w^{n-2})$, т.е. переходим к шагу 2 рекурсивно
5. Получили представления P_e и P_o в значениях:

$$y_e = (P_e(w^0), P_e(w^2), \ldots, P_e(w^{n-2})), y_o = (P_o(w^0), P_o(w^2), \ldots, P_o(w^{n-2}))$$

6. Вычислим представление в значениях для исходного полинома $P(x)$:

$$\begin{cases} P(w^j) = P_e(w^{2j}) + w^jP_o(w^{2j}) \\ P(-w^j = w^{j+(n/2)}) = P_e(w^{2j}) - w^jP_o(w^{2j}) \end{cases} , j \in \left\{0, 1, \ldots, \frac{n}{2} - 1\right\}$$

   Т.е. в терминах массивов:

$$\begin{cases} P(w^j) = y_e[j] + w^jy_o[j] \\ P(w^{j+(n/2)}) = y_e[j] - w^jy_o[j] \end{cases} , j \in \left\{0, 1, \ldots, \frac{n}{2} - 1\right\}$$

7. Возвращаем $y = (P(w^0), P(w^1), \ldots, P(w^{n-1}))$

Псевдокод:

Algorithm 1: FFT

Data: $P : [p_0, p_1, \ldots, p_{n-1}], n = 2^d$

Result: $y = [P(x_0), P(x_1), \ldots, P(x_{n-1})]$

$w \leftarrow e^{\frac{2\pi i}{n}}$

if $n == 1$ **then**

  $y \leftarrow P(1)$

  return y

end

$P_e \leftarrow [p_0, p_2, \ldots, p_{n-2}]$

$P_o \leftarrow [p_1, p_3, \ldots, p_{n-1}]$

$y_e \leftarrow \mathrm{FFT}(P_e)$

$y_o \leftarrow \mathrm{FFT}(P_o)$

for $j = 0;\ j < \frac{n}{2};\ ++j$ **do**

  $y[j] \leftarrow y_e[j] + w^j y_o[j]$

  $y[j + n/2] \leftarrow y_e[j] - w^j y_o[j]$

end

return y

iFFT: Представим значения в точках в виде матриц:

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P_{x_{n-1}} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

В FFT мы использовали $x_k = w^k$, где $w := e^{\frac{2\pi i}{n}}$. Тогда матрица имеет вид:

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P_{x_{n-1}} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

Матрица посередине называется матрицей DFT. Тогда для интерполяции (выведения коэффициетов через значения) нужно обратить данную матрицу:

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P_{x_{n-1}} \end{bmatrix}$$

Лемма 2.

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ 1 & w^2 & w^4 & \ldots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)} \end{bmatrix}^{-1} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w^{-1} & w^{-2} & \ldots & w^{-(n-1)} \\ 1 & w^{-2} & w^{-4} & \ldots & w^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{-(n-1)} & w^{-2(n-1)} & \ldots & w^{-(n-1)(n-1)} \end{bmatrix}$$

Таким образом, чтобы интерполировать результаты перемножения двух полиномов, достаточно применить FFT с $w = \frac{1}{n} e^{-\frac{2\pi i}{n}}$.

Псевдокод:

Algorithm 2: iFFT

Data: $P : [p_0, p_1, \ldots, p_{n-1}], n = 2^d$

Result: $y = [P(x_0), P(x_1), \ldots, P(x_{n-1})]$

$w \leftarrow \frac{1}{n} e^{-\frac{2\pi i}{n}}$

if $n == 1$ **then**

  $y \leftarrow P(1)$

  return y

end

$P_e \leftarrow [p_0, p_2, \ldots, p_{n-2}]$

$P_o \leftarrow [p_1, p_3, \ldots, p_{n-1}]$

$y_e \leftarrow \mathrm{FFT}(P_e)$

$y_o \leftarrow \mathrm{FFT}(P_o)$

for $j = 0;\ j < \frac{n}{2};\ ++j$ **do**

  $y[j] \leftarrow y_e[j] + w^j y_o[j]$

  $y[j + n/2] \leftarrow y_e[j] - w^j y_o[j]$

end

return y

Алгоритм умножения двух длинных чисел в имплементации подвергся некоторым изменениям в угоду оптимизации и ускорения вычисления. Объяснения оптимизаций и код к ним можно найти в репозитории проекта.

3.1.2 Деление длинных чисел

Рассмотрим алгоритм D деления длинных чисел, описанный Кнутом в [**?**]:

Задача - поделить два длинных числа, представленных цифрами с основанием b, где b в имплементации 2^{32} или 2^{64}.

Рассмотрим сначала $u = (u_n u_{n-1} \ldots, u_0)_b$ и $v = (v_{n-1} \ldots v_0)_b$, где $u/v < b$. Найдём алгоритм для вычисления $q := \lfloor u/v \rfloor$:

Заметим, что $u/v < b \Leftrightarrow u/b < v \Leftrightarrow \lfloor u/b \rfloor < v$, а это условие того, что

$$(u_n u_{n-1} \ldots, u_1)_b < (v_{n-1} \ldots v_0)_b$$

Если обозначить $r := u - qv$, то q - это уникальное число, такое что $0 \leqslant r < v$. Пусть

$$\hat{q} := \min\left(\left\lfloor \frac{u_n b + u_{n-1}}{v_{n-1}} \right\rfloor, b - 1\right)$$

Т.е. мы получаем гипотетическое значение q, поделив первые две цифры u на первую цифру v, а если результат деления больше или равен b, то берём $b - 1$. Для такого $\hat{q}$ выполняются две теоремы:

Теорема 3. $\hat{q} \geqslant q$

Теорема 4. *Если* $v_{n-1} \geqslant \lfloor b/2 \rfloor$, *то* $\hat{q} - 2 \leqslant q \leqslant \hat{q}$.

Существенно ограничили нашу гипотезу. Умножив u и v на $\lfloor b/(v_{n-1} + 1) \rfloor$, мы не изменим длину числа v и результат деления. После этого умножения станет выполнятся вторая из данных теорем.

Алгоритм D**:** Дано неотрицательное целое число $u = (u_{m+n-1}, \ldots, u_1, u_0)_b$ и $v = (v_{n-1}, \ldots, v_1, v_0)_b$, где $v_{n-1} \neq 0$ и $n > 1$. Мы хотим посчитать $\lfloor u/v \rfloor = (q_m, q_{m-1}, \ldots, q_0)_b$ и остаток $u \bmod v = (r_{n-1}, \ldots, r_0)_b$:

1. $d := \lfloor b/(v_{n-1}+1) \rfloor$. Тогда пусть $(u_{m+n} u_{m+n-1} \ldots u_1 u_0)_b := (u_{m+n-1} \ldots u_1 u_0)_b \cdot d$, аналогично, $(v_{n-1}, \ldots, v_1, v_0)_b = (v_{n-1}, \ldots, v_1, v_0)_b \cdot d$. Заметим, что новая цифра могла появиться только у u
2. Итерироваться будем по j, которая в начале равна m (Делить в следующих шагах будем $(u_{j+n} \ldots u_{j+1} u_j)_b$ на $(v_{n-1} \ldots v_1 v_0)_b$ чтобы получить цифру q_j)
3. $\hat{q} := \left\lfloor \frac{u_{j+n} b + u_{j+n-1}}{v_{n-1}} \right\rfloor$ и пусть $\hat{r}$ будет остатком, т.е. $\hat{r} := u_{j+n} b + u_{j+n-1} \pmod{v_{n-1}}$

4. Если $\hat{q} \geqslant b$ или $\hat{q}v_{n-2} > b\hat{r} + u_{j+n-2}$, то уменьшаем $\hat{q}$ на 1 и увеличиваем $\hat{r}$ на v_{n-1}. Если $\hat{r} < b$, то повторяем данный шаг

5. Заменим $(u_{j+n} \dots u_{j+1}u_j)_b$ на

$$(u_{j+n} \dots u_{j+1}u_j)_b - \hat{q}(0v_{n-1} \dots v_1v_0)_b$$

6. Назначаем $q_j = \hat{q}$

7. Если число u на 5 шаге получилось отрицательным, то добавляем к нему b^{n+1} и переходим к шагу 8, иначе переходим к шагу 9.

8. (Вероятность данного шага крайне мала, за счёт чего достигается асимптотическая быстрота алгоритма) Уменьшаем q_j на 1 и добавляем $(0v_{n-1} \dots v_1v_0)_b$ к $(u_{j+n} \dots u_{j+1}u_j)_b$ (при сложении появится цифра u_{j+n+1}, её следует проигнорировать)

9. Уменьшаем j на 1. Если $j \geqslant 0$, то возвращаемся на шаг 3

10. Теперь $q = (q_m \dots q_1q_0)$ - это искомое частное, а искомый остаток можно получить, поделив $(u_{n-1} \dots u_1u_0)$ на d наивным способом

11. Возвращаем (q, r)

3.2 Алгоритмы поля

Так как мы используем только простые поля, то имплементируем кольцо вычетов по модулю простого числа, т.е. просто остатки от деления на простое число, - оно и будет искомым полем.

Умножение в поле - умножение в длинной арифметике + взятие остатка. Деление - умножение на обратное к второму числу.

Инверсия - расширенный алгоритм Евклида: вычисляем НОД последовательным делением с остатком, где полученные частные и остатки помогают вычислить по формуле ещё две последовательности по рекуррентным формулам, которые в конце дадут нужное число - обратные к исходному числу по модулю простого числа.

Быстрое возведение в степень - стандартное: если степень n нечётная, то вычисляем степень n-1 и умножаем на исходное число, иначе вычисляем степень n / 2 и перемножаем полученное значение с собой же. На выходе получаем число в n степени, на вычисление которого ушло $O(\log_2 n)$.

Если использовать простые числа NIST, то можно сделать некоторые оптимизации, которые недоступны в общем решении. Эти оптимизации рассмотрены в руководстве.

3.3 Алгоритмы Эллиптической кривой

В качестве эллиптической кривой рассматривается группа $G := \{(x, y) \in \mathbb{F}_p | y^2 = x^3 + ax + b\colon 4a^3 + 27b^2 \neq 0\} \cup \{\infty\}$, где $a, b \in \mathbb{F}_p$. Обозначим её $\mathbb{E}(\mathbb{F}_p)$. Точку бесконечности обозначим как $\mathcal{O}$. Опишем основные алгоритмы для нормальных координат, т.к. алгоритмы для остальных координат - это те же самые алгоритмы, только в некоторых моментах нужная информация уже предподсчитана в координатах точки для ускорения вычисления.

3.3.1 Сложение

Чтобы сложить две различные точки, надо провести через них прямую на плоскости и посмотреть, какую точку на эллиптической кривой она пересекла. Тогда симметричная ей точка, относительно оси Ox и будет результатом сложения. Если прямая не пересекла никакую точку на кривой, то считаем, что сумма равна точке ∞. Чтобы сложить точку с собой, надо провести касательную через неё и повторить алгоритм выше. Точка ∞ будет нулём в данной группе.

Заметим, что при таком задании операции '+' обратная точка в группе - это точка с противоположной координатой y. С помощью уравнений выразим операцию сложения в нормальных координатах: $A, B, C \in \mathbb{E}(\mathbb{F}_p), A = (x_1, y_1), B = (x_2, y_2), C = (x_3, y_3)$ и $A + B = C$, рассмотрим 3 случая:

1. $x_1 \neq x_2$. Обозначим $k := \frac{y_1 - y_2}{x_1 - x_2}$. Пусть $C := A + B$ и $C = (x_3, y_3)$. Тогда

$$x_3 = k^2 - x_1 - x_2$$

$$y_3 = k(x_1 - x_3) - y_1$$

2. $x_1 = x_2, y_1 = -y_2$. Тогда $A + B = \infty$.

3. $A = B$. Обозначим $k := \frac{3x_1^2 + a}{2y_1}$. Тогда

$$x_3 = k^2 - 2x_1$$

$$y_3 = k(x_1 - x_3) - y_1$$

3.3.2 Умножение

Умножение на число n - это сложение n раз точки с собой. Есть несколько этапов:

1. Представить число n в *non-adjancent form*(NAF) форме - это уникальное представление числа в двоичном основании, где есть цифра -1. Такое представление помогает достичть минимального веса Хэмминга (количество цифр, отличных от 0), что заметно ускорит умножение точки на число, если удвоение точки будет быстрее сложения. Алгоритм перевода из двоичной системы исчисления в NAF:

Algorithm 3: NAF from binary

Data: $n = (n_{m-1}n_{m-2}\dots n_0)_2$
Result: $z = (z_m z_{m-1}\dots z_1 z_0)_{NAF}$
$i \leftarrow 0$
while $n > 0$ **do**
  if n is odd **then**
    $z_i \leftarrow 2 - (n \mod 4)$ $n \leftarrow n - z_i$
  else
    $z_i \leftarrow 0$
  end
  $n \leftarrow n/2$
  $i \leftarrow i + 1$
end

Есть версия данного алгоритма, которая основана на битовых операциях:

Algorithm 4: NAF from binary by bit operations

Data: $x = (x_{m-1}x_{m-2}\dots n_0)_2$
Result: $z = (z_m z_{m-1}\dots z_1 z_0)_{NAF}$
$xh \leftarrow x >> 1$
$x3 \leftarrow x + xh$
$c \leftarrow xh$ XOR $x3$
$pb \leftarrow x3$ AND c
$nb \leftarrow xh$ AND c
$z \leftarrow pb - nb$

Есть улучшенная версия данного представления:

Определение 5. Пусть $w \geqslant 2$ - положительное целое число. Тогда wNAF представление положительного числа k - это представление $k = \sum_{i=0}^{l-1} k_i 2^i$, где каждый ненулевой коэффициент k_i нечётный, $|k_i| < 2^{w-1}$, $k_{l-1} \neq 0$ и хотя бы одна ненулевая цифра есть в каждой подпоследовательности из w цифр. Длина такого представления равна l.

Теорема 6 (Свойства wNAF)**.** *Пусть k - положительное целое число, тогда*

(a) k имеет уникальное представление wNAF, которое обозначается $NAF_w(k)$

(b) $NAF_2(k) = NAF(k)$, где $NAF(k)$ - это классическое бинарное представление в NAF

(c) Длина $NAF_w(k)$ больше длины $(k)_2$ хотя бы на 1 цифру.

(d) Средняя плотность ненулевых цифр в $wNAF$ представлении длины l примерно $\frac{1}{w+1}$.

Алгоритм вычисления $wNAF$ формы:

Algorithm 5: Computing wNAF

Data: $w, k \in \mathbb{N}$
Result: $NAF_w(k)$
$i \leftarrow 0$
while $k \geqslant 1$ **do**
  if k *is odd* **then**
    $k_i \leftarrow k \pmod{2^w}$ $k \leftarrow k - k_i$
  else
    $k_i \leftarrow 0$
  end
  $k_i \leftarrow k/2$
  $i \leftarrow i + 1$
end
return $(k_{i-1}, \ldots, k_1, k_0)$

2. Теперь напишем алгоритм вычисления кратной точки:

Algorithm 6: wNAF for computing kP

Data: $w, k \in \mathbb{N}, P \in \mathbb{E}(\mathbb{F}_p)$
Result: $Q := kP$
$[k_i] \leftarrow NAF_w(k)$
Compute $P_i := iP$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
$Q \leftarrow \mathcal{O}$
for i *from* $l - 1$ *to* 0 **do**
  $Q \leftarrow 2Q$
  if $k_i \neq 0$ **then**
    if $k_i > 0$ **then**
      $Q \leftarrow Q + P_{k_i}$
    else
      $Q \leftarrow Q - P_{-k_i}$
    end
  end
end
return Q

Также есть алгоритмы для вычисления кратной фиксированной точки. Они основаны на предварительных вычислениях и сохранении некоторой информации (т.е. кешировании данных).

3.3.3 Подсчёт точек на эллиптической кривой

Рассмотрим эллиптическую кривую $\mathbb{E}(\mathbb{F}_p)$, $p > 3$.

Определение 7. Порядком точки $P \in \mathbb{E}(\mathbb{F}_p)$ называется такое наименьшее натуральное число n, что $nP = \mathcal{O}$

Определение 8. Эндоморфизмом Фробениуса Φ_p на эллиптической кривой называется:

$$\Phi_p \colon \mathbb{E}(\overline{\mathbb{F}}_p) \to \mathbb{E}(\overline{\mathbb{F}}_p)$$

$$(x, y) \mapsto (x^p, y^p)$$

где $\overline{F}_p$ - это алгебраическое замыкание простого поля $\mathbb{F}_p$.

Эндоморфизм Фробениуса удовлетворяет следующему характеристическому уравнению:

$$\Phi_p^2 - t\Phi_p + p = 0, \text{ где } t := p + 1 - \#\mathbb{E}(\mathbb{F}_p) \tag{1}$$

Мы знаем, что значение t ограниченно:

Теорема 9 (Хассе)**.**

$$\#\mathbb{E}(\mathbb{F}_p) = p + 1 - t, \text{ где } |t| \leqslant 2\sqrt{p}$$

Из (1) мы получаем, что $\forall P := (x, y) \in \mathbb{E}(\overline{F}_p)$ выполнено:

$$(x^{p^2}, y^{p^2}) + p(x, y) = t(x^p, y^p) \tag{2}$$

где умножение на скаляры p и t означает сложение точки с собой p или t раз соответственно.

Определение 10.

$$\mathbb{E}[l] := \left\{P := (x, y) \in \mathbb{E}(\overline{F}_p) \colon lP = \mathcal{O}\right\}$$

Тогда каждая точка $P \in \mathbb{E}[l]$ называется точкой l-кручения.

Пусть l - произвольное простое число, не равное p. Обозначим $\bar{t} := t \pmod l$ и $\overline{p} := p \pmod l$. Тогда, если $(x, y) \in \mathbb{E}[l]$, то из (2) следует, что

$$(x^{p^2}, y^{p^2}) + \overline{p}(x, y) = \bar{t}(x^p, y^p)$$

Значит, чтобы найти значение t, надо воспользоваться Китайской теоремой об остатках, но чтобы вычислить $t \pmod l$ для простых $l > 2$, нужно сначала определить полиномы деления:

Определение 11. Полиномы деления - это последовательность полиномов $\Psi_m \in \mathbb{Z}[x, y, a, b]$, которая определена рекурсивно:

$$\Psi_0 = 0$$

$$\Psi_1 = 1$$

$$\Psi_2 = 2y$$

$$\Psi_3 = 3x^4 + 6ax^2 + 12bx - a^2$$

$$\Psi_4 = 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)$$

Тогда следующие полиномы определены как:

$$\Psi_{2m+1} = \Psi_{m+2}\Psi_m^3 - \Psi_{m-1}\Psi_{m+1}^3, m \geqslant 2$$

$$\Psi_{2m} = \left(\frac{\Psi_m}{2y}\right) \cdot (\Psi_{m+2}\Psi_{m-1}^2 - \Psi_{m-2}\Psi_{m+1}^2), m \geqslant 3$$

Они обладают следующими свойствами:

Утверждение 12. *Пусть эллиптическая кривая $\mathbb{E}(\mathbb{F})$ задана уравнением $y^2 = x^3+ax+b$. Пусть $\Psi_m \in \mathbb{Z}[x, y]$ - полиномы деления, в которые подставили a и b из уравнения эллиптической кривой. Тогда корни Ψ_{2n+1} - это точки из $\mathbb{E}[2n + 1] \setminus \{\mathcal{O}\}$, а корни $\frac{\Psi_{2n}}{y}$ - это точки из $\mathbb{E}[2n] \setminus \mathbb{E}[2]$*

Утверждение 13. *Пусть $P := (x, y)$ - точка на эллиптической кривой $\mathbb{E}(\overline{F}_p)$, заданной уравнением $y^2 = x^3 + ax + b$. Пусть $n \in \mathbb{Z}$, тогда*

$$nP = \left(x - \frac{\Psi_{n-1}\Psi n + 1}{\Psi_n^2}, \frac{\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2}{4y\Psi_n^3}\right)$$

Эти ключевые утверждения позволяют имплементировать алгоритм Шуфа на практике. Теперь опишем сам алгоритм:

3.3.4 Алгоритм Шуфа

Дано: Эллиптическая кривая, заданная уравнением $y^2 = x^3 + ax + b$ над простым полем $\mathbb{F}_p$.
Результат: Количество точек, т.е. $\#\mathbb{E}(\mathbb{F}_p)$

1. Создадим множество относительно маленьких простых чисел, которые не равны p:

$$S := \{l_1, l_2, \ldots, l_L\} = \{2, 3, 5, 7, 11, \ldots, l_L\}$$

таких, что

$$\prod_{i=1}^{L} l_i > \lceil 4\sqrt{p} \rceil$$

2. Вычислим $t \pmod l$ для каждого l из множества:
   (a) Для $l = 2$: Если $gcd(x^3 + ax + b, x^p - x) \neq 1$, то $t \equiv 0 \pmod 2$, иначе $t \equiv 1 \pmod 2$.
   (b) Для $l \neq 2$: Пусть $p_l := p \pmod l$. Вычислим x-координату точки

$$(x', y') := (x^{p^2}, y^{p^2}) + p_l(x, y) \bmod \Psi_l$$

   (c) Для каждого $j = 1, 2, \ldots, \frac{l-1}{2}$ сделаем следующее:
      i. Вычислим x-координату точки $(x_j, y_j) := j(x, y)$.
      ii. Если $x' - x^p - j \equiv 0 \bmod \Psi_l$, то переходим к следующему шагу. Иначе, пробуем следующее значение j.
      iii. Вычисляем y' и y_j. Если $\frac{y' - y_j^p}{y} \equiv 0 \bmod \Psi_l$, то $t \equiv j \pmod l$, иначе $t \equiv -j \pmod l$.
   (d) Если ни одно из значений не подошло, определим $w^2 := p_l$. Если не существует w, то $t \equiv 0 \pmod l$.
   (e) Если $gcd(numerator(x^p - x_w), \Psi_l) = 1$, то $t \equiv 0 \pmod l$. Иначе, вычислим:

$$gcd(numerator(\frac{y^p - y_w}{y}), \Psi_l)$$

   (f) Если вычисленный $gcd \neq 1$, то $t \equiv 2w \pmod l$, иначе $t \equiv -2w \pmod l$.

3. Восстанавливаем значение t по Китайской теореме об остатках: пусть $T \equiv t \pmod N$, где $N := \prod_{i=1}^{L} l_i$.

4. Если T находится в границах теоремы Хассе, то $t = T$, иначе $t \equiv -T \pmod N$.

5. Тогда $\#\mathbb{E}(\mathbb{F}_p) = p + 1 - t$

Существует улучшенная версия алгоритма Шуфа, а именно алгоритм SEA, но оптимизации, представленные в нём достаточно сложно реализовать. Общая сложность алгоритма Шуфа - $O(\log^8 p)$, алгоритма SEA - $O(\log^6 p)$.

3.4 Алгоритмы Шифрования и цифровой подписи

3.4.1 Алгоритм Эль-Гамаля

Общие данные: простое число p, эллиптическая кривая $\mathbb{E}(\mathbb{F}_p)$, точка на этой кривой $P \in \mathbb{E}(\mathbb{F}_p)$, её порядок q, отображение сообщения на эллиптическую кривую и обратное слева к нему отображение.

Допустим Боб хочет отправить сообщение m Алисе. Тогда:

1. Алиса генерирует секретный ключ n_α: $1 \leqslant n_\alpha < q$, публичный ключ: $A := n_\alpha P$. Отправляет публичный ключ Бобу.

2. Боб обратимо отображает сообщение m на эллиптическую кривую: $M \in \mathbb{E}(\mathbb{F}_p)$, генерирует одноразовый сектретный ключ k: $1 \leqslant k < q$, вычисляет $C_1 := kP, C_2 := kA + M$ и отправляет пару (C_1, C_2) Алисе.

3. Алиса вычисляет исходную точку по формуле:

$$C_2 - n_\alpha C_1 =$$

$$= kA + M - n_\alpha kP = n_\alpha kP + M - n_\alpha kP = M$$

и отображает её обратно в m

Отображать сообщение, представленное в виде двоичного числа, на эллиптическую кривую можно следующим образом:

1. Пусть наше поле $\mathbb{F}_p$, и p - это достаточно большое число. Обозначим за q - длину числа p в битовом представлении.

2. Выберем и зафиксируем число $l \in (0, \frac{q}{2})$, и будем в первые l бит числа записывать наше сообщение m в двоичном представлении.

3. Заполним оставшиеся биты случайно. Тогда высока вероятность, что полученное число - это координата x какой-то точки на эллиптической кривой. Если нет, то повторяем этот шаг, пока не получим точку на кривой.

4. Когда Алиса посчитала точку M, она берёт координату x этой точки. Тогда первые l бит значения этой координаты и будут сообщением m.

Заметим, что отображать сообщения на кривую и обратно не является удобным. В вариации Эль-Гамаля с хешированием, выбирается хеш-функция $H\colon \mathbb{E}(\mathbb{F}_p) \to \{0,1\}^n$ и $(C_1, c_2) := (kP, m \oplus H(kA))$, т.е. сообщение в двоичном виде длины $\leqslant n$ XORится с хешем от точки kA. Тогда Алиса получает исходное сообщение через $m = c_2 \oplus H(n_\alpha C_1)$.

3.4.2 ECDSA

У этого алгоритма есть несколько основных подалгоритмов:

1. Алгоритм генерации основных параметров:

   Пользователь выбирает простое число p и уровень безопасности $L\colon 160 \leqslant L \leqslant [\log_2 p]$ и $2^L \geqslant 4\sqrt{p}$. На выходе получаем основные параметры эллиптической кривой.

   (a) Выбираем верифицировано случайным образом $a, b \in \mathbb{F}_p\colon 4a^3 + 27b^2 \neq 0$, чтобы они были параметрами эллиптической кривой. Назовём её $\mathbb{E}(\mathbb{F}_p)$

   (b) Находим $N := \#\mathbb{E}(\mathbb{F}_p)$

   (c) Проверяем, что существует простое число $n \geqslant 2^L\colon N \equiv 0 \pmod n$, т.е. что у N в делителях есть большое простое число. Если это условие неверно, то переходим на первый шаг

   (d) Проверяем, что для этого простого числа $p^k - 1 \not\equiv 0 \pmod n$ $\forall k \in \{1, 2, \ldots, 20\}$

   (e) Проверим, что $p \neq n$, иначе переходим на шаг 1

   (f) Пусть $h := \frac{N}{n}$

   (g) Генерируем случайную точку $G' \in \mathbb{E}(\mathbb{F}_p)$ и задаём $G := hG'$. Если $G = \mathcal{O}$, то повторяем данный шаг.

   (h) Возвращаем $D := (p, a, b, F, E, G, n, h)$

2. Алгоритм генерации ключей:

   Пользователь передаёт основные параметры D. На выходе получаем открытый и закрытый ключи.

   (a) Выбираем случайное число $d \in \{1, \ldots, n-1\}$

   (b) Вычисляем $Q := dG$

   (c) Возвращаем (Q, d), где точка на эллиптической кривой Q - открытый ключ, а $d \in \mathbb{N}$ - закрытый ключ

3. Алгоритм генерации цифровой подписи:

   Пользователь, который имеет основные параметры D и ключи (Q, d), хочет подписать сообщение m. Пусть H - криптографическая хеш-функция, результат которой даёт число, битовое представление которого имеет длину не более n. На выходе получаем подпись (r, s):

   (a) Выбираем случайное число $k \in \{1, \ldots, n-1\}$

   (b) Вычисляем точку $(x_1, y_1) = kG$

   (c) Вычисляем $r := x_1 \pmod{n}$. Если $r = 0$, то переходим к шагу 1

   (d) Вычисляем $e := H(m)$

   (e) Вычисляем $s := k^{-1}(e + dr) \pmod{n}$. Если $s = 0$, то переходим к шагу 1

   (f) Возвращаем (r, s)

4. Алгоритм проверки цифровой подписи:

   Другой пользователь получает основные параметры D, хеш-функцию H, сообщение m, подпись (r, s) и открытый ключ Q от первого пользователя и хочет проверить подпись. На выходе получаем решение о принятии или отклонении подписи:

   (a) Если r, s - это не целые числа, принадлежащие $[1, n-1]$, то отклоняем

   (b) Вычисляем $e := H(m)$

   (c) Вычисляем $w := s^{-1} = k(e + dr)^{-1} \pmod{n}$

   (d) Вычисляем $u_1 := ew \pmod{n}$ и $u_2 := rw \pmod{n}$

   (e) Вычисляем координаты точки $X = (x_1, y_1) := u_1 G + u_2 Q$

   (f) Если $X = \mathcal{O}$, то отклоняем

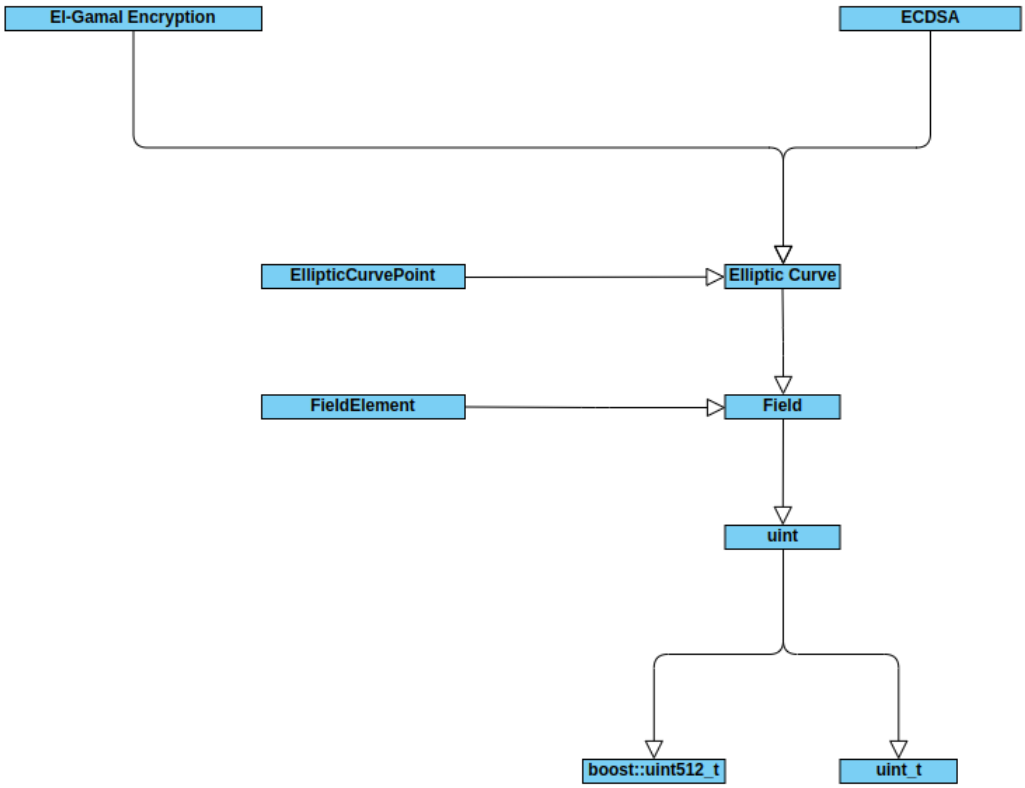
   (g) Вычисляем $v := x_1 \pmod{n}$

   (h) Если $v = r$, то принимаем, иначе отклоняем

4 Архитектура проекта

4.1 Доступные классы

Верхними классами являются классы алгоритмов шифрования и подписи, которые строятся от эллиптической кривой. Эллиптическая кривая строится на простом поле, а простое поле зависит класса длинных чисел. Цепочка зависимостей:



Какие роли они выполняют - очевидно из названия.

4.2 Расположение основных алгоритмов

Перечислим, где используются алгоритмы, и их сложность:

1. Класс uint_t:
   - Конструирование от строки длины m за $O(m)$
   - FFT - умножение длинных чисел за $O(n \log(n))$
   - Algorithm D - деление длинных чисел за $O(n^2)$
   - Конвертация к std::string длины m за $O(m)$
   - Все остальные методы беззнакового целого типа за $O(n)$
2. Класс FieldElement:
   - Расширенный алгоритм Евклида для поиска обратного в поле за $O(n)$
   - Быстрое возведение в степень q за $O(\log(q))$
3. Класс EllipticCurve:
   - Scoof's algorithm - Вычисление количества точек за $O(\log^8(n))$
   - Нахождение y координаты по x за $O(\log(p))$ при $p \equiv 3 \pmod 4$, и $O(\log^3(p))$ иначе.
   - Генерация случайной точки: генерация случайного uint и проверка наличия подходящей y координаты
4. Класс EllipticCurvePoint:
   - Сложение двух точек за $O(D + A)$, где D и A - деление и сложение в поле.

Классы шифрования естественным образом реализуют шифрование.

4.3 Дизайн решения

Создан алиас uint для удобного тестирования при разных реализациях длинной арифметики.

4.4 Взаимодействие

Взаимодействие между классами заключается в использовании одними классами объектов и открытых методов другого, без закрытого общения между классами. Исключениями являются классы FieldElement и EllipticCurvePoint, у которых есть естественный доступ к Field и EllipticCurve соответственно.

4.5 Тестирование

Все основные классы покрыты тестами на корректность и стресс-тестами. В руководство добавлены диаграммы и таблицы результатов тестов на время и сравнение полученных скоростей с другими библиотеками, реализующими данный функционал. Протестирована разная компоновка и точечные оптимизации при некоторых предположениях о входных данных.

A Глоссарий

Термины и обозначения:

- p - простое число
- q - положительное число, которое равно p^n, где p - некоторое простое число, n - некоторое натуральное число
- $(k)_2$ - беззнаковое представление целого числа k в двоичной системе счисления.
- $\mathbb{N}$ - натуральные числа, начиная с 1
- $\mathbb{Z}$ - целые числа
- $\mathbb{Z}_n$ - кольцо вычетов по модулю n, использующее наименьшие неотрицательные вычеты
- $\mathbb{Q}$ - рациональные числа
- $\mathbb{R}$ - поле вещественных чисел
- $\mathbb{C}$ - поле комплексных чисел
- $\mathbb{F}$ - произвольное поле
- $\mathbb{F}_p$ - поле простого порядка p, которое отождествляется $\mathbb{Z}_p$
- $\mathbb{F}_q$ - поле порядка $q = p^n$
- $\mathbb{E}(\mathbb{F}_q)$ - эллиптическая кривая над полем $\mathbb{F}_q$
- НОД или gcd - наибольший общий делитель
- $f(x)$ - функция от x, $\mathbb{R} \to \mathbb{R}$ или $\mathbb{C} \to \mathbb{C}$
- $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ - полином степени nот x с коэффициентами $a_i \in \mathbb{C}$ или $\mathbb{R}$
- $P[x]$ - пространство полиномов от x, где $x \in \mathbb{C}$ или $\mathbb{R}$.
- $R[a, b, c]$ - пространство полиномов от переменных a,b,c, где коэффициенты лежат в кольце R
- $P_n[x]$ - пространство полиномов степени n от x
- Сложность алгоритма $O(f(n))$ - время работы алгоритма с увеличением параметра n, характеризующего количество входной информации алгоритма, будет возрастать не быстрее, чем $f(n)$, умноженная на некоторую константу
- $a \equiv b \pmod{m}$ - существует $k \in \mathbb{Z}$: $a - b = m \cdot k$.
- $\overline{\mathbb{F}_p}$ - алгебраическое замыкание простого поля $\mathbb{F}_p$.
- $\#\mathbb{E}(\mathbb{F}_p)$ - количество точек на эллиптической кривой.
- $numerator\left(\frac{P(x,y)}{Q(x,y)}\right) := P(x, y)$ - взятие числителя рациональной функции.

Список литературы