Руководство по реализации криптографии на эллиптических кривых

Санан Корняков

2024

Содержание

Аннотация

Работа является пошаговым руководством по реализации 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. Руководство 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, Schoof'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.

1.1 Условия игры

Если вы искали данное руководство, то скорее всего где-то слышали/читали об эллиптических кривых и о возможности криптографии на них, поэтому я рассчитываю на базовое понимание математики и алгоритмов.

Здесь не будет дотошного доказательства теорем или строгости в описании математических объектов — в первую очередь акцент делается именно на имплементации (на языке C++). Данный язык был выбран в качестве общеизвестного языка среди программистов. Выберем C++20 для удобного использования шаблонов.

В данном руководстве мы будем стараться использовать как можно меньше готовых библиотек, чтобы не было огромных black box-ов в нашем коде. Это улучшит понимание и возможности алгоритмов.

1.2 База

Знаменитая формула

$$y^2 = x^3 + ax + b$$

обычно является самой первой, которую вы увидите при описании криптографии эллиптических кривых. Появляется несколько вопросов:

- Что такое x, y, a, b? Где лежат данные числа?

  Данные числа являются элементами некоего поля $\mathbb{F}$, над которым построенна эллиптическая кривая, характеристики больше 3 (забьём на последние слова, так как мы будем работать с полями достаточно больших характеристик). Поле поддерживает все стандартные математические операции: сложение, вычитание, деление на ненулевой элемент, умножение, поэтому можно пока считать его $\mathbb{R}$.

- Что такое эллиптическая кривая?

  Это группа точек в $\mathbb{F}^2$, координаты которых удовлетворяют данному уравнению, и ещё точка бесконечности $\mathcal{O}$, которая является своеобразным нулём группы. Сложение в группе происходит по специальным формулам на координаты, которые будут рассмотрены позже. Умножение точки на натуральное число приравнивается к сложению точки с собой это число раз.

- Как это используют для шифрования?

  Обычно выбирается эллиптическая кривая $\mathbb{E}$ над неким полем $\mathbb{F}$, точка P на ней и производится умножение точки на натуральное число k. Криптографическая стойкость достигается сложностью нахождения числа k по точкам P и kP.

- Чем это лучше других методов шифрования?

  Тем, что данный способ шифрования можно реализовать так, что он будет выполнятся быстрее других алгоритмов при аналогичной задаче и данных. Также, для одинаковых показателей криптографической стойкости, криптография на эллиптических кривых требует ключей (чисел для шифрования) меньшей длины, чем другие алгоритмы.

1.3 Постановка задачи

Начитавшись статей на хабре, мы воодушевились и решили написать свою криптографию на эллиптических кривых. Сначала надо определить, какие объекты нам надо реализовать:

- Нам надо реализовать эллиптическую кривую. Но эллиптическая кривая никто без поля, значит нам надо реализовать поле.
- Так как поля бывают бесконечными, а мы работаем на компьютере с числами, то ограничимся на простые поля $\mathbb{F}_p$, которые представим в виде вычетов по простому модулю p. Но этот простой модуль и числа в поле надо представить в виде целых чисел, а в криптографии обычно используются числа из более чем 200 битов. Целые числа такого размера не поддерживаются языком C++, поэтому нам надо реализовать класс целых чисел и длинную арифметику на них.

Итого 3 объекта: целые числа, поле, эллиптическая кривая. Приступим наконец к реализации!

2 Длинная арифметика

Так как мы хотим реализовать вычеты по большому модулю, то достаточно реализовать беззнаковые длинные целые числа.

Основной приём для имплементации длинной арифметики - хранение чисел в основании 2^{32} или 2^{64}. То есть просто массив из целых чисел, которые представляют цифры данного числа в соответствующих основаниях. Есть несколько видов данного представления:

1. Количество цифр в числе меняется в зависимости от размера числа. Нет математического ограничения длины числа, только аппаратное.
2. Количество цифр в числе фиксировано и не меняется от числа.

Последний вариант можно видеть, например, в типе uint64_t — присутствуют сразу все 64 бита не зависимо от содержащихся данных. Данный вид целых чисел является наиболее удобным в реализации и использовании по назначению, поэтому будем имплементировать его.

Так как количество бит в числе может разительно отличатся от задачи к задаче, то общим решением будет создать шаблонный класс по количеству содержащихся в нём бит:

```
template<size_t c_bits>
class uint_t {
};
```

Теперь надо определиться с представлением цифр в нашем классе. Из-за того, что для алгоритма деления, который будет позже, потребуется деление по две цифры, то возьмём за цифру uint32_t, чтобы можно было спокойно делить в uin64_t.

С помощью constexpr определим размеры и длину необходимого массива. Так как длина не изменяется во время жизни объекта, то возьмём std::array за контейнер. Он будет гарантировать, что длина массива сохраняет свой инвариант. Итого получилось:

```
template<size_t c_bits>
class uint_t {
    using block_t = uint32_t;
    using double_block_t = uint64_t;

    static constexpr size_t c_bits_in_byte = 8;
    static constexpr size_t c_block_size = sizeof(block_t) * c_bits_in_byte;
    static constexpr size_t c_block_number = c_bits / c_block_size;
    static constexpr size_t c_double_block_size = sizeof(double_block_t) * c_bits_in_byte;
    static constexpr size_t c_double_block_number = c_bits / c_double_block_size;

    template<size_t V>
    friend class uint_t;

    using blocks = std::array<block_t, c_block_number>;
    blocks m_blocks = {};
};
```

Дали псевдонимы используемым типам, чтобы улучшить читаемость и не менять все типы, если вдруг захотим использовать за цифру uint16_t и какой-то другой тип. 12-13 строчкой мы подружили все шаблоны друг с другом для общего взаимодействия.