МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Ярославский государственных университет им. П.Г.Демидова»

Кафедра алгебры и математической логики

Сдано на кафедру

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Заведующий кафедрой

д.ф.-м.н., доцент

Тимофеева Н.В.

Курсовая работа

**Цифровая подпись на кривых в скрученной форме Эдвардса по ГОСТ 34.10-2018**

Научный руководитель

д.ф.-м.н., доцент

\_\_\_\_\_\_\_\_ Н.В.Тимофеева

«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Студент группы КБ-51

Т. А. Ларина

«\_\_»\_\_\_\_ \_\_\_\_\_\_\_2025 г.

Ярославль, 2025

**Оглавление**

[Введение 3](#_Toc198399098)

[1. Взаимосвязь кривых в канонической форме Вейерштрасса с кривыми в скрученной форме Эдвардса 3](#_Toc198399099)

[2. Структура абелевой группы на кривой Эдвардса 6](#_Toc198399100)

[2.1 Аналитическое выражение для сложения точек и проверка замкнутости сложения 6](#_Toc198399101)

[2.2 Особые точки 7](#_Toc198399102)

[2.3 Абелева группа по сложению 9](#_Toc198399103)

[3. Общие параметры пригодных для криптографии кривых Эдвардса 10](#_Toc198399104)

[4. Параметры пригодных кривых по ГОСТ 34.10-2018 16](#_Toc198399105)

[5. Описание алгоритма цифровой подписи по ГОСТ 34.10-2018 18](#_Toc198399106)

[5.1 Параметры алгоритма 18](#_Toc198399107)

[5.2 Алгоритмы цифровой подписи 19](#_Toc198399108)

[Как это работает? 20](#_Toc198399109)

[6. Проблемы и особенности реализации 21](#_Toc198399110)

[Приложение А (некоторые функции программной реализации схемы цифровой подписи ГОСТ 34.10 - 2018) 25](#_Toc198399111)

[Список используемой литературы: 34](#_Toc198399112)

# Введение

С увеличением электронного документооборота возрастает необходимость замены «ручной» подписи на цифровую. Цифровая подпись позволяет подтвердить и закрепить авторство, проверить целостность сообщения. Российским стандартом цифровой подписи является ГОСТ 34.10-2018, в котором основой стойкости является задача дискретного логарифмирования на эллиптических кривых с дальнейшим распространением на кривые в скрученной форме Эдвардса. В рамках этой работы рассматривается связь кривых в канонической форме Вейерштрасса с кривыми в скрученной форме Эдвардса, закон сложения на кривых в скрученной форме Эдвардса, схема цифровой подписи по ГОСТ 34.10-2018, параметры кривых в скрученной форме Эдвардса, пригодных для криптографии, а также ряд теорем, направленных на ускорение генерации параметров подписи. По результатам математических умозаключений в рамках этой работы реализована схема цифровой подписи ГОСТ 34.10-2018 с хеш-функцией «Стрибог» по ГОСТ 34.11-2028.

# Взаимосвязь кривых в канонической форме Вейерштрасса с кривыми в скрученной форме Эдвардса

Кривая в скрученной форме Эдвардса над полем характеристики описывается уравнением вида [2]:

, (1)

где

Для того, чтобы из канонической формы Вейерштрасса перейти к форме Эдвардса, сначала нужно перейти к форме Монтгомери. Это возможно в рамках ограничений на характеристику поля.

Кривые в канонической форме Вейерштрасса задаются уравнением [1]:

Для удобства перехода к кривой Эдвардса сначала перейдем к форме Монтгомери

Такой переход накладывает определенные ограничения на кривую в форме Вейерштрасса. Во-первых, у правой части уравнения должен быть корень. Пусть это . Тогда мы можем переписать уравнение в виде:

Отсюда легко выразить . Далее заменой получим

То есть:

Чтобы перед коэффициент сохранился равным единице, и коэффициент перед стал равным единице, нам нужно подкорректировать замену. То есть вместо будем делать замену . Такая замена накладывает ещё одно ограничение: должно быть квадратичным вычетом в .

Для удобства введем обозначение . Тогда, выполнив замену в (4), получим:

то есть или

Тогда формула перехода из канонической формы Вейерштрасса в форму Монтгомери следующая:

Поделим (6) на . Тогда в обозначениях (7) получим

Сделаем ещё одну замену . Заметим, что так как (то есть из корень извлекается), то корень из также существует в . Тогда уравнение примет удобный вид

Мы привели кривую Монтгомери к удобному виду. Теперь можно делать преобразование к кривой Эдвардса по следующий формулам:

и в обратную сторону

Итак, чтобы перейти от канонической формы Вейерштрасса к кривой Эдвардса достаточно выполнить замену:

Эти формулы задают бирациональный изоморфизм эллиптической кривой и кривой в скрученной форме Эдвардса.  
  
Стоит заметить, что полученное преобразование переводит кривую в кривую, но не гарантирует сохранение точек даже на этапе перехода от кривой в канонической форме Вейерштрасса к форме Монтгомери.

Например, пусть дана кривая в канонической форме Вейерштрасса

Этой кривой принадлежат точки:

В качестве возьмём . Тогда . Тогда согласно формулам, коэффициенты в форме Монтгомери будут следующие: , а преобразование по , остаётся неизменным.

Тогда ей соответствует вот такая кривая в форме Монтгомери:

Согласно преобразованию, мы должны получить следующие точки на кривой (они записаны в том же порядке, что и соответствующие им точки на кривой Вейерштрасса):

Но если выписать все точки, которые в действительности принадлежат этой кривой, мы получим другой список:

А раз уже на этом этапе мы теряем большинство точек, то при переходе от формы Монтгомери к скрученной форме Эдвардса различия будут. А значит, мы не можем гарантировать, что при сложении точек на кривой Вейерштрасса обе соответствующие точки будут принадлежать кривой Эдвардса. А значит, в общем случае сложение точек на кривой Вейерштрасса не связано со сложением точек на кривой в скрученной форме Эдвардса.

# 2. Структура абелевой группы на кривой Эдвардса

## 2.1 Аналитическое выражение для сложения точек и проверка замкнутости сложения

Закон сложения на кривых в скрученной форме Эдвардса задается следующими формулами

Проверим замкнутость операции

Приведем к общему знаменателю. Посчитаем числитель.

Точки тоже принадлежат этой эллиптической кривой. То есть

Умножим второе уравнение на и вычтем из первого:

Аналогично,

Тогда

Посчитаем знаменатель:

Тогда , то есть сложение действительно замкнуто относительно кривой в скрученной форме Эдвардса.

Также заметим, что точка является нейтральным элементом относительно сложения точек.

## 2.2 Особые точки

Исследуем кривую Эдвардса на особые точки.

Найдём частные производные для данного уравнения. Для удобства, все перенесём в левую часть

Итого система распадается на 4 подсистемы

или или или

Первая подсистема не имеет решений, так как полученная точка не принадлежит кривой.

Условия из второй и третьей системы противоречивы, поэтому они тоже не имеют решений.

Если мы подставим условия последней системы в уравнение кривой, то получим следующее:

Но получившееся условие тоже противоречит определению кривой.

А значит, на кривых в скрученной форме Эдвардса особых точек в привычном смысле нет.

На кривых в скрученной форме Эдвардса особыми также называют точки, где в хотя бы одной из координат возникает деление на 0. Найдём их. Выразим координату через и наоборот

*,*

Отсюда видно, что особые точки имеют координаты

Для этих особых точек предусмотрена формальная арифметика. Так как в наших обозначениях и , появление бесконечной координаты равнозначно умножению числителей и знаменателей на и . Например, удвоить точку можно так:

Таким образом можно убедиться, что имеет порядок 2, а имеет порядок 4, при этом

Также важно заметить, что эти точки можно получить из неособых точек. Например,

## 2.3 Абелева группа по сложению

По аналитическим формулам мы можем предположить, что относительно такого закона сложения на кривой Эдвардса имеется структура абелевой группы.

Чтобы в этом убедиться, мы должны проверить следующие условия:

1. Замкнутость доказана выше
2. Коммутативность такого сложения вытекает из аналитических формул.
3. Нейтральный элемент равен (0,1)

По коммутативности

1. Противоположным к является

так как точка принадлежит кривой. То есть

1. Доказательство ассоциативности можно найти в [2]:
2. Особые точки и входят в абелеву группу по сложению, и имеют порядки 2 и 4 соответственно

# 3. Общие параметры пригодных для криптографии кривых Эдвардса

Пусть – это точка на кривой. – порядок кривой. Пусть порядок точки равен . Тогда кофактором точки будем называть число .

Минимальным кофактором кривой будем называть минимальный из всех кофакторов точек этой кривой.

Для криптографических приложений среди кривых в скрученной форме Эдвардса следует искать кривые порядка = 4𝑛 с минимальным кофактором 4 при нечетном *n*, из которых отбираются кривые с простым *n*. Выясним, какие из скрученных кривых Эдвардса удовлетворяют этому условию.

Для начала найдём точки 4го порядка. Закон сложения имеет вид

Отсюда

Тогда получим систему:

Решим первое уравнение системы

Тогда из уравнения кривой получаем ограничение на :

Тогда получим две точки и . Первая точка – это нейтральный элемент, а вторая имеет порядок 2. Так как мы ищем точки 4 порядка, дальнейшие проверки для этих двух точек бессмысленны.

Тогда из уравнения кривой получаем ограничение на :

Тогда получим две точки и (, 0).

Подставим их в (8):

Условие выполнено. Аналогично проверяем (10).

Подставим их в (9):

Все условия выполнены, а значит, эти точки являются решением системы. Проверим, не являются ли эти точки точками второго порядка.

Мы убедились, что полученные точки, действительно являются точками 4го порядка.

1. .

Из уравнения кривой получаем, что

Тогда

откуда

Решениями этого уравнения являются

Тогда мы получаем 4 точки, удовлетворяющие следующим условиям:

и ещё 4 точки, удовлетворяющие условиям:

Непосредственной проверкой можно убедиться, что они не удовлетворяют (9), а значит, не являются решениями системы.

Используя подстановку (11) и умножение на -1, получим:

откуда

Это уравнение имеет 2 пары решений:

Непосредственной проверкой можно убедиться, что вторая пара решений не удовлетворяет (9), а значит, не является решением системы.

Проверим первую пару решений. Добавим ограничение, полученное из уравнения кривой. Тогда первая пара решений формирует восемь точек, удовлетворяющих следующим условиям:

Подставим в (9):

Легко заметить, что левая и правая часть равны.

Подставим в (9)

Выражение полностью идентично выражению под буквой а) и тоже верно.

Осталось проверить условия (8) и (10).

Так как , то условия (8) и (10) тоже выполнены. А значит, мы получили ещё 8 решений.

Итого есть 10 точек 4го порядка. В зависимости от поля и параметров это число может оказаться меньше.

**Теорема 1.** Неособые точки 4-го порядка при кривой в форме (1) задаются условиями

и существуют тогда и только тогда, когда выполняются условия:

**Доказательство**.

Формулы для точек 4-го порядка мы получили выше.

Осталось доказать условия и .

Заметим, что связаны соотношением (\*)



Пусть α – примитивный элемент мультипликативной группы . Тогда имеем

То есть, если элемент имеет квадратный корень, то он имеет и корень четвертой степени.

При таких условиях является квадратичным невычетом, так как

Тогда соотношение (\*) будет разрешимо тогда и только тогда, когда будет квадратичным вычетом, то есть . Корни вычислимы тогда и только тогда, когда , то есть



Пусть α – примитивный элемент мультипликативной группы . Тогда имеем

При таких условиях является квадратичным вычетом, так как

Тогда соотношение (\*) выполнимо тогда и только тогда, когда

Корень 4 степень извлекается тогда и только тогда, когда

■

**Теорема 2:**

Если кривая в скрученной форме Эдвардса не удовлетворяет условиям Теоремы 1, и параметры и не являются квадратичными вычетами по модулю одновременно, то её порядок равен ( – нечетное).

**Доказательство**.

Кривые, которые не удовлетворяют Теореме 1, не содержат точек четвёртого порядка, удовлетворяющих любой из систем уравнений:

Любая кривая в скрученной форме Эдварса содержит точки . Выше мы выяснили, что дополнительно есть точки – второго порядка, других точек второго порядка нет. А также существуют точки – 4-го порядка, других точек 4-го порядка нет.

1. Если и не являются вычетами, то помимо этих точек, существуют точки , при этом не существует точек 4го порядка и . Выше мы перебрали все точки 4 порядка, а значит, на этой кривой точек 4го порядка не будет совсем. Точки образуют группу 4го порядка. А значит, делится на 4. Так как других точек чётного порядка нет, – нечётное.
2. Если – квадратичный вычет, а – невычет, то точек не существует, но существуют точки 4-го порядка. Сложение точек между собой, а также сложение с точками не создаёт новых точек, а значит, точки образуют группу 4го порядка. А значит, делится на 4. Других точек четного порядка нет, следовательно, – нечётное.
3. Если – квадратичный невычет, а – вычет, то точек не существует, но существуют точки 4-го порядка. Сложение точек между собой, а также сложение с точками не создаёт новых точек, а значит, точки образуют группу 4го порядка. А значит, делится на 4. Других точек четного порядка нет, следовательно, – нечётное.

■

Итак, все скрученные кривые Эдвардса, не удовлетворяющие условиям Теоремы 1, у которых параметры и не являются квадратичными вычетами одновременно, имеют порядок ( – нечетное), а значит, их можно рассматривать к использованию в криптографии.

# 4. Параметры пригодных кривых по ГОСТ 34.10-2018

Пусть – кривая в скрученной форме Эдвардса, – порядок группы точек кривой – порядок циклической подгруппы группы точек .

Помимо общих параметров, несоответствие которым влечёт уязвимость независимо от размера других параметров кривой, есть дополнительные ограничения со стороны ГОСТ 34.10-2018 [5]:

1. – должно быть простым, кроме того, должны быть выполнены следующие условия

(13)

Из этих условий следует, что мы должны найти способы подсчета порядка точек на кривой. Из-за размера групп точек, требуемых в ГОСТ 34.10-2018, мы не можем считать его напрямую, нам требуется более эффективный алгоритм.

Согласно предыдущей главе, мы ищем кривые в скрученной форме Эдвардса, у которых порядок группы точек равен где – нечетное. В криптографических целях добавим более строгое ограничение – должно быть простым.

Тогда порядок подгруппы точек этой кривой может принимать значения , из которых нам подходит только . Согласно теоремам 1 и 2, у таких кривых нет точек 4го порядка, а значит, максимальный порядок точки равен . А значит, нашей задачей будет научиться отличать точки порядка от точек порядка .

**Теорема 3:**

Для скрученной кривой Эдвардса порядка , не удовлетворяющей Теореме 1, с простым на 2 делятся точки порядка , но не делятся точки порядка .

**Доказательство:**

Так как кривая, не соответствующая условиям теоремы 1, не имеет точек 4-го порядка, то порядки точек могут быть только .

В прошлой главе мы показали, что на такой кривой существуют 3 точки второго порядка и 1 точка первого порядка. Эти точки образуют подгруппу, и из них удвоением мы не получим точки порядка или порядка Значит, мы можем попытаться получить точки порядка или порядка только из точек этих же порядков.

Пусть точка порядка . Удвоением мы получим точку порядка . То есть удвоением точки порядка мы не получим точку порядка .

Остаются точки порядка , но их удвоением мы тоже не получим точку порядка , а значит, утверждение теоремы доказано.

■

Таким образом, если известно, что порядок группы , где – простое и или , то выбрать подходящую точку можно следующим алгоритмом.

**Алгоритм 1 (Генерация точки, удовлетворяющей ГОСТ 34.10-2018, при условии, что кривая удовлетворяет условиям Теоремы 1 и ГОСТ 34.10-2018, и её порядок равен , где – простое и или**

**Шаг 1** Выбираем случайную точку кривой

**Шаг 2** Проверяем, что точка не принадлежит подгруппе

Если точка принадлежит подгруппе , то возвращаемся на Шаг 1, иначе переходим на шаг 3

**Шаг 3** Удваиваем . .

Точка – генератор подгруппы точек, удовлетворяющий ГОСТ 34.10-2018. Нетрудно заметить, что трудоемкость этого алгоритма равна , где – время операции удвоения точки, а – константа.

Таким образом, если мы выбрали подходящую кривую, то выбрать приемлемую точку можно «очень быстро».

# 5. Описание алгоритма цифровой подписи по ГОСТ 34.10-2018

## 5.1 Параметры алгоритма

Схема цифровой подписи ГОСТ 34.10-2018 имеет следующие параметры:

1. Нечетное простое число – модуль эллиптической кривой или кривой в скрученной форме Эдвардса;
2. Кривая E над порядка ;
3. Простое число – порядок циклической подгруппы точек кривой , для которой выполнены условия (13);
4. Точка кривой с координатами , удовлетворяющая равенству ;
5. Хеш-функция , отображающая - сообщения, представленные в виде двоичных векторов произвольной конечной длины, в - двоичные векторы длины бит. Хеш-функция определена в [6] и также реализована в рамках этой курсовой работы.

Если , то . Если , то .

Каждый пользователь схемы цифровой подписи должен обладать личными ключами:

1. Ключ подписи – целое числом , удовлетворяющее равенству ;
2. Ключ проверки подписи – точка кривой с координатами , удовлетворяющая равенству .

Кроме того, по ГОСТ должны выполняться следующие условия:

1. и ;

2. .

Также для удобства генерации точки на кривой и увеличения скорости вычислений согласно теоремам 1-3 должны выполняться следующие условия:

1. Если , то не выполняется условие .

Если , то не выполняется хотя бы одно из условий

1. Для лучшей производительности рекомендуется брать по следующему принципу:

## 5.2 Алгоритмы цифровой подписи

Обозначим символом || конкатенацию.

Соответствие между числами и двоичными векторами задаётся следующим образом.

Пусть есть двоичный вектор длины бит, в котором младшие биты расположены справа, а старшие – слева:

Число соответствует двоичному вектору , если выполнено равенство

Перед формированием цифровой подписи кривую и точку необходимо выбрать заранее. Точку можно использовать несколько раз при формировании подписи, она является открытой. Считается, что для компрометации точки нужно минимум полгода, поэтому для надёжности точку лучше генерировать заново раз в какой-то период (например, раз в квартал). Кривую же можно зафиксировать «навсегда». Также необходимо заранее сгенерировать число – секретный ключ подписи. И найти точку . Это ключ проверки подписи, который также является открытым.

**Алгоритм 2: Формирование цифровой подписи**

**Вход:** сообщение

**Шаг 1:** Вычислить хеш сообщения

**Шаг 2:** Вычислить , двоичным представлением которого является вектор , и определить Если , то положить ;

**Шаг 3:** Сгенерировать случайное (псевдослучайное) целое число , удовлетворяющее неравенству ;

**Шаг 4:** Вычислить точку с координатами и определить . Если , то вернуться на шаг 3;

**Шаг 5:** Вычислить значение . Если , то вернуться на шаг 3;

**Шаг 6:** Вычислить двоичные векторы и , соответствующие и , и определить цифровую подпись как конкатенацию .

**Алгоритм 3: Проверка цифровой подписи**

**Вход:**  – цифровая подпись по полученным сообщением

**Шаг 1:** По полученной подписи вычислить целые числа и по их двоичным векторам . Если выполнены неравенства , то перейти к следующему шагу. В противном случае подпись неверна.

**Шаг 2:** Вычислить хеш полученного сообщения :

**Шаг 3:** Вычислить целое число , двоичным представлением которого является и определить . Если , то положить

**Шаг 4:** Вычислить значение ;

**Шаг 5:** Вычислить значения

**Шаг 6:** Вычислить точку с координатами и определить ;

**Шаг 7:** если выполнено равенство , то подпись принимается, в противном случае подпись неверна.

## Как это работает?

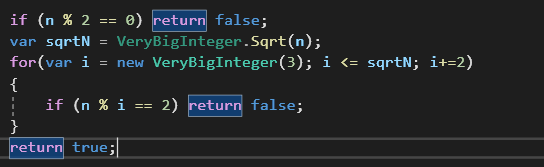
Если сообщение не было изменено, то хеш от сообщения при передаче и хеш от сообщения при получении равны. А значит, будет одним и тем же. То есть в эту подпись заложена не только проверка подписанта, но и проверка целостности сообщения.

Проверим, что правильную подпись Алгоритм 3 подтвердит.

Мы получили, что при такой проверке действительно равно , а значит, , и подпись будет принята.

# Проблемы и особенности реализации

Первой проблемой является алгоритм детерминированной проверки числа на простоту. Самым быстрым универсальным (под универсальностью я подразумеваю, что алгоритм работает для любого произвольного числа) и не требующим подготовительных расчетов является алгоритм, трудоемкость которого , где – число, которое проверяется на простоту. Так как мы работаем с числами порядка , то этот алгоритм проверки нам не подойдёт (при таких данных сложность алгоритма будет операций, а это «очень долго». На хорошей машине подобное количество операций будет выполняться не один месяц). Код этого алгоритма приведён ниже на языке С#. Тут класс VeryBigInteger реализует работу с большими целыми числами.



В связи с этим нужно придумать какой-то другой детерминированный алгоритм, возможно, с предварительно рассчитанными данными. За основу возьмём тест Миллера-Рабина.

Пусть – составное число. Число называется псевдопростым по основанию – если либо .

Если сделать проверку по нескольким основаниям, то тест становится не вероятностным, а детерминированным. При этом алгоритм проверки будет работать «очень быстро» - за несколько секунд или минут.

Остаётся выяснить, как подобрать основания для проверки чисел на простоту, чтобы после того, как число пройдёт тесты Миллера-Рабина для всех этих оснований, мы могли утверждать, что число, не большее (возьмем чуть-чуть с запасом), точно является простым и не окажется сильно псевдопростым по всем этим основаниям.

Исследовав множество различных источников, удалось найти теорему [6], которая гласит следующее:  
Пусть – это минимальное сильно псевдопростое число по первым простым основаниям (то есть в качестве оснований берутся первых простых чисел). Тогда . Отсюда мы можем получить оценку на то, какое нужно взять для нашей цифровой подписи. В рамках этой работы я буду ориентироваться на числа порядка (такие числа имеют в десятичной записи менее 155 цифр).

*,* тогда мы можем получить из неравенства

Или в общем виде получаем, что

*,* где – количество десятичных цифр числа

То есть если число меньше и прошло тест Миллера-Рабина по основаниям - первых простых чисел, то это число является простым. Практика показывает, что требуется гораздо меньшее количество простых чисел в качестве основания, но достоверные данные для известны пока только для чисел, меньших 3317044064679887385961981 (это примерно ), поэтому мы можем пользоваться только оценкой.

Найти первые простых чисел – это посильная задача, которую можно решить детерминированным алгоритмом, описанным в начале этой главы. Результаты её решения можно найти в файле Primes.txt

Итого, получаем следующий алгоритм проверки чисел на простоту.

**Алгоритм 4 (проверка числа на простоту):**

**Шаг 1**. Прочитать из файла Primes.txt простых чисел

**Шаг 2.** Запустить алгоритм Миллера-Рабина для каждого простого числа в качестве основания.

**Шаг 3.** Если тест хотя бы по одному из оснований не пройден, то число является составным. Иначе число является простым.

Именно этот алгоритм проверки числа на простоту реализован в методе IsPrime класса VeryBigInteger.

Для чисел порядка этот алгоритм выполняется не более 20 секунд в многопоточном приложении (8 потоков), для чисел порядка этот алгоритм выполняется не более 10 минут в многопоточном приложении.

Такая большая разница во времени обусловлена количеством проверяемых оснований и скорость произведения операций над большими числами. Чтобы снизить влияние второго фактора, можно использовать многомодульную арифметику. После её применения для чисел порядка алгоритм выполняется не более 7 минут.

Следующей особенностью является извлечение квадратного корня из числа в поле. Для решения этой задачи был выбран алгоритм, рассказанный в курсе «Теоретико-числовые методы криптографии» [7].

**Алгоритм 5 (извлечение квадратного корня из числа по простому нечётному модулю ):**

**Шаг 1**. Если Тогда , а результатом алгоритма будут числа

Иначе перейти на Шаг 2

**Шаг 2**. Если Тогда , а результатом алгоритма будут числа

и

Иначе перейти на Шаг 3

**Шаг 3**. Представим в виде , где - нечётно

**Шаг 4**. Выберем , которое является квадратичным невычетом по модулю

**Шаг 5**. Перебором найдем такое, что

**Шаг 6**. Результатом алгоритма будут числа

Данный алгоритм реализован в методе Sqrt класса FmodElement.

В процессе реализации этого алгоритма, необходимо реализовать алгоритм, который будет искать символ Лежандра для числа. Алгоритм вычисления символа Лежандра «в лоб» требует разложения числа на множители. С учётом десятичной длины используемых чисел такой подход займёт очень много времени, поэтому воспользуемся утверждением, которое было доказано в рамках курса «Теоретико-числовые методы криптографии»

**Утверждение 1:**

Таким образом вместо прямого вычисления символа Лежандра будем выполнять проверку Для вычисления можно воспользоваться алгоритмом быстрого возведения в степень. Таким образом, для расчёта символа Лежандра нам потребуется не более шагов.

**Заключение**

Таким образом, в рамках работы обоснована надежность кривых в скрученной форме Эдвардса для задачи формирования цифровой подписи, а также реализована цифровая подпись ГОСТ 34.10-2018 на основании этих кривых.

Кривые в скрученной форме Эдвардса обладают однородным законом сложения, в отличие от кривых в форме Вейерштрасса, тем самым становится невозможной атака по побочным каналам (из-за неоднородного закона сложения, при выполнении операции сложения с разными точками на кривой в форме Вейерштрасса будет разное время выполнения этой операции, что даёт некоторую информацию злоумышленнику). Более того, закон сложения на кривых в скрученной форме Эдвардса является полным, то есть действует для любых двух точек, принадлежащий этой кривой. Также для кривых в скрученной форме Эдвардса можно выбрать такие параметры, что у выбранной кривой совсем не будет особых точек, что очень удобно. Также при переходе в проективные координаты есть ряд приёмов, которые позволяют ускорить вычисления на кривых Эдвардса по сравнению с кривыми в форме Вейерштрасса.

Кривые Вейерштрасса в свою очередь обладают более простым с точки зрения вычислений законом сложения, тем самым прямая реализация кривых Вейерштрасса работает быстрее прямой реализации кривых Эдвардса. Также кривые в форме Вейерштрасса более тщательно изучены, чем кривые в скрученной форме Эдвардса, что, в свою очередь, может быть и минусом, и плюсом. В связи с этим вопрос генерации кривых в форме Вейерштрасса стоит менее остро, чем у кривых в скрученной форме Эдвардса, так как известно больше теорем и приёмов, которые помогают сгенерировать свою криптостойкую кривую. Для кривых Эдвардса, в свою очередь, на данный момент есть лишь очень узкий спектр теорем, которые дают не только необходимые условия криптостойкости кривой, но и достаточные. Но, к сожалению, даже эти условия очень трудоёмко (с точки зрения вычислений) проверить. Таким образом, есть лишь очень узкий спектр кривых, которые можно генерировать самим.

Иными словами, и у кривых в форме Вейерштрасса, и у кривых в скрученной форме Эдвардса есть существенные преимущества и недостатки. У кривых в форме Вейерштрасса некоторые недостатки заложены в самой структуре абелевой группы, тем самым, их нельзя исправить. А у кривых в скрученной форме Эдвардса недостатки больше связаны с их недостаточной изученностью, что даёт надежду на их исправление в будущем.

## Приложение А (некоторые функции программной реализации схемы цифровой подписи ГОСТ 34.10 - 2018)

Весь код хранится в репозитории <https://github.com/pr0gramm-ist/DigitalSignature>

В данном приложении опущены служебные методы, такие как переопределение операторов арифметических операций, операций сравнения, битовых операций, конструкторы, а также весь код, связанный с интерфейсом пользователя.

Краткое описание используемых классов:

1. *VeryBigInteger* – класс, реализующий работу с большими числами, расширяющий функциональность класса BigInteger из библиотеки System.Numerics
2. *PrimeNumbers* – класс, содержащий простые числа, выступающий в качестве оснований для теста Миллера-Рабина, а также модули для многомодульной арифметики над большими числами
3. *FmodElement* – реализует логику элемента из кольца
4. *EdwardsCurve –* реализует основные параметры кривой в скрученной форме Эдвардса, алгоритм генерации точки на кривой, а также описание стандартных кривых из ГОСТ 34.10-2018 [4]
5. *EdwardsCurvePoint –* реализует логику работы с токами на кривых в скрученной форме Эдвардса, включая операции сложения точек, умножения точки на число, подсчета порядка точки, получение противоположного элемента группы точек, а также описание точек для стандартных кривых из ГОСТ 34.10-2018 [4]
6. *Streebog –* реализация хеш-функции «Стрибог» согласно ГОСТ 34.11-2018 [5]
7. *GOST3410\_2018 –* реализует основные методы из ГОСТ 34.10-2018 [4]: проверка кривой, подписывание сообщения, проверка подписи, генерация секретного ключа.

Некоторые методы класса кривой в скрученной форме Эдвардса EdwardsCurve и класса точки этой кривой EdwardsCurvePoint

**public** **class** **EdwardsCurve**

{

//eu^2 + v^2 = 1 + du^2 \* v^2 (mod p)

**public** VeryBigInteger p { **get**; }

**public** FmodElement e { **get**; }

**public** FmodElement d { **get**; }

**public** VeryBigInteger countOfPoints { **get**; **private** **set**; }

**public** **string** Name { **get**; **set**; } = "";

**public** **bool** isGoodCurve { **get**; **private** **set**; }

**private** **static** EdwardsCurve Id\_tc26\_gost\_3410\_2012\_256\_paramSetA = **null**;

**private** **static** EdwardsCurve Id\_tc26\_gost\_3410\_2012\_512\_paramSetC = **null**;

#region КривыеИзГОСТ

**public** **static** EdwardsCurve **id\_tc26\_gost\_3410\_2012\_256\_paramSetA**()

{

EdwardsCurve curve;

**if** (Id\_tc26\_gost\_3410\_2012\_256\_paramSetA **is** **null**)

{

**var** p = **new** VeryBigInteger("00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97", **true**);

**var** e = **new** VeryBigInteger("01", **true**);

**var** d = **new** VeryBigInteger("0605F6B7C183FA81578BC39CFAD518132B9DF62897009AF7E522C32D6DC7BFFB", **true**);

**var** m = **new** VeryBigInteger("01000000000000000000000000000000003F63377F21ED98D70456BD55B0D8319C", **true**);

curve = **new** EdwardsCurve(e, d, p, m, **true**);

curve.Name = "id-tc26-gost-3410-2012-256-paramSetA";

}

**else**

{

curve = Id\_tc26\_gost\_3410\_2012\_256\_paramSetA;

}

**return** curve;

}

**public** **static** EdwardsCurve **id\_tc26\_gost\_3410\_2012\_512\_paramSetC**()

{

EdwardsCurve curve;

**if** (Id\_tc26\_gost\_3410\_2012\_512\_paramSetC **is** **null**)

{

**var** p = **new** VeryBigInteger("00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDC7", **true**);

**var** e = **new** VeryBigInteger("01", **true**);

**var** d = **new** VeryBigInteger("009E4F5D8C017D8D9F13A5CF3CDF5BFE4DAB402D54198E31EBDE28A0621050439CA6B39E0A515C06B304E2CE43E79E369E91A0CFC2BC2A22B4CA302DBB33EE7550", **true**);

**var** m = **new** VeryBigInteger("00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF26336E91941AAC0130CEA7FD451D40B323B6A79E9DA6849A5188F3BD1FC08FB4", **true**);

curve = **new** EdwardsCurve(e, d, p, m, **true**);

curve.Name = "id-tc26-gost-3410-2012-512-paramSetC";

}

**else**

{

curve = Id\_tc26\_gost\_3410\_2012\_512\_paramSetC;

}

**return** curve;

}

#endregion

**public** EdwardsCurvePoint **GeneratePoint**()

{

List<FmodElement> vVariants = **null**;

FmodElement u = **new** FmodElement(**1**, p);

**while** (vVariants **is** **null** && u != **0**)

{

u = **new** FmodElement(VeryBigInteger.NextRandomNumber(), p);

**var** vChislitel = **new** FmodElement((u \* u \* e).Value - **1**, p);

**var** vZnamenatel = **new** FmodElement((u \* u \* d).Value - **1**, p);

**var** vSquare = vChislitel / vZnamenatel;

vVariants = FmodElement.Sqrt(vSquare);

}

**var** random = **new** Random();

**var** v = vVariants[random.Next(vVariants.Count - **1**)];

**var** groupGenerator = **new** EdwardsCurvePoint(u, v, **this**);

**return** groupGenerator \* **2**;

}

}

**public** **class** **EdwardsCurvePoint**

{

**public** EdwardsCurve edwardsCurve { **get**; **private** **set**; }

**public** FmodElement x { **get**; }

**public** FmodElement y { **get**; }

**private** VeryBigInteger order = **null**;

**public** EdwardsCurvePoint **Zero**()

{

**var** xCootdinate = **new** FmodElement(VeryBigInteger.Zero(), x.p);

**var** yCootdinate = **new** FmodElement(VeryBigInteger.One(), x.p);

**return** **new** **EdwardsCurvePoint**(xCootdinate, yCootdinate, edwardsCurve);

}

**public** VeryBigInteger Order

{

**get**

{

**if**(order **is** **null**)

{

FillOrder();

}

**return** order;

}

}

**private** **void** **FillOrder**()

{

**var** pointsCnt = edwardsCurve.countOfPoints;

**var** dividers = **new** List<VeryBigInteger>();

dividers.Add(**new** VeryBigInteger(**1**));

**if** (edwardsCurve.isGoodCurve)

{

dividers.Add(**new** VeryBigInteger(**2**));

dividers.Add(**new** VeryBigInteger(**4**));

dividers.Add(edwardsCurve.countOfPoints / **4**);

dividers.Add(edwardsCurve.countOfPoints / **2**);

dividers.Add(edwardsCurve.countOfPoints);

}

**else**

{

**var** primes = PrimeNumbers.Numbers;

**var** temp = **new** VeryBigInteger(pointsCnt.**value**);

**var** tempIsPrime = **false**;

**for** (**int** i = **0**; i < primes.Count && temp > VeryBigInteger.One() && !tempIsPrime; i++)

{

**var** temp2 = VeryBigInteger.One();

**while** (temp % primes[i] == **0**)

{

temp2 \*= primes[i];

dividers.Add(temp2);

dividers.Add(temp / primes[i]);

temp /= primes[i];

}

tempIsPrime = temp.IsPrime();

}

**if** (!tempIsPrime)

{

**for** (VeryBigInteger i = primes.Last() + **2**; i \* i < temp; i += **2**)

{

**var** temp2 = VeryBigInteger.One();

**while** (temp % i == **0**)

{

temp2 \*= i;

dividers.Add(i);

dividers.Add(temp / i);

temp /= i;

}

}

}

dividers.Sort();

dividers.Add(edwardsCurve.countOfPoints);

}

**foreach** (**var** divider **in** dividers)

{

**if**(**this** \* divider == Zero())

{

order = divider;

**break**;

}

}

}

**public** EdwardsCurvePoint **OppositePoint**()

{

**return** **new** **EdwardsCurvePoint**(-**1** \* x, y, edwardsCurve);

}

**public** **static** EdwardsCurvePoint **operator** +(EdwardsCurvePoint firstValue, EdwardsCurvePoint secondValue)

{

**var** p = firstValue.edwardsCurve.p;

**var** xChislitel = firstValue.x \* secondValue.y + secondValue.x \* firstValue.y;

**var** xZnamenatel = **1** + firstValue.edwardsCurve.d \* firstValue.x \* firstValue.y \* secondValue.x \* secondValue.y;

**var** x = xChislitel / xZnamenatel;

**var** yChislitel = firstValue.y \* secondValue.y - firstValue.edwardsCurve.e \* firstValue.x \* secondValue.x;

**var** yZnamenatel = **1** - firstValue.edwardsCurve.d \* firstValue.x \* firstValue.y \* secondValue.x \* secondValue.y;

**var** y = yChislitel / yZnamenatel;

**return** **new** **EdwardsCurvePoint**(x, y, firstValue.edwardsCurve);

}

**public** **static** EdwardsCurvePoint **operator** \*(EdwardsCurvePoint x, VeryBigInteger k)

{

**if** (k == **1**)

**return** x;

**if** (k == **2**)

**return** x + x;

**if** ((k & **1**) == **0**)

{

**var** temp = x + x;

**return** temp \* (k >> **1**);

}

**else**

**return** x + x \* (k - **1**);

}

#region ТочкиИзГОСТ

**public** **static** EdwardsCurvePoint **id\_tc26\_gost\_3410\_2012\_256\_paramSetA**()

{

**var** edwardCurve = EdwardsCurve.id\_tc26\_gost\_3410\_2012\_256\_paramSetA();

**var** u = **new** VeryBigInteger("0D", **true**);

**var** v = **new** VeryBigInteger("60CA1E32AA475B348488C38FAB07649CE7EF8DBE87F22E81F92B2592DBA300E7", **true**);

**return** **new** **EdwardsCurvePoint**(**new** FmodElement(u, edwardCurve.p), **new** FmodElement(v, edwardCurve.p), edwardCurve);

}

**public** **static** EdwardsCurvePoint **id\_tc26\_gost\_3410\_2012\_512\_paramSetC**()

{

**var** edwardCurve = EdwardsCurve.id\_tc26\_gost\_3410\_2012\_512\_paramSetC();

**var** u = **new** VeryBigInteger("12", **true**);

**var** v = **new** VeryBigInteger("469AF79D1FB1F5E16B99592B77A01E2A0FDFB0D01794368D9A56117F7B38669522DD4B650CF789EEBF068C5D139732F0905622C04B2BAAE7600303EE73001A3D", **true**);

**return** **new** **EdwardsCurvePoint**(**new** FmodElement(u, edwardCurve.p), **new** FmodElement(v, edwardCurve.p), edwardCurve);

}

#endregion

}

Методы класса GOST\_3410\_2018

**public** **class** **GOST3410\_2018**

{

**public** **static** **bool** **ItsGoodCurve**(EdwardsCurve edwardsCurve)

{

**if** (!edwardsCurve.p.IsPrime()) **return** **false**;

**if** (edwardsCurve.countOfPoints == edwardsCurve.p) **return** **false**;

**var** q = edwardsCurve.countOfPoints / **4**;

**if** (q.IsPrime() == **false**) **return** **false**;

**var** number2\_254 = **new** VeryBigInteger(**254**, **true**);

**var** number2\_256 = **new** VeryBigInteger(**256**, **true**);

**var** number2\_508 = **new** VeryBigInteger(**508**, **true**);

**var** number2\_512 = **new** VeryBigInteger(**512**, **true**);

**var** B = **1**;

**if**(q > number2\_254 && q < number2\_256)

{

B = **31**;

}

**else** **if** (q > number2\_508 && q < number2\_512)

{

B = **131**;

}

**else**

{

**return** **false**;

}

**var** tempp = **new** FmodElement(edwardsCurve.p, q);

**for**(**int** t = **1**; t <= B; t++)

{

**if** (tempp == **1**) **return** **false**;

tempp \*= edwardsCurve.p;

}

**var** pmod4 = edwardsCurve.p % **4**;

**if**(pmod4 == **3** && edwardsCurve.e.LegendreSymbol() == edwardsCurve.d.LegendreSymbol() && edwardsCurve.e.LegendreSymbol() == -**1**)

{

**return** **false**;

}

**if** (pmod4 == **1** && edwardsCurve.e.LegendreSymbol() == edwardsCurve.d.LegendreSymbol() && edwardsCurve.e.LegendreSymbol() == **1**)

{

**return** **false**;

}

**return** **true**;

}

**public** **static** BitArray[] **SignAMessage**(**byte**[] messageBytes, EdwardsCurvePoint point, VeryBigInteger privateKey)

{

**var** signLen = **256**;

**if**(point.Order > **new** VeryBigInteger(**260**, **true**))

{

signLen = **512**;

}

**var** temp = **new** **byte**[messageBytes.Length];

**for**(**int** i = **0**; i < temp.Length; i++)

{

temp[i] = messageBytes[i];

}

**byte**[] hash = Streebog.GetHash(messageBytes, signLen);

**var** hashBits = **new** BitArray(hash);

**var** e = bitsToNumber(hashBits);

e = e % point.Order;

**if** (e == **0**) e = VeryBigInteger.One();

**var** r = VeryBigInteger.Zero();

**var** k = VeryBigInteger.One();

**var** s = **new** FmodElement(**0**, point.Order);

**while** (r == **0** && s == **0**)

{

k = VeryBigInteger.NextRandomNumber() % point.Order;

**var** C = k \* point;

r = C.x.Value % point.Order;

s = **new** FmodElement(r \* privateKey + k \* e, point.Order);

}

**var** rVector = numberToBits(r, signLen);

**var** sVector = numberToBits(s.Value, signLen);

**var** sign = **new** BitArray[**2**];

sign[**0**] = rVector;

sign[**1**] = sVector;

**return** sign;

}

**public** **static** **async** Task<BitArray[]> SignAMessageAsync(**byte**[] messageBytes, EdwardsCurvePoint point, VeryBigInteger privateKey)

{

**var** result = **await** Task.Run(() => SignAMessage(messageBytes, point, privateKey));

**return** result;

}

**public** **static** **bool** **CheckSign**(**byte**[] messageBytes, BitArray[] sign, EdwardsCurvePoint P, EdwardsCurvePoint Q)

{

**var** r = bitsToNumber(sign[**0**]);

**if**(r <= **0** || r >= P.Order)

{

**return** **false**;

}

**var** s = bitsToNumber(sign[**1**]);

**if** (s <= **0** || s >= P.Order)

{

**return** **false**;

}

**var** signLen = sign[**0**].Length;

**byte**[] hash = Streebog.GetHash(messageBytes, signLen);

**var** hashBits = **new** BitArray(hash);

**var** e = bitsToNumber(hashBits) % P.Order;

**if** (e == **0**) e = VeryBigInteger.One();

**var** v = **new** FmodElement(e, P.Order);

v = v.Reverse();

**var** C = v.Value \*(s \* P + (r \* Q).OppositePoint());

**if**(C.x.Value % P.Order == r)

{

**return** **true**;

}

**else**

{

**return** **false**;

}

}

**public** **static** **async** Task<**bool**> CheckSignAsync(**byte**[] messageBytes, BitArray[] sign, EdwardsCurvePoint P, EdwardsCurvePoint Q)

{

**var** result = **await** Task.Run(() => CheckSign(messageBytes, sign, P, Q));

**return** result;

}

**public** **static** VeryBigInteger **GenerateKey**(EdwardsCurvePoint P)

{

**return** VeryBigInteger.NextRandomNumber() % P.Order;

}

**private** **static** VeryBigInteger **bitsToNumber**(BitArray bits)

{

**var** binaryPow = VeryBigInteger.One();

**var** result = VeryBigInteger.Zero();

**for** (**int** i = **0**; i < bits.Length; i++)

{

**if** (bits[i])

{

result += binaryPow;

}

binaryPow \*= **2**;

}

**return** result;

}

**private** **static** BitArray **numberToBits**(VeryBigInteger number, **int** outLen)

{

**var** result = **new** BitArray(outLen);

**var** temp = number;

**int** i = **0**;

**while**(temp > **0**)

{

**var** digit = (**int**)VeryBigInteger.Log(temp, **2**);

result[digit] = **true**;

temp -= **new** VeryBigInteger(digit, **true**);

}

**return** result;

}

}

# Список используемой литературы:

1. Ростовцев А.Г. Алгебраические основы криптографии. – Спб.: НПО «Мир и семья», ООО «Интерлайн», 2000. – 354с.: илл.
2. Бессалов А.В. Эллиптические кривые в форме Эдвардса и криптография: монография. – Киев: ІВЦ «Видавництво «Політехника»», 2017. –272с.
3. Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, Christiane Peters Twisted Edwards Curves. – Department of Mathematics, Statistics, and Computer Science (M/C 249) University of Illinois at Chicago, IL 60607-7045, USA, 2008, - 17с.
4. ГОСТ 34.10-2018 – Межгосударственный стандарт «Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи», Стандартинформ, 2018. - 20с.
5. ГОСТ 34.11-2018 – Межгосударственный стандарт «Криптографическая защита информации. Функция хеширования», Стандартинформ, 2018. - 23с.
6. Eric Bach, Explicit bounds for primality testing and related problems, Math. Comp. 55 (1990), no. 191, 355{380. MR 1023756 (91m:11096)
7. С.И.Яблокова «Задачи по курсу «Теоретико-числовые методы в криптографии»»: практикум; Яросл. гос. Ун-т им. П. Г. Демидова. – Ярославль: ЯрГУ, 2024. – Ч. 1. – 64 с.