МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

ПРОФИЛЬ СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему **Исследование алгоритма эллиптического шифрования**

Студентка В.К.Байдицкая _____

Руководитель Е.М. Гунченко _____

Допустить к защите
Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

СтудентkaБайдицкая Владислава Константиновна

1. Тема Исследование алгоритма эллиптического шифрования

2. Срок сдачи студентом законченной выпускной квалификационной работы 17.06.2016

3. Исходные данные к выпускной квалификационной работе

1. Теория об эллиптических кривых
2. Стандарты отечественных алгоритмов шифрования ГОСТ Р 34.10-1994 ГОСТ Р 34.10-2001 ГОСТ Р 34.10-2012
3. Документация языка Python

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов)

Введение

Глава 1Общие теоретические сведения об эллиптических кривых

1.1 Математическое определение эллиптических кривых

1.2 Эллиптические кривые над действительными числами

1.3Сложение и умножение эллиптических кривых

Глава 2 Электронно-цифровая подпись

2.1 Эволюция цифровой подписи

2.2 Американские стандарты шифрования

2.3 Российские стандарты шифрования

2.4 Теоретическая стойкость схемы ЭЦП

Глава 3 Разработка схем ЭЦП ГОСТ Р 34.10

3.1 Практическая реализация алгоритмов ЭЦП на языке программирования Python

3.2 Тестирование программного продукта.

3.3 Производительность и эффективность схемы ЭЦП

Заключение

Список использованной литературы

5. Ориентировочный перечень графического и иллюстративного материала

1. Презентация на тему выпускной квалификационной работы
2. Блок схема алгоритма эллиптического шифрования
3. Программная реализация алгоритма.

6. Дата выдачи задания « 11 » января 2016 г.

Руководитель выпускной квалификационной работы _____ Е.М. Гунченко

Задание принял к исполнению _____ В.К. Байдицкая

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Байдицкой В.К.
по теме Исследование алгоритма эллиптического шифрования

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Исследование алгоритма эллиптического шифрования	11.01.2016	11.01.2016	выполнено	
Написание первой главы работы (Изучение теоретических сведений об эллиптических кривых)	23.01.2016	23.01.2016	выполнено	
Выявление плюсов и минусов алгоритма эллиптического шифрования	25.02.2016	25.02.2016	выполнено	
Реализация алгоритма шифрования	01.03.2016	01.03.2016	выполнено	
Написание второй главы работы (Изучение аспектов электронно-цифровой подписи)	26.03.2016	26.03.2016	выполнено	

Тестирование и выявление ошибок в реализованном алгоритме	6.04.2016	6.04.2016	выполнено	
Написание третьей главы (Реализация и тестирование алгоритма)	14.04.2016	14.04.2016	выполнено	
Подведение итогов, редактирование бакалаврской работы. Создание презентационного материала	21.04.2016	21.04.2016	выполнено	
Предварительная защита	31.05.2016	31.05.2016	выполнено	
Проверка на наличие заимствований (плагиата) в системе antiplagiat.ru	17.06.2016	17.06.2016	выполнено	
Сдача на кафедру комплекта документов для защиты	17.06.2016	17.06.2016	выполнено	
Защита бакалаврской работы	29.06.2016	29.06.2016	выполнено	

Руководитель выпускной квалификационной работы _____ Е.М. Гунченко

Задание принял к исполнению _____ В.К. Байдицкая

Аннотация

Работа на тему "Исследование алгоритма эллиптического шифрования", выполнила студентка Байдицкая Владислава Константиновна.

Объектом исследования в данной работе послужили алгоритмы, основанные на эллиптических кривых

Предметом исследованияработы являются стандарты электронно-цифровой подписи ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012

Цель: Реализация алгоритмов шифрования на основе эллиптических кривых

Для этого необходимо решить следующие задачи:

1.изучить теорию эллиптических кривых;

2.реализовать достаточно криптостойкий алгоритм на основе эллиптических кривых.

3.реализовать алгоритм с простой процедурой шифрования и дешифрования;

4.избежать чувствительности к небольшим ошибкам шифрования;

5. провести сравнительный анализ алгоритма ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012.

Бакалаврская работа выполнена на сорока страницах, состоит из введения, трёх глав, заключения и списка литературы, состоящего из двадцати литературных источников и 7 формул.

Первая глава работы посвящена изучению математической теории эллиптических кривых.

Во второй главе пойдет речь об отечественном стандарте цифровой подписи (ГОСТ Р 34.10-2012) о его эволюции, принципах дизайна и о перспективах его использования.

Третья глава раскрывает процесс реализации алгоритмов ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 на языке программирования python. Код реализации представлен в приложении.

Ключевые слова: эллиптические кривые, электронно-цифровая подпись, шифрование, дешифрование, ГОСТ Р 34.10-2001, ГОСТ Р 34.10-2012.

Оглавление

Введение

В современном мире ценность информации постоянно увеличивается, информатизация общества постоянно растет. Это приводит к необходимости совершенствовать методы и средства защиты информации.

К защищенным информационным системам предъявляется ряд особых требований, которые вытекают из свойств информации: конфиденциальности, доступности и целостности. Наиболее популярным методом защиты информации является использование криптографических алгоритмов.

В целях обеспечения защиты информации используются следующие криптографические примитивы: симметричные криптосистемы, ассиметричные криптосистемы, цифровые подписи, криптографические хэш-функции, коды проверки подлинности [14].

В связи с большим ростом сфер финансовой и коммерческой деятельности возрастает роль средств и систем криптографической защиты информации. Их рост связан с необходимостью перехода на «электронную основу», но и сильно расширившимися возможностями передачи, обработки и хранения информации в распределенных вычислительных системах. Применение специальных криптографических протоколов и криптосистем позволяет осуществлять многообразные экономические отношения «дистанционно», исключая необходимость личной встречи участников этих отношений, а также поддерживать при этом должную финансовую и правовую дисциплину, лучшим примером такого взаимодействия выступает электронно-цифровая подпись [5].

Для увеличения криптостойкости алгоритма цифровой подписи следует применять алгоритмы, основанные на эллиптических кривых.

В данной работе будут рассмотрены способы и преимущества реализации криптографических алгоритмов с использованием теории

эллиптических кривых и в качестве примера реализованы схемы электронно-цифровой подписи ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 [10].

Объектом исследования в данной работе послужили алгоритмы, основанные на эллиптических кривых

Предметом исследования работы являются стандарты электронно-цифровой подписи ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012

Целью бакалаврской работы является:

Реализация алгоритмов шифрования на основе эллиптических кривых

Задачи бакалаврской работы:

1.изучить теорию эллиптических кривых;

2.реализовать достаточно криптостойкий алгоритм на основе эллиптических кривых.

3.реализовать алгоритм с простой процедурой шифрования и дешифрования;

4.избежать чувствительности к небольшим ошибкам шифрования;

5. провести сравнительный анализ алгоритма ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012.

Глава 1Общие теоретические сведения об эллиптических кривых.

1.1 Математическое определение эллиптических кривых

Для того чтобы начать изучение алгоритмов, построенных на эллиптических кривых, необходимо определить понятие эллиптической кривой.

Эллиптические кривые — это кривые первого рода с рациональной точкой.

В свою очередь рациональные кривые (над алгебраически замкнутым полем) — это в точности алгебраические кривые рода 0.

А алгебраическая кривая — это геометрическое место (множество) точек на плоскости, которое определяется как множество нулей многочлена от двух переменных, в общем случае алгебраическое многообразие размерности 1. Алгебраическая кривая — это алгебраическое многообразие, все алгебраические подмногообразия которого состоят из одной точки [13].

Любая такая кривая может быть представлена как кубика без особенностей [6].

Кубика — плоская алгебраическая кривая третьего порядка. Множество точек плоскости, заданных кубическим уравнением которое применяется к однородным координатам на проективной плоскости. $F(x, y, z) = 0$

Пересечение двух коник (коника это пересечение плоскости с круговым конусом) является кривой четвёртого порядка рода 1, а значит, эллиптической кривой, если содержит хотя бы одну рациональную точку. В противном случае пересечение может быть рациональной кривой четвёртого порядка с особенностями, или быть разложимым на кривые меньшего порядка.

1.2 Эллиптические кривые над действительными числами

Для современной криптографии актуальна проблема повышения стойкости и уменьшения размера блоков, данных путем модификации уже существующих криптосистем.

Самый очевидный путь решения вышеупомянутой проблемы – представление блоков информации в криптографических алгоритмах не только в виде чисел (или элементов конечных полей), но и в виде иных алгебраических объектов большей сложности. Одним из весьма подходящих типов таких объектов являются точки эллиптических кривых [7]. Первоначально эллиптической кривой называлась гладкая кривая на декартовой плоскости, описываемая следующим уравнением:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad (1)$$

Если все коэффициенты и неизвестные – действительные числа, то путем замены переменных уравнение может быть преобразовано к следующему, более простому виду

$$y^2 = x^3 + ax + b \qquad (2)$$

Подставляя в данное уравнение различные значения аи bможно получить графики, изображенные на рисунке 1.1.

Эллиптические кривые делятся на два типа сингулярные и несингулярные. Эллиптические кривые представленные под значениями a,b,c,d на рисунке 1.1.называются несингулярными эллиптическим кривыми. Кривые e, f на рисунке 1.1. называются сингулярными эллиптическими кривыми [9]. Для несингулярных эллиптических кривых выполняется следующее неравенство:

(3)

Для сингулярных кривых это условие не выполняется.

Очень важен следующий факт: Нельзя использовать в схемах ЭЦП сингулярные кривые [8]. Используя сингулярные кривые есть риск значительно снизить стойкость схемы электронной цифровой подписи.

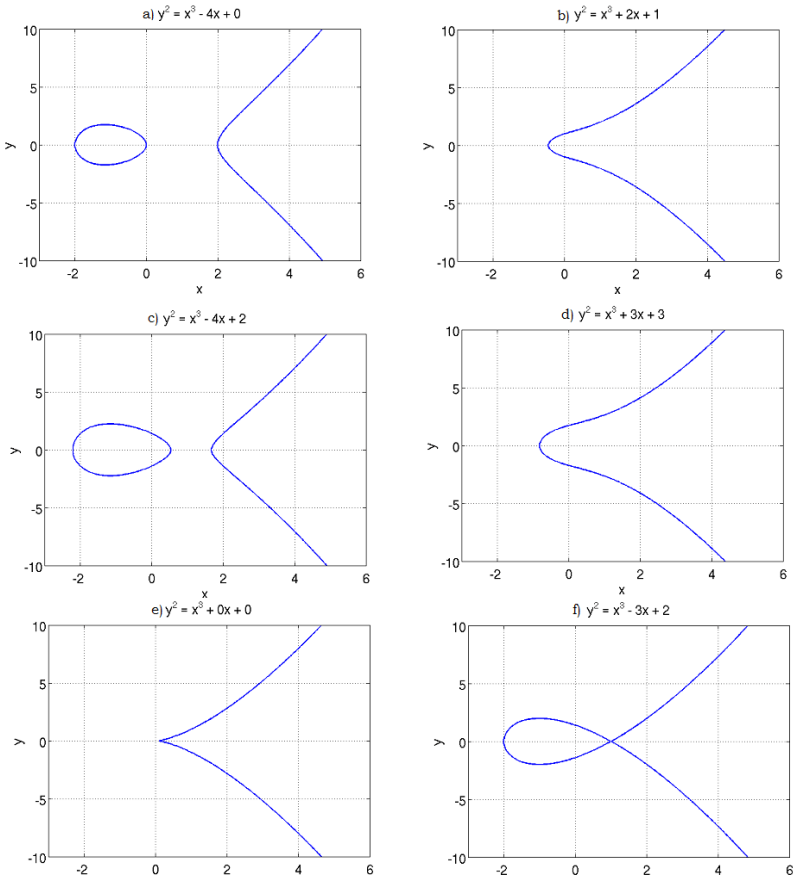

Рисунок 1.1- Виды эллиптических кривых

Эллиптические кривые не должны иметь особых точек. Геометрически это означает, что на графике не должно быть точек самопересечений и возврата. Алгебраически, достаточно того чтобы дискриминант не был равен нулю.

$$\Delta = -16(4a^3 + 27b^2) \tag{4}$$

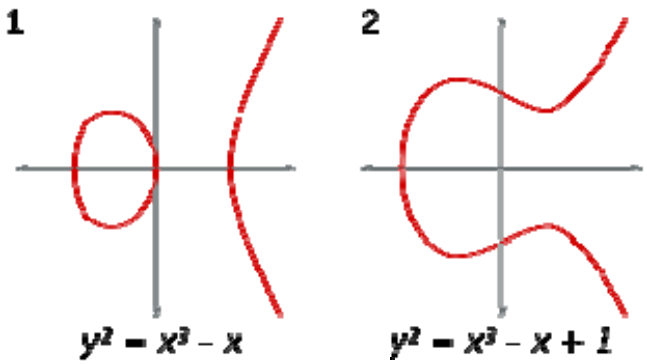

Рисунок 1.2.- Эллиптические кривые с одной и двумя компонентами связности

Если у кривой нет особых точек и дискриминант положителен, то у графика есть две связные компоненты. Если у кривой нет особых точек и дискриминант отрицателен, то у графика одна компонента. Например, для графиков выше (см. рисунок 1.2) в первом случае дискриминант равен 64, а во втором он равен −368.

Порядок точек эллиптической кривой

Порядок эллиптической кривой -порядок группы точек эллиптической кривой (число различных точек на E , включая точку O). Для эллиптической кривой E заданной над простым полем Fp , порядок m группы точек данной кривой зависит от размера поля, определяемого простым числом p .

Каждая точка P эллиптической кривой над простым полем $E(Fp)$ образует циклическую подгруппу G группы точек эллиптической кривой. Порядок циклической подгруппы группы точек эллиптической кривой (число точек в подгруппе) называется порядком точки эллиптической кривой. Точка P на $E(Fp)$ называется точкой порядка q , если: $qP = O$, где q – наименьшее натуральное число, при котором выполняется данное условие.

1.3 Сложение и умножение эллиптических кривых

Возможность сложения точек эллиптической кривой вытекает из следующих свойств кривых:

— Любая вертикальная (параллельная оси) прямая, не пересекает кривую ни разу или пересекает ее дважды.

— Любая другая прямая пересекает кривую один или три раза.

Существуют правила сложения и удвоения точек на эллиптической кривой:

Точка 0 является нулевым элементом. Так как, $0 = -0$,то для любой точки P на эллиптической кривой $P = P - 0$ [7].

Вертикальная линия пересекает кривую в двух точках с одной и той же координатой x, $S = (x, y)$ и $T = (x, -y)$. Эта прямая пересекает кривую и в бесконечно удаленной точке. Отсюда имеем и $P1 = -P2$.

Сложение:

Чтобы сложить две точки P и Q (см. рисунок 1.3) с разными координатами x, следует провести через эти точки прямую и найти её точку пересечения с эллиптической кривой. Если прямая не является касательной к кривой в точках P или Q, то существует только одна такая точка, обозначим ее S.

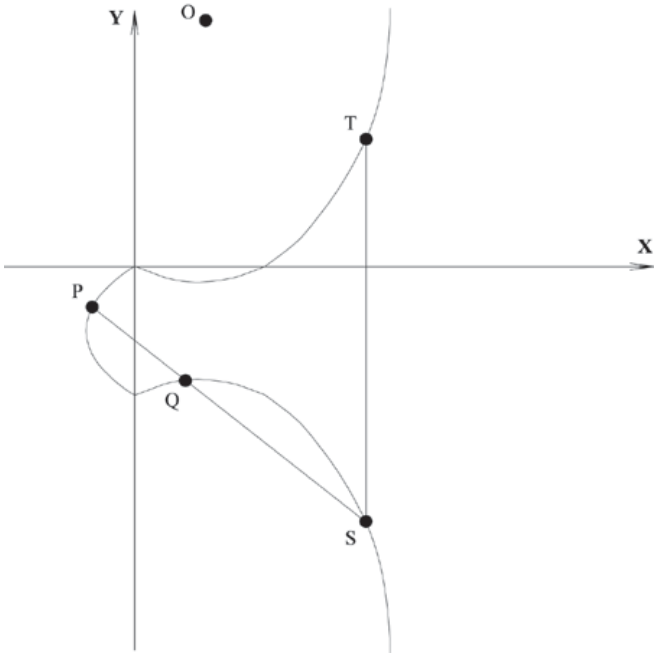

Рисунок 1.2- Сложение точек эллиптической кривой.

Согласно, предположению $P + Q + S = 0$ значит, $P + Q = -S$ или $P + Q = T$. Если прямая является касательной к кривой в какой-либо из точек P или Q, то в этом случае следует положить $S = P$ или $S = Q$.

Удвоение:

Чтобы удвоить точку Q, следует провести касательную в точке Q и найти точку пересечения S с эллиптической кривой. Тогда $Q + Q = 2xQ = -S$.

Операции сложения и удвоения подчиняются всем обычным правилам сложения, в частности коммутативному и ассоциативному

законам. Умножение точки P эллиптической кривой на положительное число k определяется как сумма k точек P.

В 1986 Н. Коблиц, В. Миллер предложили использовать эллиптические кривые в качестве основы криптографических алгоритмов в том числе и схемы подписи [8].

1.4Шифрование и дешифрование с использованием эллиптических кривых

Рассмотрим самый простой подход к шифрованию и дешифрованию с использованием эллиптических кривых. Задача состоит в том, чтобы зашифровать сообщение M, которое может быть представлено в виде точки на эллиптической кривой $Pm(x, y)$.

Так же как и в случае обмена ключом, в системе шифрования и дешифрования в качестве параметров рассматривается эллиптическая кривая $Ep(a, b)$ и точка G на ней. Участник B выбирает закрытый ключ nB и вычисляет открытый ключ $PB = nB\,x\,G$. Чтобы зашифровать сообщение Pm используется открытый ключ получателя B становится PB. Участник A выбирает случайное целое положительное число k и вычисляет зашифрованное сообщение Cm, являющееся точкой на эллиптической кривой [7].

$$Cm = \{kxG, Pm + kxPB\} \tag{5}$$

Чтобы дешифровать сообщение, человек B умножает первую координату точки на свой закрытый ключ и вычитает результат из второй координаты:

$$Pm + k\,x\,PB - nB\,x\,(k\,x\,G) = Pm + k\,x(nB\,x\,G) - nB\,x\,(k\,x\,G) = Pm \tag{6}$$

Человек A зашифровал сообщение Pm добавлением к нему $kxPB$. При этом никто не знает значения k, поэтому, хотя PB и является открытым

ключом, никто не знает $k\ x\ PB$. Противнику для восстановления сообщения придется вычислить k , зная G и $k\ x\ G$. Это трудно вычислимая задача.

Также получатель не знает ключ k , но в качестве подсказки ему посылается $k\ x\ G$. Умножив $k\ x\ G$ на свой закрытый ключ, получатель получит значение, которое было добавлено отправителем к незашифрованному сообщению. Таким образом получатель, не зная k , но имея свой закрытый ключ, может восстановить незашифрованное сообщение [11].

Глава 2 Электронно-цифровая подпись

2.1 Эволюция цифровой подписи

Для решения задач аутентификации, аппелируемости и обеспечения целостности информации в современных информационных системах используется концепция электронной цифровой подписи (ЭЦП). ЭЦП – это набор методов, которые позволяют перенести свойства рукописной подписи под документом в область электронного документооборота. Главной отличительной возможностью цифровой подписи считается возможность копировать ее неоднократное количество раз. из-за этого возникает необходимость решать данную задачу математическими методами [8].

К электронно-цифровым подписям предъявляется ряд требований схожих с ручной подписью:

— Цифровая подпись должна позволять доказать то что документ подписал именно ее автор.

— Цифровая подпись должна быть неделимой с основным документом. Для того чтобы подпись было нельзя использовать для других документов.

— Цифровая подпись должна позволять сохранять неизменность для подписанного документа в том числе и для автора подписи.

— Должно быть невозможно отказаться от подписи. Подписанный документ становится юридическим.

Существует множество вариантов реализации электронно - цифровой подписи , наиболее известными считаются варианты реализаций построенные на алгоритмах: RSA(Rivest, Shamir, Adleman), Эль-Гемаля, DSA(Digital Signature Algorithm), Шнорра, ГОСТ Р.34-10 [6].

В основе алгоритмов цифровой подписи лежит протокол обмена ключами, разработанные в свое время в 1976 г. У. Диффи и М. Хеллманом. В этом алгоритме впервые появляется понятие односторонней функции с секретом: это такая функция, что любой может вычислить $f_k(\omega)$, но только то

лицо которое знает секрет K может (за разумное время) вычислить обратную функцию $f_k^{-1}(x)$.

— Публикуется способ вычисления $f_k(x)$ – открытый ключ.

— Подпись под сообщением – это сложно вычислимая обратная функция, где $S = f_k^{-1}(x)$.

— Проверка – это удостоверение того факта, что $f_k(S) = M$.

С развитием популярности сети интернет и электронной коммерции появилась необходимость доработки протокола обмена ключами, которая вылилась в создание стандарта ЭЦП. Такой реализацией стал алгоритм RSA. В качестве односторонней функции с секретом здесь используется функция возведения в степень по составному модулю. Далее приводится алгоритм RSA на котором будет базироваться цифровая подпись (см. рисунок 2.2) [16]:

— Параметры схемы: p, q – простые числа, $N = pq$, $\varphi(N) = (p-1)(q-1)$, d, e : $ed = 1(mod\varphi(N))$.

— Секретный ключ: p, q, e; открытый ключ: N, d.

— Выработка подписи: сообщение M, $M \in \{1, \dots, N-1\}$, $S = Me\ (mod N)$.

— Проверка подписи: подпись верна $\Leftrightarrow S^d = M\ (mod N)$.

Стоит отметить, что реализация цифровой подписи на основе алгоритма RSA считается одним из первых стандартов электронно-цифровой подписи.

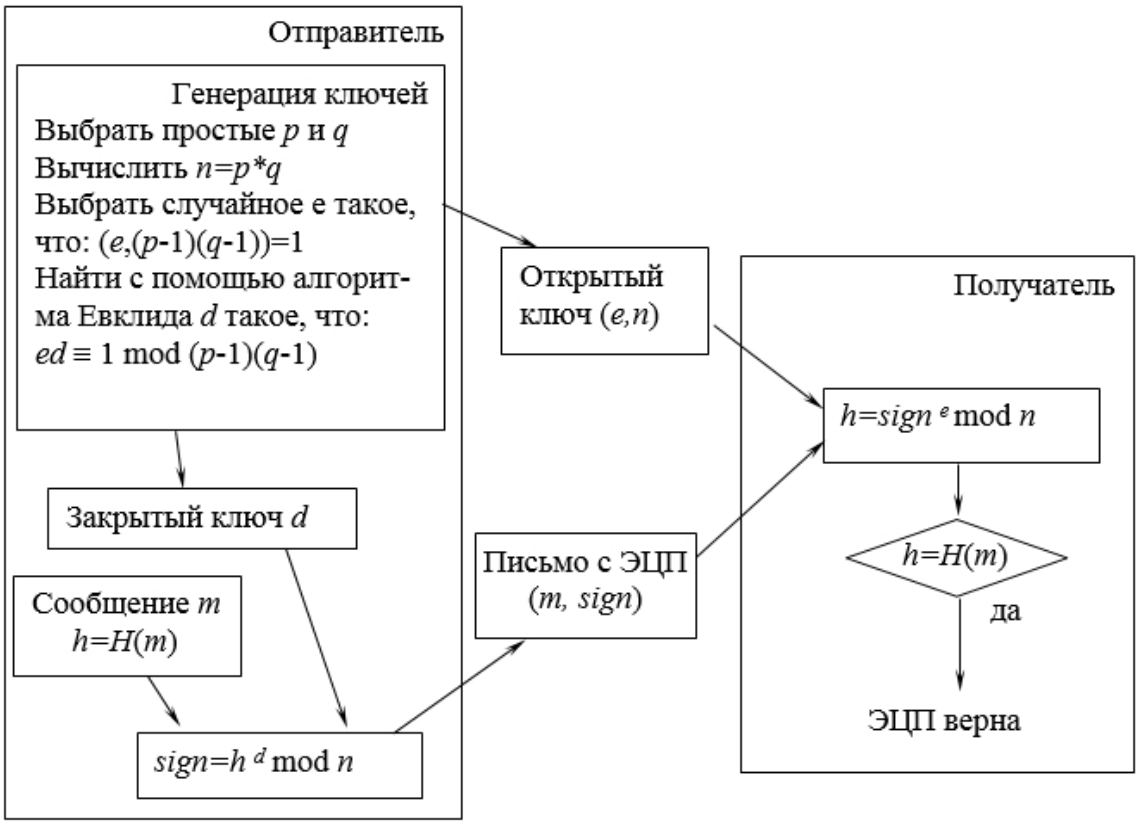

Рисунок 2.2 - Схема формирования ЭЦП на основе алгоритма RSA

Одновременно с реализацией ЭЦП на основе алгоритма RSAбыла предложена реализация цифровой подписи на основе схемы Эль-Гамаля. Но в отличии от алгоритма RSAэта реализация не стала стандартом, так как варьируя параметры схемы Эль-Гамаля можно получить различные её вариации. Схема Эль-Гамаля представлена на рисунке 2.3.

К параметрам схемы относятся: G – циклическая группа простого порядка q, a – образующий элемент группы G, хэш-функция $h: V* \rightarrow \{0, \ldots, q-1\}$, отображение,которое элементы группы переводят в числа $\pi: G\{1G\} \rightarrow \{0, \ldots, q-1\}$.

Формирование подписи происходит следующим образом:

— Выбирается секретный ключ: $d \in R\{1, \ldots, q-1\}$, где d принимает случайные значения от 1 до порядка группы –1.

— Затем выбирается открытый ключ принимающий значения $y = a^a$

— После этого генерируется подпись (r, s): $k \in R\{1, \ldots, q-1\}$.

А именно выбирается одноразовый случайный параметр k , гдепервая компонента подписи равна $r = \pi(a^k)$,далее из уравнения подписи находится вторая компонента $sU = dv + kw$, где u , v , w находится из множества $(s, r, h(M))$, $h(M)$ – хэш-код сообщения.

— Проверка подписи происходит следующим образом: если подпись верна, то выполняется уравнение проверки $\pi\left(a^{uw^{-1}(mod q)} * y^{-ww^{-1}(mod q)}\right) = r$, иначе подпись не верна.

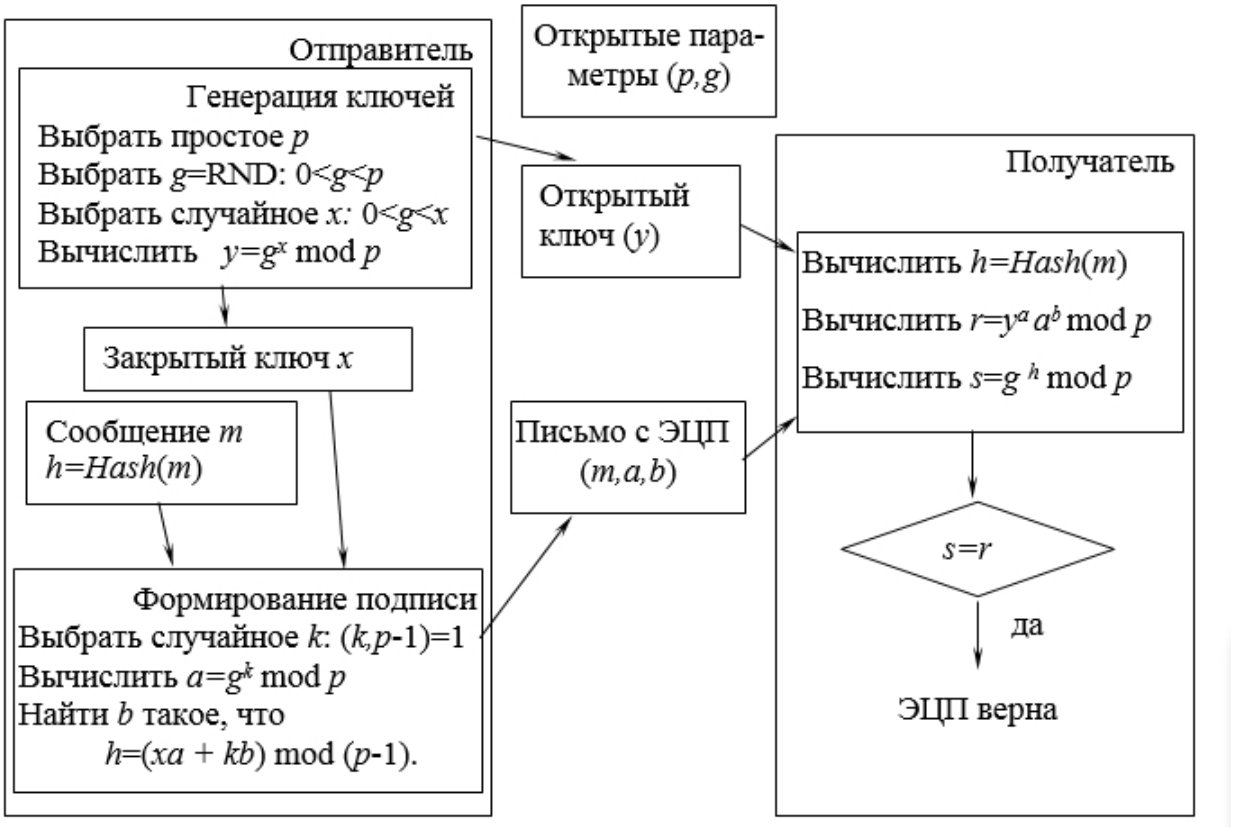

Рисунок 2.3- Схема формирования ЭЦП на основе схемы Эль-Гамаля

Наиболее популярными вариациями схемы Эль-Гамаля являются [12]:

— Американский стандарт DSA представленный на рисунке 2.4 (1991)

— Первая редакция отечественного стандарта цифровой подписи (1994) представленный на рисунке 2.5

Все они реализованы в конечном простом поле, имеют схожие параметры, так как, являются вариантами одной и той же схемы.

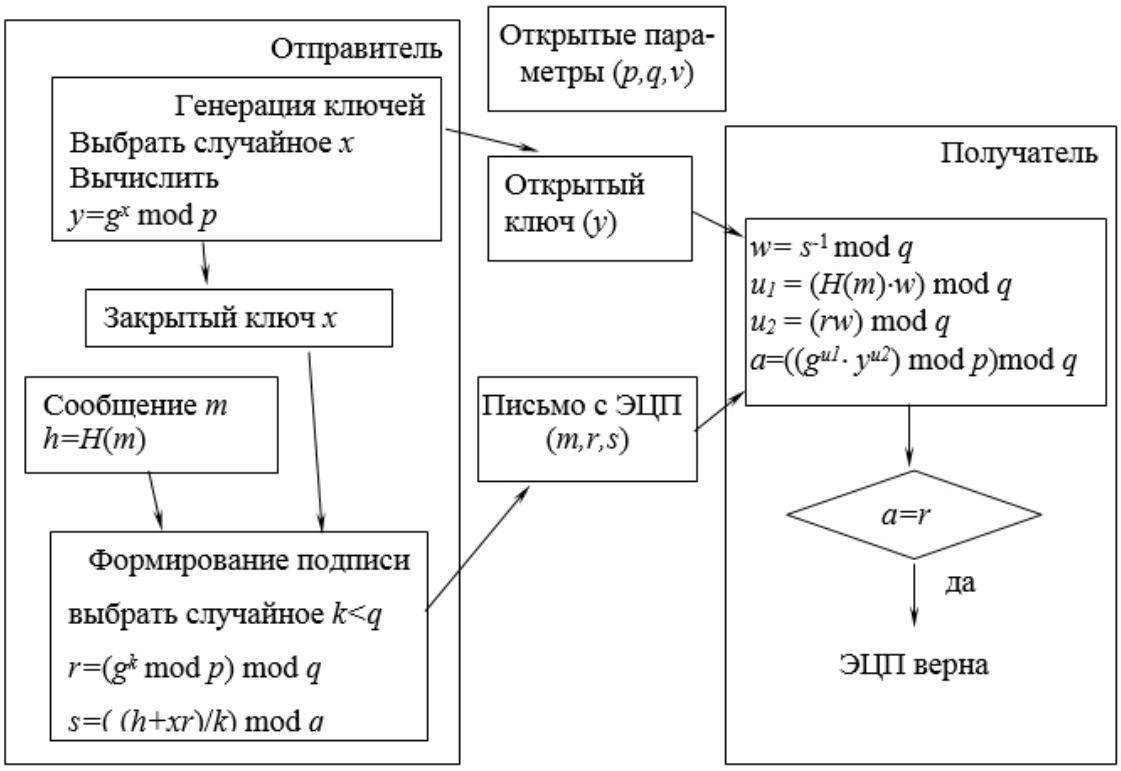

Рисунок 2.4- Схема формирования ЭЦП на основе алгоритма DSA

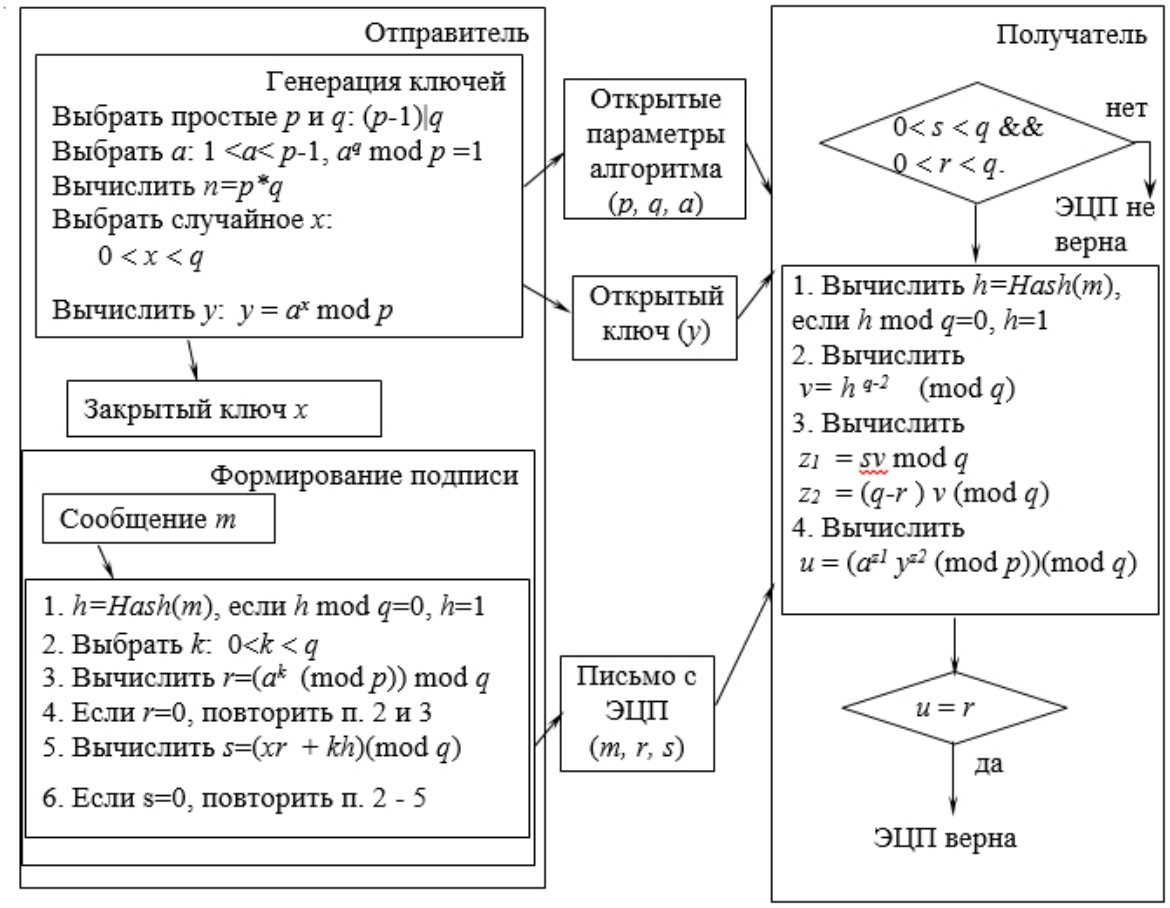

Рисунок 2.5- Схема формирования ЭЦП ГОСТ Р 34.10-94

2.2 Американские стандарты шифрования

С 1998 года использование эллиптических кривых для решения таких криптографических задач, как цифровая подпись, было закреплено в стандартах СШАANSI X9.62 иFIPS186-4. Американский стандарт шифрования FIPS 186-4(4 редакция) является основным конкурентом схемы ГОСТ 34.10 [17]. Данный стандарт базируется на алгоритме DSAи фиксирует следующие схемы подписи:

— RSA (длина ключа 1024, 2048, 3072 бит).

— DSA (длина ключа 1024, 2048, 3072 бит). (Вариант схемы Эль-Гамаля в конечном поле).

— EC-DSA над простыми полями (длина ключа 160, 224, 256, 384, 521 бит).

— EC-DSA над полями характеристики 2 (длина ключа 163, 233, 283, 409, 571 бит).

— EC-DSA на кривых Коблица над полями характеристики 2 (длина ключа 160, 224, 256, 384, 521 бит).

Данный стандарт используется в американских учреждениях совместно с хэш-функциями SHA (в зависимости от длины ключа) и фиксирует 15 эллиптических кривых. В отечественном стандарте ни одной кривой не зафиксировано, но приведены требования, защищающие от всех атак в настоящее время [19].

Стандарт ANSI X9.62, основывается на алгоритме ECDSA (Elliptic Curve Digital Signature Algorithm). Отличительной чертой которого является тот факт, что он определен не над полем целых чисел, а в группе точек эллиптической кривой.

2.3 Российские стандарты шифрования

Первая редакция стандарта была принята в 1994 году (ГОСТ Р 34.10-94) (см. рисунок 2.5) основывалась она на варианте схемы Эль-Гамаля в конечном поле. Хэш-функция ГОСТ Р 34.11-94 формирует 256-битное выходное значение, используя в качестве преобразующей операции блочный шифр ГОСТ 28147-89 представленный на рисунке 2.6.

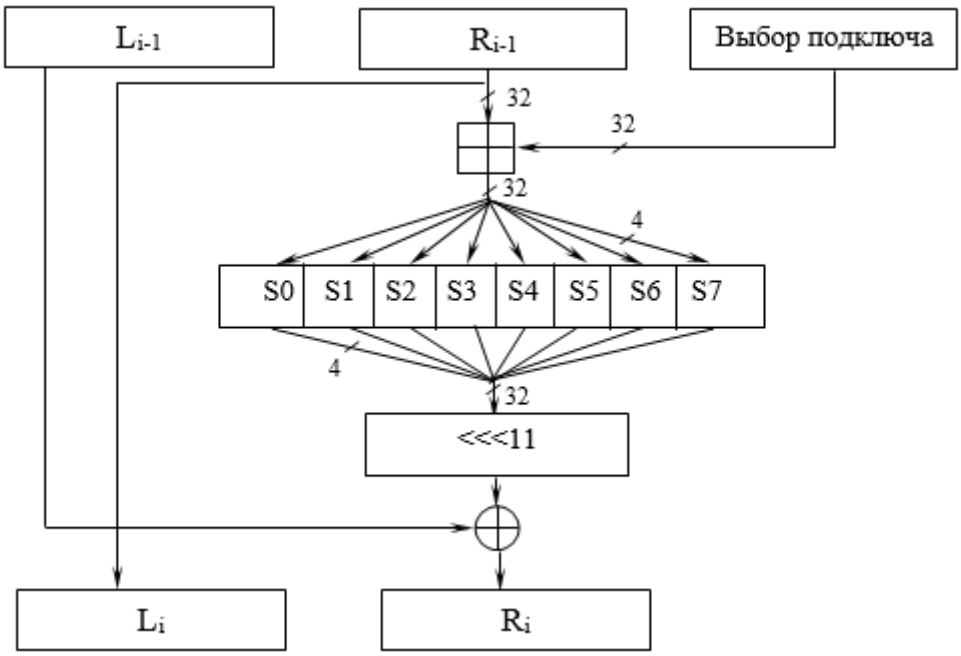

Рисунок 2.6- Образующая функция алгоритма ГОСТ 28147-89

В след за развитием методов логарифмирования в конечном поле стало ясно в том числе и усилиями Российских специалистов, что редакция 1994 года больше не может считаться крипто стойкой. В 2001 году было принято решение перейти на эллиптические кривые с сохранением структуры схемы [1].Длина ключа так же составила 256 бит. Хэш-функция использовалась та же (ГОСТ Р 34.11-94) [4].

В 2010 году наша хэш-функция стала частью международного стандарта цифровой подписи под названием ISO/IEC14888-3:EC-RDSA(Elliptic Curve Russian Digital Signature Algorithm)

В 2012 году была введена третья, действующая в настоящее время редакция пополнила стандарт вариантом требования длины ключа, то есть длина ключа была увеличена в двое (256 или 512 бит), что обеспечило нам еще больший запас по стойкости. Хэш-функция ГОСТ Р 34.11-2012, действует с 01.01.2013 г. [2].

Алгоритм ГОСТ Р 34.10-2012.

Данный алгоритм разработан главным управлением безопасности связи Федерального агентства правительственной связи и информации при

Президенте Российской Федерации при участии Всероссийского научно-исследовательского института стандартизации. Разрабатывался взамен ГОСТ Р 34.10-94 для обеспечения большей стойкости алгоритма[6].

Алгоритму ГОСТ Р 34.10-2012 присуще следующие параметры:

— P – простое число;

— G – подгруппа простого порядка q, группы точек эллиптической кривой $y^2 = x^3 + ax + b (mod p)$, где: или – это дает нам защиту от общих методов логарифмирования;

— $P \in G$,;

— Функция хеширования ГОСТ Р 34.11-2012(“Стрибог”) с длиной хэш-кода 256, если , и 512, если .

Как мы видим сделан достаточно большой запас по величине параметра, учитывая развитие методов логарифмирования в конечном поле.

Выработка ключа происходит следующим образом. Секретный ключ выбирается случайно, так же по-другому он называется ключом выработки подписи$d \in R \{1, \dots, q-1\}$. Открытый ключ или ключ проверки подписи представляет собой кратную точку $Q = dP$, то есть d раз сложенная сама с собой точкаP .

Уравнение подписи имеет вид $s = dr + k_h(M) (mod q)$, где $r = X(kp)$, d –секретный ключ, k – одноразовое секретное значение, $h(M)$– хэш-функция.

Далее выпишем уравнение проверки $x(s_h(M) mod q)P - (rh(M) mod q)Q$, где $(mod q) = r$. То есть, как и во всех вариантах схемы Эль-Гамаля у нас две компоненты подписи r и s .

Пошаговая выработка подписи состоит из следующих этапов[20]и представлена на рисунке 2.7:

1. Вычисляется хэш-функция $h = h(M)$.
2. Вычисляется$e = h (mod q)$, если $e = 0$, то полагаем.

3. Генерируется секретный ключ $k \subset R \{1, \ldots, q-1\}$.
4. Вычисляется точка эллиптической кривой $C = kP$, полагаем, что $r = xc(\mathit{mod} q)$. Если $r = 0$, то переходим к 3 шагу.
5. Вычисляем параметр $s = rd + ke(\mathit{mod} q)$. Если $s = 0$, то переходим к шагу 3.
6. Подпись является $r \| s$.

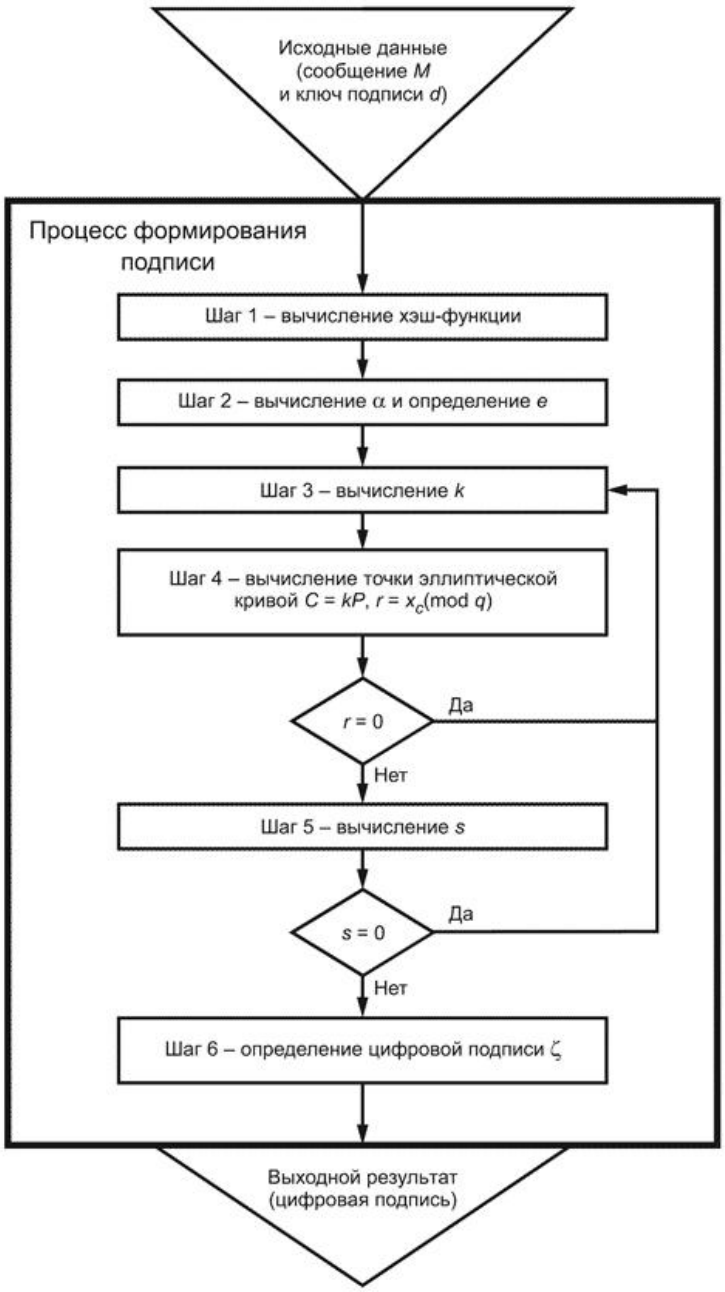

Рисунок 2.7 - Схема процесса формирования цифровой подписи

Проверка подписи происходит по следующему алгоритму и представлена на рисунке 2.8:

1. Извлекаем из подписи параметры r , s . Проверяем, что $0 < r$, $s < q$.
2. Вычисляем хэш-функцию $\hbar = h(M)$.
3. Вычисляем $e = \hbar \pmod q$, если $e = 0$, то полагаем $e = 1$.
4. Вычисляем $v = e^{-1} \pmod q$.
5. Вычисляем $z_1 = sv \pmod q$, $z_2 = -rv \pmod q$.
6. Вычисляем точку $C = z_1 P + z_2 Q$, полагаем $R = x_C \pmod q$.
7. Если $R = r \pmod q$, то подпись принимаем.

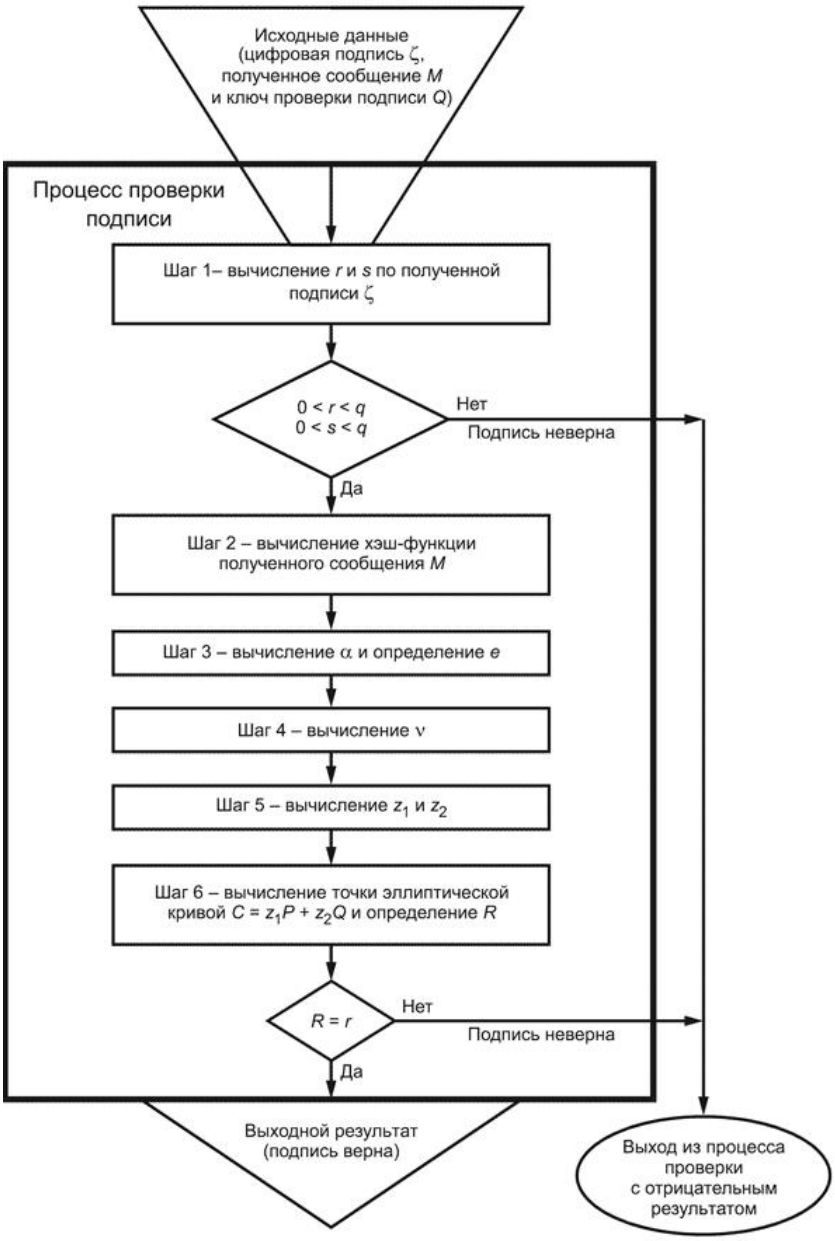

Рисунок 2.8 - Схема процесса проверки цифровой подписи

2.4 Теоретическая стойкость схемы ЭЦП

Оценка стойкости схемы ЭЦП происходит по следующим направлениям:

- Обоснование стойкости криптосхемы: сведение задачи ее компрометации к решению сложной математической задачи.
- Асимптотическая оценка трудоемкости решения: функция от размера задачи (например, длины ключа).
- Определение параметров криптосхемы (например, длины ключа), гарантирующих ее стойкость относительно известных методов анализа в течение заданного срока.

В схеме RSAстойкость основана на том, что задача факторизации является вычислительно сложной–по $N = pq$ определить p ,q .

Для обобщенной схемы Эль-Гамаля, мы рассматриваем задачу дискретного логарифмирования в группе $G = \{g\}$ по $y \in G$ найти x такое, что $g^x = y$.

Существуют несколько формальных моделей стойкости, которые позволяют доказуемо свести задачу вскрытия крипто схемы к некоторой сложной задаче.

- Модель со случайным оракулом, где хэш-функция заменяется на некоторую случайную функцию.
- Модель генетической группы, где конкретная группа заменяется идеальным объектом, в котором существуют только общие методы логарифмирования.
- Модель с защищенным модулем, где вместо хэш-функции используется некоторый идеальный блочный шифр, полностью защищенный от противника.

Д. Браун доказал, что схема DSA является стойкой в модели генерической группы. Н.П. Варновский доказал стойкость отечественной схемы в модели с защищенным модулем.

Практическая стойкость при реализации.

Противник может пытаться подделать подпись либо под произвольным сообщением (экзистенциальная подделка), либо под заданным сообщением[18].

Действия, которые может предпринять противник:

- Найти сообщение M, такое что хэш-код $h(M) = h(M_0)$ для некоторого сообщения подпись которого известна (задача построения второго прообраза хэш-функции).
- Найти сообщения M, , такие, что подпись M_0 известна или может быть получена от владельца секретного ключа, и при этом $h(M) = h(M_0)$ (задача построения коллизии хэш-функции).
- Противник может определить секретный ключ по открытому ключу и параметрам схемы.

Вопросы, которые возникают при практической реализации отечественной схемы связанны во многом с выбором одноразового случайного значения k. Как прописано в тексте стандарта оно должно выбираться случайно и уничтожаться после использования. Если разработчик этого не сделает, то k будет скомпрометировано и будет известно противнику. Следовательно, в уравнении подписи будут известны все значения кроме секретного ключа. Получаем линейное уравнение $s = rd + kh(M) (mod\ q)$, можем найти секретный ключ d. Если мы используем k два раза, следовательно, имеем два линейных уравнений $s = rd + kh(M1) (mod\ q)$ и $s = rd + kh(M2) mod\ q)$, так же находим секретный ключ d и k. При реализации стоит обращать на это большое внимание [10].

Существуют рекомендации технического комитета, состоящие из двух документов, которые задают ряд параметров. В первом документе описаны кривые в классическом виде то, что называется формой Вейерштрассе для вариантов 256 и 512 бит. Во втором документе описаны скрученные кривые в форме Эдвардса [3] –это альтернативная форма задания кривой, которая обладает практическими преимуществами. Принцип выработки параметров – это доказуемая, псевдослучайность [6].Параметры кривой получаются как результат некоторого одностороннего преобразования. И доказательством того, что они действительно выбраны псевдослучайно предъявляем прообраз того преобразования.

Глава 3 Разработка схем ЭЦП ГОСТ Р 34.10

3.1 Практическая реализация алгоритмов ЭЦП на языке программирования Python

Для реализации алгоритмов был выбран язык программирования Python. Алгоритмы библиотеки pygost языка программирования Pythonоснованы на эллиптических кривых. Реализация каждого из стандартов имеет свои тонкости, о которых кратко рассказывается в этом разделе [15].

В тексте стандарта прописано все необходимое и приведены наглядные примеры. Алгоритм работает с группой точек эллиптической кривой над полем большого простого числа p. Не лишним будет напомнить, что эллиптическая кривая над конечным простым полем – это набор точек, описывающихся уравнением Вейерштрассе:

$$y^2 = x^3 + ax + b \quad (7)$$

Соответственно при использовании алгоритма, во-первых, необходимо определиться с параметрами эллиптической кривой, а именно выбрать числа a, b, n и базовую точку P, с порядком равным большому простому числу q. Это означает, что если умножать точку на числа меньшие, чем q, то каждый раз будут получаться совершенно различные точки. После выбора параметров можно приступать к генерации пары секретный-публичный ключ.

Секретный ключ d – случайное большое число, удовлетворяющее неравенству $0 < d < q$.Публичный ключ – точка эллиптической кривой Q вычисляемая как $Q = d * P$.

Процесс формирования ЭЦП выполняется по следующему алгоритму:

1. Вычисляем хеш-значения сообщения M: $H = h(M)$.

2. Вычисляем целое число α , двоичным представлением которого

является H .

3. Определяем$e = \alpha \bmod q$, если $e = 0$, задаем$e = 1$.
4. Генерируем случайное число k , удовлетворяющее условию .
5. Вычисляем точку эллиптической кривой $C = k * P$.
6. Определяем$r = xC \bmod q$, где -координата точки C . Если $r = 0$, то возвращаемся к шагу 4.
7. Вычисляем значение $s = (rd + ke) \bmod q$. Если $s = 0$, то возвращаемся к шагу 4.
8. Возвращаем значение $r||s$ в качестве цифровой подписи.

Для проверки подписи нужно выполнить следующие шаги:

1. По полученной подписи восстанавливаем числа r и s . Если не выполняются неравенства $0 < r < q$ и $0 < s < q$, тогда возвращаем сообщение «подпись не верна».
2. Вычисляем хеш-значения сообщения M: $H = h(M)$.
3. Вычисляем целое число α , двоичным представлением которого является H .
4. Определяем$e = \alpha \bmod q$, если $e = 0$, задаем$e = 1$.
5. Вычисляем$v = e - 1 \bmod q$.
6. Вычисляем значения $z_1 = s * v \bmod q$ и $z_2 = -r * v \bmod q$.
7. Вычисляем точку эллиптической кривой $C = z_1 * G + z_2 * Q$.
8. Определяем$R = xc \bmod q$.
9. Если $R = r$, то подпись принимается. В противном случае подпись не принимается.

Для проверки алгоритма можно воспользоваться параметрами из текста стандарта[2] (см рисунок 3.1).

```
def test_gost_sign():
        p = 57896044618658097711785492504343953926634992332820282019728792003956564821041
        a = 7
        b = 43308876546767276905765904595650931995942111794451039583252968842033849580414
        x = 2
        y = 40189740565390375033354494229370597756357393899055450806909793652134315662 80
        q = 57896044618658097711785492504343953927082934583725450622380973592137631069619
        gost = DSGOST(p, a, b, q, x, y)
        key = 55441196065363246126355624130324183196576709222340016572108097750006097525544
        message = 20798893674476452017134061561508270130637142515379653289952617252661468872421
        k = 53854137677348463731403841147996619241504003434302020712960838528893196233395
        sign = gost.sign(message, key, k)

def test_gost_verify():
        p = 57896044618658097711785492504343953926634992332820282019728792003956564821041
        a = 7
        b = 43308876546767276905765904595650931995942111794451039583252968842033849580414
        x = 2
        y = 40189740565390375033354494229370597756357393899055450806909793652134315662 80
        q = 57896044618658097711785492504343953927082934583725450622380973592137631069619
        gost = DSGOST(p, a, b, q, x, y)
        message = 20798893674476452017134061561508270130637142515379653289952617252661468872421
        sign = (29700980915817952874371204983938256990422752107994319651632687982059210933395,
                    57497340027008465417892531001914703845522704264909856393371899917551583955 2)
        q_x = 57520216126176808443631405023338071176630104906313632182896741342206604859403
        q_y = 17614944419213781543809391949654080031942662045363639260709847859438286763994
        public_key = ECPoint(q_x, q_y, a, b, p)
        is_signed = gost.verify(message, sign, public_key)
```

Рисунок 3.1- Пример использования ГОСТ Р 34.10

3.2 Тестирование программного продукта.

Для тестирования своего программного продукта мною был выбран инструмент Unittest.Это стандартный модуль для написания юнит-тестов на Python. Unittest это порт JUnit с Java. Иными словами, и в коде модуля, и при написании тестов легко прослеживается объектно-ориентированный стиль программирования, что весьма удобно для тестирования процедур и классов [15].

Отличительными особенностями данного инструмента являются:

- возможность автоматизированного тестирования;
- возможность собирать тесты в группы;

- возможность собирать результаты выполнения тестов (например, для отчета);
- за счет объектно-ориентированного стиля программирования уменьшается дублирование кода при схожих объектах тестирования;

В использовании unittest присутствуют несколько концепций

— test case;

— test suite;

— test fixture;

— test runner;

Тестовый случай создаётся путём наследования от unittest.TestCase. 3 отдельных теста определяются с помощью методов, имя которых начинается на test. Это соглашение говорит исполнителю тестов о том, какие методы являются тестами.

Суть каждого теста состоит в следующем:

- вызов функции assertEqual() для проверки ожидаемого результата;
- функций assertTrue() или assertFalse() для проверки условия;
- функции assertRaises() для проверки, что метод порождает исключение;

Главным достоинством этих методов является легкость в оформлении отчетов по полученным результатам в отличие от использования обычного метода assert.

Методы setUp() и tearDown(), которые в данном простом случае не нужны, позволяют определять инструкции, выполняемые перед и после каждого теста, соответственно.

Процесс запуска тестов достаточно прост. Функция unittest.main() предоставляет собой интерфейс командной строки для тестирования программы. Будучи запущенным из командной строки, этот скрипт выводит отчёт, подобный этому (см рисунок 3.2):

```
Ran 3 tests in 0.000s
OK
```

Рисунок 3.2 - Пример отчета

Unittest может быть использован из командной строки для запуска модулей с тестами, классов или даже отдельных методов:

— python -m unittest test_module1 test_module2

— python -m unittest test_module.TestClass

— python -m unittest test_module.TestClass.test_method

Можно также указывать путь к файлу:

— python -m unittest tests/test_something.py

С помощью флага -v можно получить более детальный отчёт:

— python -m unittest -v test_module

Кроме этого флага существует еще несколько дополнительных параметров, призванных помочь в формировании отчетов:

-b (--buffer) - вывод программы при провале теста будет показан, а не скрыт, как обычно.

-c (--catch) - Ctrl+C во время выполнения теста ожидает завершения текущего теста и затем сообщает результаты на данный момент. Второе нажатие Ctrl+C вызывает обычное исключение KeyboardInterrupt.

-f (--failfast) - выход после первого же неудачного теста.

--locals (начиная с Python 3.5) - показывать локальные переменные для провалившихся тестов.

Для нашего примера подробный отчёт представлен на рисунке 3.3

```
test_isupper (__main__.TestStringMethods) ... ok
test_split (__main__.TestStringMethods) ... ok
test_upper (__main__.TestStringMethods) ... ok
Ran 3 tests in 0.001s
OK
```

Рисунок 3.3. - Подробный отчет в результате теста

Unittest поддерживает простое обнаружение тестов. Для совместимости с обнаружением тестов, все файлы тестов должны быть модулями или пакетами, импортируемыми из директории верхнего уровня проекта.

Обнаружение тестов реализовано в функции TestLoader.discover(), но может быть использовано и из командной строки:

cd project_directory

python -m unittest discover

К параметрам обнаружения тестов можно отнести:

-v (--verbose) - подробный вывод.

-s (--start-directory) directory_name - директория начала обнаружения тестов (текущая по умолчанию).

-p (--pattern) pattern - шаблон названия файлов с тестами (по умолчанию test*.py).

-t (--top-level-directory) directory_name - директория верхнего уровня проекта (по умолчанию равна start-directory).

Отметим, что для демонстрации простых случаев, которые должны быть проверены на корректность разрабатываются базовые блоки тестирования.

Тестовый случай создаётся путём наследования от подкласса unittest.TestCase.

Тестирующий код должен быть самостоятельным, то есть никак не зависеть от других тестов.

Простейший подкласс TestCase может просто реализовывать тестовый метод (метод, начинающийся с test) (см. рисунок 3.4).

```
import unittest

class DefaultWidgetSizeTestCase(unittest.TestCase):
    def test_default_widget_size(self):
        widget = Widget('The widget')
        self.assertEqual(widget.size(), (50, 50))
```

Рисунок 3.4 - Простейший подкласс TestCase

Заметьте, что для того, чтобы проверить что-то, мы используем один из

assert методов.

Тестов может быть много, и часть кода настройки может повторяться. К счастью, мы можем определить код настройки путём реализации метода setUp(), (см рисунок 3.5)который будет запускаться перед каждым тестом (см рисунок 3.6):

```
class NumbersTest(unittest.TestCase):

    def test_even(self):
        """
        Test that numbers between 0 and 5 are all even.
        """
        for i in range(0, 6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 0)
```

Рисунок 3.5 - Реализация метода setup()

```
import unittest

class SimpleWidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')

    def test_default_widget_size(self):
        self.assertEqual(self.widget.size(), (50,50),
                         'incorrect default size')

    def test_widget_resize(self):
        self.widget.resize(100,150)
        self.assertEqual(self.widget.size(), (100,150),
                         'wrong size after resize')
```

Рисунок 3.6 - Пример теста

Так же можно определить метод tearDown(), который будет запускаться после каждого теста (см рисунок 3.7):

```
import unittest

class SimpleWidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')

    def tearDown(self):
        self.widget.dispose()
```

Рисунок 3.7 – Определение метода tearDown()

Можно разместить все тесты в том же файле, что и сама программа, таком как widgets.py, заметим, что размещение тестов в отдельном файле, таком как test_widget.py имеет много преимуществ:

—Модуль с тестом может быть запущен автономно из командной строки.

—Тестовый код может быть легко отделён от программы.

—Ограничить возможность изменять тестов для соответствия коду программы без видимой причины.

—Тестовый код должен изменяться гораздо реже, чем программа.

—Протестированный код может быть легче переработан.

—В случае изменения стратегии тестирования нет необходимости изменять код программы.

Unittest поддерживает пропуск отдельных тестов, а также классов тестов.

Пропуск теста осуществляется использованием декоратора skip() или одного из его условных вариантов (см рисунок 3.8).

```
class MyTestCase(unittest.TestCase):

    @unittest.skip("demonstrating skipping")
    def test_nothing(self):
        self.fail("shouldn't happen")

    @unittest.skipIf(mylib.__version__ < (1, 3),
                     "not supported in this library version")
    def test_format(self):
        # Tests that work for only a certain version of the library.
        pass

    @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
    def test_windows_support(self):
        # windows specific testing code
        pass
test_format (__main__.MyTestCase) ... skipped 'not supported in this library version'
test_nothing (__main__.MyTestCase) ... skipped 'demonstrating skipping'
test_windows_support (__main__.MyTestCase) ... skipped 'requires Windows'
```

Рисунок 3.8 - Пример использования декоратора skip()

Ожидаемые ошибки используют декоратор expectedFailure() (см рисунок 3.9):

```
class ExpectedFailureTestCase(unittest.TestCase):
    @unittest.expectedFailure
    def test_fail(self):
        self.assertEqual(1, 0, "broken")
```

Рисунок 3.9 - Использование декоратора expectedFailure()

Очень просто сделать свой декоратор. Например, следующий декоратор пропускает тест, если переданный объект не имеет указанного атрибута (см рисунок 3.10):

```
def skipUnlessHasattr(obj, attr):
    if hasattr(obj, attr):
        return lambda func: func
    return unittest.skip("{!r} doesn't have {!r}".format(obj, attr))
```

Рисунок 3.10 - Пример своего декоратора

Примерами декораторов, пропускающих тесты или говорящих об ожидаемых ошибках являются:

- @unittest.skip(reason) - пропустить тест. reason описывает причину пропуска.
- @unittest.skipIf(condition, reason) - пропустить тест, если condition истинно.
- @unittest.skipUnless(condition, reason) - пропустить тест, если condition ложно.
- @unittest.expectedFailure - пометить тест как ожидаемая ошибка.

Для пропущенных тестов не запускаются команды setUp() и tearDown().

Для пропущенных классов не запускаются команды setUpClass() и tearDownClass().

Для пропущенных модулей не запускаются команды setUpModule() и tearDownModule().

Когда некоторые тесты имеют лишь незначительные отличия, например некоторые параметры, unittest позволяет различать их внутри одного тестового метода, используя менеджер контекста subTest().

Проверку на успешность по средствам модуля технологии Unittestможно представить с помощью множества функций для самых различных проверок (см. рисунок 3.11):

```
assertEqual(a, b) – a == b
assertNotEqual(a, b) – a != b
assertTrue(x) – bool(x) is True
assertFalse(x) – bool(x) is False
assertIs(a, b) – a is b
assertIsNot(a, b) – a is not b
assertIsNone(x) – x is None
assertIsNotNone(x) – x is not None
assertIn(a, b) – a in b
assertNotIn(a, b) – a not in b
assertIsInstance(a, b) – isinstance(a, b)
assertNotIsInstance(a, b) – not isinstance(a, b)
assertRaises(exc, fun, *args, **kwds) – fun(*args, **kwds) порождает исключение exc
assertRaisesRegex(exc, r, fun, *args, **kwds) – fun(*args, **kwds) порождает исключение exc и сообщение соответствует регулярному выражению r
assertWarns(warn, fun, *args, **kwds) – fun(*args, **kwds) порождает предупреждение
assertWarnsRegex(warn, r, fun, *args, **kwds) – fun(*args, **kwds) порождает предупреждение и сообщение соответствует регулярному выражению r
assertAlmostEqual(a, b) – round(a-b, 7) == 0
assertNotAlmostEqual(a, b) – round(a-b, 7) != 0
assertGreater(a, b) – a > b
assertGreaterEqual(a, b) – a >= b
assertLess(a, b) – a < b
assertLessEqual(a, b) – a <= b
assertRegex(s, r) – r.search(s)
assertNotRegex(s, r) – not r.search(s)
assertCountEqual(a, b) – a и b содержат те же элементы в одинаковых количествах, но порядок не важен
```

Рисунок 3.11 - Функции для проверок

Для сравнения времени выполнения алгоритмов ГОСТ Р 34.10-2012 и ГОСТ Р 34.10-2001 был построен график. В качестве исходных данных использовались результаты тестирования выполнения алгоритмов. Алгоритмы запускались в один поток по одному процессу. Количество запусков алгоритма увеличивалось с каждым шагом по 100. По результатам тестирования была сформирована таблица (см. таблица 3.1).

По таблице 3.1 средствами openoffice.org Calc был сформирован график результатов (см рисунок 3.12).

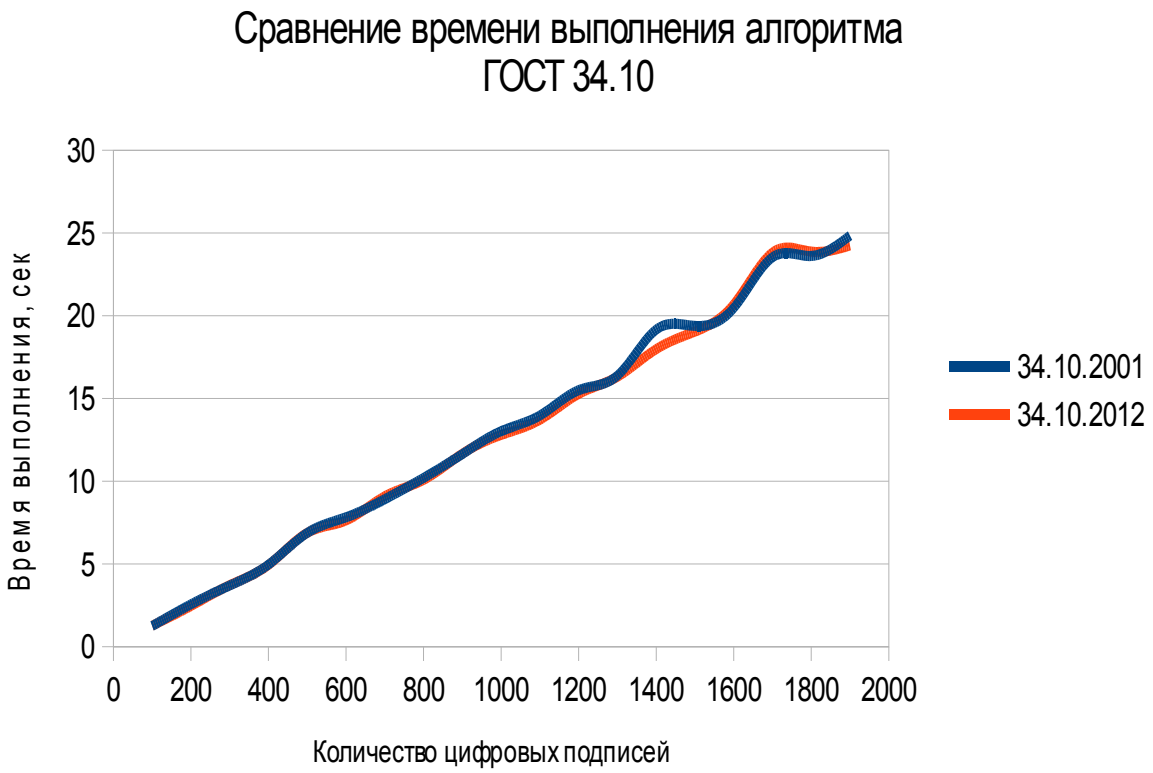

Рисунок 3.12 -График времени выполнения алгоритмов ГОСТ 34.10

По результатам тестирования очевидна линейная зависимость времени выполнения от количества запусков алгоритма. С увеличением количества запусков растёт время выполнения. Так же можно заметить то, что время выполнения алгоритма ГОСТ Р 34.10.2001 приблизительно равно времени выполнения алгоритма ГОСТ Р 34.10.2012. Это связано в первую очередь со скоростью выполнения байт-кода Python. Общеизвестно что Python один из самых используемых языков программирования. Реализация схемы ГОСТ 34.10 на Python дает возможность проводить исследования, такие как: стабильность, эффективность, надёжность и т. д.

Таблица 3.1- Сравнение алгоритмов электронно-цифровой подписи

Количество цифровых подписей	ГОСТ Р 34.10.2001(сек)	ГОСТ Р 34.10.2012(сек)
100	1,25	1,27
200	2,56	2,46
300	3,7	3,72
400	4,96	4,94
500	6,88	6,88
600	7,82	7,63
700	8,91	9,09
800	10,2	10,08
900	11,64	11,66
1000	13,02	12,8
1100	13,97	13,71
1200	15,5	15,27
1300	16,38	16,31
1400	19,16	17,98
1500	19,38	19,04
1600	20,46	20,69
1700	23,53	23,82
1800	23,61	23,87
1900	24,84	24,25

3.3 Производительность и эффективность схемы ЭЦП

Если рассмотреть типовой современный сервер (16 ядер, Intel, частота 3ГГц) можно считать, что для максимального значения 512 бит мы можем вырабатывать порядка 3500 подписей в секунду и порядка 2000 проверять.

На практике процедура схема выработки оказывается в 40-120 раз быстрее, чем схема RSAпри том же уровне стойкости.

В устройствах с ограниченными ресурсами отечественная схема лучше ложится на маленькие устройства по трудоемкости, но недостатком является необходимость использования хорошего датчика случайных чисел который не всегда можно взять в сим карте.

Есть два резерва для ускорения данной схемы. Это использование эллиптических кривых в форме Эдвардса ТК 26, 2013; и оптимизация алгоритмов вычисления кратных точек и модулярной арифметики modp. У данной схемы есть перспективы развития и улучшения производительность на 25% и больше.

Заключение

В данной работе изложены базовые понятия теории эллиптических кривых, необходимые для реализации криптографических протоколов. Рассмотрены алгоритмы шифрования RSA, DSA, схема Эль-Гамаля и алгоритмы по созданию электронно-цифровой подписи с использованием эллиптических кривых. Результатом данной работы стали примеры реализации схемы ЭЦП ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 на языке Python. Данные алгоритмы реализованы достаточно крипто стойко с простой процедурой шифрования и дешифрования.

На основании проделанной работы можно выделить основные достоинства эллиптической криптографии: в криптографии основанной на эллиптических кривых длина ключа гораздо меньшая по сравнению с другими алгоритмами асимметричной криптографии. Эллиптические алгоритмы работают гораздо быстрее чем классические. Это можно объяснить размерами ключа и применением структуры бинарного конечного поля. Из-за маленькой длины ключа, а также высокой скорости работы, алгоритмы на эллиптических кривых могут использоваться в сим-картах и других устройствах с ограниченными вычислительными ресурсами.В результате анализа алгоритмов решения задачи дискретного логарифмирования было выяснено, что взломать алгоритм шифрования на основе эллиптических кривых довольно сложно, если подобраны правильные параметры. К недостаткам можно отнести проблему выбора подходящей эллиптической кривой и проблему, связанную с генерацией ключей.

В ходе тестирования была выявлена линейная зависимость времени от количества запусков алгоритмов. Время выполнения алгоритма ГОСТ Р 34.10.2001 приблизительно равно времени выполнения алгоритма ГОСТ Р 34.10.2012. Это связано в первую очередь с линейным запуском алгоритмов.

В схемах, основанных на эллиптических кривых можно достичь желаемого уровня безопасности при длине ключа гораздо меньшей чем в схеме RSA.

В заключение отмечу, что помимо общих алгоритмов арифметики эллиптических кривых, существует и много других алгоритмов, разработанных для кривых специального вида, таких как, кривые Эдвардса ($eu^2 + v^2 = 1 + du^2v^2$), которые позволяют добиться еще большей эффективности [3].

Список использованной литературы

1. ГОСТ Р 34.10—2001 Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи.
2. ГОСТ Р 34.10 — 2012 Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи.
3. Алексеев, Е.К. О перспективах использования скрученных эллиптических кривых Эдвардса со стандартом ГОСТ Р 34.10-2012 и алгоритмом ключевого обмена на его основе./ Е.К. Алексеев, И.Б. Ошкин, В.О. Попов, С.В. Смышляев, Л.А. Сонина— Материалы XVI международной конференции "РусКрипто'2014".— 48с.
4. Байдицкая, В.К. Исследование алгоритма эллиптического шифрования. / В.К. Байдицкая —Лучшая студенческая статья 2016: сборник статей Международного научно-практического конкурса Пенза: МЦНС "Наука и просвещение", 2016. — 19-28с.
5. Блинов, А.М. Информационная безопасность: Учебное пособие. Часть 1. /А.М. Блинов — СПб.: Изд-во СПб ГУЭФ, 2010. — 96 с
6. Болотов, А.А. Элементарное введение в эллиптическую криптографию. Алгебраические и алгоритмические основы. / А.А. Болотов, С.Б. Гашков, А.Б. Фролов — М.: КомКнига, 2006. —328с. URL. http://edu-lib.net/matematika-2/dlya-studentov/bolotov-a-a-i-dr-elementarnoe-vvedenie-v-ellipticheskuyu-kriptografiyu-algebraicheskie-i-algoritmicheskie-osnovyi-onlayn
7. Болотов, А.А. Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых. / А.А. Болотов, С.Б. Гашков, А.Б. Фролов — М.: КомКнига, 2006.—280с. URL. http://edu-lib.net/matematika-2/dlya-studentov/bolotov-a-a-i-dr-elementarnoe-vvedenie-v-ellipticheskuyu-kriptografiyu-protokolyi-kriptografii-na-ellipticheskih-krivyih-onlayn

8. Варновский, Н.П. Стойкость схем электронной подписи в модели с защищенным модулем. Дискретная математика./ Н.П. Варнавский,2008. — т. 20, вып. 3, 147-159с..
9. Василенко, О.Н. О вычислении кратных точек на эллиптических кривых над конечными полями с использованием нескольких оснований систем счисления и новых видов координат. Матем. вопр. Криптогр. / О.Н. Василенко, 2011. — 28с.
10. Гребнев, С.В. Современные алгоритмы вычисления кратной точки и суммы кратных точек эллиптической кривой над конечным простым полем и их приложение к реализации схемы электронной цифровой подписи ГОСТ Р 34.10. / С.В. Гребнев, Д.М. Дыгин.—Материалы XIV международной конференции "РусКрипто'2012".—52с.
11. Жданов, О. Н. Применение эллиптических кривых в криптографии: учебное пособие / О. Н. Жданов, Т. А. Чалкин. — Красноярск: СибГАУ, 2011.— 65 с.
12. Казарин, О. В. Протоколы интерактивной идентификации, основанные на схеме электронной подписи ГОСТ Р 34.10–2012. Вопросы защиты информации. / О.В. Казарин, А.Д. Сорокин, 2014. — № 2(105). 43 — 50с.
13. Рубцова, Р. Г. Математические основы защиты информации: учебное пособие / Р. Г. Рубцова, Ш. Т. Ишмухаметов. — Казань: Казанский федеральный университет, 2012.— 138с.
14. Цирлов, В.Л. Основы информационной безопасности автоматизированных систем. Краткий курс. / В. Л. Цирлов.— Феникс 2008. — 87с.
15. Саммерфилд, М. Программирование на Python 3. Подробное руководство./ М. Саммерфилд.— Символ плюс, 2009—609с.
16. Bernstein D.J. et al. Factoring RSA keys from certified smart cards: Coppersmith in the wild. / D.J. Bernstein—2013.—http://eprint.iacr.org/2013/599.

17. Dygin D. Efficient implementation of the GOST R 34.10 digital signature scheme using modern approaches to elliptic curve scalar multiplication. / D. Dygin —2013.
18. Joux A. A new index calculus algorithm with complexity L (1/4) in small characteristic. /A. Joux —2013. —http://eprint.iacr.org/2013/095.
19. Kleinjung T. et al. Factorization of a 768-bit RSA modulus. / T. Kleinjung —2010. —http://eprint.iacr.org/2010/006.
20. Lenstra A.K.et al. Ron was wrong, Whit is right. / A. Lenstra—2012. — http://eprint.iacr.org/2012/064.

Приложение А

Реализация алгоритма электронно-цифровой подписи ГОСТ Р 34-10.2001

```
fromosimporturandom

frompygost.gost3410 import CURVE_PARAMS
from pygost.gost3410 import GOST3410Curve
from pygost.gost3410 import kek
from pygost.gost3410 import public_key
from pygost.gost3410 import sign
from pygost.gost3410 import SIZE_341001
from pygost.gost3410 import SIZE_341012
from pygost.gost3410 import verify
from pygost.utils import bytes2long
from pygost.utils import long2bytes


# 34.10.01

private_key = bytes(bytearray((
        0x7A, 0x92, 0x9A, 0xDE, 0x78, 0x9B, 0xB9, 0xBE,
        0x10, 0xED, 0x35, 0x9D, 0xD3, 0x9A, 0x72, 0xC1,
        0x1B, 0x60, 0x96, 0x1F, 0x49, 0x39, 0x7E, 0xEE,
        0x1D, 0x19, 0xCE, 0x98, 0x91, 0xEC, 0x3B, 0x28
)))
public_key_x = bytes(bytearray((
        0x7F, 0x2B, 0x49, 0xE2, 0x70, 0xDB, 0x6D, 0x90,
        0xD8, 0x59, 0x5B, 0xEC, 0x45, 0x8B, 0x50, 0xC5,
        0x85, 0x85, 0xBA, 0x1D, 0x4E, 0x9B, 0x78, 0x8F,
```

```
        0x66, 0x89, 0xDB, 0xD8, 0xE5, 0x6F, 0xD8, 0x0B
)))
public_key_y = bytes(bytearray((
        0x26, 0xF1, 0xB4, 0x89, 0xD6, 0x70, 0x1D, 0xD1,
        0x85, 0xC8, 0x41, 0x3A, 0x97, 0x7B, 0x3C, 0xBB,
        0xAF, 0x64, 0xD1, 0xC5, 0x93, 0xD2, 0x66, 0x27,
        0xDF, 0xFB, 0x10, 0x1A, 0x87, 0xFF, 0x77, 0xDA
)))
digest = bytes(bytearray((
        0x2D, 0xFB, 0xC1, 0xB3, 0x72, 0xD8, 0x9A, 0x11,
        0x88, 0xC0, 0x9C, 0x52, 0xE0, 0xEE, 0xC6, 0x1F,
        0xCE, 0x52, 0x03, 0x2A, 0xB1, 0x02, 0x2E, 0x8E,
        0x67, 0xEC, 0xE6, 0x67, 0x2B, 0x04, 0x3E, 0xE5
)))
signature = bytes(bytearray((
        0x41, 0xAA, 0x28, 0xD2, 0xF1, 0xAB, 0x14, 0x82,
        0x80, 0xCD, 0x9E, 0xD5, 0x6F, 0xED, 0xA4, 0x19,
        0x74, 0x05, 0x35, 0x54, 0xA4, 0x27, 0x67, 0xB8,
        0x3A, 0xD0, 0x43, 0xFD, 0x39, 0xDC, 0x04, 0x93,
        0x01, 0x45, 0x6C, 0x64, 0xBA, 0x46, 0x42, 0xA1,
        0x65, 0x3C, 0x23, 0x5A, 0x98, 0xA6, 0x02, 0x49,
        0xBC, 0xD6, 0xD3, 0xF7, 0x46, 0xB6, 0x31, 0xDF,
        0x92, 0x80, 0x14, 0xF6, 0xC5, 0xBF, 0x9C, 0x40
)))
private_key = bytes2long(private_key)
signature = signature[32:] + signature[:32]

c = GOST3410Curve(*CURVE_PARAMS["GostR3410_2001_TestParamSet"])
```

```
pubX, pubY = public_key(c, private_key)
print "pubX = ", pubX
print "pubY = ", pubY
print

assert long2bytes(pubX) == public_key_x
print "long2bytes(pubX) == public_key_x"
print long2bytes(pubX), " == ", public_key_x
print long2bytes(pubX) == public_key_x
print

assert long2bytes(pubY) == public_key_y
print "long2bytes(pubY) == public_key_y"
print long2bytes(pubY)," == ", public_key_y
print long2bytes(pubY) == public_key_y
print

s = sign(c, private_key, digest)
print "signature = ", s
print "Verify digest with pubX, pubY and -generated- sign = ", verify(c, pubX,
pubY, digest, s)
print "Verify digest with pubX, pubY and -known- sign = ", verify(c, pubX, pubY,
digest, signature)
```

Приложение Б

Реализация алгоритма электронно-цифровой подписи ГОСТ Р 34-10.2012

```
from os import urandom

from pygost.gost3410 import CURVE_PARAMS
from pygost.gost3410 import GOST3410Curve
from pygost.gost3410 import kek
from pygost.gost3410 import public_key
from pygost.gost3410 import sign
from pygost.gost3410 import SIZE_341001
from pygost.gost3410 import SIZE_341012
from pygost.gost3410 import verify
from pygost.utils import bytes2long
from pygost.utils import long2bytes


# 34.10.12

p = bytes(bytearray((
        0x45, 0x31, 0xAC, 0xD1, 0xFE, 0x00, 0x23, 0xC7,
        0x55, 0x0D, 0x26, 0x7B, 0x6B, 0x2F, 0xEE, 0x80,
        0x92, 0x2B, 0x14, 0xB2, 0xFF, 0xB9, 0x0F, 0x04,
        0xD4, 0xEB, 0x7C, 0x09, 0xB5, 0xD2, 0xD1, 0x5D,
        0xF1, 0xD8, 0x52, 0x74, 0x1A, 0xF4, 0x70, 0x4A,
        0x04, 0x58, 0x04, 0x7E, 0x80, 0xE4, 0x54, 0x6D,
        0x35, 0xB8, 0x33, 0x6F, 0xAC, 0x22, 0x4D, 0xD8,
        0x16, 0x64, 0xBB, 0xF5, 0x28, 0xBE, 0x63, 0x73
)))
```

```
q = bytes(bytearray((
        0x45, 0x31, 0xAC, 0xD1, 0xFE, 0x00, 0x23, 0xC7,
        0x55, 0x0D, 0x26, 0x7B, 0x6B, 0x2F, 0xEE, 0x80,
        0x92, 0x2B, 0x14, 0xB2, 0xFF, 0xB9, 0x0F, 0x04,
        0xD4, 0xEB, 0x7C, 0x09, 0xB5, 0xD2, 0xD1, 0x5D,
        0xA8, 0x2F, 0x2D, 0x7E, 0xCB, 0x1D, 0xBA, 0xC7,
        0x19, 0x90, 0x5C, 0x5E, 0xEC, 0xC4, 0x23, 0xF1,
        0xD8, 0x6E, 0x25, 0xED, 0xBE, 0x23, 0xC5, 0x95,
        0xD6, 0x44, 0xAA, 0xF1, 0x87, 0xE6, 0xE6, 0xDF
)))
a = bytes(bytearray((
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07
)))
b = bytes(bytearray((
        0x1C, 0xFF, 0x08, 0x06, 0xA3, 0x11, 0x16, 0xDA,
        0x29, 0xD8, 0xCF, 0xA5, 0x4E, 0x57, 0xEB, 0x74,
        0x8B, 0xC5, 0xF3, 0x77, 0xE4, 0x94, 0x00, 0xFD,
        0xD7, 0x88, 0xB6, 0x49, 0xEC, 0xA1, 0xAC, 0x43,
        0x61, 0x83, 0x40, 0x13, 0xB2, 0xAD, 0x73, 0x22,
        0x48, 0x0A, 0x89, 0xCA, 0x58, 0xE0, 0xCF, 0x74,
        0xBC, 0x9E, 0x54, 0x0C, 0x2A, 0xDD, 0x68, 0x97,
```

```
        0xFA, 0xD0, 0xA3, 0x08, 0x4F, 0x30, 0x2A, 0xDC
)))
x = bytes(bytearray((
        0x24, 0xD1, 0x9C, 0xC6, 0x45, 0x72, 0xEE, 0x30,
        0xF3, 0x96, 0xBF, 0x6E, 0xBB, 0xFD, 0x7A, 0x6C,
        0x52, 0x13, 0xB3, 0xB3, 0xD7, 0x05, 0x7C, 0xC8,
        0x25, 0xF9, 0x10, 0x93, 0xA6, 0x8C, 0xD7, 0x62,
        0xFD, 0x60, 0x61, 0x12, 0x62, 0xCD, 0x83, 0x8D,
        0xC6, 0xB6, 0x0A, 0xA7, 0xEE, 0xE8, 0x04, 0xE2,
        0x8B, 0xC8, 0x49, 0x97, 0x7F, 0xAC, 0x33, 0xB4,
        0xB5, 0x30, 0xF1, 0xB1, 0x20, 0x24, 0x8A, 0x9A
)))
y = bytes(bytearray((
        0x2B, 0xB3, 0x12, 0xA4, 0x3B, 0xD2, 0xCE, 0x6E,
        0x0D, 0x02, 0x06, 0x13, 0xC8, 0x57, 0xAC, 0xDD,
        0xCF, 0xBF, 0x06, 0x1E, 0x91, 0xE5, 0xF2, 0xC3,
        0xF3, 0x24, 0x47, 0xC2, 0x59, 0xF3, 0x9B, 0x2C,
        0x83, 0xAB, 0x15, 0x6D, 0x77, 0xF1, 0x49, 0x6B,
        0xF7, 0xEB, 0x33, 0x51, 0xE1, 0xEE, 0x4E, 0x43,
        0xDC, 0x1A, 0x18, 0xB9, 0x1B, 0x24, 0x64, 0x0B,
        0x6D, 0xBB, 0x92, 0xCB, 0x1A, 0xDD, 0x37, 0x1E
)))
private_key = bytes(bytearray((
        0x0B, 0xA6, 0x04, 0x8A, 0xAD, 0xAE, 0x24, 0x1B,
        0xA4, 0x09, 0x36, 0xD4, 0x77, 0x56, 0xD7, 0xC9,
        0x30, 0x91, 0xA0, 0xE8, 0x51, 0x46, 0x69, 0x70,
        0x0E, 0xE7, 0x50, 0x8E, 0x50, 0x8B, 0x10, 0x20,
        0x72, 0xE8, 0x12, 0x3B, 0x22, 0x00, 0xA0, 0x56,
```

```
        0x33, 0x22, 0xDA, 0xD2, 0x82, 0x7E, 0x27, 0x14,
        0xA2, 0x63, 0x6B, 0x7B, 0xFD, 0x18, 0xAA, 0xDF,
        0xC6, 0x29, 0x67, 0x82, 0x1F, 0xA1, 0x8D, 0xD4
)))
public_key_x = bytes(bytearray((
        0x11, 0x5D, 0xC5, 0xBC, 0x96, 0x76, 0x0C, 0x7B,
        0x48, 0x59, 0x8D, 0x8A, 0xB9, 0xE7, 0x40, 0xD4,
        0xC4, 0xA8, 0x5A, 0x65, 0xBE, 0x33, 0xC1, 0x81,
        0x5B, 0x5C, 0x32, 0x0C, 0x85, 0x46, 0x21, 0xDD,
        0x5A, 0x51, 0x58, 0x56, 0xD1, 0x33, 0x14, 0xAF,
        0x69, 0xBC, 0x5B, 0x92, 0x4C, 0x8B, 0x4D, 0xDF,
        0xF7, 0x5C, 0x45, 0x41, 0x5C, 0x1D, 0x9D, 0xD9,
        0xDD, 0x33, 0x61, 0x2C, 0xD5, 0x30, 0xEF, 0xE1
)))
public_key_y = bytes(bytearray((
        0x37, 0xC7, 0xC9, 0x0C, 0xD4, 0x0B, 0x0F, 0x56,
        0x21, 0xDC, 0x3A, 0xC1, 0xB7, 0x51, 0xCF, 0xA0,
        0xE2, 0x63, 0x4F, 0xA0, 0x50, 0x3B, 0x3D, 0x52,
        0x63, 0x9F, 0x5D, 0x7F, 0xB7, 0x2A, 0xFD, 0x61,
        0xEA, 0x19, 0x94, 0x41, 0xD9, 0x43, 0xFF, 0xE7,
        0xF0, 0xC7, 0x0A, 0x27, 0x59, 0xA3, 0xCD, 0xB8,
        0x4C, 0x11, 0x4E, 0x1F, 0x93, 0x39, 0xFD, 0xF2,
        0x7F, 0x35, 0xEC, 0xA9, 0x36, 0x77, 0xBE, 0xEC
)))
digest = bytes(bytearray((
        0x37, 0x54, 0xF3, 0xCF, 0xAC, 0xC9, 0xE0, 0x61,
        0x5C, 0x4F, 0x4A, 0x7C, 0x4D, 0x8D, 0xAB, 0x53,
        0x1B, 0x09, 0xB6, 0xF9, 0xC1, 0x70, 0xC5, 0x33,
```

```
        0xA7, 0x1D, 0x14, 0x70, 0x35, 0xB0, 0xC5, 0x91,
        0x71, 0x84, 0xEE, 0x53, 0x65, 0x93, 0xF4, 0x41,
        0x43, 0x39, 0x97, 0x6C, 0x64, 0x7C, 0x5D, 0x5A,
        0x40, 0x7A, 0xDE, 0xDB, 0x1D, 0x56, 0x0C, 0x4F,
        0xC6, 0x77, 0x7D, 0x29, 0x72, 0x07, 0x5B, 0x8C
)))
signature = bytes(bytearray((
        0x2F, 0x86, 0xFA, 0x60, 0xA0, 0x81, 0x09, 0x1A,
        0x23, 0xDD, 0x79, 0x5E, 0x1E, 0x3C, 0x68, 0x9E,
        0xE5, 0x12, 0xA3, 0xC8, 0x2E, 0xE0, 0xDC, 0xC2,
        0x64, 0x3C, 0x78, 0xEE, 0xA8, 0xFC, 0xAC, 0xD3,
        0x54, 0x92, 0x55, 0x84, 0x86, 0xB2, 0x0F, 0x1C,
        0x9E, 0xC1, 0x97, 0xC9, 0x06, 0x99, 0x85, 0x02,
        0x60, 0xC9, 0x3B, 0xCB, 0xCD, 0x9C, 0x5C, 0x33,
        0x17, 0xE1, 0x93, 0x44, 0xE1, 0x73, 0xAE, 0x36,
        0x10, 0x81, 0xB3, 0x94, 0x69, 0x6F, 0xFE, 0x8E,
        0x65, 0x85, 0xE7, 0xA9, 0x36, 0x2D, 0x26, 0xB6,
        0x32, 0x5F, 0x56, 0x77, 0x8A, 0xAD, 0xBC, 0x08,
        0x1C, 0x0B, 0xFB, 0xE9, 0x33, 0xD5, 0x2F, 0xF5,
        0x82, 0x3C, 0xE2, 0x88, 0xE8, 0xC4, 0xF3, 0x62,
        0x52, 0x60, 0x80, 0xDF, 0x7F, 0x70, 0xCE, 0x40,
        0x6A, 0x6E, 0xEB, 0x1F, 0x56, 0x91, 0x9C, 0xB9,
        0x2A, 0x98, 0x53, 0xBD, 0xE7, 0x3E, 0x5B, 0x4A
)))
private_key = bytes2long(private_key)
signature = signature[64:] + signature[:64]


c = GOST3410Curve(p, q, a, b, x, y)
```

```
pubX, pubY = public_key(c, private_key)
print "pubX = ", pubX
print "pubY = ", pubY
print

assert long2bytes(pubX) == public_key_x
print "long2bytes(pubX) == public_key_x"
print long2bytes(pubX)," == ", public_key_x
print long2bytes(pubX) == public_key_x
print

assert long2bytes(pubY) == public_key_y
print "long2bytes(pubY) == public_key_y"
print long2bytes(pubY)," == ", public_key_y
print long2bytes(pubY) == public_key_y
print

s = sign(c, private_key, digest, size=SIZE_341012)
print "signature = ", s
print "Verify digest with pubX, pubY and -generated- sign = ", verify(c, pubX,
pubY, digest, s, size=SIZE_341012)
print "Verify digest with pubX, pubY and -known- sign = ", verify(c, pubX, pubY,
digest, signature, size=SIZE_341012)
```