

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет информационных
технологий и электроники

Специальность 10.05.03 –

Специализация -

Кафедра
«Информационная безопасность
систем и технологий»

Информационная безопасность
автоматизированных систем
Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРАКТИКУМ ПО ПРОГРАММНОЙ РЕАЛИЗАЦИИ
КРИПТОГРАФИЧЕСКИХ ПРЕОБРАЗОВАНИЙ НА ЭЛЛИПТИЧЕСКИХ
КРИВЫХ

Студент		<u>Мышенков Никита Александрович</u>
Руководитель		<u>Липилин Олег Владимирович</u>
Нормоконтролёр		<u>Фатсев Алексей Григорьевич</u>
Рецензент	<u>начальник НИС АО «ПНИЭИ»</u>	 <u>Старкин Алексей Валерьевич</u>

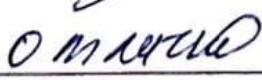
Работа допущена к защите (протокол заседания кафедры от 28.06.21 № 16)

Заведующий кафедрой



С.Л. Зёфиров

Работа защищена с отметкой



(протокол заседания ГЭК

от 28.06.21 № 7)

Секретарь ГЭК



А.П. Иванов

Пенза 2021

Оглавление

Введение.....	4
1 Обзор операций, используемых в криптографических преобразованиях на эллиптических кривых.....	5
2 Способы повышения эффективности программной реализации операций.....	11
3 Анализ угроз и выбор защитных мер программной реализации криптографических преобразований на эллиптических кривых	13
4 Структура практикума.....	15
4.1 Общие сведения	15
4.2 Консольное приложение «Teacher»	16
4.3 Заголовочный файл «Stribog.h» и исходный файл «Stribog.c»	18
4.4 Заголовочные файлы алгоритма блочного шифрования	19
4.5 Заголовочный файл «Crypt.h» и исполняемый файл «Crypt.cpp»	21
5 Методические указания по выполнению работ	26
5.1 Структура работ	26
5.2 Подготовительные действия.....	26
5.3 Реализация алгоритма нахождения обратного элемента над простым полем Галуа	28
5.4 Реализация перехода от эллиптической кривой в форме Вейерштрасса к скрученной эллиптической кривой Эдвардса и обратно	36
5.5 Реализация алгоритма сложения точек эллиптической кривой.....	46
5.6 Реализация алгоритма умножения точки эллиптической кривой на число	55
5.7 Реализация работы с ключевыми контейнерами.....	65
5.8 Реализация алгоритма формирования электронной подписи по ГОСТ Р 34.10-2012.....	71
5.9 Реализация алгоритма проверки электронной подписи по ГОСТ Р 34.10-2012.....	74

5.10 Реализация защитных мер.....	80
6 Обзор вредных психофизиологических факторов, влияющих на нервную систему специалиста по защите информации.....	85
6.1 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации.....	85
6.2 Методы и средства физической культуры, снижающие их воздействие.....	87
Заключение	89
Список используемых источников.....	90

Введение

Все сферы современного общества непосредственно связаны с информационными технологиями. Обмен данными, оказание услуг, рабочий и учебный процесс осуществляются в последнее время часто в электронном виде. В связи с этим возникает необходимость обеспечения конфиденциальности, целостности и доступности передаваемой информации во избежание нанесения ущерба личности, организации и государству в целом.

Разработано достаточное количество криптографических методов и алгоритмов, позволяющих защитить информацию. Однако с ростом вычислительной мощности автоматизированных средств, эти методы становятся либо неактуальными, либо требуют увеличения длины ключа для обеспечения стойкости преобразований.

По этим причинам учёные постоянно занимаются разработкой новых подходов и преобразований, позволяющих обеспечить заданные свойства информации. Одним из новых направлений в криптографии является использование вычислений на эллиптических кривых в ассиметричных алгоритмах. Такой подход позволяет обеспечить заданную стойкость преобразований при меньшей длине ключа, по сравнению с уже существующими алгоритмами.

Для полноценного становления будущих специалистов по информационной безопасности возникает потребность их обучения прогрессивным алгоритмам и методам криптографии. Эта идея становится осуществимой при разработке практикума, позволяющего сделать такое обучение возможным.

1 Обзор операций, используемых в криптографических преобразованиях на эллиптических кривых

В современной криптографии используется две основных криптографических схемы: симметричная и асимметричная. Особенностью первой является то, что открытый и секретный ключ легко вычислимы друг из друга. Но эта особенность накладывает и свои недостатки: абонентам необходимо передавать ключи по закрытому каналу связи. Отличительной особенностью асимметричных схем является то, что знание открытого ключа не позволяет получить секретный. Это приводит к тому, что проблема передачи секретного ключа становится неактуальной. Однако в данных схемах есть и свои недостатки: они работают медленнее по сравнению с симметричными схемами, что заметно сказывается при передаче большого объема данных.

Асимметричные схемы чаще всего используются для выполнения аутентификации пользователей, вычисления и проверки электронной подписи. В последнее время популярность набирают криптографические алгоритмы, основанные на эллиптических кривых, так как они обеспечивают заданную стойкость преобразований по сравнению с вычислениями над полем целых чисел при меньшем размере ключа. Раздел криптографии, занимающийся данными схемами, называется эллиптической криптографией (Elliptic Curve Cryptography).

Основным нормативно-правовым документом, регламентирующим использование криптографических преобразований на эллиптических кривых для вычисления и проверки электронной подписи в РФ, является ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи» [1].

Эллиптической кривой E , определенной над конечным простым полем F_p (где p – простое число), называется множество пар чисел (x, y) , удовлетворяющих тождеству [1].

$$y^2 \equiv x^3 + ax + b \pmod{p}, \quad (1)$$

где $a, b \in F_p$ и $4a^3 + 27b^2 \neq 0 \pmod{p}$.

При этом пары чисел (x, y) , такие, что удовлетворяют условию (1), называются точками эллиптической кривой E , а сами значения x и y – координатами этих точек. Приведённая в формуле (1) эллиптическая кривая называется кривой в форме Вейерштрасса.

Существуют также другие виды кривых, однако именно эта форма используется из-за отличительной особенности: при осуществлении замены координат эллиптическая кривая в любой форме может быть приведена к форме Вейерштрасса.

Эллиптическая кривая E называется сингулярной, если кривой принадлежит хотя бы одна точка, такая, что частные производные по переменной x и y будут равны нулю. В противном случае эллиптическая кривая называется несингулярной. Для криптографии практический интерес представляют именно такие кривые, так как для них в любой точке можно провести касательную.

Условие несингулярности эллиптической кривой.

$$4a^3 + 27b^2 \neq 0 \pmod{p} \quad (2)$$

Графики несингулярных эллиптических кривых представлены на рисунке 1.

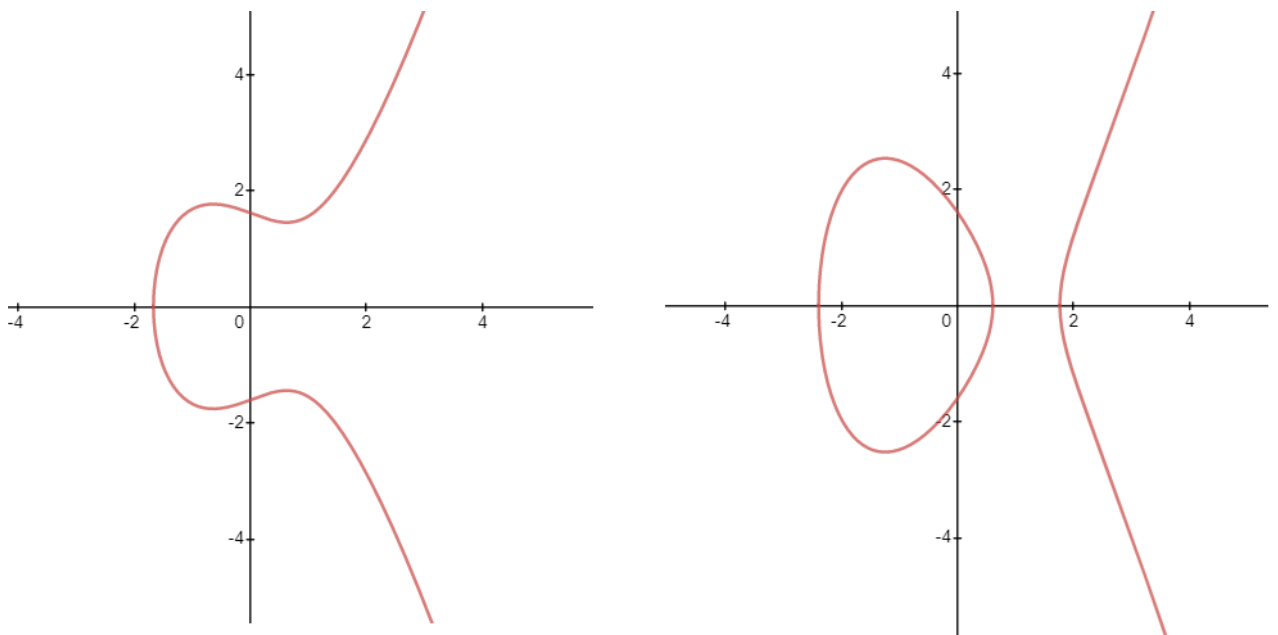


Рисунок 1 - Графики несингулярных эллиптических кривых

Группа точек эллиптической кривой образует аддитивную группу. Это означает, что точки кривой удовлетворяют следующим требованиям [2]:

— замкнутость операции сложения;

$$\forall A, B \in F_p \exists C, C = A + B$$

— ассоциативность;

$$\forall A, B, C \in F_p : (A + B) + C = A + (B + C)$$

— существование нейтрального элемента;

$$\forall A \in F_p \exists e, A + e = e + A = A$$

— существование обратного элемента.

$$\forall A \in F_p \exists (-A), A + (-A) = (-A) + A = e$$

В криптографических преобразованиях на эллиптических кривых используются следующие операции [3]:

— сложение точек;

— умножение точки на число.

График любой несингулярной эллиптической кривой симметричен относительно оси x . В связи с этим определение обратного элемента заключается в следующем: если точка эллиптической кривой R с координатами (x, y) , то обратным для неё элементом будет называться точка $(-R)$ с координатами $(x, -y)$. Графически обратный элемент интерпретируется, как показано на рисунке 2.

Для того, чтобы ввести операцию сложения точек, нужно заметить одно свойство несингулярных эллиптических кривых в форме Вейерштрасса: если на кривой выбрать две произвольные несовпадающие точки и провести через них секущую, то она обязательно пересечёт кривую в третьей и причём единственной точке.

Суммой двух точек P и Q называется точка R , обратная точке пересечения секущей, проведённой через P и Q , и графика эллиптической кривой в форме Вейерштрасса [4]. Графически сумма точек интерпретируется, как показано на рисунке 3.

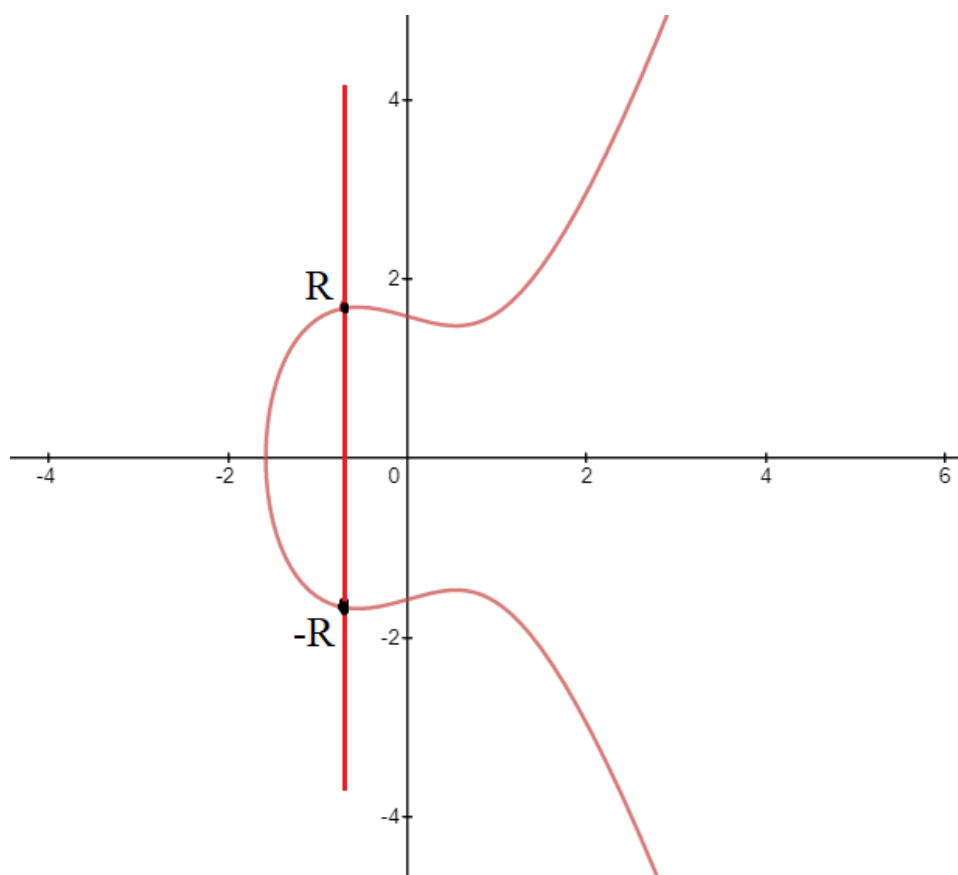


Рисунок 2 - Нахождение обратного элемента

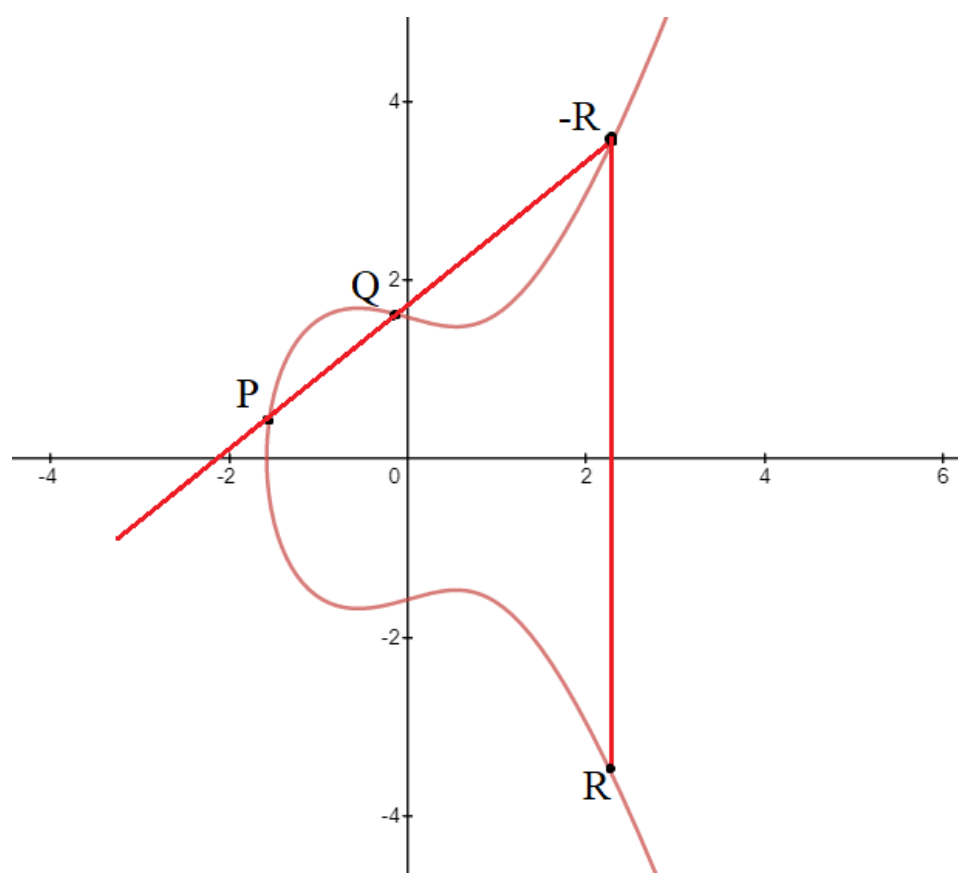


Рисунок 3 - Сложение точек

Формула нахождения суммы точек эллиптической кривой представлена ниже

$$\begin{aligned}
 P(x_1; y_1) + Q(x_2; y_2) &= R(x_3, y_3) \quad (P \neq \pm Q) \\
 \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \pmod{p} \\
 x_3 &= \lambda^2 - x_1 - x_2 \pmod{p} \\
 y_3 &= \lambda(x_1 - x_3) - y_1 \pmod{p}
 \end{aligned}
 \tag{3}$$

У эллиптических кривых есть ещё одна особенность: при попытке сложить точку саму с собой (удвоить значение) приёмы стандартного сложения не работают. В связи с этим вводится отдельная операция удвоения точки.

Для точки эллиптической кривой P удвоенная точка получается как точка пересечения графика кривой и касательной, проведённой через точку P . Графически удвоение точки интерпретируется, как показано на рисунке 4.

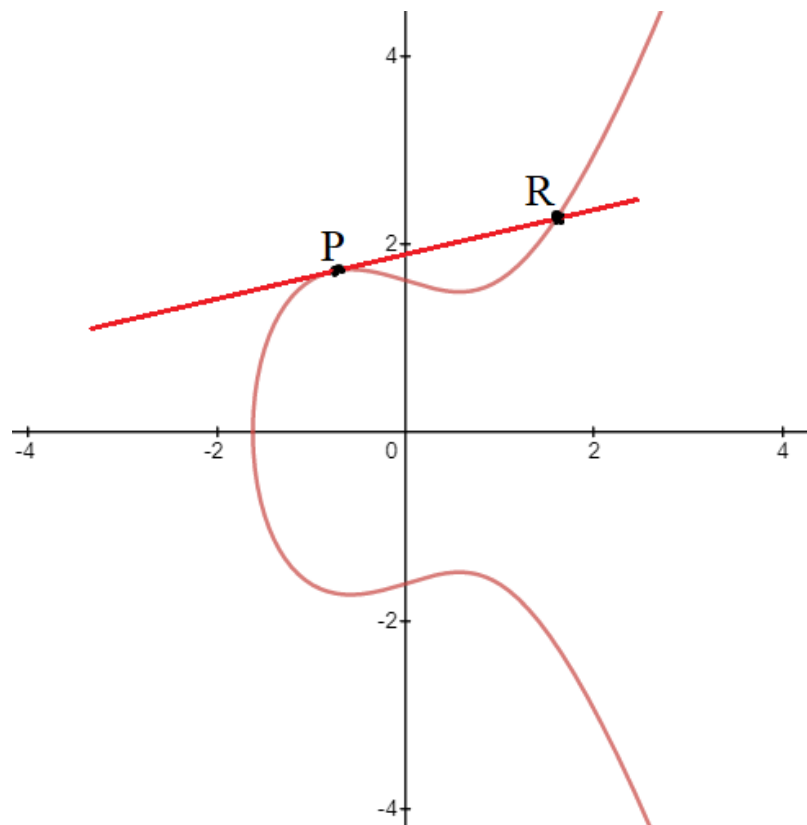


Рисунок 4 - Удвоение точки

Формула удвоения точки представлена ниже.

$$\begin{aligned}2P(x_1; y_1) &= R(x_3, y_3) \\ \lambda &= \frac{3x_1^2 + a}{2y_1} \pmod{p} \\ x_3 &= \lambda^2 - 2x_1 \pmod{p} \\ y_3 &= \lambda(x_1 - x_3) - y_1 \pmod{p}\end{aligned}\tag{4}$$

Скалярным произведением точки P на число m , называется m -кратное сложение точки P в аддитивной группе точек эллиптической кривой.

$$Q = \underbrace{P + \dots + P}_m = mP\tag{5}$$

В связи с тем, что сложение точек и удвоение точки производится по различным формулам, время выполнения этих операций получается различным. Это порождает уязвимость, эксплуатация которой может привести к проведению атаки по времени выполнения на программную реализацию.

2 Способы повышения эффективности программной реализации операций

Эллиптические кривые Вейерштрасса, как способ представления точек, были выбраны из-за своей универсальности. Однако кроме значительных преимуществ, они имеют и недостатки: закон сложения точек не является однородным (вследствие разных алгоритмов сложения и удвоения точки), а также скорость вычислений является низкой [5].

В математике существуют также и другие виды кривых, одной из которых является скрученная эллиптическая кривая Эдвардса, задаваемая уравнением ниже.

$$eu^2 + v^2 = 1 + du^2v^2 \pmod{p} \quad (6)$$

Стандарт при вычислении электронной подписи накладывает ограничения лишь в начальном и конечном представлении точек (чтобы удовлетворяли сравнению 2), однако промежуточные вычисления могут вестись в любом виде. Одним из способов нивелирования недостатков, перечисленных выше, является использование точек, удовлетворяющих условию (6), для промежуточных вычислений.

Специалистами в области криптографии была доказана стойкость и эффективность алгоритмов, в которых для промежуточных вычислений используются точки скрученной кривой Эдвардса [6].

При использовании скрученной кривой Эдвардса выявлены следующие достоинства [7]:

- однородность закона сложения (сложение точек и удвоение точки производится по одинаковым);
- ускорение работы алгоритма в среднем в 1,5 раза по сравнению с алгоритмом, использующим точки эллиптической кривой Вейерштрасса для промежуточных вычислений.

При осуществлении преобразований над эллиптической кривой достаточно длительной по времени является операция нахождения обратного

элемента в поле Галуа. Для ускорения вычислений и повышения эффективности алгоритма могут быть применены проективные координаты точек. При введении проективных координат вместо аффинных используемые формулы становятся проще, а также появляется возможность задействовать такую возможность современных языков программирования, как многопоточность. В конечном итоге скорость выполнения базовых операций над точками эллиптической кривой возрастает от 1,3 до 1,6 раз по сравнению с аффинным представлением.

3 Анализ угроз и выбор защитных мер программной реализации криптографических преобразований на эллиптических кривых

При программной реализации криптографических преобразований на эллиптических кривых были идентифицированы следующие основные угрозы:

- угроза динамического анализа программной реализации;
- угроза несанкционированного доступа к программной реализации;
- угроза конфиденциальности ключа.

Угроза динамического анализа программы заключается в анализе выполнения программы в реальном времени при помощи отладчика. При этом злоумышленник может получить доступ к оперативной памяти компьютера, а также критически важным данным, обрабатываемым в процессе выполнения программы. Утечка ключевой или криптографически опасной информации приведёт к разглашению защищаемой информации, компрометации криптографической схемы.

Для того, чтобы обеспечить защиту от выявленной угрозы, применяются методы защиты от отладки программы, которые позволяют выявить постороннее вмешательство в ход работы программной реализации и незамедлительно остановить выполнение программы.

Угроза несанкционированного доступа к программной реализации заключается в получении нарушителем доступа к среде функционирования программы с осуществлением следующих деструктивных действий: внесение изменений в программную реализацию с целью нарушить её корректное функционирование; получение информации из областей памяти после завершения работы программы при условии, что данные не были уничтожены.

Для того, чтобы обеспечить защиту от выявленной угрозы, реализуются следующие методы защиты:

- реализация встроенного механизма контроля целостности исполняемого кода программы, а также ключевой информации;

— реализация встроенного механизма гарантированного затирания областей памяти, используемых для обработки и хранения ключевой и защищаемой информации.

Угроза конфиденциальности ключа заключается в возможности анализа нарушителем времени выполнения различных криптографических преобразований при работе программы, и использовании полученных данных для определения значений ключевой информации. Полученная информация может быть впоследствии применена для доступа к секретной информации, либо подделки значения электронной подписи.

Для того, чтобы обеспечить защиту от выявленной угрозы, реализуются следующие методы защиты:

— выравнивание времени выполнения операций, за счёт выравнивания длины используемых в операциях аргументов до максимального значения. При этом нельзя будет однозначно сделать вывод о диапазоне значений операндов, используемых во время выполнения программы;

— введение дополнительных случайных временных задержек при выполнении простейших криптографических преобразований, что позволяет внести дополнительную энтропию в процесс работы алгоритма и в разы усложнить анализ времени выполнения операций криптоаналитиком.

4 Структура практикума

4.1 Общие сведения

Практикум по программной реализации криптографических преобразований на эллиптических кривых состоит из следующих основных частей:

- консольное приложение «Teacher»;
- заголовочный файл «Stribog.h» и исходный файл «Stribog.c»;
- заголовочные файлы «Kuznyechik.h» и «mycrypto.h», а также исполняемые файлы «Kuznyechik.cpp» и «mycrypto.cpp»;
- заголовочный файл «Crypt.h» и исполняемый файл «Crypt.cpp»;
- методические указания по выполнению работ с примерами на языке программирования C++.

Структура проекта наглядно продемонстрирована на рисунке 5.



Рисунок 5 - Структура проекта

4.2 Консольное приложение «Teacher»

Консольное приложение используется преподавателем дисциплины. Оно позволяет в автоматизированном режиме генерировать для каждого студента пары секретный – открытый ключ, а также уникальную комбинацию «логин/пароль/секретное слово».

Основные функции приложения показаны на рисунке 6.

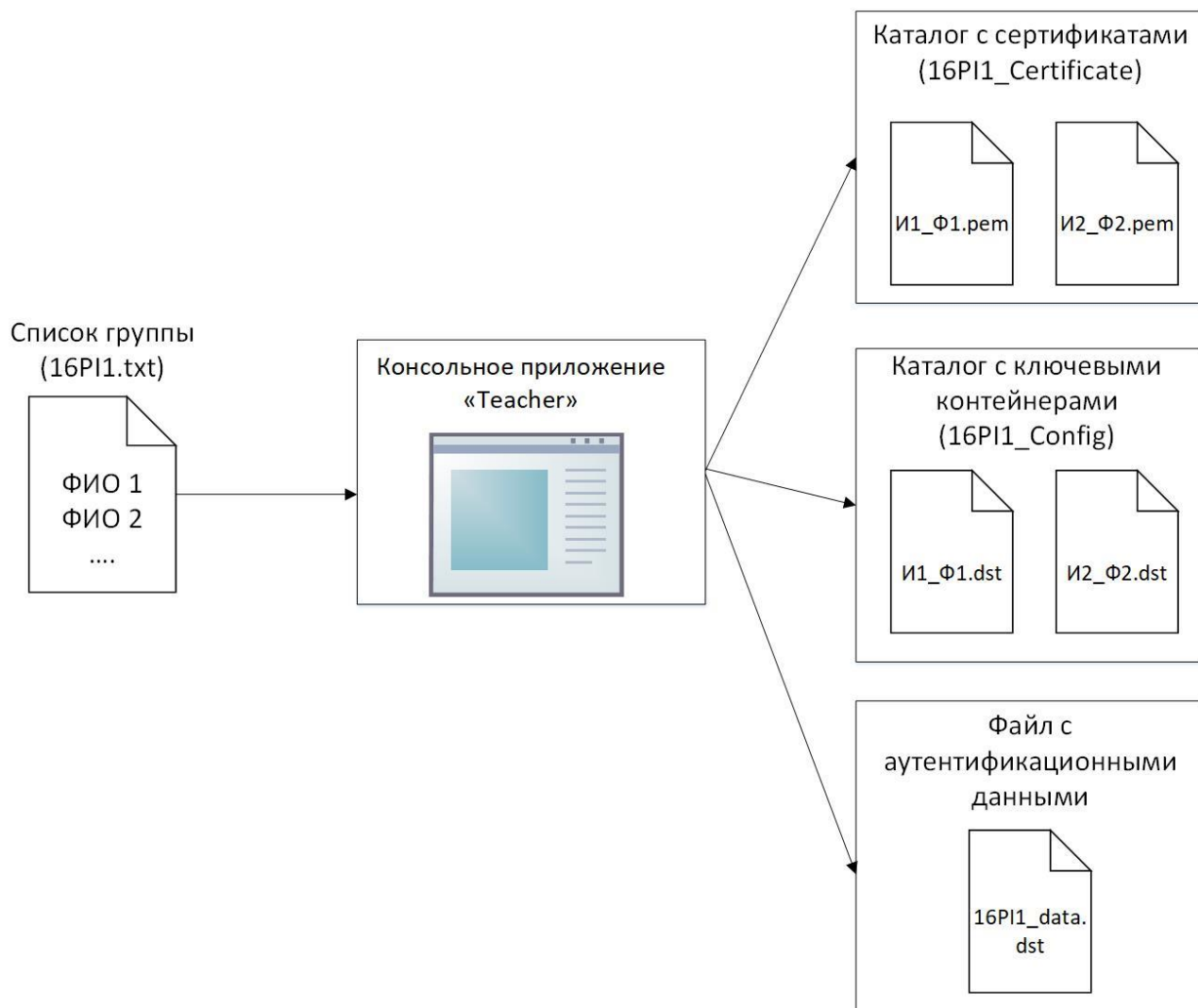


Рисунок 6 - Основные функции приложения "Teacher"

Для работы приложения необходимо создать текстовый файл со списком группы в формате: фамилия, имя отчество. Например, «Иванов Иван Иванович». Информация о каждом студенте должна размещаться на отдельной строке. Имя

файла должно совпадать с именем группы, данные студентов которой представлены в нём.

При запуске консольного приложения «Teacher» необходимо указать полный путь к созданному файлу со списком группы. Далее программа выполнит следующие действия:

- в текущем каталоге будет создана папка с именем «группа_Config», где вместо «группа» будет указана выбранная учебная группа. В данной папке будут храниться ключевые контейнеры ключа электронной подписи, индивидуальные для каждого студента;

- в текущем каталоге будет создана папка с именем «группа_Certificate», где вместо «группа» будет указана выбранная учебная группа. В данной папке будут храниться самоподписанные сертификаты открытого ключа, индивидуальные для каждого студента;

- в текущем каталоге будет создан файл «группа_data.dst», где вместо «группа» будет указана выбранная учебная группа. В этом файле будут храниться сгенерированные логин, пароль, секретное слово, индивидуальные для каждого студента.

Каждому студенту на начало выполнения практикума будет выдано: сертификат открытого ключа электронной подписи; ключевой контейнер ключа электронной подписи; логин, пароль и секретное слово.

Сертификат открытого ключа электронной подписи имеет расширение «.pem» и содержит следующую информацию:

- информацию об издателе сертификата (страна издателя, город, образовательная организация, учебная кафедра);

- информацию о получателе сертификата (фамилия, имя, отчество студента);

- информация о версии сертификата;

- серийный номер сертификата, индивидуальный для каждого изданного сертификата;

- начало и окончание действия сертификата;
- открытый ключ электронной подписи, записанный в формате «GostR3410-2012-256-key», где вместо «key» используется 256-битное 16-ричное представление открытого ключа электронной подписи;
- подпись сертификата, вычисленная на секретном ключе преподавателя.

Секретный ключ электронной подписи хранится в ключевом контейнере ключа электронной подписи в зашифрованном виде.

4.3 Заголовочный файл «Stribog.h» и исходный файл «Stribog.c»

Для вычисления и проверки электронной подписи по ГОСТ Р 34.10-2012 необходимо вычисление хэш-функции от передаваемого сообщения. При этом в качестве хэш-функции рассматривается «Стрибог» [1].

Файлы «Stribog.h» и «Stribog.c» представляют собой реализацию хэш-функции «Стрибог». Они необходимы при вычислении и проверке электронной подписи, а также для работы с ключевыми контейнерами.

Хэш-функция может быть вычислена от конечной строки, а также от файла произвольной длины. Пример вычисления хэш-функции от конечной строки на языке C++ представлен ниже:

```
string stroka, hash;  
hash = Stribog::hashString(stroka);
```

При этом, результатом вычисления хэш-функции всегда является 256-битная строка, представленная в 16-ричном виде.

Пример вычисления хэш-функции от файла на языке C++ представлен ниже:

```
unsigned char str[8]; // переменная, в которую будут считываться  
//данные из файла  
string name_file_podp = "D:\\1.txt"; //Полный путь к файлу  
ifstream file(name_file_podp); //создание файла ввода информации
```

```

Stribog buff; //создание экземпляра класса Stribog
while (file) //пока не достигли конца файла
{
    file.read((char*)&str, sizeof(str)); //считываем данные
    buff.update(&str, sizeof(str)); //обновляем содержимое буфера
}
string hash = buff.hash(); вычисляем хэш от информации

```

4.4 Заголовочные файлы алгоритма блочного шифрования

В лабораторном практикуме преподаватель формирует секретный ключ для каждого из студентов. Хранение секретного ключа в открытом виде приведёт к нарушению свойства конфиденциальности, а в дальнейшем к компрометации всей схемы электронной подписи. В связи с этим, разумным является хранение ключа в зашифрованном виде. Для этих целей используются ключевые контейнеры.

Каждому пользователю выдаётся логин, пароль, секретное слово, индивидуальные для него. От пароля пользователя вычисляется хэш-функция. Полученное значение является ключом шифрования. На этом ключе по алгоритму блочного шифрования «Кузнечик» выполняется зашифрование секретного слова, а также секретного ключа подписи. Полученные данные записываются в файл с расширением «.dst». Данный файл и будет являться ключевым контейнером.

Первое значение в ключевом контейнере служит для проверки корректности введенных данных. Если данные пользователем программы введены верно, то секретный ключ может быть извлечён и расшифрован, иначе выдаётся сообщение об ошибке.

Пример реализации зашифрования 256 битного числа при использовании вышеуказанных файлов на языке C++ представлен ниже:

```

string decrypted_string; //открытый текст
string key; //ключ

```

```

cin >> decrypted_string; //256-битное число, записанное в 16 с.с.
cin >> key; //256-битное число, записанное в 16 с.с.
ByteBlock bb_key = hex_to_bytes(key); //разбиваем ключ по байтам
Kuznyechik a(bb_key); //создаём экземпляр класса для выполнения
//зашифрования
decrypted_string_1 = decrypted_string.substr(0, 32); //128-битное
//число, записанное в 16 с.с.
decrypted_string_2 = decrypted_string.substr(32, 32); //128-битное
//число, записанное в 16 с.с.
ByteBlock bb_decrypted_string_1 = hex_to_bytes(decrypted_string_1);
//Разбиваем по байтам первую часть числа
ByteBlock bb_decrypted_string_2 = hex_to_bytes(decrypted_string_2);
//Разбиваем по байтам вторую часть числа
ByteBlock result1, result2; //переменные для хранения результата
a.encrypt(bb_decrypted_string_1, result1); //зашифровываем 1 часть
//числа
a.encrypt(bb_decrypted_string_2, result2); //зашифровываем 2 часть
//числа
string encrypted_string = (hex_representation(result1) +
hex_representation(result2)); //Получаем результат

```

Пример реализации расшифрования 256 битного числа при использовании вышеуказанных файлов на языке C++ представлен ниже:

```

string encrypted_string; //зашифрованная строка
string key; //ключ
cin >> encrypted_string; //256-битное число, записанное в 16 с.с.
cin >> key; //256-битное число, записанное в 16 с.с.
ByteBlock bb_key = hex_to_bytes(key); //разбиваем ключ на байты
Kuznyechik a(bb_key); //создаём экземпляр класса для расшифрования
encrypted_string_1 = encrypted_string.substr(0, 32); //128-битное
//число, записанное в 16 с.с.
encrypted_string_2 = encrypted_string.substr(32, 32); //128-битное
//число, записанное в 16 с.с.
ByteBlock bb_encrypted_string_1 = hex_to_bytes(encrypted_string_1);
//разбиваем по байтам первую часть числа

```

```

ByteBlock bb_encrypted_string_2 = hex_to_bytes(encrypted_string_2);
//разбиваем по байтам вторую часть числа
ByteBlock result1, result2; //переменные для хранения результата
a.decrypt(bb_encrypted_string_1, result1); //расшифровываем 1 часть
//числа
a.decrypt(bb_encrypted_string_2, result2); //расшифровываем 2 часть
//числа
string decrypted_string = (hex_representation(result1) +
hex_representation(result2)); //получаем результат

```

4.5 Заголовочный файл «Crypt.h» и исполняемый файл «Crypt.cpp»

При программной реализации алгоритма вычисления и проверки электронной подписи, все вычисления проводятся над конечным простым полем с 256 или 512 битными числами [1]. Эти особенности накладывают ограничения на использование стандартных арифметических операций языка программирования C++ и требуют реализации дополнительных функций и инструментов, позволяющих осуществлять работу с такими числами.

Заголовочный файл «Crypt.h» и исполняемый файл «Crypt.cpp» содержат реализованные функции по работе с числами, превышающими разрядную сетку компьютера. Подробное описание и пример использования каждой функции представлены ниже в таблице 1.

Таблица 1 – Описание функций для работы с длинными числами

1) Init()	
Описание	<p>Функция предназначена для инициализации констант, используемых в алгоритме формирования и проверки электронной подписи. К таковым относятся: $p, e, s, t, h, k, d, razn, u, v, zero, q, p_1$. Информация о каждой константе будет представлена по ходу выполнения работы.</p> <p>Вызываться данная функция должна в <i>main()</i> перед тем, как начнёт выполняться алгоритм вычисления или проверки электронной подписи.</p>
Объявление в файле	<code>void Init();</code>

Продолжение таблицы 1

Пример использования	<code>Init();</code>
2) StringToNumber()	
Описание	<p>Большие числа в памяти программы хранятся как массив двухбайтовых чисел (от 0x0000 до 0xFFFF). Однако в документации данные числа представлены одной строкой. Для того, чтобы была возможность вводить числа в программу и выводить их (либо на экран, либо в файл), необходимо реализовать дополнительную функцию.</p> <p>StringToNumber() как раз выполняет данную задачу. Данная функция преобразует 256-битную строку в массив из 16 двухбайтовых чисел (константа CD как раз отвечает за размер такого массива).</p>
Объявление в файле	<code>void StringToNumber(string str, Digit Res[]);</code>
Пример использования	<code>Digit a[CD];</code> <code>StringToNumber("7e7e82520f9f015faald0f18c14ab9fb35188275da3fd94206b74f34a48e0ecd", a)</code>
3) NumToStringDigit()	
Описание	Функция выполняет обратное преобразование из массива двухбайтовых чисел в строку.
Объявление в файле	<code>string NumToStringDigit(const Digit Numb[])</code>
Пример использования	<code>string s = NumToStringDigit(a);</code>
4) Assig()	
Описание	Функция представляет собой аналог оператора « <code>=</code> » и позволяет скопировать данные из first в second. При этом массив second может быть изначально пустым.
Объявление в файле	<code>void Assig(const Digit first[], Digit second[]);</code>
Пример использования	<code>Digit b[CD];</code> <code>Assig(a,b);</code>
5) Less()	
Описание	Функция представляет собой аналог оператора « <code><</code> ». Если значение первого аргумента меньше значения второго, то возвращается True, иначе возвращается False.
Объявление в файле	<code>bool Less(const Digit first[], const Digit second[]);</code>

Продолжение таблицы 1

[illegible]

Продолжение таблицы 1

Пример использования	Digit a[CD], b[CD] , c[CD]; StringToNumber("0000000000000000000000000000000000000000000000000000000000000009", a); StringToNumber("0000000000000000000000000000000000000000000000000000000000000005", b); Int_Div(a,b,c); \\ c = "00...001"
14) Mult()	
Описание	Функция представляет собой аналог оператора «*». Реализовано умножение двух 256 битных чисел по модулю p (константа, которая задана для эллиптической кривой).
Объявление в файле	void Mult(const Digit first[], const Digit second[], Digit Res[]);
Пример использования	Digit a[CD], b[CD] , c[CD]; StringToNumber("0000000000000000000000000000000000000000000000000000000000000009", a); StringToNumber("0000000000000000000000000000000000000000000000000000000000000005", b); Mult(a,b,c); \\ c = "00...002D"
15) Mult_q()	
Описание	Функция по всем параметрам идентична Mult(), только операция умножения реализована по модулю q, а не p.

5 Методические указания по выполнению работ

5.1 Структура работ

Лабораторный практикум по программной реализации криптографических преобразований на эллиптических кривых состоит из следующих работ:

- а) реализация алгоритма нахождения обратного элемента над простым полем Галуа;
- б) реализация перехода от эллиптической кривой в форме Вейерштрасса к скрученной эллиптической кривой Эдвардса и обратно;
- в) реализация алгоритма сложения точек эллиптической кривой;
- г) реализация алгоритма умножения точки эллиптической кривой на число;
- д) реализация работы с ключевыми контейнерами;
- е) реализация алгоритма формирования электронной подписи по ГОСТ Р 34.10-2012;
- ж) реализация алгоритма проверки электронной подписи по ГОСТ Р 34.10-2012;
- и) реализация защитных мер.

В каждой из работ представлены контрольные примеры для проверки корректности реализованных преобразований.

5.2 Подготовительные действия

Перед началом выполнения лабораторного практикума необходимо создать пустой проект и в него включить описанные выше файлы: «Stribog.h», «Stribog.c», «Kuznyechik.h», «mycrypto.h», «Kuznyechik.cpp», «mycrypto.cpp», «Crypt.h», «Crypt.cpp».

Далее необходимо создать исполняемый файл «Source.cpp», в котором будет располагаться функция main(), а также создать файлы «Elleptic_Curve.cpp» и «Elleptic_Curve.h» для работы с эллиптическими кривыми.

В файле «Elleptic_Curve.h» указать следующие директивы препроцессору, подключив другие заголовочные файлы:

```
#pragma once
#include "Cript.h"
#include "Stribog.h"
#include "Kuznyechik.h"
#define CD 16 //количество двухбайтовых слов в числах (в нашем случае
//16 т.к. работаем с 256-битными числами)
```

Заголовочный файл «Elleptic_Curve.h» необходимо включить также в файлы «Elleptic_Curve.cpp» и «Source.cpp».

Далее в файл «Elleptic_Curve.cpp» необходимо добавить следующий код:

```
#define _CRT_SECURE_NO_WARNINGS
#include "Elleptic_Curve.h"
#include <fstream>
#include <windows.h>
#include <direct.h>
#include <stdio.h>
#define CD 16
extern Digit g_p[CD], g_e[CD], g_s[CD], g_t[CD], g_h[CD], g_k[CD],
g_d[CD], g_razn[CD], g_u[CD], g_v[CD], g_zero[CD], g_q[CD],
g_p_1[CD]; //Глобальные переменные
```

А файл «Source.cpp» должен содержать следующий текст:

```
#include "Elleptic_Curve.h"
#include <cmath>
#include <fstream>
#include <string>
#include <iostream>
#include <windows.h>
#define CD 16
```

```

Digit g_p[CD], g_e[CD], g_s[CD], g_t[CD], g_h[CD], g_k[CD], g_d[CD],
g_razn[CD], g_u[CD], g_v[CD], g_zero[CD], g_q[CD], g_p_1[CD];
//Константы
int main()
{
    srand(time(NULL)); //для генерации случайных чисел
    setlocale(LC_ALL, "rus"); // работа с русским языком
    SetConsoleOutputCP(1251); //работа с русским языком
    SetConsoleCP(1251); //работа с русским языком
    Init(); //Инициализация глобальных переменных

    return 0;
}

```

Далее необходимо скомпилировать проект. Компиляция должна пройти без ошибок.

5.3 Реализация алгоритма нахождения обратного элемента над простым полем Галуа

5.3.1 Общие сведения о работе.

Выделяют два основных метода нахождения обратного элемента над простым полем Галуа [2]:

- расширенный алгоритм Евклида;
- модулярное возведение в степень.

Первый способ основывается на обычном алгоритме Евклида и заключается в рекурсивном вызове функции [3].

Второй способ основывается на свойстве чисел и заключается в возведении числа в достаточно большую степень [4].

Для выполнения первой работы на основном уровне необходимо реализовать только расширенный алгоритм Евклида. Для выполнения первой

работы на углубленном уровне необходимо реализовать оба этих способа и произвести их сравнительную характеристику.

5.3.2 Расширенный алгоритм Евклида.

Алгоритм расширенного алгоритма Евклида в форме блок-схемы представлен на рисунке 7.

Для нахождения обратного элемента необходимо на вход рекурсивной функции передавать копию данных. В C++ при передаче массива на вход функции, передаётся не копия данных, а ссылка на оригинал. Чтобы обойти данную проблему, необходимо ввести новую структуру данных. Пример представлен ниже:

```
// Структура для работы с алгоритмом Евклида
struct EVCL
{
    Digit a1[CD], a2[CD], a3[CD];
};
```

В таком случае становится возможным передача копии данных на вход функции, а следовательно и нахождение обратного элемента.

Основная функция нахождения обратного элемента требует на вход три аргумента. Работать с ней напрямую не совсем удобно. Поэтому необходимо создать вспомогательную функцию для нахождения обратного элемента. Пример представлен ниже:

```
// Вспомогательная функция для нахождения обратного элемента по
// модулю p
void rev(Digit prym[], Digit obr[]) {
    EVCL A; //Объявляется структура
    Assig(prym, A.a1); //Прямой элемент заносится первым значением
    Assig(g_p, A.a2); //Модуль заносится вторым и третьим значением
    Assig(g_p, A.a3); A = gcd(A); //Вызывается рекурсивная функция
    Assig(A.a2, obr); //Второй элемент в структуре – искомое значение
}
```

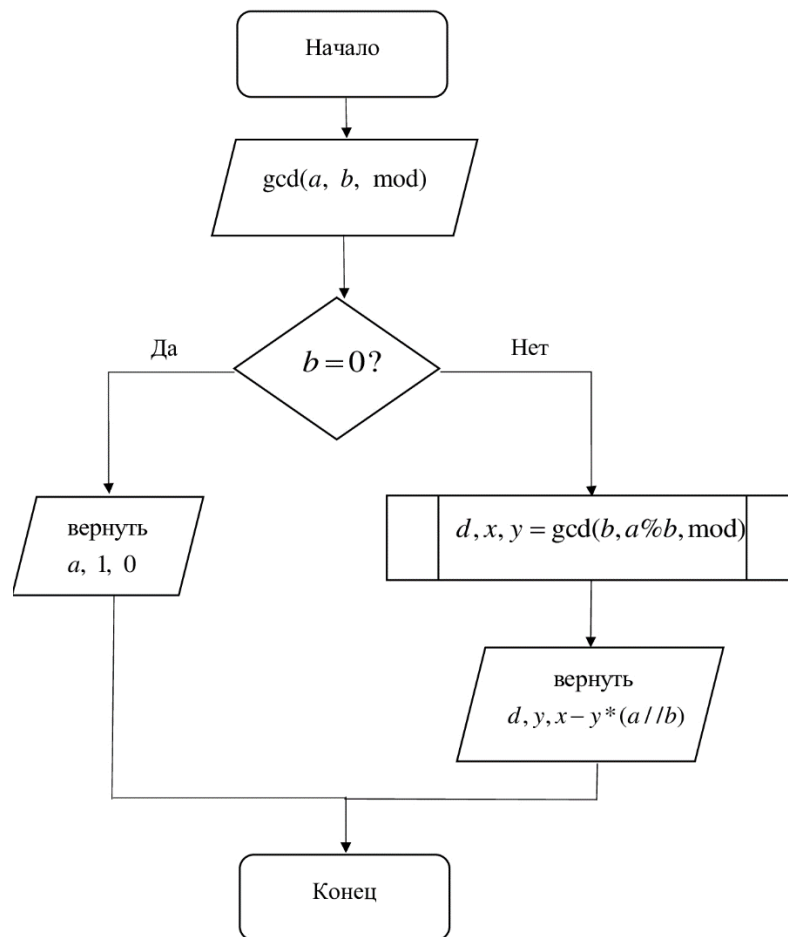


Рисунок 7 – Алгоритм расширенного алгоритма Евклида

Также для работы алгоритма будет нужна функция, проверяющая, отлично число от нуля или нет. Пример такой функции представлен ниже:

```

bool dlna(Digit elem[]) {
    for (int i = 0; i < CD; i++) //проходим по цифрам
    {
        if (elem[i] != 0) { return false; } //если встретилась
//ненулевая цифра, то возвращаем false, иначе true
    }
    return true;
}
  
```

Основную функцию, реализующую расширенный алгоритм Евклида необходимо написать самостоятельно, основываясь на алгоритме, представленном на рисунке 1. Пример объявления функции:

```

EVCL gcd(EVCL X);
  
```

При этом для осуществления промежуточных вычислений рекомендуется создавать динамические массивы с последующим освобождением памяти после их применения:

```
Digit *res = new Digit[CD]; //создание динамического массива
delete[] res;
```

Контрольные примеры нахождения обратного элемента представлены в таблице 2.

Таблица 2 – Контрольные примеры для нахождения обратного элемента

Прямой элемент	Обратный элемент (по модулю p)
cb6075cd06a5a3c7aеbe4ec82e65f8fa 44bdb635671c8383806fb26dc5041b4c	651944b867a5e1de457bcea5f2c715e8 a12941687c4ae55ad87ae87d93e52981
0d0d243259539bd1aba2736c70fc717c ba7d8629928a729204deb9eb170491a2	fafa9a56e96a23d2e84b48f982330e01 7c84cf1de7b7f15c98d4dfa498fcbae3
11f8cd98f4cf819ec83791af9bf96496 c93b66d596d798f91ac80583ab7739bc	be5871230a744f8bc953fdca085f79a3 dc22b07131d915145b78052222a85d1f

В случае успешного выполнения контрольных тестов, необходимо найти обратные элементы для чисел, представленных в таблице 3 по вариантам.

Таблица 3 – Варианты заданий для работы «Реализация алгоритма нахождения обратного элемента над простым полем Галуа»

Вар иант	№ Чис ла	Прямой элемент
1	1	07b4dc57b882bbdadfeac30c98060cb7edac68d3c85a2c983f61ebb b4328c7a2
	2	b1e2506a61369776b0db764b5232fb01d1a19d11143250930bd581f 81b6b3f29
2	1	e827d5a312e16d6eafdb3ce14d8b141954b73b011d88e30c743e3ae c0c9ca377
	2	2b288901879a8dc92a6fccd0f19f655f51b5a7dc2c488990a28690f d44570756
3	1	1e6fe3d3cb040a908661fc429f1bd5006e5c71670f80e2b548b3459 07a171384
	2	fc2e5913f75d27837a5c9a7524323a735bf8a88065f86f65915e669 1aab4ed83

Продолжение таблицы 3

4	1	5d732ba8d820d356cfc405df08bca38d383569b92f5b3eeab11a364e6d5d0f01
	2	81ef462a5ea2480a75035e399e5aed33f5596331ee0546a65e919761daf6bf19
5	1	34a9f245fb1ca75952f68509c7dc1f8ce1fad63e3b7a142fa41fc453b4065e67
	2	637b8184e24494f710418a1a905743b88dc32806540965acf24e53179c770ee6
6	1	575ae6b97798a5877f52978e1a30ea9da05deb4d9d1635dab8284143fde82f7c
	2	2020768ba8ac82d81d4b39d1cff408778e5eddea10bd7453c2d969b62193c365
7	1	91f9fb8802cdbbea95f328cdd54a4d14926cf8f1fc931f77d02408e57e71a719
	2	ae8fe01f95486b762d65e180dd2fafcc6bcf3e5e5684058a12d32acd66f91704
8	1	3be4da2e9311694404bfc521180181923e82646664b392aa7779c770c9922836
	2	fe28c041fd69ef209038be06f1f157c327d53d028b6b5c65607a565a1a2b4a15
9	1	99cd7ba5fce78b8455e13965f2ee2c7a2a873f03b6895d9037ddeef8879e40b0
	2	eb077499a363a026c570062bae86755ce944dfa6a6db157c55b09be4932a3977
10	1	f100b96d936c817c395794850b9914c28475ae6feb2f501322dd3283cb3a3f8d
	2	1de5f99a4ded640dc702de10a6e9f7bc640f8db372c974cde2beeea095d81719
11	1	56a9a4ee38984fe1c07a357a0a31820c141cfaf5012f1436e24fb6b74d215e55
	2	07f6e40c9da4c7b6729fd739eff6d1a5f6e209f0df109d2891f55ebfbe19eb02
12	1	9cec4dfe35d8f6f6d9b474f616c63ddedb6cceda3c561a75fba71223a48f96f4
	2	3509754ab34fbbbc41cf477cf5063c14171ffe788646c0c8237c4e818e95c7ac
13	1	cf2f53add6085d220892c159c612945bc25514a95afdb785deeb9e0690804a6e
	2	4240d8d00f0aae5b3dfa730b32004779f30dbca1d86652e23fb9d167dea4188a
14	1	dc7633c2cebc6057a02d3a81e6e9e43556fd6f8acfadb86ea2c4f424402edbe1
	2	051a3239e8c6b6a42eabd4b24c85d88f3bd50ae119ac47ecfe9a795461b0a3b8
15	1	c862eee3016e87ee611ef4c41865cfced425246b9925f0bcafdedf8af546a3b2
	2	6419095866cf2cdc5459ba454349bdb5cf2dff3c821de36e469c61b5073b1e9f

Продолжение таблицы 3

16	1	421cdb00d570546241d4f3ac97c5356baf1b8456dec0160f92096d73671fe976
	2	8cda632cb86269f5af684e02d52aec32b7874cc290753f7fefb4f5f7fb36a0a2
17	1	41bd5e2c7968e5b35f6a5d1cf45f595f4c93695be311ae5c04d74e5b3b59e0fc
	2	18802a56016854dfd1053afbe7f73f031f6ed88455cc517bb899d4e6c2262192
18	1	769cad0303c6a7c249c5dcff618c57d423f28e9f1ff8779430c4cf46855a955b
	2	2a9f380b6973ef144b6c77d965ac0962bed5e59f9e9ca15dfd27dd8b47c9a876
19	1	b7d0aefad7bea4aff6a3659a7c29d32090eda26edb6bf49676afe8b89be6cc96
	2	abd5d6766fa130c157a3bbf95c5bb3ea76e5431ee9fa03d1003d368a944ce3ea
20	1	a841d72fd8343492d853abb6039bfdcd242185e60dae802626c358424d79d47b
	2	07c9459a16953c7b395bf0703d2f335342c4a0a6c6fe6eb4e360b06806260561

5.3.3 Модулярное возведение в степень.

Блок-схема алгоритма представлена на рисунке 8.

Суть алгоритма заключается в следующем: для того чтобы найти обратный элемент для числа a , необходимо воспользоваться формулой, представленной ниже.

$$a^{-1} \equiv a^{p-1} \pmod{p}$$

Для ускорения вычислений логичным является использование алгоритма быстрого возведения в степень [5]. Его работа основывается на построении двоичной записи числа. На каждой итерации цикла переменная-буфер возводится в квадрат, а в том случае, если в двоичной записи самого числа стоит единица, то это значение прибавляется к переменной-результату.

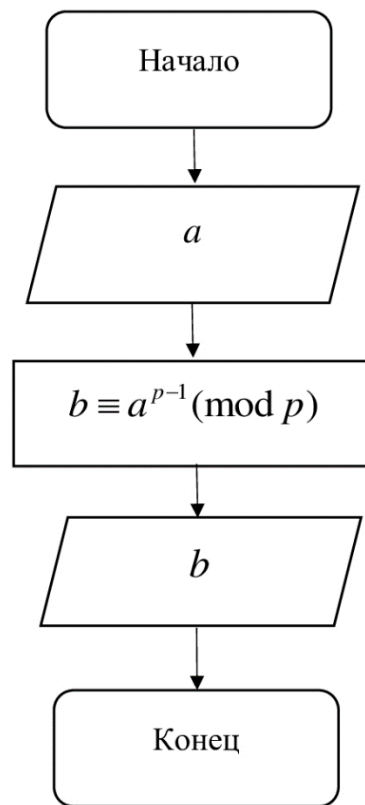


Рисунок 8 - Алгоритм модулярного возведение в степень

Пример алгоритма быстрого возведения в степень, на языке C++ представлен ниже:

```

for (int i = 0; i <= 16 * CD - 1; i++) //Проходим по 256 битам числа
{
    Digit current = step[(int)(i / 16)]; //Считаем текущую
//цифру
    if ((current >> (i % 16)) & 0x0001) { //Проверяем, что
//текущий бит равен 1
        Mult(res, bufer, a); //Умножаем результирующую
//переменную на значение буфера
        Assig(a, res); // заносим результат в переменную res
    }
    Mult(bufer, bufer, a); //возводим значение буфера в квадрат
    Assig(a, bufer);
}
  
```

Для его корректной работы значение *step* должно быть на единицу меньше *p*, начальное значение *res* должно быть равно 1, а начальное значение

переменной *buffer* должно быть равно числу *a*, для которого ищется обратный элемент.

Необходимо, воспользовавшись приведённым алгоритмом, реализовать функцию, находящую обратный элемент методом модулярного возведения в степень.

Пример объявления такой функции на языке C++ представлен ниже:

```
void rev_mod(const Digit ch[], Digit res[]);
```

Для тестирования разработанной функции нужно воспользоваться контрольными примерами, представленными в таблице 3.

Далее необходимо рассчитать обратный элемент для чисел, представленных в таблице 4 согласно своего варианта, и сравнить результаты с функцией, работающей по расширенному алгоритму Евклида.

5.3.4 Задания к работе.

Данная работа подразделяется на два уровня сложности: основной и углубленный.

Для обоих уровней сложности необходимо выполнить следующие задания:

- на языке программирования C++ написать функцию для нахождения обратного элемента в поле Галуа с использованием расширенного алгоритма Евклида. Текст разработанной функции представить в отчёте о выполненной работе;

- выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 3, и отразить результаты в отчёте о выполненной работе;

- согласно своего варианта, найти обратные элементы для чисел, представленных в таблице 4, и отразить результаты в отчёте о выполненной работе.

Для углубленного уровня сложности предназначены следующие дополнительные задания:

- на языке программирования C++ написать функцию для нахождения обратного элемента в поле Галуа с использованием модулярного возведения в

степень. Текст разработанной функции представить в отчёте о выполненной работе;

- выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 3, и отразить результаты в отчёте о выполненной работе;

- согласно своего варианта, найти обратные элементы для чисел, представленных в таблице 4, сравнить полученные значения с результатами работы первой функции и сделать вывод о совпадении найденных значений. Результаты необходимо отразить в отчёте о выполненной работе;

- замерить время выполнения 1000 итераций нахождения обратного элемента для обеих функций, сравнить полученные значения. По результатам сделать выводы о том, какой из алгоритмов работает быстрее.

5.4 Реализация перехода от эллиптической кривой в форме Вейерштрасса к скрученной эллиптической кривой Эдвардса и обратно

5.4.1 Основные сведения о работе.

В ГОСТ Р 34.10-2012 описано формирование электронной подписи с использованием эллиптических кривых Вейерштрасса [1]. Они удобны тем, что уравнение любой кривой может быть приведено к этому виду. Однако они обладают и рядом недостатков, главным из которых является неоднородность закона сложения. [6] Данный недостаток может привести к успешной реализации атаки по времени на криптографический алгоритм. Чтобы этого избежать, для промежуточных вычислений применяют скрученные эллиптические кривые Эдвардса, а результат представляют в форме Вейерштрасса. [7]

Для выполнения успешного перехода между кривыми, необходимо знание параметров, задающих эти кривые. Данные параметры были разработаны и опубликованы Техническим комитетом 026 [8]. Они будут необходимы в работе. Сами параметры представлены ниже:

```

p = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97
a = C2173F1513981673AF4892C23035A27CE25E2013BF95AA33B22C656F277E7335
b = 295F9BAE7428ED9CCC20E7C359A9D41A22FCCD9108E17BF7BA9337A6F8AE9513
e = 0000000000000000000000000000000000000000000000000000000000000001
d = 0605F6B7C183FA81578BC39CFAD518132B9DF62897009AF7E522C32D6DC7BFFB
u = 00000000000000000000000000000000000000000000000000000000000000D
v = 60CA1E32AA475B348488C38FAB07649CE7EF8DBE87F22E81F92B2592DBA300E7
x = 91E38443A5E82C0D880923425712B2BB658B9196932E02C78B2582FE742DAA28
y = 32879423AB1A0375895786C4BB46E9565FDE0B5344766740AF268ADB32322E5C
q = 400000000000000000000000000000000000000000000000000000000000FD8CDDFC87B6635C115AF556C360C67

```

Описание параметров:

- p – характеристика простого поля, над которым определяется эллиптическая кривая;
- a, b – коэффициенты эллиптической кривой в форме Вейерштрасса;
- e, d – коэффициенты эквивалентной скрученной эллиптической кривой в форме Эдвардса;
- q – порядок подгруппы точек простого порядка группы точек эллиптической кривой;
- (x, y) – координаты точки P (порождающего элемента подгруппы простого порядка) на кривой в форме Вейерштрасса;
- (u, v) – координаты точки P при эквивалентном представлении в форме скрученной кривой Эдвардса.

5.4.2 Переход от эллиптической кривой в форме Вейерштрасса к скрученной кривой Эдвардса.

Алгоритм перехода представлен на рисунке 9.

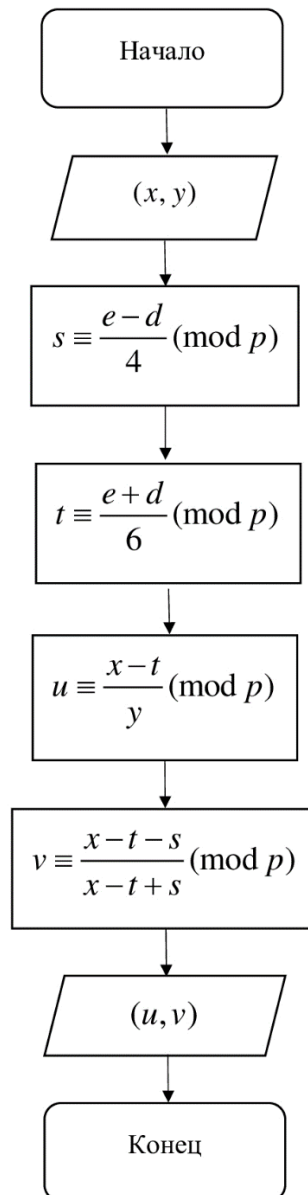


Рисунок 9 – Алгоритм перехода от формы Вейерштрасса к форме Эдвардса

Перед началом реализации алгоритма необходимо предварительно рассчитать константы s и t . Полученные значения необходимо сравнить со значениями этих констант в файле «*Cript.cpp*» и удостовериться в корректности расчётов.

Также необходимо учесть важный момент относительно вычислений: чтобы разделить на число в поле Галуа, необходимо найти для этого числа обратный элемент и выполнить умножение на этот обратный элемент. Для нахождения обратного элемента нужно применить одну из реализованных в предыдущей работе функций.

Продолжение таблицы 5

Точка В	
x	c7ca518e3ef9ad61579287c9523fc9a91c9aead6f36d7ef0e49e9314edfab1b4
y	b248bee0b70825b311795b8ea2030f7a8d210b87b4216859784722261438b028
Вариант 2	
Точка А	
x	6e25cb3cd4a8a16ba5983bd26d30231ebf65c9619f70a2289788ca0def028de1
y	7544f0ae68e82091e8375468178d984daf22221a803e1e7e9daf5d30c95331d1
Точка В	
x	5fcbfe0130206400e2ef3054f5486d2760277bcb2a502e9f3bd18846331fca2d
y	b584cbd2fbf014defdf15037c268444dda3e879a55f4df315c8b1e4ab37220c1
Вариант 3	
Точка А	
x	3508bea1bde4908b6c2572d32da08ff7f5741780e20e0a966e0f4ea20eac30b5
y	45e1798bc02b54c4dabcb8233c66eb778acd6dc44b2a21b140d22b7a8a49ac8f
Точка В	
x	0072eed242c7703f05481b4a9c341948305a42ff89f0314991660909d8dcd8b7
y	7c072bda7f21d70cc4bb573ae7eb6773f0339d7624a7763f5e2b252b81dfb149
Вариант 4	
Точка А	
x	2a350c8ea39912219b67e3f2cc3fa3a79e0c24f851ed88727b428ea7b6690f35
y	0ef2c74f541f216344f32de39591fc00164f14f35298a2966ed605f9603e668a
Точка В	
x	264836695a460dae5a1c8e7a342c4a4737ec10138ed425ffccb52d29708f5e62
y	3ef34361604ee65df5caad6cdd41970e2d73df8383325b511314db24c748c6d1
Вариант 5	
Точка А	
x	1d40c1676805f9518be1fb4c7ae460d3608581e477b07c2d0e7e1e265a6b3347
y	8291ace380fd8832baca29613ab5626c302d13348f204d727d30897a8e1f8934
Точка В	
x	dabde3d3cdf37b92712d0bf66ae5e2160b4841c9deaf2198c1868fbe4c4c2524
y	fb29a174cf79d33d8d0fee713750fc7b1d4a0b6086a2182c8c6b2879f21a9c98
Вариант 6	
Точка А	
x	2f7d6d0cd92656105d00c7636ef0d00e63a4af79909ef7e60671f0f3be2e5687
y	2aae99d9bec842e2d837eb87cae1cd897fe2061528e4345a18d4d52ce4091b00
Точка В	
x	fb7e76ffc26be53e9858bb15e3c4580f287511bbb72a718b417853d16a83fba9
y	8bdc9fee69b7903c98f8761967173b54dfd976c7282d53446255e10e913709e8
Вариант 7	
Точка А	
x	697b6d2daa6288d434f61fd813abc3242af464b21a74176f67241ae3d18b6f3d
y	9b6f68866aaa1775f8a9049142abba163c1e3538f8e32f8dab4af2148a170c74

Продолжение таблицы 5

Точка В	
x	84fa08690151695409788989f3d21340a72a6d820e02a33aa1787b56d3a0c795
y	071aa92c8a2cb0438ef722d21849299552f5d58a5830f26ad79ad65e3f8481ee
Вариант 8	
Точка А	
x	b216b37b9cc0f195906e3432c2a29c9f9a62ad46656bb509ac21801c1b03af1e
y	41a41af5f12805c26877356b6744cc1535f9f985021be36de6ae53e6d764a6b4
Точка В	
x	7b104442f1af36cf5685cfc7684c7c1fa240edfcdcf053f71d7224f85aa6e8141
y	45aabc4fb72b218b8f81044d5e839a64e65046d3514f666e6c3917949b8903d7
Вариант 9	
Точка А	
x	12d99c1046f546f4ef4788617cb45ac037b961f4116c51217ef737a6e96b09f0
y	d43c2ae4238ba290830eba9cf1d3c0307b84d400b291f3a59c8b7269fe87de6b
Точка В	
x	10bb1dc7432010dc5d14d40ad6b13c6fb06de817e0aa2589104ab4e72b2a5f92
y	dd0debfb02ebb15aa7b04a810caf9022f6151e5b3654842de8a9f513de70a7b43
Вариант 10	
Точка А	
x	5c2a559d1759ac120ed164f483394d50aaa955df43d2b3f6f1d6e4edb5b4647a
y	d9d8db567242513d0ffc32b84159c852957e8cd871c4b111e21f0cdc967c4cb1
Точка В	
x	1b92f7a3dbdc941dbb844197a79815cd4c93196100ea4fd3e226b9dd7597a3fe
y	394d654fc8303797fd38d4d80aabebe0d4e7b1baabf66a6ca057c0df4b3366c4
Вариант 11	
Точка А	
x	d405ed33759057b77ce6827cb6b7dc2bbc1a37f18eb24931917a8caa8ed0e0b3
y	430f8208ddf32a70d6fca6bce899679046bfcc1d51bd5c9abf49a9ab3ea990b9
Точка В	
x	da88cd5d4e38126339dd0360aa51278bb891d72302495988c4478e7bbfab0c29
y	38f49756aee9f19293c4aeed2aa13fc113c6ae915a357f2dae5ee13bfa020107
Вариант 12	
Точка А	
x	1fca8b91e2774568a928db2ad8af6803dd5d59df94856816674250fe3b946789
y	2ee2d5b18d83a5461208201635df201d963dc02214a3e70ead3d967f11706d2c
Точка В	
x	c9cbaeeffaabc51147130fc6fa1adbe72140e35c5911b7d54b12beecd5f5848943
y	3c2e34cdfa1e6e76d6cc57ec871a26fb5b23d01c540e6f0d8f77f4d81fe3f613

Продолжение таблицы 5

Вариант 13	
Точка А	
x	43e52883e2331cc8b94e16b6f4a78464ebeb0592a510860c85309a53844e48cd
y	a06d100ce3e672dec6545d010350aa1f6b35a11beed6a92f2a021be374ab5b11
Точка В	
x	182f29adadafa136a9635ed9d2688245885eece8b6327cd67b87d7e099243d23
y	d7f1e9c9c09f80c8d201c47524095b9cd0571e969eeedd3154ba59fb7db710f7
Вариант 14	
Точка А	
x	1a0f23dac76071e6f29112a4623f2ee48250a919d05f6b174dd749ef0242b7de
y	5bee822353dfe0468080c18214252cfb4b178dae2441fe3af3011f27b1f17ce7
Точка В	
x	3c9c274336b21af8a659d6360d27bf754fdbbc388ca09482916d41c2ac218bc8
y	eb073ac056d069bd0fea34bd1db2a52204256091c3e2a65e53066a832aa463b9
Вариант 15	
Точка А	
x	3680967cbae326099d4b1f59da76495c835900a4b15bf58fe816eeae9a6c731b
y	3ba5b103438c5819e2fb1c8fa0a82235e1f25c47a1f6fb60f920f5a75ccb5b06
Точка В	
x	a6529203be8b7be4eb5bc8d5bb2039e7f30cd5f96080afc441d68be27a0ca8f8
y	5e984e973dd41eba318ec4f6621d1224ba3cd7ce00228f794f00865bb8928dda
Вариант 16	
Точка А	
x	7c62f4ff84427d9c84074d745167bf0a1e1b5b2fa400836b4b991bec29bf999f
y	08ee3d6e617b30d21acddea988a09a02e8fae6fe1b498e2efa951e9eb1b0f2
Точка В	
x	f3d579efdb90172f52f9db2996cbc45f1b70fb3042c1c38717d995c5df469397
y	9268e964b814d81b031f66aa309fdd3ccb9be1620cc58c142e4906f0a5809ec1
Вариант 17	
Точка А	
x	33109bdf2c06340eff491a1677a8b5d36bc265bd708c863f00f21924989c75e1
y	410e4c4fcd8a27472c72f5e10e62608f8c4838945d77a6eea6fbf72c241f3818
Точка В	
x	7f60f457bd3e3b12e90da792a7c7eb680b36a88bb399c571653adb54bc0aaf2d
y	b3aba99266d64b30b2dc7e121081b1dc32be2293cf47ff402ae64361ebdcc1fd
Вариант 18	
Точка А	
x	644c9718f80b40d73557912bf11e538b4b4c1afdd011b3b54c9fbe67011f60d9
y	298410a9d8b4d12c131eca5ac158a14a96802b112ef903615777375b218949da
Точка В	
x	96a6b56a4bde04e3623fdf7a2920c22c72d61cff83273c996773f6eefa2b16b9
y	d3931048534475a1efd3208e185601b4c07ef3baaf1db673d65814f9d16af6d1

Продолжение таблицы 5

Вариант 19	
Точка А	
x	eeb02a91e4b70f9e64dacfc93142f3d040c882d37ffcc0e4b3604369d326848e
y	601b3d6e6ba613752f241aa646fb1009082a99d1e23fb59dd246ed41ab46cd50
Точка В	
x	097254064117c5f60349be07d28537bc7818aaf125538cf58ae1f897adabb273
y	93b2f78fa5acad7526b8c703519b5b64846a60243b8ca7a9e42e9f9b00cdbccc
Вариант 20	
Точка А	
x	94ef5bfa8686b475d8e5722b270e9c4fc636dff68671e66982292d0e543a1e50
y	49251fec3d7b04c91d47b76077414aa0ff6c64fff8f947461f6802f68921427a
Точка В	
x	4afc8d2a703bd0ed2f55f741fc12ad71b8cfd2d78b70758cbf844ce79e83781f
y	fc5c10486f70c07cffe06f1c64a094028a574e85ac112cefb94a0772d2381fec

5.4.3 Переход от скрученной кривой Эдвардса к эллиптической кривой в форме Вейерштрасса.

Алгоритм перехода представлен на рисунке 10.

Далее программно необходимо реализовать обратный переход между кривыми. Для этого снова необходимы значения констант s и t , рассчитанные на предыдущем этапе.

Пример объявления функции на языке C++ представлен ниже:

```
void EdvToVey(Digit u[], Digit v[]);
```

В данной функции на вход подаются (u,v) - координаты точки скрученной кривой Эдвардса. Далее необходимо выполнить преобразования по алгоритму, представленному на рисунке 4, и старые значения координат (u,v) заменить новыми значениями (x,y) .

После реализации функции перехода, необходимо выполнить её тестирование с использованием контрольных примеров, представленных в таблице 4.

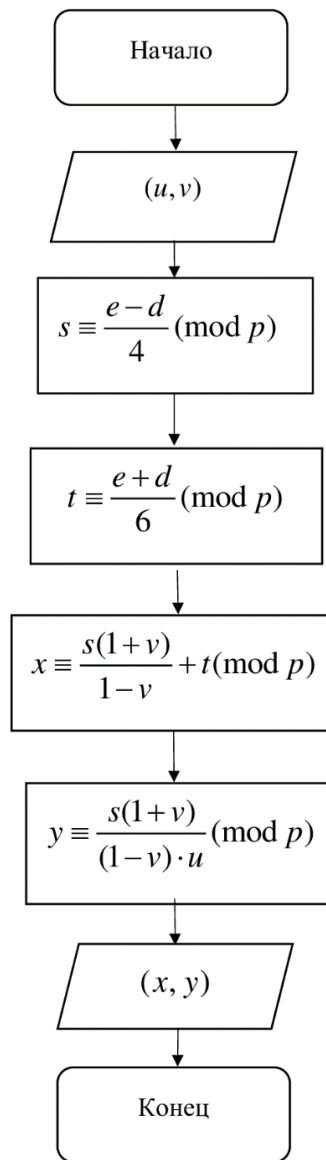


Рисунок 10 – Алгоритм перехода от формы Эдвардса к форме Вейерштрасса

В случае успешного выполнения контрольных тестов, необходимо в соответствии со своим вариантом, для точек A_1 и B_1 с координатами (u, v) , полученными на предыдущем этапе, осуществить обратный переход в форму Вейерштрасса, и сравнить результат с исходными данными варианта, представленными в таблице 5. По полученным результатам сделать соответствующие выводы.

5.4.4 Задания к работе.

В данной работе отсутствует градация уровней сложности. Задания выполняются всеми студентами в одинаковом объёме.

Задания к работе:

— на языке программирования C++ написать функцию для осуществления перехода от эллиптической кривой в форме Вейерштрасса к скрученной кривой Эдвардса. Текст разработанной функции представить в отчёте о выполненной работе;

— выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 4, и отразить результаты в отчёте о выполненной работе;

— согласно своего варианта, для точек A и B с координатами (x,y) в форме Вейерштрасса из таблицы 5, найти соответственные точки A_1 и B_1 с координатами (u,v) для скрученной кривой Эдвардса. Результаты отразить в отчёте о выполненной работе;

— на языке программирования C++ написать функцию для осуществления обратного перехода от скрученной эллиптической кривой Эдвардса к эллиптической кривой в форме Вейерштрасса. Текст разработанной функции представить в отчёте о выполненной работе;

— выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 4, и отразить результаты в отчёте о выполненной работе;

— согласно своего варианта, для точек A_1 и B_1 с координатами (u,v) , полученными в п.3, осуществить обратный переход в форму Вейерштрасса, и сравнить результат с исходными данными варианта, представленными в таблице 5. Полученные результаты отразить в отчёте о выполненной работе. Сделать выводы о совпадении полученных значений координат с первоначальными значениями в форме кривой Вейерштрасса.

5.5 Реализация алгоритма сложения точек эллиптической кривой

5.5.1 Основные сведения о работе.

В рамках данной работы будут рассмотрены два алгоритма сложения точек эллиптической кривой:

- сложение точек с использованием эллиптической кривой в форме Вейерштрасса;
- сложение точек эллиптической кривой с использованием скрученной кривой Эдвардса.

5.5.2 Сложение точек эллиптической кривой с использованием эллиптической кривой в форме Вейерштрасса.

При использовании кривых Вейерштрасса возникает одна особенность: сложение точек и удвоение точки производится по различным формулам. В связи с этим возникает необходимость реализации двух различных функций. [4] Общий алгоритм сложения точек с использованием эллиптической кривой в форме Вейерштрасса представлен на рисунке 11.

Для выполнения промежуточных преобразований необходимо создавать динамические массивы с очисткой памяти после их использования. Пример объявления функции сложения точек на языке C++ представлен ниже:

```
void Add_Vey(  
    const Digit x1[], /* Координата x первой точки */  
    const Digit y1[], /* Координата y первой точки */  
    const Digit x2[], /* Координата x второй точки */  
    const Digit y2[], /* Координата y второй точки */  
    Digit x_res[], /* Координата x итоговой точки */  
    Digit y_res[] /* Координата y итоговой точки */  
)
```

Пример объявления функции удвоения точек эллиптической кривой на языке C++ представлен ниже:

```
void Udv_Vey(  
    const Digit x1[], /* Координата x первой точки */
```


В случае успешного выполнения контрольных тестов, необходимо в соответствии со своим вариантом, найти сумму точек A и B с координатами (x,y) в форме Вейерштрасса. Варианты заданий представлены в таблице 7.

Таблица 7 – Варианты заданий для работы «Реализация алгоритма сложения точек эллиптической кривой»

Вариант 1	
Точка А	
x	47615cd873812de194b6db9ba05c88bc7d69e978ec062b020c0a98b5bd6568bb
y	ff0563dcc219632ad7df0dcdde9882b2e34c6fc2077f22280679422afbfa08e3
Точка В	
x	c7ca518e3ef9ad61579287c9523fc9a91c9aead6f36d7ef0e49e9314edfab1b4
y	b248bee0b70825b311795b8ea2030f7a8d210b87b4216859784722261438b028
Вариант 2	
Точка А	
x	e8fb19165d79fbc24f55f0b1e33d8a71be0226be05294e49caf388b19fb1d0d2
y	dbc bce59da39a13596428feae904e8de8ce929b481d267bed50d5751f1be2169
Точка В	
x	5fcbfe0130206400e2ef3054f5486d2760277bcb2a502e9f3bd18846331fca2d
y	b584cbd2fbf014defdf15037c268444dda3e879a55f4df315c8b1e4ab37220c1
Вариант 3	
Точка А	
x	5e37edf83f668a1d344828d8e482c13e1a451e98187283828b10861138c8a49b
y	3325fa691287835a92299e695063d097ea5539599fd3853eabb6747f37f50d10
Точка В	
x	ed513616221958938a69e12b2be16629f0532c1280503b5a7a5c4bd2272e6cb1
y	535f28050ee88df27430f1deca80ae7131a51b07a923a35160868e1015aef47e
Вариант 4	
Точка А	
x	6891696e0f20268f806d5b6fc7de3646971b7f447f09cd08b338edc464ae5cc6
y	c21bb1477bd52489e61240b12f80af22424efa0af43ed5de106f29d088bf383c
Точка В	
x	264836695a460dae5a1c8e7a342c4a4737ec10138ed425ffccb52d29708f5e62
y	3ef34361604ee65df5caad6cdd41970e2d73df8383325b511314db24c748c6d1
Вариант 5	
Точка А	
x	3e9f00cb35367f8c6558e17ec641ec0db3f6aa4eb87359b38ef9a27ea5436587
y	a6245c8d7cac688a3855351f8b9f96d58305e37b48189061c3ce49e2c0bd5ffd
Точка В	
x	dabde3d3cdf37b92712d0bf66ae5e2160b4841c9deaf2198c1868fbe4c4c2524
y	fb29a174cf79d33d8d0fee713750fc7b1d4a0b6086a2182c8c6b2879f21a9c98

Продолжение таблицы 7

Вариант 6	
Точка А	
x	1a4e6a6810a2f6b2b77dd40976fc5ff09ba88301ab6f6ab60011302aa4640657
y	b27fd1e6e590874fb8928f1785043f5943b75ccec6098cd83405311f96d8fddf
Точка В	
x	fb7e76ffc26be53e9858bb15e3c4580f287511bbb72a718b417853d16a83fba9
y	8bdc9fee69b7903c98f8761967173b54dfd976c7282d53446255e10e913709e8
Вариант 7	
Точка А	
x	c5f1f9ed5fb856b7a13793ae1f6a42321cce4bf4594e50cef853d766c41a4bab
y	219e5871df01fb7bca92887a446fd680b94908ab1405bd8c5135b72f38ac62d9
Точка В	
x	84fa08690151695409788989f3d21340a72a6d820e02a33aa1787b56d3a0c795
y	071aa92c8a2cb0438ef722d21849299552f5d58a5830f26ad79ad65e3f8481ee
Вариант 8	
Точка А	
x	b216b37b9cc0f195906e3432c2a29c9f9a62ad46656bb509ac21801c1b03af1e
y	41a41af5f12805c26877356b6744cc1535f9f985021be36de6ae53e6d764a6b4
Точка В	
x	7b104442f1af36cf5685cfc7684c7c1fa240edfcdcf053f71d7224f85aa6e8141
y	45aabc4fb72b218b8f81044d5e839a64e65046d3514f666e6c3917949b8903d7
Вариант 9	
Точка А	
x	182f29adadafa136a9635ed9d2688245885eece8b6327cd67b87d7e099243d23
y	d7f1e9c9c09f80c8d201c47524095b9cd0571e969eedd3154ba59fb7db710f7
Точка В	
x	10bb1dc7432010dc5d14d40ad6b13c6fb06de817e0aa2589104ab4e72b2a5f92
y	dd0debef02ebb15aa7b04a810caf9022f6151e5b3654842de8a9f513de70a7b43
Вариант 10	
Точка А	
x	b216b37b9cc0f195906e3432c2a29c9f9a62ad46656bb509ac21801c1b03af1e
y	41a41af5f12805c26877356b6744cc1535f9f985021be36de6ae53e6d764a6b4
Точка В	
x	1b92f7a3dbdc941dbb844197a79815cd4c93196100ea4fd3e226b9dd7597a3fe
y	394d654fc8303797fd38d4d80aabebe0d4e7b1baabf66a6ca057c0df4b3366c4
Вариант 11	
Точка А	
x	d405ed33759057b77ce6827cb6b7dc2bbc1a37f18eb24931917a8caa8ed0e0b3
y	430f8208ddf32a70d6fca6bce899679046bfcc1d51bd5c9abf49a9ab3ea990b9
Точка В	
x	dabde3d3cdf37b92712d0bf66ae5e2160b4841c9deaf2198c1868fbe4c4c2524
y	fb29a174cf79d33d8d0fee713750fc7b1d4a0b6086a2182c8c6b2879f21a9c98

Продолжение таблицы 7

Вариант 12	
Точка А	
x	264836695a460dae5a1c8e7a342c4a4737ec10138ed425ffccb52d29708f5e62
y	3ef34361604ee65df5caad6cdd41970e2d73df8383325b511314db24c748c6d1
Точка В	
x	c9cbaeeffaabc51147130fc6fa1adbe72140e35c5911b7d54b12beecdf5848943
y	3c2e34cdfa1e6e76d6cc57ec871a26fb5b23d01c540e6f0d8f77f4d81fe3f613
Вариант 13	
Точка А	
x	3508bea1bde4908b6c2572d32da08ff7f5741780e20e0a966e0f4ea20eac30b5
y	45e1798bc02b54c4dabcb8233c66eb778acd6dc44b2a21b140d22b7a8a49ac8f
Точка В	
x	182f29adadafa136a9635ed9d2688245885eece8b6327cd67b87d7e099243d23
y	d7f1e9c9c09f80c8d201c47524095b9cd0571e969eedd3154ba59fb7db710f7
Вариант 14	
Точка А	
x	c7ca518e3ef9ad61579287c9523fc9a91c9aead6f36d7ef0e49e9314edfab1b4
y	b248bee0b70825b311795b8ea2030f7a8d210b87b4216859784722261438b028
Точка В	
x	3c9c274336b21af8a659d6360d27bf754fdbbc388ca09482916d41c2ac218bc8
y	eb073ac056d069bd0fea34bd1db2a52204256091c3e2a65e53066a832aa463b9
Вариант 15	
Точка А	
x	e8c6740e58d616ca220db7da0d9c3e19b53e86e38bf3e8747774631452ec174c
y	0b837a5e560a29a2327b575f29b4be8baef4bc947fcc2ed4f3264bc434309381
Точка В	
x	a6529203be8b7be4eb5bc8d5bb2039e7f30cd5f96080afc441d68be27a0ca8f8
y	5e984e973dd41eba318ec4f6621d1224ba3cd7ce00228f794f00865bb8928dda
Вариант 16	
Точка А	
x	91e38443a5e82c0d880923425712b2bb658b9196932e02c78b2582fe742daa28
y	32879423ab1a0375895786c4bb46e9565fde0b5344766740af268adb32322e5c
Точка В	
x	f3d579efdb90172f52f9db2996cbc45f1b70fb3042c1c38717d995c5df469397
y	9268e964b814d81b031f66aa309fdd3ccb9be1620cc58c142e4906f0a5809ec1
Вариант 17	
Точка А	
x	fb7e76ffc26be53e9858bb15e3c4580f287511bbb72a718b417853d16a83fba9
y	8bdc9fee69b7903c98f8761967173b54dfd976c7282d53446255e10e913709e8
Точка В	
x	7f60f457bd3e3b12e90da792a7c7eb680b36a88bb399c571653adb54bc0aaf2d
y	b3aba99266d64b30b2dc7e121081b1dc32be2293cf47ff402ae64361ebdcc1fd

Продолжение таблицы 7

Вариант 18	
Точка А	
x	84fa08690151695409788989f3d21340a72a6d820e02a33aa1787b56d3a0c795
y	071aa92c8a2cb0438ef722d21849299552f5d58a5830f26ad79ad65e3f8481ee
Точка В	
x	96a6b56a4bde04e3623fdf7a2920c22c72d61cff83273c996773f6eefa2b16b9
y	d3931048534475a1efd3208e185601b4c07ef3baaf1db673d65814f9d16af6d1
Вариант 19	
Точка А	
x	644c9718f80b40d73557912bf11e538b4b4c1afdd011b3b54c9fbe67011f60d9
y	298410a9d8b4d12c131eca5ac158a14a96802b112ef903615777375b218949da
Точка В	
x	097254064117c5f60349be07d28537bc7818aaf125538cf58ae1f897adabb273
y	93b2f78fa5acad7526b8c703519b5b64846a60243b8ca7a9e42e9f9b00cdbccc
Вариант 20	
Точка А	
x	7c62f4ff84427d9c84074d745167bf0a1e1b5b2fa400836b4b991bec29bf999f
y	08ee3d6e617b30d21acddea988a09a02e8fae6felb498e2efa951e9eb1b0f2
Точка В	
x	4afc8d2a703bd0ed2f55f741fc12ad71b8cfd2d78b70758cbf844ce79e83781f
y	fc5c10486f70c07cffe06f1c64a094028a574e85ac112cefb94a0772d2381fec

5.5.3 Сложение точек эллиптической кривой с использованием скрученной кривой Эдвардса.

При использовании скрученных кривых Эдвардса сложение и удвоение точек производится по одной формуле. Это позволяет избежать недостатка, связанного с использованием кривых в форме Вейертрасса, и обеспечить защиту от атак по времени на алгоритм. [7]

Алгоритм сложения точек с использованием эллиптической кривой Эдвардса представлен на рисунке 12.

Для выполнения промежуточных преобразований необходимо создавать динамические массивы с очисткой памяти после их использования. Пример объявления функции сложения точек на языке C++ представлен ниже:

```
void Add(
    const Digit x1[], /* Координата x первой точки */
    const Digit y1[], /* Координата y первой точки */
```

```

const Digit x2[], /* Координата x второй точки */
const Digit y2[], /* Координата y второй точки */
Digit x_res[], /* Координата x итоговой точки */
Digit y_res[] /* Координата y итоговой точки */)
{

```

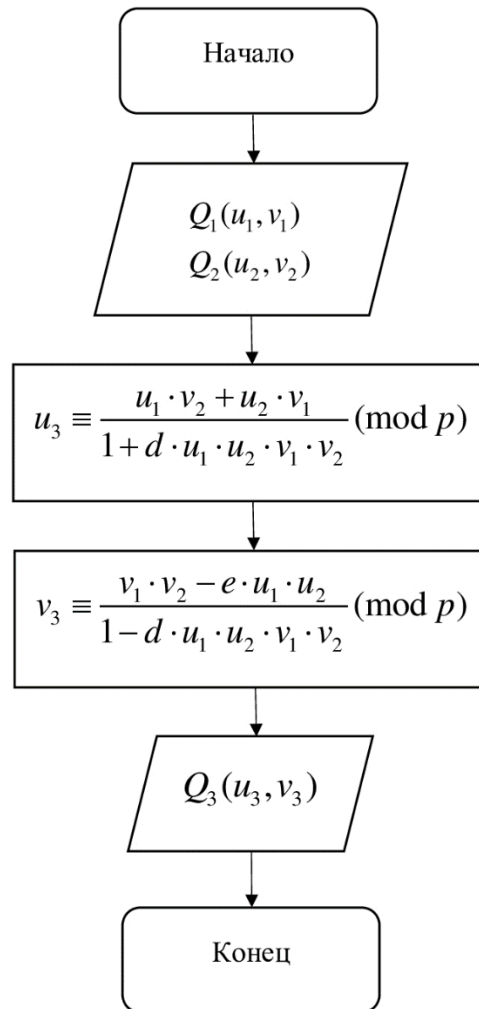


Рисунок 12 - Алгоритм сложения точек с использованием скрученной кривой Эдвардса

При вычислениях используется коэффициент d . Он доступен в программе по имени g_d . Значение данной константы представлено в исполняемом файле *Cript.cpp*.

Далее необходимо написать функции сложения точек с использованием скрученных эллиптических кривых Эдвардса.

После реализации описанных функций, необходимо выполнить их тестирование с использованием контрольных примеров, представленных в

таблице 6. Необходимо учесть, что перед выполнением сложения точек, необходимо выполнить их преобразование к виду точек скрученной кривой Эдвардса. А после выполнения сложения выполнить обратное преобразование к кривой Вейерштрасса.

В случае успешного выполнения контрольных тестов, необходимо в соответствии со своим вариантом, найти сумму точек A и B с координатами (x, y) (координаты даны в форме Вейерштрасса). Варианты заданий представлены в таблице 7. Перед выполнением сложения необходимо выполнить преобразования, описанные абзацем выше.

5.5.4 Задания к работе.

Данная работа подразделяется на два уровня сложности: основной и углубленный.

Для обоих уровней сложности необходимо выполнить следующие задания:

- на языке программирования C++ написать функцию сложения точек с использованием эллиптической кривой Эдвардса. Текст разработанной функции представить в отчёте о выполненной работе;

- выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 6, и отразить результаты в отчёте о выполненной работе;

- согласно своего варианта, выполнить сложение точек A и B с координатами (x, y) в форме Вейерштрасса из таблицы 7, а также удвоение точки A . Предварительно выполнить описанные преобразования. Результаты отразить в отчёте о выполненной работе.

Для углубленного уровня сложности предназначены следующие дополнительные задания:

- на языке программирования C++ написать две функции сложения и удвоения точек с использованием эллиптической кривой в форме Вейерштрасса. Текст разработанных функций представить в отчёте о выполненной работе;

— выполнить тестирование разработанных функций с помощью контрольных примеров, представленных в таблице 6, и отразить результаты в отчёте о выполненной работе;

— согласно своего варианта, выполнить сложение точек A и B с координатами (x,y) в форме Вейерштрасса из таблицы 7, а также удвоение точки A . Результаты отразить в отчёте о выполненной работе;

— сравнить результаты сложения и удвоения точек, полученные с использованием преобразований на кривой Вейерштрасса и преобразований на кривой Эдвардса, и сделать вывод.

5.6 Реализация алгоритма умножения точки эллиптической кривой на число

5.6.1 Основные сведения о работе.

При реализации умножения точки эллиптической кривой на число для ускорения вычислений необходимо использовать быстрый алгоритм умножения [5], аналогичный алгоритму быстрого возведения в степень.

Алгоритм умножения представлен на рисунке 13.

На вход алгоритма подаётся точка A с координатами (x,y) . При этом, если работа ведётся с эллиптическими кривыми Вейерштрасса, то координаты точки должны быть представлены в аналогичном виде, а если работа ведётся с эллиптическими кривыми Эдвардса, то и координаты точки должны быть представлены в таком виде. Также на вход подаётся число k , на которое умножается точка.

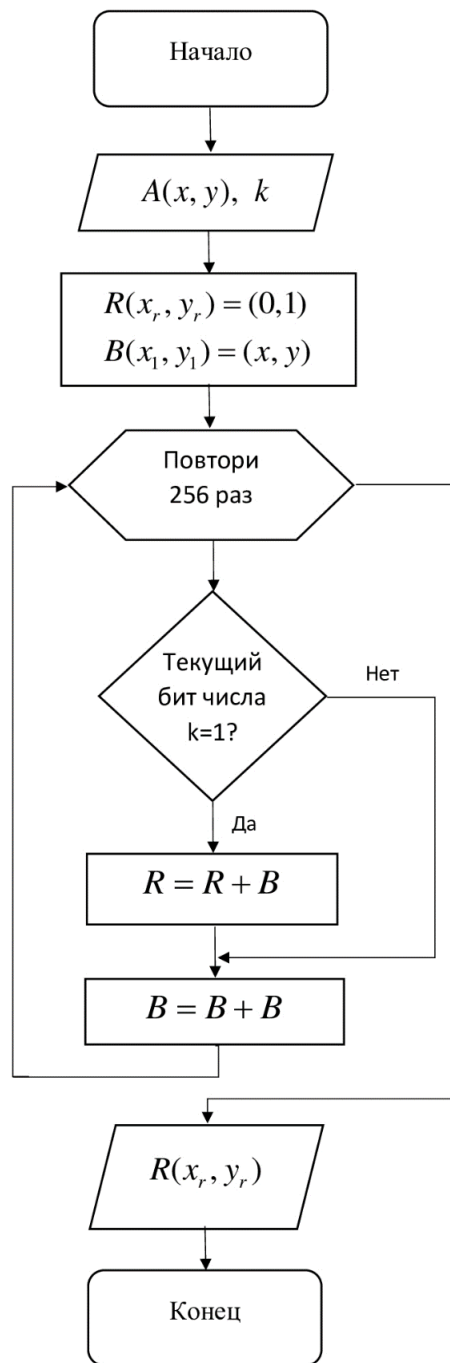


Рисунок 13 - Алгоритм умножения точки на число

Далее задаются начальные значения результирующей точки (x_r, y_r) , а также вспомогательной точки (x_1, y_1) . После этого запускается цикл на 256 итераций (так как работа ведётся с 256 битными числами), и на каждой итерации цикла контролируется двоичный разряд числа k (на 0-й итерации 0-й разряд числа k , на 1-й итерации 1-й разряд числа k и т.д.). В случае, если двоичный разряд коэффициента равен единице, к результирующей точке добавляется

вспомогательная. На каждой итерации цикла вспомогательная точка удваивается.

При этом необходимо учесть некоторые особенности:

— при работе со скрученными кривыми Эдвардса сложение точек и удвоение точки осуществляется одной и той же функцией, а начальные значения результирующей и вспомогательной точек соответствуют приведённым на рисунке 7;

— при работе с эллиптическими кривыми Вейерштрасса сложение точек осуществляется отдельной функцией сложения точек, а удвоение отдельной функцией удвоения точки, как это было реализовано в предыдущей работе. При этом начальное значение вспомогательной точки соответствует значению, представленному на рисунке 7, а с результирующей точкой ситуация обстоит немного сложнее. Первоначально необходимо присвоить ей координаты $(0,0)$, но точка с такими координатами не является нулевой, поэтому дальнейший результат будет искажён. Чтобы этого избежать, необходимо в цикле дополнительно поставить условие: при выполнении сложения, если координаты результирующей точки равны $(0,0)$, то в R просто заносится вспомогательная точка.

Пример реализации быстрого алгоритма умножения точки на число с использованием эллиптической кривой Вейерштрасса на языке C++ представлен ниже:

```
for (int i = 0; i <= 16 * CD - 1; i++) //Проходим по 256 битам числа
{
    Digit current = k[(int)(i / 16)]; //Вычисляем текущую цифру
    if ((current >> (i % 16)) & 0x0001) { //Проверяем, что
//текущий бит числа равен 1
//если в результате ещё ничего нет, то заносим значения из буфера
        if (Equal(x_res, g_zero) && Equal(y_res, g_zero)) {
            Assig(x1, x_res);
            Assig(y1, y_res);
        }
    }
```

```

//если в результате значение есть, то складываем с буфером
    else {
        Add_Vey(x_res, y_res, x1, y1, c1, c2);
        Assig(c1, x_res);
        Assig(c2, y_res);
    }
}

//удваиваем точку
    Udv_Vey(x1, y1, c1, c2);
    Assig(c1, x1);
    Assig(c2, y1);
}

```

При использовании скрученной кривой Эдвардса таких дополнительных проверок делать не нужно.

Пример объявления функции умножения точки на число с использованием эллиптической кривой в форме Вейерштрасса на языке C++ представлен ниже:

```

void Mult_Vey(
    const Digit x[], /* Координата x точки */
    const Digit y[], /* Координата y точки */
    const Digit k[], /* Число, на которое осуществляется умножение */
    Digit x_res[], /* Координата x итоговой точки */
    Digit y_res[] /* Координата y итоговой точки */
)

```

Пример объявления функции умножения точки на число с использованием скрученной эллиптической кривой Эдвардса на языке C++ представлен ниже:

```

void Mult(
    const Digit x[], /* Координата x точки */
    const Digit y[], /* Координата y точки */
    const Digit k[], /* Число, на которое осуществляется умножение */
    Digit x_res[], /* Координата x итоговой точки */
    Digit y_res[] /* Координата y итоговой точки */
)

```

Далее необходимо написать соответствующие функции на языке C++.

После реализации описанных функций необходимо выполнить их тестирование с использованием контрольных примеров, представленных в таблице 8. Обратите внимание на то, что перед умножением точки на число с использованием скрученных кривых Эдвардса, необходимо выполнить преобразование координат точки из вида Вейерштрасса в вид Эдвардса. После выполнения умножения, необходимо выполнить обратное преобразование.

Таблица 8 – Контрольные примеры для тестирования функции умножения точки на число

Контрольный пример 1	
Точка A1	
x	91e38443a5e82c0d880923425712b2bb658b9196932e02c78b2582fe742daa28
y	32879423ab1a0375895786c4bb46e9565fde0b5344766740af268adb32322e5c
Коэффициент k1	
k	e8c6740e58d616ca220db7da0d9c3e19b53e86e38bf3e8747774631452ec174c
Точка B1 = A1*k1	
x	715dc6983eb402d2abdc9ef0a20f4ba8f91bb1998992fecb8d80d6a3422d0588
y	d4e78f07d0473de3ee38f856199b4dc9ba4a87c263bb53938ef21ccef37203fe
Контрольный пример 2	
Точка A2	
x	1d40c1676805f9518be1fb4c7ae460d3608581e477b07c2d0e7e1e265a6b3347
y	8291ace380fd8832baca29613ab5626c302d13348f204d727d30897a8e1f8934
Коэффициент k2	
k	c9cbaeefaabc51147130fc6fa1adbe72140e35c5911b7d54b12beecdf5848943
Точка B2 = A2*k2	
x	1dfbf03073736c14453519f06fe712a3814f954168878907e02b9700678e3de1
y	e8c641cca3f7a197ebf8e4f4f85a64f0842a2b74de1627443d29ee63e1e8ca28

В случае успешного выполнения контрольных тестов, необходимо в соответствии со своим вариантом, выполнить умножение точек точек A и B с координатами (x,y) в форме Вейерштрасса на предложенный коэффициент. Варианты заданий представлены в таблице 9.

Таблица 9 – Варианты заданий для работы «Реализация алгоритма умножения точки эллиптической кривой на число»

Вариант 1	
Точка A	
x	47615cd873812de194b6db9ba05c88bc7d69e978ec062b020c0a98b5bd6568bb
y	ff0563dcc219632ad7df0dcdde9882b2e34c6fc2077f22280679422afbfa08e3
Точка B	
x	c7ca518e3ef9ad61579287c9523fc9a91c9aead6f36d7ef0e49e9314edfab1b4

Продолжение таблицы 9

y	b248bee0b70825b311795b8ea2030f7a8d210b87b4216859784722261438b028
Коэффициент	
k	2d6f3c230e6f7ebb284521bb80c8b8fe6e7ebd471a817a078fed8d207a450374
Вариант 2	
Точка А	
x	e8fb19165d79fbc24f55f0b1e33d8a71be0226be05294e49caf388b19fb1d0d2
y	dbcbce59da39a13596428feae904e8de8ce929b481d267bed50d5751f1be2169
Точка В	
x	5fcbfe0130206400e2ef3054f5486d2760277bcb2a502e9f3bd18846331fca2d
y	b584cbd2fbf014defdf15037c268444dda3e879a55f4df315c8b1e4ab37220c1
Коэффициент	
k	953abb37ce138a0613f36df6e7d966a69ee06569a38e6dcf6989e94ba4e410fd
Вариант 3	
Точка А	
x	5e37edf83f668a1d344828d8e482c13e1a451e98187283828b10861138c8a49b
y	3325fa691287835a92299e695063d097ea5539599fd3853eabb6747f37f50d10
Точка В	
x	ed513616221958938a69e12b2be16629f0532c1280503b5a7a5c4bd2272e6cb1
y	535f28050ee88df27430f1deca80ae7131a51b07a923a35160868e1015aef47e
Коэффициент	
k	72e8ad16f7862e67eb195ad225079f1700c9680a52eb66be3621f3c58722e051
Вариант 4	
Точка А	
x	6891696e0f20268f806d5b6fc7de3646971b7f447f09cd08b338edc464ae5cc6
y	c21bb1477bd52489e61240b12f80af22424efa0af43ed5de106f29d088bf383c
Точка В	
x	264836695a460dae5a1c8e7a342c4a4737ec10138ed425ffccb52d29708f5e62
y	3ef34361604ee65df5caad6cdd41970e2d73df8383325b511314db24c748c6d1
Коэффициент	
k	3eb368c026c1ab670b462dccb5d7fca291c5c12e928bab65288619c7159f887a
Вариант 5	
Точка А	
x	3e9f00cb35367f8c6558e17ec641ec0db3f6aa4eb87359b38ef9a27ea5436587
y	a6245c8d7cac688a3855351f8b9f96d58305e37b48189061c3ce49e2c0bd5ffd
Точка В	
x	dabde3d3cdf37b92712d0bf66ae5e2160b4841c9deaf2198c1868fbe4c4c2524
y	fb29a174cf79d33d8d0fee713750fc7b1d4a0b6086a2182c8c6b2879f21a9c98
Коэффициент	
k	0e950fb2a3e27042f85006e8516398379ff24949674218b9812acf3087ca6c43
Вариант 6	
Точка А	
x	1a4e6a6810a2f6b2b77dd40976fc5ff09ba88301ab6f6ab60011302aa4640657
y	b27fd1e6e590874fb8928f1785043f5943b75ccec6098cd83405311f96d8fddf

Продолжение таблицы 9

Точка В	
x	fb7e76ffc26be53e9858bb15e3c4580f287511bbb72a718b417853d16a83fba9
y	8bdc9fee69b7903c98f8761967173b54dfd976c7282d53446255e10e913709e8
Коэффициент	
k	8156cf52bcb8e5e08aab8a8d324cfddc3afbe738407d7da0877c590f0ee701a4
Вариант 7	
Точка А	
x	c5f1f9ed5fb856b7a13793ae1f6a42321cce4bf4594e50cef853d766c41a4bab
y	219e5871df01fb7bca92887a446fd680b94908ab1405bd8c5135b72f38ac62d9
Точка В	
x	84fa08690151695409788989f3d21340a72a6d820e02a33aa1787b56d3a0c795
y	071aa92c8a2cb0438ef722d21849299552f5d58a5830f26ad79ad65e3f8481ee
Коэффициент	
k	9092592667a67d3f10f70625a54dac1f7d66af3a12f38845d9159232607b3df9
Вариант 8	
Точка А	
x	b216b37b9cc0f195906e3432c2a29c9f9a62ad46656bb509ac21801c1b03af1e
y	41a41af5f12805c26877356b6744cc1535f9f985021be36de6ae53e6d764a6b4
Точка В	
x	7b104442f1af36cf5685cfc7684c7c1fa240edfcdcf053f71d7224f85aa6e8141
y	45aabc4fb72b218b8f81044d5e839a64e65046d3514f666e6c3917949b8903d7
Коэффициент	
k	3399b9b2537b96450162b9ebe3d531a14b7e2521f1562bc2ec1be73583b483bf
Вариант 9	
Точка А	
x	182f29adadafa136a9635ed9d2688245885eece8b6327cd67b87d7e099243d23
y	d7f1e9c9c09f80c8d201c47524095b9cd0571e969eedd3154ba59fb7db710f7
Точка В	
x	10bb1dc7432010dc5d14d40ad6b13c6fb06de817e0aa2589104ab4e72b2a5f92
y	dd0debfb02ebb15aa7b04a810caf9022f6151e5b3654842de8a9f513de70a7b43
Коэффициент	
k	99f629b4ae1af1633f91071252f3b2c0060d207b59b0a190dc1755e673794587
Вариант 10	
Точка А	
x	b216b37b9cc0f195906e3432c2a29c9f9a62ad46656bb509ac21801c1b03af1e
y	41a41af5f12805c26877356b6744cc1535f9f985021be36de6ae53e6d764a6b4
Точка В	
x	1b92f7a3dbdc941dbb844197a79815cd4c93196100ea4fd3e226b9dd7597a3fe
y	394d654fc8303797fd38d4d80aabebe0d4e7b1baabf66a6ca057c0df4b3366c4
Коэффициент	
k	255249b3f4de52d6977b3bfec0465a2625adb0b64d86d4444f1fb86d6820cafb

Продолжение таблицы 9

Вариант 11	
Точка А	
x	d405ed33759057b77ce6827cb6b7dc2bbc1a37f18eb24931917a8caa8ed0e0b3
y	430f8208ddf32a70d6fca6bce899679046bfcc1d51bd5c9abf49a9ab3ea990b9
Точка В	
x	dabde3d3cdf37b92712d0bf66ae5e2160b4841c9deaf2198c1868fbe4c4c2524
y	fb29a174cf79d33d8d0fee713750fc7b1d4a0b6086a2182c8c6b2879f21a9c98
Коэффициент	
k	6a82a81fec59f302af897b1740064455dca1f27500365e9ec6c2e1be04a65eee
Вариант 12	
Точка А	
x	264836695a460dae5a1c8e7a342c4a4737ec10138ed425ffccb52d29708f5e62
y	3ef34361604ee65df5caad6cdd41970e2d73df8383325b511314db24c748c6d1
Точка В	
x	c9cbaeeffaabc51147130fc6fa1adbe72140e35c5911b7d54b12beecdf5848943
y	3c2e34cdfale6e76d6cc57ec871a26fb5b23d01c540e6f0d8f77f4d81fe3f613
Коэффициент	
k	b1f997403ae8b20011c658d92d63b4aa178f94c5e104f275c5f08c164d93cefc
Вариант 13	
Точка А	
x	3508bea1bde4908b6c2572d32da08ff7f5741780e20e0a966e0f4ea20eac30b5
y	45e1798bc02b54c4dabcb8233c66eb778acd6dc44b2a21b140d22b7a8a49ac8f
Точка В	
x	182f29adadafa136a9635ed9d2688245885eece8b6327cd67b87d7e099243d23
y	d7f1e9c9c09f80c8d201c47524095b9cd0571e969eedd3154ba59fb7db710f7
Коэффициент	
k	b7e2e16f9951ce2b132b6f9b947606a826db96a81895883871941d6c0a8980d1
Вариант 14	
Точка А	
x	c7ca518e3ef9ad61579287c9523fc9a91c9aead6f36d7ef0e49e9314edfab1b4
y	b248bee0b70825b311795b8ea2030f7a8d210b87b4216859784722261438b028
Точка В	
x	3c9c274336b21af8a659d6360d27bf754fdbbc388ca09482916d41c2ac218bc8
y	eb073ac056d069bd0fea34bd1db2a52204256091c3e2a65e53066a832aa463b9
Коэффициент	
k	2b7e6e75880182213071bc4e26c92a9fbecaf7faecc20539e37bc4adfb450efd
Вариант 15	
Точка А	
x	e8c6740e58d616ca220db7da0d9c3e19b53e86e38bf3e8747774631452ec174c
y	0b837a5e560a29a2327b575f29b4be8baef4bc947fcc2ed4f3264bc434309381
Точка В	
x	a6529203be8b7be4eb5bc8d5bb2039e7f30cd5f96080afc441d68be27a0ca8f8
y	5e984e973dd41eba318ec4f6621d1224ba3cd7ce00228f794f00865bb8928dda

Продолжение таблицы 9

Коэффициент	
k	2916f431b3fff74511c2d1563ccf5076ea4284153ff7d6c67268849996a25ba5
Вариант 16	
Точка А	
x	91e38443a5e82c0d880923425712b2bb658b9196932e02c78b2582fe742daa28
y	32879423ab1a0375895786c4bb46e9565fde0b5344766740af268adb32322e5c
Точка В	
x	f3d579efdb90172f52f9db2996cbc45f1b70fb3042c1c38717d995c5df469397
y	9268e964b814d81b031f66aa309fdd3ccb9be1620cc58c142e4906f0a5809ec1
Коэффициент	
k	212595b0c6c89132259f02198654f41cab9c6039233dc97e22940d5a35586c33
Вариант 17	
Точка А	
x	fb7e76ffc26be53e9858bb15e3c4580f287511bbb72a718b417853d16a83fba9
y	8bdc9fee69b7903c98f8761967173b54dfd976c7282d53446255e10e913709e8
Точка В	
x	7f60f457bd3e3b12e90da792a7c7eb680b36a88bb399c571653adb54bc0aaf2d
y	b3aba99266d64b30b2dc7e121081b1dc32be2293cf47ff402ae64361ebdcc1fd
Коэффициент	
k	a254316f8969b568fcb7ed43cc3a0531f47061e8aa05054e29cdb6fe67fa5b15
Вариант 18	
Точка А	
x	84fa08690151695409788989f3d21340a72a6d820e02a33aa1787b56d3a0c795
y	071aa92c8a2cb0438ef722d21849299552f5d58a5830f26ad79ad65e3f8481ee
Точка В	
x	96a6b56a4bde04e3623fdf7a2920c22c72d61cff83273c996773f6eefa2b16b9
y	d3931048534475a1efd3208e185601b4c07ef3baaf1db673d65814f9d16af6d1
Коэффициент	
k	48d634db1a57ce425ed89bb6c04a43fbfa836b48f468d060660958a6effa95c0
Вариант 19	
Точка А	
x	644c9718f80b40d73557912bf11e538b4b4c1afdd011b3b54c9fbe67011f60d9
y	298410a9d8b4d12c131eca5ac158a14a96802b112ef903615777375b218949da
Точка В	
x	097254064117c5f60349be07d28537bc7818aaf125538cf58ae1f897adabb273
y	93b2f78fa5acad7526b8c703519b5b64846a60243b8ca7a9e42e9f9b00cdbccc
Коэффициент	
k	a6861d1a4bb9c93afd9a638b46b8341fe9b01fb4af6af7e396b710636c932d32
Вариант 20	
Точка А	
x	7c62f4ff84427d9c84074d745167bf0a1e1b5b2fa400836b4b991bec29bf999f

Продолжение таблицы 9

y	08ee3d6e617b30d21acddea988a09a02e8fae6fe1b498e2efa951e9eb1b0f2
Точка В	
x	4afc8d2a703bd0ed2f55f741fc12ad71b8cfd2d78b70758cbf844ce79e83781f
y	fc5c10486f70c07cffe06f1c64a094028a574e85ac112cefb94a0772d2381fec
Коэффициент	
k	c679a161fe4411015024048db645d8e070c8c6c3c224ba8b4439df97aa0db4f9

5.6.2 Задания к работе.

Данная работа подразделяется на два уровня сложности: основной и углубленный.

Для обоих уровней сложности необходимо выполнить следующие задания:

— на языке программирования C++ написать функцию умножения точки на число с использованием скрученной эллиптической кривой Эдвардса. Текст разработанной функции представить в отчёте о выполненной работе;

— выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 8, и отразить результаты в отчёте о выполненной работе;

— согласно своего варианта, выполнить умножение точек A и B с координатами (x, y) в форме Вейерштрасса на предложенный коэффициент из таблицы 9. Предварительно выполнить описанные преобразования. Результаты отразить в отчёте о выполненной работе.

Для углубленного уровня сложности предназначены следующие дополнительные задания:

— на языке программирования C++ написать функцию умножения точки на число с использованием эллиптической кривой в форме Вейерштрасса. Текст разработанной функции представить в отчёте о выполненной работе;

— выполнить тестирование разработанной функции с помощью контрольных примеров, представленных в таблице 8, и отразить результаты в отчёте о выполненной работе;

— согласно своего варианта, выполнить умножение точек A и B с координатами (x, y) в форме Вейерштрасса на предложенный коэффициент из таблицы 9. Результаты отразить в отчёте о выполненной работе;

— сравнить полученные с использованием преобразований на кривой Вейерштрасса и на кривой Эдвардса результаты, сделать соответствующие выводы.

5.7 Реализация работы с ключевыми контейнерами

5.7.1 Основные сведения о работе.

Основная идея электронной подписи заключается в том, что подпись вычисляется владельцем на секретном ключе, принадлежащем только ему, а проверить подпись может любой пользователь, владеющий сертификатом открытого ключа. При таких условиях могут возникнуть следующие проблемы:

— сертификат открытого ключа может быть изменён злоумышленником;

— секретный ключ может быть похищен злоумышленником.

Для решения первой проблемы генерируемый сертификат является самоподписанным, что позволяет обеспечить целостность хранящейся там информации. Для решения второй проблемы необходимо использование криптографических методов, позволяющих обеспечить конфиденциальность секретного ключа. В этих целях и разработаны ключевые контейнеры. [9]

Структура ключевого контейнера представлена на рисунке 14. Пример ключевого контейнера представлен на рисунке 15.

Ключевой контейнер (* .dst)

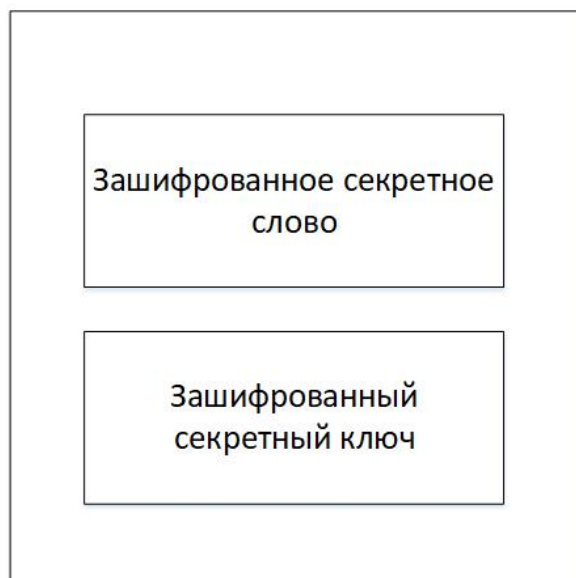


Рисунок 14 - Структура ключевого контейнера

Пример ключевого контейнера

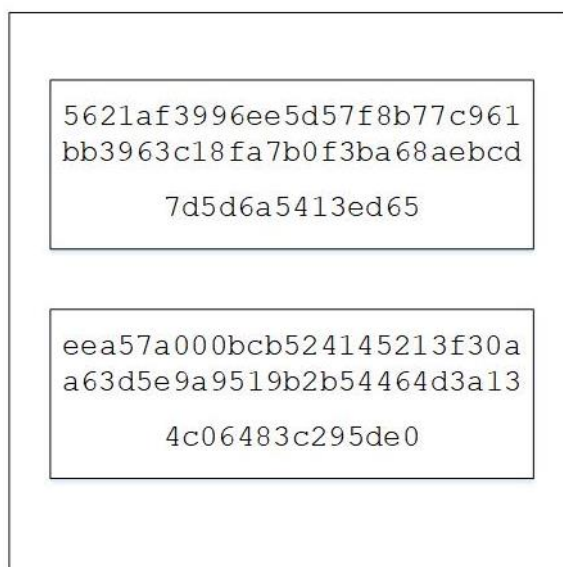


Рисунок 15 - Пример ключевого контейнера

Перед началом выполнения работ студентам должны быть выданы:

- логин, пароль и секретное слово;
- сертификат открытого ключа (файл с расширением «.pem»);
- ключевой контейнер (файл с расширением «.dst»).

В ключевом контейнере содержится два 256-битных числа, записанных на отдельных строках. Первое число – зашифрованное ключевое слово пользователя, а второе – зашифрованный секретный ключ.

На этом этапе выполнения работы в заголовочном файле «Elleptic_Curve.h» разумным является создать класс, который будет необходим для работы с электронной подписью. Пример объявления класса на языке C++ представлен ниже:

```
class Signature{
private:
public:
    Digit o_key_x[CD]; //открытый ключ подписи (x)
    Digit o_key_y[CD]; //открытый ключ подписи (y)
    Digit z_key[CD]; //секретный ключ подписи
    Digit hash[CD]; //хэш файла
    string login, password, secret_word; //логин, пароль, секретное
//слово
    void FormPodp(); //Формирование подписи
    bool ProvPodp(); //Проверка подписи
};
```

Алгоритм работы с ключевыми контейнерами представлен на рисунке 16.

Для корректной работы с ключевым контейнером, необходимо, чтобы он был расположен в одном каталоге с программой, в подкаталоге «Config».

Работу с ключевыми контейнерами необходимо организовывать внутри функции «*FormPodp()*», служащей для формирования электронной подписи.

При запуске функции с пользователя необходимо запросить логин, пароль и секретное слово. Затем сформировать путь к файлу с настройками.

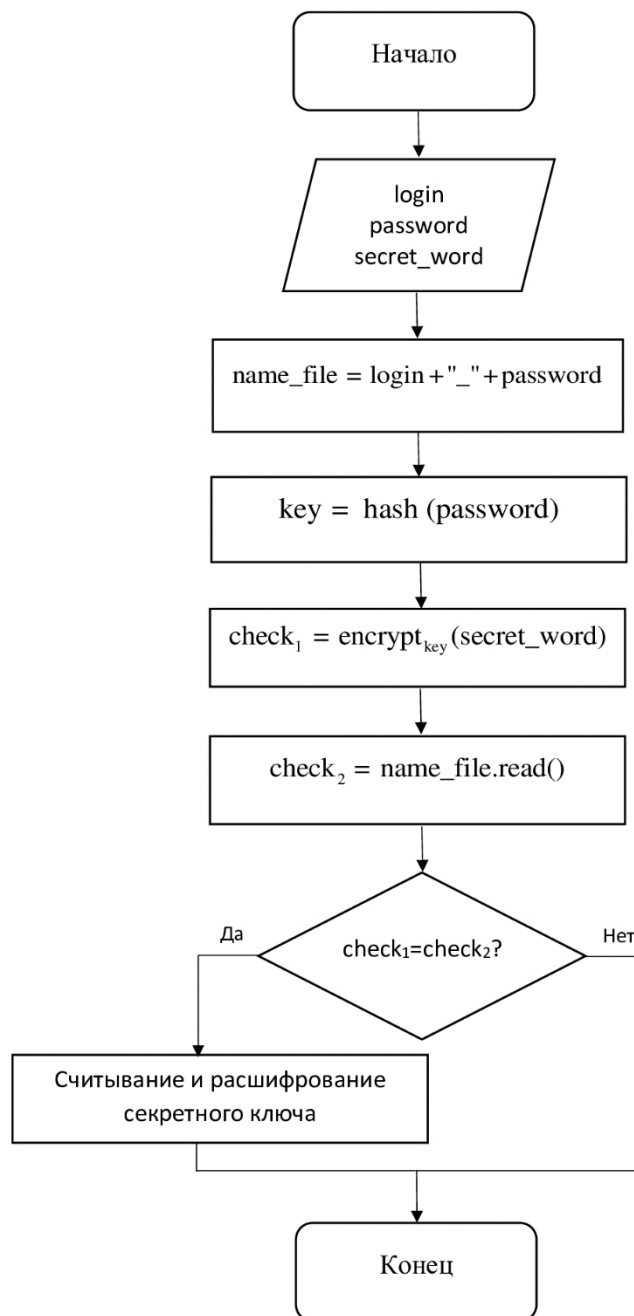


Рисунок 16 - Алгоритм работы с ключевыми контейнерами

Пример формирования пути к файлу на языке C++ представлен ниже:

```

char path1[MAX_PATH];
GetCurrentDirectory(sizeof(path1), path1); //Получаем текущую
//директорию
std::string path_in_string_conf = path1; //заносим это в строку
//Добавляем в пути папку "Config"
path_in_string_conf = path_in_string_conf + "\\Config\\";
//Создаём имя файла
name_file = path_in_string_conf+password+'_'+login + ".dst";
  
```

```
//Делаем файл записываемым  
ifstream file(name_file);
```

Далее необходимо вычислить хэш от пароля. Данный хэш будет являться ключом шифрования. Пример вычисления хэша от произвольной строки представлен ниже:

```
string stroka, hash;  
hash = Stribog::hashString(stroka); //Вычисление хэша от строки
```

На полученном ключе необходимо выполнить зашифрование секретного слова. Пример зашифрования на языке C++ представлен ниже:

```
string decrypted_string; //открытый текст  
string key; //ключ  
cin >> decrypted_string; //256-битное число, записанное в 16 с.с.  
cin >> key; //256-битное число, записанное в 16 с.с.  
ByteBlock bb_key = hex_to_bytes(key); //разбиваем ключ по байтам  
Kuznyechik a(bb_key); //создаём экземпляр класса для выполнения  
//зашифрования  
decrypted_string_1 = decrypted_string.substr(0, 32); //128-битное  
//число, записанное в 16 с.с.  
decrypted_string_2 = decrypted_string.substr(32, 32); //128-битное  
//число, записанное в 16 с.с.  
ByteBlock bb_decrypted_string_1 = hex_to_bytes(decrypted_string_1);  
//Разбиваем по байтам первую часть числа  
ByteBlock bb_decrypted_string_2 = hex_to_bytes(decrypted_string_2);  
//Разбиваем по байтам вторую часть числа  
ByteBlock result1, result2; //переменные для хранения результата  
a.encrypt(bb_decrypted_string_1, result1); //зашифровываем 1 часть  
//числа  
a.encrypt(bb_decrypted_string_2, result2); //зашифровываем 2 часть  
//числа  
string encrypted_string = (hex_representation(result1) +  
hex_representation(result2)); //Получаем результат
```

Полученное после операции зашифрования 256-битное число необходимо сравнить с первым 256-битным числом, считанным из ключевого контейнера.

Если полученные значения равны, то пользователь считается авторизованным, иначе авторизация считается неуспешной.

В случае, если данные введены корректно, необходимо считать из ключевого контейнера второе число, являющееся зашифрованным секретным ключом и расшифровать его.

После написания текста функции, необходимо протестировать её следующим образом: запустить программу и ввести свои логин, пароль и секретное слово, выданные преподавателем; программа должна обратиться к вашему ключевому контейнеру и выполнить проверку корректности введенных данных; если все описанные действия были выполнены верно, то авторизация должна завершиться успешно, в противном случае необходимо выполнить поиск и устранение ошибок в написанном коде и повторить процедуру проверки.

5.7.2 Задания к работе.

Данная работа не имеет уровней сложности. Все задания выполняются студентами в полном объёме.

Задания к работе:

- на языке программирования C++ написать функцию работы с ключевыми контейнерами. Текст разработанной функции представить в отчёте о выполненной работе;

- выполнить тестирование разработанной функции по алгоритму, описанному выше, и отразить результаты в отчёте о выполненной работе;

- выполнить умножение точек A и B, представленных в таблице 9, согласно своего варианта, на значение секретного ключа, полученного из ключевого контейнера. Результаты представить в отчёте о выполненной работе.

5.8 Реализация алгоритма формирования электронной подписи по ГОСТ Р 34.10-2012

5.8.1 Основные сведения о работе.

Алгоритм формирования электронной подписи по ГОСТ Р 34.10-2012 представлен на рисунке 17 [1].

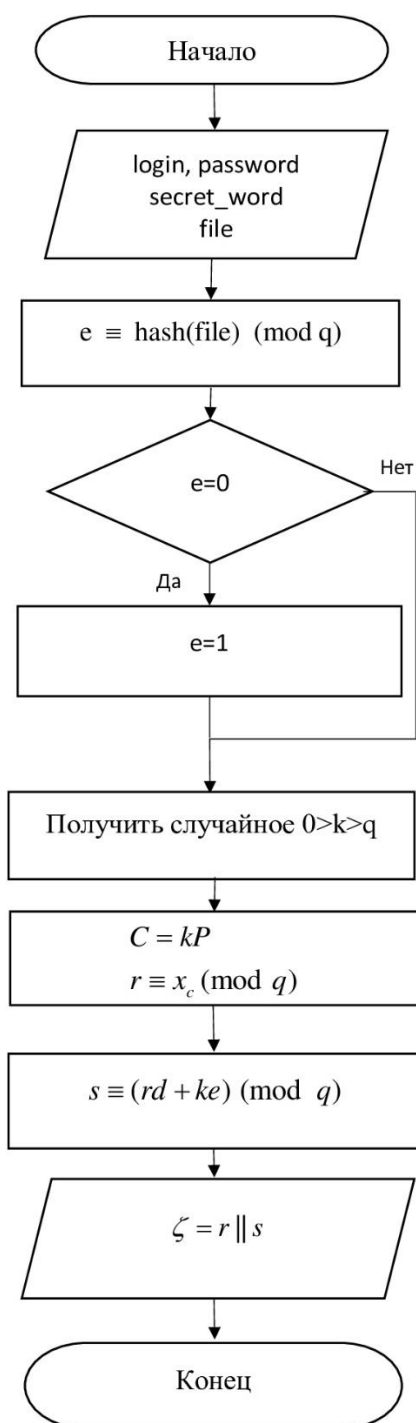


Рисунок 17 - Алгоритм формирования электронной подписи

Алгоритм формирования электронной подписи необходимо реализовывать в объявленной ранее функции:

```
void Signature::FormPodp()
```

Формировать подпись необходимо после того, как пользователь корректно введёт данные и секретный ключ считывается из файла в расшифрованном виде.

На первом этапе формирования подписи необходимо запросить у пользователя полный путь к файлу, электронную подпись которого необходимо вычислить. После этого считается хэш от этого файла. Пример вычисления хэша на языке C++ представлен ниже:

```
unsigned char str[8]; // переменная, в которую будут считываться
//данные из файла
string name_file_podp = "D:\\1.txt"; //Полный путь к файлу
ifstream file(name_file_podp); //создание файла ввода информации
Stribog buff; //создание экземпляра класса Stribog
while (file) //пока не достигли конца файла
{
    file.read((char*)&str, sizeof(str)); //считываем данные
    buff.update(&str, sizeof(str)); //обновляем содержимое буфера
}
string hash = buff.hash(); вычисляем хэш от информации
```

Полученные значения хэша необходимо представить по модулю q и результат записать в переменную e . Значение модуля q хранится в глобальной переменной g_q . Пример описанных действий на языке C++ представлен ниже:

```
Digit *e = new Digit[CD]; //Создаём динамический массив
Assig(hash, e); //Заносим хэш в созданный массив
while (More(e, g_q)) { // Пока  $e > g_q$ 
    Digit *z = new Digit[CD];
    Sub(e, g_q, z); //Выполняем  $e - g_q$ 
    Assig(z, e);
    delete[] z; }
```

В случае если e получилась равна 0, занести в неё 1.

Далее необходимо сгенерировать случайное 256-битное число k . Пример генерации на языке C++ представлен ниже:

```
for (int i = 0; i < CD; i++) //Вначале обнуляем число
    k[i] = 0x0000;
for (int i = 0; i < CD - 1; i++) //Затем заполняем случайными числами
    k[i] = (rand() + rand()) % 0x10000;
```

После этого необходимо выполнить умножение образующей точки P , координаты которой записаны в глобальных переменных g_u и g_v в форме Скрученной кривой Эдвардса, на значение сформированного коэффициента k . Координаты образующей точки в форме Вейерштрасса представлены ниже:

```
x = 91E38443A5E82C0D880923425712B2BB658B9196932E02C78B2582FE742DAA28
y = 32879423AB1A0375895786C4BB46E9565FDE0B5344766740AF268ADB32322E5C
```

Значение координаты x полученной точки по модулю q необходимо записать в переменную k .

Дальнейшие операции сложения и умножения длинных чисел производятся над полем Галуа порядка q . Для этого необходимо воспользоваться функциями *Mult_q* и *Add_q*. Пример получения числа s на языке C++ представлен ниже:

```
Digit *x1 = new Digit[CD];
Digit *y1 = new Digit[CD];
Mult_q(r, d, x1); //x1 = (r*d) (mod q)
Mult_q(k, e, y1); //x2 = (k*e) (mod q)
Add_q(x1, y1, s); //s = x1 + x2 (mod q)
```

При этом значения переменных r, k, e получены на предыдущих этапах вычисления электронной подписи, а значение d – секретный ключ пользователя, полученный из ключевого контейнера.

Значением подписи является конкатенация двух сток, являющихся 16-ричным представлением 256-битных чисел r и s .

Полученное значение подписи необходимо записать в файл, имя которого совпадает с именем введенного пользователя файла, а расширение файла – «.sig».

5.8.2 Задания к работе.

Данная работа не имеет уровней сложности. Все задания выполняются студентами в полном объеме.

— на языке программирования C++ написать функцию формирования электронной подписи по ГОСТ Р 34.10-2012 [1]. Текст разработанной функции представить в отчёте о выполненной работе;

— создать непустой файл с расширением «.txt» и именем, совпадающим с номером зачётки студента, и вычислить значение электронной подписи для созданного файла. Результат представить в отчёте о выполненной работе;

— создать непустой файл с расширением «.pdf» и именем, совпадающим с номером зачётки студента, и вычислить значение электронной подписи для созданного файла. Результат представить в отчёте о выполненной работе;

— создать непустой файл с расширением «.docx» и именем, совпадающим с номером зачётки студента, и вычислить значение электронной подписи для созданного файла. Результат представить в отчёте о выполненной работе.

5.9 Реализация алгоритма проверки электронной подписи по ГОСТ Р 34.10-2012

5.9.1 Основные сведения о работе.

Алгоритм проверки электронной подписи по ГОСТ Р 34.10-2012 представлен на рисунке 18 [1].

Пример объявления функции проверки электронной подписи на языке C++:

```
bool Signature::ProvPodp() ;
```

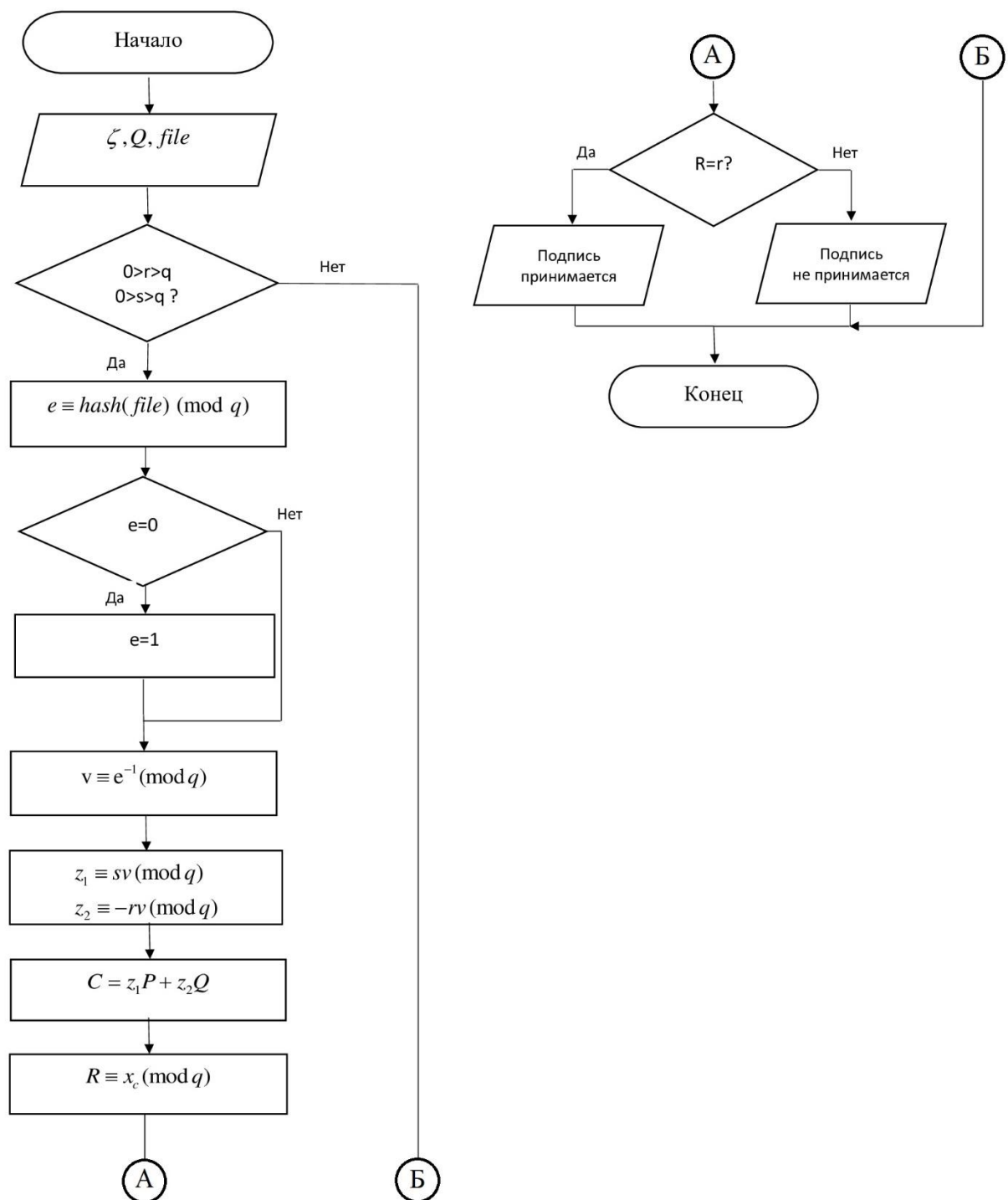


Рисунок 18 - Алгоритм проверки электронной подписи

Учитывая, что проверку подписи может осуществлять любой пользователь, владеющий сертификатом открытого ключа, то при вызове данной функции с пользователя не требуется запрашивать логин, пароль и секретное слово.

Информация, которую должен указать пользователь: имя файла, подпись которого проверяется (по имени файла необходимо сгенерировать имя файла,

содержащего электронную подпись с расширением «.sig»); полный путь к сертификату открытого ключа, откуда будет извлечён открытый ключ.

После того, как необходимая информация от пользователя будет получена, начинается процесс проверки электронной подписи. Из файла с расширением «.sig» считываются значения переменных r и s . Они должны лежать в диапазоне от 0 до q . Если это условие не выполняется, то подпись не принимается, иначе проверка идёт дальше.

Далее необходимо вычислить хэш файла, подпись которого проверяется. Вычисление осуществляется аналогично предыдущей работе. Полученное значение необходимо привести по модулю q и записать в переменную e . Если значение переменной e равно 0, то обратить переменную e в единицу.

Для элемента e вычисляется обратный элемент над полем с порядком q . Для этого необходимо создать функцию нахождения обратного элемента по модулю q . Вспомогательная функция при этом выглядит так:

```
// Вспомогательная функция для нахождения обратного элемента по
модулю q
void rev2(Digit prym[], Digit obr[])
{
    EVCL A; //Создаём структуру для нахождения обратного элемента
    Assig(prym, A.a1); //прямой элемент заносим первым аргументом
    Assig(g_q, A.a2); //Модуль заносим вторым и третьим аргументом
    Assig(g_q, A.a3);
    A = gcd_q(A); //Вызываем рекурсивную функцию
    Assig(A.a2, obr); //Второй элемент структуры - искомый
}
```

Объявление основной функции на языке C++:

```
EVCL gcd_q(EVCL X)
```

Текст основной функции идентичен функции нахождения обратного элемента по модулю p за некоторыми исключениями: умножение и вычитание чисел должно осуществляться по модулю q (используются функции *Mult_q* и *Sub_q*); при рекурсивном вызове функции необходимо вызывать новую функцию

gcd_q , а не gcd . Контрольные примеры для проверки корректности реализованной функции представлены в таблице 10.

Таблица 10 – Контрольные примеры для обратного элемента по модулю q

Прямой элемент	Обратный элемент (по модулю q)
2b6075cd06a5a3c7aebe4ec82e65f8fa4 4bdb635671c8383806fb26dc5041b4c	305234803634c00f75e73bca6ca28a3c8 c7adaca8b3e7f24bd87abe7221dd1cd
0d0d243259539bd1aba2736c70fc717cb a7d8629928a729204deb9eb170491a2	04289ea9a687f3ec1a06e240135e7d8ff 35944e583dcfc450676b8eb30710050
11f8cd98f4cf819ec83791af9bf96496c 93b66d596d798f91ac80583ab7739bc	23bb5366e4d233cf51e70534936fe920f 7388aaaffa557e297493bd1a3e11b6a

Далее по заданным формулам необходимо рассчитать коэффициенты z_1 и z_2 . Вычисления при этом ведутся по модулю q . Противоположный элемент (со знаком «-») можно найти по формуле ниже.

$$-rv \equiv q - rv \pmod{q}$$

Образующая точка P (была описана в предыдущей работе) умножается на коэффициент z_1 . Точка Q (открытый ключ подписи, извлеченный из сертификата открытого ключа) умножается на коэффициент z_2 .

Для того, чтобы произвести необходимые вычисления, нужно извлечь ключ из сертификата открытого ключа.

Формат сертификата представлен на рисунке 19. Пример сертификата представлен на рисунке 20.

Сертификат открытого ключа (* .pem)

```
----- BEGIN CERTIFICATE -----
1) Имя сертификата
2) Издатель
3) Код страны
4) Название области
5) Название города
6) Название организации

7) Фамилия владельца
8) Имя владельца
9) Отчество владельца

10) Версия сертификата
11) Серийный номер

12) Недействителен до
13) Недействителен после

14) Алгоритм открытого ключа
15) Открытый ключ (256 бит):
16) GostR3410-2012-256-Key
17) Подпись сертификата
----- END CERTIFICATE -----
```

Рисунок 19 - Формат сертификата открытого ключа

Пример сертификата открытого ключа

```
----- BEGIN CERTIFICATE -----
Имя сертификата      Сертификат открытого ключа
Издатель              Кафедра ИБСТ
Код страны             RU
Название области      Пензенская
Название города       Пенза
Название организации  ПГУ

Фамилия владельца     Myshenkov
Имя владельца         Nikita
Отчество владельца    Aleksandrovich

Версия сертификата    2
Серийный номер        44354

Недействителен до...  06:04:2021 9:38:00
Недействителен после... 06:04:2023 9:38:00

Алгоритм открытого ключа      ГОСТ Р 34.10-2012
Открытый ключ (256 бит):
GostR3410-2012-256-5956888fa237f9a1fda096able81aa8ed
b2b4cebd006f05523d48a88535f3cb1204d50fca3b17313b51b0
c4afede7e892f16cf548cb1bc546349d94564e1c641

7f60f457bd3e3b12e90da792a7c7eb680b36a88bb399c571653a
db54bc0aaf2db3aba99266d64b30b2dc7e121081b1dc32be2293
cf47ff402ae64361ebdcc1fd
----- END CERTIFICATE -----
```

Рисунок 20 - Пример сертификата открытого ключа

Открытый ключ подписи из сертификата можно получить следующим образом:

```
string sert, key; //sert - файл с сертификатом; key - открытый ключ
cout << "Введите полный путь к сертификату открытого ключа. При этом
в пути указывается два слеша" << endl;
cin >> sert;
ifstream file_sert(sert); //Объявляем файл
//Пропускаем 20 строк, на 21 находится ключ
for (int i = 1; i <= 21; i++) {
    getline(file_sert, key);
}
StringToNumber(key.substr(19, 64), o_key_x);
StringToNumber(key.substr(83, 64), o_key_y);
```

После считывания ключа из файла необходимо разово проверить содержимое переменных `o_key_x` и `o_key_y` и убедиться в корректности считывания.

Две полученные точки эллиптической кривой далее нужно сложить. Координату x результирующей точки привести по модулю q и сравнить со значением r , считанным из файла электронной подписи. Если значения подписи совпали, то подпись считается принятой. Если не совпали, то считается не принятой.

5.9.2 Задания к работе.

Данная работа не имеет уровней сложности. Все задания выполняются студентами в полном объёме.

— на языке программирования C++ написать функцию проверки электронной подписи по ГОСТ Р 34.10-2012 [1]. Текст разработанной функции представить в отчёте о выполненной работе;

— для файлов с расширением «.txt», «.pdf», «.docx» необходимо проверить сформированное в предыдущей работе значение электронной подписи. Результаты отразить в отчёте о выполненной работе;

- внести произвольные изменения в файл с расширением «.txt», не изменяя при этом значение электронной подписи, и проверить корректность сформированной подписи. Результаты отразить в отчёте о выполненной работе;
- вернуть все изменения в исходное состояние и выполнить повторную проверку сформированной подписи. Результаты отразить в отчёте о выполненной работе;
- внести произвольные изменения в значение электронной подписи, не изменяя при этом содержимое файла с расширением «.txt», и проверить корректность сформированной подписи. Результаты отразить в отчёте о выполненной работе;
- вернуть все изменения в исходное состояние и выполнить повторную проверку сформированной подписи. Результаты отразить в отчёте о выполненной работе;
- выбрать из числа студентов группы напарника. Необходимо обменяться с ним:
 - сертификатами открытого ключа;
 - двумя файлами с расширением «.txt» (один передать без внесения изменений, а в другой заведомо внести изменения);
 - файлом электронной подписи;
- выполнить проверку корректности сформированной электронной подписи для двух полученных файлов, используя сертификат открытого ключа напарника. Результаты отразить в отчёте о выполненной работе.

5.10 Реализация защитных мер

5.10.1 Механизм гарантированного затирания областей памяти.

В процессе работы программы секретный ключ хранится в программе в незашифрованном виде. Вследствии этого, после выполнения программы он может быть извлечён из памяти злоумышленниками. Аналогично может быть

извлечена другая защищаемая информация (значения логина, пароля и секретного слова пользователя и т.д.).

Чтобы этого не произошло, необходимо предусмотреть механизм гарантированного затирания областей памяти [9]. Он заключается в многократной перезаписи значений используемых переменных случайными величинами. В зависимости от типа переменной, в которой хранилась секретная информация, гарантированное затирание следует осуществлять разными способами.

Алгоритм гарантированного затирания областей памяти, используемых для обработки и хранения ключевой информации, представлен на рисунке 21.



Рисунок 21 – Алгоритм гарантированного затирания областей памяти

Пример механизма гарантированного затирания областей памяти, отводимых для хранения массива на языке C++ представлен ниже:

```
void Clear(  
    Digit x[] /* Затираемое число */  
{  
    for (int i = 0; i < 100; i++) { //100 итераций перезаписи  
        for (int j = 0; j < CD; j++) {  
            x[i] = (rand() + rand()) % 0x10000;
```

```

    }
}
}

```

Пример механизма гарантированного затирания областей памяти, отводимых для хранения переменной строкового типа на языке C++ представлен ниже:

```

void Clear(string& str) {
    const char * ar[] = { "a", "b", "c", "d", "e", "f", "g", "h",
        "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u",
        "v", "w", "x", "y", "z" };
    for (int i = 0; i < 100; i++) { //100 итераций перезаписи
        str = "";
        for (int j = 0; j < 64; j++) {
            str = str + ar[rand() % 26];
        }
    }
}

```

5.10.2 Механизм защиты программного средства электронной подписи от атак по времени выполнения.

Наряду с механизмом гарантированного затирания областей используемой памяти, необходимо реализовать механизм защиты программного средства электронной подписи от атак по времени выполнения [9].

В алгоритме вычисления электронной подписи базовой операцией является операция сложения точек эллиптической кривой. Она используется во всех остальных функциях и методах. Для того, чтобы обеспечить защиту от атаки по времени выполнения, необходимо в эту функцию внести случайные задержки времени, позволяющие увеличить энтропию процесса, что в свою очередь приведёт к значительному затруднению анализа времени выполнения сложения точек эллиптической кривой.

Алгоритм защиты от атак по времени представлен на рисунке 22.



Рисунок 22 - Алгоритм защиты от атак по времени

Пример внесения временных задержек на языке C++ представлен ниже:

```
int r_sleep = rand() % 2 + 1; //Задержка от 0 до 2 миллисекунд. 50%  
от среднего времени выполнения операции  
Sleep(r_sleep); //Вызов задержки
```

В представленном примере вносится случайная задержка от 0 до 2 миллисекунд, что эквивалентно половине от максимального времени выполнения операции сложения точек эллиптической кривой.

Данная мера замедлит время выполнения алгоритма вычисления электронной подписи, однако обеспечит защиту от атак по времени выполнения на программное средство.

5.10.3 Задания к работе.

Данная работа не имеет уровней сложности. Все задания выполняются студентами в полном объёме.

— на языке программирования C++ реализовать алгоритм гарантированного затирания областей памяти для различных типов переменных. Текст реализованной защитной меры представить в отчёте о выполненной работе;

— применить реализованную защитную меру для алгоритма вычисления и проверки электронной подписи. Результаты отразить в отчёте о выполненной работе;

— на языке программирования C++ реализовать механизм защиты программного средства электронной подписи от атак по времени выполнения. Текст реализованной защитной меры представить в отчёте о выполненной работе.

6 Обзор вредных психофизиологических факторов, влияющих на нервную систему специалиста по защите информации

6.1 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации.

Работа специалиста по защите информации является малоподвижной, так как связана с непрерывной работой за компьютером. Такие особенности влекут за собой присутствие вредных психофизиологических факторов, влияющих на здоровье работника. Необходимо обладать знаниями этих факторов и методами их минимизации для преждевременного предупреждения негативных последствий.

Постоянный сидячий образ жизни негативно влияет на нейроны головного мозга. При отсутствии физических нагрузок создаются новые связи между нейронами, по которым передаются дополнительные сигналы симпатической нервной системе, раздражающие её. Это приводит к нарушению нормальной работы данной системы. Так как симпатическая нервная система отвечает за функционирование сердечно-сосудистой системы, то нарушения в ней повлекут и возникновение сердечно-сосудистых заболеваний, к которым относятся:

- артериальная гипертония;
- ишемическая болезнь сердца;
- вегетососудистая дистония.

Зачастую специалисты по защите информации не соблюдают правила посадки перед компьютером. Это приводит к дегенеративно-дистрофическим заболеваниям позвоночника, таким как:

- сколиоз;
- спондилез;
- остеохондроз.

Недостаточная освещенность рабочего места и большое время работы за компьютером приводит к ухудшению зрения.

Учитывая особенности работы специалиста по информационной безопасности, можно также выделить следующие вредные факторы: нервное перенапряжение, умственное перенапряжение, монотонность работы.

Перечисленные факторы негативно влияют на здоровье работника и могут привести к заболеваниям, связанным с нарушениями в работе центральной нервной системы:

- синдром хронической усталости;
- депрессия;
- синдром выгорания;
- нервный срыв;
- обсессивно - компульсивное расстройство.

Выделенные заболевания проявляются в следующем:

- синдром хронической усталости - расстройство сна, постоянное желание уединения, беспричинная утомляемость;
- депрессия - апатия, раздражительность, пребывание в плохом настроении;
- синдром выгорания - истощение, пребывание в плохом настроении, отсутствие желания работать, раздражительность, неоправданная агрессия, перепады в настроении;
- нервный срыв - острые головные боли, непрекращающаяся усталость, резкие смены настроения, нарушения работы сердечно-сосудистой системы;
- обсессивно - компульсивное расстройство - постоянное появление неотступных, пугающих мыслей.

6.2 Методы и средства физической культуры, снижающие их воздействие

Любое заболевание легче предупредить, чем вылечить. Именно поэтому для специалиста по защите информации необходимо обладать знанием методов снижения воздействия вредных психофизиологических факторов на здоровье.

Работникам следует придерживаться следующих основных рекомендаций:

- осуществлять вечерние прогулки не менее 30 минут каждый день;
- отдавать приоритет использованию лестниц вместо эскалатора;
- по утрам делать небольшую 5-минутную зарядку;
- на выходных и во время отпусков заниматься активным отдыхом(катание на роликах, велосипеде, коньках, пробежки и т.д.);
- во время рабочих будней делать 2-минутную разминку после каждого часа работы;
- при возможности отдавать приоритет хождению пешком вместо езды на общественном или личном транспорте;
- знать и придерживаться правил правильной посадки перед компьютером;
- надевать специальные очки при работе с компьютером;
- постоянно проветривать помещение, в котором осуществляется работа;
- придерживаться оптимистичной точки зрения при выполнении трудовых функций;
- не выполнять трудовые обязанности сверх 8 рабочих часов в день.

При выполнении 2-минутной разминки можно использовать следующие упражнения:

- руки на поясе, вращение головой вправо и влево по 5 раз;
- руки на поясе, наклоны головой вперёд, назад, влево, вправо 5 раз;
- руки на плечи, вращение плечами вперёд-назад по 10 раз;
- руки перед собой, выполнение упражнения «ножницы» 10 раз;

- руки на поясе, вращение тазобедренным суставом в обе стороны по 5 раз;
- руки на поясе, наклоны вперёд, назад, влево, вправо 5 раз;
- руки на поясе, вращение голеностопным суставом по 5 раз каждой ногой;
- руки на коленных суставах, вращение коленями по кругу по 5 раз в каждую сторону;
- ноги на ширине плеч, наклоны руками до левой ноги, между ног, правой ноги по 5 раз;
- приседания 10 раз.

Заключение

В большинстве организаций для ведения электронного документооборота, электронной отчетности, осуществления электронных торгов используется электронная подпись. Количество операций, в которых задействуются криптографические преобразования, постоянно растёт. В связи с этим безопасное использование электронной подписи становится всё более актуальное темой.

Именно поэтому студенты специальности «Информационная безопасность» должны быть не только теоретически развиты, но и практически подкованы в данной сфере. Практикум по программной реализации криптографических преобразований на эллиптических кривых как раз нацелен на это.

Список используемых источников

- 1 ГОСТ Р 34.10-2012 Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – М.: Стандартинформ, 2012. – 33 с..
- 2 Болотов А.А., Гашков С.Б., Фролов А.Б., Часовских А.А. Элементарное введение в эллиптическую криптографию: Алгебраические и алгоритмические основы. – М.: КомКнига, 2012. – 356 с..
- 3 Болотов А.А., Гашков С.Б., Фролов А.Б. Элементарное введение в эллиптическую криптографию: Протоколы криптографии на эллиптических кривых. – М.: КомКнига, 2012. – 304 с..
- 4 Доступно о криптографии на эллиптических кривых [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/335906/> (дата обращения 15.04.2021).
- 5 Куляс М.Е. «Эффективная реализация элементарных операций в группе точек эллиптической кривой» // Материалы ||| Международной научно-практической конференции. Научно-образовательное учреждение «Вектор науки». – М.: Изд-во «Перо», 2015.
- 6 Скрученная кривая Эдвардса: Википедия. [Электронный ресурс]. https://ru.qaz.wiki/wiki/Twisted_Edwards_curvehttps://ru.qaz.wiki/wiki/Twisted_Edwards_curve (дата обращения 17.04.2021).
- 7 Бессалов А.В. «Производительность групповых операций на скрученной кривой Эдвардса над простым полем» / А.В. Бессалов, О.В. Цыганкова // Радиотехника. – 2015. – Вып. 181. – С. 58-63.
- 8 Рекомендации по стандартизации Р 50.1.114-2016 «Информационная технология. Криптографическая защита информации. Параметры эллиптических кривых для криптографических алгоритмов и протоколов». – М: Стандартинформ, 2016.

9 Методические рекомендации «Принципы разработки и модернизации шифровальных (криптографических) средств защиты информации». – М: Стандартиформ, 2017.