Оглавление

Введение

Цель работы – программная реализация криптосистемы шифрования с открытым ключом на основе эллиптических кривых.

Программа должна поддерживать следующие **функции**:

- возможность шифровать/расшифровывать короткие сообщения на случайных (или извлекаемых из выбираемых файлов) асимметрических ключах выбираемой пользователем длины;
- возможность сохранения случайной пары ключей в двух файлах с задаваемыми пользователем именами (закрытый ключ должен при этом шифроваться на ключе, выводимом из специальной парольной фразы);
- возможность определять при расшифровании закрытого ключа факт ввода неверной парольной фразы.

Для выполнения работы были поставлены следующие **задачи**:

– проектирование и реализация интерфейса программы;
– проектирование логики взаимодействия компонентов программы;
– реализация арифметических операций над точками кривой;
– реализация генерации ключей;
– реализация шифрования;
– реализация расшифрования;
– тестирование и отладка программы.

В **первой** главе будут приведены краткие сведения об эллиптических кривых и криптографии на них. Во **второй** главе будут описаны результаты проектирования программной реализации: созданные классы и методы. В **третьей** главе будут приведены результаты тестирования программы на предмет корректности работы алгоритмов, соответствия спроектированной реализации логике поставленной задачи.

При выполнении используется язык высокого уровня C# в среде Microsoft Visual Studio 2019, технология Windows Forms Microsoft .NET Framework.

1 О криптографии на эллиптических кривых

1.1 Ассиметричная криптография

Ассиметричное шифрование (или шифрование с открытым ключом) – асимметричная схема, в которой применяются пары ключей: **открытый ключ** (public key), который зашифровывает данные, и соответствующий ему **закрытый ключ** (private key), который их расшифровывает. Ключ, используемый для шифрования, может быть опубликован для использования всеми пользователями системы, которые зашифровывают данные. Для расшифрования данных получатель пользуется вторым ключом, являющимся секретным, и он не может быть определен из открытого ключа. Пользователь, владеющий открытым ключом, способен только зашифровывать данные, при этом он не может их расшифровывать.

Главное достижение асимметричного шифрования в том, что оно позволяет людям, не имеющим существующей договорённости о безопасности, обмениваться секретными сообщениями. Необходимость отправителю и получателю согласовывать тайный ключ по специальному защищённому каналу (проблема симметричного шифрования) полностью отпала. Все коммуникации затрагивают только открытые ключи, тогда как закрытые хранятся в безопасности.

На рис. 1.1.1 показана обобщенная схема асимметричной криптосистемы с открытым ключом. Здесь Ко – открытый ключ, Кс – секретный ключ получателя. Генератор ключей располагается на стороне получателя, так как это дает возможность не пересылать секретный ключ Кс по незащищенному каналу.

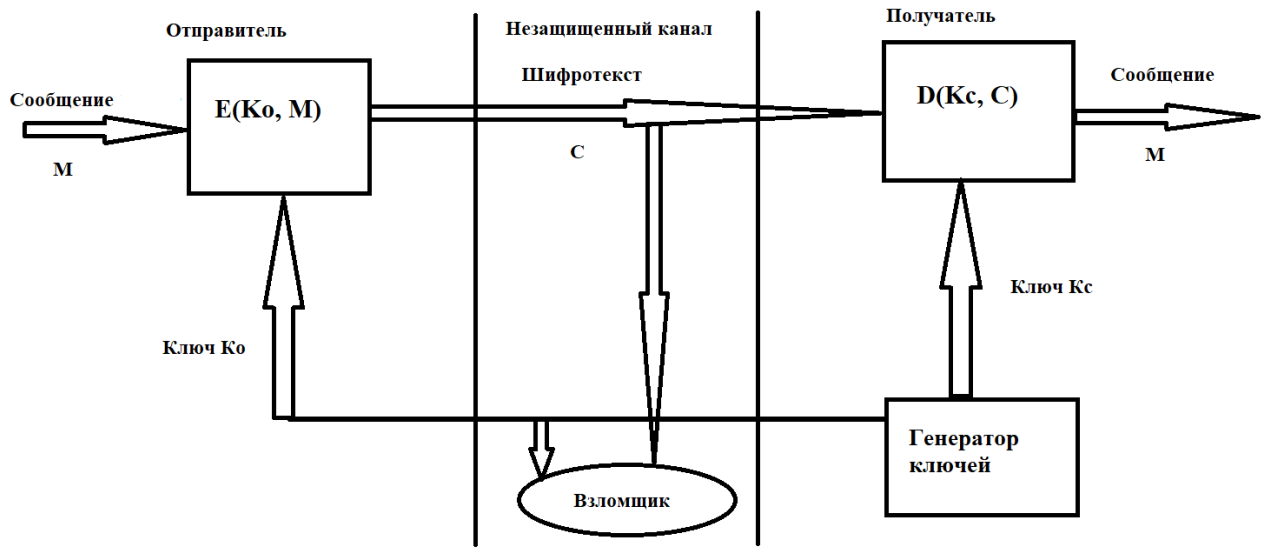


Рисунок 1.1.1 Обобщенная схема ассиметричной криптосистемы с открытым ключом

В зависимости от приложения отправитель использует либо секретный ключ, либо открытый ключ получателя, либо же оба, если требуется выполнить

какую-то специальную криптографическую функцию. В практике шифрования использование криптосистем с открытым ключом можно отнести к следующим категориям:

- зашифрование/расшифрование – отправитель шифрует сообщение с использованием открытого ключа получателя;
- цифровая подпись – отправитель «подписывает» сообщение с помощью своего секретного ключа. Подпись получается в результате применения криптографического алгоритма к сообщению или небольшому блоку данных, являющемуся хэш-функцией сообщения.

В данной работе будет реализован первый вариант системы.

Криптосистемы с открытым ключом имеют следующие особенности:

- шифротекст С и открытый ключ Ко могут быть отправлены по незащищенному каналу, т. е. третьему лицу могут быть известны только С и Ко;
- открытыми являются алгоритмы шифрования и расшифрования.

Защищенность информации (конфиденциальность) в ассиметричной криптосистеме основана на секретности ключа Кс.

Безопасность ассиметричной криптосистемы обеспечивается выполнением следующих требований:

- генерация пары ключей (Ко, Кс) должна быть вычислительно простой задачей;
- отправитель может легко вычислить шифротекст С сообщения М (зашифровать), зная открытый ключ Ко:

$$\mathrm{C} = \mathrm{E}(\mathrm{Ко}, \mathrm{M})$$

- получатель может легко восстановить М (расшифровать), используя закрытый ключ Кс:

$$\mathrm{M} = D(\mathrm{Кс}, \mathrm{C})$$

- при попытке вычислить закрытый ключ Кс третье лицо наталкивается на непреодолимую вычислительную проблему, даже зная открытый ключ Ко;
- при попытке вычислить исходное сообщение М противник наталкивается на непреодолимую вычислительную проблему, даже зная пару (Ко, С).

1.2 Особенности эллиптической криптографии

Эллиптические кривые являются одним из основных объектов изучения в современной теории чисел и криптографии. **Эллиптическая криптография** образует самостоятельный раздел криптографии, посвященный изучению ассиметричных криптосистем на базе эллиптических кривых над конечными полями (состоящих из конечного числа элементов).

Основное преимущество эллиптической криптографии заключается в том, что на сегодняшний день не известно субэкспоненциальных алгоритмов для решения задачи дискретного логарифмирования в группах точек эллиптических кривых (ECDLP). Порядок группы точек эллиптической кривой определяет сложность ECDLP. Считается, что для достижения такого же уровня безопасности как и в RSA требуются группы меньших порядков, что уменьшает затраты на хранение и передачу информации.

В приведенной ниже таблице 1.2.1. сравниваются приблизительные размеры параметров эллиптических систем и RSA, обеспечивающих одинаковую стойкость шифра, которая рассчитывается на основе современных методов решения ECDLP и факторинга (поиска делителей) для больших целых чисел.

Таблица 1.2.1. Размеры параметров эллиптических систем и RSA, обеспечивающих одинаковую стойкость шифра

Система на основе ЭК (длина базовой точки)	**RSA (длина модуля n)**
106 бит	512 бит
132 бит	768 бит
160 бит	1024 бита
224 бита	2048 бит
256 бит	3072 бита
384 бита	7680 бит
521 бит	15360 бит

Следовательно, использование эллиптических кривых позволяет строить высоко защищенные системы с ключами явно меньших размеров по сравнению с аналогичными “традиционными” системами. Хотя ЭК все больше и больше применяются в сфере защиты данных, разумеется, существуют и проблемы, которые ограничивают повсеместное распространение таких систем.

1.3 Эллиптические кривые над конечным полем

В общем случае эллиптическая кривая описывается уравнением вида:

$$y^2 + axy + by = x^3 + cx^2 + dx + e,$$

$$a, b, c, d, e \in \mathbb{R}$$

Общее уравнение эллиптической кривой с помощью замены координат приводится к канонической форме:

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

называемой *формой Вейерштрасса*. Причем $4a^3 + 27b^2 \neq 0$ для того, чтобы избежать особенностей.

В зависимости от значений a и b эллиптические кривые могут принимать на плоскости разные формы. Эллиптические кривые симметричны относительно оси X.

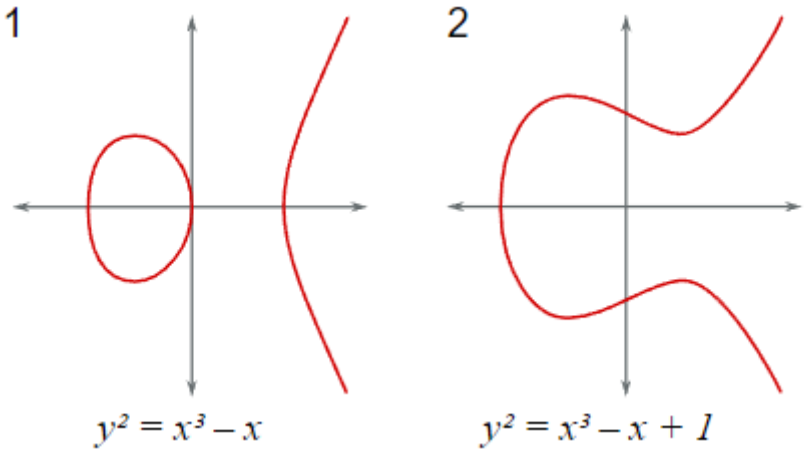


Рисунок 1.3.1 Эллиптические кривые разной формы

Определение эллиптической кривой уточняется следующим образом:

$$\{(x, y) \in \mathbb{R}^2 | \, y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0\} \cup \{0\},$$

где $0 \equiv (\infty, \infty)$ – бесконечно удаленная точка.

В криптографии с использованием эллиптических кривых приходится иметь дело с редуцированной формой эллиптической кривой, которая определяется над **конечным полем**. Конечное поле – это множество конечного числа элементов. Примером конечного поля является множество целых чисел по модулю p ($\mathbb{F}_p$), где p – простое число.

Ограничим эллиптические кривые полем $\mathbb{F}_p$:

$$\left\{(x, y) \in \left(\mathbb{F}_p\right)^2 \middle| \, y^2 \equiv x^3 + ax + b \ (mod\ p), 4a^3 + 27b^2 \not\equiv 0 \ (mod\ p)\right\} \cup \{0\},$$

где 0 – точка в бесконечности, $a, b \in \mathbb{F}_p$ – два целых числа в $\mathbb{F}_p$.

1.4 Деление по простому модулю p

Определим, что означает s/t над полем $\mathbb{F}_p$. Для выполнения этой операции требуется найти обратную величину числа t (t^{-1}), а затем выполнить простое умножение $s \cdot t^{-1}$.

Вычисление обратного числа выполняется с помощью **расширенного алгоритма Евклида**. Это очень важный алгоритм в эллиптической криптографии, ниже приводится его полное изложение.

Известно, что алгоритм Евклида может находить наибольший общий делитель (НОД) двух чисел t, m. Расширенный алгоритм Евклида не просто находит НОД, но и коэффициенты уравнения:

$$\text{НОД}(t, m) = t \cdot a + m \cdot b,$$

где a, b – коэффициенты Безу – целые числа со знаком.

Разложим следующую дробь: $\frac{t}{m} = \left\lfloor \frac{t}{m} \right\rfloor + \frac{r}{m} = q + \frac{r}{m}$, где $\left\lfloor \frac{t}{m} \right\rfloor$ – округленный вниз результат деления (результат целочисленного деления), обозначаемый через q (quotient), а r – остаток от деления. Выразим остаток от деления: $r = t - m \cdot q$.

Рассмотрим следующий пример построения цепной дроби:

$$\frac{13}{5} = 2 + \frac{3}{5} = 2 + \frac{1}{\frac{5}{3}} = 2 + \frac{1}{1 + \frac{2}{3}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2 + 0}}}$$

Действия продолжаются до тех пор, пока остаток не станет равным нулю.

Цепная дробь для частного случая, приведенного выше, в наших обозначениях примет вид:

$$\frac{t}{m} = q_0 + \frac{r_0}{m} = q_0 + \frac{1}{\frac{m}{r_0}} = q_0 + \frac{1}{q_1 + \frac{r_1}{r_0}} = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{r_2}{r_1}}}$$

$$= q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \frac{r_3}{r_2}}}}$$

Отсюда просто получить следующее представление:

$\frac{t}{m} = q_0 + \frac{r_0}{m} \rightarrow r_0 = t - m \cdot q_0$, где $q_0 = \left\lfloor \frac{t}{m} \right\rfloor$

$\frac{m}{r_0} = q_1 + \frac{r_1}{r_0} \rightarrow r_1 = m - r_0 \cdot q_1$, где $q_1 = \left\lfloor \frac{m}{r_0} \right\rfloor$

$\frac{r_0}{r_1} = q_2 + \frac{r_2}{r_1} \rightarrow r_2 = r_0 - r_1 \cdot q_2$, где $q_2 = \left\lfloor \frac{r_0}{r_1} \right\rfloor$

$\frac{r_1}{r_2} = q_3 + \frac{r_3}{r_2} \rightarrow r_3 = r_1 - r_2 \cdot q_3$, где $q_3 = \left\lfloor \frac{r_1}{r_3} \right\rfloor$

Обозначим $r_{-2} = t, a_{-2} = 1, r_{-1} = m, a_{-1} = 0$

Для любого индекса $k = 0, 1, \dots, K$ видна закономерность:

$\frac{r_{k-2}}{r_{k-1}} = q_k + \frac{r_k}{r_{k-1}} \rightarrow r_k = r_{k-2} - r_{k-1} \cdot q_k$, где $q_k = \left\lfloor \frac{r_{k-2}}{r_{k-1}} \right\rfloor$

Также легко получить закономерность и для коэффициент a Безу:

$a_k = a_{k-2} - a_{k-1} \cdot q_k, \forall k = 0,1, \dots, K$

Эти действия при программировании можно записать с помощью цикла, на каждой итерации проверяя значение остатка: если остаток сравнялся с нулем – выходим из цикла (при $k = K$). На выходе из него получаем значение НОД – r_{K-1} и $a = a_{K-1}$.

При поиске обратного числа t^{-1} в поле $\mathbb{F}_p$ в качестве входных данных алгоритм получает само число t и p. В силу простоты числа p НОД$(t, p) = 1$ и значение обратного числа $t^{-1} \equiv a \ (mod\ p)$.

1.5 Сложение двух точек

Для множества точек на эллиптической кривой справедливы групповые законы. Пусть P, Q, R – ненулевые точки ЭК, лежащие на одной прямой:

- **сложение**: $P + Q + R = 0$
- единичный элемент – бесконечно удаленная точка 0: $P + 0 = 0 + + P = P$
- у точки P есть обратная величина – это точка, симметричная относительно оси X
- ассоциативность: $(P + Q) + R = P + (Q + R)$
- коммутативность: $P + Q = Q + P$

Исходя из этих правил, можно вычислять сумму двух произвольных точек. Можно записать $P + Q + R = 0$ как $P + Q = -R$, где $-R$ – обратная величина точки R.

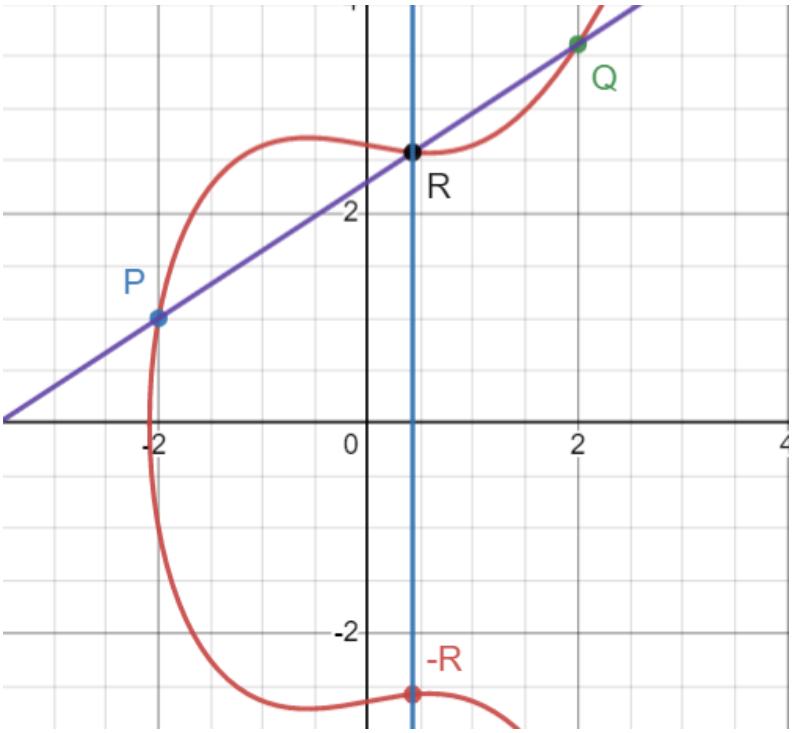


Рисунок 1.5.1 Проведена прямую через P, Q. Прямая пересекает R. Симметричная ей точка $-R$ является результатом $P + Q$

Если P и Q не совпадают ($x_P \neq x_Q$), то проходящая через них прямая имеет наклон:

$$\lambda = \frac{y_P - y_Q}{x_P - x_Q}$$

В случае если P и Q совпадают ($x_P = x_Q, y_P = y_Q$), то наклон прямой (касательной к кривой)

$$\lambda = \frac{3x_P^2 + a}{2y_P}$$

Пересечение этой прямой с эллиптической кривой – это третья точка R:

$$x_R = \lambda^2 - x_P - x_Q$$

$$y_R = y_P + \lambda(x_R - x_P) = y_Q + \lambda(x_R - x_Q)$$

Поэтому $(x_P, y_P) + \left(x_Q, y_Q\right) = (x_R, -y_R)$.

Если $x_P = x_Q, y_P \neq y_Q$, то $R = 0$.

В конечном поле $\mathbb{F}_p$ сложение точек сохраняет приведенные выше свойства:

- $P + 0 = 0 + P = P$
- для R обратная величина $-R = (x_R, -y_R \ mod \ p)$
- $Q + (-Q) = 0$

В $\mathbb{F}_p$ соотношения для R принимают вид:

$$\lambda = \begin{cases} \left(y_P - y_Q\right)\left(x_P - x_Q\right)^{-1} mod \ p, P \neq Q \\ (3x_P^2 + a)(2y_P)^{-1} mod \ p, P = Q \end{cases}$$

$$x_R = [\lambda^2 - x_P - x_Q] \ (mod \ p)$$

$$y_R = [y_P + \lambda(x_R - x_P)](mod \ p) = \left[y_Q + \lambda\left(x_R - x_Q\right)\right](mod \ p)$$

Приведенные выше уравнения работают над конечными и бесконечными полями (за исключением $\mathbb{F}_2$ и $\mathbb{F}_3$, в которых появляются особенности).

1.6 Скалярное умножение

Операция **скалярного умножения** точки P с помощью сложения определяется как

$$nP \stackrel{\text{def}}{=} \underbrace{P + P + \cdots + P}_{n \text{ раз}},$$

где n – натуральное число.

Эффективным методом реализации скалярного умножения на эллиптической кривой является **алгоритм удвоения-сложения**. Требуется представить число n в двоичной системе $n = b_t b_{t-1} \dots b_0$, где $b_i \in \{0; 1\}$, вычислить все точки $2P, 4P, \dots, 2^t P$ и подсчитать сумму тех точек $2^i P$, для которых $b_i = 1$.

Умножение точек на скаляр для эллиптических кривых над $\mathbb{F}_p$ обладает интересным свойством. Если взять некоторую кривую $E(\mathbb{F}_p)$ и точку $P \in E(\mathbb{F}_p)$, то все значения, кратные P, представляют собой конечное число различных точек, которые повторяются **циклически**. Остальные точки кривой $E(\mathbb{F}_p)$ никогда ими не становятся. Также при сложении двух значений, кратных P, получается значение, кратное P. Отсюда следует, что множество кратных P значений – это **циклическая подгруппа** группы, образованной эллиптической кривой. Точка P называется **генератором** или **базовой точкой** циклической подгруппы.

1.7 Порядок подгруппы

Количество элементов группы называется **порядком группы**. Обозначим порядок группы за N. Для эллиптической кривой, определенной над конечным полем, N – это количество точек этой кривой.

Порядок циклической подгруппы, порожденной генератором P (**порядок P**) – это минимальное положительное целое n, такое, что $nP = 0$. Порядок P связан с N теоремой Лагранжа, согласно которой порядок подгруппы n – это делитель порядка исходной группы N. n обычно выбирается простым.

Теорема Лагранжа также подразумевает, что число $h = \frac{N}{n}$ всегда целое. Число h называют **кофактором подгруппы**.

2 Проектирование программной реализации

2.1 Интерфейс программы

Реализованный способ организации графического интерфейса многооконного приложения – SDI. В программе организовано 5 окон: главное, окно “О программе”, “Генерация ключей”, “Шифрование”, “Расшифрование”.

В главном окне меню из трех пунктов: “Файл”, “Действия”, “Помощь”. Пункт “Файл” → “Выйти” – закрытие основного окна. Пункт “Помощь” → “О программе” – открывание диалоговой формы “О программе”. Пункт “Действия” → “Генерация ключей” – открывание формы “Генерация ключей”. Пункт “Действия” → “Шифрование” – открывание формы “Шифрование”. Пункт “Действия” → “Расшифрование” – открывание формы “Расшифрование”.

В окне “О программе” приведена краткая информация о проекте.

В окне “Генерация ключей” пользователь может выбрать или создать файлы для записей открытого и закрытого ключей. После выбора файла для закрытого ключа требуется два раза (второй раз для подтверждения) ввести парольную фразу генерации ключа шифрования для закрытого ключа. После выбора файлов и ввода парольной фразы пользователю доступен выбор длины ключа (размера генератора в битах). Установленное значение по умолчанию – 256 бит. Пара ключей указанного размера генерируется и сохраняется в указанных файлах нажатием кнопки “Генерировать и сохранить”, которая становится доступной после выполнения выбора файлов и выбора длины.

В окне “Шифрование” пользователь вводит сообщение для шифрования (по умолчанию в поле текст “Сообщение”; поле не может быть пустым – в случае, если оно пустое, шифрование не выполняется и выводится соответствующее предупреждение). Далее выбирается файл, в котором записан **открытый ключ**. После выбора файла с открытым ключом становится доступной кнопка “Шифровать и сохранить шифротекст”. После ее нажатия пользователь выбирает или создает файл для записи шифротекста. После выбора происходит шифрование и сохранение шифротекста.

В окне “Расшифрование” пользователь выбирает файл с шифротекстом. Затем выбирается файл с **закрытым ключом**. После выбора файла с закрытым ключом необходимо ввести парольную фразу, придуманную на этапе генерации. В случае неправильного ее ввода выводится соответствующее сообщение и форма закрывается. В случае корректного ввода также выводится соответствующее сообщение. В этом случае пользователю становится доступной кнопка “Расшифровать”. При ее нажатии сообщение расшифровывается и результат этой операции выводится в соответствующее поле.

Копии экранных форм, описанных выше, приведены в приложении А.

2.2 Класс ECurve

Алгоритмы эллиптических кривых будут работать в циклической подгруппе эллиптической кривой над конечным полем. Поэтому алгоритмам потребуются следующие параметры:

- **Простое число** P, задающее размер конечного поля.
- **Коэффициенты** A, B уравнения эллиптической кривой.
- **Базовая точка** G, генерирующая подгруппу.
- **Порядок** N подгруппы.
- **Кофактор** H подгруппы.

Соответствующие поля в классе являются публичными и инициализируются в конструкторе.

Публичный конструктор принимает единственный параметр – размер ключа. В зависимости от размера ключа выбирается одна из кривых:

- “nistp192” – размер 192 бита;
- “nistp256” – размер 256 бит;
- “nistp384” – размер 384 бита;
- “nistp521” – размер 521 бит.

Значения приведенных выше параметров для этих кривых **известны** и описаны в [5] – нет необходимости в их вычислении.

В классе также имеется метод, выполняющий проверку принадлежности заданной точки данной кривой. В случае, когда точка не принадлежит (ошибка в реализации какого-либо алгоритма), вылетает исключение. Этот метод удобен при отладке программы.

Полный код класса `ECurve` приведен в Приложении В.

2.3 Класс EPoint

В этом классе реализованы алгоритмы для выполнения арифметических операций над точками эллиптической кривой, описанные в предыдущей главе.

Конструктор принимает три параметра: значение координаты X точки, значение координаты Y точки и саму кривую (объект класса `ECurve`). В этом конструкторе инициализируются соответствующие поля класс `EPoint`.

Свойство `InfinityPoint` для нуля – бесконечно удаленной точки. Также есть метод проверки, является ли точка нулем – `IsInfinityPoint`, в качестве параметра принимающий проверяемую точку.

Метод `Modulus` находит остаток от деления, и случае, если полученный остаток меньше 0, то к нему прибавляется значение модуля.

Метод `ModularInverse` реализует расширенный алгоритм Евклида, который подробно описан в пункте 1.4. Проверяется, что значение, для

которого ищется обратная величина, ненулевое, что НОД равен одному – в случае, если это не так, выбрасывается соответствующее исключение – удобно при отладке программы.

Метод `ECAdd` реализует алгоритм сложения двух точек, который подробно описан в пункте 1.5. Проверяется, что обе точки расположены на одной кривой – в случае, если это не так, выбрасывается исключение.

Метод `Negate` находит обратную величину для заданной точки. Проверяется, что исходная и полученная точки расположены на заданной кривой.

Метод `ECMultiply` реализует алгоритм умножения точки на скаляр, который подробно описан в пункте 1.6. Проверяется, что исходная и результирующая точки расположены на заданной кривой – в случае, если это не так, выбрасывается исключение. Этот метод является публичным и статическим – он используется в алгоритмах шифрования/расшифрования.

Полный код для класса `EPoint` приведен в Приложении С.

2.4 Класс RandomBigInt

В этом вспомогательном классе реализован один публичный, статический метод, который используется при генерации закрытого ключа и шифровании. Этот метод находит псевдослучайное число в диапазоне $[1, N-1]$.

Метод необходим, так как для класса `BigInteger`, который в C# реализует операции с большими целыми числами, которые теоретически не имеют верхней и нижней границ, не существует встроенного генератора псевдослучайных чисел.

Тем не менее в методе используется класс `RandomNumberGenerator` пространства имен `System.Security.Cryptography`, который предоставляет функциональные возможности для создания случайных значений, в том числе и для `BigInteger`.

Код метода:

```
public static BigInteger randomBigInteger(ECurve curve)
        {
            byte[] unsignedBytes = new byte[] { 0x00 };
            byte[] randomBytes = new byte[curve.Length / 8];

            RandomNumberGenerator randomNumberGenerator =
RandomNumberGenerator.Create();
            randomNumberGenerator.GetBytes(randomBytes);

            byte[] positiveRandomBytes =
randomBytes.Concat(unsignedBytes).ToArray();
            BigInteger randomValue = new BigInteger(positiveRandomBytes);
```

```
            BigInteger result;
            do
            {
                result = randomValue % curve.N;
            }
            while (result == 0);

            return result;
        }
```

2.5 Реализация генерации ключей

При генерации пары ключей выполняется следующий алгоритм:

1) На основе выбранной пользователем длины ключа выбирается соответствующая кривая
2) Выбирается случайное число `privKey` (из диапазона $[1, N-1]$) – это закрытый ключ
3) Вычисляется открытый ключ `pubKey` = `privKey` $\cdot G$ (результат скалярного умножения)
4) Выполняется шифрование `privKey`:
   - к закрытому ключу добавляется специальная сигнатура для проверки правильности введенной парольной фразы
   - на основе введенной пользователем парольной фразы (и сгенерированной примеси) и алгоритма SHA-256 строится симметричный ключ шифрования
   - на основе симметричного алгоритма AES на сгенерированном в предыдущем пункте ключе строится шифратор, который шифрует закрытый ключ со спец. сигнатурой
   - в начало зашифрованного файла добавляется примесь, затем шифротекст
5) Открытый ключ `pubKey` записывается в соответствующий файл

Код представленного алгоритма:

```
 /* выбор кривой */
 ECurve crv = new ECurve(size);
 /* генерация закрытого ключа */
 BigInteger privKey = RandomBigInt.randomBigInteger(crv);
 /* вычисление открытого ключа */
 EPoint pubKey = EPoint.ECMultiply(crv.G, privKey);
 /* шифрование закрытого ключа */
 EncryptPrivateKey(privKey, private_key.Text);
 /* запись открытого ключа */
 File.WriteAllText(open_key.Text, pubKey.X.ToString() + " " +
pubKey.Y.ToString());
```

Полный код этапа генерации представлен в приложении D.

2.6 Реализация шифрования

Для реализации шифрования необходимо выполнить следующий алгоритм:

1) Считывается открытый ключ `Qa` и в зависимости от его размера выбирается кривая `crv`
2) Генерируется случайное число `r` (из диапазона $[1, N-1]$)
3) Вычисляется точка `rG` = `r` $\cdot$ G (результат скалярного умножения)
4) Вычисляется точка `S` = `r` $\cdot$ `Qa` (результат скалярного умножения), `S=(Sx, Sy)`
5) Сообщение `msg` представляется в виде числа `P`
6) Вычисляется шифротекст `Cm` = (`Sx+P`) $mod\ N$
7) Шифротексты `Cm` и `rG` записываются в файл

Код алгоритма:

```
/* выбор кривой */
ECurve crv = new ECurve(size);
EPoint Qa = new EPoint(x, y, crv);

/* генерация случайного числа */
BigInteger r = RandomBigInt.randomBigInteger(crv);

/* вычисление rG */
EPoint rG = EPoint.ECMultiply(crv.G, r);
/* вычисление S */
EPoint S = EPoint.ECMultiply(Qa, r);

/* преобразование текста сообщения msg в число P */
byte[] m = Encoding.UTF8.GetBytes(msg.Text);
BigInteger P = new BigInteger(m);

/* вычисление шифротекста */
BigInteger Cm = (S.X + P) % crv.N;

/* запись шифротекстов */
File.WriteAllText(shifr.Text, Cm.ToString());
File.WriteAllText(shifr.Text.Replace(".", "RG."), rG.X.ToString() + " " +
rG.Y.ToString());
```

Полный код этапа шифрования представлен в приложении D.

2.7 Реализация расшифрования

Для расшифрования потребуется следующий алгоритм:

1) Выполняется расшифрование закрытого ключа:
   - из начала зашифрованного файла считывается примесь

- на основе введенной парольной фразы (она может быть правильной или нет), примеси и алгоритма SHA-256 восстанавливается построенный на этапе генерации ключ
- на основе симметричного алгоритма AES на восстановленном в предыдущем пункте ключе строится дешифратор, который расшифровывает закрытый ключ со спец. сигнатурой

2) Проверяется сигнатура: в случае неверной сигнатуры (неправильная введенная парольная фраза) выводится сообщение об ошибке и форма закрывается

3) Считываются шифротексты `Cm` и `rG`

4) Выбирается кривая `crv` (размер ключа можно сохранить в зашифрованном виде вместе с сигнатурой и закрытым ключом)

5) Поскольку `privKey · rG = privKey · r · G = r · pubKey = r · Qa`, то можно восстановить `S = privKey · rG` (результат скалярного умножения), `S = (Sx, Sy)`

6) Восстанавливается `P = (Cm − Sx)` $mod\ N$

7) Число `P` преобразуется обратно в сообщение `msg`

Код алгоритма:

```
/* расшифрование закрытого ключа */
key_with_sign =
Encoding.Unicode.GetString(DecryptPrivateKey(pr_key.Text)).Split(' ');
/* проверка корректности введенной парольной фразы */
if (key_with_sign[0] == "SpecSignature") …
/* */

BigInteger privKey = BigInteger.Parse(key_with_sign[2]);

/* crv, Cm, rG */
ECurve crv = new ECurve(UInt32.Parse(key_with_sign[1]));

BigInteger Cm = BigInteger.Parse(Cmff);

EPoint rG = new EPoint(RGx, RGy, crv);

/* восстанавливается S */
EPoint S = EPoint.ECMultiply(rG, privKey);

/* восстанавливается P */
BigInteger P = (Cm - S.X) % crv.N;

/* P в текст msg */
msg1.Text = Encoding.UTF8.GetString(P.ToByteArray());
```

Полный код этапа расшифрования представлен в приложении D.

3 Тестирование и отладка

3.1 Этап отладки

На этапе отладки, при первом запуске реализованных модулей, появлялось множество непредвиденных ошибок. Огромную роль и неоценимую помощь в быстрой локализации и устранении ошибок сыграли многочисленные проверки и операторы обработки исключений.

3.2 Этап тестирования

После устранения всех ошибок, связанных с запуском программы, необходимо проверить, отвечает ли спроектированная реализация логике поставленной задачи – нет ли каких-либо багов.

Для этого требуется проверить две основные ситуации:

- ввод пользователем неверной парольной фразы – ожидаем, что форма “Расшифрование” должна экстренно закрыться
- при правильном вводе парольной фразы сравниваем исходный текст с полученным расшифрованным – аналогичное делаем для всех остальных длин ключей

Приступим к тестированию:

1. Открываем форму “Генерация ключей”. При первом вводе парольной фразы для шифрования закрытого ключа сразу тестируем ситуацию с некорректным повторным вводом. Видим предупреждающее сообщение:

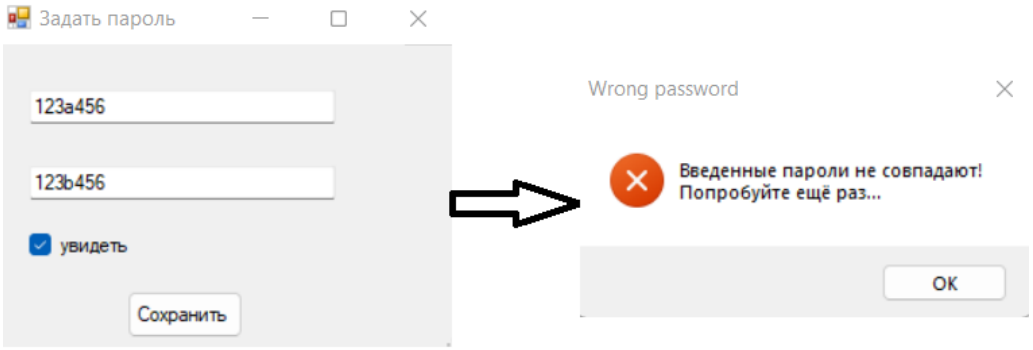


Рисунок 3.2.1 Предупреждающее сообщение о неверном повторном вводе

Вводим верную парольную фразу, выбираем длину ключа 192, генерируем и сохраняем ключи.

2. Открываем форму “Шифрование”. Здесь можно проверить ситуацию, когда поле сообщения остается пустым (по умолчанию оно содержит текст):

Рисунок 3.2.2 Предупреждающее сообщение об отсутствии текста

Вводим какое-нибудь сообщение и шифруем его.

3. Открываем форму “Расшифрование”. Вводим неверную парольную фразу:

Рисунок 3.2.3 Сообщение о неправильном вводе парольной фразы

Снова открываем эту форму и вводим правильную парольную фразу:

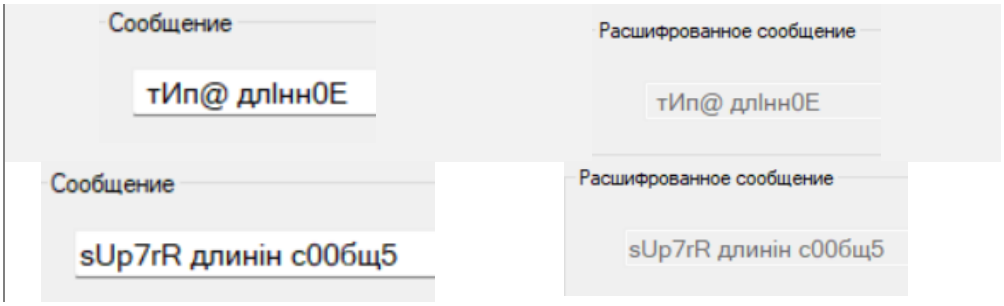
Рисунок 3.2.4 Сообщение о правильном вводе парольной фразы

4. Проверяем исходный текст с полученным расшифрованным:

Таблица 3.2.1 Таблица сравнения введенного текста с расшифрованным

Длина ключа	*Исходный текст*	*Текст расшифрованный*
192	Сообщение cOобщение1	Расшифрованное сообщение cOобщение1
256	Сообщение ght_сообщ№1	Расшифрованное сообщение ght_сообщ№1

384



521

Результаты тестирования соответствуют ожидаемым – тестирование можно успешно завершить.

На основе тестирования можно сделать вывод, что программа отвечает логике поставленной задачи.