**Московский Авиационный институт**

**(Национальный исследовательский университет)**

****

**Институт №8**

**«Компьютерные науки и прикладная математика»**

Кафедра 813

**«Компьютерная математика»**

**Курсовой проект по дисциплине «Основы криптографии»**

**Тема: «Разработка криптографического приложения на основе алгоритмов Camellia и ElGamal»**

Студент: Солдатов В.А.

Группа: M8О-311Б-19

Преподаватель: Романенков А. М.

Оценка:

Дата:

Москва 2022

Оглавление

1 [Введение 3](#_Toc105377722)

2 [Теоретическая часть 5](#_Toc105377723)

[Алгоритм Camellia 5](#_Toc105377724)

[Дополнение 11](#_Toc105377725)

3 [Практическая часть 14](#_Toc105377726)

[Общая структура программы 14](#_Toc105377727)

[Алгоритм работы с приложением 15](#_Toc105377728)

4 [Вывод 23](#_Toc105377846)

5 [Источники 24](#_Toc105377847)

6 [Приложение 25](#_Toc105377848)

[Файл key.cs 25](#_Toc105377849)

[Файл algorithm.cs 36](#_Toc105377850)

[Файл utility.cs 43](#_Toc105377851)

[Файл prime.cs 45](#_Toc105377852)

[Файл AuthenticationService.cs 48](#_Toc105377853)

[Файл FileStorageService.cs 50](#_Toc105377854)

## 1 Введение

1. Реализовать симметричный алгоритм шифрования.
2. Реализовать асимметричный алгоритм шифрования.
3. Реализовать приложение (оконное или web), позволяющее:
   * + Генерировать сеансовый ключ симметричного алгоритма;
     + Генерировать ключи асимметричного алгоритма в целях распределения между сторонами, участвующими в обмене данными, сеансового ключа (простые числа, требуемые при генерации ключей, должны иметь в битовом представлении размер не менее 64 бит и должны генерироваться вероятностными тестами простоты (Соловея-Штрассена, Миллера-Рабина, Ферма));
     + Генерировать вектор инициализации (IV) для его применения в режимах шифрования: CBC, CFB, OFB, CTR, RD, RD+H;
     + Асинхронно и многопоточно (если возможно) шифровать файл распределённым сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB) на одной стороне с последующей передачей ею зашифрованного файла (вместе с вектором инициализации) другой стороне;
     + Асинхронно и многопоточно (если возможно) дешифровать переданный зашифрованный файл распределённым сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB), с избавлением от набивки (padding);
     + Отображать прогресс операций шифрования и дешифрования при помощи элемента управления ProgressBar;
     + Опционально: отменить операцию [де]шифрования/передачи/ скачивания по запросу пользователя.

Передача файлов должна быть организована при помощи сервера, на который можно отправить зашифрованный файл и скачать его. На сервер одновременно можно отправлять или скачивать с него произвольное количество файлов. Для симметричного алгоритма используйте тип набивки (padding) PKCS7.

Симметричный алгоритм – Camellia

Асимметричный алгоритм – ElGamal

## 2 Теоретическая часть

* условиях поставленной задачи необходимо реализовать два алгоритма шифрования, симметричный алгоритм Camellia и асимметричный алгоритм

ElGamal. Остановимся подробнее на каждом из них.

## Алгоритм Camellia

Camellia — алгоритм симметричного блочного шифрования, один из финалистов европейского конкурса NESSIE (наряду с AES и Shacal-2), разработка японских компаний Nippon Telegraph and Telephone Corporation и Mitsubishi Electric Corporation, представлен 10 марта 2000 года. Сертифицирован японской организацией CRYPTREC как рекомендованный для промышленного и государственного использования алгоритм.

Основные характеристики:

* Допустимо использовать ключи размером от 128, 192 и 256 бит.
* Размер блока составляет 128 бит.
* Шифрование выполняется в 18 или 24 раунда, в зависимости от размера ключа.

**Шифрование:**

Перед первым раундом выполняется входное отбеливание данных— на шифруемый блок операцией XOR накладывается 128-битный фрагмент расширенного ключа kwe. Затем 128-битный блок данных делится на два подблока по 64 бита, после чего подблоки «прогоняются» через раунды шифрования. Между каждыми шестью раундами левый подблок обрабатывается функция FL, а правый — функцией FLI. Цикловая функция F использует нелинейное преобразование (S-блоки), блок линейного рассеивания каждые 16 циклов (побайтовая операция XOR) и байтовую перестановку.

По завершении последнего раунда подблоки меняются местами. Затем выполняется выходное отбеливание данных.

В каждом раунде левый подблок обрабатывается функцией F, которая использует 64-битный фрагмент ключа kn, и накладывается на правый подблок операцией XOR.

Структура алгоритма на примере варианта для 192-битного и 256-битного ключей приведена на рис. 1.

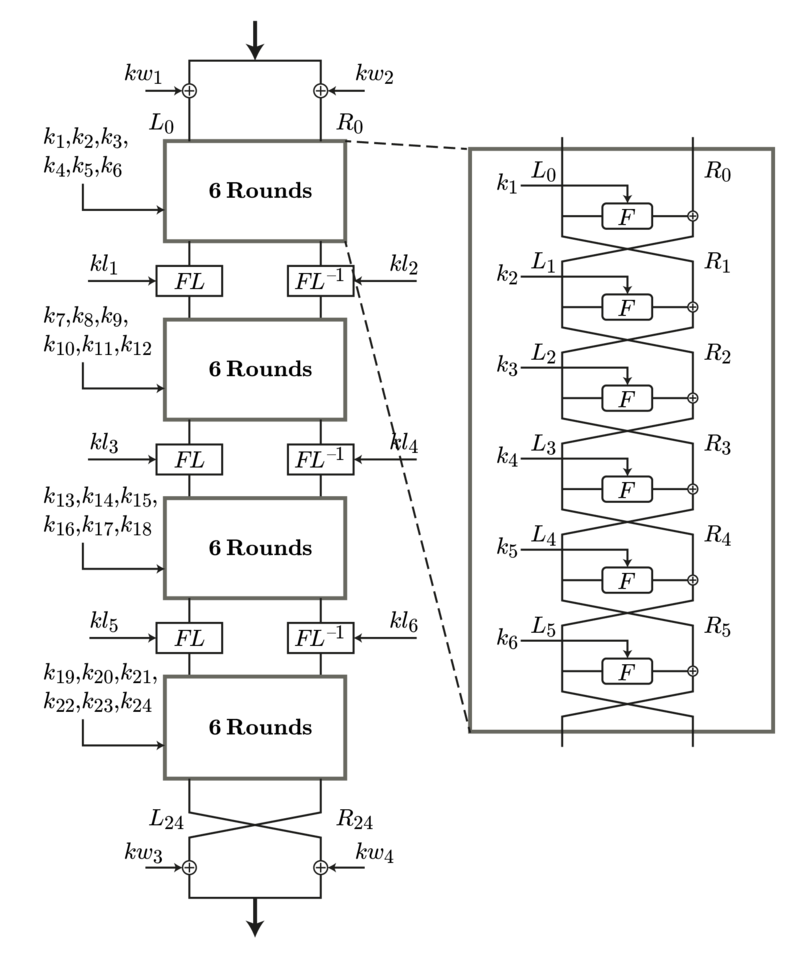


Рисунок 1: Структура алгоритма Camellia

**Расшифровывание:**

Расшифровывание данных алгоритмом Camellia выполняется полностью аналогично зашифровыванию, но с использованием фрагментов расширенного ключа в обратной последовательности, т. е.:

1. Подключ*кт)* используется при входном отбеливании, и наоборот, под- ключ*kwi* используется при выходном отбеливании данных;
2. В раундах шифрования ключи*k* применяются в обратном порядке*;*
3. В операциях*FL* и*FLI* четные фрагменты расширенного ключа используются при обработке левого подблока, нечетные — правого; и те, и другие берутся в обратной последовательности относительно зашифровывания.

**Процедура расширения ключа:**

Задача расширения ключа состоит в формировании необходимого количества фрагментов расширенного ключа (для перечисленных выше операций) из 128-, 192-битного или 256-битного исходного ключа шифрования*К.* Данная процедура состоит из нескольких этапов.

Прежде всего выполняется инициализация переменных*KL* и*KR* таким образом:

1. Для 128-битного ключа*KL~ К*,*KR* =0;
2. 192-битный ключ делится на 3 фрагмента по 64 бита, первые два из которых формируют*KL*; третий фрагмент и его побитовый комплемент формируют*KR;*
3. 256-битный ключ делится на 2 части: KL и КR.

На следующем этапе вычисляются еще две ключевые переменные:*КА*и*Кв* . Это выполняется с использованием функции F следующим образом (рис. 2):

1. Результат операции*KL XOR KR* дважды обрабатывается функцией*F*, в качестве фрагментов ключа которой берутся константы С1 и С2.
2. На результат предыдущей операции операцией XOR накладывается*KL.*
3. Снова дважды применяется функция*F* с использованием констант СЗ и С4 в качестве фрагментов ключа. В результате получается переменная*КА.*
4. В случае, если используются 192- или 256-битные ключи, необходимо также вычислить переменную*КB.* Для этого на*КА* операцией XOR накладывается*KR*, результат дважды обрабатывается функцией*F* с использованием констант С5 и С6.

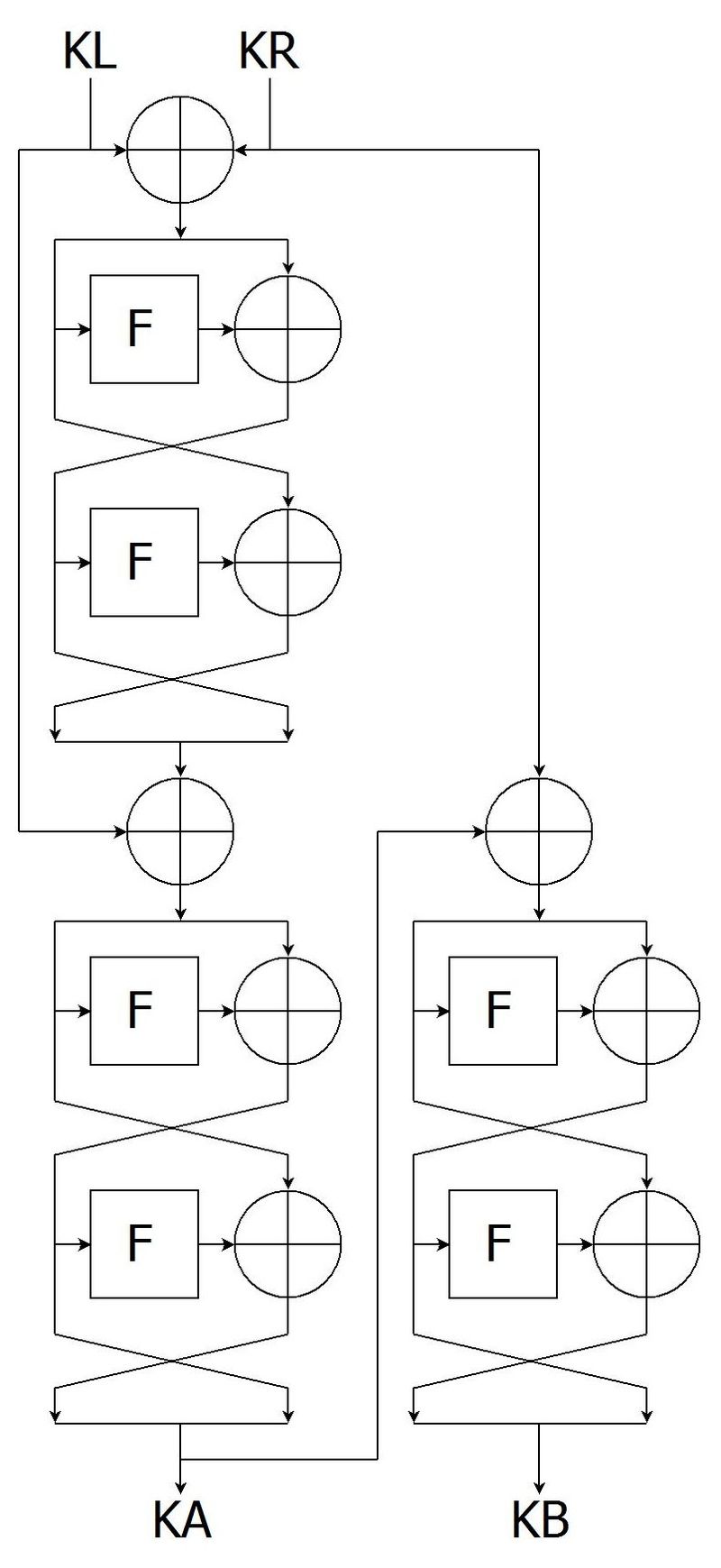


Рисунок 2: Вычисление KA и KB

Константы С1…С6 приведены в табл. 1 (указаны шестнадцатеричные значения).

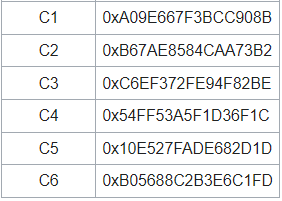


Таблица 1: Константы C

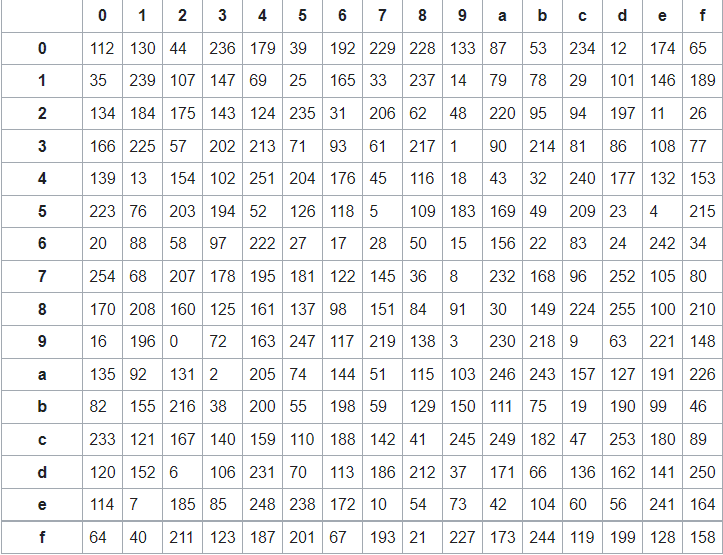


Таблица 2: Перестановки

**Алгоритм ElGamal**

Алгоритм Эль-Гамаля (ElGamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле.

**Генерация ключей:**

1. Генерируется случайное простое число p.
2. Выбирается целое число g — первообразный корень p.
3. Выбирается случайное целое число x такое, что (1 <𝑥 <𝑝 − 1).

4. Вычисляется y= 𝑔𝑥 𝑚𝑜𝑑 𝑝.

5. Открытым ключом является y, g, p, закрытым ключом — число x.

**Процесс шифрования:**

Сообщение M должно быть меньше числа p. Сообщение шифруется следующим образом:

1. Выбирается сессионный ключ — случайное целое число, взаимно простое с *(p – 1)*, k такое, что *1 < k < p – 1*
2. Вычисляются числа *a = gk mod p*
3. Пара чисел (a, b) является шифротекстом.

Нетрудно заметить, что длина шифротекста в схеме Эль-Гамаля вдвое больше исходного сообщения M.

**Процесс расшифрования:**

Зная закрытый ключ x, исходное сообщение можно вычислить из шифротекста (a, b) по формуле: 𝑀 = 𝑏(𝑎𝑥)−1𝑚𝑜𝑑 𝑝

При этом нетрудно проверить, что: (𝑎𝑥)−1 = 𝑔−𝑘𝑥 𝑚𝑜𝑑 𝑝, и поэтому

𝑏(𝑎𝑥)−1 = (𝑦𝑘𝑀)𝑔−𝑥𝑘 ≡ (𝑔𝑘𝑥𝑀)𝑔−𝑘𝑥 ≡ 𝑀 𝑚𝑜𝑑 𝑝

Для практических вычислений больше подходит следующая формула:

𝑀 = 𝑏(𝑎𝑥)−1 = 𝑏 ∙ 𝑎(𝑝−1−𝑥) 𝑚𝑜𝑑 𝑝

## Дополнение

Дополнение (англ. padding) в криптографии — добавление ничего не значащих данных к зашифровываемой информации, нацеленное на повышение криптостойкости. Различные техники дополнения применялись в классической криптографии, обширное применение техники дополнений нашли в компьютерных системах шифрования.

**PKCS#7:**

Дополнение в целых байтах. Значение каждого байта равно числу добавленных байтов, то есть добавляется N байт со значением N. Число добавленных байтов зависит от границы блока, до которого необходимо

расширить сообщение. Дополнение будет одним из:

Данный метод дополнения (так же как и два предыдущих) хорошо определён, только если N меньше, чем 256.

**Тест Ферма**

Если n — простое число, то оно удовлетворяет сравнению an-1=1 (mod n) для любого a, которое не делится на n.

Выполнение сравнения an-1=1 (mod n) является необходимым, но не достаточным признаком простоты числа. То есть, если найдётся хотя бы одно a, для которого an-1≠1, то число n — составное; в противном случае ничего сказать нельзя, хотя шансы на то, что число является простым, увеличиваются. Если для составного числа n выполняется сравнение an-1=1 (mod n), то число n называют псевдопростым по основанию a . При проверке числа на простоту тестом Ферма выбирают несколько чисел a. Чем больше количество a, для которых an-1=1 (mod n), тем больше шансы, что число n простое. Однако существуют составные числа, для которых сравнение an-1=1 n выполняется для всех a, взаимно простых с n — это числа Кармайкла. Чисел Кармайкла — бесконечное множество, наименьшее число Кармайкла — 561. Тем не менее, тест Ферма довольно эффективен для обнаружения составных чисел.

**Тест Соловея - Штрассена**

Алгоритм Соловея — Штрассена параметризуется количеством раундов k. В каждом раунде случайным образом выбирается число a < n. Если [НОД](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B8%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B9_%D0%BE%D0%B1%D1%89%D0%B8%D0%B9_%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C)(a,n) > 1, то выносится решение, что n составное. Иначе проверяется справедливость сравнения а(n-1)/2 = () (mod n). Если оно не выполняется, то выносится решение, что n — составное. Если это сравнение выполняется, то a является свидетелем простоты числа n. Далее выбирается другое случайное a и процедура повторяется. После нахождения k свидетелей простоты в k раундах выносится заключение, что n является простым числом с вероятностью 1-2-k .

**Тест Миллера - Рабина**

Как и тесты Ферма и Соловея — Штрассена, тест Миллера — Рабина опирается на проверку ряда равенств, которые выполняются для простых чисел. Если хотя бы одно такое равенство не выполняется, это доказывает, что число составное. Для теста Миллера — Рабина используется следующее утверждение:

Пусть n — простое число и n - 1=2s d, где d — нечётно. Тогда для любого a из Zn выполняется хотя бы одно из условий:

1. ad=1 (mod n)
2. Существует целое число r < s такое, что a2r d = -1 (mod n)

## 3 Практическая часть

## Общая структура программы:

Приложение состоит из серверной и клиентской части. Клиентская часть предназначена для:

1. Регистрации пользователей в системе сервера;
2. Отправки открытых пользователями файлов на удалённый сервер;
3. Получения файлов с сервера;
4. Получения информации о других пользователях сервера;
5. Шифрования файлов алгоритмом Camellia;
6. Шифрования файлов алгоритмом ElGamal;

Серверная часть приложения предназначена для:

1. Регистрации пользователей в системе;
2. Удаления пользователей из системы;
3. Хранение файлов, отправленные пользователями;
4. Предоставление файлов пользователю для скачивания;
5. Хранение данных о пользователях и публичных данных о ключах шифрования;

К серверу единовременно может быть подключено множество пользователей с различными именами, производящие обмен файлами через серверное приложение.

Работа сервера с клиентами происходит в формате запросов с обязательным получением ответа.

## Алгоритм работы с приложением:

## При открытии программы пользователь попадает на страницу для автономной работы с файлами. Пользователю будет предложено открыть целевой файл в данном приложении.

## После открытия файла пользователь выбирает алгоритм шифрования и метод шифрования (зашифрование, расшифрование)

## Если выбран алгоритм шифрования Camellia, пользователь может ввести свой ключ или воспользоваться сгенерированным приложением ключом

## Также, если выбран алгоритм шифрования Camellia, пользователь может выбрать один из режимов шифрования: ECB, CBC, CFB, OFB, CTR, RD, RDH.

## При нажатии кнопки «Start» начнется процесс шифрования файла, пользователь может наблюдать за его ходом в реальном времени при помощи окна «Preview».

## Полученный в результате шифрования новый файл будет иметь расширение .encrypted или .decrypted, в зависимости от выбранного метода шифрования. Далее пользователь может повторять пункты 1-3 неограниченное число раз.

## При переключении окна приложения на страницу для работы с другими пользователями пользователю будет предложено подключиться к удаленному серверу.

## При подключении пользователю будет предложено ввести свое имя или воспользоваться стандартным именем «User».

## Каждому пользователю присваивается публичный уникальный идентификатор, который позволяет избежать совпадения имен пользователей.

## После успешного подключения пользователь получает список других пользователей, подключенных к серверу и файлов, хранящихся в общедоступной части хранилища сервера. При неуспешном подключении пользователь будет уведомлен.

## Пользователь может загрузить открытый в п.1 файл с помощью кнопки «Upload opened file» в общедоступное хранилище на сервере.

## Пользователь может установить соединение с другим пользователем, подключенном к серверу.

## Если другой пользователь недоступен, или соединение уже установлено, пользователь будет об этом уведомлен.

## Если соединение может быть установлено, между пользователями производится обмен публичным ключом ассиметричного алгоритма и безопасный обмен ключом симметричного алгоритма, на сервере создается личная папка пользователей для обмена файлами.

## После успешного установления соединения пользователь может производить обмен файлами с другим пользователем при помощи функции загрузки файла (п. 2.3): при загрузке файлов на сервер файлы будут сохранены в личной папке «соединенных» пользователей и не будут видны другим, сторонним пользователям.

## Оба подключенных пользователя могут загружать в их личную папку файлы, производя таким образом обмен данными, а также, каждый из них имеет возможность скачивать файлы из общего хранилища.

## По запросу одного из пользователей соединение может быть сброшено.

## Пункты 2.3-2.7 могут быть повторены пользователем неограниченное число раз.

## Клиентское приложение:

## Клиентское приложение состоит из интерфейсов IKeyExpansion (файл i\_key.cs), IRoundCiphering, IAlgorithm (файл i\_algorithm.cs) и следующих основных классов: KeyExpansionCamellia, KeyExpansionElGamal (файл key.cs) – реализации интерфейса IKeyExpansion; FeistelNet, Camellia, ElGamal (файл algorithm.cs) –реализации интерфейса IAlgorithm; Prime (файл prime.cs); Utility (файл utility.cs).

## Интерфейс IKeyExpansion:

## public byte[][] getKeysRound(byte[] key) метод для получения «расширенного» ключа алгоритмов.

## Интерфейс IRoundCiphering:

## public ulong functionF(ulong bits\_input\_part, ulong key\_round, RoundCipheringCamelia.FunctionFModes mode) – метод для выполнения циклической операции (функция F).

## public byte[] performRound(byte[] bytes, byte[] key\_round) – метод для выполнения раунда шифрования.

## Интерфейс IAlgorithm:

## Поля:

## public string Name {get;} – название алгоритма.

## public int MessageLength {get;} – размер блока для шифрования в байтах.

## public bool IsSymmetric {get;} – тип алгоритма.

## Методы:

## public byte[][] getKeysRound(byte[] key) – метод для получения «расширенного» ключа алгоритмов.

## public void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – метод для зашифровывания одного блока байтов (сообщения).

## public void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – метод для расшифровывания одного блока байтов (сообщения).

## Класс KeyExpansionCamellia:

## Поля:

## public readonly List<BigInteger> \_constants – константы С.

## public RoundCipheringCamelia \_round\_ciphering – объект с реализацией раундового шифрования.

## Методы:

## public byte[][] getKeysRound(byte[] key) – реализация метода из интерфейса.

## Класс RoundCipheringCamelia:

## Поля:

## public enum FunctionFModes – перечисление для выбора режимов работы циклической функции.

## private readonly byte[,] \_sboxes= new byte[4, 256] – «таблица» перестановок.

## Методы:

## public ulong functionF(ulong bits\_input, ulong ke, FunctionFModes mode=FunctionFModes.F) – реализация циклической функции.

## public byte[] performRound(byte[] bytes, byte[] key\_round) – реализация метода для выполнения раунда шифрования

## Класс KeyExpansionElGamal:

## Поля:

## public enum PrimalityTestingMode – перечисления для выбора метода для тестирования чисел на простоту

## private PrimalityTestingMode \_primality\_testing\_mode

## private readonly double \_probability\_minimal – минимальная вероятность успешного тестирования числа на простоту.

## private readonly int \_prime\_numbers\_length\_bits – размер простого числа в битах.

## Методы:

## public byte[][] getKeysRound(byte[] key) - реализация метода для получения «расширенного» ключа алгоритмов.

## Класс FeistelNet:

## Поля:

## public string Name

## public int MessageLength

## public bool IsSymmetric

## Методы:

## public byte[][] getKeysRound(byte[] key) – метод для получения «расширенного» ключа алгоритмов.

## public void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для зашифровывания одного блока байтов (сообщения).

## public void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для расшифровывания одного блока байтов (сообщения).

## Класс Camellia:

## Поля:

## private readonly IKeyExpansion \_key\_expansion – объект с реализацией «‎расширения» ключа.

## Методы:

## public byte[][] getKeysRound(byte[] key) – реализация метода для получения «расширенного» ключа алгоритмов.

## public void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для зашифровывания одного блока байтов (сообщения).

## public void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для расшифровывания одного блока байтов (сообщения).

## Класс ElGamal:

## Поля:

## private readonly IKeyExpansion \_key\_expansion – объект с реализацией «‎расширения» ключа.

## Методы:

## public byte[][] getKeysRound(byte[] key) – реализация метода для получения «расширенного» ключа алгоритмов.

## public void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для зашифровывания одного блока байтов (сообщения).

## public void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round) – реализация метода для расшифровывания одного блока байтов (сообщения).

## Класс Prime:

## Методы:

## private static sbyte getJacobiSymbol(BigInteger a, BigInteger b) – метод для получения символа Якоби/Лежандра.

## public static bool performFermatTest(BigInteger number, double probability\_minimal) – метод для выполнения теста Ферма на простоту.

## public static bool performSolovayStrassenTest(BigInteger number, double probability\_minimal) – метод для выполнения теста Соловея-Штрассена на простоту.

## public static bool performMillerRabinTest(BigInteger number, double probability\_minimal) - метод для выполнения теста Миллера-Рабина на простоту.

## Класс Utility:

## Поля:

## public enum CipheringMode – перечисление для выбора режима шифрования.

## public enum PaddingType – перечисления для выбора типа дополнения.

## public const ulong MASK8 =0xff;

## public const ulong MASK32=0xffffffff;

## public const ulong MASK64=0xffffffffffffffff;

## public static readonly Random Rng;

## Методы:

## public static BigInteger getRandomNBitNumber(int n)

## public static BigInteger getRandomBigInteger(BigInteger min, BigInteger max)

## public static BigInteger byteArrayConvertToBigInteger(byte[] bytes, int start, int end)

## public static BigInteger circularShiftBigInteger(BigInteger number, int shift\_bits)

## public static BigInteger modMultiplyBigInteger(BigInteger a, BigInteger b, BigInteger modulo)

## public static ulong bigIntegerConvertToUlong(BigInteger number)

## public static void pad(ref byte[] bytes, int block\_length, PaddingType padding\_type=PaddingType.ZERO, int number\_bytes\_to\_pad=0)

## Серверное приложение:

## Серверное приложение состоит из классов AuthenticationService (AuthenticationService.cs), FileStorageService (FileStorageService.cs).

## Класс AuthenticationService:

## Поля:

## public static Dictionary<uint, User?> Users=new Dictionary<uint,User?>() – структура для хранения данных о подключенных пользователях

## Методы:

## public override Task<AuthenticationConnectReply> connect(AuthenticationConnectRequest request, ServerCallContext context) – метод для подключения к серверу.

## Класс FileStorageService:

## Поля:

## private List<FileInfo> files=new List<FileInfo>() – структура для хранения информации о файлах на сервере.

## private string files\_folder – название папки для общедоступных файлов.

## private string user\_chat\_folder – название папки для хранения личных папок для обмена файлами двух пользователей.

## Методы:

## private FileStorageServerDataReply getServerData(uint id) – метод для отправки актуальных данных о пользователях и файлах клиенту.

## public override Task<FileStorageUploadReply> upload(FileStorageUploadRequest request, ServerCallContext context) – метод для загрузки файлов на сервер.

## public override Task<FileStorageDownloadReply> download(FileStorageDownloadRequest request, ServerCallContext context) – метод для скачивания файлов с сервера.

## public override Task<FileStorageJoinUserReply> joinUser(FileStorageJoinUserRequest request, ServerCallContext context) – метод для установления соединения между двумя пользователями.

## 4 Вывод

* процессе работы над курсовым проектом изучены и реализованы алгоритмы шифрования Camellia и ElGamal. Создано клиентское приложение,

способное шифровать данные, и серверное приложение, реализующее общение между клиентами. В клиентском так же реализованы: базовые классы алгоритмов шифрования и разделения массива данных на блоки с применением дополнения, класс для осуществления тестов на простоту, а также несколько режимов шифрования. Общение клиента и сервера построено на основе технологии GRPC. Так же предусмотрены проверки на случай попадания в программу неверных данных, как, например, пустой файл, предлагаемый для зашифровки.

Архитектура программного обеспечения предусматривает выполнение процессов шифрования и дешифрования в асинхронном режиме, что способствует корректной работе приложения без зависаний во время шифровки и дешифровки файлов.

* результате разработано программное обеспечение, которое посредством сервера позволяет нескольким клиентом передавать данные по открытому каналу связи, без риска прочтения этой информации третьими лицами.

## 5 Источники

1. **Н. Смарт, С.А. Кулешова,** Мир Программирования: Криптография.

Москва, 2006 г. 529 с.

1. **Шнайер Б.** Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. М.: Триумф, 2003. 806 с.
2. **Сингх С.** Книга шифров. Тайная история шифров и их расшифровки.

М.: Аст, Астрель, 2006. 447 с.

1. **Бауэр Ф.** Расшифрованные секреты. Методы и принципы криптологии. М.: Мир, 2007. 550 c.
2. **Мао B.** Современная криптография. Теория и практика. М.: Вильямс, 2005. 763 с.
3. **Алферов А.П., Зубов А.Ю.,** Основы криптографии. 2001. 479 с.
4. **Wikipedia.Org** Схема Эль-Гамаля [Статья] 2018 г. [https://ru.wikipedia.org/wiki/Схема\_Эль-Гамаля](https://ru.wikipedia.org/wiki/Ð¡ÑÐµÐ¼Ð°_Ð­Ð)
5. **Wikipedia.Org** Алгоритм Camellia [Статья] 2017 г. [https://ru.wikipedia.org/wiki/Camellia](https://ru.wikipedia.org/wiki/FROG)
6. **Wikipedia.Org** Режимы шифрования [Статья] 2016 г. [https://ru.wikipedia.org/wiki/Режим\_шифрования](https://ru.wikipedia.org/wiki/Ð ÐµÐ¶Ð¸Ð¼_ÑÐ¸ÑÑÐ¾Ð²Ð°Ð½Ð¸Ñ)
7. **Wikipedia.Org** Тест Ферма [Статья] 2015 г. [https://ru.wikipedia.org/wiki/Тест\_Ферма](https://ru.wikipedia.org/wiki/Ð¢ÐµÑÑ_Ð¤ÐµÑÐ¼Ð°)

11. **Wikipedia.Org** Миллера — Рабина [Статья] 2020 г. [https://ru.wikipedia.org/wiki/Тест\_Миллера\_—\_Рабина](https://ru.wikipedia.org/wiki/Ð¢ÐµÑÑ_ÐÐ¸Ð)

12. **Wikipedia.Org** Соловея — Штрассена [Статья] 2022 г. [https://ru.wikipedia.org/wiki/Тест\_Соловея\_—\_Штрассена](https://ru.wikipedia.org/wiki/Ð¢ÐµÑÑ_Ð¡Ð¾Ð)

## 6 Приложение

## Файл key.cs

using System;

using System.Collections.Generic;

using System.Numerics;

using KP.Context.Interface;

namespace KP.Context

{

public class KeyExpansionCamellia : IKeyExpansion

{

public readonly List<BigInteger> \_constants = new List<BigInteger>(6) {

0xA09E667F3BCC908B,

0xB67AE8584CAA73B2,

0xC6EF372FE94F82BE,

0x54FF53A5F1D36F1C,

0x10E527FADE682D1D,

0xB05688C2B3E6C1FD

};

public RoundCipheringCamelia \_round\_ciphering;

public KeyExpansionCamellia()

{

\_round\_ciphering=new RoundCipheringCamelia();

}

public byte[][] getKeysRound(byte[] key)

{

BigInteger key\_bits=new BigInteger(key), key\_part\_l\_bits, key\_part\_r\_bits;

BigInteger ka, kb;

BigInteger[] kw=new BigInteger[4], ke, k, d=new BigInteger[2];

byte[][] keys\_round;

switch(key.Length)

{

case 16:

ke=new BigInteger[4]; k=new BigInteger[18];

keys\_round=new byte[26][];

key\_part\_l\_bits=key\_bits;

key\_part\_r\_bits=0;

break;

case 24:

ke=new BigInteger[6]; k=new BigInteger[24];

keys\_round=new byte[34][];

key\_part\_l\_bits=key\_bits>>64;

key\_part\_r\_bits=((key\_bits&Utility.MASK64)<<64) | (~(key\_bits&Utility.MASK64));

break;

case 32:

ke=new BigInteger[6]; k=new BigInteger[24];

keys\_round=new byte[34][];

key\_part\_l\_bits=key\_bits>>128;

key\_part\_r\_bits=key\_bits&BigInteger.Parse("10000000000000000000000000000000");

break;

default:

return null;

}

d[0]=(key\_part\_l\_bits^key\_part\_r\_bits) >> 64;

d[1]=(key\_part\_l\_bits^key\_part\_r\_bits) & Utility.MASK64;

d[1]^=\_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[0]), (ulong)\_constants[0]);

d[0]^=\_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[1]), (ulong)\_constants[1]);

d[0]^=key\_part\_l\_bits>>64;

d[1]^=key\_part\_l\_bits&Utility.MASK64;

try

{

d[1]^=\_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[0]), (ulong)\_constants[2]);

d[0]^=\_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[1]), (ulong)\_constants[3]);

}

catch

{

Console.WriteLine();

}

ka = (d[0] << 64) | d[1];

d[0] = ((ka ^ key\_part\_r\_bits) >> 64);

d[1] = ((ka ^ key\_part\_r\_bits) & Utility.MASK64);

d[1] ^= \_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[0]), (ulong)\_constants[4]);

d[0] ^= \_round\_ciphering.functionF(Utility.bigIntegerConvertToUlong(d[1]), (ulong)\_constants[5]);

kb = (d[0] << 64) | d[1];

if(key.Length==16)

{

k[0] =ka>>64;

k[1] =ka&Utility.MASK64;

k[2] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 15)>>64&Utility.MASK64;

k[3] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 15)&Utility.MASK64;

k[4] =Utility.circularShiftBigInteger(ka, 15)>>64&Utility.MASK64;

k[5] =Utility.circularShiftBigInteger(ka, 15)&Utility.MASK64;

k[6] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 45)>>64&Utility.MASK64;

k[7] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 45)&Utility.MASK64;

k[8] =Utility.circularShiftBigInteger(ka, 45)>>64&Utility.MASK64;

k[9] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 60)&Utility.MASK64;

k[10]=Utility.circularShiftBigInteger(ka, 60)>>64&Utility.MASK64;

k[11]=Utility.circularShiftBigInteger(ka, 60)&Utility.MASK64;

k[12]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 94)>>64&Utility.MASK64;

k[13]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 94)&Utility.MASK64;

k[14]=Utility.circularShiftBigInteger(ka, 94)>>64&Utility.MASK64;

k[15]=Utility.circularShiftBigInteger(ka, 94)&Utility.MASK64;

k[16]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 111)>>64&Utility.MASK64;

k[17]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 111)&Utility.MASK64;

ke[0]=Utility.circularShiftBigInteger(ka, 30)>>64&Utility.MASK64;

ke[1]=Utility.circularShiftBigInteger(ka, 30)&Utility.MASK64;

ke[2]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 77)>>64&Utility.MASK64;

ke[3]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 77)&Utility.MASK64;

kw[0]=key\_part\_l\_bits>>64;

kw[1]=key\_part\_l\_bits&Utility.MASK64;

kw[2]=Utility.circularShiftBigInteger(ka, 111)>>64&Utility.MASK64;

kw[3]=Utility.circularShiftBigInteger(ka, 111)&Utility.MASK64;

}

else

{

k[0] =kb>>64;

k[1] =kb&Utility.MASK64;

k[2] =Utility.circularShiftBigInteger(key\_part\_r\_bits, 15)>>64&Utility.MASK64;

k[3] =Utility.circularShiftBigInteger(key\_part\_r\_bits, 15)&Utility.MASK64;

k[4] =Utility.circularShiftBigInteger(ka, 15)>>64&Utility.MASK64;

k[5] =Utility.circularShiftBigInteger(ka, 15)&Utility.MASK64;

k[6] =Utility.circularShiftBigInteger(kb, 30)>>64&Utility.MASK64;

k[7] =Utility.circularShiftBigInteger(kb, 30)&Utility.MASK64;

k[8] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 45)>>64&Utility.MASK64;

k[9] =Utility.circularShiftBigInteger(key\_part\_l\_bits, 45)&Utility.MASK64;

k[10]=Utility.circularShiftBigInteger(ka, 45)>>64&Utility.MASK64;

k[11]=Utility.circularShiftBigInteger(ka, 45)&Utility.MASK64;

k[12]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 60)>>64&Utility.MASK64;

k[13]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 60)&Utility.MASK64;

k[14]=Utility.circularShiftBigInteger(kb, 60)>>64&Utility.MASK64;

k[15]=Utility.circularShiftBigInteger(kb, 60)&Utility.MASK64;

k[16]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 77)>>64&Utility.MASK64;

k[17]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 77)&Utility.MASK64;

k[18]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 94)>>64&Utility.MASK64;

k[19]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 94)&Utility.MASK64;

k[20]=Utility.circularShiftBigInteger(ka, 94)>>64&Utility.MASK64;

k[21]=Utility.circularShiftBigInteger(ka, 94)&Utility.MASK64;

k[22]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 111)>>64&Utility.MASK64;

k[23]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 111)&Utility.MASK64;

ke[0]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 30)>>64&Utility.MASK64;

ke[1]=Utility.circularShiftBigInteger(key\_part\_r\_bits, 30)&Utility.MASK64;

ke[2]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 60)>>64&Utility.MASK64;

ke[3]=Utility.circularShiftBigInteger(key\_part\_l\_bits, 60)&Utility.MASK64;

ke[4]=Utility.circularShiftBigInteger(ka, 77)>>64&Utility.MASK64;

ke[5]=Utility.circularShiftBigInteger(ka, 77)&Utility.MASK64;

kw[0]=key\_part\_l\_bits>>64;

kw[1]=key\_part\_l\_bits&Utility.MASK64;

kw[2]=Utility.circularShiftBigInteger(kb, 111)>>64&Utility.MASK64;

kw[3]=Utility.circularShiftBigInteger(kb, 111)&Utility.MASK64;

}

byte[] tmp;

byte i=0, j;

for(j=0; j<kw.Length; i++, j++)

{

tmp=kw[j].ToByteArray();

keys\_round[i]=new byte[9] {0, 0, 0, 0, 0, 0, 0, 0, 0};

Array.Copy(tmp, 0, keys\_round[i], 0, tmp.Length);

}

for(j=0; j<ke.Length; i++, j++)

{

tmp=ke[j].ToByteArray();

keys\_round[i]=new byte[9] {0, 0, 0, 0, 0, 0, 0, 0, 0};

Array.Copy(tmp, 0, keys\_round[i], 0, tmp.Length);

}

for(j=0; j<k.Length; i++, j++)

{

tmp=k[j].ToByteArray();

keys\_round[i]=new byte[9] {0, 0, 0, 0, 0, 0, 0, 0, 0};

Array.Copy(tmp, 0, keys\_round[i], 0, tmp.Length);

}

return keys\_round;

}

}

public class RoundCipheringCamelia : IRoundCiphering

{

public enum FunctionFModes

{

F,

FL,

FL\_INVERSE

}

private readonly byte[,] \_sboxes= new byte[4, 256] {

{0x70, 0x82, 0x2C, 0xEC, 0xB3, 0x27, 0xC0, 0xE5, 0xE4, 0x85, 0x57, 0x35, 0xEA, 0x0C, 0xAE, 0x41,

0x23, 0xEF, 0x6B, 0x93, 0x45, 0x19, 0xA5, 0x21, 0xED, 0x0E, 0x4F, 0x4E, 0x1D, 0x65, 0x92, 0xBD,

0x86, 0xB8, 0xAF, 0x8F, 0x7C, 0xEB, 0x1F, 0xCE, 0x3E, 0x30, 0xDC, 0x5F, 0x5E, 0xC5, 0x0B, 0x1A,

0xA6, 0xE1, 0x39, 0xCA, 0xD5, 0x47, 0x5D, 0x3D, 0xD9, 0x01, 0x5A, 0xD6, 0x51, 0x56, 0x6C, 0x4D,

0x8B, 0x0D, 0x9A, 0x66, 0xFB, 0xCC, 0xB0, 0x2D, 0x74, 0x12, 0x2B, 0x20, 0xF0, 0xB1, 0x84, 0x99,

0xDF, 0x4C, 0xCB, 0xC2, 0x34, 0x7E, 0x76, 0x05, 0x6D, 0xB7, 0xA9, 0x31, 0xD1, 0x17, 0x04, 0xD7,

0x14, 0x58, 0x3A, 0x61, 0xDE, 0x1B, 0x11, 0x1C, 0x32, 0x0F, 0x9C, 0x16, 0x53, 0x18, 0xF2, 0x22,

0xFE, 0x44, 0xCF, 0xB2, 0xC3, 0xB5, 0x7A, 0x91, 0x24, 0x08, 0xE8, 0xA8, 0x60, 0xFC, 0x69, 0x50,

0xAA, 0xD0, 0xA0, 0x7D, 0xA1, 0x89, 0x62, 0x97, 0x54, 0x5B, 0x1E, 0x95, 0xE0, 0xFF, 0x64, 0xD2,

0x10, 0xC4, 0x00, 0x48, 0xA3, 0xF7, 0x75, 0xDB, 0x8A, 0x03, 0xE6, 0xDA, 0x09, 0x3F, 0xDD, 0x94,

0x87, 0x5C, 0x83, 0x02, 0xCD, 0x4A, 0x90, 0x33, 0x73, 0x67, 0xF6, 0xF3, 0x9D, 0x7F, 0xBF, 0xE2,

0x52, 0x9B, 0xD8, 0x26, 0xC8, 0x37, 0xC6, 0x3B, 0x81, 0x96, 0x6F, 0x4B, 0x13, 0xBE, 0x63, 0x2E,

0xE9, 0x79, 0xA7, 0x8C, 0x9F, 0x6E, 0xBC, 0x8E, 0x29, 0xF5, 0xF9, 0xB6, 0x2F, 0xFD, 0xB4, 0x59,

0x78, 0x98, 0x06, 0x6A, 0xE7, 0x46, 0x71, 0xBA, 0xD4, 0x25, 0xAB, 0x42, 0x88, 0xA2, 0x8D, 0xFA,

0x72, 0x07, 0xB9, 0x55, 0xF8, 0xEE, 0xAC, 0x0A, 0x36, 0x49, 0x2A, 0x68, 0x3C, 0x38, 0xF1, 0xA4,

0x40, 0x28, 0xD3, 0x7B, 0xBB, 0xC9, 0x43, 0xC1, 0x15, 0xE3, 0xAD, 0xF4, 0x77, 0xC7, 0x80, 0x9E},

{0xE0, 0x05, 0x58, 0xD9, 0x67, 0x4E, 0x81, 0xCB, 0xC9, 0x0B, 0xAE, 0x6A, 0xD5, 0x18, 0x5D, 0x82,

0x46, 0xDF, 0xD6, 0x27, 0x8A, 0x32, 0x4B, 0x42, 0xDB, 0x1C, 0x9E, 0x9C, 0x3A, 0xCA, 0x25, 0x7B,

0x0D, 0x71, 0x5F, 0x1F, 0xF8, 0xD7, 0x3E, 0x9D, 0x7C, 0x60, 0xB9, 0xBE, 0xBC, 0x8B, 0x16, 0x34,

0x4D, 0xC3, 0x72, 0x95, 0xAB, 0x8E, 0xBA, 0x7A, 0xB3, 0x02, 0xB4, 0xAD, 0xA2, 0xAC, 0xD8, 0x9A,

0x17, 0x1A, 0x35, 0xCC, 0xF7, 0x99, 0x61, 0x5A, 0xE8, 0x24, 0x56, 0x40, 0xE1, 0x63, 0x09, 0x33,

0xBF, 0x98, 0x97, 0x85, 0x68, 0xFC, 0xEC, 0x0A, 0xDA, 0x6F, 0x53, 0x62, 0xA3, 0x2E, 0x08, 0xAF,

0x28, 0xB0, 0x74, 0xC2, 0xBD, 0x36, 0x22, 0x38, 0x64, 0x1E, 0x39, 0x2C, 0xA6, 0x30, 0xE5, 0x44,

0xFD, 0x88, 0x9F, 0x65, 0x87, 0x6B, 0xF4, 0x23, 0x48, 0x10, 0xD1, 0x51, 0xC0, 0xF9, 0xD2, 0xA0,

0x55, 0xA1, 0x41, 0xFA, 0x43, 0x13, 0xC4, 0x2F, 0xA8, 0xB6, 0x3C, 0x2B, 0xC1, 0xFF, 0xC8, 0xA5,

0x20, 0x89, 0x00, 0x90, 0x47, 0xEF, 0xEA, 0xB7, 0x15, 0x06, 0xCD, 0xB5, 0x12, 0x7E, 0xBB, 0x29,

0x0F, 0xB8, 0x07, 0x04, 0x9B, 0x94, 0x21, 0x66, 0xE6, 0xCE, 0xED, 0xE7, 0x3B, 0xFE, 0x7F, 0xC5,

0xA4, 0x37, 0xB1, 0x4C, 0x91, 0x6E, 0x8D, 0x76, 0x03, 0x2D, 0xDE, 0x96, 0x26, 0x7D, 0xC6, 0x5C,

0xD3, 0xF2, 0x4F, 0x19, 0x3F, 0xDC, 0x79, 0x1D, 0x52, 0xEB, 0xF3, 0x6D, 0x5E, 0xFB, 0x69, 0xB2,

0xF0, 0x31, 0x0C, 0xD4, 0xCF, 0x8C, 0xE2, 0x75, 0xA9, 0x4A, 0x57, 0x84, 0x11, 0x45, 0x1B, 0xF5,

0xE4, 0x0E, 0x73, 0xAA, 0xF1, 0xDD, 0x59, 0x14, 0x6C, 0x92, 0x54, 0xD0, 0x78, 0x70, 0xE3, 0x49,

0x80, 0x50, 0xA7, 0xF6, 0x77, 0x93, 0x86, 0x83, 0x2A, 0xC7, 0x5B, 0xE9, 0xEE, 0x8F, 0x01, 0x3D},

{0x38, 0x41, 0x16, 0x76, 0xD9, 0x93, 0x60, 0xF2, 0x72, 0xC2, 0xAB, 0x9A, 0x75, 0x06, 0x57, 0xA0,

0x91, 0xF7, 0xB5, 0xC9, 0xA2, 0x8C, 0xD2, 0x90, 0xF6, 0x07, 0xA7, 0x27, 0x8E, 0xB2, 0x49, 0xDE,

0x43, 0x5C, 0xD7, 0xC7, 0x3E, 0xF5, 0x8F, 0x67, 0x1F, 0x18, 0x6E, 0xAF, 0x2F, 0xE2, 0x85, 0x0D,

0x53, 0xF0, 0x9C, 0x65, 0xEA, 0xA3, 0xAE, 0x9E, 0xEC, 0x80, 0x2D, 0x6B, 0xA8, 0x2B, 0x36, 0xA6,

0xC5, 0x86, 0x4D, 0x33, 0xFD, 0x66, 0x58, 0x96, 0x3A, 0x09, 0x95, 0x10, 0x78, 0xD8, 0x42, 0xCC,

0xEF, 0x26, 0xE5, 0x61, 0x1A, 0x3F, 0x3B, 0x82, 0xB6, 0xDB, 0xD4, 0x98, 0xE8, 0x8B, 0x02, 0xEB,

0x0A, 0x2C, 0x1D, 0xB0, 0x6F, 0x8D, 0x88, 0x0E, 0x19, 0x87, 0x4E, 0x0B, 0xA9, 0x0C, 0x79, 0x11,

0x7F, 0x22, 0xE7, 0x59, 0xE1, 0xDA, 0x3D, 0xC8, 0x12, 0x04, 0x74, 0x54, 0x30, 0x7E, 0xB4, 0x28,

0x55, 0x68, 0x50, 0xBE, 0xD0, 0xC4, 0x31, 0xCB, 0x2A, 0xAD, 0x0F, 0xCA, 0x70, 0xFF, 0x32, 0x69,

0x08, 0x62, 0x00, 0x24, 0xD1, 0xFB, 0xBA, 0xED, 0x45, 0x81, 0x73, 0x6D, 0x84, 0x9F, 0xEE, 0x4A,

0xC3, 0x2E, 0xC1, 0x01, 0xE6, 0x25, 0x48, 0x99, 0xB9, 0xB3, 0x7B, 0xF9, 0xCE, 0xBF, 0xDF, 0x71,

0x29, 0xCD, 0x6C, 0x13, 0x64, 0x9B, 0x63, 0x9D, 0xC0, 0x4B, 0xB7, 0xA5, 0x89, 0x5F, 0xB1, 0x17,

0xF4, 0xBC, 0xD3, 0x46, 0xCF, 0x37, 0x5E, 0x47, 0x94, 0xFA, 0xFC, 0x5B, 0x97, 0xFE, 0x5A, 0xAC,

0x3C, 0x4C, 0x03, 0x35, 0xF3, 0x23, 0xB8, 0x5D, 0x6A, 0x92, 0xD5, 0x21, 0x44, 0x51, 0xC6, 0x7D,

0x39, 0x83, 0xDC, 0xAA, 0x7C, 0x77, 0x56, 0x05, 0x1B, 0xA4, 0x15, 0x34, 0x1E, 0x1C, 0xF8, 0x52,

0x20, 0x14, 0xE9, 0xBD, 0xDD, 0xE4, 0xA1, 0xE0, 0x8A, 0xF1, 0xD6, 0x7A, 0xBB, 0xE3, 0x40, 0x4F},

{0x70, 0x2C, 0xB3, 0xC0, 0xE4, 0x57, 0xEA, 0xAE, 0x23, 0x6B, 0x45, 0xA5, 0xED, 0x4F, 0x1D, 0x92,

0x86, 0xAF, 0x7C, 0x1F, 0x3E, 0xDC, 0x5E, 0x0B, 0xA6, 0x39, 0xD5, 0x5D, 0xD9, 0x5A, 0x51, 0x6C,

0x8B, 0x9A, 0xFB, 0xB0, 0x74, 0x2B, 0xF0, 0x84, 0xDF, 0xCB, 0x34, 0x76, 0x6D, 0xA9, 0xD1, 0x04,

0x14, 0x3A, 0xDE, 0x11, 0x32, 0x9C, 0x53, 0xF2, 0xFE, 0xCF, 0xC3, 0x7A, 0x24, 0xE8, 0x60, 0x69,

0xAA, 0xA0, 0xA1, 0x62, 0x54, 0x1E, 0xE0, 0x64, 0x10, 0x00, 0xA3, 0x75, 0x8A, 0xE6, 0x09, 0xDD,

0x87, 0x83, 0xCD, 0x90, 0x73, 0xF6, 0x9D, 0xBF, 0x52, 0xD8, 0xC8, 0xC6, 0x81, 0x6F, 0x13, 0x63,

0xE9, 0xA7, 0x9F, 0xBC, 0x29, 0xF9, 0x2F, 0xB4, 0x78, 0x06, 0xE7, 0x71, 0xD4, 0xAB, 0x88, 0x8D,

0x72, 0xB9, 0xF8, 0xAC, 0x36, 0x2A, 0x3C, 0xF1, 0x40, 0xD3, 0xBB, 0x43, 0x15, 0xAD, 0x77, 0x80,

0x82, 0xEC, 0x27, 0xE5, 0x85, 0x35, 0x0C, 0x41, 0xEF, 0x93, 0x19, 0x21, 0x0E, 0x4E, 0x65, 0xBD,

0xB8, 0x8F, 0xEB, 0xCE, 0x30, 0x5F, 0xC5, 0x1A, 0xE1, 0xCA, 0x47, 0x3D, 0x01, 0xD6, 0x56, 0x4D,

0x0D, 0x66, 0xCC, 0x2D, 0x12, 0x20, 0xB1, 0x99, 0x4C, 0xC2, 0x7E, 0x05, 0xB7, 0x31, 0x17, 0xD7,

0x58, 0x61, 0x1B, 0x1C, 0x0F, 0x16, 0x18, 0x22, 0x44, 0xB2, 0xB5, 0x91, 0x08, 0xA8, 0xFC, 0x50,

0xD0, 0x7D, 0x89, 0x97, 0x5B, 0x95, 0xFF, 0xD2, 0xC4, 0x48, 0xF7, 0xDB, 0x03, 0xDA, 0x3F, 0x94,

0x5C, 0x02, 0x4A, 0x33, 0x67, 0xF3, 0x7F, 0xE2, 0x9B, 0x26, 0x37, 0x3B, 0x96, 0x4B, 0xBE, 0x2E,

0x79, 0x8C, 0x6E, 0x8E, 0xF5, 0xB6, 0xFD, 0x59, 0x98, 0x6A, 0x46, 0xBA, 0x25, 0x42, 0xA2, 0xFA,

0x07, 0x55, 0xEE, 0x0A, 0x49, 0x68, 0x38, 0xA4, 0x28, 0x7B, 0xC9, 0xC1, 0xE3, 0xF4, 0xC7, 0x9E}

};

public ulong functionF(ulong bits\_input, ulong ke, FunctionFModes mode=FunctionFModes.F)

{

switch(mode)

{

case FunctionFModes.F:

ulong x\_F = bits\_input^ke;

ulong[] t\_F=new ulong[8], y\_F=new ulong[8];

t\_F[0]=x\_F>> 56;

for(sbyte i=48; i>=0; i-=8)

t\_F[i/8+1]=(x\_F>>(48-i))&Utility.MASK8;

t\_F[0]=\_sboxes[0,t\_F[0]];

t\_F[1]=\_sboxes[1,t\_F[1]];

t\_F[2]=\_sboxes[2,t\_F[2]];

t\_F[3]=\_sboxes[3,t\_F[3]];

t\_F[4]=\_sboxes[1,t\_F[4]];

t\_F[5]=\_sboxes[2,t\_F[5]];

t\_F[6]=\_sboxes[3,t\_F[6]];

t\_F[7]=\_sboxes[0,t\_F[7]];

y\_F[0]=t\_F[0]^t\_F[2]^t\_F[3]^t\_F[5]^t\_F[6]^t\_F[7];

y\_F[1]=t\_F[0]^t\_F[1]^t\_F[3]^t\_F[4]^t\_F[6]^t\_F[7];

y\_F[2]=t\_F[0]^t\_F[1]^t\_F[2]^t\_F[4]^t\_F[5]^t\_F[7];

y\_F[3]=t\_F[1]^t\_F[2]^t\_F[3]^t\_F[4]^t\_F[5]^t\_F[6];

y\_F[4]=t\_F[0]^t\_F[1]^t\_F[5]^t\_F[6]^t\_F[7];

y\_F[5]=t\_F[1]^t\_F[2]^t\_F[4]^t\_F[6]^t\_F[7];

y\_F[6]=t\_F[2]^t\_F[3]^t\_F[4]^t\_F[5]^t\_F[7];

y\_F[7]=t\_F[0]^t\_F[3]^t\_F[4]^t\_F[5]^t\_F[6];

return (y\_F[0]<<56) | (y\_F[1]<<48) | (y\_F[2]<<40) | (y\_F[3]<<32)| (y\_F[4]<<24) | (y\_F[5]<<16) | (y\_F[6]<<8) | y\_F[7];

case FunctionFModes.FL:

uint[] x\_FL=new uint[2] {(uint)(bits\_input>>32), (uint)(bits\_input&Utility.MASK32)}, k\_FL=new uint[2] {(uint)(ke>>32), (uint)(ke&Utility.MASK32)};

uint t\_FL;

t\_FL=x\_FL[0]&k\_FL[0];

x\_FL[1]^=(t\_FL<<1) | (t\_FL>>31);

x\_FL[0]^=x\_FL[1] | k\_FL[1];

return ((ulong)x\_FL[0] << 32) | x\_FL[1];

case FunctionFModes.FL\_INVERSE:

uint[] y\_FL\_INVERSE=new uint[2] {(uint)(bits\_input>>32), (uint)(bits\_input&Utility.MASK32)}, k\_FL\_INVERSE=new uint[2] {(uint)(ke>>32), (uint)(ke&Utility.MASK32)};

uint t\_FL\_INVERSE;

y\_FL\_INVERSE[0]^=y\_FL\_INVERSE[1] | k\_FL\_INVERSE[1];

t\_FL\_INVERSE=y\_FL\_INVERSE[0]&k\_FL\_INVERSE[0];

y\_FL\_INVERSE[1]^=(t\_FL\_INVERSE<<1) | (t\_FL\_INVERSE>>31);

return ((ulong)y\_FL\_INVERSE[0]<<32) | y\_FL\_INVERSE[1];

default:

return 0;

}

}

public byte[] performRound(byte[] bytes, byte[] key\_round)

{

ulong[] bits\_parts=new ulong[2] {BitConverter.ToUInt64(bytes, 0), BitConverter.ToUInt64(bytes, 8)};

byte[] result=new byte[16];

bits\_parts[1]^=functionF(bits\_parts[0], BitConverter.ToUInt64(key\_round, 0));

Array.Copy(BitConverter.GetBytes(bits\_parts[1]), 0,result, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_parts[0]), 0,result, 8, 8);

return result;

}

}

public class KeyExpansionElGamal : IKeyExpansion

{

public enum PrimalityTestingMode

{

FERMAT,

SOLOVAY\_STRASSEN,

MILLER\_RABIN

}

private PrimalityTestingMode \_primality\_testing\_mode;

private readonly double \_probability\_minimal;

private readonly int \_prime\_numbers\_length\_bits;

public KeyExpansionElGamal(PrimalityTestingMode primality\_testing\_mode, double probability\_minimal, int prime\_numbers\_length\_bits)

{

\_primality\_testing\_mode=primality\_testing\_mode;

\_probability\_minimal=probability\_minimal;

\_prime\_numbers\_length\_bits=prime\_numbers\_length\_bits;

}

private BigInteger getPrimeNumber(int min, int max)

{

BigInteger number=Utility.getRandomNBitNumber(Utility.Rng.Next(min, max));

switch(\_primality\_testing\_mode)

{

case PrimalityTestingMode.FERMAT:

while(!Primality.performFermatTest(number, \_probability\_minimal))

number=Utility.getRandomNBitNumber(Utility.Rng.Next(min, max));

break;

case PrimalityTestingMode.SOLOVAY\_STRASSEN:

while(!Primality.performSolovayStrassenTest(number, \_probability\_minimal))

number=Utility.getRandomNBitNumber(Utility.Rng.Next(min, max));

break;

case PrimalityTestingMode.MILLER\_RABIN:

while(!Primality.performMillerRabinTest(number, \_probability\_minimal))

number=Utility.getRandomNBitNumber(Utility.Rng.Next(min, max));

break;

}

return number;

}

public byte[][] getKeysRound(byte[] key)

{

BigInteger p=getPrimeNumber(\_prime\_numbers\_length\_bits, \_prime\_numbers\_length\_bits), p\_minus\_one=p-1, q=p\_minus\_one/2, g=1, x, y;

for( ; g<p\_minus\_one; g++)

if(BigInteger.ModPow(g, 2, p)!=1 && BigInteger.ModPow(g, q, p)!=1)

break;

x=Utility.getRandomBigInteger(1, p\_minus\_one);

y=BigInteger.ModPow(g, x, p);

return new byte[][] {p.ToByteArray(), g.ToByteArray(), y.ToByteArray(), x.ToByteArray()};

}

}

}

## Файл algorithm.cs

using System;

using System.Linq;

using System.Numerics;

using System.Windows.Media.Animation;

using KP.Context.Interface;

namespace KP.Context

{

public class FeistelNet : IAlgorithm

{

public IRoundCiphering \_round\_ciphering;

public string Name

{

get;

}

public int MessageLength

{

get {return 16;}

}

public bool IsSymmetric

{

get {return true;}

}

public FeistelNet(IRoundCiphering round\_ciphering)

{

\_round\_ciphering=round\_ciphering;

}

public virtual byte[][] getKeysRound(byte[] key)

{

return null;

}

public virtual void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

ulong[] keys\_round\_converted=new ulong[keys\_round.Length];

byte[] bytes\_input\_local;

byte i=0;

for( ; i<keys\_round.Length; i++)

keys\_round\_converted[i]=BitConverter.ToUInt64(keys\_round[i], 0);

bytes\_input\_local=new byte[bytes\_input.Length];

bytes\_input.CopyTo(bytes\_input\_local, 0);

ulong[] bits\_input\_parts=new ulong[2] {BitConverter.ToUInt64(bytes\_input\_local, 8)^keys\_round\_converted[0], BitConverter.ToUInt64(bytes\_input\_local, 0)^keys\_round\_converted[1]};

byte iterations=0, iteration\_rounds=6, k, e=4;

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

i=0;

if(keys\_round\_converted.Length==26)

{

k=8;

for( ; iterations<2; iterations++, iteration\_rounds+=6, e++)

{

for(; i<iteration\_rounds; i++, k++)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

bits\_input\_parts[0]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 0), keys\_round\_converted[e], RoundCipheringCamelia.FunctionFModes.FL);

bits\_input\_parts[1]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 8), keys\_round\_converted[++e], RoundCipheringCamelia.FunctionFModes.FL\_INVERSE);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

}

for( ; i<iteration\_rounds; i++, k++)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

}

else

{

k=10;

for( ; iterations<3; iterations++, iteration\_rounds+=6, e++)

{

for(; i<iteration\_rounds; i++, k++)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

bits\_input\_parts[0]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 0), keys\_round\_converted[e], RoundCipheringCamelia.FunctionFModes.FL);

bits\_input\_parts[1]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 8), keys\_round\_converted[++e], RoundCipheringCamelia.FunctionFModes.FL\_INVERSE);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

}

for( ; i<iteration\_rounds; i++, k++)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

}

bits\_input\_parts[1]=BitConverter.ToUInt64(bytes\_input\_local, 8)^keys\_round\_converted[2];

bits\_input\_parts[0]=BitConverter.ToUInt64(bytes\_input\_local, 0)^keys\_round\_converted[3];

bytes\_output=(((BigInteger)bits\_input\_parts[1]<<64)|bits\_input\_parts[0]).ToByteArray().Where((source, index) =>index != 16).ToArray();

}

public virtual void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

ulong[] keys\_round\_converted=new ulong[keys\_round.Length];

byte[] bytes\_input\_local;

byte i=0;

for( ; i<keys\_round.Length; i++)

keys\_round\_converted[i]=BitConverter.ToUInt64(keys\_round[i], 0);

bytes\_input\_local=new byte[bytes\_input.Length];

bytes\_input.CopyTo(bytes\_input\_local, 0);

ulong[] bits\_input\_parts=new ulong[2] {BitConverter.ToUInt64(bytes\_input\_local, 8)^keys\_round\_converted[2], BitConverter.ToUInt64(bytes\_input\_local, 0)^keys\_round\_converted[3]};

byte iterations=0, iteration\_rounds=6, k, e;

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

i=0;

if(keys\_round\_converted.Length==26)

{

k=25; e=7;

for( ; iterations<2; iterations++, iteration\_rounds+=6, e--)

{

for(; i<iteration\_rounds; i++, k--)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

bits\_input\_parts[0]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 0), keys\_round\_converted[e], RoundCipheringCamelia.FunctionFModes.FL);

bits\_input\_parts[1]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 8), keys\_round\_converted[--e], RoundCipheringCamelia.FunctionFModes.FL\_INVERSE);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

}

for( ; i<iteration\_rounds; i++, k--)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

}

else

{

k=33; e=9;

for( ; iterations<3; iterations++, iteration\_rounds+=6, e--)

{

for(; i<iteration\_rounds; i++, k--)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

bits\_input\_parts[0]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 0), keys\_round\_converted[e], RoundCipheringCamelia.FunctionFModes.FL);

bits\_input\_parts[1]=\_round\_ciphering.functionF(BitConverter.ToUInt64(bytes\_input\_local, 8), keys\_round\_converted[--e], RoundCipheringCamelia.FunctionFModes.FL\_INVERSE);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[0]), 0, bytes\_input\_local, 0, 8);

Array.Copy(BitConverter.GetBytes(bits\_input\_parts[1]), 0, bytes\_input\_local, 8, 8);

}

for( ; i<iteration\_rounds; i++, k--)

bytes\_input\_local=\_round\_ciphering.performRound(bytes\_input\_local, keys\_round[k]);

}

bits\_input\_parts[1]=BitConverter.ToUInt64(bytes\_input\_local, 8)^keys\_round\_converted[0];

bits\_input\_parts[0]=BitConverter.ToUInt64(bytes\_input\_local, 0)^keys\_round\_converted[1];

bytes\_output=(((BigInteger)bits\_input\_parts[1]<<64)|bits\_input\_parts[0]).ToByteArray().Where((source, index) =>index!=16).ToArray();

}

}

public sealed class Camellia : FeistelNet

{

private readonly IKeyExpansion \_key\_expansion;

public string Name

{

get {return "Camellia"; }

}

public int MessageLength

{

get {return 16;}

}

public bool IsSymmetric

{

get {return true;}

}

public Camellia() : base(new RoundCipheringCamelia())

{

\_key\_expansion=new KeyExpansionCamellia();

}

public override byte[][] getKeysRound(byte[] key)

{

return \_key\_expansion.getKeysRound(key);

}

public override void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

base.encrypt(in bytes\_input, ref bytes\_output, keys\_round);

}

public override void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

base.decrypt(in bytes\_input, ref bytes\_output, keys\_round);

}

}

public sealed class ElGamal : IAlgorithm

{

private readonly IKeyExpansion \_key\_expansion;

public string Name

{

get {return "ElGamal";}

}

public int MessageLength

{

get;

}

public bool IsSymmetric

{

get {return false;}

}

public ElGamal(KeyExpansionElGamal.PrimalityTestingMode primality\_testing\_mode=KeyExpansionElGamal.PrimalityTestingMode.FERMAT, double probability\_minimal=0.999, int prime\_numbers\_length\_bits=64)

{

\_key\_expansion=new KeyExpansionElGamal(primality\_testing\_mode, probability\_minimal, prime\_numbers\_length\_bits);

MessageLength=prime\_numbers\_length\_bits/8;

}

public byte[][] getKeysRound(byte[] key)

{

return \_key\_expansion.getKeysRound(key);

}

public void encrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

BigInteger p=new BigInteger(keys\_round[0]), p\_minus\_one=p-1, k;

byte[] tmp;

int ie=0;

begin:

do

k=Utility.getRandomBigInteger(2, p\_minus\_one-1);

while(BigInteger.GreatestCommonDivisor(k, p\_minus\_one)!=1);

bytes\_output=new byte[MessageLength\*2];

tmp=BigInteger.ModPow(new BigInteger(keys\_round[1]), k, p).ToByteArray();

tmp.CopyTo(bytes\_output, 0);

var a=Utility.modMultiplyBigInteger(BigInteger.ModPow(new BigInteger(keys\_round[2]), k, p), new BigInteger(bytes\_input), p);

if(a==0 && ie<1000)

{

ie++;

goto begin;

}

tmp=a.ToByteArray().Where((source, index) => index!=MessageLength+1).ToArray();

tmp.CopyTo(bytes\_output, MessageLength);

}

public void decrypt(in byte[] bytes\_input, ref byte[] bytes\_output, byte[][] keys\_round)

{

BigInteger p=new BigInteger(keys\_round[0]);

bytes\_output=Utility.modMultiplyBigInteger(Utility.byteArrayConvertToBigInteger(bytes\_input, MessageLength, MessageLength\*2), BigInteger.ModPow(Utility.byteArrayConvertToBigInteger(bytes\_input, 0, MessageLength), p-1-new BigInteger(keys\_round[3]), p), p).ToByteArray();

}

}

}

## Файл utility.cs

using System;

using System.Numerics;

namespace KP.Context

{

public static class Utility

{

public enum CipheringMode

{

ECB,

CBC,

CFB,

OFB,

CTR,

RD,

RD\_H

}

public enum PaddingType

{

NONE,

ZERO,

PKCS7

}

public const ulong MASK8 =0xff;

public const ulong MASK32=0xffffffff;

public const ulong MASK64=0xffffffffffffffff;

public static readonly Random Rng;

static Utility()

{

Rng=new Random();

}

public static BigInteger getRandomNBitNumber(int n)

{

BigInteger t=1;

n--;

return (t<<n)|(Rng.Next()%(t<<n));

}

public static BigInteger getRandomBigInteger(BigInteger min, BigInteger max)

{

byte[] max\_bytes=max.ToByteArray();

Rng.NextBytes(max\_bytes);

max\_bytes[max\_bytes.Length-1]&=0x7F;

return new BigInteger(max\_bytes)%max+min;

}

public static BigInteger byteArrayConvertToBigInteger(byte[] bytes, int start, int end)

{

BigInteger result=0;

for(int i=end-1; i>start-1; i--)

result=(result<<8)|bytes[i];

return result;

}

public static BigInteger circularShiftBigInteger(BigInteger number, int shift\_bits)

{

return number<<shift\_bits | number>>(number.ToByteArray().Length\*8-shift\_bits);

}

public static BigInteger modMultiplyBigInteger(BigInteger a, BigInteger b, BigInteger modulo)

{

BigInteger res=0;

a%=modulo;

while(b>0)

{

if((b&1)>0)

res=(res+a)%modulo;

a=(2\*a)%modulo;

b>>=1;

}

return res;

}

public static ulong bigIntegerConvertToUlong(BigInteger number)

{

return number<0 ? (ulong)Int64.Parse(number.ToString()) : UInt64.Parse(number.ToString());

}

public static void pad(ref byte[] bytes, int block\_length, PaddingType padding\_type=PaddingType.ZERO, int number\_bytes\_to\_pad=0)

{

byte value;

switch(padding\_type)

{

case PaddingType.ZERO:

value=0;

break;

case PaddingType.PKCS7:

value=(byte)(block\_length-bytes.Length);

break;

default:

return;

}

if(number\_bytes\_to\_pad==0)

{

byte[] bytes\_padded=new byte[block\_length];

if(bytes.Length>block\_length)

Array.Copy(bytes, 0, bytes\_padded, 0, block\_length);

else

bytes.CopyTo(bytes\_padded, 0);

for(int i=bytes.Length; i<block\_length; i++)

bytes\_padded[i]=value;

bytes=bytes\_padded;

}

else

for(int i=bytes.Length-number\_bytes\_to\_pad; i<bytes.Length; i++)

bytes[i]=value;

}

}

}

## Файл prime.cs

using System;

using System.Numerics;

namespace KP.Context

{

public static class Primality

{

private static sbyte getJacobiSymbol(BigInteger a, BigInteger b)

{

if(BigInteger.GreatestCommonDivisor(a, b)!=1)

return 0;

sbyte r=1;

a%=b;

for(BigInteger t; a!=0; a%=b)

{

while(a%2==0)

{

t=b%8;

a/=2;

if(t==3 || t==5)

r\*=-1;

}

(a, b)=(b, a);

if(a%4==3 && b%4==3)

r\*=-1;

}

return b==1 ?r :(sbyte)0;

}

public static bool performFermatTest(BigInteger number, double probability\_minimal)

{

if(number==0 || number==4)

return false;

ulong i=1;

for(BigInteger a; (1-Math.Pow(0.5, i))<=probability\_minimal; i++)

{

a=2+Utility.Rng.Next()%(number-4);

if(BigInteger.GreatestCommonDivisor(a, number)!=1 || BigInteger.ModPow(a, number-1, number)!=1)

return false;

}

return true;

}

public static bool performSolovayStrassenTest(BigInteger number, double probability\_minimal)

{

if(number==0)

return false;

ulong i=1;

for(BigInteger a, j; (1-Math.Pow(0.5, i))<=probability\_minimal; i++)

{

a=Utility.Rng.Next()%(number-1)+1;

j=(number+getJacobiSymbol(a, number))%number;

if(j==0 || BigInteger.ModPow(a, (number-1)/2, number)!=j)

return false;

}

return true;

}

public static bool performMillerRabinTest(BigInteger number, double probability\_minimal)

{

if(number<=1 || number==4)

return false;

if(number<=3)

return true;

BigInteger d=number-1;

while(d%2==0)

d/=2;

ulong i=1;

for(BigInteger a, x; (1-Math.Pow(0.5, i))<=probability\_minimal; i++)

{

a=2+Utility.Rng.Next()%(number-4);

x=BigInteger.ModPow(a, d, number);

if(x==1 || x==number-1)

continue;

for( ; d!=number-1; d\*=2)

{

x=(x\*x)%number;

if(x==1)

return false;

if(x==number-1)

break;

}

if(x==number-1)

continue;

return false;

}

return true;

}

}

}

## Файл AuthenticationService.cs

using Grpc.Core;

using Proto;

namespace KP\_Server.Services;

public class AuthenticationService : Authentication.AuthenticationBase

{

private readonly ILogger<AuthenticationService> \_logger;

public class User

{

public enum Status

{

DEFAULT,

PENDING

}

public uint id;

public uint id\_connected\_to;

public string? name;

public DateTime last\_time\_connected;

public DirectoryInfo? chat\_directory;

public byte[][]? public\_key=null;

public byte[]? symmetric\_key=null;

public Status status;

}

private uint \_id\_counter=0;

public static Dictionary<uint, User?> Users=new Dictionary<uint,User?>();

private string \_data\_folder=Environment.CurrentDirectory+"\\data\\";

private FileInfo \_file\_info\_users;

public AuthenticationService()

{

\_file\_info\_users=new FileInfo(\_data\_folder+"Users.txt");

}

private async void invalidateUsers(int delay=500)

{

DateTime date\_time;

while(true)

{

date\_time=DateTime.Now;

foreach (var user in Users)

{

if(user.Value!=null)

if((date\_time-user.Value.last\_time\_connected).Seconds>10)

Users.Remove(user.Key);

}

await Task.Delay(delay);

}

}

public override Task<AuthenticationConnectReply> connect(AuthenticationConnectRequest request, ServerCallContext context)

{

Users.Add(\_id\_counter, new User

{id=\_id\_counter,

id\_connected\_to=\_id\_counter,

name=request.UserName,

last\_time\_connected=DateTime.Now});

AuthenticationConnectReply reply=new AuthenticationConnectReply

{ Result=new ResultReply

{ Result=true,

Info=$"Connected as {request.UserName} #{\_id\_counter}" },

Id=\_id\_counter };

\_id\_counter++;

return Task.FromResult(reply);

}

public override Task<AuthenticationMaintainConnectionReply> maintainConnection(AuthenticationMaintainConnectionRequest request, ServerCallContext context)

{

if(Users[request.Id]==null)

{

return Task.FromResult(new AuthenticationMaintainConnectionReply

{Result=new ResultReply

{Result=false,

Info="ERROR: Connection blocked, user is not authorised."}});

}

if(!Users.ContainsKey(request.Id))

{

Users.Add(\_id\_counter++, null);

return Task.FromResult(new AuthenticationMaintainConnectionReply

{Result=new ResultReply

{Result=false,

Info="ERROR: Blocking connection, user id not found."}});

}

Users[request.Id].last\_time\_connected=DateTime.Now;

return Task.FromResult(new AuthenticationMaintainConnectionReply

{Result=new ResultReply

{Result=true,

Info="Welcome "}});

}

}

## Файл FileStorageService.cs

using Google.Protobuf;

using Grpc.Core;

using Proto;

namespace KP\_Server.Services;

public class FileStorageService : FileStorage.FileStorageBase

{

private List<FileInfo> files=new List<FileInfo>();

private string files\_folder=Environment.CurrentDirectory+"\\file\_storage\\";

private string user\_chat\_folder="user\_chat\\";

public FileStorageService()

{

foreach (FileInfo file in new DirectoryInfo(files\_folder).GetFiles())

files.Add(file);

}

private FileStorageServerDataReply getServerData(uint id)

{

AuthenticationService.Users[id].last\_time\_connected=DateTime.Now;

FileStorageServerDataReply reply=new FileStorageServerDataReply();

if(AuthenticationService.Users[id].status==AuthenticationService.User.Status.PENDING)

{

if(AuthenticationService.Users[id].public\_key==null)

{

AuthenticationService.Users[id].public\_key=AuthenticationService.Users[AuthenticationService.Users[id].id\_connected\_to].public\_key;

reply.IdConnectedTo=AuthenticationService.Users[id].id\_connected\_to;

for(int i=0; i<3; i++)

reply.PublicKey.Add(ByteString.CopyFrom(AuthenticationService.Users[id].public\_key[i]));

AuthenticationService.Users[id].status=AuthenticationService.User.Status.DEFAULT;

}

else if(AuthenticationService.Users[id].symmetric\_key==null)

{

AuthenticationService.Users[id].symmetric\_key=AuthenticationService.Users[AuthenticationService.Users[id].id\_connected\_to].symmetric\_key;

reply.SymmetricKey=ByteString.CopyFrom(AuthenticationService.Users[id].symmetric\_key);

AuthenticationService.Users[id].status=AuthenticationService.User.Status.DEFAULT;

}

}

foreach (var user in AuthenticationService.Users)

{

if(user.Value!=null && !String.IsNullOrWhiteSpace(user.Value.name) && user.Key!=id)

{

reply.Users.Add(new FileStorageServerUser()

{Name=user.Value.name,

Id=user.Key});

}

}

if(AuthenticationService.Users[id].id\_connected\_to!=id && AuthenticationService.Users[id].symmetric\_key!=null && AuthenticationService.Users[id].chat\_directory!=null)

{

foreach (FileInfo file in AuthenticationService.Users[id].chat\_directory.GetFiles())

reply.FileNames.Add(file.Name);

}

else

{

foreach (FileInfo file in new DirectoryInfo(files\_folder).GetFiles())

reply.FileNames.Add(file.Name);

}

reply.Result=new ResultReply();

return reply;

}

public override Task<FileStorageUploadReply> upload(FileStorageUploadRequest request, ServerCallContext context)

{

if(request.FileName==null || request.Content==null)

return Task.FromResult(new FileStorageUploadReply

{Result=new ResultReply

{Result=false,

Info="ERROR: Filename of file content is empty."}});

string file\_full\_name;

if(AuthenticationService.Users[request.Id].id\_connected\_to!=request.Id && AuthenticationService.Users[request.Id].symmetric\_key!=null)

file\_full\_name=files\_folder+user\_chat\_folder+AuthenticationService.Users[request.Id].chat\_directory.Name+"\\"+request.FileName;

else

file\_full\_name=files\_folder+request.FileName;

if(File.Exists(file\_full\_name))

file\_full\_name+=$"{DateTime.Now.Ticks}";

files.Add(new FileInfo(file\_full\_name));

using (FileStream file\_stream=files[files.Count-1].OpenWrite())

{

file\_stream.Write(request.Content.ToByteArray());

}

return Task.FromResult(new FileStorageUploadReply

{Result=new ResultReply

{Result=true,

Info=$"File {request.FileName} was successfully uploaded."}});

}

public override Task<FileStorageDownloadReply> download(FileStorageDownloadRequest request, ServerCallContext context)

{

string directory\_name;

if(AuthenticationService.Users[request.Id].id\_connected\_to!=request.Id && AuthenticationService.Users[request.Id].symmetric\_key!=null)

directory\_name=AuthenticationService.Users[request.Id].chat\_directory.Name;

else

directory\_name=files\_folder.Remove(0, 1);

if(!files.Any(f => f.Name==request.FileName && f.Directory.Name==directory\_name))

return Task.FromResult(new FileStorageDownloadReply

{Result=new ResultReply

{Result=false,

Info="ERROR: Requested file doesn't exist."}});

if(AuthenticationService.Users[request.Id].id\_connected\_to!=request.Id && AuthenticationService.Users[request.Id].symmetric\_key!=null)

directory\_name=files\_folder+user\_chat\_folder+directory\_name+"\\";

return Task.FromResult(new FileStorageDownloadReply

{Result=new ResultReply

{Result=true,

Info=$"Downloaded file {request.FileName}."},

Content=ByteString.CopyFrom(File.ReadAllBytes(directory\_name+request.FileName))});

}

public override Task<FileStorageRemoveReply> remove(FileStorageRemoveRequest request, ServerCallContext context)

{

string directory\_name;

if(AuthenticationService.Users[request.Id].id\_connected\_to!=request.Id && AuthenticationService.Users[request.Id].public\_key!=null)

directory\_name=AuthenticationService.Users[request.Id].chat\_directory.Name;

else

directory\_name=files\_folder.Remove(0, 1);

for(int i=0; i<files.Count; i++)

{

if(files[i].Name==request.FileName && files[i].Directory.Name==directory\_name)

{

files.RemoveAt(i);

return Task.FromResult(new FileStorageRemoveReply

{Result=new ResultReply

{Result=true,

Info=$"File {request.FileName} removed from server."}});

}

}

return Task.FromResult(new FileStorageRemoveReply

{Result=new ResultReply

{Result=false,

Info=$"ERROR: File {request.FileName} was not found on server."}});

}

public override Task<FileStorageServerDataReply> getServerData(FileStorageServerDataRequest request, ServerCallContext context)

{

FileStorageServerDataReply reply=getServerData(request.Id);

if(!request.Key.IsEmpty && AuthenticationService.Users[request.Id].symmetric\_key==null && AuthenticationService.Users[request.Id].id\_connected\_to!=request.Id)

{

AuthenticationService.Users[request.Id].symmetric\_key=request.Key.ToByteArray();

AuthenticationService.Users[AuthenticationService.Users[request.Id].id\_connected\_to].status=AuthenticationService.User.Status.PENDING;

}

reply.Result.Result=true;

reply.Result.Info="Data successfully gathered.";

return Task.FromResult(reply);

}

public override Task<FileStorageJoinUserReply> joinUser(FileStorageJoinUserRequest request, ServerCallContext context)

{

if((AuthenticationService.Users[request.IdFrom].id\_connected\_to==request.IdTo) || (AuthenticationService.Users[request.IdFrom].symmetric\_key!=null && AuthenticationService.Users[request.IdTo].symmetric\_key==null))

{

AuthenticationService.Users[request.IdFrom].chat\_directory=null;

AuthenticationService.Users[request.IdFrom].public\_key=null;

AuthenticationService.Users[request.IdFrom].symmetric\_key=null;

AuthenticationService.Users[AuthenticationService.Users[request.IdFrom].id\_connected\_to].id\_connected\_to=AuthenticationService.Users[request.IdFrom].id\_connected\_to;

AuthenticationService.Users[AuthenticationService.Users[request.IdFrom].id\_connected\_to].public\_key=null;

AuthenticationService.Users[AuthenticationService.Users[request.IdFrom].id\_connected\_to].symmetric\_key=null;

AuthenticationService.Users[AuthenticationService.Users[request.IdFrom].id\_connected\_to].chat\_directory=null;

AuthenticationService.Users[request.IdFrom].id\_connected\_to=request.IdFrom;

return Task.FromResult(new FileStorageJoinUserReply()

{Result=new ResultReply()

{Result=false,

Info="Previous connection aborted, try connecting again." }

});

}

if(AuthenticationService.Users[request.IdTo].symmetric\_key!=null)

{

return Task.FromResult(new FileStorageJoinUserReply()

{Result=new ResultReply()

{Result=false,

Info="ERROR: This user is already connected to someone;" }

});

}

AuthenticationService.Users[request.IdFrom].id\_connected\_to=request.IdTo;

AuthenticationService.Users[request.IdTo].id\_connected\_to=request.IdFrom;

AuthenticationService.Users[request.IdFrom].public\_key=new byte[3][];

for(int i=0; i<3; i++)

AuthenticationService.Users[request.IdFrom].public\_key[i]=request.PublicKey[i].ToByteArray();

AuthenticationService.Users[request.IdTo].status=AuthenticationService.User.Status.PENDING;

int d=0;

for( ; d<15; d++)

{

if(AuthenticationService.Users[request.IdTo].public\_key!=null)

break;

Task.Delay(120).Wait();

}

if(d>=15)

{

AuthenticationService.Users[request.IdFrom].public\_key=null;

AuthenticationService.Users[request.IdFrom].id\_connected\_to=request.IdFrom;

AuthenticationService.Users[request.IdTo].id\_connected\_to=request.IdTo;

return Task.FromResult(new FileStorageJoinUserReply()

{Result=new ResultReply()

{Result=false,

Info="ERROR: Connection not estabilished;" }

});

}

AuthenticationService.Users[request.IdFrom].chat\_directory=AuthenticationService.Users[request.IdTo].chat\_directory=new DirectoryInfo($"{files\_folder}user\_chat\\{request.IdFrom}\_{request.IdTo}\_{DateTime.Now.ToString().Replace(":", "")}");

AuthenticationService.Users[request.IdFrom].chat\_directory.Create();

return Task.FromResult(new FileStorageJoinUserReply()

{Result=new ResultReply()

{Result=true,

Info="Connection to other user in progress..." }

});

}

}