

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Российский технологический университет»**

**МИРЭА**

Институт кибернетики

**Кафедра информационной безопасности**

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ №1,2,3,4,5

по дисциплине

«Криптографические протоколы»

На тему:

**«Реализация протокола TLS с помощью криптопримитивов»**

**Подготовил**

студент группы ККСО−01−14 А.С. Першин

**Руководитель работы**

А.П. Никитин

Москва, 2019

Оглавление

[I. *Реализация блочного шифра. Шифр AES* 5](#_Toc10375725)

[1. Описание 5](#_Toc10375726)

[2. Терминология 5](#_Toc10375727)

[3. Шифрование 6](#_Toc10375728)

[3.1 Преобразование SubBytes 7](#_Toc10375729)

[3.2 Преобразование ShiftRows 8](#_Toc10375730)

[3.3 Преобразование MixColomns 8](#_Toc10375731)

[3.4 Преобразование AddRoundKey 9](#_Toc10375732)

[3.5 Процедура KeyExpansion 9](#_Toc10375733)

[4. Расшифрование 10](#_Toc10375734)

[4.1 Преобразование InvShiftRows 11](#_Toc10375735)

[4.2 Преобразование InvSubBytes 12](#_Toc10375736)

[4.3 Преобразование InvMixColomns 12](#_Toc10375737)

[5. Режимы шифрования 12](#_Toc10375738)

[5.1 Режим ECB (Electronic Code Book) 13](#_Toc10375739)

[5.2 Режим OFB (Output Feed Back) 13](#_Toc10375740)

[5.3 Режим CTR (Counter) 14](#_Toc10375741)

[6. Дополнение некратных блоков (Padding) 15](#_Toc10375742)

[7. Результаты реализации алгоритма AES 128/192/256 15](#_Toc10375743)

[Литература 20](#_Toc10375744)

[Листинг кода 20](#_Toc10375745)

[II. *Реализация хеш-функции. Хеш-функция SHA-512* 45](#_Toc10375746)

[1. Описание 45](#_Toc10375747)

[2. Основные операции 45](#_Toc10375748)

[3. Функции и константы 46](#_Toc10375749)

[3.1 Функции 46](#_Toc10375750)

[3.2 Константы 46](#_Toc10375751)

[4. Подготовка к вычислению хеш-значения 46](#_Toc10375752)

[4.1 Дополнение сообщения 46](#_Toc10375753)

[4.2 Получение сообщения 47](#_Toc10375754)

[4.3 Настройка инициализации начальных хеш-значений  47](#_Toc10375755)

[5. Хеш-функция SHA-512 47](#_Toc10375756)

[5.1 Подготовка к алгоритму SHA-512 48](#_Toc10375757)

[5.2 Вычисление хеш-значения сообщения по алгоритму SHA-512 48](#_Toc10375758)

[6. Результаты реализации алгоритма SHA-512 50](#_Toc10375759)

[Литература 51](#_Toc10375760)

[Листинг кода 51](#_Toc10375761)

[III. *Реализация цифровой подписи. Алгоритм ЦП ECDSA* 57](#_Toc10375762)

[1. Описание 57](#_Toc10375763)

[2. Рекомендации NIST по выбору эллиптических кривых 58](#_Toc10375764)

[3. Эллиптические кривые над простыми полями *GF(p)* 58](#_Toc10375765)

[4. Математические операции над эллиптическими кривыми 61](#_Toc10375766)

[5. Параметры пользователя 62](#_Toc10375767)

[6. Формирование цифровой подписи 62](#_Toc10375768)

[7. Проверка цифровой подписи 63](#_Toc10375769)

[8. Результаты реализации алгоритма ECDSA 63](#_Toc10375770)

[Литература 66](#_Toc10375771)

[Листинг кода 66](#_Toc10375772)

[IV. *Реализация ГПСЧ. Алгоритм CSPRNG AES256\_OFB* 78](#_Toc10375773)

[1. Основные понятия о ГПСЧ 78](#_Toc10375774)

[2. Виды ГПСЧ 79](#_Toc10375775)

[3. Пример реализации ГПСЧ 81](#_Toc10375776)

[4. ГПСЧ на основе AES256\_OFB 84](#_Toc10375777)

[5. Результаты реализации ГПСЧ AES256\_OFB 85](#_Toc10375778)

[Литература 89](#_Toc10375779)

[Листинг кода 89](#_Toc10375780)

[V. *Реализация криптографического протокола. Протокол защиты сетевого трафика TLS* 91](#_Toc10375781)

[1. Описание 91](#_Toc10375782)

[2. Принцип работы протокола SSL/TLS 92](#_Toc10375783)

[3. Реализация протокола на основе принципов SSL/TLS 93](#_Toc10375784)

[4. Свойства, характеризующие безопасность протокола 99](#_Toc10375785)

[5. Результаты реализации протокола 100](#_Toc10375786)

[Литература 101](#_Toc10375787)

[Листинг кода 101](#_Toc10375788)

# *Реализация блочного шифра. Шифр AES*

**Лабораторная работа №1**

Для выполнения лабораторной работы по реализации криптографического протокола TLS необходима реализация криптографических примитивов, которые используются при построении протокола. Одним из таковых является блочный шифр, который обеспечивает шифрование/расшифрование информации. В работе будет реализован популярный блочный шифр AES 128/192/256 с режимами шифрования/расшифрования ECB, CTR, OFB.

# Описание

AES представляет собой алгоритм шифрования 128-битных блоков данных ключами по 128, 192 и 256 бит. AES является упрощенной версией алгоритма Rijndael. Оригинальный алгоритм Rijndael отличается тем, что поддерживает более широкий набор длин блоков.

26 мая 2002 года AES был объявлен стандартом шифрования. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования.

Поддержка AES (и только его) введена фирмой Intel в семейство процессоров x86 начиная с Intel Core i7-980X Extreme Edition, а затем на процессорах последующих поколений.

В ходе выполнения работы был изучен документ Fips Pub 197 «Advanced Encryption Standard (AES)» Национального института стандартов и технологий США(NIST).

# Терминология

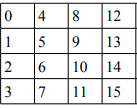
*Байт* ― последовательность из 8 битов. В контексте данного алгоритма байт рассматривается как элемент поля Галуа. Операции над байтами производятся как над элементами поля Галуа , то есть байту соответствует многочлен  в поле .

*Блок* ― последовательность из 16 байтов, над которой оперирует алгоритм. Блок служит входным и выходным данными алгоритма. Байты в блоке нумеруются с нуля.

*Ключ* ― последовательность из 16, 24 или 32 байтов, используемая в качестве ключа шифрования. Байты в ключе нумеруются с нуля. Ключ, наряду с блоком, является входным данным алгоритма.

*Форма* (*State*) ― двумерный массив байтов, состоящий из четырех строк. Байты в форме располагаются в порядке, изображенном в Табл. 1. В алгоритме AES форма используется для представления блока.

**Табл. 1.** Порядок байтов в форме



*Раунд* ― итерация цикла преобразований над формой. В зависимости от длины ключа раундов может быть от 10 до 14, как показано в Табл. 2.

*Ключ раунда* (*round key*) ― ключ, применяемый в раунде. Вычисляется для каждого раунда.

*Таблица подстановок* (*S-box*) ― таблица, задающая биективное отображение байта в байт. Таблица подстановок представлена в Табл. 3.

*Обратная таблица подстановок* (*InvS-box*) ― таблица, задающая отображение, обратное задаваемому таблицей подстановок. Обратная таблица подстановок представлена в Табл. 4.

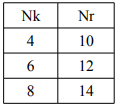
*Nb* ― количество слов (word) в блоке.

*Nk* ― количество слов в ключе.

*Nk* может принимать значения 4, 6, 8.

*Nr* ― количество раундов. Параметр *Nr* зависит от значений *Nk*. Соответствующие значения данных параметров приведены в Табл. 2.

**Табл. 2.** Зависимость *Nr* от *Nk*.



# Шифрование

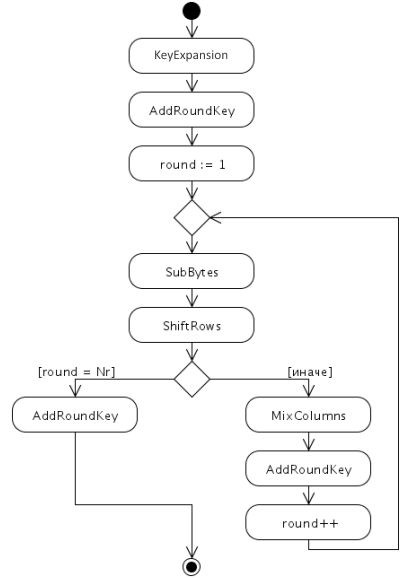
Для шифрования в алгоритме AES применяются следующие процедуры преобразования данных:

1. KeyExpansion — Вычисление раундовых ключей для всех раундов.
2. SubBytes — Подстановка байтов с помощью таблицы подстановок;
3. ShiftRows — Циклический сдвиг строк в форме на различные величины;
4. MixColumns — Смешивание данных внутри каждого столбца формы;
5. AddRoundKey — Сложение ключа раунда с формой.

Порядок выполнения процедур 2 и 3 можно поменять местами в силу линейности этих операций.

Процедуры 4 и 5 тоже можно выполнять в разном порядке, но при этом изменяется количество их вызовов, поскольку MixColumns(AddRoundKey(A, B)) = AddRoundKey(MixColumns(A), MixColumns(B)).

Шифрование производится по алгоритму, приведенному на Риc. 1.

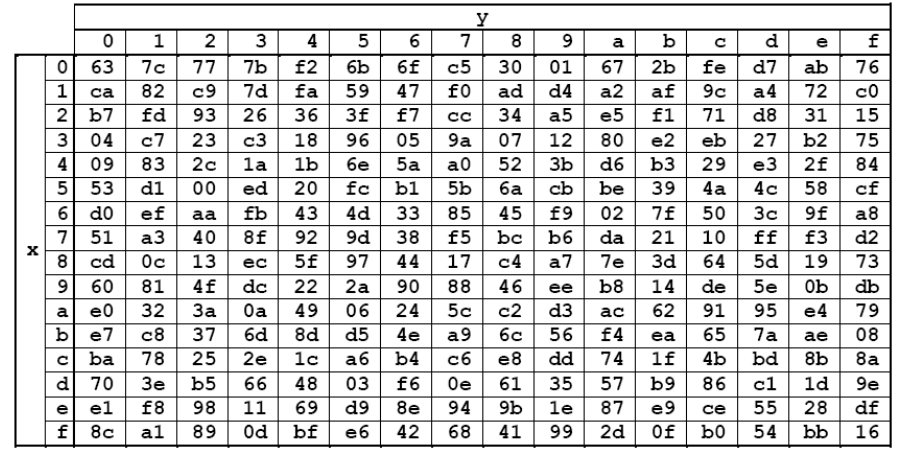


**Риc. 1.** Алгоритм шифрования

## Преобразование SubBytes

Преобразование SubBytes заключается в замене каждого байта {xy} формы (где x и y обозначают шестнадцатиричные цифры) на другой в соответствии с Табл. 3.

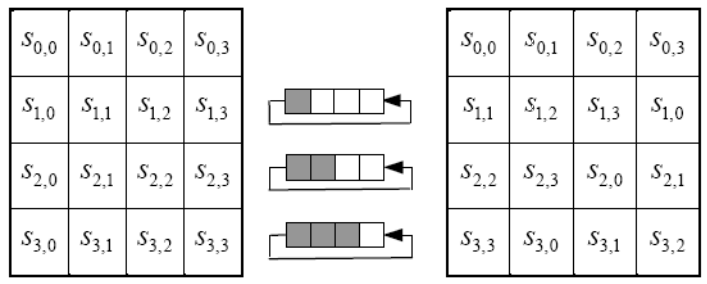
**Табл. 3.** Таблица подстановок



Например, байт {fe} заменится на {bb}.

## Преобразование ShiftRows

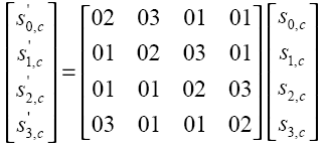
Преобразование ShiftRows заключается в циклическом сдвиге влево строк формы. Преобразование схематично представлено на Риc. 2. Первая строка остается неизменной. Во второй производится сдвиг на 1 байт, то есть первый байт переносится в конец. В третьей ― сдвиг на 2 байта, в четвертой ― на 3.



**Риc. 2.** Преобразование ShiftRows

## Преобразование MixColomns

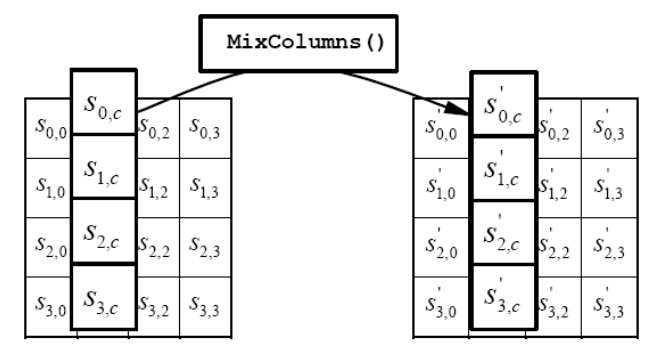
Преобразование MixColumns заключается в умножении квадратной матрицы 4-го порядка на каждый столбец формы:



Умножение производится в поле Галуа .

Над каждым столбцом операция производится отдельно, как показано на Риc. 3.

Для реализации быстрого произведения в поле были использованы готовые таблицы, заменяющие умножения для всех элементов из  на элементы (*0x02* и *0x03*) на подстановку [[ресурс](http://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Rijndael_mix_columns.html)].



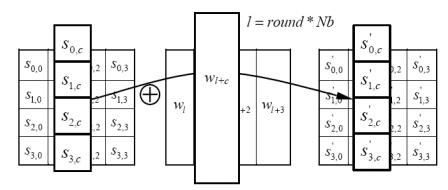
**Риc. 3.** Преобразование MixColumns

## Преобразование AddRoundKey

В преобразовании AddRoundKey 32-битные слова раундового ключа прибавляются к столбцам формы с помощью побитовой операции XOR:



Здесь  ― это столбцы ключа. Над каждым столбцом операция производится отдельно, как показано на Риc. 4.



**Риc. 4.** Преобразование AddRoundKey

## Процедура KeyExpansion

В алгоритме AES генерируются раундовые ключи на основе ключа шифрования с помощью процедуры KeyExpansion. Процедура KeyExpansion создает *Nb* \* (*Nr* + 1) слов: алгоритму требуется начальный ключ размером *Nb*, плюс каждый из *Nr* раундов требует ключ из *Nb* слов. Ниже приведен псевдокод процедуры KeyExpansion:

// Процедура вычисляет ключи раундов.

// key — ключ

// out — результат

// Nk — количество слов в ключе

ExpandKey**(**byte key**[**4**\***Nk**],** word **out[**Nb**\*(**Nr**+**1**)],** int Nk**)**

**begin**

i **=** 0

**while** **(**i **<** Nk**)**

**out[**i**]** **=** word**(**key**[**4**\***i**],** key**[**4**\***i**+**1**],** key**[**4**\***i**+**2**],** key**[**4**\***i**+**3**])**

i **=** i **+** 1

**end** **while**

i **=** Nk

**while** **(**i **<** Nb **\*** **(**Nr**+**1**))**

word temp **=** **out[**i**-**1**]**

**if** **(**i **mod** Nk **=** 0**)**

temp **=** SubWord**(**RotWord**(**temp**))** **xor** Rcon**(**i**/**Nk**)**

**else** **if** **((**Nk **>** 6**)** **and** **(**i **mod** Nk **==** 4**))**

temp **=** SubWord**(**temp**)**

**end** **if**

**out[**i**]** **=** **out[**i**-**Nk**]** **xor** temp

i **=** i **+** 1

**end** **while**

**end**

Здесь использованы следующие функции:

*SubWord* осуществляет замену каждого байта в слове в соответствии с таблицей подстановок, представленной в Табл. 3.

*RotWord* осуществляет циклический сдвиг байтов в слове влево, как показано на Риc. 5.



**Риc. 5.** Процедура RotWord

*Rcon(i)* формирует слово .

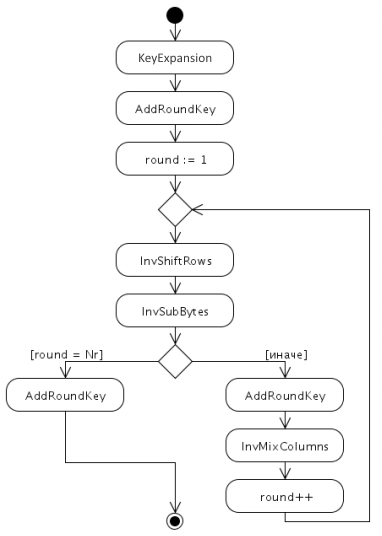
# Расшифрование

При расшифровании все преобразования производятся в обратном порядке. Используются следующие обратные преобразования вместо соответствующих шифрующих:

InvSubBytes ― Подстановка байтов с помощью обратной таблицы подстановок;

InvShiftRows ― Циклический сдвиг строк в форме на различные величины;

InvMixColumns ― Смешивание данных внутри каждого столбца формы;

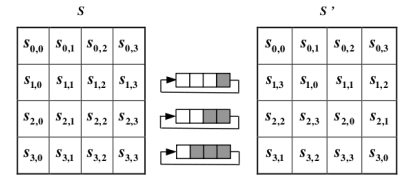


**Риc. 5.** Алгоритм расшифрования

Процедуры KeyExpansion и AddRoundKey остаются неизменными. Ключи раунда используются в обратном порядке. Алгоритм расшифрования представлен на Риc. 6.

## Преобразование InvShiftRows

Это преобразование обратно преобразованию ShiftRows. Первая строка формы остается неизменной. Вторая строка циклически сдвигается вправо на 1 байт. Третья ― на 2, четвертая ― на 3. Схематично преобразование показано на Риc. 7.

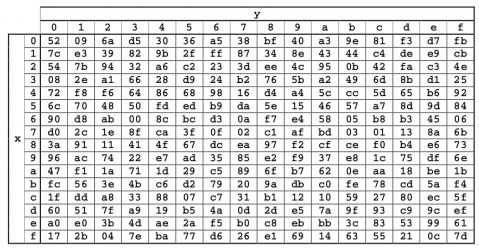


**Риc. 7.** Процедура InvShiftRows

## Преобразование InvSubBytes

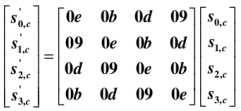
Это преобразование обратно преобразованию SubBytes. Подстановка байтов происходит аналогично с помощью обратной таблицы подстановок, представленной в Табл. 4.

**Табл. 4.** Обратная таблица подстановок



## Преобразование InvMixColomns

Это преобразование обратно преобразованию MixColumns. InvMixColumns преобразует в форме каждый столбец отдельно. Преобразование происходит по следующей формуле:



Здесь умножение также производится в поле Галуа .

Для реализации быстрого произведения в поле были использованы готовые таблицы, заменяющие умножения (для всех элементов из  на элементы *0x09 0x0b* и *0x0d*) на подстановку [[ресурс](http://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Rijndael_mix_columns.html)].

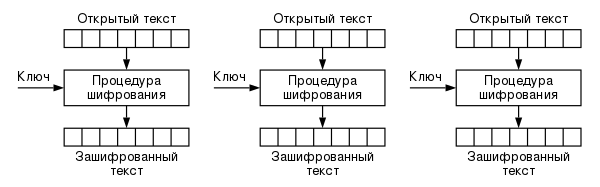
# Режимы шифрования

Далее описаны 3 режима работы алгоритма, которые были реализованы в данной работе на практике.

## Режим ECB (Electronic Code Book)

В режиме ECB каждый блок шифруется независимо от других, как показано на Риc. 8. Таким образом, одинаковые блоки открытого текста преобразуются в одинаковые блоки зашифрованного текста.

Шифрование



**Риc. 8.** Режим ECB

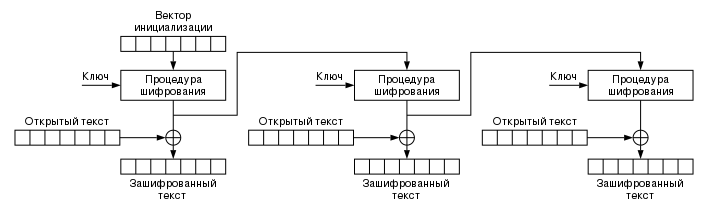
Расшифрование происходит по аналогичной схеме.

В режиме ECB можно производить шифрование и расшифрование нескольких блоков параллельно.

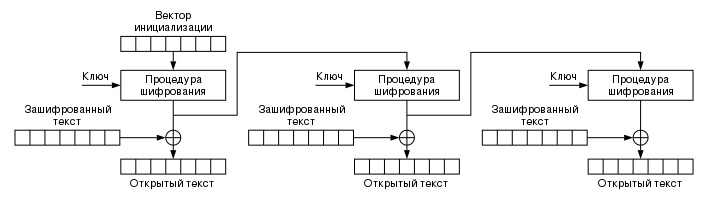
## Режим OFB (Output Feed Back)

В режиме OFB входным блоком служит результат применения шифрования к предыдущему входному блоку. Первым входным блоком служит Initialization Vector.

Шифрование и расшифрование в режиме OFB представлены на Риc. 9 и Риc. 10.



**Риc. 9.** Шифрование в режиме OFB

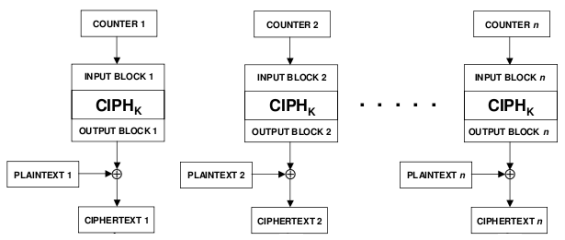


**Риc. 10.** Расшифрование в режиме OFB

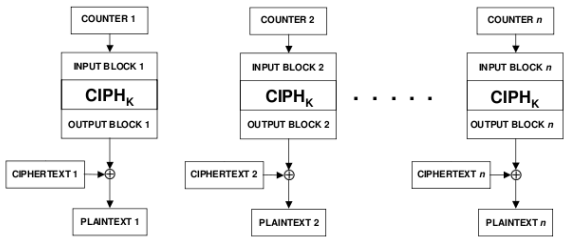
В данном режиме работы шифра шифрование и расшифрование нескольких блоков одновременно произвести не получится.

## Режим CTR (Counter)

В режиме CTR входными блоками являются значения некоторой функции *T(i)*, называемой счетчиком, где *i* — номер блока. Шифрование и расшифрование в режиме CTR представлены на Риc. 11 и Риc. 12.



**Риc. 11.** Шифрование в режиме CTR



**Риc. 12.** Расшифрование в режиме CTR

Режим CTR допускает параллельное шифрование или расшифрование нескольких блоков.

# Дополнение некратных блоков (Padding)

На практике происходят случаи, когда количество исходных данных в блочном шифрование не кратно числу элементов самого блока, для решения этой ситуации есть специальные методы дополнения данных до нужной кратности.

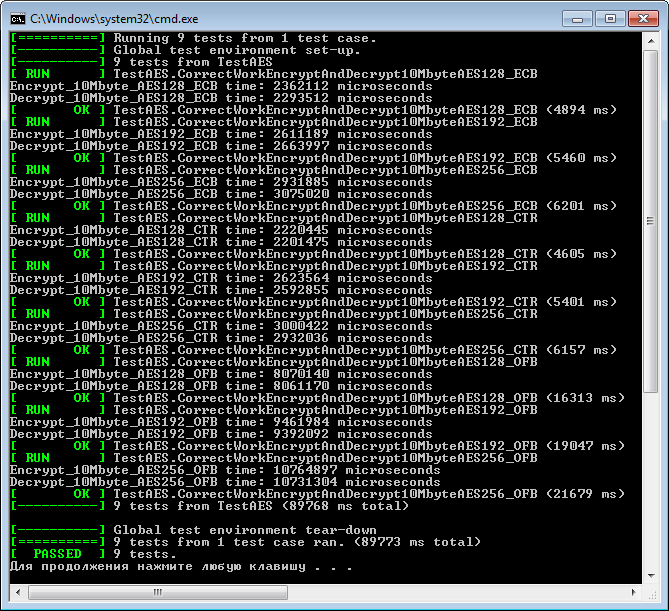
Алгоритм дополнения некратных блоков взят из ГОСТ 34.13-2015 «Режимы работы блочных шифров» из раздела 4.1 «Дополнение сообщения» процедуру №2. Пусть |P|≡r mod L. Положим P\*=P||1||0L-r-1. Где |P| - мощность исходного открытого текста (ОТ), L = 16 – размер входного блока, r - остаток от деления мощности ОТ на размер входного блока. Если r=0, то мы ничего не дополняем, иначе, мы в конце ОТ дополняем его сначала 1(единицей), а затем числом 0(нулей), равным L-r-1. В конечном итоге получаем текст P\* кратный нашему размеру входного блока L.

# Результаты реализации алгоритма AES 128/192/256

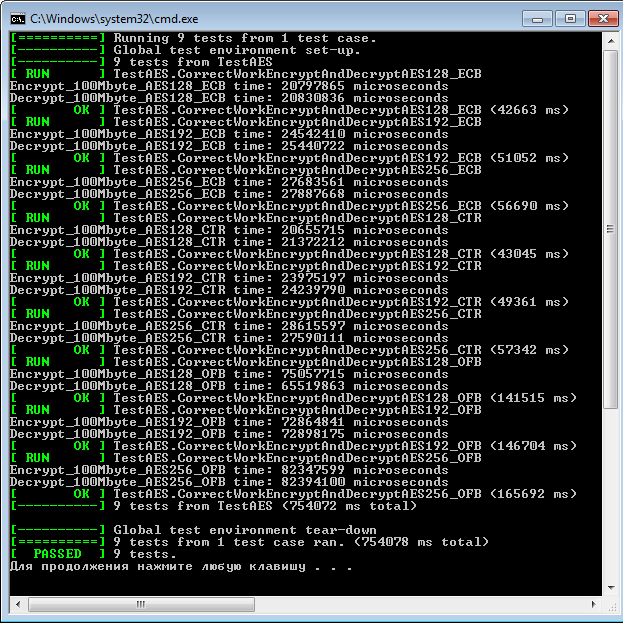
Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация алгоритма AES входит в проект AES\_BlocksCipher решения CryptoProtocols.

Для тестирования корректности разрабатываемых проектов в решении CryptoProtocols был создан отдельный проект GoogleTestingSolutionProject модульного тестирования gtest (для unit testing) и gmock (для проверки корректности вызовов методов). Данные пакеты устанавливались через менеджер пакетов NuGet для Visual Studio.

Результат выполнения тест кейсов для проверки корректности работы функций шифрования/расшифрования одного блока AES 128/192/256 и фиксации времени выполнения для подсчета производительности работы (т.к. gtest замеряет работу вызовов кейсов в микросекундах, то для повышения точности была использована библиотека <chrono> c++11 с точностью до микросекунд) приведены на Рис. 13 и Рис. 14.

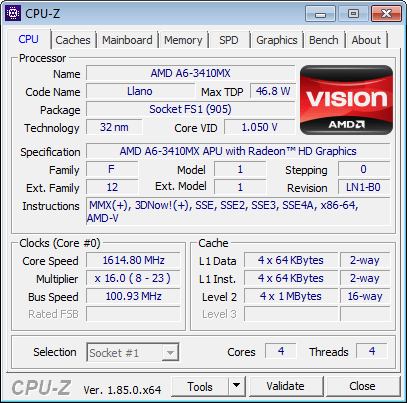
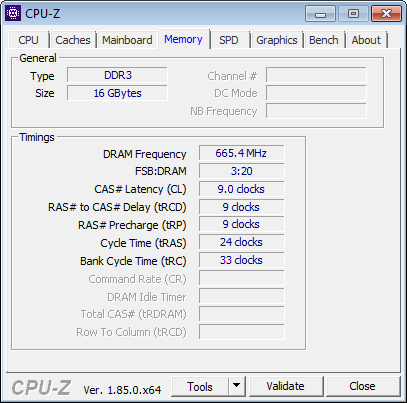


**Рис. 13.** Результат тестирования реализованного алгоритма AES (10 Мбайт данных)



**Рис. 14.** Результат тестирования реализованного алгоритма AES (100 Мбайт данных)

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.15. По полученным данным посчитаем скорость шифрования/расшифрования для данного ЦП. Данные приведены в Табл. 3.

**Рис. 15.** ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

**Табл. 3.** Скорость выполнения шифрования/расшифрования алгоритма AES

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм и размер ключа | Размер данных [Мбайт] | Шифрование/Расшифрование | Скорость [Мбайт/с] |
| В РЕЖИМЕ ECB (Реализован Multi Thread подход) | | | |
| Данные размером 10 Мбайт | | | |
| AES-128 | 10 | Шифрование | 4,2334995123 |
| AES-128 | 10 | Расшифрование | 4,36012543209 |
| AES-192 | 10 | Шифрование | 3,82967299571 |
| AES-192 | 10 | Расшифрование | 3,75375798096 |
| AES-256 | 10 | Шифрование | 3,41077497924 |
| AES-256 | 10 | Расшифрование | 3,25201136903 |
| Данные размером 100 Мбайт | | | |
| AES-128 | 100 | Шифрование | 4,80818584023 |
| AES-128 | 100 | Расшифрование | 4,80057545458 |
| AES-192 | 100 | Шифрование | 4,07457947284 |
| AES-192 | 100 | Расшифрование | 3,93070605465 |
| AES-256 | 100 | Шифрование | 3,61225205096 |
| AES-256 | 100 | Расшифрование | 3,58581434633 |
| В РЕЖИМЕ CTR (Реализован Multi Thread подход) | | | |
| Данные размером 10 Мбайт | | | |
| AES-128 | 10 | Шифрование | 4,5036017555 |
| AES-128 | 10 | Расшифрование | 4,54240906665 |
| AES-192 | 10 | Шифрование | 3,81160894112 |
| AES-192 | 10 | Расшифрование | 3,85675249869 |
| AES-256 | 10 | Шифрование | 3,33286451039 |
| AES-256 | 10 | Расшифрование | 3,41059932416 |
| Данные размером 100 Мбайт | | | |
| AES-128 | 100 | Шифрование | 4,84127516283 |
| AES-128 | 100 | Расшифрование | 4,67897286439 |
| AES-192 | 100 | Шифрование | 4,17097719781 |
| AES-192 | 100 | Расшифрование | 4,12544828152 |
| AES-256 | 100 | Шифрование | 3,49459771886 |
| AES-256 | 100 | Расшифрование | 3,624487049 |
| В РЕЖИМЕ OFB (Multi Thread подход не предусмотрен структурой) | | | |
| Данные размером 10 Мбайт | | | |
| AES-128 | 10 | Шифрование | 1,23913587621 |
| AES-128 | 10 | Расшифрование | 1,24051471437 |
| AES-192 | 10 | Шифрование | 1,05686080213 |
| AES-192 | 10 | Расшифрование | 1,06472551589 |
| AES-256 | 10 | Шифрование | 0,92894525604 |
| AES-256 | 10 | Расшифрование | 0,93185320255 |
| Данные размером 100 Мбайт | | | |
| AES-128 | 100 | Шифрование | 1,33230807786 |
| AES-128 | 100 | Расшифрование | 1,5262547176 |
| AES-192 | 100 | Шифрование | 1,37240401032 |
| AES-192 | 100 | Расшифрование | 1,37177645394 |
| AES-256 | 100 | Шифрование | 1,21436448924 |
| AES-256 | 100 | Расшифрование | 1,21367913479 |

# Литература

1. «Rijndael MixColomns» [Интернет ресурс], ссылка: <https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Rijndael_mix_columns.html>
2. FIPS PUB 197 «ADVANCED ENCRYPTION STANDARD (AES)» [Интернет ресурс], ссылка https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

# Листинг кода

файл Rijndael.h

#ifndef RIJNDAEL\_H

#define RIJNDAEL\_H

/\*

Federal Information

Processing Standards Publication 197

November 26, 2001

Announcing the

ADVANCED ENCRYPTION STANDARD (AES)

RUSSIAN TECHNOLOGICAL UNIVERSITY [RTU MIREA]

REALIZATION Rijndael Algorithm Block Cipher

\*/

#include <iostream>

#include <string>

#include <vector>

#include <iterator>

using namespace std;

class Rijndael {

private:

uint8\_t Nb{ 0 }; // number of rows in Matrix State, in standard FIPS197 this value is 4

uint8\_t Nk{ 0 }; // key length variable

uint8\_t Nr{ 0 }; // nuMber of rounds

vector<uint8\_t>\* Key; // array of Key

vector<uint8\_t>\* RoundKeys; // array of RoundKeys

//vector<vector<uint8\_t>>\* State; // matrix of State

/\*Crypt Functions\*/

void SubBytes(vector<vector<uint8\_t>>\* State);

void ShiftRows(vector<vector<uint8\_t>>\* State);

void MixColomns(vector<vector<uint8\_t>>\* State);

void AddRoundKey(uint8\_t byCurrentRound, vector<vector<uint8\_t>>\* State);

void KeyExpansion();

/\*Decrypt Function\*/

void InvShiftRows(vector<vector<uint8\_t>>\* State);

void InvSubBytes(vector<vector<uint8\_t>>\* State);

void InvMixColomns(vector<vector<uint8\_t>>\* State);

public:

vector<uint8\_t> Encrypt(vector<uint8\_t>& arrbyBlockPlainText, vector<uint8\_t>\* byarrKey);

vector<uint8\_t> Decrypt(vector<uint8\_t>& arrbyBlockCipherText, vector<uint8\_t>\* byarrKey);

Rijndael(uint8\_t, uint8\_t, uint8\_t);

Rijndael() : Rijndael(4, 4, 10) {};

~Rijndael();

};

#endif //RIJNDAEL\_H

файл Rijndael.cpp

#include "Rijndael.h"

#include "Tables.h"

#include <iostream>

using namespace std;

void Rijndael::SubBytes(vector<vector<uint8\_t>>\* State){

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) { (\*State)[j][i] = Sbox[(\*State)[j][i]]; }

}

};

void Rijndael::ShiftRows(vector<vector<uint8\_t>>\* State){

//Shift 1 row

swap((\*State)[0][1], (\*State)[3][1]);

swap((\*State)[0][1], (\*State)[1][1]);

swap((\*State)[1][1], (\*State)[2][1]);

//Shift 2 row

swap((\*State)[0][2], (\*State)[2][2]);

swap((\*State)[1][2], (\*State)[3][2]);

//Shift 3 row

swap((\*State)[0][3], (\*State)[3][3]);

swap((\*State)[1][3], (\*State)[3][3]);

swap((\*State)[2][3], (\*State)[3][3]);

};

void Rijndael::MixColomns(vector<vector<uint8\_t>>\* State){

vector<vector<uint8\_t>> TempState(\*State);

//MixColoms 0,1,2,3; Callc Cells in Colomn[i]

for (uint8\_t i = 0; i < 4; i++) {

(\*State)[i][0] = mul0x02[TempState[i][0]] ^ mul0x03[TempState[i][1]] ^ TempState[i][2] ^ TempState[i][3];

(\*State)[i][1] = TempState[i][0] ^ mul0x02[TempState[i][1]] ^ mul0x03[TempState[i][2]] ^ TempState[i][3];

(\*State)[i][2] = TempState[i][0] ^ TempState[i][1] ^ mul0x02[TempState[i][2]] ^ mul0x03[TempState[i][3]];

(\*State)[i][3] = mul0x03[TempState[i][0]] ^ TempState[i][1] ^ TempState[i][2] ^ mul0x02[TempState[i][3]];

}

TempState.clear();

};

void Rijndael::AddRoundKey(uint8\_t byCurrentRound, vector<vector<uint8\_t>>\* State){

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) { (\*State)[i][j] ^= (\*RoundKeys)[byCurrentRound \* Nb \* Nb + i \* Nb + j]; }

}

};

void Rijndael::KeyExpansion(){

for (uint8\_t i = 0; i < Nk; i++) {

(\*RoundKeys)[i \* Nb] = (\*Key)[i \* 4];

(\*RoundKeys)[i \* Nb + 1] = (\*Key)[i \* Nb + 1];

(\*RoundKeys)[i \* Nb + 2] = (\*Key)[i \* Nb + 2];

(\*RoundKeys)[i \* Nb + 3] = (\*Key)[i \* Nb + 3];

}

uint8\_t i = Nk, byarrWord[4];

while (i < (Nb\*(Nr+1))){

for (uint8\_t j = 0; j < 4; j++) { byarrWord[j] = (\*RoundKeys)[(i - 1) \* Nb + j]; }

if (i%Nk == 0) {

//RotWord Function

{

swap(byarrWord[0], byarrWord[3]);

swap(byarrWord[0], byarrWord[1]);

swap(byarrWord[1], byarrWord[2]);

}

//SubWord Function

{

byarrWord[0] = Sbox[byarrWord[0]];

byarrWord[1] = Sbox[byarrWord[1]];

byarrWord[2] = Sbox[byarrWord[2]];

byarrWord[3] = Sbox[byarrWord[3]];

}

byarrWord[0] = byarrWord[0] ^ Rcon[i / Nk];

}

else if ((Nk > 6) && (i%Nk == 4)) {

//SubWord Function

{

byarrWord[0] = Sbox[byarrWord[0]];

byarrWord[1] = Sbox[byarrWord[1]];

byarrWord[2] = Sbox[byarrWord[2]];

byarrWord[3] = Sbox[byarrWord[3]];

}

}

(\*RoundKeys)[i \* Nb + 0] = (\*RoundKeys)[(i - Nk) \* Nb + 0] ^ byarrWord[0];

(\*RoundKeys)[i \* Nb + 1] = (\*RoundKeys)[(i - Nk) \* Nb + 1] ^ byarrWord[1];

(\*RoundKeys)[i \* Nb + 2] = (\*RoundKeys)[(i - Nk) \* Nb + 2] ^ byarrWord[2];

(\*RoundKeys)[i \* Nb + 3] = (\*RoundKeys)[(i - Nk) \* Nb + 3] ^ byarrWord[3];

i++;

}

};

void Rijndael::InvShiftRows(vector<vector<uint8\_t>>\* State){

//Shift 1 row

swap((\*State)[0][1], (\*State)[3][1]);

swap((\*State)[3][1], (\*State)[1][1]);

swap((\*State)[3][1], (\*State)[2][1]);

//Shift 2 row

swap((\*State)[0][2], (\*State)[2][2]);

swap((\*State)[1][2], (\*State)[3][2]);

//Shift 3 row

swap((\*State)[0][3], (\*State)[3][3]);

swap((\*State)[0][3], (\*State)[1][3]);

swap((\*State)[2][3], (\*State)[1][3]);

};

void Rijndael::InvSubBytes(vector<vector<uint8\_t>>\* State){

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) { (\*State)[j][i] = InvSbox[(\*State)[j][i]]; }

}

};

void Rijndael::InvMixColomns(vector<vector<uint8\_t>>\* State){

vector<vector<uint8\_t>> TempState(\*State);

//InvMixColoms 0,1,2,3; Callc Cells in Colomn[i]

for (uint8\_t i = 0; i < 4; i++) {

(\*State)[i][0] = mul0x0e[TempState[i][0]] ^ mul0x0b[TempState[i][1]] ^ mul0x0d[TempState[i][2]] ^ mul0x09[TempState[i][3]];

(\*State)[i][1] = mul0x09[TempState[i][0]] ^ mul0x0e[TempState[i][1]] ^ mul0x0b[TempState[i][2]] ^ mul0x0d[TempState[i][3]];

(\*State)[i][2] = mul0x0d[TempState[i][0]] ^ mul0x09[TempState[i][1]] ^ mul0x0e[TempState[i][2]] ^ mul0x0b[TempState[i][3]];

(\*State)[i][3] = mul0x0b[TempState[i][0]] ^ mul0x0d[TempState[i][1]] ^ mul0x09[TempState[i][2]] ^ mul0x0e[TempState[i][3]];

}

TempState.clear();

};

vector<uint8\_t> Rijndael::Encrypt(vector<uint8\_t>& arrbyBlockPlainText, vector<uint8\_t>\* byarrKey){

//Create matrix State with size 4\*Nb and Value in Cells is 0

auto State = new vector<vector<uint8\_t>>(Nb, vector<uint8\_t>(Nb, 0));

//Expansion Work Key

if (Key != byarrKey) {

Key = byarrKey;

KeyExpansion();

}

//Create Buffer Block Cipher Text

vector<uint8\_t> arrbyBlockCipherText;

//Add OT Block in State

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) {

(\*State)[i][j] = arrbyBlockPlainText[i \* Nb + j];

}

}

//Round {0}

AddRoundKey(0, State);

//Rounds {1, 2, 3, ..., 9} or {1, 2, 3, ..., 11} or {1, 2, 3, ..., 13}

for (uint8\_t byCurrentRound = 1; byCurrentRound < Nr; byCurrentRound++) {

SubBytes(State);

ShiftRows(State);

MixColomns(State);

AddRoundKey(byCurrentRound, State);

}

//Last Round {10} or {12} or {14}

SubBytes(State);

ShiftRows(State);

AddRoundKey(Nr, State);

//Write CipherTextBlock in arrbyBufferCipherText

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) { arrbyBlockCipherText.push\_back((\*State)[i][j]); }

}

delete State;

return arrbyBlockCipherText;

};

vector<uint8\_t> Rijndael::Decrypt(vector<uint8\_t>& arrbyBlockCipherText, vector<uint8\_t>\* byarrKey){

//Create matrix State with size 4\*Nb and Value in Cells is 0

auto State = new vector<vector<uint8\_t>>(Nb, vector<uint8\_t>(Nb, 0));

//Expansion Work Key

if (Key != byarrKey) {

Key = byarrKey;

KeyExpansion();

}

//Create Buffer Block Plain Text

vector<uint8\_t> arrBlockPlainText;

//Add OT Block in State

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) {

(\*State)[i][j] = arrbyBlockCipherText[i \* Nb + j];

}

}

//Round {0}

AddRoundKey(Nr, State);

//Rounds {1, 2, 3, ..., 9} or {1, 2, 3, ..., 11} or {1, 2, 3, ..., 13}

for (uint8\_t byCurrentRound = Nr - 1; byCurrentRound > 0; byCurrentRound--) {

InvShiftRows(State);

InvSubBytes(State);

AddRoundKey(byCurrentRound, State);

InvMixColomns(State);

}

//Last Round {10} or {12} or {14}

InvSubBytes(State);

InvShiftRows(State);

AddRoundKey(0, State);

//Write CipherTextBlock in arrbyBufferCipherText

for (uint8\_t i = 0; i < 4; i++) {

for (uint8\_t j = 0; j < 4; j++) { arrBlockPlainText.push\_back((\*State)[i][j]); }

}

delete State;

return arrBlockPlainText;

};

Rijndael::Rijndael(uint8\_t valueNb, uint8\_t valueNk, uint8\_t valueNr) {

Nb = valueNb;

Nk = valueNk;

Nr = valueNr;

RoundKeys = new vector<uint8\_t>((Nb\*(Nr + 1)\*Nb), 0); //create buffer of RoundKeys size Nb\*(Nr+1)\*Nb and Value in Cells is 0

};

Rijndael::~Rijndael() {

memset(RoundKeys->data(), 0x00, RoundKeys->size()); //Security Clear buffer RoundKeys

delete RoundKeys;

};

файл Tables.h

#ifndef TABLES\_H

#define TABLES\_H

/\*Tables for Crypt Rijndael\*/

static uint8\_t Rcon[255] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, //0

0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, //1

0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, //2

0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, //3

0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, //4

0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, //5

0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, //6

0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, //7

0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, //8

0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, //9

0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, //A

0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, //B

0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, //C

0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, //D

0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, //E

0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb //F

};

static uint8\_t Sbox[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D

0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E

0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 //F

};

static uint8\_t mul0x02[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,

0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,

0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,

0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,

0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,

0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,

0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,

0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,

0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,

0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,

0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,

0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,

0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,

0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,

0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,

0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5

};

static uint8\_t mul0x03[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11,

0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21,

0x60, 0x63, 0x66, 0x65, 0x6c, 0x6F, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,

0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41,

0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1,

0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,

0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1,

0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81,

0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,

0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba,

0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea,

0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,

0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a,

0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a,

0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,

0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a

};

/\*Tables for Decrypt Rijndael\*/

static uint8\_t InvSbox[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, //0

0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, //1

0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e, //2

0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, //3

0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, //4

0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84, //5

0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, //6

0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, //7

0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, //8

0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, //9

0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, //A

0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4, //B

0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, //C

0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, //D

0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61, //E

0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d //F

};

static uint8\_t mul0x09[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53, 0x6c, 0x65, 0x7e, 0x77,

0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca, 0xc3, 0xfc, 0xf5, 0xee, 0xe7,

0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,

0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8, 0xc7, 0xce, 0xd5, 0xdc,

0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25, 0x1a, 0x13, 0x08, 0x01,

0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5, 0x8a, 0x83, 0x98, 0x91,

0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e, 0x21, 0x28, 0x33, 0x3a,

0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e, 0xb1, 0xb8, 0xa3, 0xaa,

0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf, 0x80, 0x89, 0x92, 0x9b,

0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f, 0x10, 0x19, 0x02, 0x0b,

0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84, 0xbb, 0xb2, 0xa9, 0xa0,

0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14, 0x2b, 0x22, 0x39, 0x30,

0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9, 0xf6, 0xff, 0xe4, 0xed,

0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59, 0x66, 0x6f, 0x74, 0x7d,

0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2, 0xcd, 0xc4, 0xdf, 0xd6,

0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b, 0x62, 0x5d, 0x54, 0x4f, 0x46

};

static uint8\_t mul0x0b[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x0b, 0x16, 0x1d, 0x2c, 0x27, 0x3a, 0x31, 0x58, 0x53, 0x4e, 0x45, 0x74, 0x7f, 0x62, 0x69,

0xb0, 0xbb, 0xa6, 0xad, 0x9C, 0x97, 0x8a, 0x81, 0xe8, 0xe3, 0xfe, 0xf5, 0xc4, 0xcf, 0xd2, 0xd9,

0x7b, 0x70, 0x6d, 0x66, 0x57, 0x5c, 0x41, 0x4a, 0x23, 0x28, 0x35, 0x3e, 0x0f, 0x04, 0x19, 0x12,

0xcb, 0xc0, 0xdd, 0xd6, 0xe7, 0xec, 0xf1, 0xfa, 0x93, 0x98, 0x85, 0x8e, 0xbf, 0xb4, 0xa9, 0xa2,

0xf6, 0xfd, 0xe0, 0xeb, 0xda, 0xd1, 0xcc, 0xc7, 0xae, 0xa5, 0xb8, 0xb3, 0x82, 0x89, 0x94, 0x9f,

0x46, 0x4d, 0x50, 0x5b, 0x6a, 0x61, 0x7c, 0x77, 0x1e, 0x15, 0x08, 0x03, 0x32, 0x39, 0x24, 0x2f,

0x8d, 0x86, 0x9b, 0x90, 0xa1, 0xaa, 0xb7, 0xbc, 0xd5, 0xde, 0xc3, 0xc8, 0xf9, 0xf2, 0xef, 0xe4,

0x3d, 0x36, 0x2b, 0x20, 0x11, 0x1a, 0x07, 0x0c, 0x65, 0x6e, 0x73, 0x78, 0x49, 0x42, 0x5f, 0x54,

0xf7, 0xfc, 0xe1, 0xea, 0xdb, 0xd0, 0xcd, 0xc6, 0xaf, 0xa4, 0xb9, 0xb2, 0x83, 0x88, 0x95, 0x9e,

0x47, 0x4c, 0x51, 0x5a, 0x6b, 0x60, 0x7d, 0x76, 0x1f, 0x14, 0x09, 0x02, 0x33, 0x38, 0x25, 0x2e,

0x8c, 0x87, 0x9a, 0x91, 0xa0, 0xab, 0xb6, 0xbd, 0xd4, 0xdf, 0xc2, 0xc9, 0xf8, 0xf3, 0xee, 0xe5,

0x3c, 0x37, 0x2a, 0x21, 0x10, 0x1b, 0x06, 0x0d, 0x64, 0x6f, 0x72, 0x79, 0x48, 0x43, 0x5e, 0x55,

0x01, 0x0a, 0x17, 0x1c, 0x2d, 0x26, 0x3b, 0x30, 0x59, 0x52, 0x4f, 0x44, 0x75, 0x7e, 0x63, 0x68,

0xb1, 0xba, 0xa7, 0xac, 0x9d, 0x96, 0x8b, 0x80, 0xe9, 0xe2, 0xff, 0xf4, 0xc5, 0xce, 0xd3, 0xd8,

0x7A, 0x71, 0x6c, 0x67, 0x56, 0x5d, 0x40, 0x4b, 0x22, 0x29, 0x34, 0x3f, 0x0e, 0x05, 0x18, 0x13,

0xca, 0xc1, 0xdc, 0xd7, 0xe6, 0xed, 0xf0, 0xfb, 0x92, 0x99, 0x84, 0x8f, 0xbe, 0xb5, 0xa8, 0xa3

};

static uint8\_t mul0x0d[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x0d, 0x1a, 0x17, 0x34, 0x39, 0x2e, 0x23, 0x68, 0x65, 0x72, 0x7f, 0x5c, 0x51, 0x46, 0x4b,

0xd0, 0xdd, 0xca, 0xc7, 0xe4, 0xe9, 0xfe, 0xf3, 0xb8, 0xb5, 0xa2, 0xaf, 0x8c, 0x81, 0x96, 0x9b,

0xbb, 0xb6, 0xa1, 0xac, 0x8f, 0x82, 0x95, 0x98, 0xd3, 0xde, 0xc9, 0xc4, 0xe7, 0xea, 0xfd, 0xf0,

0x6b, 0x66, 0x71, 0x7c, 0x5f, 0x52, 0x45, 0x48, 0x03, 0x0e, 0x19, 0x14, 0x37, 0x3a, 0x2d, 0x20,

0x6d, 0x60, 0x77, 0x7a, 0x59, 0x54, 0x43, 0x4e, 0x05, 0x08, 0x1f, 0x12, 0x31, 0x3c, 0x2b, 0x26,

0xbd, 0xb0, 0xa7, 0xaa, 0x89, 0x84, 0x93, 0x9e, 0xd5, 0xd8, 0xcf, 0xc2, 0xe1, 0xec, 0xfb, 0xf6,

0xd6, 0xdb, 0xcc, 0xc1, 0xe2, 0xef, 0xf8, 0xf5, 0xbe, 0xb3, 0xa4, 0xa9, 0x8a, 0x87, 0x90, 0x9d,

0x06, 0x0b, 0x1c, 0x11, 0x32, 0x3f, 0x28, 0x25, 0x6e, 0x63, 0x74, 0x79, 0x5a, 0x57, 0x40, 0x4d,

0xda, 0xd7, 0xc0, 0xcd, 0xee, 0xe3, 0xf4, 0xf9, 0xb2, 0xbf, 0xa8, 0xa5, 0x86, 0x8b, 0x9C, 0x91,

0x0a, 0x07, 0x10, 0x1d, 0x3e, 0x33, 0x24, 0x29, 0x62, 0x6f, 0x78, 0x75, 0x56, 0x5b, 0x4c, 0x41,

0x61, 0x6c, 0x7b, 0x76, 0x55, 0x58, 0x4f, 0x42, 0x09, 0x04, 0x13, 0x1e, 0x3d, 0x30, 0x27, 0x2a,

0xb1, 0xbc, 0xab, 0xa6, 0x85, 0x88, 0x9f, 0x92, 0xd9, 0xd4, 0xc3, 0xce, 0xed, 0xe0, 0xf7, 0xfa,

0xb7, 0xba, 0xad, 0xa0, 0x83, 0x8e, 0x99, 0x94, 0xdf, 0xd2, 0xc5, 0xc8, 0xeb, 0xe6, 0xf1, 0xfc,

0x67, 0x6a, 0x7d, 0x70, 0x53, 0x5e, 0x49, 0x44, 0x0f, 0x02, 0x15, 0x18, 0x3b, 0x36, 0x21, 0x2c,

0x0c, 0x01, 0x16, 0x1b, 0x38, 0x35, 0x22, 0x2f, 0x64, 0x69, 0x7e, 0x73, 0x50, 0x5d, 0x4a, 0x47,

0xdc, 0xd1, 0xc6, 0xcb, 0xe8, 0xe5, 0xf2, 0xff, 0xb4, 0xb9, 0xae, 0xa3, 0x80, 0x8d, 0x9a, 0x97

};

static uint8\_t mul0x0e[256] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x00, 0x0e, 0x1c, 0x12, 0x38, 0x36, 0x24, 0x2a, 0x70, 0x7e, 0x6c, 0x62, 0x48, 0x46, 0x54, 0x5a,

0xe0, 0xee, 0xfc, 0xf2, 0xd8, 0xd6, 0xc4, 0xca, 0x90, 0x9e, 0x8c, 0x82, 0xa8, 0xa6, 0xb4, 0xba,

0xdb, 0xd5, 0xc7, 0xc9, 0xe3, 0xed, 0xff, 0xf1, 0xab, 0xa5, 0xb7, 0xb9, 0x93, 0x9d, 0x8f, 0x81,

0x3b, 0x35, 0x27, 0x29, 0x03, 0x0d, 0x1f, 0x11, 0x4b, 0x45, 0x57, 0x59, 0x73, 0x7d, 0x6f, 0x61,

0xad, 0xa3, 0xb1, 0xbf, 0x95, 0x9b, 0x89, 0x87, 0xdd, 0xd3, 0xc1, 0xcf, 0xe5, 0xeb, 0xf9, 0xf7,

0x4d, 0x43, 0x51, 0x5f, 0x75, 0x7b, 0x69, 0x67, 0x3d, 0x33, 0x21, 0x2f, 0x05, 0x0b, 0x19, 0x17,

0x76, 0x78, 0x6a, 0x64, 0x4e, 0x40, 0x52, 0x5c, 0x06, 0x08, 0x1a, 0x14, 0x3e, 0x30, 0x22, 0x2c,

0x96, 0x98, 0x8a, 0x84, 0xae, 0xa0, 0xb2, 0xbc, 0xe6, 0xe8, 0xfa, 0xf4, 0xde, 0xd0, 0xc2, 0xcc,

0x41, 0x4f, 0x5d, 0x53, 0x79, 0x77, 0x65, 0x6b, 0x31, 0x3f, 0x2d, 0x23, 0x09, 0x07, 0x15, 0x1b,

0xa1, 0xaf, 0xbd, 0xb3, 0x99, 0x97, 0x85, 0x8b, 0xd1, 0xdf, 0xcd, 0xc3, 0xe9, 0xe7, 0xf5, 0xfb,

0x9A, 0x94, 0x86, 0x88, 0xa2, 0xac, 0xbe, 0xb0, 0xea, 0xe4, 0xf6, 0xf8, 0xd2, 0xdc, 0xce, 0xc0,

0x7A, 0x74, 0x66, 0x68, 0x42, 0x4c, 0x5e, 0x50, 0x0a, 0x04, 0x16, 0x18, 0x32, 0x3c, 0x2e, 0x20,

0xec, 0xe2, 0xf0, 0xfe, 0xd4, 0xda, 0xc8, 0xc6, 0x9c, 0x92, 0x80, 0x8e, 0xa4, 0xaa, 0xb8, 0xb6,

0x0c, 0x02, 0x10, 0x1e, 0x34, 0x3a, 0x28, 0x26, 0x7c, 0x72, 0x60, 0x6e, 0x44, 0x4a, 0x58, 0x56,

0x37, 0x39, 0x2b, 0x25, 0x0f, 0x01, 0x13, 0x1d, 0x47, 0x49, 0x5b, 0x55, 0x7f, 0x71, 0x63, 0x6d,

0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5, 0x9f, 0x91, 0x83, 0x8d

};

#endif //TABLES\_H

файл AES.h

#ifndef AES\_H

#define AES\_H

/\*

Federal Information

Processing Standards Publication 197

November 26, 2001

Announcing the

ADVANCED ENCRYPTION STANDARD (AES)

RUSSIAN TECHNOLOGICAL UNIVERSITY [RTU MIREA]

REALIZATION AES 128/192/256 Block Cipher

\*/

/\*

ANNOTATION:

-> When you are create instance of AES\_(128/192/256) the default encryption mode is ECB encryption mode

-> SetEncryptionMode allows change encryption mode in Runtime

-> EncryptionModeId is:

--> '0' - ECB encryption mode;

--> '1' - CTR encryption mode;

--> '2' - OFB encryption mode;

--> 'other' - save previous mode;

->

\*/

#include "Rijndael.h"

#include "EncryptionMode.h"

/\*Interface class AES\*/

class IAES {

protected:

IEncryptionMode\* \_pEncryptionMode{ nullptr };

Rijndael\* \_pRijndael{ nullptr };

uint8\_t Nb{ 0 };

uint8\_t Nk{ 0 };

uint8\_t Nr{ 0 };

public:

virtual vector<uint8\_t>\* Encrypt(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) = 0;

virtual vector<uint8\_t>\* Decrypt(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) = 0;

void SetEncryptionMode(uint8\_t EncryptionModeID);

};

/\*AES 128 Class\*/

class AES\_128 : public IAES {

public:

vector<uint8\_t>\* Encrypt(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decrypt(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

AES\_128() {

Nb = 4;

Nk = 4;

Nr = 10;

\_pRijndael = new Rijndael(Nb,Nk,Nr);

\_pEncryptionMode = new ECB(\_pRijndael);

}

~AES\_128() {

delete \_pEncryptionMode;

delete \_pRijndael;

}

};

/\*AES 192 Class\*/

class AES\_192 : public IAES {

public:

vector<uint8\_t>\* Encrypt(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decrypt(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

AES\_192() {

Nb = 4;

Nk = 6;

Nr = 12;

\_pRijndael = new Rijndael(Nb, Nk, Nr);

\_pEncryptionMode = new ECB(\_pRijndael);

}

~AES\_192() {

delete \_pEncryptionMode;

delete \_pRijndael;

}

};

/\*AES 256 Class\*/

class AES\_256 : public IAES {

public:

vector<uint8\_t>\* Encrypt(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decrypt(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

AES\_256() {

Nb = 4;

Nk = 8;

Nr = 14;

\_pRijndael = new Rijndael(Nb, Nk, Nr);

\_pEncryptionMode = new ECB(\_pRijndael);

}

~AES\_256() {

delete \_pEncryptionMode;

delete \_pRijndael;

}

};

#endif

файл AES.cpp

#include "AES.h"

#include "AES.h"

/\*Start InterfaceAES Methods Realization\*/

void IAES::SetEncryptionMode(uint8\_t EncryptionModeID){

switch (EncryptionModeID) {

//EncryptionModeID = '0' is ECB mode

case 0: {

delete \_pEncryptionMode;

\_pEncryptionMode = new ECB(\_pRijndael);

break;

}

//EncryptionModeID = '1' is CTR mode

case 1: {

delete \_pEncryptionMode;

\_pEncryptionMode = new CTR(\_pRijndael);

break;

}

//EncryptionModeID = '2' is OFB mode

case 2: {

delete \_pEncryptionMode;

\_pEncryptionMode = new OFB(\_pRijndael);

break;

}

default: break;

}

}

/\*End InterfaceAES Methods Realization\*/

/\*Start AES 128 Methods Realization\*/

vector<uint8\_t>\* AES\_128::Encrypt(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey) {

if ((!byarrBufferPlainText->empty() && byarrBufferPlainText != nullptr) && (!byarrKey->empty() && byarrKey->size() % 16 == 0)) {

return \_pEncryptionMode->Encryption(byarrBufferPlainText, byarrKey);

}

else {

return nullptr;

}

}

vector<uint8\_t>\* AES\_128::Decrypt(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey){

if ((!byarrBufferCipherText->empty() && byarrBufferCipherText != nullptr && byarrBufferCipherText->size() % 16 == 0) && (!byarrKey->empty() && byarrKey->size() % 16 == 0)) {

return \_pEncryptionMode->Decryption(byarrBufferCipherText, byarrKey);

}

else {

return nullptr;

}

}

/\*End AES 128 Methods Realization\*/

/\*Start AES 192 Methods Realization\*/

vector<uint8\_t>\* AES\_192::Encrypt(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey){

if ((!byarrBufferPlainText->empty() && byarrBufferPlainText != nullptr) && (!byarrKey->empty() && byarrKey->size() % 24 == 0)) {

return \_pEncryptionMode->Encryption(byarrBufferPlainText, byarrKey);

}

else {

return nullptr;

}

}

vector<uint8\_t>\* AES\_192::Decrypt(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey){

if ((!byarrBufferCipherText->empty() && byarrBufferCipherText != nullptr && byarrBufferCipherText->size() % 16 == 0) && (!byarrKey->empty() && byarrKey->size() % 24 == 0)) {

return \_pEncryptionMode->Decryption(byarrBufferCipherText, byarrKey);

}

else {

return nullptr;

}

}

/\*End AES 192 Methods Realization\*/

/\*Start AES 256 Methods Realization\*/

vector<uint8\_t>\* AES\_256::Encrypt(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey) {

if ((!byarrBufferPlainText->empty() && byarrBufferPlainText != nullptr) && (!byarrKey->empty() && byarrKey->size() % 32 == 0)) {

return \_pEncryptionMode->Encryption(byarrBufferPlainText, byarrKey);

}

else {

return nullptr;

}

}

vector<uint8\_t>\* AES\_256::Decrypt(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey) {

if ((!byarrBufferCipherText->empty() && byarrBufferCipherText != nullptr && byarrBufferCipherText->size() % 16 == 0) && (!byarrKey->empty() && byarrKey->size() % 32 == 0)) {

return \_pEncryptionMode->Decryption(byarrBufferCipherText, byarrKey);

}

else {

return nullptr;

}

}

/\*End AES 256 Methods Realization\*/

файл EncryptionMode.h

#ifndef ENCRYPTIONMODE\_H

#define ENCRYPTIONMODE\_H

#include "Rijndael.h"

#include <thread>

using namespace std;

/\*Interface Encryption Mode Class\*/

class IEncryptionMode {

protected:

Rijndael\* \_pRijndael{ nullptr };

public:

IEncryptionMode(Rijndael\* pRijndael) : \_pRijndael(pRijndael) {};

virtual vector<uint8\_t>\* Encryption(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) = 0;

virtual vector<uint8\_t>\* Decryption(std::vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) = 0;

void AdditionBlocksRatio(vector<uint8\_t>\* arrbyBufferPublicText);

};

/\*ECB Encryption Mode Class\*/

class ECB : public IEncryptionMode {

private:

/\*Realization Multi threading\*/

void ThreadEncription(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock);

void ThreadDecryption(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock);

public:

ECB(Rijndael\* pRijndael) : IEncryptionMode(pRijndael) {};

vector<uint8\_t>\* Encryption(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decryption(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

};

/\*CTR Encryption Mode Class\*/

class CTR : public IEncryptionMode {

private:

/\*Realization Multi threading\*/

void ThreadEncription(vector<uint8\_t> IV, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock);

void ThreadDecryption(vector<uint8\_t> \_IV, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock);

public:

CTR(Rijndael\* pRijndael) : IEncryptionMode(pRijndael) {};

vector<uint8\_t>\* Encryption(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decryption(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

};

/\*OFB Encryption Mode Class\*/

class OFB : public IEncryptionMode {

public:

OFB(Rijndael\* pRijndael) : IEncryptionMode(pRijndael) {};

vector<uint8\_t>\* Encryption(vector<uint8\_t>\* PlainText, vector<uint8\_t>\* Key) override;

vector<uint8\_t>\* Decryption(vector<uint8\_t>\* CipherText, vector<uint8\_t>\* Key) override;

};

#endif //ENCRYPTIONMODE\_h

файл EncryptionMode.cpp

#include "EncryptionMode.h"

#include <random>

#include <chrono>

/\*Start InterfaceEncryptionMode Methods Realization\*/

void IEncryptionMode::AdditionBlocksRatio(vector<uint8\_t>\* arrbyBufferPublicText) {

//Work by GOST 34.13-2015

arrbyBufferPublicText->push\_back(0x80);

for (uint8\_t i = 0; i < (arrbyBufferPublicText->size() % 16); i++) { arrbyBufferPublicText->push\_back(0x00); }

}

/\*End InterfaceEncryptionMode Methods Realization\*/

/\*Start ECB Methods Realization\*/

void ECB::ThreadEncription(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock) {

vector<uint8\_t> byarrBlockCipherText;

for (uint64\_t qwCurrentBlock = qwStartBlock; qwCurrentBlock < qwEndBlock; qwCurrentBlock++) {

byarrBlockCipherText = \_pRijndael->Encrypt(vector<uint8\_t>(byarrBufferPlainText->begin() + qwCurrentBlock \* 16, byarrBufferPlainText->begin() + (qwCurrentBlock + 1) \* 16), byarrKey);

byarrBufferCipherText->insert(byarrBufferCipherText->end(), byarrBlockCipherText.begin(), byarrBlockCipherText.end());

byarrBlockCipherText.clear();

}

return;

}

vector<uint8\_t>\* ECB::Encryption(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey) {

//Check AdditionBlockRatio

if (byarrBufferPlainText->size() % 16 != 0) { AdditionBlocksRatio(byarrBufferPlainText); }

vector<uint8\_t>\* byarrBufferCipherText = new vector<uint8\_t>;

vector<thread\*> ParentArrayChildThreads;

vector<vector<uint8\_t>> ThreadsBuffers\_byarrCipherText(thread::hardware\_concurrency());

uint64\_t qwSizePlainTextBlocks = byarrBufferPlainText->size() / 16;

//Use One Main Thread Where SizePlainTextBlocks < 1024 for One CPU Thread

if (qwSizePlainTextBlocks < thread::hardware\_concurrency() \* 1024) {

ThreadEncription(byarrBufferCipherText, byarrBufferPlainText, byarrKey, 0, qwSizePlainTextBlocks);

return byarrBufferCipherText;

}

/\*Multi threading realization\*/

uint64\_t qwSizeBufferOfBlocksToOneThread = byarrBufferPlainText->size() / (16 \* thread::hardware\_concurrency());

uint64\_t qwCurrentStartBlock = 0;

uint64\_t qwCurrentEndBlock = qwCurrentStartBlock + qwSizeBufferOfBlocksToOneThread;

uint8\_t byCurrentThread = 0;

while (qwCurrentStartBlock != qwSizePlainTextBlocks) {

if ((ParentArrayChildThreads.size() == thread::hardware\_concurrency() - 1)) {

qwCurrentEndBlock = qwSizePlainTextBlocks;

ParentArrayChildThreads.push\_back(new thread([this, &byCurrentThread, &ThreadsBuffers\_byarrCipherText, &byarrBufferPlainText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadEncription(&ThreadsBuffers\_byarrCipherText[byCurrentThread], byarrBufferPlainText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

}

else {

ParentArrayChildThreads.push\_back(new thread([this, &byCurrentThread, &ThreadsBuffers\_byarrCipherText, &byarrBufferPlainText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadEncription(&ThreadsBuffers\_byarrCipherText[byCurrentThread], byarrBufferPlainText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

qwCurrentEndBlock += qwSizeBufferOfBlocksToOneThread;

}

}

for (uint8\_t i = 0; i < ParentArrayChildThreads.size(); i++) {

//Parent Wait your Child Threads For Write Final Result

ParentArrayChildThreads[i]->join();

byarrBufferCipherText->insert(byarrBufferCipherText->end(), ThreadsBuffers\_byarrCipherText[i].begin(), ThreadsBuffers\_byarrCipherText[i].end());

delete ParentArrayChildThreads[i];

ThreadsBuffers\_byarrCipherText[i].clear();

}

return byarrBufferCipherText;

}

void ECB::ThreadDecryption(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock) {

vector<uint8\_t> byarrBlockPlainText;

for (uint64\_t dwCurrentBlock = qwStartBlock; dwCurrentBlock < qwEndBlock; dwCurrentBlock++) {

byarrBlockPlainText = \_pRijndael->Decrypt(vector<uint8\_t>(byarrBufferCipherText->begin() + dwCurrentBlock \* 16, byarrBufferCipherText->begin() + (dwCurrentBlock + 1) \* 16), byarrKey);

byarrBufferPlainText->insert(byarrBufferPlainText->end(), byarrBlockPlainText.begin(), byarrBlockPlainText.end());

byarrBlockPlainText.clear();

}

return;

}

vector<uint8\_t>\* ECB::Decryption(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey) {

vector<uint8\_t>\* byarrBufferPlainText = new vector<uint8\_t>;

vector<thread\*> ParentArrayChildThreads;

vector<vector<uint8\_t>> ThreadsBuffers\_byarrPlainText(thread::hardware\_concurrency());

uint64\_t qwSizeCipherTextBlocks = byarrBufferCipherText->size() / 16;

//Use One Main Thread Where SizeCipherTextBlocks < 1024 for One CPU Thread

if (qwSizeCipherTextBlocks < thread::hardware\_concurrency() \* 1024) {

ThreadDecryption(byarrBufferPlainText, byarrBufferCipherText, byarrKey, 0, qwSizeCipherTextBlocks);

return byarrBufferPlainText;

}

/\*Multi threading realization\*/

uint64\_t qwSizeBufferOfBlocksToOneThread = byarrBufferCipherText->size() / (16 \* thread::hardware\_concurrency());

uint64\_t qwCurrentStartBlock = 0;

uint64\_t qwCurrentEndBlock = qwCurrentStartBlock + qwSizeBufferOfBlocksToOneThread;

uint8\_t byCurrentThread = 0;

while (qwCurrentStartBlock != qwSizeCipherTextBlocks) {

if ((ParentArrayChildThreads.size() == thread::hardware\_concurrency() - 1)) {

qwCurrentEndBlock = qwSizeCipherTextBlocks;

ParentArrayChildThreads.push\_back(new thread([this, &byCurrentThread, &ThreadsBuffers\_byarrPlainText, &byarrBufferCipherText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadDecryption(&ThreadsBuffers\_byarrPlainText[byCurrentThread], byarrBufferCipherText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

}

else {

ParentArrayChildThreads.push\_back(new thread([this, &byCurrentThread, &ThreadsBuffers\_byarrPlainText, &byarrBufferCipherText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadDecryption(&ThreadsBuffers\_byarrPlainText[byCurrentThread], byarrBufferCipherText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

qwCurrentEndBlock += qwSizeBufferOfBlocksToOneThread;

}

}

for (uint8\_t i = 0; i < ParentArrayChildThreads.size(); i++) {

//Parent Wait your Child Threads For Write Final Result

ParentArrayChildThreads[i]->join();

byarrBufferPlainText->insert(byarrBufferPlainText->end(), ThreadsBuffers\_byarrPlainText[i].begin(), ThreadsBuffers\_byarrPlainText[i].end());

delete ParentArrayChildThreads[i];

ThreadsBuffers\_byarrPlainText[i].clear();

}

return byarrBufferPlainText;

}

/\*End ECB Methods Realization\*/

/\*Start CTR Methods Realization\*/

void CTR::ThreadEncription(vector<uint8\_t> IV, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock){

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

for (uint8\_t i = 0; i < 16; i++) { FormatIV.byArray[i] = IV[i]; }

vector<uint8\_t> \_IV(IV);

vector<uint8\_t> byarrBlockCipherText;

for (uint64\_t qwCurrentBlock = qwStartBlock; qwCurrentBlock < qwEndBlock; qwCurrentBlock++) {

byarrBlockCipherText = \_pRijndael->Encrypt(vector<uint8\_t>(\_IV.begin(), \_IV.end()), byarrKey);

for (uint8\_t i = 0; i < 16; i++) { byarrBlockCipherText[i] ^= (\*byarrBufferPlainText)[qwCurrentBlock \* 16 + i]; }

//Add Counter += 1 (if dwCurrentBlock % 2 == 0 -> add +1 to hight 64 bits IV) (else -> add +1 to low 64 bits IV)

if (qwCurrentBlock % 2 == 0) { FormatIV.qwArray[0]++; }

else { FormatIV.qwArray[1]++; }

//Update Counter

\_IV.clear();

for (uint8\_t i : FormatIV.byArray) { \_IV.push\_back(i); }

//Add Cipher Text in Buffer

byarrBufferCipherText->insert(byarrBufferCipherText->end(), byarrBlockCipherText.begin(), byarrBlockCipherText.end());

byarrBlockCipherText.clear();

}

return;

}

vector<uint8\_t>\* CTR::Encryption(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey){

union FormattedGeneratorNumbers{

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

//Get current time in nanoseconds to mt19937\_64 Seed

auto current\_time\_now = chrono::high\_resolution\_clock::now();

mt19937\_64 urandom\_generator;

//Set Seed

urandom\_generator.seed(current\_time\_now.time\_since\_epoch().count());

//Generate IV

FormatIV.qwArray[0] = urandom\_generator();

FormatIV.qwArray[1] = urandom\_generator();

//Write IV, Where IV = Counter

vector<uint8\_t> IV;

for (uint8\_t i : FormatIV.byArray) { IV.push\_back(i); }

//Output Buffer

vector<uint8\_t>\* byarrBufferCipherText = new vector<uint8\_t>;

//Insert IV in BufferCipherText

byarrBufferCipherText->insert(byarrBufferCipherText->end(), IV.begin(), IV.end());

//Check AdditionBlockRatio

if (byarrBufferPlainText->size() % 16 != 0) { AdditionBlocksRatio(byarrBufferPlainText); }

vector<thread\*> ParentArrayChildThreads;

vector<vector<uint8\_t>> ThreadsBuffers\_byarrCipherText(thread::hardware\_concurrency());

uint64\_t qwSizePlainTextBlocks = byarrBufferPlainText->size() / 16;

//Use One Main Thread Where SizePlainTextBlocks < 1024 for One CPU Thread

if (qwSizePlainTextBlocks < thread::hardware\_concurrency() \* 1024) {

ThreadEncription(IV, byarrBufferCipherText, byarrBufferPlainText, byarrKey, 0, qwSizePlainTextBlocks);

return byarrBufferCipherText;

}

/\*Multi threading realization\*/

uint64\_t qwSizeBufferOfBlocksToOneThread = byarrBufferPlainText->size() / (16 \* thread::hardware\_concurrency());

uint64\_t qwCurrentStartBlock = 0;

uint64\_t qwCurrentEndBlock = qwCurrentStartBlock + qwSizeBufferOfBlocksToOneThread;

uint8\_t byCurrentThread = 0;

while (qwCurrentStartBlock != qwSizePlainTextBlocks) {

if ((ParentArrayChildThreads.size() == thread::hardware\_concurrency() - 1)) {

qwCurrentEndBlock = qwSizePlainTextBlocks;

ParentArrayChildThreads.push\_back(new thread([this, &IV, &byCurrentThread, &ThreadsBuffers\_byarrCipherText, &byarrBufferPlainText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadEncription(IV, &ThreadsBuffers\_byarrCipherText[byCurrentThread], byarrBufferPlainText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

}

else {

ParentArrayChildThreads.push\_back(new thread([this, &IV, &byCurrentThread, &ThreadsBuffers\_byarrCipherText, &byarrBufferPlainText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadEncription(IV, &ThreadsBuffers\_byarrCipherText[byCurrentThread], byarrBufferPlainText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

}

//Update Counter For Next Thread

for (auto qwCurrentBlock = qwCurrentStartBlock; qwCurrentBlock < qwCurrentEndBlock; qwCurrentBlock++) {

//Add Counter += 1 (if dwCurrentBlock % 2 == 0 -> add +1 to hight 64 bits IV) (else -> add +1 to low 64 bits IV)

if (qwCurrentBlock % 2 == 0) { FormatIV.qwArray[0]++; }

else { FormatIV.qwArray[1]++; }

}

//Update qwCurrentStartBlock And qwCurrentEndBlock

qwCurrentStartBlock = qwCurrentEndBlock;

qwCurrentEndBlock += qwSizeBufferOfBlocksToOneThread;

//Update Counter

IV.clear();

for (uint8\_t i : FormatIV.byArray) { IV.push\_back(i); }

}

for (uint8\_t i = 0; i < ParentArrayChildThreads.size(); i++) {

//Parent Wait your Child Threads For Write Final Result

ParentArrayChildThreads[i]->join();

byarrBufferCipherText->insert(byarrBufferCipherText->end(), ThreadsBuffers\_byarrCipherText[i].begin(), ThreadsBuffers\_byarrCipherText[i].end());

delete ParentArrayChildThreads[i];

ThreadsBuffers\_byarrCipherText[i].clear();

}

return byarrBufferCipherText;

}

void CTR::ThreadDecryption(vector<uint8\_t> IV, vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey, uint64\_t qwStartBlock, uint64\_t qwEndBlock) {

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

for (uint8\_t i = 0; i < 16; i++) { FormatIV.byArray[i] = IV[i]; }

vector<uint8\_t> \_IV(IV);

vector<uint8\_t> byarrBlockCipherText;

for (uint64\_t qwCurrentBlock = qwStartBlock; qwCurrentBlock < qwEndBlock; qwCurrentBlock++) {

byarrBlockCipherText = \_pRijndael->Encrypt(vector<uint8\_t>(\_IV.begin(), \_IV.end()), byarrKey);

for (uint8\_t i = 0; i < 16; i++) { byarrBlockCipherText[i] ^= (\*byarrBufferCipherText)[qwCurrentBlock \* 16 + i]; }

//Revert add Counter +=1 because qwCurrentStartBlock Start with 1;

//Add Counter += 1 (if dwCurrentBlock % 2 == 1 -> add +1 to hight 64 bits IV) (else -> add +1 to low 64 bits IV)

if (qwCurrentBlock % 2 == 1) { FormatIV.qwArray[0]++; }

else { FormatIV.qwArray[1]++; }

//Update Counter

\_IV.clear();

for (uint8\_t i : FormatIV.byArray) { \_IV.push\_back(i); }

//Add Cipher Text in Buffer

byarrBufferPlainText->insert(byarrBufferPlainText->end(), byarrBlockCipherText.begin(), byarrBlockCipherText.end());

byarrBlockCipherText.clear();

}

return;

}

vector<uint8\_t>\* CTR::Decryption(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey){

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

//Write IV, Where IV = Counter

vector<uint8\_t> IV(byarrBufferCipherText->begin(), byarrBufferCipherText->begin() + 16);

for (uint8\_t i = 0; i < 16; i++) { FormatIV.byArray[i] = IV[i]; }

//Output Buffer

vector<uint8\_t>\* byarrBufferPlainText = new vector<uint8\_t>;

vector<thread\*> ParentArrayChildThreads;

vector<vector<uint8\_t>> ThreadsBuffers\_byarrPlainText(thread::hardware\_concurrency());

uint64\_t qwSizeCipherTextBlocks = byarrBufferCipherText->size() / 16;

//Use One Main Thread Where SizePlainTextBlocks < 1024 for One CPU Thread

if (qwSizeCipherTextBlocks < thread::hardware\_concurrency() \* 1024) {

ThreadDecryption(IV, byarrBufferPlainText, byarrBufferCipherText, byarrKey, 1, qwSizeCipherTextBlocks);

return byarrBufferPlainText;

}

/\*Multi threading realization\*/

uint64\_t qwSizeBufferOfBlocksToOneThread = byarrBufferCipherText->size() / (16 \* thread::hardware\_concurrency());

uint64\_t qwCurrentStartBlock = 1;

uint64\_t qwCurrentEndBlock = qwCurrentStartBlock + qwSizeBufferOfBlocksToOneThread;

uint8\_t byCurrentThread = 0;

while (qwCurrentStartBlock != qwSizeCipherTextBlocks) {

if ((ParentArrayChildThreads.size() == thread::hardware\_concurrency() - 1)) {

qwCurrentEndBlock = qwSizeCipherTextBlocks;

ParentArrayChildThreads.push\_back(new thread([this, &IV, &byCurrentThread, &ThreadsBuffers\_byarrPlainText, &byarrBufferCipherText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadDecryption(IV, &ThreadsBuffers\_byarrPlainText[byCurrentThread], byarrBufferCipherText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

qwCurrentStartBlock = qwCurrentEndBlock;

}

else {

ParentArrayChildThreads.push\_back(new thread([this, &IV, &byCurrentThread, &ThreadsBuffers\_byarrPlainText, &byarrBufferCipherText, &byarrKey, qwCurrentStartBlock, qwCurrentEndBlock]() { this->ThreadDecryption(IV, &ThreadsBuffers\_byarrPlainText[byCurrentThread], byarrBufferCipherText, byarrKey, qwCurrentStartBlock, qwCurrentEndBlock); }));

byCurrentThread++;

}

//Update Counter For Next Thread

for (auto qwCurrentBlock = qwCurrentStartBlock; qwCurrentBlock < qwCurrentEndBlock; qwCurrentBlock++) {

//Revert add Counter +=1 because dwCurrentBlock Start with 1;

//Add Counter += 1 (if dwCurrentBlock % 2 == 1 -> add +1 to hight 64 bits IV) (else -> add +1 to low 64 bits IV)

if (qwCurrentBlock % 2 == 1) { FormatIV.qwArray[0]++; }

else { FormatIV.qwArray[1]++; }

}

//Update qwCurrentStartBlock And qwCurrentEndBlock

qwCurrentStartBlock = qwCurrentEndBlock;

qwCurrentEndBlock += qwSizeBufferOfBlocksToOneThread;

//Update Counter

IV.clear();

for (uint8\_t i : FormatIV.byArray) { IV.push\_back(i); }

}

for (uint8\_t i = 0; i < ParentArrayChildThreads.size(); i++) {

//Parent Wait your Child Threads For Write Final Result

ParentArrayChildThreads[i]->join();

byarrBufferPlainText->insert(byarrBufferPlainText->end(), ThreadsBuffers\_byarrPlainText[i].begin(), ThreadsBuffers\_byarrPlainText[i].end());

delete ParentArrayChildThreads[i];

ThreadsBuffers\_byarrPlainText[i].clear();

}

return byarrBufferPlainText;

}

/\*End CTR Methods Realization\*/

/\*Start OFB Methods Realization\*/

vector<uint8\_t>\* OFB::Encryption(vector<uint8\_t>\* byarrBufferPlainText, vector<uint8\_t>\* byarrKey) {

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

//Get current time in nanoseconds to mt19937\_64 Seed

auto current\_time\_now = chrono::high\_resolution\_clock::now();

mt19937\_64 urandom\_generator;

//Set Seed

urandom\_generator.seed(current\_time\_now.time\_since\_epoch().count());

//Generate IV

FormatIV.qwArray[0] = urandom\_generator();

FormatIV.qwArray[1] = urandom\_generator();

//Write IV, Where IV = Counter

vector<uint8\_t> IV;

for (uint8\_t i : FormatIV.byArray) { IV.push\_back(i); }

//Output Buffer

vector<uint8\_t>\* arrbyBufferCipherText = new vector<uint8\_t>;

//Insert IV in BufferCipherText

arrbyBufferCipherText->insert(arrbyBufferCipherText->end(), IV.begin(), IV.end());

//Check AdditionBlockRatio

if (byarrBufferPlainText->size() % 16 != 0) { AdditionBlocksRatio(byarrBufferPlainText); }

vector<uint8\_t> BlockCipherText;

for (uint32\_t dwCurrentBlock = 0; dwCurrentBlock < (byarrBufferPlainText->size() / 16); dwCurrentBlock++) {

IV = \_pRijndael->Encrypt(vector<uint8\_t>(IV.begin(), IV.end()), byarrKey);

for (uint8\_t i = 0; i < 16; i++) { BlockCipherText.push\_back(IV[i] ^ (\*byarrBufferPlainText)[dwCurrentBlock \* 16 + i]); }

//Add Cipher Text in Buffer

arrbyBufferCipherText->insert(arrbyBufferCipherText->end(), BlockCipherText.begin(), BlockCipherText.end());

BlockCipherText.clear();

}

return arrbyBufferCipherText;

}

vector<uint8\_t>\* OFB::Decryption(vector<uint8\_t>\* byarrBufferCipherText, vector<uint8\_t>\* byarrKey) {

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatIV;

//Write IV, Where IV = Counter

vector<uint8\_t> IV(byarrBufferCipherText->begin(), byarrBufferCipherText->begin() + 16);

for (uint8\_t i = 0; i < 16; i++) { FormatIV.byArray[i] = IV[i]; }

//Output Buffer

vector<uint8\_t>\* arrbyBufferPlainText = new vector<uint8\_t>;

vector<uint8\_t> BlockPlainText;

for (uint32\_t dwCurrentBlock = 1; dwCurrentBlock < (byarrBufferCipherText->size() / 16); dwCurrentBlock++) {

IV = \_pRijndael->Encrypt(vector<uint8\_t>(IV.begin(), IV.end()), byarrKey);

for (uint8\_t i = 0; i < 16; i++) { BlockPlainText.push\_back(IV[i] ^ (\*byarrBufferCipherText)[dwCurrentBlock \* 16 + i]); }

//Add Cipher Text in Buffer

arrbyBufferPlainText->insert(arrbyBufferPlainText->end(), BlockPlainText.begin(), BlockPlainText.end());

BlockPlainText.clear();

}

return arrbyBufferPlainText;

}

/\*End OFB Methods Realization\*/

# *Реализация хеш-функции. Хеш-функция SHA-512*

**Лабораторная работа №2**

Для выполнения лабораторной работы по реализации криптографического протокола TLS необходима реализация криптографических примитивов, которые используются при построении протокола. Одним из таковых является хеш-функция, которая позволяет вычислить «цифровой слепок» (хеш-значение) сообщения. В работе будет реализована быстрая для современных х64 ЦП хеш-функция семейства SHA2 под названием SHA-512.

# Описание

Рассмотрим стандарт FIPS (Федеральный стандарт обработки информации) PUB 180-4, объединяющий все семейство хеш-функций SHA1 и SHA2. Остановимся на хеш-функциях семейства SHA2.

Хеш-функции  SHA-2  разработаны  Агентством национальной безопасности США и опубликованы Национальным институтом стандартов и технологий в федеральном стандарте обработки информации FIPS PUB 180-2 в августе 2002 года. В этот стандарт также вошла хеш-функция SHA-1, разработанная в 1995 году. В феврале 2004 года в FIPS PUB 180-2 была добавлена SHA-224. В октябре 2008 года вышла новая редакция стандарта – FIPS PUB 180-3. В августе 2015 года вышла последняя на данный момент редакция FIPS PUB 180-4, в которой были добавлены функции SHA-512/256 и SHA-512/224, основанные на SHA-512 (поскольку на 64-битных архитектурах SHA-512 работает быстрее, чем SHA-256). Длина хеш-значения сообщения алгоритма SHA-512 равна 512 бит.

Хеш-алгоритмы, указанные в этом стандарте FIPS PUB 180-4 называются безопасными потому, что по заданному алгоритму невозможно вычислить следующее:

1) восстановить сообщение по конкретному хеш-значению сообщения;

2) найти два различных сообщения, у которых одно и тот же хеш-значение сообщения (найти коллизию). Любые изменения в сообщении, с очень высокой вероятностью, приводят к различным хеш-значениям.

# Основные операции

Помимо основных (базовых) операций, используемых в ЭВМ: *XOR*, *AND*, *OR*, *NE *, сложение по модулю *264*, вводятся операции правого поворота (*ROTR*) и правого сдвига (которое также присутствует в ЭВМ) (*SHR*).

*ROTRn(x)* – поворот вправо (циклический правый сдвиг) операция, где *x* это 64 битное слово и *n* целое число, которое *0 ≤ n < 64*, операция математически определена как: *ROTR n (x)=(x >> n) ∨ (x << 64 - n)*.

*SHR n (x)* – операция правого сдвига, где *x* это 64 битное слово и *n* целое число, которое *0 ≤ n < 64*, операция математически определена как: *SHR n (x)=x >> n*.

# Функции и константы

## Функции













## Константы

SHA-512 использует последовательности из 80 констант по 64 битных слов  .



# Подготовка к вычислению хеш-значения

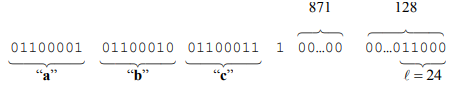
## Дополнение сообщения

Предположим, что длина сообщения *M*, измеренное в битах, равно *λ* бит.

Добавим в конец бит равный «*1*», за ним последующие *k* нулевых битов, где *k* есть маленькое неотрицательное решение сравнения *λ +1+ k ≡ 896(mod 1024)* .

Затем после всех предыдущих операций добавим в конец *128* битный блок, который есть двоичное представление длины сообщения *λ*.

Например, сообщение составленное из байт «*abc*» (*1* байт равен *8* бит *ASCII*) в битах имеет длину *8×3 = 24* бит, добавим к сообщению бит «*1*», затем добавим *896 − (24 +1) = 871* нулевых битов, и только после этого добавим *128* битное представление длины сообщения. На выходе получаем *1024* битное дополненное сообщение.



Длина дополненного сообщения составляет *1024* бита.

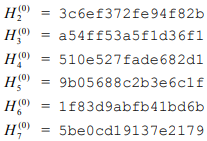
## Получение сообщения

Для *SHA-512* сообщение и его дополнение представляется как *N 1024* битных блоков, . Так, полученный 1024 битный блок может быть представлен как 16 64 битных слов, первый 64 битный блок сообщения *i* обозначается , следующий блок в 64 бита обозначается как ,и так далее до.

## Настройка инициализации начальных хеш-значений

Для *SHA-512* инициализация начальных хеш-значений  представляет собой восемь *64* битных слов в шестнадцатеричной системе счисления:





# Хеш-функция SHA-512

*SHA-512* может быть использован для вычисления хеш-значения сообщения *M*, имеющего длину *λ* бит, где *0 ≤ λ < 2128* .

Алгоритм использует:

1) Схему представления блока сообщения как восемь *64* битных слов;

2) Восемь инициализационных хеш-значений по *64* бита каждое;

3) Хеш-значение сообщения представляет собой восемь *64* битных слов. Конечный результат работы алгоритма *SHA-512* – это *512* битное хеш-значение сообщения.

Слова в схеме сообщения маркируются как .

Восемь рабочих переменных обозначаются как *a*, *b*, *c*, *d*, *e*, *f*, *g*, и *h*.

Слова хеш-значений маркируются как  , которые начинаются с начальных хеш-значений , затем итеративно вычисляются их промежуточные хеш-значения обозначаемые как , закачиваются они хеш-значениями .

*SHA512* также использует две временные переменные T1 и T2.

## Подготовка к алгоритму SHA-512

1. Установка начальных хеш-значений , описанных в секции 4.3

2. Принимаемое сообщение дополняется и преобразуется по правилам в секциях 4.1 и 4.2.

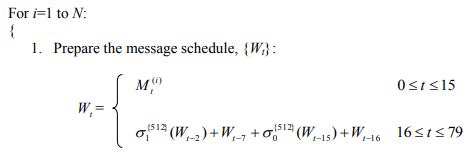
## Вычисление хеш-значения сообщения по алгоритму SHA-512

При вычислении алгоритмом *SHA-512* хеш-значения сообщения используются функции и константы, определенные в разделах 3.1 и 3.2. Сложение «*+*» производится по модулю *264* .

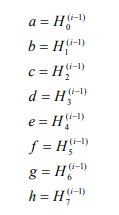
Каждый блок сообщения  обрабатываются в порядке, используя следующие шаги:

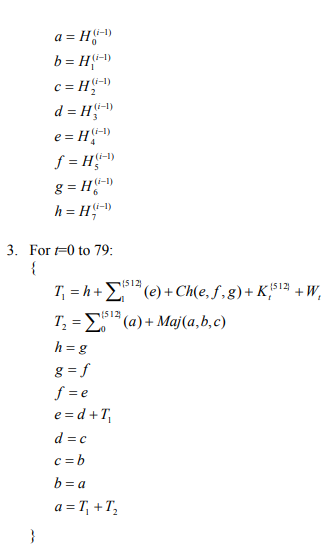


1. Подготовка схемы сообщения , 

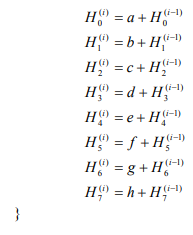


2. Инициализация восьми рабочих переменных: *a*, *b*, *c*, *d*, *e*, *f*, *g*, и *h*, а также  вычисленными хеш-значениями:





4. Вычисление следующих хеш-значений :



После повторения данных этапов один за другим, на протяжении *N* шагов, итоговое *512* битовое хеш-значение сообщения *M* будет иметь вид:

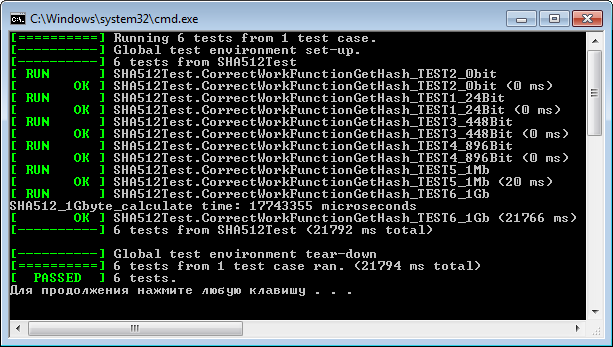


# Результаты реализации алгоритма SHA-512

Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация алгоритма SHA-512 входит в проект SHA512\_Hash решения CryptoProtocols.

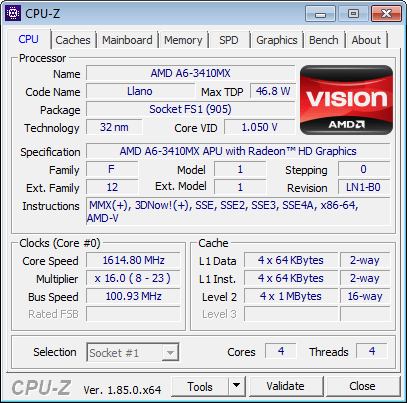
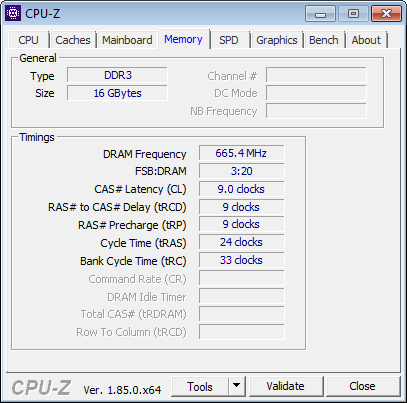
Для тестирования корректности разрабатываемых проектов в решении CryptoProtocols был создан отдельный проект GoogleTestingSolutionProject модульного тестирования gtest (для unit testing) и gmock (для проверки корректности вызовов методов). Данные пакеты устанавливались через менеджер пакетов NuGet для Visual Studio.

Результат выполнения тест кейсов (значения взяты из [[ссылка](http://www.di-mgt.com.au/sha_testvectors.html)]) для проверки корректности работы функции хеширования SHA-512 и фиксации времени выполнения для подсчета производительности работы (т.к. gtest замеряет работу вызовов кейсов в микросекундах, то для повышения точности была использована библиотека <chrono> c++11 с точностью до микросекунд) приведены на Рис. 1.



**Рис. 1.** Результат тестирования реализованного алгоритма SHA-512

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.2. По полученным данным посчитаем скорость хеширования для данного ЦП. Данные приведены в Табл. 1.

**Рис. 2.** ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

**Табл. 1.** Скорость выполнения хеширования алгоритма SHA-512

|  |  |  |
| --- | --- | --- |
| Алгоритм и размер хеш-значения | Размер данных [Мбайт] | Скорость [Мбайт/с] |
| SHA-512/512 бит | 1024 | 57,711746172 |

# Литература

1. FIPS PUB 180-4 «Secure Hash Standard (SHS)» [Интернет ресурс], ссылка <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>
2. Test vectors for SHA-1, SHA-2 and SHA-3 [Интернет ресурс], ссылка <https://www.di-mgt.com.au/sha_testvectors.html>

# Листинг кода

файл SHA512.h

#ifndef SHA512\_H

#define SHA512\_H

/\*

FIPS PUB 180-4

FEDERAL INFORMATION PROCESSING STANDARDS

PUBLICATION

Secure Hash Standard (SHS)

CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPTOGRAPHY

RUSSIAN TECHNOLOGICAL UNIVERSITY [RTU MIREA]

REALIZATION SHA512 HASH FUNCTION

\*/

#include <iostream>

#include <cstring>

#include <vector>

#include <memory>

using namespace std;

namespace AlgorithmSHA512 {

class SHA512 {

private:

uint64\_t H0{ 0 };

uint64\_t H1{ 0 };

uint64\_t H2{ 0 };

uint64\_t H3{ 0 };

uint64\_t H4{ 0 };

uint64\_t H5{ 0 };

uint64\_t H6{ 0 };

uint64\_t H7{ 0 };

vector<uint8\_t>\* byarrMessage{ nullptr };

unique\_ptr<vector<uint64\_t>> W{ nullptr };

unique\_ptr<vector<uint64\_t>> M{ nullptr };

void PaddingTheMessage();

void Preprocessing();

void HashCompulation();

void HashComplulationBlock();

uint64\_t ROTR(uint64\_t x, uint8\_t n);

uint64\_t SHR(uint64\_t x, uint8\_t n);

uint64\_t SIGMA0(uint64\_t x);

uint64\_t SIGMA1(uint64\_t x);

uint64\_t sigma0(uint64\_t x);

uint64\_t sigma1(uint64\_t x);

public:

vector<uint8\_t>\* GetHash(vector<uint8\_t>\* ptrMessage);

};

};

#endif //SHA512\_H

файл SHA512.cpp

#include "SHA512.h"

using namespace AlgorithmSHA512;

#define Ch(x, y, z) (((x) & (y)) ^ ((~x) & (z)))

#define Maj(x, y, z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))

uint64\_t K[80]{

0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc,

0x3956c25bf348b538, 0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118,

0xd807aa98a3030242, 0x12835b0145706fbe, 0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,

0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235, 0xc19bf174cf692694,

0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,

0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5,

0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4,

0xc6e00bf33da88fc2, 0xd5a79147930aa725, 0x06ca6351e003826f, 0x142929670a0e6e70,

0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed, 0x53380d139d95b3df,

0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,

0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30,

0xd192e819d6ef5218, 0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,

0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8,

0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373, 0x682e6ff3d6b2b8a3,

0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,

0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b,

0xca273eceea26619c, 0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178,

0x06f067aa72176fba, 0x0a637dc5a2c898a6, 0x113f9804bef90dae, 0x1b710b35131c471b,

0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc, 0x431d67c49c100d4c,

0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817

};

uint64\_t SHA512::ROTR(uint64\_t x, uint8\_t n) { return ((x >> n) | (x << (64 - n))); };

uint64\_t SHA512::SHR(uint64\_t x, uint8\_t n) { return (x >> n); };

uint64\_t SHA512::SIGMA0(uint64\_t x) { return (ROTR(x, 28) ^ ROTR(x, 34) ^ ROTR(x, 39)); };

uint64\_t SHA512::SIGMA1(uint64\_t x) { return (ROTR(x, 14) ^ ROTR(x, 18) ^ ROTR(x, 41)); };

uint64\_t SHA512::sigma0(uint64\_t x) { return (ROTR(x, 1) ^ ROTR(x, 8) ^ SHR(x, 7)); };

uint64\_t SHA512::sigma1(uint64\_t x) { return (ROTR(x, 19) ^ ROTR(x, 61) ^ SHR(x, 6)); };

void SHA512::PaddingTheMessage() {

union Transformation {

uint8\_t BytesOfNumber[8];

uint64\_t qwNumber;

};

Transformation transformationNumbers;

uint64\_t qwInputMessageSize = byarrMessage->size();

byarrMessage->push\_back(0x80);

/\*Add Bytes 0x00 in Massage to resolve equation : [InputMessageSize + 1 + NumberOfBytes0x00 = 112(mod 128)] <=> [l + 1 + k = 896(mod 1024)]\*/

while (byarrMessage->size() % 128 != 112) { byarrMessage->push\_back(0x00); }

/\*Add 8 bytes (64 bits) 0x00 because the length of input message is less 2^64 bits\*/

for (uint8\_t i = 0; i < 8; i++){ byarrMessage->push\_back(0x00); }

/\*Add Block 64 bit Number of Bits InputMessageSize\*/

transformationNumbers.qwNumber = 8 \* qwInputMessageSize;

for (int i = 7; i > -1; i--) { byarrMessage->push\_back(transformationNumbers.BytesOfNumber[i]); }

};

void SHA512::Preprocessing() {

PaddingTheMessage();

};

void SHA512::HashCompulation() {

H0 = 0x6a09e667f3bcc908;

H1 = 0xbb67ae8584caa73b;

H2 = 0x3c6ef372fe94f82b;

H3 = 0xa54ff53a5f1d36f1;

H4 = 0x510e527fade682d1;

H5 = 0x9b05688c2b3e6c1f;

H6 = 0x1f83d9abfb41bd6b;

H7 = 0x5be0cd19137e2179;

union FormatedMessageToAlgorithm {

uint64\_t M;

uint8\_t ByteArrayOfM[8];

};

FormatedMessageToAlgorithm formatMessage;

for (register uint64\_t i = 0; i < byarrMessage->size(); ) {

while (M->size() != 16) {

for (int j = 7; j > -1; j--) {

#pragma warning (disable: 4244)

formatMessage.ByteArrayOfM[j] = (\*byarrMessage)[i];

#pragma warning (default: 4244)

i++;

}

M->push\_back(formatMessage.M);

}

HashComplulationBlock();

M->clear();

}

};

void SHA512::HashComplulationBlock() {

for (uint8\_t t = 0; t < 16; t++) { W->push\_back((\*M)[t]); }

for (uint8\_t t = 16; t < 80; t++) { W->push\_back(sigma1((\*W)[t - 2]) + (\*W)[t - 7] + sigma0((\*W)[t - 15]) + (\*W)[t - 16]); }

uint64\_t a, b, c, d, e, f, g, h , T1, T2;

a = H0;

b = H1;

c = H2;

d = H3;

e = H4;

f = H5;

g = H6;

h = H7;

for (uint8\_t t = 0; t < 80; t++) {

T1 = h + SIGMA1(e) + Ch(e, f, g) + K[t] + (\*W)[t];

T2 = SIGMA0(a) + Maj(a, b, c);

h = g;

g = f;

f = e;

e = d + T1;

d = c;

c = b;

b = a;

a = T1 + T2;

}

H0 += a;

H1 += b;

H2 += c;

H3 += d;

H4 += e;

H5 += f;

H6 += g;

H7 += h;

W->clear();

};

vector<uint8\_t>\* SHA512::GetHash(vector<uint8\_t>\* ptrMessage) {

byarrMessage = ptrMessage;

M = make\_unique<vector<uint64\_t>>();

W = make\_unique<vector<uint64\_t>>();

Preprocessing();

HashCompulation();

union DigestFormat {

uint64\_t qwH;

uint8\_t byArray[8];

};

DigestFormat formatDigest;

auto Digest = new vector<uint8\_t>;

formatDigest.qwH = H0;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H1;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H2;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H3;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H4;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H5;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H6;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

formatDigest.qwH = H7;

for (int i = 7; i > -1; i--) { Digest->push\_back(formatDigest.byArray[i]); }

H0 = H1 = H2 = H3 = H4 = H5 = H6 = H7 = 0;

return Digest;

};

# *Реализация цифровой подписи. Алгоритм ЦП ECDSA*

**Лабораторная работа №3**

Для выполнения лабораторной работы по реализации криптографического протокола TLS необходима реализация криптографических примитивов, которые используются при построении протокола. Одним из таковых является цифровая подпись.

Согласно Федеральному закону от 06.04.2011 №63-ФЗ «Об электронной подписи» под электронной подписью понимают - информацию в электронной форме, которая присоединена к другой информации в электронной форме (подписываемой информации) или иным образом связана с такой информацией и которая используется для определения лица, подписывающего информацию.

В данной лабораторной работе будет реализована цифровая подпись на основе алгоритма ECDSA над простым полем (Prime Field).

# Описание

Стойкость алгоритма цифровой подписи (далее - ЦП) основывается на проблеме дискретного логарифма в группе точек эллиптической кривой. В отличие от проблемы простого дискретного логарифма и проблемы факторизации целого числа, не существует субэкспоненциального алгоритма для проблемы дискретного логарифма в группе точек эллиптической кривой. По этой причине «сила на один бит ключа» существенно выше в алгоритме, который использует эллиптические кривые.

Алгоритм ECDSA в 1999 г. был принят как стандарт ANSI, в 2000 г. — как стандарт IEEE и NIST. Также в 1998 г. алгоритм был принят стандартом ISO. Несмотря на то, что стандарты ЦП созданы совсем недавно и находятся на этапе совершенствования, одним из наиболее перспективных из них на сегодняшний день является ANSI X9.62 ECDSA от 1999 — DSA для эллиптических кривых. На данный момент базовым американским стандартом, описывающим ECDSA, является стандарт от июня 2013 года NIST FIPS PUB 186-4 «Digital Signature Standard».

Д. Брауном (Daniel R. L. Brown) было доказано, что алгоритм ECDSA не является более безопасным, чем DSA. Им было сформулировано ограничение безопасности для ECDSA, которое привело к следующему заключению:

«Если группа эллиптической кривой может быть смоделирована основной группой и её хеш-функция удовлетворяет определенному обоснованному предположению, то ECDSA устойчива к атаке на основе подобранного открытого текста с существующей фальсификацией».

В Российской Федерации с 2001 года существует стандарт, описывающий процессы формирования и проверки ЦП, его последней редакцией является ГОСТ 34.10-2012 «Процессы формирования и проверки электронной цифровой подписи».

# Рекомендации NIST по выбору эллиптических кривых

NIST рекомендует выбирать эллиптические кривые трех видов:

- Псевдослучайная кривая над полем *GF(p)*, где *p* – простое;

- Псевдослучайная кривая над полем *GF(2m)*;

- Псевдослучайная кривая над полем *GF(2m)*, названные кривыми *Koblitz* или аномальные двоичные кривые.

Каждая эллиптическая кривая имеет базовую точку порядка *n*, где *n* – порядок подгруппы группы точек эллиптической кривой. Такая точка в стандарте NIST называется базовой точкой. Каждая кривая имеет свою базовую точку .

В реализации ЦП в данной работе будет использоваться реализация арифметики для эллиптической кривой над полем *GF(p)*.

# Эллиптические кривые над простыми полями *GF(p)*

Для каждого простого *p* существует псевдослучайная кривая



простого порядка *n*. Различные виды рекомендованных псевдослучайных кривых приведены в стандарте NIST, где для всех кривых параметр *a* (коэффициент при *x*) равен «-3».

Каждая кривая описывается параметрами:

- простым модулем *р*;

- порядком подгруппы точек эллиптической кривой *n*;

- коэффициентом *b*, таким, что:



- координатой *x* базовой точки ;

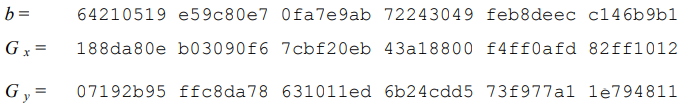
- координатой *y* базовой точки ;

Параметр *n* обладает следующим свойством: *nG = O* (нулевая точка). Описание операций над точками эллиптической кривой приведены в следующем разделе.

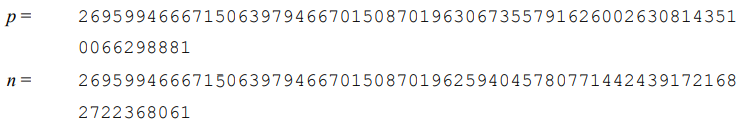
Параметры *p* и *n* представлены в стандарте в десятичной форме, остальные представляются в шестнадцатеричной системе счисления.

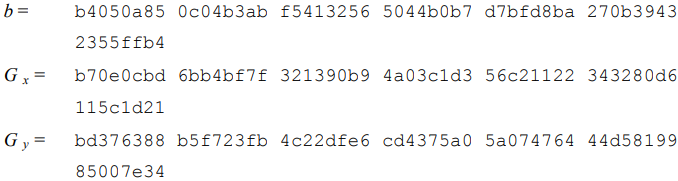
*Кривая NIST P-192*



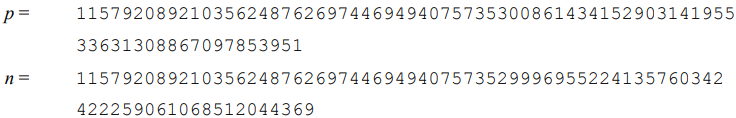


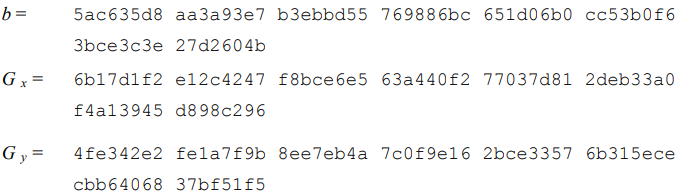
*Кривая NIST Р-224*



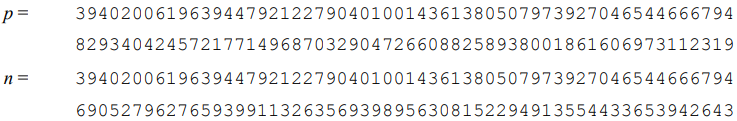


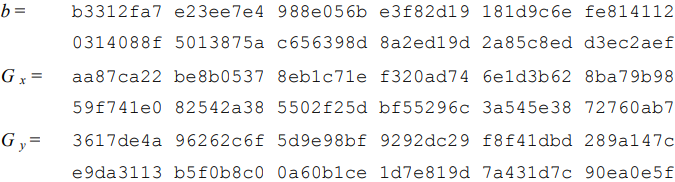
*Кривая NIST Р-256*



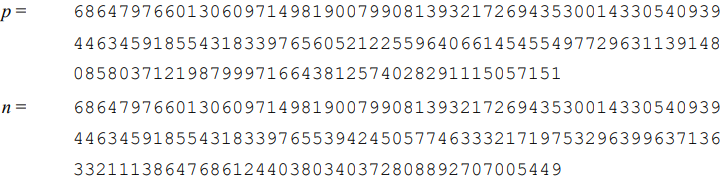


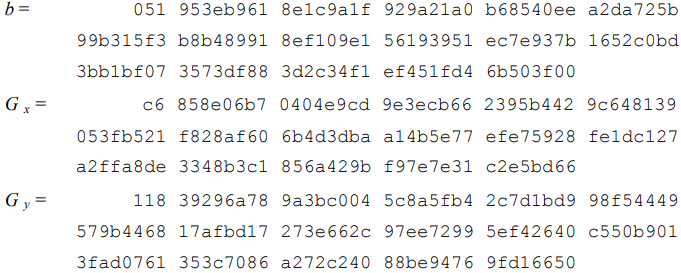
*Кривая NIST Р-384*





*Кривая NIST Р-521*

**



В стандарте ГОСТ Р 34.10-2012 приведены два типа эллиптических кривых Р-256 и Р-512, которые задаются следующими параметрами, представленными в десятичной системе счисления:

*Кривая ГОСТ Р-256*

*a =* 7

*p =* 57896044618658097711785492504343953926634992332820282019728792003956564821041

*n* = 57896044618658097711785492504343953927082934583725450622380973592137631069619

*b* = 43308876546767276905765904595650931995942111794451039583252968842033849580414

*Gx* = 2

*Gy* = 4018974056539037503335449422937059775635739389905545080690979365213431566280

*Кривая ГОСТ Р-512*

*a =* 7

*p =* 36239861022290036359077887536838743060213209255346786050865

46150450856166624002482588482022271496854025090823603058735

1637342638 22371964987228582907372403

*n* = 36239861022290036359077887536838743060213209255346786050865

46150450856166623969164898305032863068499961404079437936585

455865192212970734808812618120619743

*b* = 15186550692108285345089500347140431549287475277402064361940

18823352809982443793732829756914785974674866041605397883677

596626326413990136959047435811826396

*Gx* = 1928356944067022849399309401243137598997786635459507974357

0754913077665926858354410655576810031848748196580049032123

32884252335830250729527632383493573274

*Gy* = 22887286933719728599700121555294784163535623273295061803  
 14497425931102860301572814141997072271708807066593850650

334152381857347798885864807605098724013854

# Математические операции над эллиптическими кривыми

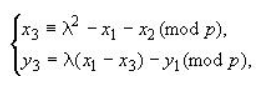
Парой (*x*, *y*), где *x* и *y* – элементы поля *GF(p)* и удовлетворяющие уравнению эллиптической кривой *E* называются точками эллиптической кривой *E*, а *x* и *y* координатами этой точки.

Точка эллиптической кривой обозначается как *C*(*x, y*) или просто *C*.

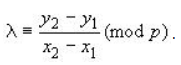
Две точки эллиптической кривой *С1*(*x1, y1*) и *С2*(*x2, y2*) равны, если равны их соответствующие координаты (*С1* = *С2*, если *x1* = *x2* и *y1* = *y2*).

На множестве точек эллиптической кривой *E* операцию сложения обозначают знаком «+». Для двух произвольных точек *С1*(*x1, y1*) и *С2*(*x2, y2*) эллиптической кривой *Е* рассматривают несколько случаев:

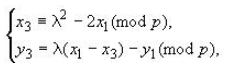
1. Для точек *С1*(*x1, y1*) и *С2*(*x2, y2*), координаты которых удовлетворяют условию *x1* ≠ *x2*, их суммой называется точка *С3*(*x3, y3*), координаты которой определяются сравнениями:



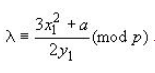
Где:



1. Для точек *С1*(*x1, y1*) и *С2*(*x2, y2*), координаты которых удовлетворяют условию *x1* = *x2* и *y1* = *y2* ≠ 0, их суммой называется точка *С3*(*x3, y3*), координаты которой определяются сравнениями:



Где:



1. Для точек *С1*(*x1, y1*) и *С2*(*x2, y2*), координаты которых удовлетворяют условию *x1* = *x2* и *y1* = *y2*(*mod p*), их суммой называется точка *С3*(*x3, y3*) = *O* – нулевой точкой без определения её *х* и *у* координат. В этом случае точка *С2* называется отрицанием точки *С1*. Для нулевой точки *O* выполнены равенства:

*С* + *О* = *О + С = С*,

Где:

*С* – произвольная точка эллиптической кривой *E*.

Относительно введенной операции сложения множество точек эллиптической кривой *E* вместе с нулевой точкой образуют конечную абелевую (коммутативную) группу порядка *m*, для которого выполнено неравенство:



Точка *C* называется точкой кратности *k* или просто кратной точкой эллиптической кривой *E*, если для некоторой точки *Р* выполнено равенство:



# Параметры пользователя

Каждый пользователь схемы ЦП должен обладать личными параметрами:

- ключом подписи – целым числом *d*, удовлетворяющим неравенству:



- ключом проверки подписи – точкой эллиптической кривой *Q* с координатами (*xQ*, *yQ*) удовлетворяющая равенству:



# Формирование цифровой подписи

Для получения цифровой подписи под сообщением *М* необходимо выполнить следующие шаги:

1. Вычислить хеш-значение сообщения *М*:



1. Вычислить целое число *e*:



Если *e* = 0, то определить *e* = 1.

1. Получить случайное (псевдослучайное) целое число *k*, удовлетворяющее неравенству:



1. Вычислить точку эллиптической кривой *С = kG* и определить:

,

Где: *xC*– *x* координата точки *С*.

Если *r* = 0, то вернуться к шагу 3).

1. Вычислить значение:



Если *s* = 0, то вернуться к шагу 3).

1. Определить цифровую подпись: как два выходных параметра *r* и *s*.

Исходными данными данного процесса являются ключ подписи *d* и подписываемое сообщение *М*, а выходным результатом – цифровая подпись в виде двух параметров *r* и *s*.

# Проверка цифровой подписи

Для проверки цифровой подписи под полученным сообщением *M* необходимо выполнить следующие шаги:

1. Получение параметров *r* и *s* – цифровой подписи сообщения *М*. Если выполнены неравенства  и , то перейти к следующему шагу. В противном случае подпись неверна.
2. Вычислить хеш-значение полученного сообщения *М*:



1. Вычислить целое значение *е*:



Если *e* = 0, то определить *e* = 1.

1. Вычислить значение:



1. Вычислить значения:



1. Вычислить точку эллиптической кривой  и определить:



Где: *xC*– *x* координата точки *С*.

1. Если выполнено равенство , то подпись принимается, в противном случае – подпись неверна.

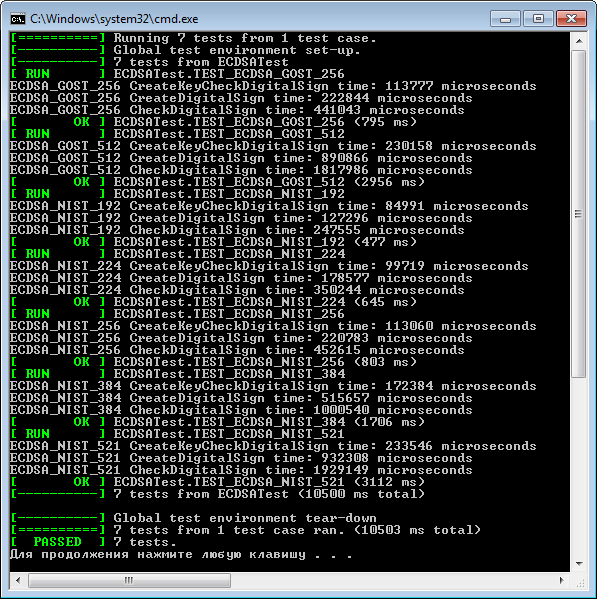
Исходными данными этого процесса являются подписанное сообщение *М*, цифровая подпись в виде двух параметров *r* и *s*, а также ключ проверки подписи *Q*, а выходным результатом – свидетельство о достоверности или ошибочности данной подписи.

# Результаты реализации алгоритма ECDSA

Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация алгоритма ЦП ECDSA входит в проект ECDSA решения CryptoProtocols.

Для тестирования корректности разрабатываемых проектов в решении CryptoProtocols был создан отдельный проект GoogleTestingSolutionProject модульного тестирования gtest (для unit testing) и gmock (для проверки корректности вызовов методов). Данные пакеты устанавливались через менеджер пакетов NuGet для Visual Studio.

Результат выполнения тест кейсов для проверки корректности работы формирования и проверки ЦП на различных видах кривых NIST и GOST, а также фиксация времени выполнения отдельных элементов в процессе ЦП (т.к. gtest замеряет работу вызовов кейсов в микросекундах, то для повышения точности была использована библиотека <chrono> c++11 с точностью до микросекунд) приведены на Рис. 1.



**Рис. 1.** Результат тестирования реализованного алгоритма ECDSA

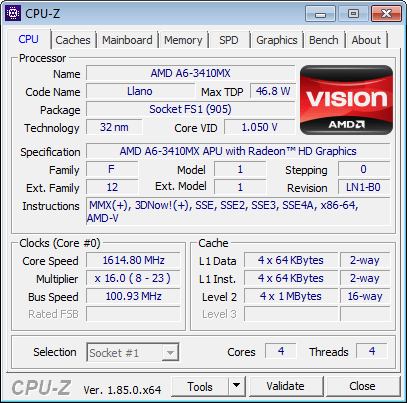
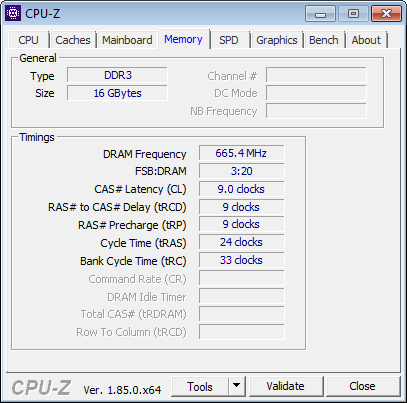
Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.2. По полученным данным увидим:

- время выработки ключа проверки ЦП (*Q*);

- время создания ЦП;

- время проверки ЦП.

для данного ЦП. Данные приведены в Табл. 1.

**Рис. 2.** ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

**Табл. 1.** Скорость выполнения операций ЦП

|  |  |  |  |
| --- | --- | --- | --- |
| Используемая хеш-функция | | Выработка ключа проверки ЦП / Создание ЦП/ Проверка ЦП | Скорость  [секунд] |
| *Кривая ГОСТ Р-256* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,114 |
| SHA-512 | Создание ЦП | | 0,223 |
| SHA-512 | Проверка ЦП | | 0,441 |
| *Кривая ГОСТ Р-512* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,230 |
| SHA-512 | Создание ЦП | | 0,891 |
| SHA-512 | Проверка ЦП | | 1,818 |
| *Кривая NIST P-192* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,085 |
| SHA-512 | Создание ЦП | | 0,127 |
| SHA-512 | Проверка ЦП | | 0,248 |
| *Кривая NIST P-224* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,100 |
| SHA-512 | Создание ЦП | | 0,179 |
| SHA-512 | Проверка ЦП | | 0,350 |
| *Кривая NIST P-256* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,113 |
| SHA-512 | Создание ЦП | | 0,221 |
| SHA-512 | Проверка ЦП | | 0,453 |
| *Кривая NIST P-384* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,172 |
| SHA-512 | Создание ЦП | | 0,516 |
| SHA-512 | Проверка ЦП | | 1,001 |
| *Кривая NIST P-521* | | | |
| SHA-512 | Выработка ключа проверки ЦП | | 0,234 |
| SHA-512 | Создание ЦП | | 0,932 |
| SHA-512 | Проверка ЦП | | 1,929 |

# Литература

1. NIST FIPS PUB 186-4 «Digital Signature Standard (DSS)» [Интернет ресурс], ссылка: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
2. ГОСТ Р 34.10-2012 «Процессы формирования и проверки электронной цифровой подписи» [Интернет ресурс], ссылка: <http://docs.cntd.ru/document/gost-r-34-10-2012>

# Листинг кода

файл ECDSA\_PrimeField.h

#ifndef ECDSA\_PrimeField\_H

#define ECDSA\_PrimeField\_H

#include "ttmath/ttmath.h"

#include "ttmath/ttmathint.h"

#include <iostream>

#include <string>

#include <memory>

using namespace std;

using bigint = ttmath::Int<32>;

/\*Class ECDSA\_PrimeField\*/

class ECDSA\_PrimeField {

private:

/\*Curve Coefficients\*/

bigint \_a; // Coefficient 'a' of Curve equal y^2 = x^3 + a\*x + b (mod p) | Anallogically abbreviation in GOST 34.10-2012

bigint \_b; // Coefficient 'b' of Curve equal y^2 = x^3 + a\*x + b (mod p) | Anallogically abbreviation in GOST 34.10-2012

bigint \_p; // Prime module of Curve equal y^2 = x^3 + a\*x + b (mod p) | Anallogically abbreviation in GOST 34.10-2012

bigint \_Gx; // Coordinate 'x' of Point G of Elliptical Curve [ y^2 = x^3 + a\*x + b (mod p) ] order n | In GOST 34.10-2012 it is 'x' coordinate of point P

bigint \_Gy; // Coordinate 'y' of Point G of Elliptical Curve [ y^2 = x^3 + a\*x +b (mod p) ] order n | In GOST 34.10-2012 it is 'y' coordinate of point P

bigint \_n; // Order of SubGroup of Points of Elliptical Curve [ y^2 = x^3 + a\*x + b (mod p) ] | In GOST 34.10-2012 it is 'q' parametr

string hexStr(vector<uint8\_t>\* hexArray);

public:

//Return Public parameters : first - 'r', second - 's'

pair<string, string> CreateDigitalSign(const string& PrivateKeyDigitalSign, const string& Message);

bool CheckDigitalSign(const pair<string, string>& DigitalSign, const string& Message, const pair<string, string>& KeyCheckDigitalSign);

//Return KeyCheckDigitalSign <=> Public Elliptic Curve Point Q : first - 'x' coordinate of Q point, second - 'y' coordinate of Q point, [PrivateKeyDigitalSign is Big number in dec system]

pair<string, string> CreateKeyCheckDigitalSign(const string& PrivateKeyDigitalSign);

pair<string, string> MultiplyOnBasePoint(const bigint& Number);

public:

ECDSA\_PrimeField(bigint& a, bigint& b, bigint& p, bigint& Gx, bigint& Gy, bigint& n);

~ECDSA\_PrimeField();

friend class ECPoint;

};

/\*Class ECPoint\*/

class ECPoint {

private:

/\*Elliptical Curve Point Coordinates 'x' and 'y' \*/

bigint \_x;

bigint \_y;

ECDSA\_PrimeField\* \_parentECDSA{ nullptr };

ECPoint DoubleAndAdd(const bigint& k, const ECPoint& point);

static bigint ReverseElementInField(const bigint& Element, const bigint& Module);

static void ExtendedEuclidAlgorithm(bigint& a, bigint& b, bigint& x, bigint& y, bigint& d);

public:

void setCoordinate(const string& x, const string& y);

string getXCoordinate();

string getYCoordinate();

ECPoint& operator = (const ECPoint& rhs);

ECPoint operator + (const ECPoint& rhs);

ECPoint operator \* (const bigint& rhs);

bool operator == (const ECPoint& rhs);

ECPoint(ECDSA\_PrimeField\* parentECDSA);

~ECPoint();

friend class ECDSA\_PrimeField;

};

#endif //ECDSA\_PRIMEFIELD\_H

файл ECDSA\_PrimeField.cpp

#include "ECDSA\_PrimeField.h"

#include "../SHA512\_Hash/SHA512.h"

#include "../CSPRNG/CSPRNG.h"

#include <list>

using namespace AlgorithmSHA512;

const uint8\_t hexmap[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };

/\*Start ECPoint Methods\*/

void ECPoint::setCoordinate(const string& x, const string& y){

\_x.FromString(x, 10);

\_y.FromString(y, 10);

};

string ECPoint::getXCoordinate(){ return this->\_x.ToString(); }

string ECPoint::getYCoordinate() { return this->\_y.ToString(); }

ECPoint ECPoint::DoubleAndAdd(const bigint& k, const ECPoint& pointP){

/\*Calculating binary representation of 'k'\*/

bigint CurrentNumberPower2("1");

int wPower = 0;

while (CurrentNumberPower2\*2 <= k) {

CurrentNumberPower2 \*= 2;

wPower++;

}

list<bool> tempFormatBinaryPowerK;

bigint tempk;

tempk = k;

for (int i = wPower; i >= 0; i--) {

if (tempk >= CurrentNumberPower2) {

tempk -= CurrentNumberPower2;

tempFormatBinaryPowerK.push\_front(true);

}

else {

tempFormatBinaryPowerK.push\_front(false);

}

CurrentNumberPower2 /= 2;

}

vector<bool> FormatBinaryPowerK(tempFormatBinaryPowerK.begin(), tempFormatBinaryPowerK.end());

/\*Double And Add Algorithm of Fast Multiplication\*/

ECPoint N(\_parentECDSA);

ECPoint Q(\_parentECDSA);

Q.setCoordinate("-1", "-1");

N = pointP;

for (uint16\_t i = 0; i < FormatBinaryPowerK.size(); i++) {

if (FormatBinaryPowerK[i] == 1) {

Q = Q + N;

}

N = N + N;

}

return Q;

};

bigint ECPoint::ReverseElementInField(const bigint& Element, const bigint& Module){

bigint a, b, x, y, d;

a = Element;

b = Module;

if (a < 0) { a = (a % b) + b; }

ExtendedEuclidAlgorithm(a, b, x, y, d);

if (x < 0) { x += Module; }

return x;

};

void ECPoint::ExtendedEuclidAlgorithm(bigint& a, bigint& b, bigint& x, bigint& y, bigint& d){

bigint q, r, x1, x2, y1, y2;

if (b == 0) {

x = 0;

return;

}

x2 = 1;

x1 = 0;

y2 = 0;

y1 = 1;

while (b > 0) {

q = a / b;

r = a - q\*b;

x = x2 - q\*x1;

y = y2 - q\*y1;

a = b;

b = r;

x2 = x1;

x1 = x;

y2 = y1;

y1 = y;

}

d = a;

x = x2;

y = y2;

};

/\*Check that 2 Points belong to the one common Elliptic Curve\*/

bool ECPoint::operator==(const ECPoint& rhs) {

if (\_parentECDSA == rhs.\_parentECDSA) return true;

else return false;

};

ECPoint::ECPoint(ECDSA\_PrimeField\* parentECDSA){

\_parentECDSA = parentECDSA;

return;

};

ECPoint::~ECPoint(){

return;

};

ECPoint ECPoint::operator + (const ECPoint& rhs) {

if (\*this == rhs) {

ECPoint ecp(\_parentECDSA);

if ((rhs.\_x >= 0 && rhs.\_y >= 0) && (\_x >= 0 && \_y >= 0)) {

if (\_x != rhs.\_x) {

bigint Lambda = ((rhs.\_y - \_y) \* ReverseElementInField((rhs.\_x - \_x), \_parentECDSA->\_p)) % \_parentECDSA->\_p;

Lambda = Lambda < 0 ? Lambda + \_parentECDSA->\_p : Lambda;

ecp.\_x = ((Lambda\*Lambda) - \_x - rhs.\_x) % \_parentECDSA->\_p;

ecp.\_x = ecp.\_x < 0 ? ecp.\_x + \_parentECDSA->\_p : ecp.\_x;

ecp.\_y = ((Lambda\*(\_x - ecp.\_x)) - \_y) % \_parentECDSA->\_p;

ecp.\_y = ecp.\_y < 0 ? ecp.\_y + \_parentECDSA->\_p : ecp.\_y;

return ecp;

}

else if ((\_x == rhs.\_x) && (\_y == ((-rhs.\_y) % \_parentECDSA->\_p < 0 ? ((-rhs.\_y) % \_parentECDSA->\_p) + \_parentECDSA->\_p : (-rhs.\_y) % \_parentECDSA->\_p))) {

ecp.\_x = -1;

ecp.\_y = -1;

return ecp;

}

else if ((\_x == rhs.\_x) && (\_y == rhs.\_y) && (\_y != 0) && (rhs.\_y != 0)) {

bigint Lambda = (((\_x\*\_x \* 3) + \_parentECDSA->\_a) \* ReverseElementInField((\_y \* 2), \_parentECDSA->\_p)) % \_parentECDSA->\_p;

Lambda = Lambda < 0 ? Lambda + \_parentECDSA->\_p : Lambda;

ecp.\_x = ((Lambda\*Lambda) - (\_x \* 2)) % \_parentECDSA->\_p;

ecp.\_x = ecp.\_x < 0 ? ecp.\_x + \_parentECDSA->\_p : ecp.\_x;

ecp.\_y = ((Lambda\*(\_x - ecp.\_x)) - \_y) % \_parentECDSA->\_p;

ecp.\_y = ecp.\_y < 0 ? ecp.\_y + \_parentECDSA->\_p : ecp.\_y;

return ecp;

}

}

/\*Check if in addition attend NULL Elliptic Curve Point\*/

else {

if (\_x == -1 && \_y == -1) {

ecp.\_x = rhs.\_x;

ecp.\_y = rhs.\_y;

return ecp;

}

else if (rhs.\_x == -1 && rhs.\_y == -1) {

ecp.\_x = \_x;

ecp.\_y = \_y;

return ecp;

}

else {

ecp.\_x = -1;

ecp.\_y = -1;

return ecp;

}

}

}

else {

throw 1;

cout << "here!" << endl;

return \*this;

}

};

ECPoint ECPoint::operator \* (const bigint& rhs) {

//Check, that 0 < rhs < n

bigint \_d(rhs);

\_d %= (\*this->\_parentECDSA).\_n;

\_d = \_d < 0 ? \_d + (\*this->\_parentECDSA).\_n : \_d;

return DoubleAndAdd(\_d, \*this);

};

ECPoint& ECPoint::operator = (const ECPoint& rhs){

\_x = rhs.\_x;

\_y = rhs.\_y;

\_parentECDSA = rhs.\_parentECDSA;

return \*this;

};

/\*End ECPoint Methods\*/

/\*Start ECDSA\_PrimeField Methods\*/

string ECDSA\_PrimeField::hexStr(vector<uint8\_t> \*data){

string s(data->size() \* 2, ' ');

for (register uint64\_t i = 0; i < data->size(); ++i) {

s[2 \* i] = hexmap[((\*data)[i] & 0xF0) >> 4];

s[2 \* i + 1] = hexmap[(\*data)[i] & 0x0F];

}

return s;

};

pair<string, string> ECDSA\_PrimeField::CreateDigitalSign(const string& d, const string& Message){

//Create Elliptic Curve Point G

ECPoint \_G(this);

\_G.\_x = \_Gx;

\_G.\_y = \_Gy;

bigint \_d;

\_d.FromString(d, 10);

\_d %= \_n;

\_d = \_d < 0 ? \_d + \_n : \_d;

//STEP 1

SHA512 HashCalculateObject;

//vector<uint8\_t> \_Message(Message.begin(), Message.end());

vector<uint8\_t>\* \_MessageDigest;

\_MessageDigest = HashCalculateObject.GetHash(&vector<uint8\_t>(Message.begin(), Message.end()));

//STEP 2

bigint \_Alpha;

\_Alpha.FromString(hexStr(\_MessageDigest), 16);

delete \_MessageDigest;

bigint e = \_Alpha % \_n;

e = (e < 0 ? e + \_n : e);

e = (e == 0 ? 1 : e);

bigint r = 0;

bigint s = 0;

bigint k;

while (s == 0) {

while (r == 0) {

//STEP 3

//Generate PRN k : 0 < k < n

CSPRNG generatorPRNG;

vector<uint8\_t>\* byarrPRN;

byarrPRN = generatorPRNG.GeneratePRN(256);

//PRN k in bigint format

k.FromString(hexStr(byarrPRN), 16);

delete byarrPRN;

k %= \_n;

k = k < 0 ? k + \_n : k;

//STEP 4

//Create Elliptic Curve Point C

ECPoint C(this);

C = \_G\*k;

//Calculate r = Xc(mod q), where Xc - 'x' coordinate of Elliptic Curve C

r = C.\_x % \_n;

r = r < 0 ? r + \_n : r;

}

//STEP 5

s = (r\*\_d + k\*e) % \_n;

s = s < 0 ? s + \_n : s;

}

//STEP 6

//Create Digital Sign

return pair<string, string>(r.ToString(), s.ToString());

};

bool ECDSA\_PrimeField::CheckDigitalSign(const pair<string, string>& DigitalSign, const string& Message, const pair<string, string>& Q){

bigint r;

bigint s;

r.FromString(DigitalSign.first, 10);

s.FromString(DigitalSign.second, 10);

ECPoint \_G(this);

\_G.\_x = \_Gx;

\_G.\_y = \_Gy;

ECPoint \_Q(this);

\_Q.setCoordinate(Q.first, Q.second);

//STEP 1

if (!((r > 0 && r < \_n) && (s > 0 && s < \_n))) { return false; }

//STEP 2

SHA512 HashCalculateObject;

vector<uint8\_t> \_Message(Message.begin(), Message.end());

vector<uint8\_t>\* \_MessageDigest;

\_MessageDigest = HashCalculateObject.GetHash(&\_Message);

//STEP 3

bigint \_Alpha;

\_Alpha.FromString(hexStr(\_MessageDigest), 16);

delete \_MessageDigest;

bigint e = \_Alpha % \_n;

e = (e < 0 ? e + \_n : e);

e = (e == 0 ? 1 : e);

//STEP 4

bigint v = ECPoint::ReverseElementInField(e, \_n);

//STEP 5

bigint z1 = (s\*v) % \_n;

z1 = z1 < 0 ? z1 + \_n : z1;

bigint z2 = (-r\*v % \_n);

z2 = z2 < 0 ? z2 + \_n : z2;

//STEP 6

ECPoint C(this);

C = \_G\*z1 + \_Q\*z2;

bigint R = C.\_x % \_n;

R = R < 0 ? R + \_n : R;

if (R != r) { return false; }

return true;

};

pair<string, string> ECDSA\_PrimeField::CreateKeyCheckDigitalSign(const string& d){

bigint \_d;

\_d.FromString(d, 10);

\_d %= \_n;

\_d = \_d < 0 ? \_d + \_n : \_d;

ECPoint \_G(this);

\_G.\_x = \_Gx;

\_G.\_y = \_Gy;

ECPoint \_Q(this);

\_Q = \_G\*\_d;

return pair<string, string>(\_Q.\_x.ToString(), \_Q.\_y.ToString());

};

pair<string, string> ECDSA\_PrimeField::MultiplyOnBasePoint(const bigint& Number){

bigint \_d(Number);

//Check that 0 < \_d < n

\_d %= \_n;

\_d = \_d < 0 ? \_d + \_n : \_d;

//Create Elliptic Curve Point G

ECPoint \_G(this);

\_G.\_x = \_Gx;

\_G.\_y = \_Gy;

ECPoint \_Q(this);

\_Q = \_G\*\_d;

return pair<string, string>(\_Q.\_x.ToString(), \_Q.\_y.ToString());

};

ECDSA\_PrimeField::ECDSA\_PrimeField(bigint& a, bigint& b, bigint& p, bigint& Gx, bigint& Gy, bigint& n){

\_a = a;

\_b = b;

\_p = p;

\_Gx = Gx;

\_Gy = Gy;

\_n = n;

};

ECDSA\_PrimeField::~ECDSA\_PrimeField(){

return;

};

/\*End ECDSA\_PrimeField Methods\*/

файл ECDSA.h

#ifndef ECDSA\_H

#define ECDSA\_H

#include "ECDSA\_PrimeField.h"

using namespace std;

class ECDSA\_GOST\_256 : public ECDSA\_PrimeField {

public:

ECDSA\_GOST\_256() : ECDSA\_PrimeField(

bigint("7"),

bigint("43308876546767276905765904595650931995942111794451039583252968842033849580414"),

bigint("57896044618658097711785492504343953926634992332820282019728792003956564821041"),

bigint("2"),

bigint("4018974056539037503335449422937059775635739389905545080690979365213431566280"),

bigint("57896044618658097711785492504343953927082934583725450622380973592137631069619")

) { };

};

class ECDSA\_GOST\_512 : public ECDSA\_PrimeField {

public:

ECDSA\_GOST\_512() : ECDSA\_PrimeField(

bigint("7"),

bigint("1518655069210828534508950034714043154928747527740206436194018823352809982443793732829756914785974674866041605397883677596626326413990136959047435811826396"),

bigint("3623986102229003635907788753683874306021320925534678605086546150450856166624002482588482022271496854025090823603058735163734263822371964987228582907372403"),

bigint("1928356944067022849399309401243137598997786635459507974357075491307766592685835441065557681003184874819658004903212332884252335830250729527632383493573274"),

bigint("2288728693371972859970012155529478416353562327329506180314497425931102860301572814141997072271708807066593850650334152381857347798885864807605098724013854"),

bigint("3623986102229003635907788753683874306021320925534678605086546150450856166623969164898305032863068499961404079437936585455865192212970734808812618120619743")

) { };

};

class ECDSA\_NIST\_192 : public ECDSA\_PrimeField {

public:

ECDSA\_NIST\_192() : ECDSA\_PrimeField(

bigint("-3"),

bigint("2455155546008943817740293915197451784769108058161191238065"),

bigint("6277101735386680763835789423207666416083908700390324961279"),

bigint("602046282375688656758213480587526111916698976636884684818"),

bigint("174050332293622031404857552280219410364023488927386650641"),

bigint("6277101735386680763835789423176059013767194773182842284081")

) { };

};

class ECDSA\_NIST\_224 : public ECDSA\_PrimeField {

public:

ECDSA\_NIST\_224() : ECDSA\_PrimeField(

bigint("-3"),

bigint("18958286285566608000408668544493926415504680968679321075787234672564"),

bigint("26959946667150639794667015087019630673557916260026308143510066298881"),

bigint("19277929113566293071110308034699488026831934219452440156649784352033"),

bigint("19926808758034470970197974370888749184205991990603949537637343198772"),

bigint("26959946667150639794667015087019625940457807714424391721682722368061")

) { };

};

class ECDSA\_NIST\_256 : public ECDSA\_PrimeField {

public:

ECDSA\_NIST\_256() : ECDSA\_PrimeField(

bigint("-3"),

bigint("41058363725152142129326129780047268409114441015993725554835256314039467401291"),

bigint("115792089210356248762697446949407573530086143415290314195533631308867097853951 "),

bigint("48439561293906451759052585252797914202762949526041747995844080717082404635286"),

bigint("36134250956749795798585127919587881956611106672985015071877198253568414405109"),

bigint("115792089210356248762697446949407573529996955224135760342422259061068512044369")

) { };

};

class ECDSA\_NIST\_384 : public ECDSA\_PrimeField {

public:

ECDSA\_NIST\_384() : ECDSA\_PrimeField(

bigint("-3"),

bigint("27580193559959705877849011840389048093056905856361568521428707301988689241309860865136260764883745107765439761230575"),

bigint("39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319"),

bigint("26247035095799689268623156744566981891852923491109213387815615900925518854738050089022388053975719786650872476732087"),

bigint("8325710961489029985546751289520108179287853048861315594709205902480503199884419224438643760392947333078086511627871"),

bigint("39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643")

) { };

};

class ECDSA\_NIST\_521 : public ECDSA\_PrimeField {

public:

ECDSA\_NIST\_521() : ECDSA\_PrimeField(

bigint("-3"),

bigint("1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984"),

bigint("6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151"),

bigint("2661740802050217063228768716723360960729859168756973147706671368418802944996427808491545080627771902352094241225065558662157113545570916814161637315895999846"),

bigint("3757180025770020463545507224491183603594455134769762486694567779615544477440556316691234405012945539562144444537289428522585666729196580810124344277578376784"),

bigint("6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449")

) { };

};

#endif

# *Реализация ГПСЧ. Алгоритм CSPRNG AES256\_OFB*

**Лабораторная работа №4**

Для выполнения лабораторной работы по реализации криптографического протокола TLS необходима реализация криптографических примитивов, которые используются при построении протокола. Одним из таковых является генератор псевдослучайных чисел (ГПСЧ).

Генераторы случайных чисел - ключевая часть безопасности, наибольшее применение находит в генерации паролей. В качестве генератора случайных чисел в данной работе будет использоваться алгоритм AES256 в режиме OFB. Проверка на случайность последовательности будет осуществляться на батарее NIST тестов пакета Dieharder.

# Основные понятия о ГПСЧ

Для определения понятия ГПСЧ введем некоторые понятия:

Случайное число – число, представляющее собой реализацию случайной величины.

Детерминированный алгоритм – алгоритм, который возвращает те же выходные значения при тех же входных значениях.

Псевдослучайное число – число, полученное детерминированным алгоритмом, используемое в качестве случайного числа.

Физическое случайное число (истинно случайное) – случайное число, полученное на основе некоторого физического явления.

Как правило, генерация случайного числа состоит из двух этапов:

1. генерация нормализованного случайного числа (то есть равномерно распределенного от 0 до 1);
2. преобразование нормализованных случайных чисел  в случайные числа , которые распределены по заданному закону распределения или в необходимом интервале.

Генератор случайных бит(ГСБ) — это устройство или алгоритм, который выдает последовательность статистически независимых и несмещенных бит (то есть подчиняющихся закону распределения).

Генератор случайных бит может быть использован для генерации равномерно распределенных случайных чисел. Например, случайное целое число в интервале  может быть получено из сгенерированной последовательности случайных бит длины путем конвертации её в соответствующую систему исчисления.

Если полученное в результате целое число превосходит *n*, то его можно отбросить и сгенерировать еще одну последовательность бит. Поэтому далее мы будем использовать термин генератор случайных чисел наравне с термином генератор случайных бит.

Генератором псевдослучайных бит (детерминированным ГПСБ) – будем называть детерминированный алгоритм (функция), который получает на вход двоичную последовательность длины *k* и выдает на выходе двоичную последовательность длины *l* ≫ *k* (*l* значительно больше *k*), которая «выглядит случайной»[[1]](#footnote-1). Входное значение ГПСБ называется начальным вектором (также называют инициализационным вектором и обозначают *IV*), а выход называется псевдослучайной последовательностью бит.

Говорят, что ГПСБ проходит все полиномиальные по времени вероятностные тесты на статистическую случайность, если не существует полиномиального по времени[[2]](#footnote-2) вероятностного алгоритма, который бы мог корректно отличить выходную последовательность генератора от истинно случайной последовательности той же длины с вероятностью превышающей .

Говорят, что ГПСБ успешно проходит тест на следующий бит, если не существует полиномиального по времени алгоритма, который может по входным *l* битам последовательности *s* предсказать (*l* + 1)-й бит *s* с вероятностью превышающей .

# Виды ГПСЧ

Генераторы случайных чисел по способу получения чисел делятся на:

* + - аппаратные;
    - табличные;
    - алгоритмические.

Аппаратные генераторы (истинно) случайных последовательностей должны обладать источником энтропии[[3]](#footnote-3). Разработка генераторов, использующих источники энтропии, генерирующих некоррелированные и статистически независимые числа – достаточно сложная задача. Кроме того, для большинства криптографических приложений такой ГПСЧ не должен быть предметом изучения и воздействий стороны противника.

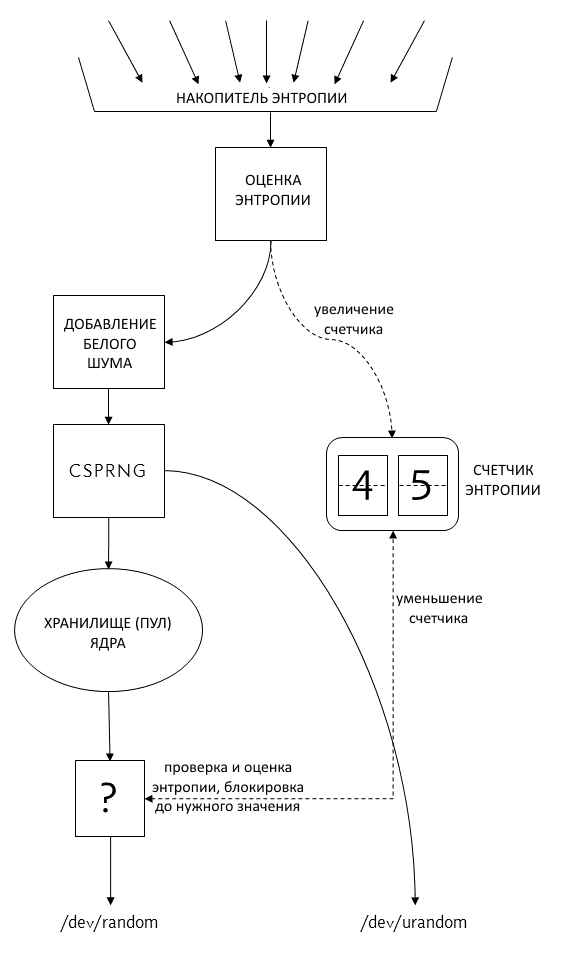
Табличные генераторы в качестве источника случайных чисел используют заранее подготовленные таблицы, содержащие проверенные некоррелированные числа и не являются генераторами в строгом понимании этого понятия. Недостатки такого способа очевидны: использование внешнего ресурса для хранения чисел, ограниченность последовательности, предопределенность значений. В качестве примера табличного метода можно привести книгу.

Алгоритмический генератор является комбинацией физического генератора и детерминированного алгоритма. Такой генератор использует ограниченный набор данных, полученный с выхода физического генератора для создания длинной последовательности чисел преобразованиями исходных чисел. Данный вид генераторов представляет наибольший интерес в силу его очевидных преимуществ над генераторами случайных чисел других видов.

# Пример реализации ГПСЧ

В качестве реализуемого ГПСЧ возьмем структурную схему ГПСЧ из библиотек /dev/random и /dev/urandom ядра операционной системы Linux.

*Структура Linux’s ГСЧ/ГПСЧ*



Работа схемы заключается в следующем. Существует три накопителя энтропии:

* первичный;
* для /dev/random;
* для /dev/urandom.

Последние два накопителя получают данные из первичного. У каждого накопителя присутствует свой счетчик энтропии, однако для /dev/random и /dev/urandom они близки к 0 (нулю). Для увеличения их энтропии при запросе пользователя они используют в качестве источника энтропии первичный накопитель.

Первичный накопитель энтропии собирает её из различных источников (физических/аппаратных датчиков [USB контроллер подключаемых временных устройств, датчики температуры, встроенные часы, положение указателя мыши, время нажатия клавиш на клавиатуре и другие]). Вычисляется энтропия этой накопленной информации и немедленно это значение добавляется к значению счетчика энтропии. Если энтропия принятой информации имеет малое значение, то происходит коррекция до того момента, пока значение энтропии будет в пределах нормы, установленной в параметрах генератора.

После исправления недостатков происходит привнесение белого шума (равномерно распределенных битов).

CSPRNG представляет собой стойкий криптографический ГПСЧ, то есть ГПСЧ с определенными свойствами, позволяющими использовать его в криптографии. Одна из возможных реализаций CSPRNG основывается на использовании криптографических алгоритмов.

Примером такой реализации может выступать безопасный блочный шифр, который преобразуется в режиме счетчика (Counter mode [CTR]) или гаммирования с обратной связью (Output Feed Back mode [OFB]) в ГПСЧ (работает по принципу поточного шифра). Таким образом, выбрав случайный ключ, можно получать следующий случайный блок. Очевидно, что периодом такого генератора будет не больше, чем  для *n*-битного блочного шифра. Также очевидно, что безопасность такой схемы полностью зависит от секретного ключа.

В роли CSPRNG может выступать и криптографически стойкая хеш-функция. В таком случае исходное значение счетчика должно оставаться в секрете.

Поточные шифры работают на основе генерации псевдослучайного потока бит, которые некоторым образом комбинируются (с помощью операции XOR) с битами открытого текста. Запуск такого шифра на входной последовательности даст новую псевдослучайную последовательность, возможно, даже с более длинным периодом. Такой метод безопасен, только если в самом поточном шифре используется надежный криптографически стойкий ГПСЧ. При этом, начальное состояние счетчика должно оставаться секретным.

После прохождения СSPRNG информация попадает в хранилище (пул) ядра, откуда /dev/urandom берет псевдослучайные числа, получая их из пула напрямую, если у счетчика энтропии имеется запрашиваемое количество чисел (бит). Для /dev/random происходит оценка энтропии полученной информации и только после принятия решения результат поступает на /dev/random к пользователю.

Существует большое множество криптографически стойких блочных шифров. Один из них – AES256 с размером блока 128 бит, который можно использовать в режиме OFB, чтобы получить хорошую ПСП.

Проверим получаемую ПСП, реализуемую данным алгоритмом в режиме гаммирования с обратной связью, на статистические тесты, входящие в пакет статистических тестов Dieharder, предлагаемые NIST в документе NIST SP 800-22.

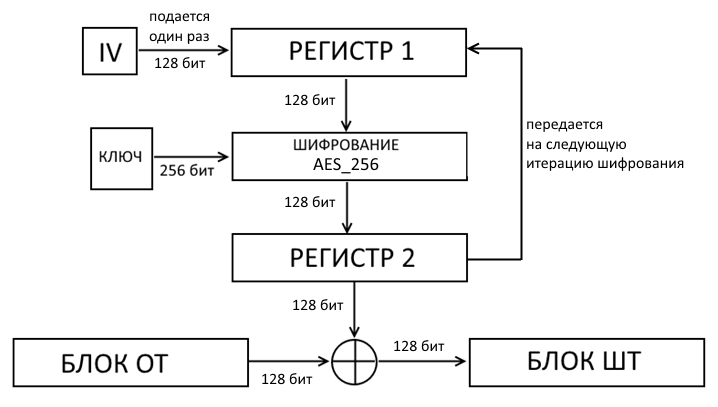
Статистические тесты NIST – пакет статистических тестов, разработанный Лабораторией информационных технологий, являющейся главной исследовательской организацией Национального института стандартов и технологий (NIST). В его состав входят 15 статистических тестов, целью которых является определение меры случайности двоичных последовательностей, порожденных либо аппаратными, либо программными ГСЧ.

В пакет тестов входят:

* Частотный побитовый тест;
* Частотный блочный тест;
* Тест на последовательность одинаковых битов;
* Тест на самую длинную последовательность единиц в блоке;
* Тест рангов бинарных матриц;
* Спектральный тест;
* Тест на совпадение неперекрывающихся шаблонов;
* Тест на совпадение перекрывающихся шаблонов;
* Универсальный статистический тест Маурера;
* Тест на линейную сложность;
* Тест на периодичность;
* Тест приблизительной энтропии;
* Тест кумулятивных сумм;
* Тест на произвольные отклонения;
* Другой тест на произвольные отклонения.

# ГПСЧ на основе AES256\_OFB

Рассмотрим работу блочного шифра AES256 в режиме OFB (гаммирования с обратной связью) в виде структурной схемы:



Режим гаммирования с обратной связью работает следующим образом. Содержимое РЕГИСТР 1 сначала получает вектор инициализации (IV), затем перед каждым шифрованием получает содержимое из РЕГИСТРА 2 (результат работы алгоритма AES256).

Открытый текст не шифруют напрямую: вначале шифруется вектор инициализации (IV), а уже полученный в результате шифртекст ксорится (XOR) с блоком открытого текста. Затем шифруется результат работы алгоритма AES256 на предыдущем шаге и ксорится (XOR) со следующим блоком открытого текста и так далее.

Таким образом. работа в режиме OFB заключается в следующем:

ВХОД:

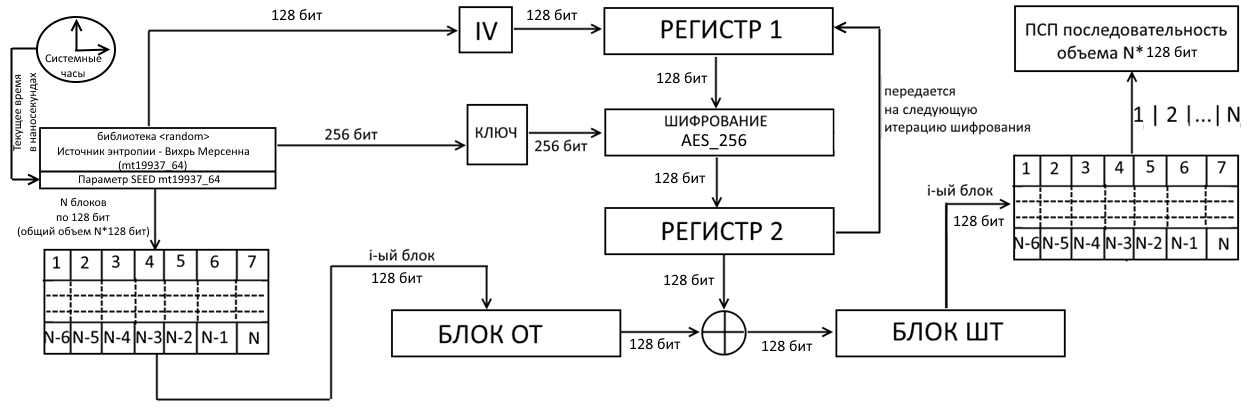
* вектор инициализации IV (128 бит);
* ключ (256 бит);
* блоки открытого текста [ОТ] (128 бит).

ВЫХОД:

* блоки шифртекста [ШТ] (128 бит).

Конкатенируя блоки ШТ на выходе алгоритма мы получаем ПСП, которую можно проверить на тесты NIST SP 800-22.

Структурная схема предложенного ГПСЧ будет выглядеть следующим образом:



Для моделирования источника накопления энтропии будем использовать библиотеку языка C++11 <random>, в котором реализован криптографически нестойкий ГПСЧ Вихрь Мерсенна (mt19937\_64), который для генерации ПСП принимает на вход значение SEED (семени). Семя, как вариант, можно получать из текущего значения системных часов (в наносекундах). Вихрь Мерсенна будет вырабатывать в нашем эксперименте:

* вектор инициализации IV (объемом 128 бит = 16 байт);
* энтропия [которая на схеме отмечается как ОТ] (объемом 83886080 бит = 10485760 байт = 655360 блоков размером 128 бит);
* ключ (объемом 256 бит = 32 байта).

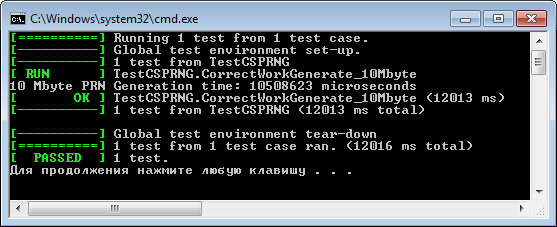
Далее был применен алгоритм выработки ПСП через алгоритм AES256 в режиме OFB (гаммирования с обратной связью).

# Результаты реализации ГПСЧ AES256\_OFB

Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация алгоритма ГПСЧ входит в проект CSPRNG решения CryptoProtocols.

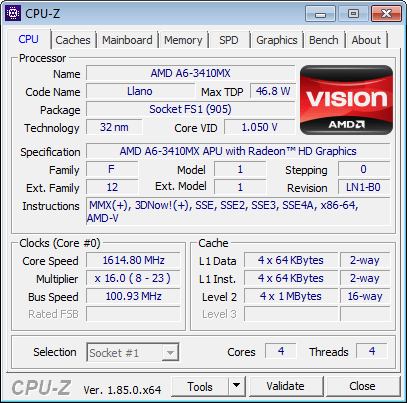
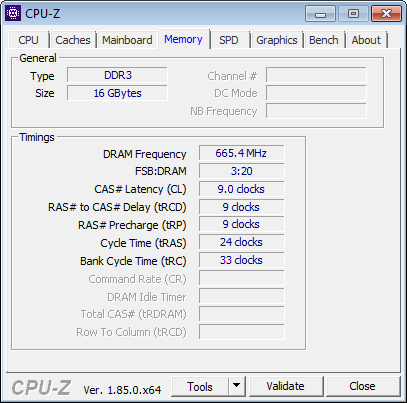
Для тестирования корректности разрабатываемых проектов в решении CryptoProtocols был создан отдельный проект GoogleTestingSolutionProject модульного тестирования gtest (для unit testing) и gmock (для проверки корректности вызовов методов). Данные пакеты устанавливались через менеджер пакетов NuGet для Visual Studio.

Результат выполнения тест кейсов для проверки корректности работы функции выработки ПСП и времени выполнения для подсчета производительности работы (т.к. gtest замеряет работу вызовов кейсов в микросекундах, то для повышения точности была использована библиотека <chrono> c++11 с точностью до микросекунд) приведены на Рис. 1.



**Рис. 1.** Результат тестирования реализованного алгоритма CSPRNG

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис. 2. По полученным данным посчитаем время выработки ПСП для данного ЦП. Данные приведены в Табл. 1.

**Рис. 2.** ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

**Табл. 1.** Скорость выполнения выработки ПСП алгоритма CSPRNG

|  |  |  |
| --- | --- | --- |
| Алгоритм | Размер данных [Мбайт] | Скорость [Мбайт/с] |
| AES256\_OFB | 10 | 0,95159946265 |

Проверим полученную последовательность на батарею тестов NIST SP800-22 из пакета Dieharder.

Запуск проверки сгенерированной последовательности осуществляется командой:

dieharder -g 201 -f testrands.txt -a

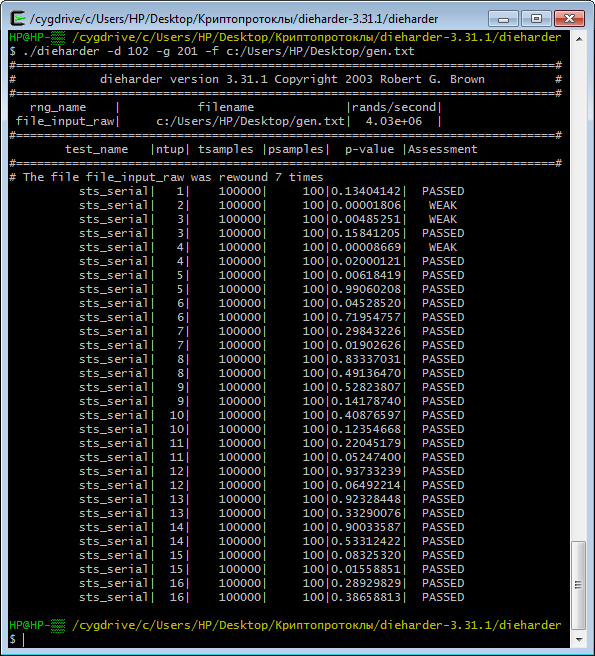
Где:

* + - -g 201 (формат тестируемых данных – полученный на выходе ГПСЧ файл ASCII с ПСП);
    - -f (указывает путь к файлу testrands.txt);
    - -a (выполнить проверку по всем тестам, которые есть в сборке библиотеки, посмотреть конкретные тесты можно флагом -l, запуск через флаг -d [номер теста])

Батарея тестов NIST SP800-22 в пакете Dieharder имеет номер -d 102

dieharder -g 201 -f testrands.txt -d 102

Результат проверки последовательности длиной 83886080 бит = 10 Мбайт приведен на Рис. 3.



**Рис. 3.** Результат проверки последовательности длиной 83886080 бит = 10 Мбайт

Таким образом, последовательность, выработанная реализованным CSPRNG, прошла проверку на случайность, значит данный ГПСЧ пригоден для дальнейшего использования.

# Литература

1. Зязин В.П. «Курс лекций ПСП», РТУ (МИРЭА), 2018 - 2019 г.
2. Режимы шифрования блочных шифров. [Интернет ресурс], ссылка https://ru-wiki.ru/wiki/Режим\_шифрования.
3. Meths about /dev/urandom. [Интернет ресурс], ссылка https://www.2uo.de/myths-about-urandom/#structure.
4. FIPS PUB 197 «ADVANCED ENCRYPTION STANDARD (AES)» [Интернет ресурс], ссылка <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>

# Листинг кода

файл CSPRNG.h

#ifndef CSPRNG\_H

#define CSPRNG\_H

#include <iostream>

#include <vector>

using namespace std;

class CSPRNG {

public:

vector<uint8\_t>\* GeneratePRN(uint64\_t PRNSizeInBytes);

};

#endif //CSPRNG\_H

файл CSPRNG.cpp

#include "CSPRNG.h"

#include "../AES256\_BlocksCipher/AES.h"

#include <chrono>

#include <random>

#include <vector>

vector<uint8\_t>\* CSPRNG::GeneratePRN(uint64\_t qwPRNSize){

//Create AES\_256 Instance and Set it in OFB mode

AES\_256 aes256Instance;

aes256Instance.SetEncryptionMode(2);

union FormattedGeneratorNumbers {

uint8\_t byArray[16];

uint64\_t qwArray[2];

};

FormattedGeneratorNumbers FormatData;

//Get current time in nanoseconds to mt19937\_64 Seed

auto current\_time\_now = chrono::high\_resolution\_clock::now();

mt19937\_64 urandom\_generator;

//Set Seed

urandom\_generator.seed(current\_time\_now.time\_since\_epoch().count());

//Generate PRN = Key = 32 byte (256 bit)

vector<uint8\_t>\* arrbyKey = new vector<uint8\_t>;

FormatData.qwArray[0] = urandom\_generator();

FormatData.qwArray[1] = urandom\_generator();

for (uint8\_t i : FormatData.byArray) { arrbyKey->push\_back(i); }

FormatData.qwArray[0] = urandom\_generator();

FormatData.qwArray[1] = urandom\_generator();

for (uint8\_t i : FormatData.byArray) { arrbyKey->push\_back(i); }

//Generate PlainText from mt19937\_64 generator

vector<uint8\_t>\* arrbyPlainText = new vector<uint8\_t>;

for (uint64\_t dwCurrentBlock = 0; dwCurrentBlock < qwPRNSize/16; dwCurrentBlock++) {

FormatData.qwArray[0] = urandom\_generator();

FormatData.qwArray[1] = urandom\_generator();

for (uint8\_t i : FormatData.byArray) { arrbyPlainText->push\_back(i); }

}

//If qwPRNSize % 16 != 0 add qwPRNSize bytes in arrbyPlainText

if (qwPRNSize % 16 != 0) {

FormatData.qwArray[0] = urandom\_generator();

FormatData.qwArray[1] = urandom\_generator();

for (uint8\_t i = 0; i < qwPRNSize % 16; i++) { arrbyPlainText->push\_back(FormatData.byArray[i]); }

}

vector<uint8\_t>\* arrbyPRN = aes256Instance.Encrypt(arrbyPlainText, arrbyKey);

//Delete unnecessary bytes

while (arrbyPRN->size() != qwPRNSize){ arrbyPRN->pop\_back(); }

delete arrbyPlainText;

delete arrbyKey;

return arrbyPRN;

};

# *Реализация криптографического протокола. Протокол защиты сетевого трафика TLS*

**Лабораторная работа №5**

Для выполнения лабораторной работы по реализации криптографического протокола TLS необходима реализация криптографических примитивов, которые используются при построении протокола. В лабораторных работах №1,2,3,4 были реализованы криптографические примитивы: блочный шифр, хеш-функция, цифровая подпись, генератор псевдослучайных чисел. Теперь, используя данные криптопримитивы в текущей лабораторной работе реализуем протокол TLS.

# Описание

TLS и SSL упоминаются в последнее время все чаще и чаще, более актуальным становится использование цифровых сертификатов, и даже появились компании, готовые бесплатно предоставлять цифровые сертификаты всем желающим, чтобы гарантировать шифрование трафика между посещаемыми сайтами и браузером клиента. Нужно это, естественно, для безопасности, чтобы никто в сети не мог получить данные, которые передаются от клиента серверу и обратно.

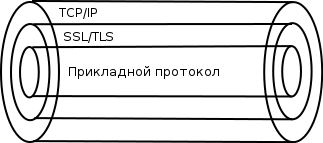
SSL — Secure Socket Layer, уровень защищенных сокетов. TLS — Transport Layer Security, безопасность транспортного уровня. SSL является более ранней системой, TLS появился позднее, он основан на спецификации SSL 3.0, разработанной компанией Netscape Communications. Тем не менее, задача у этих протоколов одна — обеспечение защищенной передачи данных между двумя компьютерами в сети Интернет.

Безопасная передача обеспечивается при помощи аутентификации и шифрования передаваемой информации. По сути эти протоколы, TLS и SSL, работают одинаково, принципиальных различий нет. TLS, можно сказать, является преемником SSL, хотя они и могут использоваться одновременно, причем даже на одном и том же сервере. Такая поддержка необходима для того, чтобы обеспечить работу как с новыми клиентами (устройствами и браузерами), так и с устаревшими, которые TLS не поддерживают. Последовательность возникновения этих протоколов выглядит вот так:

SSL 1.0 — никогда не публиковался  
 SSL 2.0 — февраль 1995 года  
 SSL 3.0 — 1996 год  
 TLS 1.0 — январь 1999 года  
 TLS 1.1 — апрель 2006 года  
 TLS 1.2 — август 2008 года

# Принцип работы протокола SSL/TLS

Принцип работы SSL/TLS следующий. Поверх протокола TCP/IP устанавливается зашифрованный канал, внутри которого передаются данные по прикладному протоколу — HTTP, FTP, и так далее. Вот как это можно представить графически:



Прикладной протокол «заворачивается» в TLS/SSL, а тот в свою очередь в TCP/IP. По сути данные по прикладному протоколу передаются по TCP/IP, но они зашифрованы. И расшифровать передаваемые данные могут только те машины, которые установили соединение. Для всех остальных, кто получит передаваемые пакеты, эта информация будет бессмысленной, если они не смогут ее расшифровать.

Установка соединения обеспечивается в несколько этапов:

1) Клиент устанавливает соединение с сервером и запрашивает защищенное подключение. Это может обеспечиваться либо установлением соединения на порт, который изначально предназначен для работы с SSL/TLS, например, 443.

2) При установке соединения клиент предоставляет список алгоритмов шифрования, которые он «знает». Сервер сверяет полученный список со списком алгоритмов, которые «знает» сам сервер, и выбирает наиболее надежный алгоритм, после чего сообщает клиенту, какой алгоритм использовать

3) Сервер отправляет клиенту свой цифровой сертификат, подписанный удостоверяющим центром, и открытый ключ сервера.

4) Клиент может связаться с сервером доверенного центра сертификации, который подписал сертификат сервера, и проверить, валиден ли сертификат сервера. Но может и не связываться. В браузерах обычно уже установлены корневые сертификаты центров сертификации, с которыми сверяют подписи серверных сертификатов.

5) Генерируется сеансовый ключ для защищенного соединения. Это делается следующим образом:

— Клиент генерирует случайную цифровую последовательность  
— Клиент шифрует ее открытым ключом сервера и посылает результат на сервер  
— Сервер расшифровывает полученную последовательность при помощи закрытого ключа

Учитывая, что алгоритм шифрования является асимметричным, расшифровать последовательность может только сервер. При использовании асимметричного шифрования используется два ключа — приватный и публичный. Публичным отправляемое сообщение шифруется, а приватным расшифровывается. Расшифровать сообщение, имея публичный, ключ нельзя.

6) В новой версии SSL/TLS протоколе TLS используются алгоритмы для выработки общего ключа для более быстрого симметричного шифрования.

7) Таким образом устанавливается зашифрованное соединение. Данные, передаваемые по нему, зашифровываются и расшифровываются до тех пор, пока соединение не будет разорвано.

# Реализация протокола на основе принципов SSL/TLS

Пусть в соединении участвуют три стороны:

* CA – Certificate Authority Server
* Alice – User1
* Bob – User2

Ставится задача: необходимо установить защищенное соединение между пользователями Alice и Bob.

В свою очередь CA выполняет следующие функции:

* получение письменных заявок (с указанием идентификационных данных) от пользователей
* подпись публичных ключей асимметричного шифрования пользователей 
* хранение хеш-значения  публичного ключа пользователя и его заявления

Пользователи имеют в распоряжении следующие алгоритмы:

* RSA – для установления начального соединения, где шифрование описывается как , а расшифрование 
* SHA512 – для вычисления хэш-значения параметров, где вычисление хеш-значения обозначается: 
* ECDHE – протокол для выработки симметричного ключа шифрования (на основе Elliptic Curve GOST256)
* ГПСЧ – для выработки случайных чисел (на основе AES256-OFB)
* ЦП (на основе Elliptic Curve GOST256)
* AES – для быстрого шифрования, где шифрование описывается как , а расшифрование 

Иллюстрация работы протокола выглядит следующим образом:



Вычисляет: 



Alice

Bob

CA

Имеет:

1. 

Используя RSA генерирует:

1. 

2. 

Имеет:

1. 

2. 

Имеет:

1. 

Используя RSA генерирует:

1. 

2. 

Шаг 1

Вычисляет:







Вычисляет:



если да, имеет: 

если нет, то подделка подписи ключа, повторяем шаги начиная с генерации ключевой пары

Вычисляет:





Вычисляет:



если да, имеет: 

если нет, то подделка подписи ключа, повторяем шаги начиная с генерации ключевой пары

Примечание: на «Шаг 1» CA получает письменные заявки с указанием идентификационных данных заявителя. Получение подписи  происходит в «письменном» порядке,  хранится в CA вместе с заявлением.

Шаг 2

Имеет:

1. 

2. 

3. 

4. 



Имеет:

1. 

2. 

3. 

4. 



Вычисляет:





если да, имеет: 

если нет, то идет подмена (при поиске злоумышленника, он – в базе CA, т.к.  производится на , а все данные об обратившихся пользователях хранятся в CA)

Вычисляет:





если да, имеет: 

если нет, то идет подмена (при поиске злоумышленника, он – в базе CA, т.к.  производится на , а все данные об обратившихся пользователях хранятся в CA)

Таким образом, после выполнения 2 шага пользователи обменялись публичными ключами. Теперь необходимо выработать общий сеансовый симметричный ключ для более быстрого обмена информацией. Для этого воспользуемся алгоритмом Диффи-Хеллмана на эллиптических кривых (ECDHE).

Имеет:

1. 

2. 

3. 

4. 

5. 

Вычисляет:



Вычисляет:



Имеет:

1. 

2. 

3. 

4. 

5. 

Шаг 3

Имеет:

1. 

2. 

3. 

Вычисляет:





Имеет:

1. 

2. 

3. 

Вычисляет:





Вычисляет:



Вычисляет:







Вычисляет:



Имеет:

1. 

2. 

3. 

4. 

5. 

6. *SessionKey*

7. 

Имеет:

1. 

2. 

3. 

4. 

5. 

6. *SessionKey*

7. 

Вычисляет:



если да, имеет: общий с Bob’ом симметричный *SessionKey*

если нет, то идет подмена *SessionKey*, повторить «Шаг 3»

если да, имеет: общий с Alice симметричный *SessionKey*

если нет, то идет подмена *SessionKey*, повторить «Шаг 3»

Шаг 4

Имеет:

1. 

2. 

Имеет:

1. 

2. 

Вычисляет:



Вычисляет:







Вычисляет:



Вычисляет:



если result = false, то:

*AnswerBob*

если result = true, то:



если result = false, то:

*AnswerAlice*

если result = true, то:







Вычисляет:



если да, то сообщение передалось успешно

если нет, то сообщение не передалось, повторить «Шаг 4»

Вычисляет:



если да, то сообщение передалось успешно

если нет, то сообщение не передалось, повторить «Шаг 4»

# Свойства, характеризующие безопасность протокола

Свойства, характеризующие безопасность протоколов:

1. Аутентификация (не широковещательная)

* G1 (аутентификация субъекта);
* G2 (аутентификация сообщения);
* G3 (защита от повтора)

1. Аутентификация при рассылке по многим адреса

* G4 (неявная скрытая аутентификация получателя);
* G5 (аутентификация источника)

1. Авторизация 3-ей доверенной стороной

* G6 (авторизация 3-ей доверенной стороной);

1. Свойства совместной генерации ключа

* G7 (аутентификация ключа);
* G8 (подтверждение правильности ключа);
* G9 (защита от чтения назад);
* G10 (формирование новых ключей);
* G11 (защита от возможности договориться о параметрах безопасности)

1. Конфиденциальность

* G12 (конфиденциальность)

1. Анонимность

* G13 (защита идентификатора от прослушивания);
* G14 (защита идентификатора от других участников)

1. Защита от отказа в обслуживании

* G15 (защита то DDoS);

1. Инвариантность

* G16 (инвариантность отправителя)

1. Невозможность отказа от ранее совершенных действий

* G17 (подотчетность);
* G18 (доказательство источника);
* G19 (доказательство получателя)

1. Временное свойство

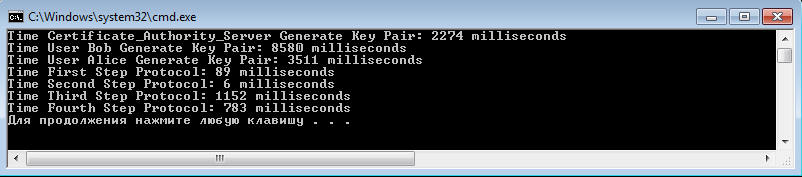
* G20 (безопасное временное свойство)

Данному протоколу присущи следующие свойства: G1, G2, G3, G6, G7, G8, G9, G10, G11, G12, G13, G14, G16, G17, G18, G19, G20.

# Результаты реализации протокола

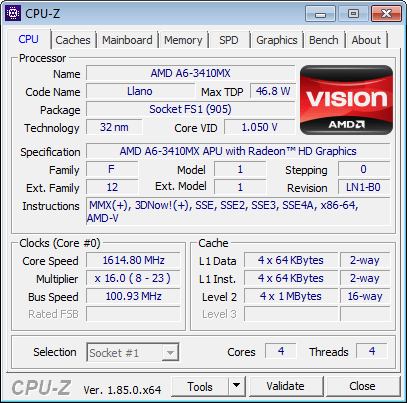
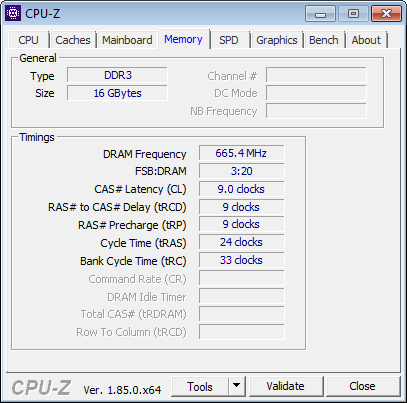
Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация протокола входит в главный проект CryptoProtocols решения CryptoProtocols. В главный проект были подключены проекты AES\_BlocksCipher, CSPRNG, ECDSA, SHA512\_Hash, которые собираются в подключаемые статические библиотеки и реализуют: блочный шифр, ГПСЧ, ЦП, Хеш-функцию. Реализация ассиметричного шифрования была взята из библиотеки OpenSSL.

Результат выполнения тест кейсов для проверки корректности работы протокола и фиксации времени выполнения для подсчета производительности работы приведены на Рис. 1.



**Рис. 1.** Результат тестирования реализованного протокола

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.2.

**Рис. 2.** ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

Таким образом, общее время установки соединения (без времени выработки ключей ассиметричного шифрования) составляет ~1,3 секунды, а обмена сообщениями ~ 0,8 секунды для каждой из сторон, что подтверждает быструю работы данного протокола.

**Табл. 1.** Скорость выполнения этапов протокола

|  |  |  |  |
| --- | --- | --- | --- |
| Пользователь | | Операция | Время выполнения [секунд] |
| *Генерация ассиметричной ключевой пары (выполняется до начала протокола)* | | | |
| CA | Выработка ключевой пары | | 2,274 |
| Alice | Выработка ключевой пары | | 3,511 |
| Bob | Выработка ключевой пары | | 8,580 |
| *Шаг 1 «Получение подписанных CA ключей пользователей»* | | | |
| Alice/Bob | Получение подписанных CA ключей пользователей | | 0,089 |
| *Шаг 2 «Обмен публичными ключами между собеседниками»* | | | |
| Alice/Bob | Обмен публичными ключами между собеседниками | | 0,006 |
| *Шаг 3 «Генерация сеансового ключа собеседников»* | | | |
| Alice/Bob | Генерация сеансового ключа собеседников | | 1,152 |
| *Шаг 4 «Обмен сообщениями между собеседниками»* | | | |
| Alice/Bob | Обмен сообщениями между собеседниками | | 0,783 |

# Литература

1. «Протокол Диффи — Хеллмана на эллиптических кривых» [Интернет ресурс], ссылка: <https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана_на_эллиптических_кривых>
2. «TLS и SSL: Необходимый минимум знаний» [Интернет ресурс], ссылка: <https://mnorin.com/tls-ssl-neobhodimy-j-minimum-znanij.html>
3. Никитин А.П., курс лекций «Криптографические протоколы», РТУ(МИРЭА), 2019г.

# Листинг кода

файл Protocol.h

#ifndef PROTOCOL\_H

#define PROTOCOL\_H

#include <iostream>

#include <openssl/rsa.h>

#include <openssl/pem.h>

#include <vector>

#include <string>

#include "../AES256\_BlocksCipher/AES.h"

#include "../SHA512\_Hash/SHA512.h"

#include "../CSPRNG/CSPRNG.h"

#include "../ECDSA/ECDSA.h"

using namespace std;

class IUser {

public:

ECDSA\_GOST\_256 ECDSA;

RSA\* UserRSA = nullptr;

BIO\* UserPublicKey = nullptr;

BIO\* UserPrivateKey = nullptr;

//ECDHE User's Parameters

bigint d;

pair<string, string> Q;

//Common Data

vector<uint8\_t>\* SessionKey;

string CorrectSessionKeyCheck;

//Paths

string strRSA\_Path\_UserPublicKey;

string strRSA\_Path\_UserPrivateKey;

string strRSA\_Signature\_UserPublicKey;

string strRSA\_Path\_CAPublicKey;

string strRSA\_Path\_FriendPublicKey;

string strRSA\_Signature\_FriendPublicKey;

string strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ;

string strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ;

string strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ;

string strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ;

string strECDHE\_Friend\_X\_CoordinatePublicPointQ;

string strECDHE\_Friend\_Y\_CoordinatePublicPointQ;

string strECDHE\_UserSessionKey;

string strECDHE\_UserCheckCorrectSessionKey;

string strECDHE\_FriendCheckCorrectSessionKey;

string strUser\_MessagePath;

string strUser\_EncryptionMessagePath;

string strUser\_Parametr\_R\_DigitalSignMessagePath;

string strUser\_Parametr\_S\_DigitalSignMessagePath;

string strUser\_X\_KeyCheckDigitalSignMessagePath;

string strUser\_Y\_KeyCheckDigitalSignMessagePath;

string strUser\_AnswerPath;

string strFriend\_EncryptionMessagePath;

string strFriend\_DecryptionMessagePath;

string strFriend\_Parametr\_R\_DigitalSignMessagePath;

string strFriend\_Parametr\_S\_DigitalSignMessagePath;

string strFriend\_X\_KeyCheckDigitalSignMessagePath;

string strFriend\_Y\_KeyCheckDigitalSignMessagePath;

string strFriend\_AnswerPath;

void GenerateKeyPair();

bool checkSignHashPublicKey(string& Path\_SignHashUserPublicKey, string& Path\_UserPublicKey);

vector<uint8\_t>\* GetFileHash(string& strPath\_File);

void Send\_SignHashUserPublicKey\_UserPublicKey(IUser& toUser);

void Calculate\_ECDHE\_Parametrs();

void Send\_PublicUserEllipticCurvePointQ(IUser& toUser);

void CalculateSymmetricSessionKey();

void Send\_CheckCorrectSessionKey(IUser& toUser);

bool checkCorrectSessionKey();

void CreateMessage();

void SendMessage(IUser& toUser);

void CheckMessage\_CreateAnswer();

void SendAnswer(IUser& toUser);

bool CheckAnswer();

string hexStr(vector<uint8\_t> \*data);

IUser(

string& \_strRSA\_Path\_UserPublicKey,

string& \_strRSA\_Path\_UserPrivateKey,

string& \_strRSA\_Signature\_UserPublicKey,

string& \_strRSA\_Path\_CAPublicKey,

string& \_strRSA\_Path\_FriendPublicKey,

string& \_strRSA\_Signature\_FriendPublicKey,

string& \_strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ,

string& \_strECDHE\_Friend\_X\_CoordinatePublicPointQ,

string& \_strECDHE\_Friend\_Y\_CoordinatePublicPointQ,

string& \_strECDHE\_UserSessionKey,

string& \_strECDHE\_UserCheckCorrectSessionKey,

string& \_strECDHE\_FriendCheckCorrectSessionKey,

string& \_strUser\_MessagePath,

string& \_strUser\_EncryptionMessagePath,

string& \_strUser\_Parametr\_R\_DigitalSignMessagePath,

string& \_strUser\_Parametr\_S\_DigitalSignMessagePath,

string& \_strUser\_X\_KeyCheckDigitalSignMessagePath,

string& \_strUser\_Y\_KeyCheckDigitalSignMessagePath,

string& \_strUser\_AnswerPath,

string& \_strFriend\_EncryptionMessagePath,

string& \_strFriend\_DecryptionMessagePath,

string& \_strFriend\_Parametr\_R\_DigitalSignMessagePath,

string& \_strFriend\_Parametr\_S\_DigitalSignMessagePath,

string& \_strFriend\_X\_KeyCheckDigitalSignMessagePath,

string& \_strFriend\_Y\_KeyCheckDigitalSignMessagePath,

string& \_strFriend\_AnswerPath

);

~IUser();

};

class User\_Alice : public IUser {

public:

User\_Alice();

~User\_Alice();

};

class User\_Bob : public IUser {

public:

User\_Bob();

~User\_Bob();

};

class Certificate\_Authority\_Server {

RSA\* CAServerRSA = nullptr;

BIO\* CAServerPublicKey = nullptr;

BIO\* CAServerPrivateKey = nullptr;

public:

void GenerateKeyPair();

Certificate\_Authority\_Server();

~Certificate\_Authority\_Server();

void signatureHashPublicKey(vector<uint8\_t>\* HashPublicKey, const IUser& User);

};

#endif//PROTOCOL\_H

файл Protocol.cpp

#include "Protocol.h"

#include <openssl/bn.h>

#include <fstream>

const uint8\_t hexmap[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };

User\_Alice::User\_Alice() : IUser(

string("protocol/ALICE/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/ALICE/RSA/PRIVATE\_KEY/PRIVATE\_KEY.pem"),

string("protocol/ALICE/FROM/CA\_SERVER/RSA/SIGN\_ALICE\_PUBLIC\_KEY.sig"),

string("protocol/ALICE/FROM/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/ALICE/FROM/BOB/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/ALICE/FROM/BOB/RSA/SIGN\_BOB\_PUBLIC\_KEY.sig"),

string("protocol/ALICE/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qx.cor"),

string("protocol/ALICE/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qy.cor"),

string("protocol/ALICE/FROM/BOB/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qx.cor"),

string("protocol/ALICE/FROM/BOB/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qy.cor"),

string("protocol/ALICE/FROM/BOB/ECDHE/PUBLIC POINT/Qx.cor"),

string("protocol/ALICE/FROM/BOB/ECDHE/PUBLIC POINT/Qy.cor"),

string("protocol/ALICE/ECDHE/SESSION KEY/SECRET.KEY"),

string("protocol/ALICE/ECDHE/CHECKER CORRECT/CHECK.SIG"),

string("protocol/ALICE/FROM/BOB/ECDHE/CHECKER CORRECT/CHECK.SIG"),

string("protocol/ALICE/MESSAGE/MESSAGE.TXT"),

string("protocol/ALICE/MESSAGE/EncMESSAGE.TXT"),

string("protocol/ALICE/MESSAGE/R\_DIGITALSIGN.TXT"),

string("protocol/ALICE/MESSAGE/S\_DIGITALSIGN.TXT"),

string("protocol/ALICE/MESSAGE/X\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/ALICE/MESSAGE/Y\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/ALICE/MESSAGE/ANSWER.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/EncMESSAGE.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/MESSAGE.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/R\_DIGITALSIGN.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/S\_DIGITALSIGN.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/X\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/Y\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/ALICE/FROM/BOB/MESSAGE/ANSWER.TXT")

) { };

User\_Alice::~User\_Alice() { };

User\_Bob::User\_Bob() : IUser(

string("protocol/BOB/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/BOB/RSA/PRIVATE\_KEY/PRIVATE\_KEY.pem"),

string("protocol/BOB/FROM/CA\_SERVER/RSA/SIGN\_BOB\_PUBLIC\_KEY.sig"),

string("protocol/BOB/FROM/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/BOB/FROM/ALICE/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem"),

string("protocol/BOB/FROM/ALICE/RSA/SIGN\_ALICE\_PUBLIC\_KEY.sig"),

string("protocol/BOB/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qx.cor"),

string("protocol/BOB/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qy.cor"),

string("protocol/BOB/FROM/ALICE/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qx.cor"),

string("protocol/BOB/FROM/ALICE/RSA/ECDHE/PUBLIC POINT/EncRSA\_Qy.cor"),

string("protocol/BOB/FROM/ALICE/ECDHE/PUBLIC POINT/Qx.cor"),

string("protocol/BOB/FROM/ALICE/ECDHE/PUBLIC POINT/Qy.cor"),

string("protocol/BOB/ECDHE/SESSION KEY/SECRET.KEY"),

string("protocol/BOB/ECDHE/CHECKER CORRECT/CHECK.SIG"),

string("protocol/BOB/FROM/ALICE/ECDHE/CHECKER CORRECT/CHECK.SIG"),

string("protocol/BOB/MESSAGE/MESSAGE.TXT"),

string("protocol/BOB/MESSAGE/EncMESSAGE.TXT"),

string("protocol/BOB/MESSAGE/R\_DIGITALSIGN.TXT"),

string("protocol/BOB/MESSAGE/S\_DIGITALSIGN.TXT"),

string("protocol/BOB/MESSAGE/X\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/BOB/MESSAGE/Y\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/BOB/MESSAGE/ANSWER.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/EncMESSAGE.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/MESSAGE.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/R\_DIGITALSIGN.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/S\_DIGITALSIGN.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/X\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/Y\_KEYCHECK\_DIGITALSIGN.TXT"),

string("protocol/BOB/FROM/ALICE/MESSAGE/ANSWER.TXT")

) { };

User\_Bob::~User\_Bob() { };

void IUser::GenerateKeyPair(){

BIGNUM \*e = BN\_new();

BN\_set\_word(e, RSA\_F4);

UserRSA = RSA\_new();

RSA\_generate\_key\_ex(UserRSA, 4096, e, nullptr);

UserPublicKey = BIO\_new\_file(strRSA\_Path\_UserPublicKey.data(), "wb");

PEM\_write\_bio\_RSAPublicKey(UserPublicKey, UserRSA);

BIO\_free\_all(UserPublicKey);

UserPrivateKey = BIO\_new\_file(strRSA\_Path\_UserPrivateKey.data(), "wb");

PEM\_write\_bio\_RSAPrivateKey(UserPrivateKey, UserRSA, nullptr, nullptr, 0, nullptr, nullptr);

BIO\_free\_all(UserPrivateKey);

BN\_free(e);

RSA\_free(UserRSA);

}

bool IUser::checkSignHashPublicKey(string& Path\_SignHashUserPublicKey, string& Path\_UserPublicKey) {

auto HashFunction = new AlgorithmSHA512::SHA512;

fstream FileInput;

//Read User's PublicKey

FileInput.open(Path\_UserPublicKey, ios\_base::in | ios\_base::binary);

//Get HashUserPublicKey

auto UserPublicKey = new vector<uint8\_t>();

while (FileInput.peek() != -1) { UserPublicKey->push\_back(FileInput.get()); }

auto HashUserPublicKey = HashFunction->GetHash(UserPublicKey);

FileInput.close();

//Read User's SignHashUserPublicKey

FileInput.open(Path\_SignHashUserPublicKey, ios\_base::in | ios\_base::binary);

//Get SignHashUserPublicKey

auto SignHashUserPublicKey = new vector<uint8\_t>();

while (FileInput.peek() != -1) { SignHashUserPublicKey->push\_back(FileInput.get()); }

FileInput.close();

RSA\* checkPubKeyCARSA = RSA\_new();

BIO\* CAServerPublicKey = BIO\_new\_file(strRSA\_Path\_CAPublicKey.data(), "rb");

checkPubKeyCARSA = PEM\_read\_bio\_RSAPublicKey(CAServerPublicKey, &checkPubKeyCARSA, nullptr, nullptr);

auto decryptHashUserPubKey = new vector<uint8\_t>(HashUserPublicKey->size(), 0);

RSA\_public\_decrypt(SignHashUserPublicKey->size(), SignHashUserPublicKey->data(), decryptHashUserPubKey->data(), checkPubKeyCARSA, RSA\_PKCS1\_PADDING);

//Check, that RSA\_Decrypt(CAPublicKey, SignHashUserPublicKey) == HashUserPublicKey

if (string(decryptHashUserPubKey->begin(), decryptHashUserPubKey->end()) != string(HashUserPublicKey->begin(), HashUserPublicKey->end())) {

RSA\_free(checkPubKeyCARSA);

delete HashFunction;

delete UserPublicKey;

delete HashUserPublicKey;

delete SignHashUserPublicKey;

delete decryptHashUserPubKey;

BIO\_free\_all(CAServerPublicKey);

return false;

}

RSA\_free(checkPubKeyCARSA);

delete HashFunction;

delete UserPublicKey;

delete HashUserPublicKey;

delete SignHashUserPublicKey;

delete decryptHashUserPubKey;

BIO\_free\_all(CAServerPublicKey);

return true;

};

vector<uint8\_t>\* IUser::GetFileHash(string& strPath\_File) {

auto HashFunction = new AlgorithmSHA512::SHA512;

fstream FileInput;

//CA sign User's PublicKey

FileInput.open(strRSA\_Path\_UserPublicKey, ios\_base::in | ios\_base::binary);

auto UserPublicKey = new vector<uint8\_t>();

while (FileInput.peek() != -1) { UserPublicKey->push\_back(FileInput.get()); }

auto HashPublicKey = HashFunction->GetHash(UserPublicKey);

FileInput.close();

delete HashFunction;

delete UserPublicKey;

return HashPublicKey;

};

void IUser::Send\_SignHashUserPublicKey\_UserPublicKey(IUser& User){

fstream FileInput;

fstream FileOutput;

//Send PublicKey

FileInput.open(strRSA\_Path\_UserPublicKey, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strRSA\_Path\_FriendPublicKey, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send SignHashUserPublicKey

FileInput.open(strRSA\_Signature\_UserPublicKey, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strRSA\_Signature\_FriendPublicKey, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

};

void IUser::Calculate\_ECDHE\_Parametrs(){

//Generate Pseudo Random Number

CSPRNG generatorPRN;

vector<uint8\_t>\* PRN = generatorPRN.GeneratePRN(1024);

//Get Secret Key for Elliptic Curve

d.FromString(hexStr(PRN), 16);

//Get Public Elliptic Curve Point

Q = ECDSA.MultiplyOnBasePoint(d);

//RSA Encryption Public Elliptic Curve Point Q

RSA\* RSAEncryptPublicEllipticCurveCoordinates = RSA\_new();

BIO\* FriendPublicKey = BIO\_new\_file(strRSA\_Path\_FriendPublicKey.data(), "rb");

RSAEncryptPublicEllipticCurveCoordinates = PEM\_read\_bio\_RSAPublicKey(FriendPublicKey, &RSAEncryptPublicEllipticCurveCoordinates, nullptr, nullptr);

//Encryption X coordinate

auto Encryption\_X\_Coordinate = new vector<uint8\_t>(RSA\_size(RSAEncryptPublicEllipticCurveCoordinates), 0);

vector<uint8\_t> X\_Coordinate(Q.first.begin(), Q.first.end());

RSA\_public\_encrypt(X\_Coordinate.size(), X\_Coordinate.data(), Encryption\_X\_Coordinate->data(), RSAEncryptPublicEllipticCurveCoordinates, RSA\_PKCS1\_PADDING);

//Encryption Y coordinate

auto Encryption\_Y\_Coordinate = new vector<uint8\_t>(RSA\_size(RSAEncryptPublicEllipticCurveCoordinates), 0);

vector<uint8\_t> Y\_Coordinate(Q.second.begin(), Q.second.end());

RSA\_public\_encrypt(Y\_Coordinate.size(), Y\_Coordinate.data(), Encryption\_Y\_Coordinate->data(), RSAEncryptPublicEllipticCurveCoordinates, RSA\_PKCS1\_PADDING);

//Write Result to Files

fstream FileOutput;

FileOutput.open(strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < Encryption\_X\_Coordinate->size(); i++) FileOutput.put((\*Encryption\_X\_Coordinate)[i]);

FileOutput.close();

FileOutput.open(strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < Encryption\_Y\_Coordinate->size(); i++) FileOutput.put((\*Encryption\_Y\_Coordinate)[i]);

FileOutput.close();

RSA\_free(RSAEncryptPublicEllipticCurveCoordinates);

BIO\_free\_all(FriendPublicKey);

delete PRN;

delete Encryption\_X\_Coordinate;

delete Encryption\_Y\_Coordinate;

};

void IUser::Send\_PublicUserEllipticCurvePointQ(IUser& User){

fstream FileInput;

fstream FileOutput;

//Send RSA Encryption X Coordinate of Elliptic Curve Point Q to User

FileInput.open(strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send RSA Encryption Y Coordinate of Elliptic Curve Point Q to User

FileInput.open(strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

}

void IUser::CalculateSymmetricSessionKey(){

//RSA Decryption Friend's Public Point Q

RSA\* RSADecryptFriendPublicEllipticCurveCoordinates = RSA\_new();

BIO\* UserPrivateKey = BIO\_new\_file(strRSA\_Path\_UserPrivateKey.data(), "rb");

RSADecryptFriendPublicEllipticCurveCoordinates = PEM\_read\_bio\_RSAPrivateKey(UserPrivateKey, &RSADecryptFriendPublicEllipticCurveCoordinates, nullptr, nullptr);

//Read Friend's X Coordinate

fstream FileInput;

FileInput.open(strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ, ios\_base::in | ios\_base::binary);

//Get Friend's X Coordinate

vector<uint8\_t> EncryptionFriend\_X\_Coordinate;

while (FileInput.peek() != -1) { EncryptionFriend\_X\_Coordinate.push\_back(FileInput.get()); }

FileInput.close();

//Decrypt X Coordinate

auto DecryptionFriend\_X\_Coordinate = new vector<uint8\_t>(RSA\_size(RSADecryptFriendPublicEllipticCurveCoordinates), 0);

auto X\_CoordinateSize = RSA\_private\_decrypt(EncryptionFriend\_X\_Coordinate.size(), EncryptionFriend\_X\_Coordinate.data(), DecryptionFriend\_X\_Coordinate->data(), RSADecryptFriendPublicEllipticCurveCoordinates, RSA\_PKCS1\_PADDING);

EncryptionFriend\_X\_Coordinate.clear();

//Read Friend's Y Coordinate

FileInput.open(strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ, ios\_base::in | ios\_base::binary);

//Get Friend's Y Coordinate

vector<uint8\_t> EncryptionFriend\_Y\_Coordinate;

while (FileInput.peek() != -1) { EncryptionFriend\_Y\_Coordinate.push\_back(FileInput.get()); }

FileInput.close();

//Decrypt Y Coordinate

auto DecryptionFriend\_Y\_Coordinate = new vector<uint8\_t>(RSA\_size(RSADecryptFriendPublicEllipticCurveCoordinates), 0);

auto Y\_CoordinateSize = RSA\_private\_decrypt(EncryptionFriend\_Y\_Coordinate.size(), EncryptionFriend\_Y\_Coordinate.data(), DecryptionFriend\_Y\_Coordinate->data(), RSADecryptFriendPublicEllipticCurveCoordinates, RSA\_PKCS1\_PADDING);

EncryptionFriend\_Y\_Coordinate.clear();

//Write Result to Files

fstream FileOutput;

//Write X\_Coordinate

FileOutput.open(strECDHE\_Friend\_X\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < X\_CoordinateSize; i++) FileOutput.put((\*DecryptionFriend\_X\_Coordinate)[i]);

FileOutput.close();

//Write Y\_Coordinate

FileOutput.open(strECDHE\_Friend\_Y\_CoordinatePublicPointQ, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < Y\_CoordinateSize; i++) FileOutput.put((\*DecryptionFriend\_Y\_Coordinate)[i]);

FileOutput.close();

RSA\_free(RSADecryptFriendPublicEllipticCurveCoordinates);

BIO\_free\_all(UserPrivateKey);

//Calculate Common Session Key

ECPoint FriendQ(&ECDSA);

FriendQ.setCoordinate(

string(DecryptionFriend\_X\_Coordinate->begin(), DecryptionFriend\_X\_Coordinate->begin() + X\_CoordinateSize),

string(DecryptionFriend\_Y\_Coordinate->begin(), DecryptionFriend\_Y\_Coordinate->begin() + Y\_CoordinateSize)

);

ECPoint SecretPoint = FriendQ\*d;

auto X\_CoordinateSecretPoint = SecretPoint.getXCoordinate();

//Get Hash of X Coordinate of Secret Point

auto HashFunction = new AlgorithmSHA512::SHA512;

auto X\_CoordinateSecretPointHash = HashFunction->GetHash(&vector<uint8\_t>(X\_CoordinateSecretPoint.begin(), X\_CoordinateSecretPoint.end()));

SessionKey = new vector<uint8\_t>(X\_CoordinateSecretPointHash->begin(), X\_CoordinateSecretPointHash->begin() + 32);

//Get Y Coordinate to Check Correct SessionKey

CorrectSessionKeyCheck = SecretPoint.getYCoordinate();

//Write Session Key

FileOutput.open(strECDHE\_UserSessionKey, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < SessionKey->size(); i++) FileOutput.put((\*SessionKey)[i]);

FileOutput.close();

delete SessionKey;

delete X\_CoordinateSecretPointHash;

delete HashFunction;

delete DecryptionFriend\_X\_Coordinate;

delete DecryptionFriend\_Y\_Coordinate;

}

void IUser::Send\_CheckCorrectSessionKey(IUser& User){

//Get Hash of string 'CorrectSessionKeyCheck'

auto HashFunction = new AlgorithmSHA512::SHA512;

auto CheckCorrectSessionKeyHash = HashFunction->GetHash(&vector<uint8\_t>(CorrectSessionKeyCheck.begin(), CorrectSessionKeyCheck.end()));

//AES256 with Session Key Encryption 'CheckCorrectSessionKeyHash'

AES\_256 AES;

AES.SetEncryptionMode(1);

//Read Session Key

SessionKey = new vector<uint8\_t>;

fstream FileInput;

FileInput.open(strECDHE\_UserSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { SessionKey->push\_back(FileInput.get()); }

FileInput.close();

//Encryption 'CheckCorrectSessionKeyHash'

auto EncryptionCheckCorrectSessionKeyHash = AES.Encrypt(CheckCorrectSessionKeyHash, SessionKey);

//Write Result to File

fstream FileOutput;

//Write EncryptionCheckCorrectSessionKeyHash

FileOutput.open(strECDHE\_UserCheckCorrectSessionKey, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < EncryptionCheckCorrectSessionKeyHash->size(); i++) FileOutput.put((\*EncryptionCheckCorrectSessionKeyHash)[i]);

FileOutput.close();

delete HashFunction;

delete SessionKey;

delete CheckCorrectSessionKeyHash;

delete EncryptionCheckCorrectSessionKeyHash;

//Send EncryptionCheckCorrectSessionKeyHash to User

FileInput.open(strECDHE\_UserCheckCorrectSessionKey, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strECDHE\_FriendCheckCorrectSessionKey, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

}

bool IUser::checkCorrectSessionKey(){

//Read Session Key

SessionKey = new vector<uint8\_t>;

fstream FileInput;

FileInput.open(strECDHE\_UserSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { SessionKey->push\_back(FileInput.get()); }

FileInput.close();

//Read Friend EncryptionCheckCorrectSessionKeyHash

auto EncryptionFriendCheckCorrectSessionKeyHash = new vector<uint8\_t>;

FileInput.open(strECDHE\_FriendCheckCorrectSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { EncryptionFriendCheckCorrectSessionKeyHash->push\_back(FileInput.get()); }

FileInput.close();

//Decryption Friend EncryptionCheckCorrectSessionKeyHash

AES\_256 AES;

AES.SetEncryptionMode(1);

auto DecryptionFriendCheckCorrectSessionKeyHash = AES.Decrypt(EncryptionFriendCheckCorrectSessionKeyHash, SessionKey);

delete SessionKey;

delete EncryptionFriendCheckCorrectSessionKeyHash;

//Get Hash of string 'CorrectSessionKeyCheck'

auto HashFunction = new AlgorithmSHA512::SHA512;

auto CheckCorrectSessionKeyHash = HashFunction->GetHash(&vector<uint8\_t>(CorrectSessionKeyCheck.begin(), CorrectSessionKeyCheck.end()));

//Check, that CheckCorrectSessionKeyHash == DecryptionFriendCheckCorrectSessionKeyHash

//If equal => return true

//Else => return false

for (uint32\_t i = 0; i < CheckCorrectSessionKeyHash->size(); i++) {

if ((\*CheckCorrectSessionKeyHash)[i] == (\*DecryptionFriendCheckCorrectSessionKeyHash)[i]) { continue; }

else { return false; }

}

return true;

}

void IUser::CreateMessage(){

//Generate Pseudo Random Number

CSPRNG generatorPRN;

auto PRN = generatorPRN.GeneratePRN(1024);

//Get Secret Key for Elliptic Curve

bigint SecretKeyDS;

SecretKeyDS.FromString(hexStr(PRN), 16);

//CreateKeyCheckDigitalSign

pair<string, string> KeyCheckDigitalSign = ECDSA.CreateKeyCheckDigitalSign(SecretKeyDS.ToString());

//Write X\_KeyCheckDigitalSign

fstream FileOutput;

FileOutput.open(strUser\_X\_KeyCheckDigitalSignMessagePath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < KeyCheckDigitalSign.first.size(); i++) FileOutput.put(KeyCheckDigitalSign.first[i]);

FileOutput.close();

//Write Y\_KeyCheckDigitalSign

FileOutput.open(strUser\_Y\_KeyCheckDigitalSignMessagePath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < KeyCheckDigitalSign.second.size(); i++) FileOutput.put(KeyCheckDigitalSign.second[i]);

FileOutput.close();

//Read Message

auto arrbyMessage = new vector<uint8\_t>;

fstream FileInput;

FileInput.open(strUser\_MessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { arrbyMessage->push\_back(FileInput.get()); }

FileInput.close();

//CreateDigitalSign

pair<string, string> DigitalSign = ECDSA.CreateDigitalSign(SecretKeyDS.ToString(), string(arrbyMessage->begin(), arrbyMessage->end()));

//Write R

FileOutput.open(strUser\_Parametr\_R\_DigitalSignMessagePath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < DigitalSign.first.size(); i++) FileOutput.put(DigitalSign.first[i]);

FileOutput.close();

//Write S

FileOutput.open(strUser\_Parametr\_S\_DigitalSignMessagePath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < DigitalSign.second.size(); i++) FileOutput.put(DigitalSign.second[i]);

FileOutput.close();

//AES with CTR mode

AES\_256 AES;

AES.SetEncryptionMode(1);

//Read Session Key

SessionKey = new vector<uint8\_t>;

FileInput.open(strECDHE\_UserSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { SessionKey->push\_back(FileInput.get()); }

FileInput.close();

//Save Message Size

union FormatedWriteMessageSize {

uint64\_t qwMessageSize;

uint8\_t byArrMessageSize[sizeof(uint64\_t)];

};

FormatedWriteMessageSize qwFormat;

qwFormat.qwMessageSize = arrbyMessage->size();

//Encryption Message

auto EncryptionMessage = AES.Encrypt(arrbyMessage, SessionKey);

//Write EncryptionMessage

FileOutput.open(strUser\_EncryptionMessagePath, ios\_base::out | ios\_base::binary);

//Write MessageSize

for (unsigned char i : qwFormat.byArrMessageSize) { FileOutput << i; }

//Write Message

for (uint32\_t i = 0; i < EncryptionMessage->size(); i++) FileOutput.put((\*EncryptionMessage)[i]);

FileOutput.close();

delete PRN;

delete arrbyMessage;

delete SessionKey;

delete EncryptionMessage;

}

void IUser::SendMessage(IUser& User){

fstream FileInput;

fstream FileOutput;

//Send X\_KeyCheckDigitalSign to User

FileInput.open(strUser\_X\_KeyCheckDigitalSignMessagePath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_X\_KeyCheckDigitalSignMessagePath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send Y\_KeyCheckDigitalSign to User

FileInput.open(strUser\_Y\_KeyCheckDigitalSignMessagePath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_Y\_KeyCheckDigitalSignMessagePath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send R to User

FileInput.open(strUser\_Parametr\_R\_DigitalSignMessagePath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_Parametr\_R\_DigitalSignMessagePath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send S to User

FileInput.open(strUser\_Parametr\_S\_DigitalSignMessagePath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_Parametr\_S\_DigitalSignMessagePath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

//Send EncryptionMessage to User

FileInput.open(strUser\_EncryptionMessagePath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_EncryptionMessagePath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

}

void IUser::CheckMessage\_CreateAnswer(){

//AES with CTR mode

AES\_256 AES;

AES.SetEncryptionMode(1);

//Read Session Key

fstream FileInput;

SessionKey = new vector<uint8\_t>;

FileInput.open(strECDHE\_UserSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { SessionKey->push\_back(FileInput.get()); }

FileInput.close();

//Read EncryptionMessage

auto EncryptionMessage = new vector<uint8\_t>;

FileInput.open(strFriend\_EncryptionMessagePath, ios\_base::in | ios\_base::binary);

//Read Message Size

union FormatedWriteMessageSize {

uint64\_t qwMessageSize;

uint8\_t byArrMessageSize[sizeof(uint64\_t)];

};

FormatedWriteMessageSize qwFormat;

for (unsigned char & i : qwFormat.byArrMessageSize) { i = FileInput.get(); }

uint64\_t MessageSize = qwFormat.qwMessageSize;

//Read Message

while (FileInput.peek() != -1) { EncryptionMessage->push\_back(FileInput.get()); }

FileInput.close();

//Decryption Message

auto DecryptionMessage = AES.Decrypt(EncryptionMessage, SessionKey);

string Message(DecryptionMessage->begin(), DecryptionMessage->begin() + MessageSize);

//Write Message

fstream FileOutput;

FileOutput.open(strFriend\_DecryptionMessagePath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < Message.size(); i++) FileOutput.put(Message[i]);

FileOutput.close();

//Read X\_KeyCheckDigitalSign

string X\_KeyCheckDigitalSign;

FileInput.open(strFriend\_X\_KeyCheckDigitalSignMessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { X\_KeyCheckDigitalSign.push\_back(FileInput.get()); }

FileInput.close();

//Read Y\_KeyCheckDigitalSign

string Y\_KeyCheckDigitalSign;

FileInput.open(strFriend\_Y\_KeyCheckDigitalSignMessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { Y\_KeyCheckDigitalSign.push\_back(FileInput.get()); }

FileInput.close();

//Create KeyCheckDigitalSign

pair<string, string> KeyCheckDigitalSign(X\_KeyCheckDigitalSign, Y\_KeyCheckDigitalSign);

//Read R

string R;

FileInput.open(strFriend\_Parametr\_R\_DigitalSignMessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { R.push\_back(FileInput.get()); }

FileInput.close();

//Read S

string S;

FileInput.open(strFriend\_Parametr\_S\_DigitalSignMessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { S.push\_back(FileInput.get()); }

FileInput.close();

//CreateDigitalSign

pair<string, string> DigitalSign(R, S);

//CheckDigitalSign

bool result = ECDSA.CheckDigitalSign(DigitalSign, Message, KeyCheckDigitalSign);

vector<uint8\_t>\* Answer;

if (result) {

//Get Message Hash

AlgorithmSHA512::SHA512 HashFunction;

Answer = AES.Encrypt(HashFunction.GetHash(&vector<uint8\_t>(Message.begin(), Message.end())), SessionKey);

}

else {

//Generate Pseudo Random Number

CSPRNG generatorPRN;

Answer = generatorPRN.GeneratePRN(80);

}

//Write Answer

FileOutput.open(strUser\_AnswerPath, ios\_base::out | ios\_base::binary);

for (uint32\_t i = 0; i < Answer->size(); i++) FileOutput.put((\*Answer)[i]);

FileOutput.close();

delete SessionKey;

delete EncryptionMessage;

delete DecryptionMessage;

delete Answer;

}

void IUser::SendAnswer(IUser& User){

fstream FileInput;

fstream FileOutput;

//Send Answer to User

FileInput.open(strUser\_AnswerPath, ios\_base::in | ios\_base::binary);

FileOutput.open(User.strFriend\_AnswerPath, ios\_base::out | ios\_base::binary);

while (FileInput.peek() != -1) { FileOutput.put(FileInput.get()); }

FileInput.close();

FileOutput.close();

}

bool IUser::CheckAnswer(){

//AES with CTR mode

AES\_256 AES;

AES.SetEncryptionMode(1);

//Read Session Key

fstream FileInput;

SessionKey = new vector<uint8\_t>;

FileInput.open(strECDHE\_UserSessionKey, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { SessionKey->push\_back(FileInput.get()); }

FileInput.close();

//Read Answer

auto Answer = new vector<uint8\_t>;

FileInput.open(strFriend\_AnswerPath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { Answer->push\_back(FileInput.get()); }

FileInput.close();

//Decrypt Answer

auto AnswerMessageHash = AES.Decrypt(Answer, SessionKey);

//Read Message

auto arrbyMessage = new vector<uint8\_t>;

FileInput.open(strUser\_MessagePath, ios\_base::in | ios\_base::binary);

while (FileInput.peek() != -1) { arrbyMessage->push\_back(FileInput.get()); }

FileInput.close();

//Get MessageHash

AlgorithmSHA512::SHA512 HashFunction;

auto MessageHash = HashFunction.GetHash(&vector<uint8\_t>(arrbyMessage->begin(), arrbyMessage->end()));

//Check, that MessageHash == AnswerMessageHash

//If equal => return true

//Else => return false

for (uint32\_t i = 0; i < MessageHash->size(); i++) {

if ((\*MessageHash)[i] == (\*AnswerMessageHash)[i]) { continue; }

else { return false; }

}

return true;

}

string IUser::hexStr(vector<uint8\_t>\* data) {

string s(data->size() \* 2, ' ');

for (register uint64\_t i = 0; i < data->size(); ++i) {

s[2 \* i] = hexmap[((\*data)[i] & 0xF0) >> 4];

s[2 \* i + 1] = hexmap[(\*data)[i] & 0x0F];

}

return s;

};

IUser::IUser(

string& \_strRSA\_Path\_UserPublicKey,

string& \_strRSA\_Path\_UserPrivateKey,

string& \_strRSA\_Signature\_UserPublicKey,

string& \_strRSA\_Path\_CAPublicKey,

string& \_strRSA\_Path\_FriendPublicKey,

string& \_strRSA\_Signature\_FriendPublicKey,

string& \_strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ,

string& \_strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ,

string& \_strECDHE\_Friend\_X\_CoordinatePublicPointQ,

string& \_strECDHE\_Friend\_Y\_CoordinatePublicPointQ,

string& \_strECDHE\_UserSessionKey,

string& \_strECDHE\_UserCheckCorrectSessionKey,

string& \_strECDHE\_FriendCheckCorrectSessionKey,

string& \_strUser\_MessagePath,

string& \_strUser\_EncryptionMessagePath,

string& \_strUser\_Parametr\_R\_DigitalSignMessagePath,

string& \_strUser\_Parametr\_S\_DigitalSignMessagePath,

string& \_strUser\_X\_KeyCheckDigitalSignMessagePath,

string& \_strUser\_Y\_KeyCheckDigitalSignMessagePath,

string& \_strUser\_AnswerPath,

string& \_strFriend\_EncryptionMessagePath,

string& \_strFriend\_DecryptionMessagePath,

string& \_strFriend\_Parametr\_R\_DigitalSignMessagePath,

string& \_strFriend\_Parametr\_S\_DigitalSignMessagePath,

string& \_strFriend\_X\_KeyCheckDigitalSignMessagePath,

string& \_strFriend\_Y\_KeyCheckDigitalSignMessagePath,

string& \_strFriend\_AnswerPath

) {

strRSA\_Path\_UserPublicKey = \_strRSA\_Path\_UserPublicKey;

strRSA\_Path\_UserPrivateKey = \_strRSA\_Path\_UserPrivateKey;

strRSA\_Signature\_UserPublicKey = \_strRSA\_Signature\_UserPublicKey;

strRSA\_Path\_CAPublicKey = \_strRSA\_Path\_CAPublicKey;

strRSA\_Path\_FriendPublicKey = \_strRSA\_Path\_FriendPublicKey;

strRSA\_Signature\_FriendPublicKey = \_strRSA\_Signature\_FriendPublicKey;

strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ = \_strRSA\_ECDHE\_User\_X\_CoordinatePublicPointQ;

strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ = \_strRSA\_ECDHE\_User\_Y\_CoordinatePublicPointQ;

strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ = \_strRSA\_ECDHE\_Friend\_X\_CoordinatePublicPointQ;

strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ = \_strRSA\_ECDHE\_Friend\_Y\_CoordinatePublicPointQ;

strECDHE\_Friend\_X\_CoordinatePublicPointQ = \_strECDHE\_Friend\_X\_CoordinatePublicPointQ;

strECDHE\_Friend\_Y\_CoordinatePublicPointQ = \_strECDHE\_Friend\_Y\_CoordinatePublicPointQ;

strECDHE\_UserSessionKey = \_strECDHE\_UserSessionKey;

strECDHE\_UserCheckCorrectSessionKey = \_strECDHE\_UserCheckCorrectSessionKey;

strECDHE\_FriendCheckCorrectSessionKey = \_strECDHE\_FriendCheckCorrectSessionKey;

strUser\_MessagePath = \_strUser\_MessagePath;

strUser\_EncryptionMessagePath = \_strUser\_EncryptionMessagePath;

strUser\_Parametr\_R\_DigitalSignMessagePath = \_strUser\_Parametr\_R\_DigitalSignMessagePath;

strUser\_Parametr\_S\_DigitalSignMessagePath = \_strUser\_Parametr\_S\_DigitalSignMessagePath;

strUser\_X\_KeyCheckDigitalSignMessagePath = \_strUser\_X\_KeyCheckDigitalSignMessagePath;

strUser\_Y\_KeyCheckDigitalSignMessagePath = \_strUser\_Y\_KeyCheckDigitalSignMessagePath;

strUser\_AnswerPath = \_strUser\_AnswerPath;

strFriend\_EncryptionMessagePath = \_strFriend\_EncryptionMessagePath;

strFriend\_DecryptionMessagePath = \_strFriend\_DecryptionMessagePath;

strFriend\_Parametr\_R\_DigitalSignMessagePath = \_strFriend\_Parametr\_R\_DigitalSignMessagePath;

strFriend\_Parametr\_S\_DigitalSignMessagePath = \_strFriend\_Parametr\_S\_DigitalSignMessagePath;

strFriend\_X\_KeyCheckDigitalSignMessagePath = \_strFriend\_X\_KeyCheckDigitalSignMessagePath;

strFriend\_Y\_KeyCheckDigitalSignMessagePath = \_strFriend\_Y\_KeyCheckDigitalSignMessagePath;

strFriend\_AnswerPath = \_strFriend\_AnswerPath;

};

IUser::~IUser() { };

void Certificate\_Authority\_Server::GenerateKeyPair(){

BIGNUM \*e = BN\_new();

BN\_set\_word(e, RSA\_F4);

CAServerRSA = RSA\_new();

RSA\_generate\_key\_ex(CAServerRSA, 4096, e, nullptr);

CAServerPublicKey = BIO\_new\_file("protocol/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem", "wb");

PEM\_write\_bio\_RSAPublicKey(CAServerPublicKey, CAServerRSA);

BIO\_free\_all(CAServerPublicKey);

CAServerPublicKey = BIO\_new\_file("protocol/ALICE/FROM/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem", "wb");

PEM\_write\_bio\_RSAPublicKey(CAServerPublicKey, CAServerRSA);

BIO\_free\_all(CAServerPublicKey);

CAServerPublicKey = BIO\_new\_file("protocol/BOB/FROM/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem", "wb");

PEM\_write\_bio\_RSAPublicKey(CAServerPublicKey, CAServerRSA);

BIO\_free\_all(CAServerPublicKey);

CAServerPrivateKey = BIO\_new\_file("protocol/CA\_SERVER/RSA/PRIVATE\_KEY/PRIVATE\_KEY.pem", "wb");

PEM\_write\_bio\_RSAPrivateKey(CAServerPrivateKey, CAServerRSA, nullptr, nullptr, 0, nullptr, nullptr);

BIO\_free\_all(CAServerPrivateKey);

BN\_free(e);

RSA\_free(CAServerRSA);

}

Certificate\_Authority\_Server::Certificate\_Authority\_Server(){ };

Certificate\_Authority\_Server::~Certificate\_Authority\_Server(){ };

void Certificate\_Authority\_Server::signatureHashPublicKey(vector<uint8\_t>\* HashPublicKey, const IUser& User){

CAServerRSA = RSA\_new();

CAServerPrivateKey = BIO\_new\_file("protocol/CA\_SERVER/RSA/PRIVATE\_KEY/PRIVATE\_KEY.pem", "rb");

CAServerPublicKey = BIO\_new\_file("protocol/CA\_SERVER/RSA/PUBLIC\_KEY/PUBLIC\_KEY.pem", "rb");

CAServerRSA = PEM\_read\_bio\_RSAPrivateKey(CAServerPrivateKey, &CAServerRSA, nullptr, nullptr);

CAServerRSA = PEM\_read\_bio\_RSAPublicKey(CAServerPublicKey, &CAServerRSA, nullptr, nullptr);

auto signHashUserPubKey = new vector<uint8\_t>(RSA\_size(CAServerRSA), 0);

RSA\_private\_encrypt(HashPublicKey->size(), HashPublicKey->data(), signHashUserPubKey->data(), CAServerRSA, RSA\_PKCS1\_PADDING);

//CAServer send to User Message with SIGN\_User\_PUBLIC\_KEY.sig

fstream FileOutput;

FileOutput.open(User.strRSA\_Signature\_UserPublicKey, ios\_base::out | ios\_base::binary);

for (unsigned char i : \*signHashUserPubKey) { FileOutput.put(i); }

delete HashPublicKey;

delete signHashUserPubKey;

FileOutput.close();

BIO\_free\_all(CAServerPrivateKey);

BIO\_free\_all(CAServerPublicKey);

RSA\_free(CAServerRSA);

};

файл CryptoProtocols.cpp

// CryptoProtocols.cpp : Defines the entry point for the console application.

//

#include "Protocol.h"

#include <chrono>

void ProtocolFirstStep(Certificate\_Authority\_Server& CA, IUser& User) {

//User send to CAServer his HashUserPublicKey for Sign

//CAServer send SignHashUserPublicKey to User

CA.signatureHashPublicKey(User.GetFileHash(User.strRSA\_Path\_UserPublicKey), User);

//Check, that RSA\_Decrypt(CAPublicKey, SignHashUserPublicKey) == HashUserPublicKey

//If true, then User have true SignHashUserPublicKey for install signal

//Else false, then Man in the Middle listen signal channel, repeat Step 1 ProtocolFirstStep()

if (!User.checkSignHashPublicKey(User.strRSA\_Signature\_UserPublicKey, User.strRSA\_Path\_UserPublicKey)) { cout << false << endl; ProtocolFirstStep(CA, User); }

}

void ProtocolSecondStep(IUser& Sender, IUser& Receiver) {

//Sender Send to Receiver SignHashSenderPublicKey and SenderPublicKey

Sender.Send\_SignHashUserPublicKey\_UserPublicKey(Receiver);

//Receiver Check, that RSA\_Decrypt(CAPublicKey, SignHashSenderPublicKey) == HashSenderPublicKey

//If true, then Receiver have true SenderPublicKey for install signal

//Else false, then Man in the Middle listen signal channel, repeat Step 2 ProtocolSecondStep()

if (!Receiver.checkSignHashPublicKey(Receiver.strRSA\_Signature\_FriendPublicKey, Receiver.strRSA\_Path\_FriendPublicKey)) { cout << false << endl; ProtocolSecondStep(Sender, Receiver); }

}

void ProtocolThirdStep(IUser& User1, IUser& User2) {

//User1 Calculate ECDHA Parameters

// 1. Generate PseudoRandom Number and get from it the Private Key 'd' for CurvePoint (size of PseudoRandom Number is 1024 bytes) and ( 0 < d < q)

// 2. Getting Public EllipticCurve Point Q = d\*G, where G - base Point of EllipticCurve

// 3. Encryption RSA EllipticCurve Point 'Q' Coordinates 'x' and 'y' [RSAenc(ReceiverPubKey, Q) = EncQ]

User1.Calculate\_ECDHE\_Parametrs();

//User2 Calculate ECDHA Parameters

// 1. Generate PseudoRandom Number and get from it the Private Key 'd' for CurvePoint (size of PseudoRandom Number is 1024 bytes) and ( 0 < d < q)

// 2. Getting Public EllipticCurve Point Q = d\*G, where G - base Point of EllipticCurve

// 3. Encryption RSA EllipticCurve Point 'Q' Coordinates 'x' and 'y' [RSAenc(ReceiverPubKey, Q) = EncQ]

User2.Calculate\_ECDHE\_Parametrs();

//User1 Send to User2 yours 'EncQ'

User1.Send\_PublicUserEllipticCurvePointQ(User2);

//User2 Send to User1 yours 'EncQ'

User2.Send\_PublicUserEllipticCurvePointQ(User1);

//User1 Calculate Common Symmetric Session Key and Information of Correct SessionKeyCheck

User1.CalculateSymmetricSessionKey();

//User2 Calculate Common Symmetric Session Key and Information of Correct SessionKeyCheck

User2.CalculateSymmetricSessionKey();

//User1 Send to User2 Encryption Hash of Information of Correct SessionKeyCheck

User1.Send\_CheckCorrectSessionKey(User2);

//User2 Send to User1 Encryption Hash of Information of Correct SessionKeyCheck

User2.Send\_CheckCorrectSessionKey(User1);

//Check, that User1CheckCorrectSessionKeyHash == DecryptionUser2CheckCorrectSessionKeyHash

//If equal => User1 Session Key = User2 Session Key

//Else => Man in the Middle, repeat Protocol Third Step

if (!User1.checkCorrectSessionKey()) { cout << false << endl; ProtocolThirdStep(User1, User2); }

//Check, that User2CheckCorrectSessionKeyHash == DecryptionUser1CheckCorrectSessionKeyHash

//If equal => User2 Session Key = User1 Session Key

//Else => Man in the Middle, repeat Protocol Third Step

if (!User2.checkCorrectSessionKey()) { cout << false << endl; ProtocolThirdStep(User2, User1); }

}

void ProtocolFourthStep(IUser& Sender, IUser& Receiver) {

//Sender Create Message for Receiver

Sender.CreateMessage();

//Sender Send Message to Receiver

Sender.SendMessage(Receiver);

//Receiver Check Massage Digital Sign and then Create Answer

Receiver.CheckMessage\_CreateAnswer();

//Receiver send Answer to Sender

Receiver.SendAnswer(Sender);

//Sender Check Answer if true - send is ok, if false - send is fail - repeat FourthStep

if (!Sender.CheckAnswer()) { cout << false << endl; ProtocolFourthStep(Sender, Receiver); }

}

int main()

{

using myclock = chrono::steady\_clock;

Certificate\_Authority\_Server CA;

myclock::time\_point start = myclock::now();

CA.GenerateKeyPair();

myclock::time\_point end = myclock::now();

cout << "Time Certificate\_Authority\_Server Generate Key Pair: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

User\_Bob Bob;

start = myclock::now();

Bob.GenerateKeyPair();

end = myclock::now();

cout << "Time User Bob Generate Key Pair: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

User\_Alice Alice;

start = myclock::now();

Alice.GenerateKeyPair();

end = myclock::now();

cout << "Time User Alice Generate Key Pair: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

ProtocolFirstStep(CA, Alice);

start = myclock::now();

ProtocolFirstStep(CA, Bob);

end = myclock::now();

cout <<"Time First Step Protocol: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

start = myclock::now();

ProtocolSecondStep(Alice, Bob);

end = myclock::now();

cout << "Time Second Step Protocol: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

ProtocolSecondStep(Bob, Alice);

start = myclock::now();

ProtocolThirdStep(Alice, Bob);

end = myclock::now();

cout << "Time Third Step Protocol: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

ProtocolFourthStep(Alice, Bob);

start = myclock::now();

ProtocolFourthStep(Bob, Alice);

end = myclock::now();

cout << "Time Fourth Step Protocol: " << chrono::duration\_cast<chrono::milliseconds>(end - start).count() << " milliseconds" << endl;

return 0;

}

1. Поясним понятие «выглядит случайной». Понятно, что последовательность, сгенерированная детерминированным алгоритмом, не является случайной. Однако цель алгоритма в том, чтобы взять некоторую маленькую последовательность истинно случайных чисел и использовать её для генерации длинной последовательности, не отличимой от истинно случайной последовательности чисел той же длины. Убедиться в том, что последовательность чисел случайна (или не случайна) можно либо при помощи статистических тестов, выявляющих специфические особенности случайных последовательностей, либо аналитико-вычислительными методами. [↑](#footnote-ref-1)
2. То есть время выполнения теста ограничено сверху значением полинома, вычисленного от длины *l* выходной последовательности. Полиномиальным алгоритмом или алгоритмом полиномиальной временной сложности называется алгоритм, у которого временная сложность равна O(p(n)), где p(n) - некоторая полиномиальная функция, а n - входная длина. [↑](#footnote-ref-2)
3. Источники энтропии используются для накопления энтропии, с последующим получением из неё начального значения. Под энтропией понимают меру, определяющую «неопределенность», то есть то, насколько полученная из системы информация говорит о «неизвестности» работы самой системы выработки последовательности. [↑](#footnote-ref-3)