



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
1. Описание	5
2. Терминология	5
3. Шифрование	6
3.1 Преобразование SubBytes	7
3.2 Преобразование ShiftRows	8
3.3 Преобразование MixColumns	8
3.4 Преобразование AddRoundKey	9
3.5 Процедура KeyExpansion	9
4. Расшифрование	10
4.1 Преобразование InvShiftRows	11
4.2 Преобразование InvSubBytes	12
4.3 Преобразование InvMixColumns	12
5. Режимы шифрования	12
5.1 Режим ECB (Electronic Code Book)	13
5.2 Режим OFB (Output Feed Back)	13
5.3 Режим CTR (Counter)	14
6. Дополнение некрatных блоков (Padding)	15
7. Результаты реализации алгоритма AES 128/192/256	15
Литература	20
Листинг кода	20
II. Реализация хеш-функции. Хеш-функция SHA-512	45
1. Описание	45
2. Основные операции	45
3. Функции и константы	46
3.1 Функции	46
3.2 Константы	46
4. Подготовка к вычислению хеш-значения	46
4.1 Дополнение сообщения	46
4.2 Получение сообщения	47

4.3	Настройка инициализации начальных хеш-значений $H^{(0)}$	47
5.	Хеш-функция SHA-512	47
5.1	Подготовка к алгоритму SHA-512	48
5.2	Вычисление хеш-значения сообщения по алгоритму SHA-512	48
6.	Результаты реализации алгоритма SHA-512.....	50
	Литература	51
	Листинг кода	51
III.	<i>Реализация цифровой подписи. Алгоритм ЦП ECDSA</i>	57
1.	Описание	57
2.	Рекомендации NIST по выбору эллиптических кривых	58
3.	Эллиптические кривые над простыми полями $GF(p)$	58
4.	Математические операции над эллиптическими кривыми	61
5.	Параметры пользователя	62
6.	Формирование цифровой подписи.....	62
7.	Проверка цифровой подписи	63
8.	Результаты реализации алгоритма ECDSA	63
	Литература	66
	Листинг кода	66
IV.	<i>Реализация ГПСЧ. Алгоритм CSPRNG AES256_OFB</i>	78
1.	Основные понятия о ГПСЧ	78
2.	Виды ГПСЧ.....	79
3.	Пример реализации ГПСЧ	81
4.	ГПСЧ на основе AES256_OFB	84
5.	Результаты реализации ГПСЧ AES256_OFB	85
	Литература	89
	Листинг кода	89
V.	<i>Реализация криптографического протокола. Протокол защиты сетевого трафика TLS</i>	91
1.	Описание	91

2. Принцип работы протокола SSL/TLS	92
3. Реализация протокола на основе принципов SSL/TLS	93
4. Свойства, характеризующие безопасность протокола.....	99
5. Результаты реализации протокола	100
Литература	101
Листинг кода	101

I. 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.

1. 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).

2. 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 $GF(2^8)$, то есть байту $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ соответствует многочлен $\sum_{i=0}^7 b_i \cdot x^i$ в поле $GF(2^8)$.

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

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

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 .

Nk	Nr
4	10
6	12
8	14

3. 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 $\text{MixColumns}(\text{AddRoundKey}(A, B)) = \text{AddRoundKey}(\text{MixColumns}(A), \text{MixColumns}(B))$.

Шифрование производится по алгоритму, приведенному на Рис. 1.

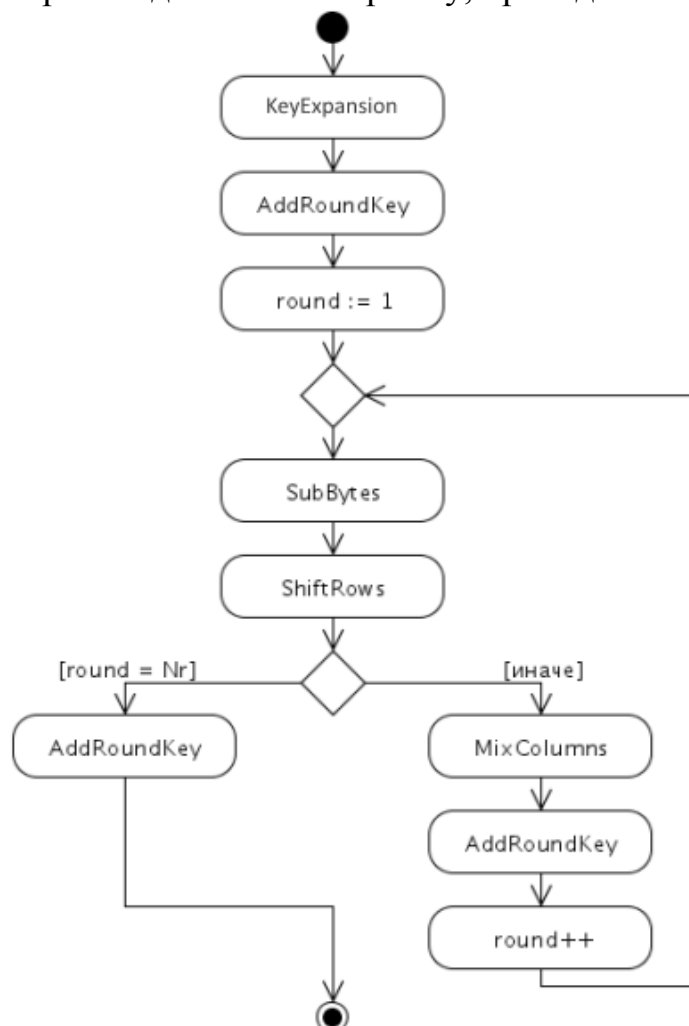


Рис. 1. Алгоритм шифрования

3.1 Преобразование SubBytes

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

Табл. 3. Таблица подстановок

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

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

3.2 Преобразование ShiftRows

Преобразование ShiftRows заключается в циклическом сдвиге влево строк формы. Преобразование схематично представлено на Рис. 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.

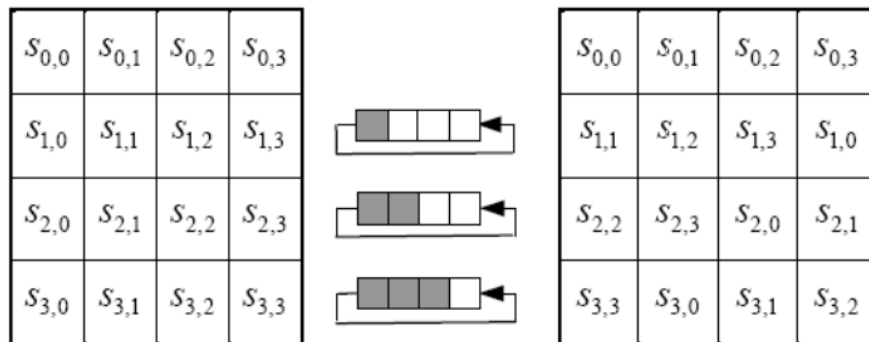


Рис. 2. Преобразование ShiftRows

3.3 Преобразование MixColumns

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

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Умножение производится в поле Галуа $GF(2^8)$.

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

Для реализации быстрого произведения в поле были использованы готовые таблицы, заменяющие умножения для всех элементов из $GF(2^8)$ на элементы (0x02 и 0x03) на подстановку [ресурс].

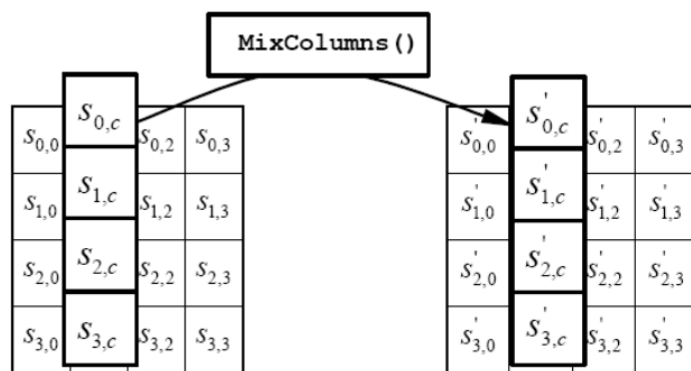


Рис. 3. Преобразование MixColumns

3.4 Преобразование AddRoundKey

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

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round * Nb + c}]$$

Здесь w_i — это столбцы ключа. Над каждым столбцом операция производится отдельно, как показано на Рис. 4.

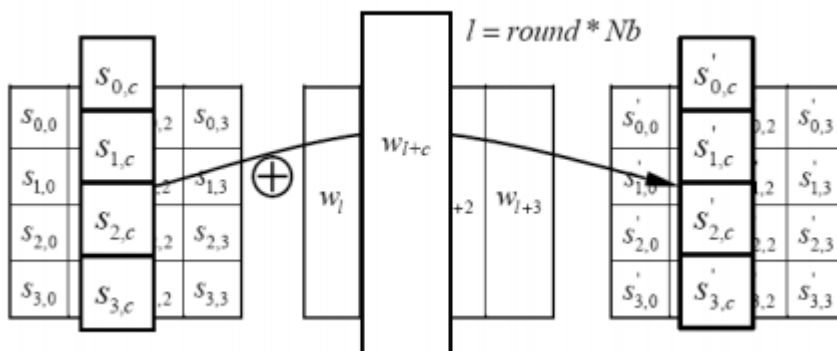


Рис. 4. Преобразование AddRoundKey

3.5 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 Рис. 5.

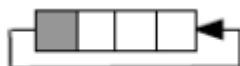


Рис. 5. Процедура RotWord

$Rcon(i)$ формирует слово $[02^{i-1}, 00, 00, 00]$.

4. 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;

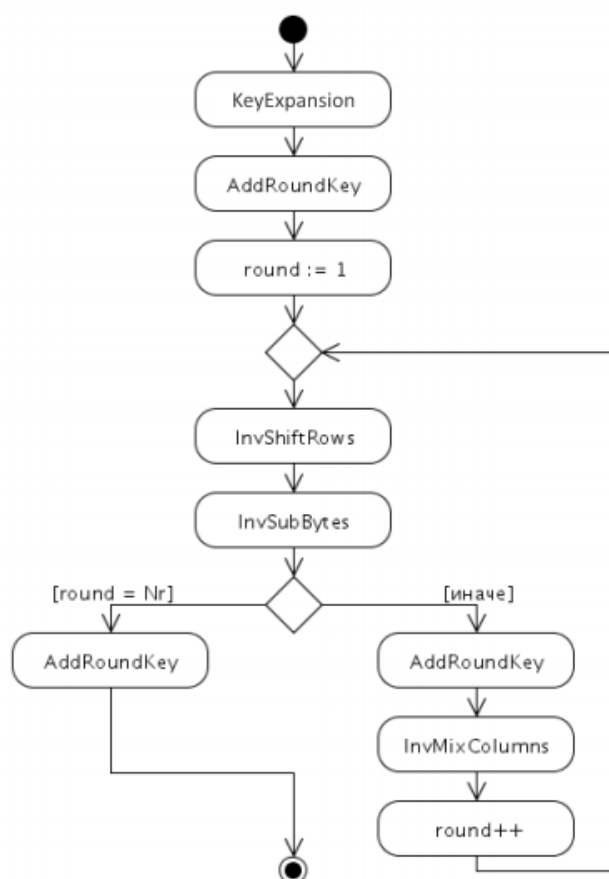


Рис. 5. Алгоритм расшифрования

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

4.1 Преобразование InvShiftRows

Это преобразование обратное 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 Рис. 7.

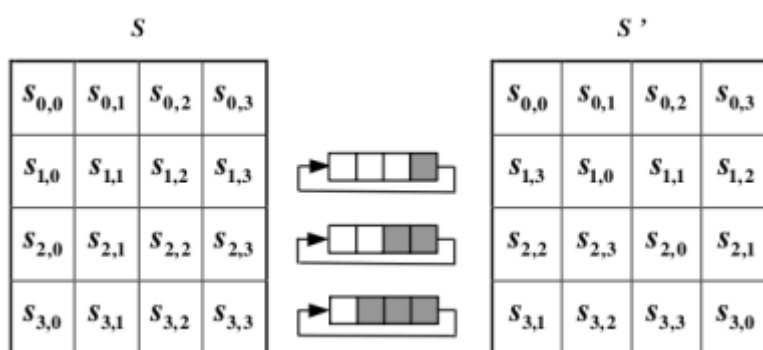


Рис. 7. Процедура InvShiftRows

4.2 Преобразование InvSubBytes

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

Табл. 4. Обратная таблица подстановок

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

4.3 Преобразование InvMixColumns

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

$$\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Здесь умножение также производится в поле Галуа $GF(2^8)$.

Для реализации быстрого произведения в поле были использованы готовые таблицы, заменяющие умножения (для всех элементов из $GF(2^8)$ на элементы $0x09$ $0x0b$ и $0x0d$) на подстановку [\[ресурс\]](#).

5. Режимы шифрования

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

5.1 Режим ECB (Electronic Code Book)

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

Шифрование

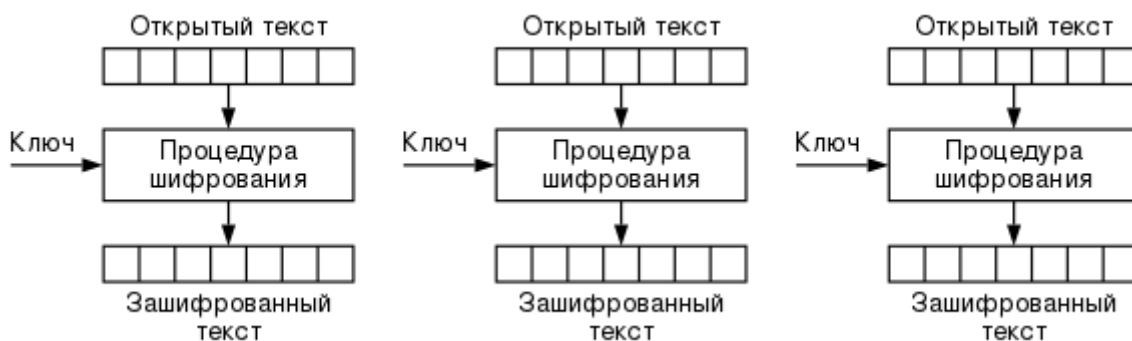


Рис. 8. Режим ECB

Расшифрование происходит по аналогичной схеме.

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

5.2 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 Рис. 9 и Рис. 10.

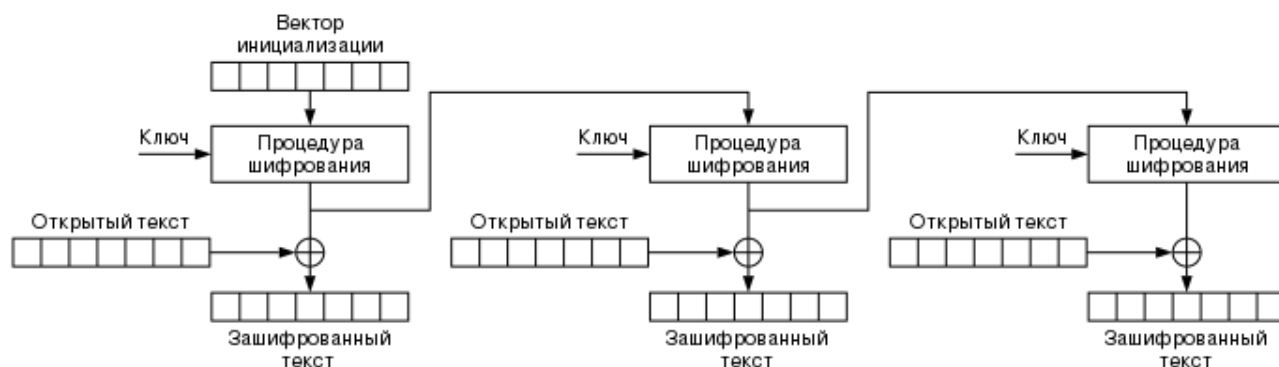


Рис. 9. Шифрование в режиме OFB

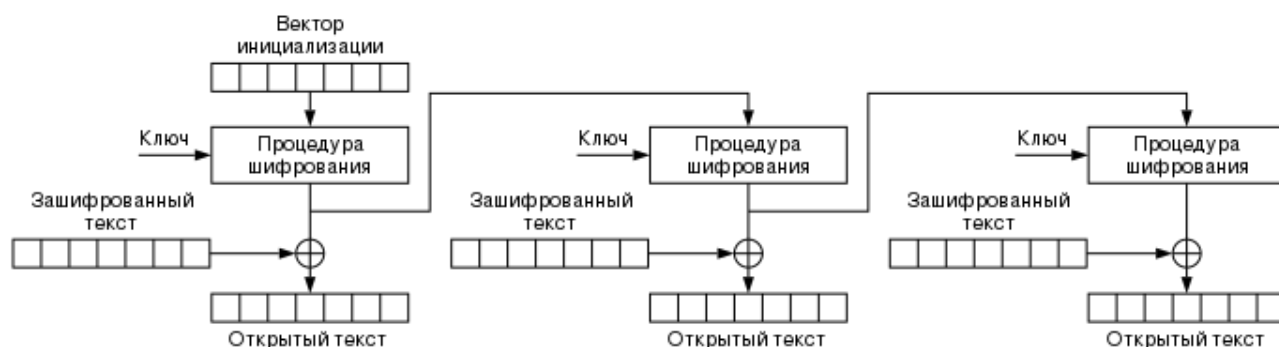


Рис. 10. Расшифрование в режиме OFB

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

5.3 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 Рис. 11 и Рис. 12.

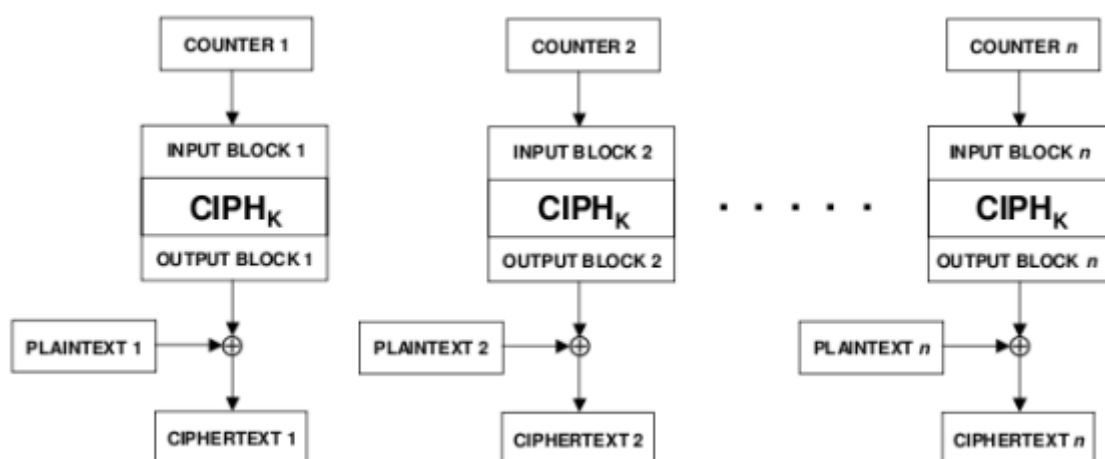


Рис. 11. Шифрование в режиме CTR

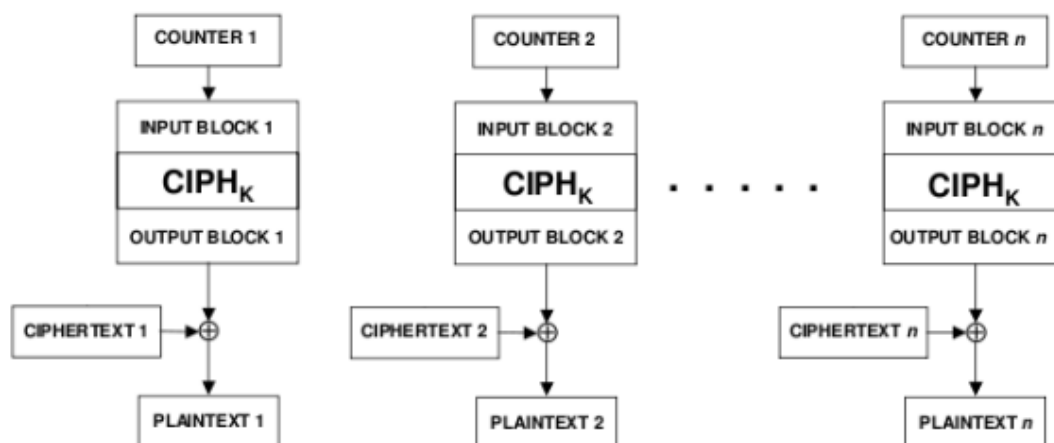


Рис. 12. Расшифрование в режиме CTR

Режим CTR допускает параллельное шифрование или расшифрование нескольких блоков.

6. Дополнение некрatных блоков (Padding)

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

Алгоритм дополнения некрatных блоков взят из ГОСТ 34.13-2015 «Режимы работы блочных шифров» из раздела 4.1 «Дополнение сообщения» процедуру №2. Пусть $|P| \equiv r \pmod{L}$. Положим $P^* = P \parallel 1 \parallel 0^{L-r-1}$. 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 .

7. 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.

```

C:\Windows\system32\cmd.exe
[=====] Running 9 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 9 tests from TestAES
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_ECB
Encrypt_10Mbyte_AES128_ECB time: 2362112 microseconds
Decrypt_10Mbyte_AES128_ECB time: 2293512 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_ECB <4894 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_ECB
Encrypt_10Mbyte_AES192_ECB time: 2611189 microseconds
Decrypt_10Mbyte_AES192_ECB time: 2663997 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_ECB <5460 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_ECB
Encrypt_10Mbyte_AES256_ECB time: 2931885 microseconds
Decrypt_10Mbyte_AES256_ECB time: 3075020 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_ECB <6201 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_CTR
Encrypt_10Mbyte_AES128_CTR time: 2220445 microseconds
Decrypt_10Mbyte_AES128_CTR time: 2201475 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_CTR <4605 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_CTR
Encrypt_10Mbyte_AES192_CTR time: 2623564 microseconds
Decrypt_10Mbyte_AES192_CTR time: 2592855 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_CTR <5401 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_CTR
Encrypt_10Mbyte_AES256_CTR time: 3000422 microseconds
Decrypt_10Mbyte_AES256_CTR time: 2932036 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_CTR <6157 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_OFB
Encrypt_10Mbyte_AES128_OFB time: 8070140 microseconds
Decrypt_10Mbyte_AES128_OFB time: 8061170 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES128_OFB <16313 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_OFB
Encrypt_10Mbyte_AES192_OFB time: 9461984 microseconds
Decrypt_10Mbyte_AES192_OFB time: 9392092 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES192_OFB <19047 ms>
[ RUN      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_OFB
Encrypt_10Mbyte_AES256_OFB time: 10764897 microseconds
Decrypt_10Mbyte_AES256_OFB time: 10731304 microseconds
[ OK      ] TestAES.CorrectWorkEncryptAndDecrypt10MbyteAES256_OFB <21679 ms>
[-----] 9 tests from TestAES <89768 ms total>
[-----] Global test environment tear-down
[=====] 9 tests from 1 test case ran. <89773 ms total>
[ PASSED  ] 9 tests.
Для продолжения нажмите любую клавишу . . .

```

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


```

C:\Windows\system32\cmd.exe
[=====] Running 9 tests from 1 test case.
[=====] Global test environment set-up.
[=====] 9 tests from TestAES
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES128_ECB
Encrypt_100Mbyte_AES128_ECB time: 20797865 microseconds
Decrypt_100Mbyte_AES128_ECB time: 20830836 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES128_ECB <42663 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES192_ECB
Encrypt_100Mbyte_AES192_ECB time: 24542410 microseconds
Decrypt_100Mbyte_AES192_ECB time: 25440722 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES192_ECB <51052 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES256_ECB
Encrypt_100Mbyte_AES256_ECB time: 27683561 microseconds
Decrypt_100Mbyte_AES256_ECB time: 27887668 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES256_ECB <56690 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES128_CTR
Encrypt_100Mbyte_AES128_CTR time: 20655715 microseconds
Decrypt_100Mbyte_AES128_CTR time: 21372212 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES128_CTR <43045 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES192_CTR
Encrypt_100Mbyte_AES192_CTR time: 23975197 microseconds
Decrypt_100Mbyte_AES192_CTR time: 24239790 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES192_CTR <49361 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES256_CTR
Encrypt_100Mbyte_AES256_CTR time: 28615597 microseconds
Decrypt_100Mbyte_AES256_CTR time: 27590111 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES256_CTR <57342 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES128_OFB
Encrypt_100Mbyte_AES128_OFB time: 75057715 microseconds
Decrypt_100Mbyte_AES128_OFB time: 65519863 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES128_OFB <141515 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES192_OFB
Encrypt_100Mbyte_AES192_OFB time: 72864841 microseconds
Decrypt_100Mbyte_AES192_OFB time: 72898175 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES192_OFB <146704 ms>
[ RUN ] TestAES.CorrectWorkEncryptAndDecryptAES256_OFB
Encrypt_100Mbyte_AES256_OFB time: 82347599 microseconds
Decrypt_100Mbyte_AES256_OFB time: 82394100 microseconds
[ OK ] TestAES.CorrectWorkEncryptAndDecryptAES256_OFB <165692 ms>
[=====] 9 tests from TestAES <754072 ms total>

[=====] Global test environment tear-down
[=====] 9 tests from 1 test case ran. <754078 ms total>
[ PASSED ] 9 tests.
Для продолжения нажмите любую клавишу . . .

```

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.

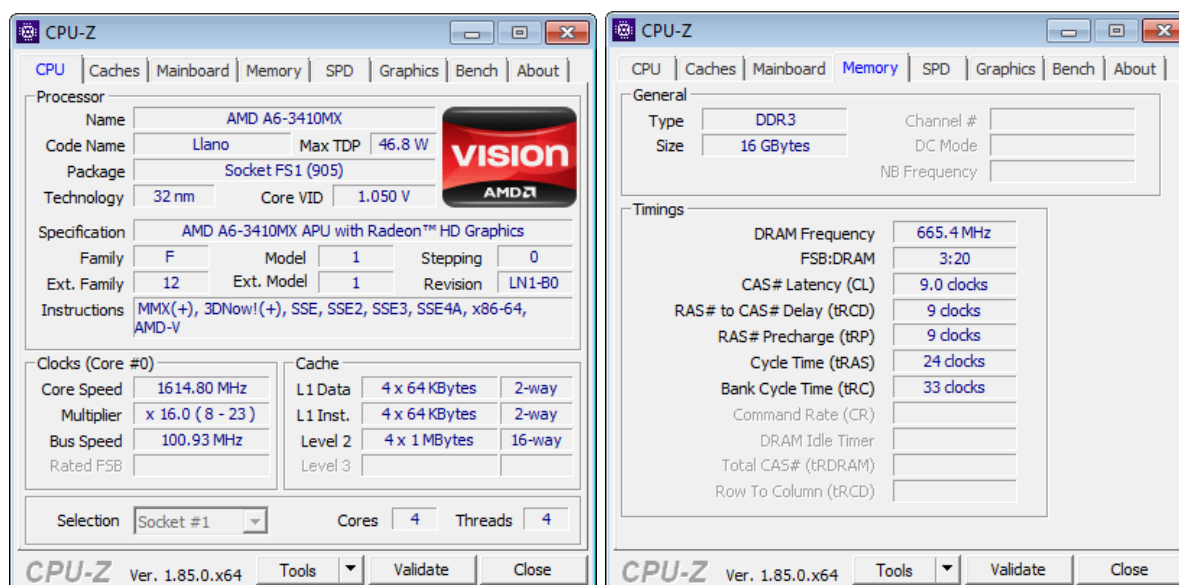


Рис. 15. ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

Табл. 3. Скорость выполнения шифрования/расшифрования алгоритма AES

Алгоритм и размер ключа	Размер данных [Мбайт]	Шифрование/Расшифрование	Скорость [Мбайт/с]
В РЕЖИМЕ ЕСВ (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 MixColumns» [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 MixColumns(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 Column[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);
        MixColumns(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);
        InvMixColumns(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, 0xf6, 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 ThreadEncryption(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
high 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*/

```

II. 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 x64 ЦП хеш-функция семейства SHA2 под названием SHA-512.

1. 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.

2. Основные операции

Помимо основных (базовых) операций, используемых в ЭВМ: *XOR*, *AND*, *OR*, *NE* (\neg), сложение по модулю 2^{64} , вводятся операции правого поворота (*ROTR*) и правого сдвига (которое также присутствует в ЭВМ) (*SHR*).

$ROTR^n(x)$ – поворот вправо (циклический правый сдвиг) операция, где x это 64 битное слово и n целое число, которое $0 \leq n < 64$, операция математически определена как: $ROTR^n(x) = (x \gg n) \vee (x \ll 64 - n)$.

$SHR^n(x)$ – операция правого сдвига, где x это 64 битное слово и n целое число, которое $0 \leq n < 64$, операция математически определена как: $SHR^n(x) = x \gg n$.

3. Функции и константы

3.1 Функции

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma_0^{[512]}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$$

$$\Sigma_1^{[512]}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$$

$$\sigma_0^{[512]}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$$

$$\sigma_1^{[512]}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$$

3.2 Константы

SHA-512 использует последовательности из 80 констант по 64 битных слов $K_0^{[512]}, K_1^{[512]}, \dots, K_{79}^{[512]}$.

```

428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240calcc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcdbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273ecceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817

```

4. Подготовка к вычислению хеш-значения

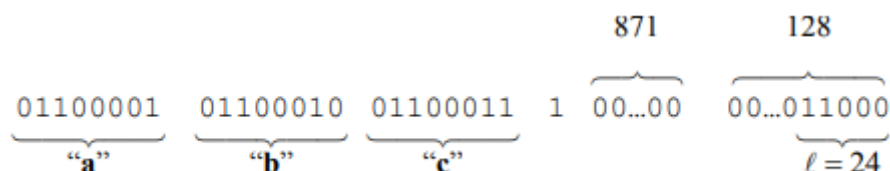
4.1 Дополнение сообщения

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 $\lambda + 1 + k \equiv 896 \pmod{1024}$.

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 $8 \times 3 = 24$ 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.

4.2 Получение сообщения

Для *SHA-512* сообщение и его дополнение представляется как N 1024 битных блоков, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Так, полученный 1024 битный блок может быть представлен как 16 64 битных слов, первый 64 битный блок сообщения i обозначается $M_0^{(i)}$, следующий блок в 64 бита обозначается как $M_1^{(i)}$, и так далее до $M_{15}^{(i)}$.

4.3 Настройка инициализации начальных хеш-значений $H^{(0)}$

Для *SHA-512* инициализация начальных хеш-значений $H^{(0)}$ представляет собой восемь 64 битных слов в шестнадцатеричной системе счисления:

$$\begin{aligned}
 H_0^{(0)} &= 6a09e667f3bcc908 \\
 H_1^{(0)} &= bb67ae8584caa73b \\
 H_2^{(0)} &= 3c6ef372fe94f82b \\
 H_3^{(0)} &= a54ff53a5f1d36f1 \\
 H_4^{(0)} &= 510e527fade682d1 \\
 H_5^{(0)} &= 9b05688c2b3e6c1f \\
 H_6^{(0)} &= 1f83d9abfb41bd6b \\
 H_7^{(0)} &= 5be0cd19137e2179
 \end{aligned}$$

5. Хеш-функция SHA-512

SHA-512 может быть использован для вычисления хеш-значения сообщения M , имеющего длину λ бит, где $0 \leq \lambda < 2^{128}$.

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 W_0, W_1, \dots, W_{79} .

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

Слова хеш-значений маркируются как $H_0^{(i)}, H_1^{(i)}, \dots, H_7^{(i)}$, которые начинаются с начальных хеш-значений $H^{(0)}$, затем итеративно вычисляются их промежуточные хеш-значения обозначаемые как $H^{(i)}$, записываются они хеш-значениями $H^{(N)}$.

SHA512 также использует две временные переменные T_1 и T_2 .

5.1 Подготовка к алгоритму *SHA-512*

1. Установка начальных хеш-значений $H^{(0)}$, описанных в секции 4.3
2. Принимаемое сообщение дополняется и преобразуется по правилам в секциях 4.1 и 4.2.

5.2 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 2^{64} .

Каждый блок сообщения $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ обрабатываются в порядке, используя следующие шаги:

For $i=1$ to N :
{

1. Подготовка схемы сообщения W_0, W_1, \dots, W_{79} , $\{W_t\}$

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{(512)}(W_{t-2}) + W_{t-7} + \sigma_0^{(512)}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 79 \end{cases}$$

2. Инициализация восьми рабочих переменных: a, b, c, d, e, f, g , и h , а также $H^{(i-1)}$ вычисленными хеш-значениями:

$$a = H_0^{(i-1)}$$

$$b = H_1^{(i-1)}$$

$$c = H_2^{(i-1)}$$

$$d = H_3^{(i-1)}$$

$$e = H_4^{(i-1)}$$

$$f = H_5^{(i-1)}$$

$$g = H_6^{(i-1)}$$

$$h = H_7^{(i-1)}$$

3. For $t=0$ to 79:

$$\left\{ \begin{array}{l} T_1 = h + \sum_i^{[512]}(e) + Ch(e, f, g) + K_i^{[512]} + W_i \\ T_2 = \sum_0^{[512]}(a) + Maj(a, b, c) \\ h = g \\ g = f \\ f = e \\ e = d + T_1 \\ d = c \\ c = b \\ b = a \\ a = T_1 + T_2 \end{array} \right\}$$

4. Вычисление следующих хеш-значений $H^{(i)}$:

$$\left\{ \begin{array}{l} H_0^{(i)} = a + H_0^{(i-1)} \\ H_1^{(i)} = b + H_1^{(i-1)} \\ H_2^{(i)} = c + H_2^{(i-1)} \\ H_3^{(i)} = d + H_3^{(i-1)} \\ H_4^{(i)} = e + H_4^{(i-1)} \\ H_5^{(i)} = f + H_5^{(i-1)} \\ H_6^{(i)} = g + H_6^{(i-1)} \\ H_7^{(i)} = h + H_7^{(i-1)} \end{array} \right\}$$

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

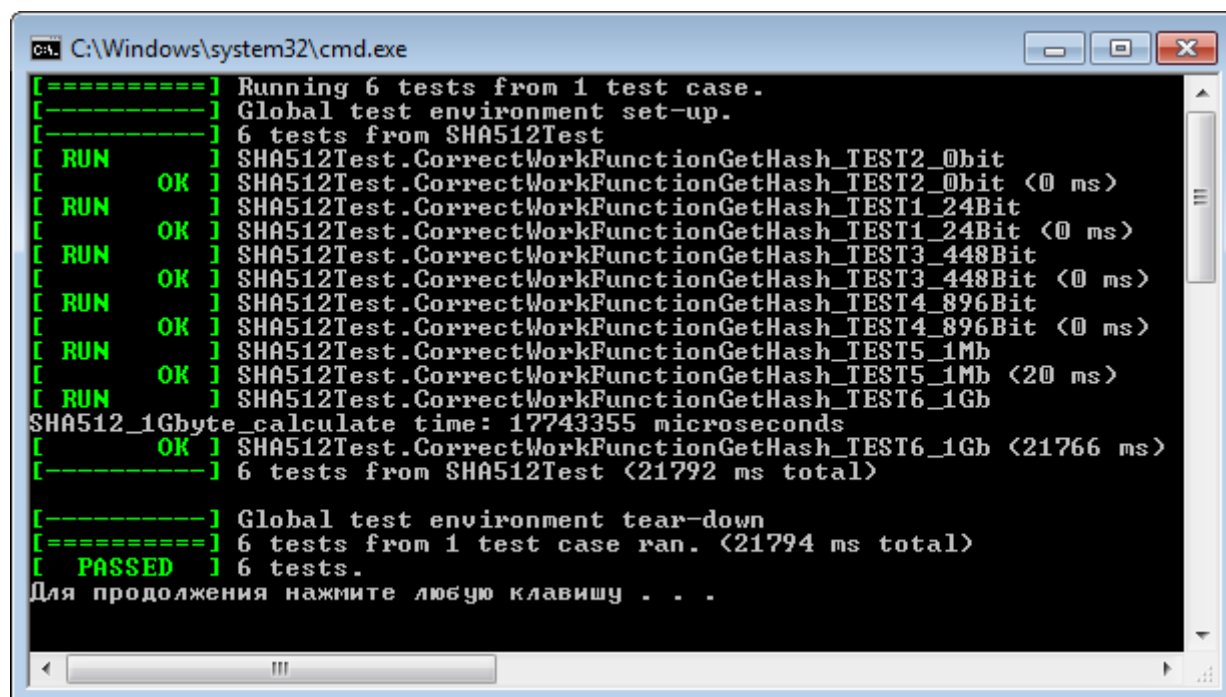
$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

6. 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 [\[ссылка\]](#)) 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.



```

C:\Windows\system32\cmd.exe

[=====] Running 6 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 6 tests from SHA512Test
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST2_0bit
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST2_0bit <0 ms>
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST1_24Bit
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST1_24Bit <0 ms>
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST3_448Bit
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST3_448Bit <0 ms>
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST4_896Bit
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST4_896Bit <0 ms>
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST5_1Mb
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST5_1Mb <20 ms>
[ RUN     ] SHA512Test.CorrectWorkFunctionGetHash_TEST6_1Gb
[ OK      ] SHA512Test.CorrectWorkFunctionGetHash_TEST6_1Gb <21766 ms>
SHA512_1Gbyte_calculate time: 17743355 microseconds
[-----] 6 tests from SHA512Test <21792 ms total>

[-----] Global test environment tear-down
[=====] 6 tests from 1 test case ran. <21794 ms total>
[ PASSED ] 6 tests.
Для продолжения нажмите любую клавишу . . .
  
```

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.

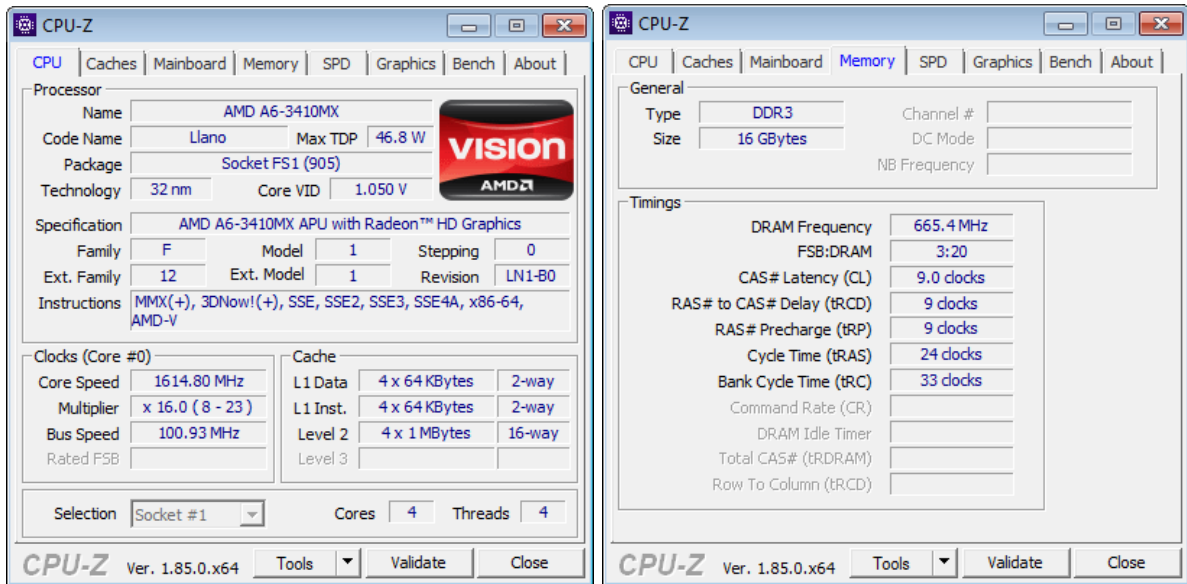


Рис. 2. ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

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, 0x5cb0a9dcdbd41fbd4,
0x76f988da831153b5,
        0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f,
0xbf597fc7beef0ee4,
        0xc6e00bf33da88fc2, 0xd5a79147930aa725, 0x06ca6351e003826f,
0x142929670a0e6e70,
        0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
0x53380d139d95b3df,
        0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edae6,
0x92722c851482353b,
        0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791,
0xc76c51a30654be30,
        0xd192e819d6ef5218, 0xd69906245565a910, 0xf40e35855771202a,
0x106aa07032bbd1b8,
        0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99,
0x34b0bcb5e19b48a8,
        0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
0x682e6ff3d6b2b8a3,
        0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72,
0x8cc702081a6439ec,
        0x90befffa23631e28, 0xa4506cebd82bde9, 0xbef9a3f7b2c67915,
0xc67178f2e372532b,
        0xca273ecee26619c, 0xd186b8c721c0c207, 0xead7dd6cde0eb1e,
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 Message to resolve equation : [InputMessageSize + 1 +
NumberOfBytes0x00 = 112(mod 128)] <=> [1 + 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::HashComplutation() {

    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);
        }
        HashComplutationBlock();
        M->clear();
    }
};

void SHA512::HashComplutationBlock() {
    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();
    HashComputation();

    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;
};

```


III. *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).

1. 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. Браун (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 подобранных 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».

2. 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 $GF(2^m)$;
- Псевдослучайная кривая над полем $GF(2^m)$, 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 $G = (G_x, G_y)$.

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

3. Эллиптические кривые над простыми полями $GF(p)$

Для каждого простого p существует псевдослучайная кривая

$$E : y^2 \equiv x^3 - 3x + b \pmod{p}$$

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 p ;
- порядком подгруппы точек эллиптической кривой n ;
- коэффициентом b , таким, что:

$$4a^3 + 27b^2 \not\equiv 0 \pmod{p}$$

- координатой x базовой точки G_x ;
- координатой y базовой точки G_y ;

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

$p =$ 6277101735386680763835789423207666416083908700390324961279
 $n =$ 6277101735386680763835789423176059013767194773182842284081
 $b =$ 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
 $G_x =$ 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
 $G_y =$ 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Кривая NIST P-224

$p =$ 2695994666715063979466701508701963067355791626002630814351
 0066298881
 $n =$ 2695994666715063979466701508701962594045780771442439172168
 2722368061
 $b =$ b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943
 2355ffb4
 $G_x =$ b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6
 115c1d21
 $G_y =$ bd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199
 85007e34

Кривая NIST P-256

$p =$ 1157920892103562487626974469494075735300861434152903141955
 33631308867097853951
 $n =$ 115792089210356248762697446949407573529996955224135760342
 422259061068512044369
 $b =$ 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6
 3bce3c3e 27d2604b
 $G_x =$ 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0
 f4a13945 d898c296
 $G_y =$ 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece
 cbb64068 37bf51f5

Кривая NIST P-384

$p =$ 3940200619639447921227904010014361380507973927046544666794
 8293404245721771496870329047266088258938001861606973112319
 $n =$ 3940200619639447921227904010014361380507973927046544666794
 6905279627659399113263569398956308152294913554433653942643
 $b =$ b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112
 0314088f 5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef
 $G_x =$ aa87ca22 be8b0537 8eb1c71e f320ad74 6eld3b62 8ba79b98
 59f741e0 82542a38 5502f25d bf55296c 3a545e38 72760ab7
 $G_y =$ 3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c
 e9da3113 b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f

Кривая NIST P-521

$p =$ 686479766013060971498190079908139321726943530014330540939
 446345918554318339765605212255964066145455497729631139148
 0858037121987999716643812574028291115057151
 $n =$ 686479766013060971498190079908139321726943530014330540939
 446345918554318339765539424505774633321719753296399637136
 3321113864768612440380340372808892707005449
 $b =$ 051 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b
 99b315f3 b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd
 3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00
 $G_x =$ c6 858e06b7 0404e9cd 9e3ecb66 2395b442 9c648139
 053fb521 f828af60 6b4d3dba a14b5e77 efe75928 fe1dc127
 a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66
 $G_y =$ 118 39296a78 9a3bc004 5c8a5fb4 2c7d1bd9 98f54449
 579b4468 17afbd17 273e662c 97ee7299 5ef42640 c550b901
 3fad0761 353c7086 a272c240 88be9476 9fd16650

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

Кривая ГОСТ P-256

$a = 7$
 $p =$ 57896044618658097711785492504343953926634992332820282019728792003956564821041
 $n =$ 57896044618658097711785492504343953927082934583725450622380973592137631069619
 $b =$ 43308876546767276905765904595650931995942111794451039583252968842033849580414
 $G_x = 2$
 $G_y =$ 4018974056539037503335449422937059775635739389905545080690979365213431566280

Кривая ГОСТ P-512

$a = 7$
 $p =$ 36239861022290036359077887536838743060213209255346786050865
 46150450856166624002482588482022271496854025090823603058735
 1637342638 22371964987228582907372403
 $n =$ 36239861022290036359077887536838743060213209255346786050865
 46150450856166623969164898305032863068499961404079437936585
 455865192212970734808812618120619743
 $b =$ 15186550692108285345089500347140431549287475277402064361940
 18823352809982443793732829756914785974674866041605397883677
 596626326413990136959047435811826396
 $G_x =$ 1928356944067022849399309401243137598997786635459507974357
 0754913077665926858354410655576810031848748196580049032123
 32884252335830250729527632383493573274
 $G_y =$ 22887286933719728599700121555294784163535623273295061803
 14497425931102860301572814141997072271708807066593850650
 334152381857347798885864807605098724013854

4. 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 $C_1(x_1, y_1)$ и $C_2(x_2, y_2)$ равны, если равны их соответствующие координаты ($C_1 = C_2$, если $x_1 = x_2$ и $y_1 = y_2$).

На множестве точек эллиптической кривой E операцию сложения обозначают знаком «+». Для двух произвольных точек $C_1(x_1, y_1)$ и $C_2(x_2, y_2)$ эллиптической кривой E рассматривают несколько случаев:

1. Для точек $C_1(x_1, y_1)$ и $C_2(x_2, y_2)$, координаты которых удовлетворяют условию $x_1 \neq x_2$, их суммой называется точка $C_3(x_3, y_3)$, координаты которой определяются сравнениями:

$$\begin{cases} x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}, \\ y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}, \end{cases}$$

Где:

$$\lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod{p}.$$

2. Для точек $C_1(x_1, y_1)$ и $C_2(x_2, y_2)$, координаты которых удовлетворяют условию $x_1 = x_2$ и $y_1 = y_2 \neq 0$, их суммой называется точка $C_3(x_3, y_3)$, координаты которой определяются сравнениями:

$$\begin{cases} x_3 \equiv \lambda^2 - 2x_1 \pmod{p}, \\ y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}, \end{cases}$$

Где:

$$\lambda \equiv \frac{3x_1^2 + a}{2y_1} \pmod{p}.$$

3. Для точек $C_1(x_1, y_1)$ и $C_2(x_2, y_2)$, координаты которых удовлетворяют условию $x_1 = x_2$ и $y_1 = y_2 \pmod{p}$, их суммой называется точка $C_3(x_3, y_3) = O$ – нулевой точкой без определения её x и y координат. В этом случае точка C_2 называется отрицанием точки C_1 . Для нулевой точки O выполнены равенства:

$$C + O = O + C = C,$$

Где:

C – 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:

$$p+1-2\sqrt{p} \leq m \leq p+1+2\sqrt{p}.$$

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

$$C = \underbrace{P + \dots + P}_k = kP$$

5. Параметры пользователя

Каждый пользователь схемы ЦП должен обладать личными параметрами:

- ключом подписи – целым числом d , удовлетворяющим неравенству:

$$0 < d < n$$

- ключом проверки подписи – точкой эллиптической кривой Q с координатами (x_Q, y_Q) удовлетворяющая равенству:

$$dG = Q$$

6. Формирование цифровой подписи

Для получения цифровой подписи под сообщением M необходимо выполнить следующие шаги:

- 1) Вычислить хеш-значение сообщения M :

$$h = \text{HASH}(M)$$

- 2) Вычислить целое число e :

$$e \equiv h \pmod{n}$$

Если $e = 0$, то определить $e = 1$.

- 3) Получить случайное (псевдослучайное) целое число k , удовлетворяющее неравенству:

$$0 < k < n$$

- 4) Вычислить точку эллиптической кривой $C = kG$ и определить:

$$r = x_C \pmod{n},$$

Где: x_C – x координата точки C .

Если $r = 0$, то вернуться к шагу 3).

- 5) Вычислить значение:

$$s \equiv (rd + ke) \pmod{n}$$

Если $s = 0$, то вернуться к шагу 3).

- 6) Определить цифровую подпись: как два выходных параметра r и s .

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

7. Проверка цифровой подписи

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

- 1) Получение параметров r и s – цифровой подписи сообщения M . Если выполнены неравенства $0 < r < n$ и $0 < s < n$, то перейти к следующему шагу. В противном случае подпись неверна.

- 2) Вычислить хеш-значение полученного сообщения M :

$$h = \text{HASH}(M)$$

- 3) Вычислить целое значение e :

$$e \equiv h(\bmod n)$$

Если $e = 0$, то определить $e = 1$.

- 4) Вычислить значение:

$$v \equiv e^{-1}(\bmod n)$$

- 5) Вычислить значения:

$$\begin{aligned} z_1 &\equiv sv(\bmod n) \\ z_2 &\equiv -rv(\bmod n) \end{aligned}$$

- 6) Вычислить точку эллиптической кривой $C = z_1G + z_2Q$ и определить:

$$R \equiv x_C(\bmod n)$$

Где: x_C – x координата точки C .

- 7) Если выполнено равенство $R = r$, то подпись принимается, в противном случае – подпись неверна.

Исходными данными этого процесса являются подписанное сообщение M , 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.

8. 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> с++11 с точностью до микросекунд) приведены на Рис. 1.

```

C:\Windows\system32\cmd.exe
[=====] Running 7 tests from 1 test case.
[=====] Global test environment set-up.
[=====] 7 tests from ECDSATest
[ RUN      ] ECDSATest.TEST_ECDSA_GOST_256
ECDSA_GOST_256 CreateKeyCheckDigitalSign time: 113777 microseconds
ECDSA_GOST_256 CreateDigitalSign time: 222844 microseconds
ECDSA_GOST_256 CheckDigitalSign time: 441043 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_GOST_256 (795 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_GOST_512
ECDSA_GOST_512 CreateKeyCheckDigitalSign time: 230158 microseconds
ECDSA_GOST_512 CreateDigitalSign time: 890866 microseconds
ECDSA_GOST_512 CheckDigitalSign time: 1817986 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_GOST_512 (2956 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_NIST_192
ECDSA_NIST_192 CreateKeyCheckDigitalSign time: 84991 microseconds
ECDSA_NIST_192 CreateDigitalSign time: 127296 microseconds
ECDSA_NIST_192 CheckDigitalSign time: 247555 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_NIST_192 (477 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_NIST_224
ECDSA_NIST_224 CreateKeyCheckDigitalSign time: 99719 microseconds
ECDSA_NIST_224 CreateDigitalSign time: 178577 microseconds
ECDSA_NIST_224 CheckDigitalSign time: 350244 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_NIST_224 (645 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_NIST_256
ECDSA_NIST_256 CreateKeyCheckDigitalSign time: 113060 microseconds
ECDSA_NIST_256 CreateDigitalSign time: 220783 microseconds
ECDSA_NIST_256 CheckDigitalSign time: 452615 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_NIST_256 (803 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_NIST_384
ECDSA_NIST_384 CreateKeyCheckDigitalSign time: 172384 microseconds
ECDSA_NIST_384 CreateDigitalSign time: 515657 microseconds
ECDSA_NIST_384 CheckDigitalSign time: 1000540 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_NIST_384 (1706 ms)
[ RUN      ] ECDSATest.TEST_ECDSA_NIST_521
ECDSA_NIST_521 CreateKeyCheckDigitalSign time: 233546 microseconds
ECDSA_NIST_521 CreateDigitalSign time: 932308 microseconds
ECDSA_NIST_521 CheckDigitalSign time: 1929149 microseconds
[ OK      ] ECDSATest.TEST_ECDSA_NIST_521 (3112 ms)
[=====] 7 tests from ECDSATest (10500 ms total)

[=====] Global test environment tear-down
[=====] 7 tests from 1 test case ran. (10503 ms total)
[ PASSED  ] 7 tests.
Для продолжения нажмите любую клавишу . . .

```

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.

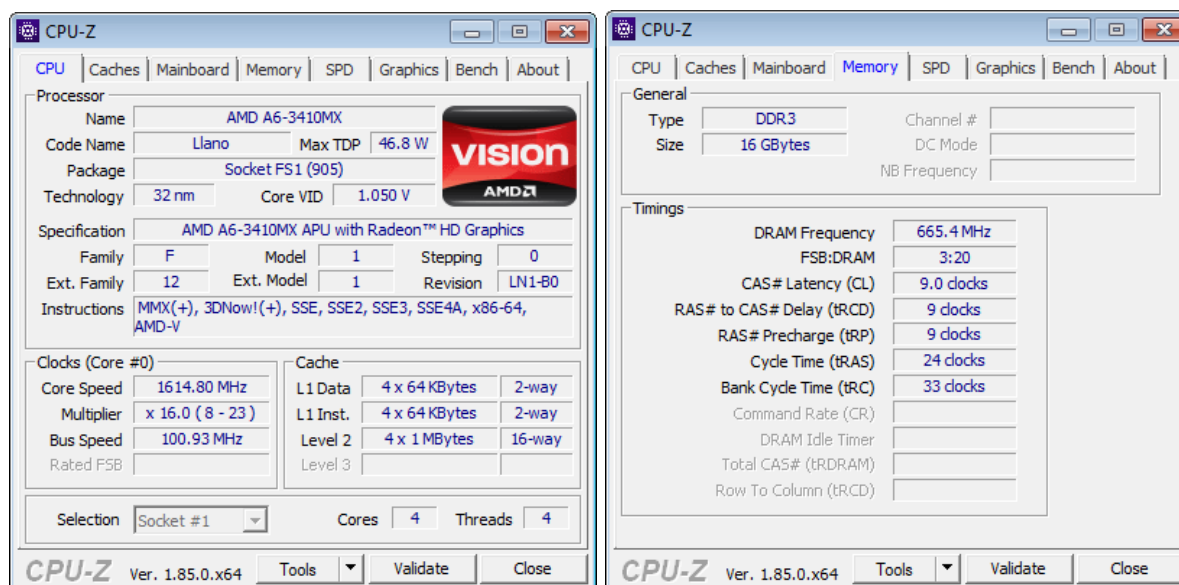


Рис. 2. ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

Табл. 1. Скорость выполнения операций ЦП

Используемая хеш-функция	Выработка ключа проверки ЦП / Создание ЦП/ Проверка ЦП	Скорость [секунд]
<i>Кривая ГОСТ P-256</i>		
SHA-512	Выработка ключа проверки ЦП	0,114
SHA-512	Создание ЦП	0,223
SHA-512	Проверка ЦП	0,441
<i>Кривая ГОСТ P-512</i>		
SHA-512	Выработка ключа проверки ЦП	0,230
SHA-512	Создание ЦП	0,891
SHA-512	Проверка ЦП	1,818
<i>Кривая NIST P-192</i>		
SHA-512	Выработка ключа проверки ЦП	0,085
SHA-512	Создание ЦП	0,127
SHA-512	Проверка ЦП	0,248
<i>Кривая NIST P-224</i>		
SHA-512	Выработка ключа проверки ЦП	0,100
SHA-512	Создание ЦП	0,179
SHA-512	Проверка ЦП	0,350

<i>Кривая NIST P-256</i>		
SHA-512	Выработка ключа проверки ЦП	0,113
SHA-512	Создание ЦП	0,221
SHA-512	Проверка ЦП	0,453
<i>Кривая NIST P-384</i>		
SHA-512	Выработка ключа проверки ЦП	0,172
SHA-512	Создание ЦП	0,516
SHA-512	Проверка ЦП	1,001
<i>Кривая NIST P-521</i>		
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  $y^2 = x^3 + a*x + b \pmod{p}$  | Analogically abbreviation in GOST 34.10-2012
```

```

        bigint _b;                // Coefficient 'b' of Curve equal  $y^2 = x^3 + a*x + b \pmod{p}$  | Analogically abbreviation in GOST 34.10-2012
        bigint _p;                // Prime module of Curve equal  $y^2 = x^3 + a*x + b \pmod{p}$  | Analogically abbreviation in GOST 34.10-2012
        bigint _Gx;               // Coordinate 'x' of Point G of Elliptical Curve [ $y^2 = x^3 + a*x + b \pmod{p}$ ] order n | In GOST 34.10-2012 it is 'x' coordinate of point P
        bigint _Gy;               // Coordinate 'y' of Point G of Elliptical Curve [ $y^2 = x^3 + a*x + b \pmod{p}$ ] order n | In GOST 34.10-2012 it is 'y' coordinate of point P
        bigint _n;                // Order of SubGroup of Points of Elliptical Curve [ $y^2 = x^3 + a*x + b \pmod{p}$ ] | 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("433088765467672769057659045956509319959421117944510395832529688420
33849580414"),

        bigint("578960446186580977117854925043439539266349923328202820197287920039
56564821041"),
        bigint("2"),

        bigint("401897405653903750333544942293705977563573938990554508069097936521
3431566280"),

        bigint("578960446186580977117854925043439539270829345837254506223809735921
37631069619")
    ) { };
};

class ECDSA_GOST_512 : public ECDSA_PrimeField {
public:
    ECDSA_GOST_512() : ECDSA_PrimeField(
        bigint("7"),

        bigint("151865506921082853450895003471404315492874752774020643619401882335
28099824437937328297569147859746748660416053978836775966263264139901369590474358
11826396"),

        bigint("362398610222900363590778875368387430602132092553467860508654615045
08561666240024825884820222714968540250908236030587351637342638223719649872285829
07372403"),

        bigint("192835694406702284939930940124313759899778663545950797435707549130
77665926858354410655576810031848748196580049032123328842523358302507295276323834
93573274"),

        bigint("228872869337197285997001215552947841635356232732950618031449742593
11028603015728141419970722717088070665938506503341523818573477988858648076050987
24013854"),

        bigint("362398610222900363590778875368387430602132092553467860508654615045
08561666239691648983050328630684999614040794379365854558651922129707348088126181
20619743")
    ) { };
};

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("189582862855666080004086685444939264155046809686793210757872346725
64"),

        bigint("269599466671506397946670150870196306735579162600263081435100662988
81"),

        bigint("192779291135662930711103080346994880268319342194524401566497843520
33"),

        bigint("199268087580344709701979743708887491842059919906039495376373431987
72"),

        bigint("269599466671506397946670150870196259404578077144243917216827223680
61")
    ) { };
};

class ECDSA_NIST_256 : public ECDSA_PrimeField {
public:
    ECDSA_NIST_256() : ECDSA_PrimeField(
        bigint("-3"),

        bigint("410583637251521421293261297800472684091144410159937255548352563140
39467401291"),

        bigint("115792089210356248762697446949407573530086143415290314195533631308
867097853951 "),

        bigint("484395612939064517590525852527979142027629495260417479958440807170
82404635286"),

        bigint("361342509567497957985851279195878819566111066729850150718771982535
68414405109"),

        bigint("115792089210356248762697446949407573529996955224135760342422259061
068512044369")
    ) { };
};

class ECDSA_NIST_384 : public ECDSA_PrimeField {
public:
    ECDSA_NIST_384() : ECDSA_PrimeField(
        bigint("-3"),

        bigint("275801935599597058778490118403890480930569058563615685214287073019
88689241309860865136260764883745107765439761230575"),

        bigint("394020061963944792122790401001436138050797392704654466679482934042
45721771496870329047266088258938001861606973112319"),

        bigint("262470350957996892686231567445669818918529234911092133878156159009
25518854738050089022388053975719786650872476732087"),

        bigint("832571096148902998554675128952010817928785304886131559470920590248
0503199884419224438643760392947333078086511627871"),

        bigint("394020061963944792122790401001436138050797392704654466679469052796
27659399113263569398956308152294913554433653942643")
    ) { };
};

class ECDSA_NIST_521 : public ECDSA_PrimeField {
public:

```

```

ECDSA_NIST_521() : ECDSA_PrimeField(
    bigint("-3"),

    bigint("109384903807373427451111239076680556993620759895168374899458639449
59531161507350160137087375737596232485921322967063133094384525315910129121423274
88478985984"),

    bigint("686479766013060971498190079908139321726943530014330540939446345918
55431833976560521225596406614545549772963113914808580371219879997166438125740282
91115057151"),

    bigint("266174080205021706322876871672336096072985916875697314770667136841
88029449964278084915450806277719023520942412250655586621571135455709168141616373
15895999846"),

    bigint("375718002577002046354550722449118360359445513476976248669456777961
55444774405563166912344050129455395621444445372894285225856667291965808101243442
77578376784"),

    bigint("686479766013060971498190079908139321726943530014330540939446345918
55431833976553942450577463332171975329639963713633211138647686124403803403728088
92707005449")
    ) { };
};

#endif

```

IV. 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.

1. 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 r_i в случайные числа x_i , 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 $[0;n]$ может быть получено из сгенерированной последовательности случайных бит длины $\lceil \lg n \rceil + 1$ 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 $l \gg k$ (l значительно больше k), которая «выглядит случайной»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 времени² вероятностного алгоритма, который бы мог корректно отличить выходную последовательность генератора от истинно случайной последовательности той же длины с вероятностью превышающей $\frac{1}{2}$.

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 $\frac{1}{2}$.

2. Виды ГПСЧ

Генераторы случайных чисел по способу получения чисел делятся на:

- аппаратные;
- табличные;
- алгоритмические.

Аппаратные генераторы (истинно) случайных последовательностей должны обладать источником энтропии³. Разработка генераторов,

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.

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.

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.

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.

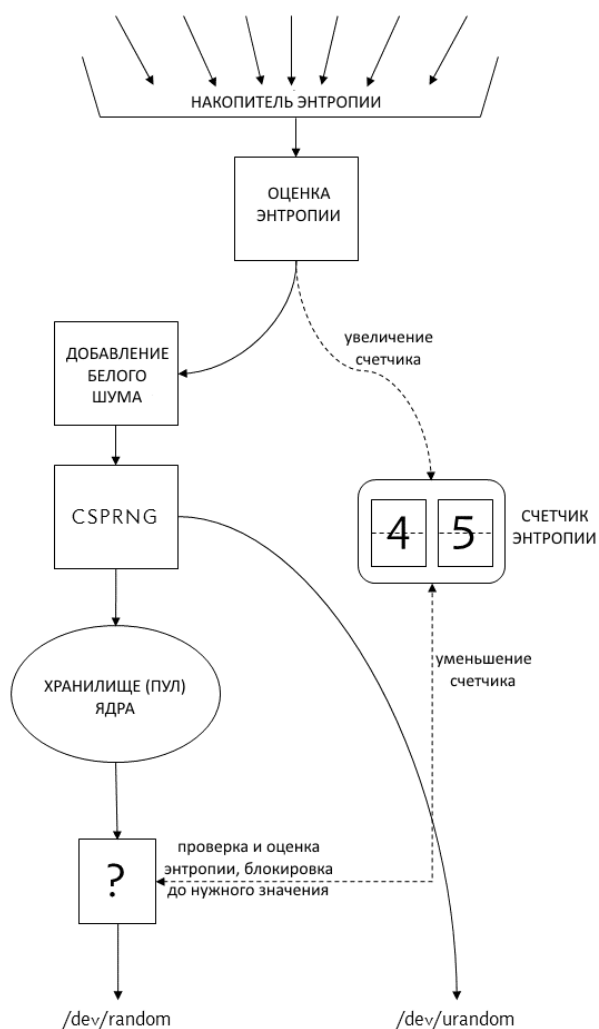
Табличные генераторы в качестве источника случайных чисел используют заранее подготовленные таблицы, содержащие проверенные некоррелированные числа и не являются генераторами в строгом понимании этого понятия. Недостатки такого способа очевидны: использование внешнего ресурса для хранения чисел, ограниченность 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.

3. 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 2^n 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 CSPRNG 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.

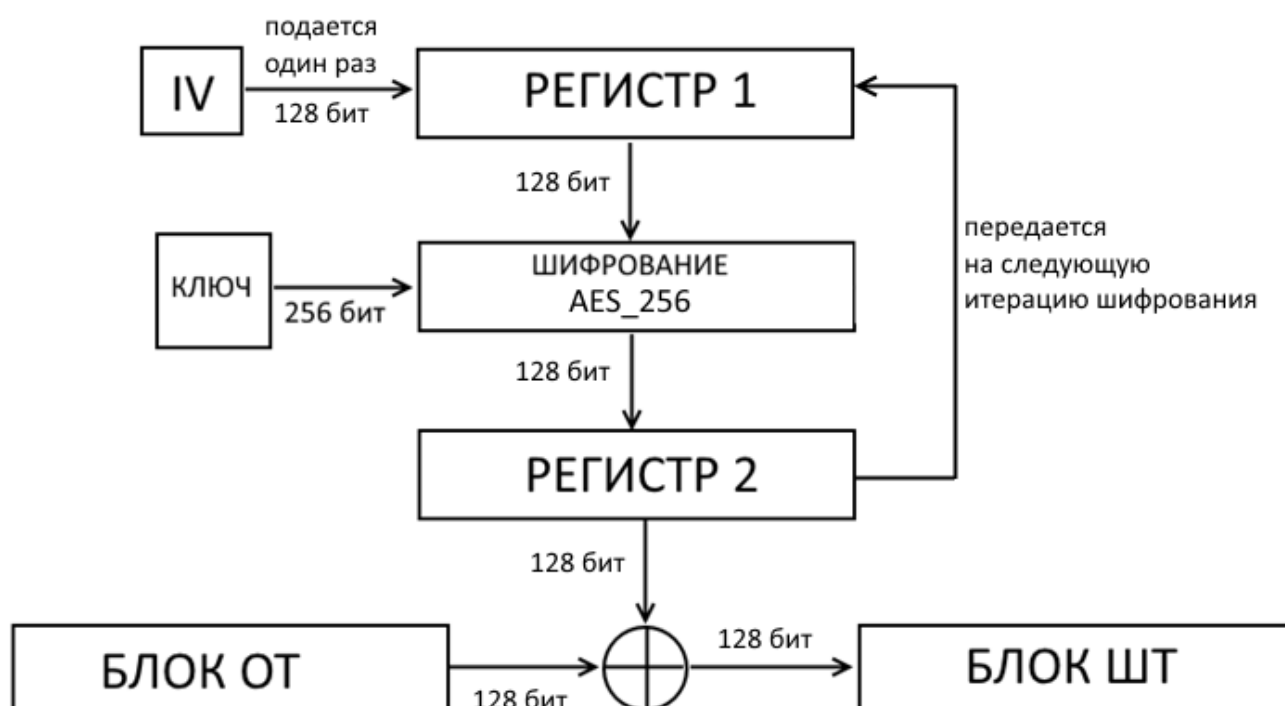
Статистические тесты NIST – пакет статистических тестов, разработанный Лабораторией информационных технологий, являющейся главной исследовательской организацией Национального института стандартов и технологий (NIST). В его состав входят 15 статистических тестов, целью которых является определение меры 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 приближительной энтропии;
- Тест кумулятивных сумм;
- Тест на произвольные отклонения;
- Другой тест на произвольные отклонения.

4. 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:

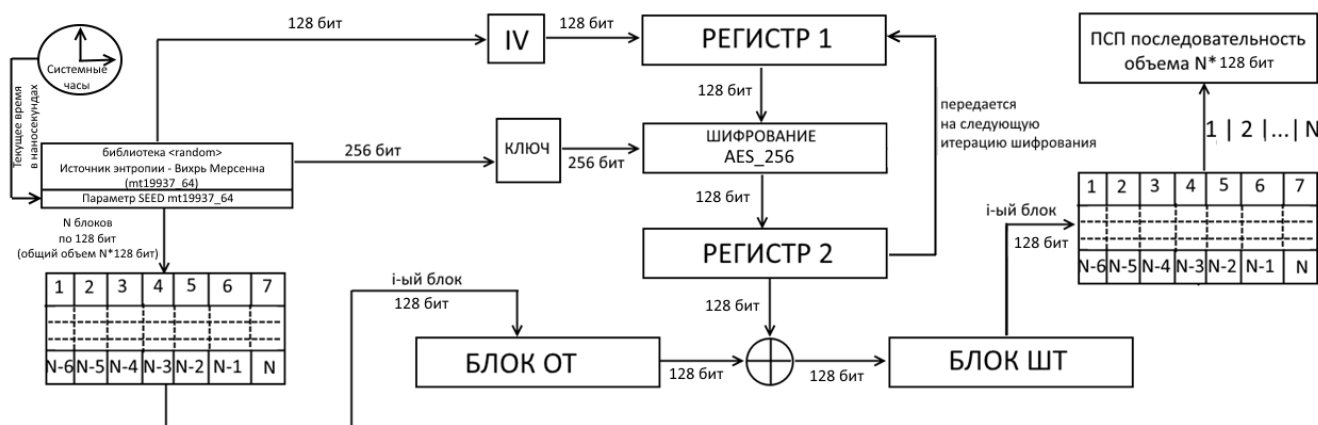
- вектор инициализации IV (128 бит);
- ключ (256 бит);
- блоки открытого текста [ОТ] (128 бит).

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 (семена). 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).

5. 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.

```

C:\Windows\system32\cmd.exe

[=====] Running 1 test from 1 test case.
[=====] Global test environment set-up.
[=====] 1 test from TestCSPRNG
[ RUN      ] TestCSPRNG.CorrectWorkGenerate_10Mbyte
10 Mbyte PRN Generation time: 10508623 microseconds
[ OK       ] TestCSPRNG.CorrectWorkGenerate_10Mbyte (12013 ms)
[=====] 1 test from TestCSPRNG (12013 ms total)

[=====] Global test environment tear-down
[=====] 1 test from 1 test case ran. (12016 ms total)
[ PASSED   ] 1 test.
Для продолжения нажмите любую клавишу . . .

```

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.

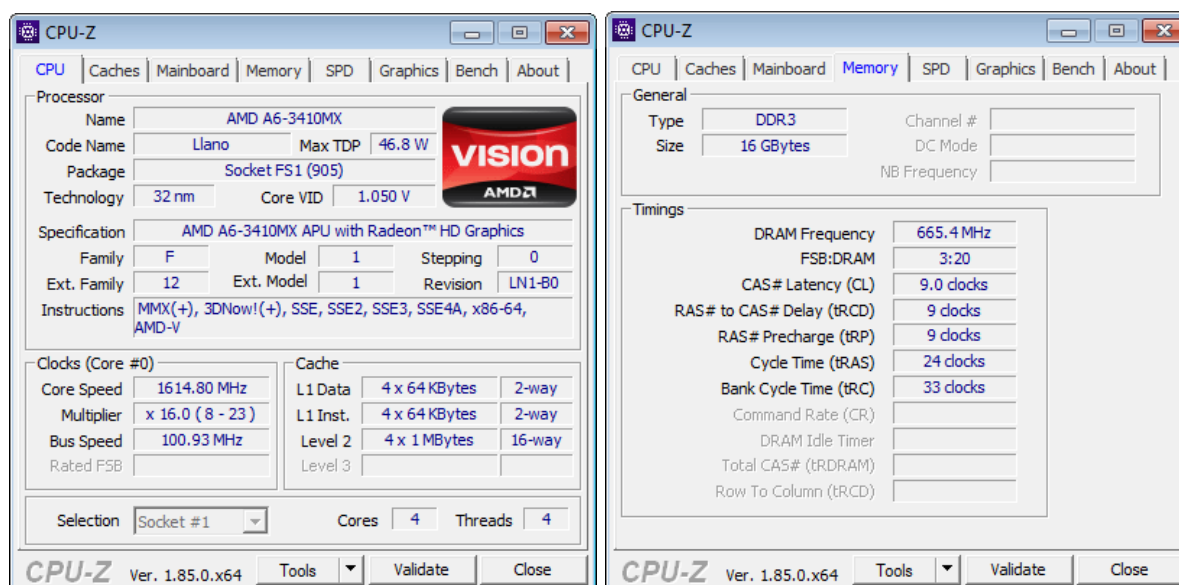


Рис. 2. ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

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.

```

/cygdrive/c/Users/HP/Desktop/Криптопротоклы/dieharder-3.31.1/dieharder
HP@HP- /cygdrive/c/Users/HP/Desktop/Криптопротоклы/dieharder-3.31.1/dieharder
$ ./dieharder -d 102 -g 201 -f c:/Users/HP/Desktop/gen.txt
#=====
# dieharder version 3.31.1 Copyright 2003 Robert G. Brown
#=====
#
# rng_name | filename | rands/second |
# file_input_raw | c:/Users/HP/Desktop/gen.txt | 4.03e+06 |
#=====
#
# test_name | intup | tsamples | psamples | p-value | Assessment
#=====
# The file file_input_raw was rewound 7 times
#
# sts_serial | 1 | 100000 | 100 | 0.13404142 | PASSED
# sts_serial | 2 | 100000 | 100 | 0.00001806 | WEAK
# sts_serial | 3 | 100000 | 100 | 0.00485251 | WEAK
# sts_serial | 3 | 100000 | 100 | 0.15841205 | PASSED
# sts_serial | 4 | 100000 | 100 | 0.00008669 | WEAK
# sts_serial | 4 | 100000 | 100 | 0.02000121 | PASSED
# sts_serial | 5 | 100000 | 100 | 0.00618419 | PASSED
# sts_serial | 5 | 100000 | 100 | 0.99060208 | PASSED
# sts_serial | 6 | 100000 | 100 | 0.04528520 | PASSED
# sts_serial | 6 | 100000 | 100 | 0.71954757 | PASSED
# sts_serial | 7 | 100000 | 100 | 0.29843226 | PASSED
# sts_serial | 7 | 100000 | 100 | 0.01902626 | PASSED
# sts_serial | 8 | 100000 | 100 | 0.83337031 | PASSED
# sts_serial | 8 | 100000 | 100 | 0.49136470 | PASSED
# sts_serial | 9 | 100000 | 100 | 0.52823807 | PASSED
# sts_serial | 9 | 100000 | 100 | 0.14178740 | PASSED
# sts_serial | 10 | 100000 | 100 | 0.40876597 | PASSED
# sts_serial | 10 | 100000 | 100 | 0.12354668 | PASSED
# sts_serial | 11 | 100000 | 100 | 0.22045179 | PASSED
# sts_serial | 11 | 100000 | 100 | 0.05247400 | PASSED
# sts_serial | 12 | 100000 | 100 | 0.93733239 | PASSED
# sts_serial | 12 | 100000 | 100 | 0.06492214 | PASSED
# sts_serial | 13 | 100000 | 100 | 0.92328448 | PASSED
# sts_serial | 13 | 100000 | 100 | 0.33290076 | PASSED
# sts_serial | 14 | 100000 | 100 | 0.90033587 | PASSED
# sts_serial | 14 | 100000 | 100 | 0.53312422 | PASSED
# sts_serial | 15 | 100000 | 100 | 0.08325320 | PASSED
# sts_serial | 15 | 100000 | 100 | 0.01558851 | PASSED
# sts_serial | 16 | 100000 | 100 | 0.28929829 | PASSED
# sts_serial | 16 | 100000 | 100 | 0.38658813 | PASSED
#
HP@HP- /cygdrive/c/Users/HP/Desktop/Криптопротоклы/dieharder-3.31.1/dieharder
$

```

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;
};

```

V. 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.

1. 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.

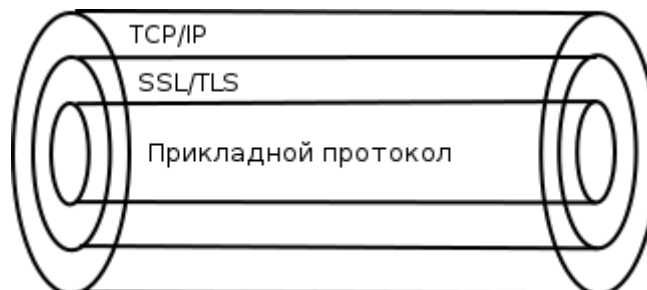
SSL — Secure Socket Layer, уровень защищенных сокетов. TLS — Transport Layer Security, безопасность транспортного уровня. SSL является более ранней системой, TLS появился позднее, он основан на спецификации SSL 3.0, разработанной компанией Netscape Communications. Тем не менее, задача у этих протоколов 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

2. 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.

3. 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:

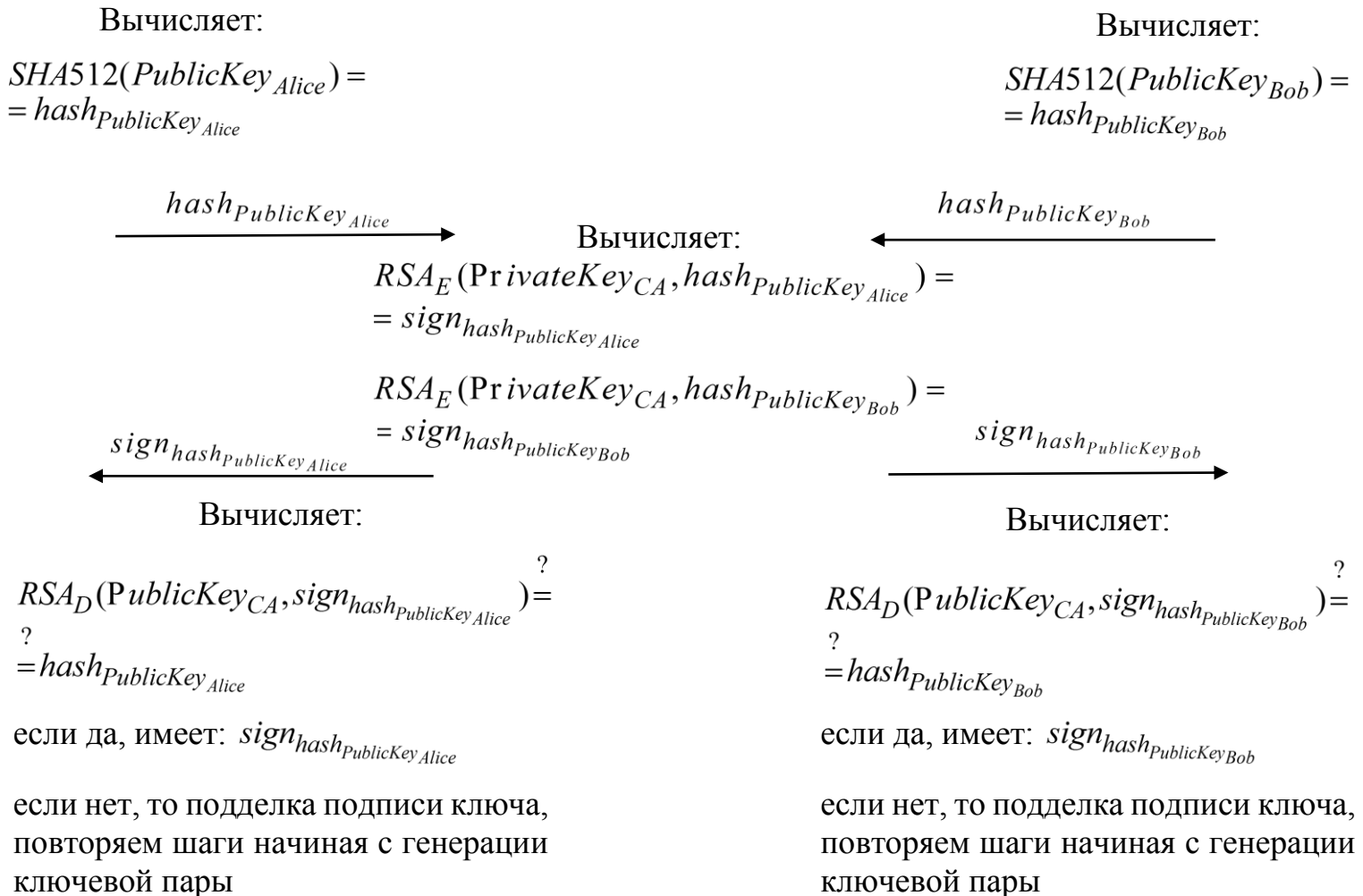
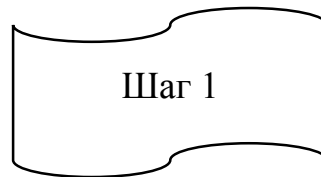
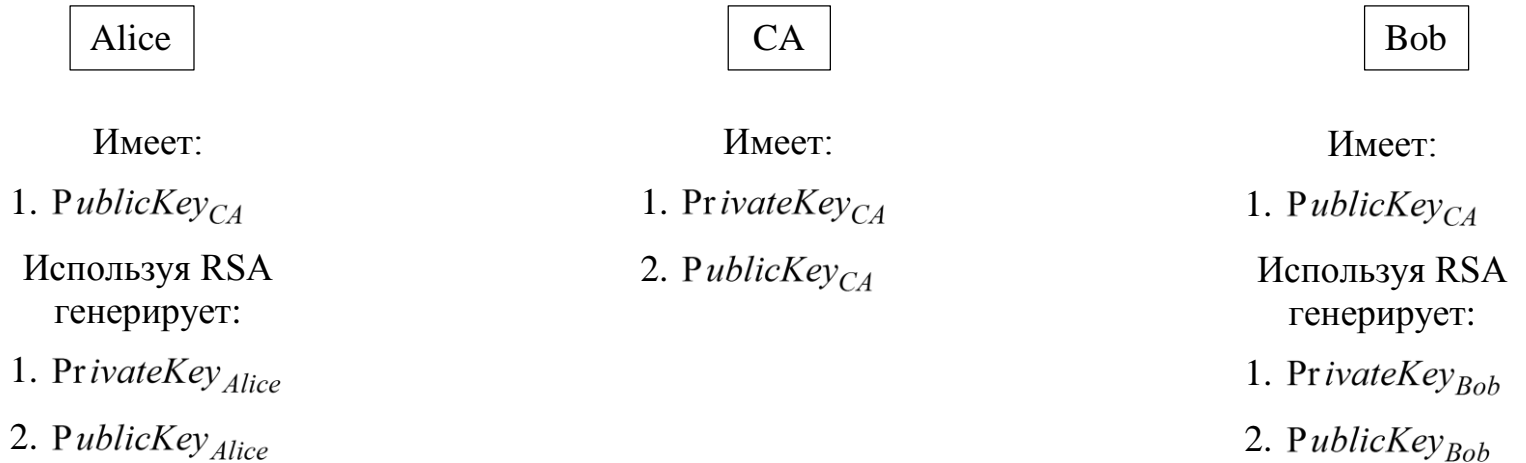
- получение письменных заявок (с указанием идентификационных данных) от пользователей
- подпись публичных ключей асимметричного шифрования пользователей $sign_{hash_{PublicKey_{User}}}$
- хранение хеш-значения $hash_{PublicKey_{User}}$ публичного ключа пользователя и его заявления

Пользователи имеют в распоряжении следующие алгоритмы:

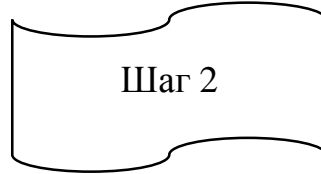
- RSA — для установления начального соединения, где шифрование описывается как $RSA_E(key, data)$, а расшифрование $RSA_D(key, data)$
- SHA512 — для вычисления хэш-значения параметров, где вычисление хеш-значения обозначается: $SHA512(data) = hash_{data}$
- ECDHE — протокол для выработки симметричного 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 $AES_E(key, data)$, а расшифрование $AES_D(key, data)$

Иллюстрация работы протокола выглядит следующим образом:

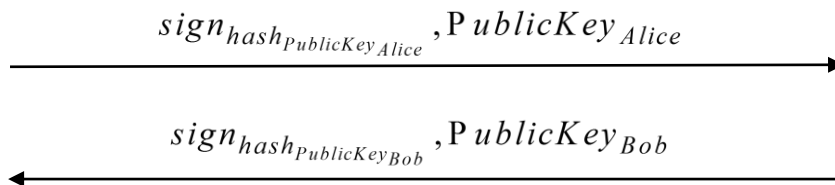


Примечание: на «Шаг 1» СА получает письменные заявки с указанием идентификационных данных заявителя. Получение подписи $sign_{hash_{PublicKey_{User}}}$ происходит в «письменном» порядке, $hash_{PublicKey_{User}}$ хранится в СА вместе с заявлением.



Имеет:

1. $PublicKey_{CA}$
2. $PrivateKey_{Alice}$
3. $PublicKey_{Alice}$
4. $sign_{hash_{PublicKey_{Alice}}}$



Вычисляет:

$$SHA512(PublicKey_{Bob}) = hash_{PublicKey_{Bob}}$$

$$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Bob}}}) = ?$$

$$= hash_{PublicKey_{Bob}}$$

если да, имеет: $PublicKey_{Bob}$

если нет, то идет подмена (при поиске злоумышленника, он – в базе СА, т.к. $sign_{hash_{PublicKey_{User}}}$ производится на $PrivateKey_{CA}$, а все данные об обратившихся пользователях хранятся в СА)

Имеет:

1. $PublicKey_{CA}$
2. $PrivateKey_{Bob}$
3. $PublicKey_{Bob}$
4. $sign_{hash_{PublicKey_{Bob}}}$

Вычисляет:

$$SHA512(PublicKey_{Alice}) = hash_{PublicKey_{Alice}}$$

$$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Alice}}}) = ?$$

$$= hash_{PublicKey_{Alice}}$$

если да, имеет: $PublicKey_{Bob}$

если нет, то идет подмена (при поиске злоумышленника, он – в базе СА, т.к. $sign_{hash_{PublicKey_{User}}}$ производится на $PrivateKey_{CA}$, а все данные об обратившихся пользователях хранятся в СА)

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. $PrivateKey_{Alice}$
2. $PublicKey_{Alice}$
3. $PublicKey_{Bob}$

Вычисляет:

вырабатывает
случайное число

1. ГПСЧ $\rightarrow d_{Alice} \pmod n$
2. $d_{Alice} \times G^{x,y} = Q^{x,y}_{Alice} = (Q^x_{Alice}, Q^y_{Alice})$
3. $RSA_E(PublicKey_{Bob}, Q^{x,y}_{Alice}) =$
 $= Enc_{RSA} Q^{x,y}_{Alice} = (Enc_{RSA} Q^x_{Alice}, Enc_{RSA} Q^y_{Alice})$

 $Enc_{RSA} Q^x_{Alice}, Enc_{RSA} Q^y_{Alice}$
 $Enc_{RSA} Q^x_{Bob}, Enc_{RSA} Q^y_{Bob}$

Вычисляет:

1. $RSA_D(PrivateKey_{Alice}, Enc_{RSA} Q^{x,y}_{Bob}) =$
 $= Q^{x,y}_{Bob} = (Q^x_{Bob}, Q^y_{Bob})$

Имеет:

1. $PrivateKey_{Alice}$
2. $PublicKey_{Alice}$
3. $PublicKey_{Bob}$
4. d_{Alice}
5. $Q^{x,y}_{Bob} = (Q^x_{Bob}, Q^y_{Bob})$

Вычисляет:

2. $d_{Alice} \times Q^{x,y}_{Bob} = Secret^{x,y}_{Alice} = (Secret^x_{Alice}, Secret^y_{Alice})$
3. $SHA512(Secret^x_{Alice}) = hash_{Secret^x_{Alice}} = SessionKey$
4. $SHA512(Secret^y_{Alice}) = hash_{Secret^y_{Alice}}$
5. $AES_E(SessionKey, hash_{Secret^y_{Alice}}) = Enc_{AES} hash_{Secret^y_{Alice}}$

 $Enc_{AES} hash_{Secret^y_{Alice}}$
 $Enc_{AES} hash_{Secret^y_{Bob}}$

Имеет:

1. $PrivateKey_{Bob}$
2. $PublicKey_{Bob}$
3. $PublicKey_{Alice}$

Вычисляет:

вырабатывает
случайное число

1. ГПСЧ $\rightarrow d_{Bob} \pmod n$
2. $d_{Bob} \times G^{x,y} = Q^{x,y}_{Bob} = (Q^x_{Bob}, Q^y_{Bob})$
3. $RSA_E(PublicKey_{Alice}, Q^{x,y}_{Bob}) =$
 $= Enc_{RSA} Q^{x,y}_{Bob} = (Enc_{RSA} Q^x_{Bob}, Enc_{RSA} Q^y_{Bob})$

Вычисляет:

1. $RSA_D(PrivateKey_{Bob}, Enc_{RSA} Q^{x,y}_{Alice}) =$
 $= Q^{x,y}_{Alice} = (Q^x_{Alice}, Q^y_{Alice})$

Имеет:

1. $PrivateKey_{Bob}$
2. $PublicKey_{Bob}$
3. $PublicKey_{Alice}$
4. d_{Bob}
5. $Q^{x,y}_{Alice} = (Q^x_{Alice}, Q^y_{Alice})$

Вычисляет:

2. $d_{Bob} \times Q^{x,y}_{Alice} = Secret^{x,y}_{Bob} = (Secret^x_{Bob}, Secret^y_{Bob})$
3. $SHA512(Secret^x_{Bob}) = hash_{Secret^x_{Bob}} = SessionKey$
4. $SHA512(Secret^y_{Bob}) = hash_{Secret^y_{Bob}}$
5. $AES_E(SessionKey, hash_{Secret^y_{Bob}}) = Enc_{AES} hash_{Secret^y_{Bob}}$

Имеет:

1. $PrivateKey_{Alice}$
2. $PublicKey_{Alice}$
3. $PublicKey_{Bob}$
4. d_{Alice}
5. $Q^{x,y}_{Bob} = (Q^x_{Bob}, Q^y_{Bob})$
6. $SessionKey$
7. $hash_{Secret^y_{Alice}}$

Вычисляет:

$$AES_D(SessionKey, Enc_{AES} hash_{Secret^y_{Bob}}) = ?$$

$$= hash_{Secret^y_{Alice}}$$

если да, имеет: общий с Bob'ом
симметричный $SessionKey$

если нет, то идет подмена
 $SessionKey$, повторить «Шаг 3»



Имеет:

1. $PrivateKey_{Bob}$
2. $PublicKey_{Bob}$
3. $PublicKey_{Alice}$
4. d_{Bob}
5. $Q^{x,y}_{Alice} = (Q^x_{Alice}, Q^y_{Alice})$
6. $SessionKey$
7. $hash_{Secret^y_{Bob}}$

Вычисляет:

$$AES_D(SessionKey, Enc_{AES} hash_{Secret^y_{Alice}}) = ?$$

$$= hash_{Secret^y_{Bob}}$$

если да, имеет: общий с Alice
симметричный $SessionKey$

если нет, то идет подмена
 $SessionKey$, повторить «Шаг 3»

Имеет:

1. $SessionKey$
2. $Message_{Alice}$

Вычисляет:

вырабатывает
случайное число

$$1. \text{ГПСЧ} \rightarrow SecretKeyDS(\text{mod } n)$$

$$2. KeyCheckDS_{Alice} = \\ = CreateKeyCheckDS(SecretKeyDS)$$

$$3. DS_{Message_{Alice}} = \\ = CreateDS(SecretKeyDS, Message_{Alice})$$

$$4. AES_E(SessionKey, Message_{Alice}) = \\ = Enc_{AES} Message_{Alice}$$

Имеет:

1. $SessionKey$
2. $Message_{Bob}$

Вычисляет:

вырабатывает
случайное число

$$1. \text{ГПСЧ} \rightarrow SecretKeyDS(\text{mod } n)$$

$$2. KeyCheckDS_{Bob} = \\ = CreateKeyCheckDS(SecretKeyDS)$$

$$3. DS_{Message_{Bob}} = \\ = CreateDS(SecretKeyDS, Message_{Bob})$$

$$4. AES_E(SessionKey, Message_{Bob}) = \\ = Enc_{AES} Message_{Bob}$$

$Enc_{AES} Message_{Alice}, KeyCheckDS_{Alice}, DS_{Message_{Alice}}$

→

$Enc_{AES} Message_{Bob}, KeyCheckDS_{Bob}, DS_{Message_{Bob}}$

←

Вычисляет:

1. $AES_D(SessionKey, Enc_{AES} Message_{Bob}) = Message_{Bob}$
2. $result = CheckDS(DS_{Message_{Bob}}, Message_{Bob}, KeyCheckDS_{Bob})$

если $result = false$, то:

$ГПСЧ \xrightarrow{80байт(IV+HASH)} Answer_{Alice}$

если $result = true$, то:

1. $SHA512(Message_{Bob}) = hash_{Message_{Bob}}$
2. $AES_E(SessionKey, hash_{Message_{Bob}}) = Answer_{Alice}$

Вычисляет:

1. $AES_D(SessionKey, Enc_{AES} Message_{Alice}) = Message_{Alice}$
2. $result = CheckDS(DS_{Message_{Alice}}, Message_{Alice}, KeyCheckDS_{Alice})$

если $result = false$, то:

$ГПСЧ \xrightarrow{80байт(IV+HASH)} Answer_{Bob}$

если $result = true$, то:

1. $SHA512(Message_{Alice}) = hash_{Message_{Alice}}$
2. $AES_E(SessionKey, hash_{Message_{Alice}}) = Answer_{Bob}$

$Answer_{Alice}$

→

$Answer_{Bob}$

←

Вычисляет:

$AES_D(SessionKey, Answer_{Bob}) = ?$
 $= hash_{Message_{Alice}}$

если да, то сообщение
передалось успешно

если нет, то сообщение не
передалось, повторить «Шаг 4»

Вычисляет:

$AES_D(SessionKey, Answer_{Alice}) = ?$
 $= hash_{Message_{Bob}}$

если да, то сообщение
передалось успешно

если нет, то сообщение не
передалось, повторить «Шаг 4»

4. Свойства, характеризующие безопасность протокола

Свойства, характеризующие безопасность протоколов:

- 1) Аутентификация (не широковещательная)
 - G1 (аутентификация субъекта);
 - G2 (аутентификация сообщения);
 - G3 (защита от повтора)
- 2) Аутентификация при рассылке по многим адреса
 - G4 (неявная скрытая аутентификация получателя);
 - G5 (аутентификация источника)
- 3) Авторизация 3-ей доверенной стороной
 - G6 (авторизация 3-ей доверенной стороной);
- 4) 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)
- 5) Конфиденциальность
 - G12 (конфиденциальность)
- 6) Анонимность
 - G13 (защита идентификатора от прослушивания);
 - G14 (защита идентификатора от других участников)
- 7) Защита от отказа в обслуживании
 - G15 (защита то DDoS);
- 8) Инвариантность
 - G16 (инвариантность отправителя)
- 9) Невозможность отказа от ранее совершенных действий
 - G17 (подотчетность);
 - G18 (доказательство источника);
 - G19 (доказательство получателя)
- 10) 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.

5. 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 асимметричного 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.

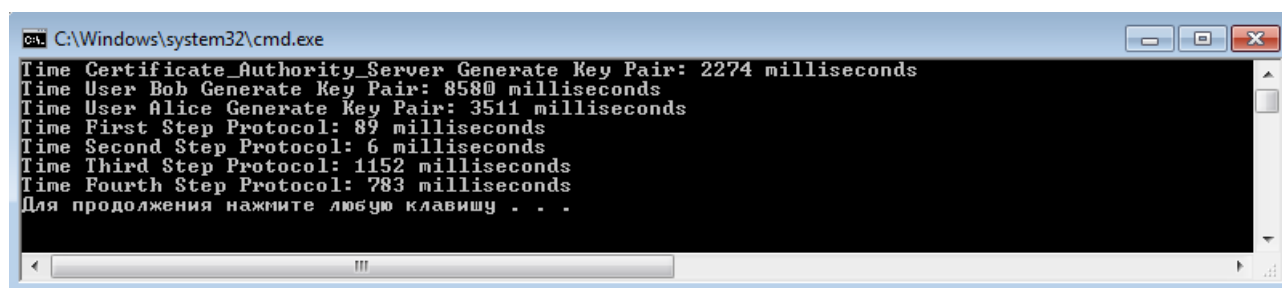


Рис. 1. Результат тестирования реализованного протокола

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.2.

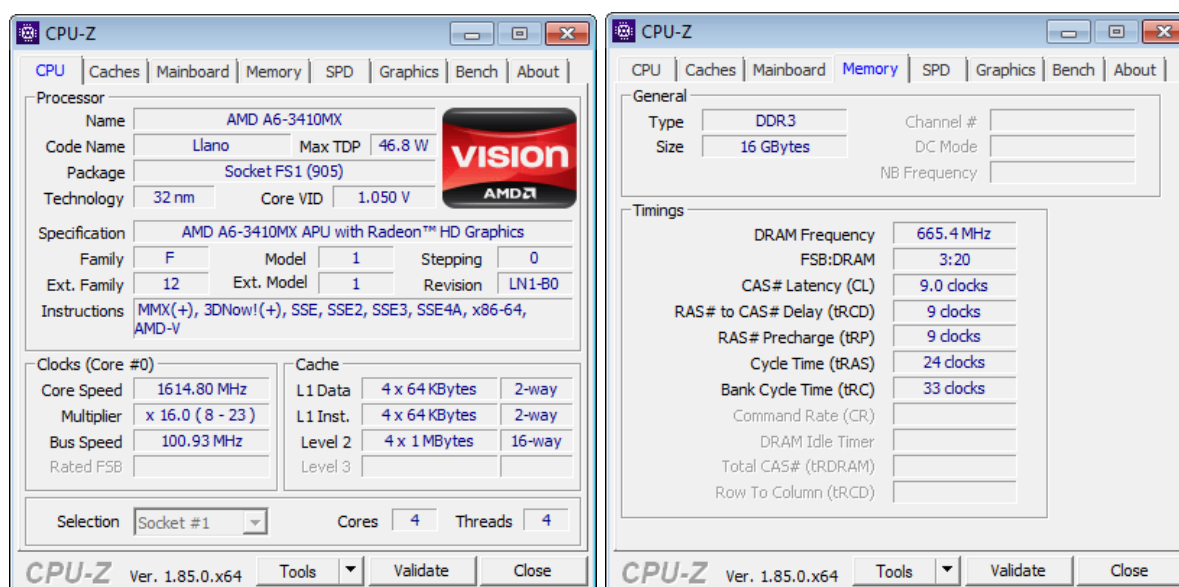


Рис. 2. ЦП AMD A6-3410MX (4 ядра, 4 потока) и ОЗУ

Таким образом, общее время установки соединения (без времени выработки ключей асимметричного 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]
<i>Генерация асимметричной ключевой пары (выполняется до начала протокола)</i>		
СА	Выработка ключевой пары	2,274
Alice	Выработка ключевой пары	3,511
Bob	Выработка ключевой пары	8,580
<i>Шаг 1 «Получение подписанных СА ключей пользователей»</i>		
Alice/Bob	Получение подписанных СА ключей пользователей	0,089
<i>Шаг 2 «Обмен публичными ключами между собеседниками»</i>		
Alice/Bob	Обмен публичными ключами между собеседниками	0,006
<i>Шаг 3 «Генерация сеансового ключа собеседников»</i>		
Alice/Bob	Генерация сеансового ключа собеседников	1,152
<i>Шаг 4 «Обмен сообщениями между собеседниками»</i>		
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-znaniy.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_Parameters();

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* CAsServerRSA = nullptr;
    BIO* CAsServerPublicKey = nullptr;
    BIO* CAsServerPrivateKey = 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_Parameters(){
    //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, '\0');
    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);

    CASServerRSA = RSA_new();
    RSA_generate_key_ex(CASServerRSA, 4096, e, nullptr);

    CASServerPublicKey =
    BIO_new_file("protocol/CA_SERVER/RSA/PUBLIC_KEY/PUBLIC_KEY.pem", "wb");
    PEM_write_bio_RSAPublicKey(CASServerPublicKey, CASServerRSA);
    BIO_free_all(CASServerPublicKey);

    CASServerPublicKey =
    BIO_new_file("protocol/ALICE/FROM/CA_SERVER/RSA/PUBLIC_KEY/PUBLIC_KEY.pem",
"wb");
    PEM_write_bio_RSAPublicKey(CASServerPublicKey, CASServerRSA);
    BIO_free_all(CASServerPublicKey);

    CASServerPublicKey =
    BIO_new_file("protocol/BOB/FROM/CA_SERVER/RSA/PUBLIC_KEY/PUBLIC_KEY.pem", "wb");
    PEM_write_bio_RSAPublicKey(CASServerPublicKey, CASServerRSA);
    BIO_free_all(CASServerPublicKey);

    CASServerPrivateKey =
    BIO_new_file("protocol/CA_SERVER/RSA/PRIVATE_KEY/PRIVATE_KEY.pem", "wb");
    PEM_write_bio_RSAPrivateKey(CASServerPrivateKey, CASServerRSA, nullptr,
    nullptr, 0, nullptr, nullptr);
    BIO_free_all(CASServerPrivateKey);

    BN_free(e);
    RSA_free(CASServerRSA);
}

Certificate_Authority_Server::Certificate_Authority_Server() { };

Certificate_Authority_Server::~Certificate_Authority_Server() { };

void Certificate_Authority_Server::signatureHashPublicKey(vector<uint8_t>*
HashPublicKey, const IUser& User) {
    CASServerRSA = RSA_new();
    CASServerPrivateKey =
    BIO_new_file("protocol/CA_SERVER/RSA/PRIVATE_KEY/PRIVATE_KEY.pem", "rb");
    CASServerPublicKey =
    BIO_new_file("protocol/CA_SERVER/RSA/PUBLIC_KEY/PUBLIC_KEY.pem", "rb");
    CASServerRSA = PEM_read_bio_RSAPrivateKey(CASServerPrivateKey, &CASServerRSA,
    nullptr, nullptr);
    CASServerRSA = PEM_read_bio_RSAPublicKey(CASServerPublicKey, &CASServerRSA,
    nullptr, nullptr);

    auto signHashUserPubKey = new vector<uint8_t>(RSA_size(CASServerRSA), 0);

    RSA_private_encrypt(HashPublicKey->size(), HashPublicKey->data(),
    signHashUserPubKey->data(), CASServerRSA, 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_Parameters();
}

```



```

//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_Parameters();

//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 Message 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;
}

```

}