МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра дифференциальных уравнений и системного анализа

Крипосистема AES

Курсовая работа

Петроченко Виктории
Андреевны

студентки 2 курса,
специальность 1-31 03 09
Компьютерная математика
и системный анализ

Научный руководитель:
кандидат физ.-мат. наук,
доцент Д. Н. Чергинец

МИНСК – 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент _____

1 Тема: Криптосистема AES

2 Срок представления курсовой работы к защите 30.04.2018

3 Исходные данные для научного исследования

3.1. Мао, Венбо Современная криптография = Modern Cryptography : теория и практика / Венбо Мао ; [пер. с англ. и ред. Д. А. Клюшина]. - Москва; Санкт-Петербург; Киев: Вильямс, 2005. - 764с.
3.2. Фергюсон, Нильс Практическая криптография = Practical Cryptography / Нильс Фергюсон, Брюс Шнайер ; [пер. с англ. Н. Н. Селиной ; под ред. А. В. Журавлева]. - Москва; Санкт-Петербург; Киев: Диалектика, 2005. - 422с.
3.3. Смарт, Н. Криптография / Н. Смарт; пер. с англ. С. А. Кулешова под ред. С. К. Ландо. - Москва: Техносфера, 2006. - 525 с.
3.4. Joan Daemen, Vincent Rijmen, The Rijndael Block Cipher AES Proposal, 1999.
3.5. Federal Information Processing Standards Publication 197 (FIPS 197), 2001.

4 Содержание курсовой работы

4.1 Поле GF(p^m). Вычисление обратного относительно умножения.
4.2 Криптосистема AES.
4.3 Учебная реализация криптосистемы.
4.4 Табличная реализация криптосистемы (см. 3.4 пункт 5.2).

Руководитель курсовой работы _____
(подпись, дата, инициалы, фамилия)

Задание принял к исполнению _____
(подпись, дата)

ОГЛАВЛЕНИЕ

стр.

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

AES – (Advanced Encryption Standard) стандарт шифрования.

AddRoundKey() – Преобразование, которое складывает при помощи операции *XOR* каждый байт матрицы состояния с соответствующим байтом развернутого ключа шифрования (ключа раунда).

SubBytes() – Преобразование, которое заменяет байты матрицы состояний при помощи таблицы (S-блока).

InvSubBytes() – Преобразование в процедуре дешифрования, которое является обратным для *SubBytes*().

ShiftRows() – Преобразование, которое выполняет циклический сдвиг строк матрицы состояния.

InvShiftRows() – Преобразование в процедуре дешифрования, которое является обратным для *ShiftRows*().

MixColumns() – Преобразование, которое умножает каждый столбец матрицы состояния на фиксированный многочлен.

InvMixColumns() – Преобразование в процедуре дешифрования, которое является обратным для *MixColumns*().

Rcon() – Массив слов-констант раундов.

RotWord() – Функция, которая выполняет циклическую перестановку внутри 4-байтного слова.

SubWord() – Функция, которая заменяет байты 4-байтного слова.

XOR – Операция исключающего ИЛИ.

Nk – Количество 32-битных слов, которые составляют ключ шифрования.

Nr – Количество раундов.

Nb – Количество столбцов (32-битных слов) в матрице состояния.

$\oplus$ – Операция исключающего ИЛИ.

$\otimes$ – Операция перемножения двух многочленов, степень каждого из которых меньше 4, по модулю $x^4 + 1$.

$\bullet$ – Операция умножения в конечном поле.

S-блок – Таблица замен байтов

Байт – Последовательность из 8 бит

Бит – Двоичная цифра, которая может принимать значение 0 или 1.

Блок	–	Последовательность бит, в виде которой представляются вход, выход, матрица состояния и ключ раунда.
Длина блока	–	Количество содержащихся в блоке бит.
Ключ шифрования	–	Секретная информация, используемая при зашифровании/дешифровании информации.
Ключи раунда	–	Числа, которые получаются из ключа шифрования с помощью процедуры расширения ключа шифрования.
Массив	–	Совокупность однотипных пронумерованных объектов
Матрица состояния	–	Промежуточный результат шифрования, который представляется в виде прямоугольной матрицы байт, которая имеет 4 строки и Nb столбцов.
Открытый текст	–	Входные данный процедуры операции шифрования или выходные данные операции дешифрования.
Процедура дешифрования	–	Серия преобразований, которая использует ключ шифрования для преобразования шифр-текста в открытый текст.
Процедура расширения ключа	–	Процедура, которая используется для создания ключей раундов из ключа шифрования.
Процедура шифрования	–	Серия преобразований, которая использует ключ шифрования для преобразования открытого текста и шифр-текст.
Слово	–	Совокупность из 32 бит, которая может обрабатываться как единое целое, либо как массив из 4 байт.
Шифр-текст	–	Результат операции шифровании. Выходные данный процедуры шифрования или входные данные процедуры дешифрования

ВВЕДЕНИЕ

С древнейших времен криптография использовалась для защиты военной и дипломатической связи. Необходимость защиты правительственной связи вполне очевидна, и до недавнего времени широкое применение криптографии было почти исключительно правом государства. В настоящее время большинство правительств контролирует или сами исследования в этой области, или, по крайней мере, производство криптографического оборудования и программного обеспечения. В Республике Беларусь деятельность по технической и (или) криптографической защите информации является лицензируемой. С началом информационного века возникла срочная необходимость использования криптографии в частном секторе. Сегодня огромное количество конфиденциальной информации (такой, например, как персональные данные, истории болезней, юридические документы, данные о финансовых операциях) передается между ЭВМ по обычным линиям связи. Поэтому возникает необходимость обеспечения секретности и подлинности подобной информации.

Криптология (происходит от греческих корней, означающих "тайный" и "слово") – наука о шифровании и дешифровании.

Шифрование – метод, используемый для преобразования исходных данных в зашифрованный текст (криптограмму) для того, чтобы они могли быть прочитаны только пользователем, обладающим соответствующим ключом шифрования для расшифрования содержимого.

Криптология делится на две части: криптографию (шифрование) и криптоанализ. Киптография занимается разработкой методов обеспечения секретности и (или) аутентичности (подлинности) сообщений. Криптоанализ предназначен для решения обратной задачи – раскрытия (взлома) шифра с целью получения возможности несанкционированного чтения зашифрованного сообщения или осмысленной подделки такого сообщения. Кроме того, криптоанализ применяют при исследовании шифров с целью улучшения их свойств, например, криптографической стойкости.

В современных информационных системах ключ – это двоичная строка, которая может быть интерпретирована набором символом другого алфавита.

Цель криптографической системы, чаще всего, заключается в том, чтобы зашифровать осмысленный исходный текст, получив в результате зашифрованный текст бессмысленный с точки зрения постороннего наблюдателя. Получатель, которому он предназначен, должен быть способен расшифровать этот криптотекст, восстановив, таким образом, соответствующий ей открытый текст. При этом противник (криптоаналитик) должен быть неспособен раскрыть исходный текст.

AES(Advanced Encryption Standard) – стандарт шифрования, основанный на алгоритме Rijndael, разработанном двумя бельгийскими криптографами разработанного двумя Бельгийскими криптографами Винсентом Рейменом и Джоаном Дейменом.

Алгоритм Rijndael представляет собой итеративный блочный шифр с возможностью выбора длины блока и длины ключа 128, 192 или 256 бит. Для стандарта AES длина блока только 128 бит, а длина ключа 128, 192 или 256 бит.

В 2003 году Агентство национальной безопасности США признало шифр AES достаточно надежным для защиты сведений, составляющих государственную тайну (classified information). Для защиты информации до уровня SECRET включительно разрешено использовать ключи длиной 128 бит, а для уровня TOP SECRET – ключи длиной 192 и 256 бит.

Целью настоящей работы является анализ криптосистемы AES с последующей реализацией алгоритмических преобразований шифрования/расшифрования в виде статической библиотеки классов на языке C++. Результаты работы будут верифицированы на данных, определенных в стандарте.

1. КЛАССИФИКАЦИЯ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ И ХАРАКТЕРИСТИКА КРИПТОСИТЕМЫ AES

1.1. Классификация криптографических алгоритмов

Криптографические алгоритмы можно разделить на три категории в зависимости от числа ключей, которые они используют:

– бесключевые алгоритмы – алгоритмы, которые не используют какие-либо ключи;

– одноключевые алгориты – алгоритмы, которые работают с одним секретным ключом;

– двухключевые алгоритмы – алгоритмы, которые на различных этапах применяют два вида ключей: секретный и открытый.

Каждая категория алгоритмов также имеет свою классификацию, одна из которых показана на рисунке 1.1.

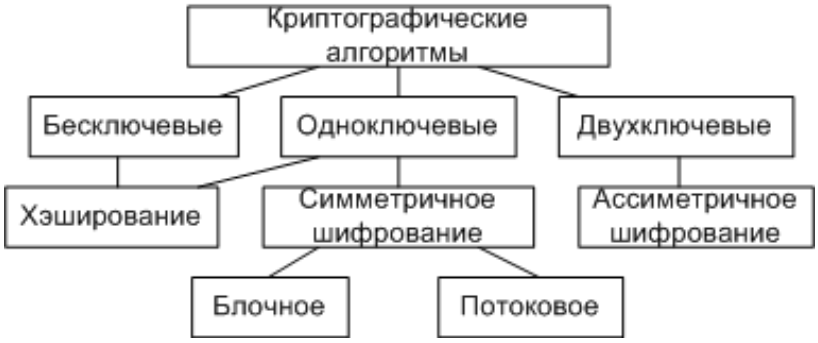

Рисунок 1.1. Классификация криптографических алгоритмов

Хеширование – это сворачивание данных переменной длины в последовательность фиксированного размера. Иными словами, это контрольное суммирование данных.

Симметричное шифрование – это вид шифрования, который использует один и тот же ключ для зашифровывания и расшифровывания данных. Симметричное шифрование можно разделить на две категории: блочное и потоковое.

Основной идеей *блочного шифрования* является разбиение информации на блоки фиксированной длины. Далее ко всем блокам применяются функции шифрования, таким образом данные зашифровываются. Текущее поколение блочных шифров работает с блоками текста, имеющие длину 128 бит: на входе шифр принимает 128-битовый открытый текст и на выходе выдает 128-битовый шифрованный текст. Блочные шифры являются обратимыми, т.е. существует функция дешифрования, которая получает на входе из зашифрованный текст и на выходе выдает открытый.

Потоковое шифрование применяют тогда, когда информацию невозможно разбить на блоки, например, есть какой-то поток данных, каждый символ которого необходимо зашифровать, не дожидаясь остальных данных, чтобы сформировать блок. Потоковое шифрование шифрует данные по одному биту за такт шифрования.

Ассиметричное шифрование – это вид шифрования, который использует два вида ключей для зашифровывания и расшифровывания данных. В качестве ключа для зашифровывания информации берется открытый ключ, а для расшифровывания – закрытый.

1.2. Характеристика криптосистемы AES

Криптосистема AES – это симметричный блочный шифр, который может обрабатывать данные блоками по 128 бит, используя при этом ключи шифрования длинной 128, 192 и 256 бит. Другие длины ключа шифрования в стандарте не предусмотрены.

На входе алгоритм AES получает информацию – последовательность из 128 бит, и на выходе выдает зашифрованную информацию, которая также будет представлена в виде 128-битовой последовательности. Как говорилось выше, ключи шифрования – это последовательности из 128, 192 или 256 бит.

Биты в пределах всех последовательностей пронумерованы. Нумерация начинается справа с нуля и заканчивается на единицу меньше длины последовательности. Номер бита i называется индексом бита. Таким образом в зависимости от блока и длины ключа индекс может принадлежать следующим диапазонам: $0 \le i < 128$, $0 \le i < 192$, $0 \le i < 256$.

1.2.1. Байты

В алгоритме AES базовыми элементами являются байты – последовательности из 8 бит, которые обрабатывается как единое целое. Что касается последовательностей, которые представляют вход и выход алгоритма, а также ключей шифрования, то они обрабатываются как массив байт. Массивы байт получаются за счет деления всей последовательности бит на группы из 8 рядом стоящих бит. Байты в сформированном массиве обозначаются как a_n, или $a[n]$, где a – это вход, выход или ключ шифрования, n – количество байт в последовательности. Таким образом n может принимать следующие значения:

- $0 \le n < 16$, когда длина ключа равна 128 бит;
- $0 \le n < 24$, когда длина ключа равна 192 бита;
- $0 \le n < 32$, когда длина ключа равна 256 бит;
- $0 \le n < 16$, когда длина блока равна 128 бит.

В алгоритме AES все значения байта представляются в виде последовательности из 8 отдельных бит, которые составляют этот байт. Эта последовательность бит заключается в скобки и будет иметь следующий вид: $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Все байты рассматриваются как элементы конечного поля и представляются в виде многочленов:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (1.1)$$

Например, последовательность бит {01100111} определяет конкретный многочлен $x^6 + x^5 + x^2 + x + 1$.

В алгоритме AES последовательность байт указывается в шестнадцатеричной форме. Все байты разделяются на группы из 4 бит, и каждая группа записывается одним символом, согласно шестнадцатеричной форме записи (таблица 1.1).

Комбинация бит	Символ	Комбинация бит	Символ
0000	0	1000	8
0001	1	1001	9
0010	2	1010	a
0011	3	1011	b
0100	4	1100	c
0101	5	1101	d
0110	6	1110	e
0111	7	1111	f

Таблица 1.1. Шестнадцатеричное представление комбинации бит

Таким образом последовательность бит {01100111} можно записать как {67}.

Иногда в алгоритмах шифрования и дешифрования информации будет использоваться дополнительный бит b_8, который располагается слева от 8-битного байта. Присутствие дополнительного бита b_8 обозначается дописыванием перед 8-битным байтом {01}. Например 9-битная последовательность {110110011} будет записана как {01}{b3}.

1.2.2. Массивы байт

Массивы байт представляются в следующем виде: $a_0 a_1 a_2 ... a_{14} a_{15}$.

Входная 128-битная последовательность $input_0, input_1, input_2, ..., input_{127}$ образует массивы байт и упорядоченные биты в этих байтах следующим образом:

$$a_0 = \{input_0, input_1, ..., input_7\}$$

$$a_1 = \{input_8, input_9, ..., input_{15}\}$$

…

$$a_{15} = \{input_{120}, input_{121}, ..., input_{127}\}$$

Этот образец можно расширить на более длинные, такие как 192- и 256-битная, последовательности. Таким образом в общем виде байт можно записать так:

$$a_n = \{input_{8n}, input_{8n+1}, ..., input_{8n+7}\} \quad (1.2)$$

Объединяя изложенное в подразделах 1.2.1 и 1.2.2, рисунок 1.2 показывает нумерацию бит в пределах каждого байта.

Входная последовательность бит	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
Номер байта	0								1								…
Номера битов в байте	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…

Рисунок 1.2. Нумерация байт и бит

1.2.3. Матрица состояния

Внутри алгоритма AES выполняются операции над двумерным массивом байт, который называется **матрицей состояния**.

Матрица состояния образуется четырьмя строками, каждая из которых содержит *Nb* байт (*Nb* – длина блока, деленная на 32). В матрице состояния s, каждый отдельный байт имеет два индекса: номер строки r ($0 \le r < 4$) и номер столбца c ($0 \le c < Nb$), что позволяет обращаться к отдельному байту матрицы состояния либо как $s_{r,c}$, либо $s[r,c]$. Для данного стандарта длина блока равна 128, а значит $Nb = 4$.

В начале любой из процедур шифрования или дешифрования, на вход поступает массив байт $in_0 in_1 in_2 ... in_{14} in_{15}$, который копируется в матрицу состояния, как показано на рисунке 1.3. После чего процедуры шифрования и дешифрования изменяют матрицу состояния, и финальный результат копируется в выходной массив байт $out_0 out_1 out_2 ... out_{14} out_{15}$.

Рисунок 1.3. Матрица состояния, вход и выход

Таким образом, входной массив in в начале процедур шифрования или дешифрования копируется в матрицу состояния в соответствии с выражением (1.3). В конце процедур шифрования или дешифрования матрица состояния копируется в выходной массив out в соответствии с выражением (1.4).

$$s[r,c] = in[r+4c], 0 \le r < 4, 0 \le c < Nb \tag{1.3}$$

$$out[r+4c] = s[r,c], 0 \le r < 4, 0 \le c < Nb \tag{1.4}$$

1.2.4. Матрица состояния в виде массива столбцов

Четыре байта в каждом столбце матрицы состояния образуют 32-битные слова. Для этих четырех байт номер строки r – индекс в пределах каждого слова. Следовательно, можно сказать, что матрица состояния – это одномерный массив 32-битных слов-столбцов w_0, w_1, w_2, w_3, для которых номер столбца c – это индекс в этом массиве. Например, матрица состояния может рассматриваться как массив из четырех слов следующим образом:

$$w_0 = s_{0,0} s_{1,0} s_{2,0} s_{3,0}$$

$$w_1 = s_{0,1} s_{1,1} s_{2,1} s_{3,1}$$

$$w_2 = s_{0,2} s_{1,2} s_{2,2} s_{3,2}$$

$$w_3 = s_{0,3} s_{1,3} s_{2,3} s_{3,3}$$

2. МАТЕМАТИЧЕСКОЕ ОБОСНОВАНИЕ АЛГОРИТМА AES

2.1. Элементы теории конечных полей

В криптосистеме AES все арифметические операции для работы с информацией при кодировании и декодировании данных выполняются в конечных полях. Конечное поле, или поле Галуа, - это такое поле, которое состоит из конечного числа элементов.

Поле – это непустое множество, которое обладает следующими свойствами:

– определены операции сложения и умножения

– для любых трех элементов a,b,c, принадлежащих полю, выполняются свойства ассоциативности, дистрибутивности и коммутативности

$$a+b=b+a$$

$$ab=ba$$

$$a+(b+c)=(a+b)+c$$

$$a(bc)=(ab)c$$

$$a(b+c)=ab+ac$$

– в поле должны существовать элементы $0,1$, обратный относительно сложения $-a$ и, для любого $a\neq 0$, обратный элемент относительно умножения a^{-1}

$$0+a=a$$

$$a+(-a)=0$$

$$0a=0$$

$$1a=a$$

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

– все ненулевые элементы конечного поля могут быть представлены в степени некоторого фиксированного элемента поля w, который называется примитивным элементом

– Поле, состоящие из конечного числа элементов, называется конечным полем или полем Галуа. Конечное поле обозначается $GF(L)$, где L – число элементов конечного поля.

2.1.1. Поле GF(p)

Простейшие поля получаются следующим образом. Выберем простое число p, тогда поле $GF(p)$ будет содержать целые числа от 0 до $p-1$, т.е. $GF(p)=\{0,1,...,p-1\}$, и все арифметические операции в этом поле будут выпол-

ниться по модулю p. Т.е. через 0 будут обозначаться все числа, дающие при делении на p остаток 0, через 1 – все числа, дающие при делении на p остаток 1 и т.д.

Что касается сравнения двух элементов поля $GF(p)$, то эта операция также выполняется по модулю p. А именно, $a = b$ означает, что $a - b$ делится на p, и говорят, что a сравнимо с b по модулю p, и обозначается $a \equiv b(\mathrm{mod}\, p)$.

Пример 2.1.

Пусть поле образовано пятью элементами, т.е. $p = 5$ и $GF(p) = GF(5) = \{0,1,2,3,4\}$. Рассмотрим арифметические операции в этом поле.

$$3 + 2 = 5(\mathrm{mod}\, 5) = 0$$

$$1 - 4 = -3 = 0 - 3 = 5 - 3(\mathrm{mod}\, 5) = 2$$

$$4 * 3 = 12 = 2(\mathrm{mod}\, 5)$$

$$\frac{3}{2} = \frac{(0+3)}{2} = \frac{(5+3)(\mathrm{mod}\, 5)}{2} = 4$$

Выберем, к примеру, число 2 в качестве примитивного элемента, тогда все ненулевые элементы поля могут быть образованы как 2 в какой-то степени:

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8 = 3(\mathrm{mod}\, 5)$$

2.1.2. Поле GF(p^n)

Поле может быть образовано из элементов $L = p^n$, где p – простое число, а n – натурально число. Поле $GF(p^n)$ состоит из элементов представленных в виде многочленов степени не более $n-1$ с коэффициентами из поля $GF(p)$.

Операция сложения в поле $GF(p^n)$ происходит по обычным правилам сложения многочленов, при этом операция приведение подобных членов происходит по модулю p.

Для определения операции умножения, необходимо ознакомится с понятием неприводимого многочлена. Неприводимый многочлен – это многочлен, который не является произведением двух многочленов меньшей степени, т.е. тот, который делится только на себя либо на единицу. Примитивный многочлен является неприводимым многочленом. Выберем неприводимый многочлен $\varphi(x)$ степени n. Тогда под операцией умножения двух многочленов в поле $GF(p^n)$ понимают обычное перемножения двух многочленов, при этом если в результате умножения получился многочлен степени больше или равной n, то результат берется по модулю $\varphi(x)$.

Пример 2.2.

Пусть поле образовано из $L = 2^4$ элементов, т.е. $p = 2, n = 4$ и $GF(L) = GF(p^n) = GF(2^4) = \{0,1,x,x+1,x^2,x^2+1,x^2+x,x^2+x+1,x^3,...,x^3+x^2+x+1\}$. Выберем примитивный многочлен над $GF(2^4)$ — $\varphi(x) = x^4 + x + 1$.

Рассмотрим арифметические операции в этом поле.

$(x^2+1) \oplus (x^3+x^2+x+1) = x^3+x^2+x^2+x+1+1 =$ (т.к. сложение по модулю $p = 2$, то $x^2 + x^2 = 2x^2 = 0$, $1+1 = 2 = 0$) $= x^3 \oplus x$ (далее сложение по модулю 2 будет обозначаться $\oplus$).

$(x \oplus 1)(x^2 \oplus x \oplus 1) = x^3 \oplus x^2 \oplus x \oplus x^2 \oplus x \oplus 1 = x^3 \oplus 1$

$(x^2 \oplus 1)(x^3 \oplus 1) = x^5 \oplus x^2 \oplus x^3 \oplus 1 =$ (т.к. результат умножения многочлен степени 5, а максимальная степень многочлена в заданном поле равна 3, то результат умножения берем по модулю $\varphi(x)$) $= x^5 \oplus x^3 \oplus x^2 \oplus 1 (\mathrm{mod}\, x^4 \oplus x \oplus 1) = x^3 \oplus x \oplus 1$.

Выберем примитивный многочлен в поле $GF(2^4)$: $w = x + 1$. Все ненулевые элементы поля можно представить как w в какой-то степени.

$w^0 = 1$

$w^1 = x \oplus 1$

$w^2 = (x \oplus 1)^2 = x^2 \oplus 1$

$w^3 = (x \oplus 1)^3 = x^3 \oplus x^2 \oplus x \oplus 1$

…

$w^{14} = (x \oplus 1)^{14} = x^{14} \oplus x^{12} \oplus x^{10} \oplus ... \oplus x^4 \oplus x^2 \oplus 1 (\mathrm{mod}\, x^4 \oplus x \oplus 1) = x^3 \oplus x^2 \oplus 1$

$w^{15} = (x \oplus 1)^{15} = x^{15} \oplus x^{14} \oplus x^{13} \oplus ... \oplus x^4 \oplus x^3 \oplus x^2 \oplus x \oplus 1 (\mathrm{mod}\, x^4 \oplus x \oplus 1) = 1$

Таким образом можно представить операцию умножения в поле $GF(2^4)$ следующим образом:

$$(x \oplus 1)(x^2 \oplus 1) = w \cdot w^2 = w^3 = x^3 \oplus x^2 \oplus x \oplus 1$$

В поле существуют понятия обратного элемента относительно сложения и обратного элемента относительно умножения. В рассматриваемом поле $GF(p^n) = GF(2^4)$ операция сложения происходит по модулю $p = 2$, следовательно для многочлена $a(x)$, принадлежащему этому полю, обратным элементов относительно сложения является такой же многочлен $a(x)$, т.е. $a(x) + a(x) = 2a(x) = 0$.

Для нахождения обратного элемента относительно умножения используется *расширенный алгоритм Евклида*. На вход подаются два многочлена $\varphi(x)$ и $a(x)$, $\deg \varphi(x) \geq \deg a(x)$. Проверяется существование обратного элемента: если

$НОД(\varphi(x), a(x)) = 1$, то обратный элемент для $a(x)$ существует. Формируется матрица:

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Делим многочлен $\varphi(x)$ на $a(x)$ с остатком, пусть $q(x)$ – частное, $r(x)$ – остаток, т.е. $\varphi(x) = q(x)a(x) + r(x)$. Если $r(x) = 0$, то для элемента $a(x)$ обратным относительно умножения является $a^{-1}(x) = M_{2,2}$. Если $r(x) \neq 0$, то переопределяем переменные: $\varphi(x) = a(x)$, $a(x) = r(x)$ и

$$M = M\begin{pmatrix} 0 & 1 \\ 1 & q(x) \end{pmatrix}.$$

Делим новое $\varphi(x)$ на новое $a(x)$ с остатком, до тех пор, пока не выполнится сравнение $r(x) = 0$.

Пример 2.3.

Найдем обратный относительно умножения к элементу $a(x) = x^2 \oplus x \oplus 1$ из поля $GF(2^4)$ при помощи расширенного алгоритма Евклида. В качестве $\varphi(x)$ берется примитивный многочлен над $GF(2^4)$, в нашем случаем $\varphi(x) = x^4 \oplus x \oplus 1$.

Проверяем существование обратного элемента относительно умножения для $a(x)$:

$$НОД(\varphi(x), a(x)) = НОД(x^4 \oplus x \oplus 1, x^2 \oplus x \oplus 1) = 1$$

– обратный элемент существует.

Делим $\varphi(x)$ на $a(x)$ с остатком, получаем:

$$x^4 \oplus x \oplus 1 = (x^2 \oplus x)(x^2 \oplus x \oplus 1) \oplus 1$$

т.е. $q(x) = x^2 \oplus x$, $r(x) = 1$. Так как $r(x) \neq 0$, то переопределяем переменные:

$$\varphi(x) = x^2 \oplus x \oplus 1$$

$$a(x) = 1$$

$$M = M\begin{pmatrix} 1 & 0 \\ 0 & x^2 \oplus x \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & x^2 \oplus x \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & x^2 \oplus x \end{pmatrix}.$$

Делим $\varphi(x)$ на $a(x)$ с остатком, получаем:

$$x^2 \oplus x \oplus 1 = (x^2 \oplus x \oplus 1) \cdot 1$$

т.е. $q(x) = x^2 \oplus x \oplus 1$, $r(x) = 0$. Так как $r(x) = 0$, то обратный к $a(x) = x^2 \oplus x \oplus 1$ относительно умножения – $a^{-1}(x) = x^2 \oplus x$.

Обратный элемент относительно умножения можно также находить, используя факт, что любой ненулевой элемент поля можно представить в виде примитивного элемента поля в какой-то степени.

Пример 2.4.

Найдем обратный относительно умножения к элементу $a(x) = x^2 \oplus x \oplus 1$ из поля $GF(2^4)$, используя примитивный элемент поля.

Пусть $w = x \oplus 1$ – примитивный многочлен в поле $GF(2^4)$, тогда $a(x) = x^2 \oplus x \oplus 1 = w^{10}$. Обратным относительно умножения к многочлену $a(x)$ является многочлен $w^5 = x^2 \oplus x$, так как $w^{10} \cdot w^5 = w^{15} = 1$.

2.2. Поле GF(2^8)

Так как в алгоритме AES базовыми элементами являются байты – последовательности из 8 бит, то все байты можно рассматривать как элементы конечного поля. Т.е. последовательность из 8 бит можно представить, как многочлен поля $GF(p^n) = GF(2^8)$.

Пример 2.5.

Пусть есть байт a. Его можно представить следующим образом:

$a = \{10010111\}$	(двоичная запись)
$a = \{97\}$	(шестнадцатеричная запись)
$a = x^7 \oplus x^4 \oplus x^2 \oplus x \oplus 1$	(запись в виде многочлена)

При представлении байт в виде многочленов операции сложения и умножения выполняются аналогично операциям сложения и умножения многочленов конечного поля $GF(p^n)$. Сложение выполняется по модулю $p = 2$, умножение выполняется по модулю неприводимого многочлена $\varphi(x)$. Для алгоритма AES неприводимым многочленом является многочлен $m(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ или $m(x) = \{01\}\{1b\}$ в шестнадцатеричной записи.

Пример 2.6.

Пусть есть 2 байта: $a = \{01010111\}, b = \{10000011\}$. Покажем операции сложения и умножения бит.

$a \oplus b = \{57\} \oplus \{83\} = \{01010111\} \oplus \{10000011\} = \{11010100\} = \{d4\}$

$a \otimes b = \{57\} \otimes \{83\} = \{01010111\} \otimes \{10000011\} = \{10101101111001\}$ (т.к. количество получившихся бит превышает 1 байт, следовательно, необходимо взять результат по модулю $m = \{100011011\}$) $= \{10101101111001\} (\mathrm{mod} 100011011) = \{11000001\} = \{c1\}$.

При реализации алгоритмов шифрования и дешифрования в AES используется понятие обратного элемента. Обратный байт относительно сложения и относительно умножения находится по методам, описанным в подпункте 2.1.2.

2.3. Многочлены с коэффициентами, принадлежащими полю GF(2^8)

Раундовые преобразования в алгоритме AES оперируют 32-битными словами. Четырехбайтному слову может быть поставлен в соответствие многочлен $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ с коэффициентами из конечного поля $GF(2^8)$. Многочлен $a(x)$ будет представляться в виде слова как $[a_0, a_1, a_2, a_3]$. Необходимо обратить внимание, что коэффициенты многочлена $a(x)$ сами являются элементами конечного поля, т.е. байтами, а не битами, как в определении элементов конечного поля.

Определим второй четырехчленный многочлен $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$. Покажем операции сложения и умножения многочленов.

Сложение двух многочленов с коэффициентами из $GF(2^8)$ равносильно операции сложения многочленов с приведением подобных членов в поле $GF(2^8)$, т.е. путем сложения коэффициентов, принадлежащих конечному полю, при одинаковых степенях x. Таким образом, операции сложения соответствует операция XOR между соответствующими байтами каждого слова:

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

Умножение – более сложная операция, которая выполняется в два шага:

1. Многочлен-произведение алгебраически раскрывается и группируются одинаковые степени;
2. Приведение многочлена-произведения к модулю, задаваемому многочленом степени 4.

Предположим мы перемножаем два многочлена $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ и $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$. Результатом умножения будет многочлен

$$a(x)b(x) = a_3b_3x^6 + (a_3b_2 \oplus a_2b_3)x^5 + (a_3b_1 \oplus a_2b_2 \oplus a_1b_3)x^4 + (a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3)x^3 +$$
$$+ (a_2b_0 \oplus a_1b_1 \oplus a_0b_2)x^2 + (a_1b_0 \oplus a_0b_1)x + a_0b_0 =$$
$$= c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = c(x).$$

Для того, чтобы результат умножения можно было представить в виде 4-байтного слова, необходимо взять результат $c(x)$ по модулю многочлена не более 4 степени. Авторы шифра выбрали многочлен $x^4 + 1$, для которого справедливо $x^i \bmod (x^4 + 1) = x^{i \bmod 4}$. Таким образом результатом $d(x)$ умножения $\otimes$ двух многочленов $a(x)$ и $b(x)$ по модулю $x^4 + 1$ будет многочлен:

$$d(x) = a(x) \otimes b(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0,$$

где:

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$
$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$$
$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$$
$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$$

В матричной форме это может быть записано следующим образом

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad (2.1)$$

Так как многочлен $x^4 + 1$ является приводимым ($x^4 + 1 = (x^2 + 1)(x^2 + 1)$) в поле $GF(2^8)$, то умножение на фиксированный четырехчленный многочлен не всегда обратимо. Однако, в алгоритме AES используется такой многочлен, который имеет обратный, а именно:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$
$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

Так же в алгоритме AES используется еще один многочлен x^3, т.е. значение его коэффициентов $a_0 = a_1 = a_2 = \{00\}$ и $a_3 = \{01\}$. Результатом его использования согласно выражению (2.1) является выходное слово, которое получилось благодаря перемешиванию байтов входного слова. А именно: последовательность байт $[b_0, b_1, b_2, b_3]$ будет преобразована в последовательность $[b_1, b_2, b_3, b_0]$.

3. ОПИСАНИЕ АЛГОРИТМА AES

В алгоритме AES входной блок, выходной блок и матрица состояния содержат 128 бит (16 байт). Следовательно количество столбцов матрицы состояния (количество 32-битных слов) – $Nb = 4$.

В алгоритме AES длина ключа шифрования K может быть равна только 128, 192 или 256 бит. Длина ключа обозначается переменной Nk, которая может принимать значения 4, 6 и 8, таким образом отражая количество столбцов (количество 32-битных слов) в ключе шифрования.

В алгоритме AES количество раундов выполнения алгоритма шифрования или дешифрования зависит от длины ключа. Количество раундов обозначается переменной Nr, которая может принимать следующие значения: $Nr = 10$, при $Nk = 4$; $Nr = 12$, при $Nk = 6$; $Nr = 14$, при $Nk = 8$.

Длина ключа (в битах)	Длина блока (Nb слов)	Длина ключа (Nk слов)	Количество раундов (Nr)
128	4	4	10
192	4	6	12
256	4	8	14

Таблица 3.1. Комбинации длины блока, длины ключа и количества раундов

3.1. Процедура шифрования

В начале процедуры шифрования входные данные копируются в матрицу состояния по правилам, описанным в подразделе 1.2.3. Ключ шифрования копируется по тем же правилам, однако, так как длина ключа шифрования может принимать значения 128, 192 и 256 бит, то количество столбцов при записи ключа шифрования может быть 4, 6 и 8 соответственно.

Раунд шифрования состоит из четырех различных преобразований матрицы состояния:

1. *SubBytes*() – преобразование, которое заменяет байты матрицы состояний при помощи таблицы (S-блока);
2. *ShiftRows*() – преобразование, которое выполняет циклический сдвиг строк матрицы состояния;
3. *MixColumns*() – преобразование, которое умножает каждый столбец матрицы состояния на фиксированный многочлен $a(x)$;
4. *AddRoundKey*() – преобразование, которое складывает при помощи операции *XOR* каждый байт матрицы состояния с соответствующим байтом развернутого ключа шифрования (ключа раунда).

```
void CEncoder::Encrypt()
{
  AddRoundKey(0);
  for(int i=1; i<=_Nr;i++)
  {
        while(i<_Nr)
        {
              SubBytes();
              ShiftRows();
              MixColumns();
              AddRoundKey(i);
              i++;
        }
        SubBytes();
        ShiftRows();
        AddRoundKey(i);
  }
}
```

Рисунок 3.1. Код процедуры шифрования на языке C++

3.1.1. Преобразование *SubBytes()*

Преобразование *SubBytes*() выполняет замену байт матрицы состояния при помощи таблицы (*S*-блока), при этом каждый байт обрабатывается отдельно, не зависимо от других.

Таблица (*S*-блок) формируется двумя действиями:

1. Нахождением обратного элемента относительно умножения в поле $GF(2^8)$, при этом элемент $\{00\}$ переходит сам в себя;
2. Применением преобразования над $GF(2)$, определенным следующим образом:

$$b_i^{'} = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i,\ 0 \le i < 8,$$

где b_i – i-тый бит байта, c_i – i-тый бит байта $c = \{63\} = \{01100011\}$.

Другими словами, суть преобразования может быть описана в матричной форме:

$$\begin{bmatrix} b_0^{'} \\ b_1^{'} \\ b_2^{'} \\ b_3^{'} \\ b_4^{'} \\ b_5^{'} \\ b_6^{'} \\ b_7^{'} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

На рисунке 3.2. показано выполнение преобразования *SubBytes*() над матрицей состояния.

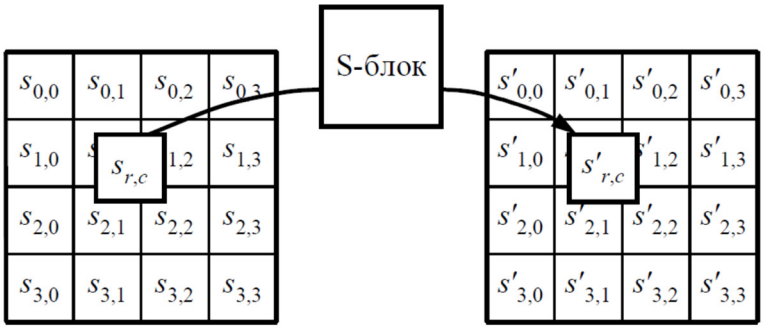

Рисунок 3.2. Преобразование *SubBytes*() изменяет каждый байт матрицы состояния с помощью *S*-блока

В преобразовании *SubBytes*() *S*-блок представляется в шестнадцатеричном виде (таблица 3.2.). Логика работы *S*-блока при преобразовании байта {*xy*} отражена в таблице 3.2. Например, преобразование байта {53} находится на пересечение строки с номером 5 и столбца с номером 3 – {*ed*}.

		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

Таблица 3.2. *S*-блок: значение замен для байта {*xy*}

3.1.2. Преобразование *ShiftRows()*

Преобразование *ShiftRows*() выполняет циклический сдвиг байт в строках матрицы состояния на различное число смещений:

- первая строка с номером $r = 0$ не сдвигается;
- вторая строка с номером $r = 1$ сдвигается на один байт влево;
- третья строка с номером $r = 2$ сдвигается на два байта влево;
- четвертая строка с номером $r = 3$ сдвигается на три байта влево.

Рисунок 3.3. иллюстрирует преобразование *ShiftRows*().

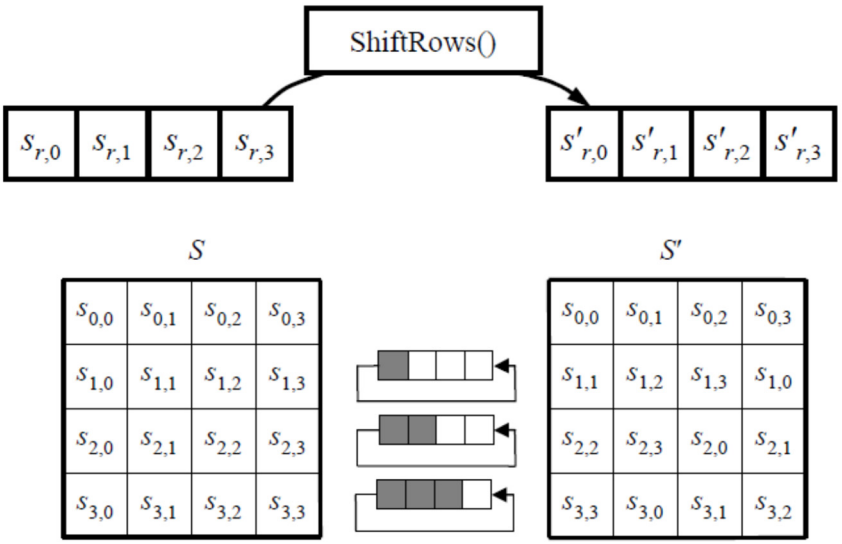

Рисунок 3.3. Преобразование *ShiftRows*() циклически сдвигает три последние строки матрицы состояния

3.1.3. Преобразование *MixColumns()*

Преобразование *MixColumns*() обрабатывает каждый столбец матрицы состояния. Каждый столбец рассматривается как многочлен над полем $GF(2^8)$ и умножаются на многочлен $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ по модулю $x^4 + 1$. Это может быть представлено в матричном виде следующим образом:

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

В результате умножения байты столбца изменяются в соответствии со следующими выражениями:

$$s'_{0,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{1,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

Рисунок 3.4. иллюстрирует преобразование *MixColumns*().

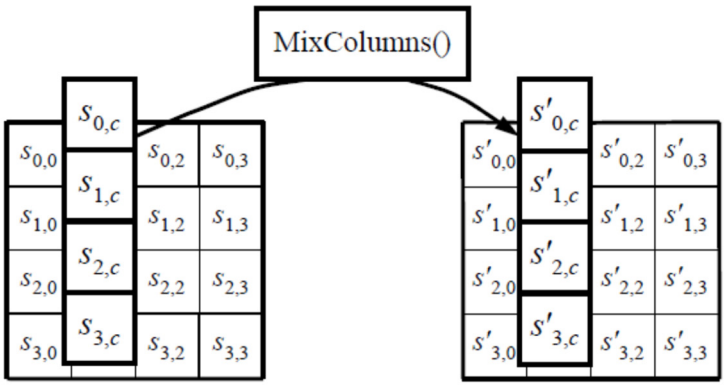

Рисунок 3.4. Преобразование *MixColumns*() обрабатывает каждый столбец матрицы состояния

3.1.4. Преобразование *AddRoundKey()*

Преобразование *AddRoundKey*() выполняет сложение каждого байта матрицы состояния с соответствующим байтом расширенного ключа шифрования (ключа раунда). Сложение осуществляется при помощи побитовой операции *XOR*. Раундовый ключ вырабатывается из ключа шифрования посредством алгоритма расширения ключа (алгоритм описан в подразделе 3.1.5.). Сложение каждого из *Nb* слов с ключом раунда можно описать согласно выражению:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \oplus [w_{l+c}],\ l = round \cdot Nb,$$

где *round* – текущий раунд шифрования ($0 \le round < Nr$), $[w_i]$ – слово из массива подключений (раздел 3.1.5). Выполнение преобразования *AddRoundKey*() иллюстрирует рисунок 3.5.

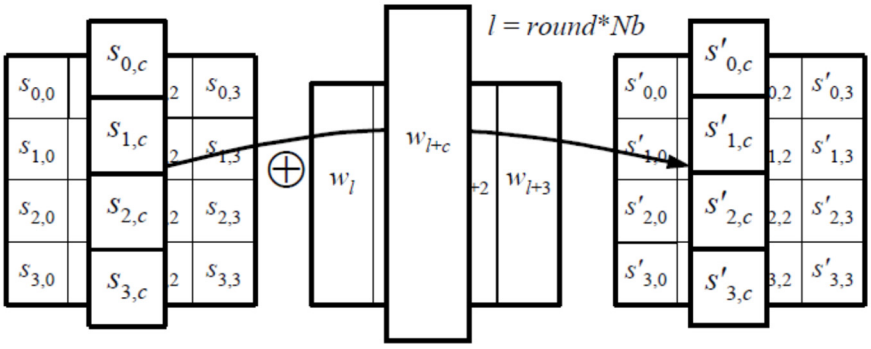

Рисунок 3.5. Преобразование *AddRoundKey*() выполняет сложение при помощи операции *XOR* между каждым столбцом матрицы состояния и словом из массива подключений

3.1.5. Процедура расширения ключа

Массив подключений получается из ключа шифрования посредством алгоритма расширения ключа. Массив подключений состоит из $Nb(Nr+1)$ 4-байтных слов, обозначаемых $[w_i]$, $0 \le i < Nb(Nr+1)$. Преобразование ключа в массив подключений выполняется посредством следующих действий:

1. Первые Nk слов массива подключений равны Nk словам ключа шифрования. Оставшиеся $Nb(Nr+1) - Nk$ слова массива подключений получаются путем выполнения следующего алгоритма.
2. Каждое последующее слово $[w_i]$ получается посредством операции XOR предыдущего слова $[w_{i-1}]$ и слова на Nk позиций ранее $[w_{i-Nk}]$: $[w_i] = [w_{i-1}] \oplus [w_{i-Nk}]$.
3. Для слов, позиция которых кратна Nk перед операцией XOR выполняется преобразование к слову $[w_{i-1}]$ – $SubWord(RotWord([w_{i-1}]))$, после чего прибавляется константа $Rcon(\frac{i}{Nk})$ при помощи операции XOR.

Функция $RotWord()$ осуществляет побайтовый сдвиг 4-байтного слова: получая слово $[a_0, a_1, a_2, a_3]$, возвращает слово $[a_1, a_2, a_3, a_0]$.

Функция $SubWord()$ принимает на вход 4-байтное слово. Выходное слово формируется благодаря замене каждого из 4-х байт с помощью *S*-блока.

Массив слов констант $Rcon(j)$ состоит из значений вида $[\{02\}^{j-1}, \{00\}, \{00\}, \{00\}]$, причем j начинается с одного.

Важно отметить, что для ключа шифрования, содержащего 256 бит, процедура расширения ключа будет немного выполнятся иначе от процедуры расширения ключа, содержащего 128 или 192 бита. Если $Nk = 8$ и $i - 4$ кратно Nk, то перед операцией XOR слово $[w_{i-1}]$ обрабатывается функцией $SubWord()$.

```
void CEncoder::ExpansionKey()
{
	U_4Bytes temp;
	for (int i = 0; i<_Nk; i++)
	{
		_w[i]._ui32 = _key[i]._ui32;
	}
	for (int i = _Nk; i<_Nb*(_Nr + 1); i++)
	{
		temp = _w[i - 1];
		if (i % _Nk == 0)
		{
			temp._ui32 = SubWord(RotWord(temp))._ui32 ^ Rcon[i / _Nk];
		}
		else if ((_Nk > 6) && (i % _Nk == 4))
		{
			temp = SubWord(temp);
		}
		_w[i]._ui32 = _w[i - _Nk]._ui32 ^ temp._ui32;
	}
}
```

Рисунок 3.6. Код процедуры расширения ключа на языке C++

3.2. Процедура дешифрования

Если вместо преобразований *SubBytes*(), *ShiftRows*(), *MixColumns*() и *AddRoundKey*() в обратной последовательности выполнить инверсные им преобразования: *InvSubBytes*(), *InvShiftRows*(), *InvMixColumns*() и *AddRoundKey*(), то можно построить функцию дешифрования. При этом порядок использования раундовых ключей является обратным по отношению к тому, который использовался при шифровании.

```
void CEncoder::Decrypt()
{
 AddRoundKey(_Nr);
 for (int i = _Nr-1; i >= 0; i--)
 {
	while (i>0)
	{
		InvShiftRows();
		InvSubBytes();
		AddRoundKey(i);
		InvMixColumns();
		i--;
	}
	InvShiftRows();
	InvSubBytes();
	AddRoundKey(i);
 }
}
```

Рисунок 3.7. Код процедуры дешифрования на языке C++

3.2.1. Преобразование *InvSubBytes()*

Преобразование *InvSubBytes*() является обратным к преобразованию *SubBytes*(). Преобразование *InvSubBytes*() изменяет каждый байт матрицы состояния при помощи инвертированного *S*-блока. Инвертированный *S*-блок показан на таблице 3.3.

		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

Таблица 3.3. Инвертированный *S*-блок: значение замен для байта {*xy*}

3.2.2. Преобразование *InvShiftRows()*

Преобразование *InvShiftRows*() является обратным к преобразованию *ShiftRows*(). Преобразование *InvShiftRows*() выполняет циклический сдвиг байт в строках матрицы состояния на различное число смещений:

- первая строка с номером $r = 0$ не сдвигается;
- вторая строка с номером $r = 1$ сдвигается на один байт вправо;
- третья строка с номером $r = 2$ сдвигается на два байта вправо;
- четвертая строка с номером $r = 3$ сдвигается на три байта вправо.

Рисунок 3.8. иллюстрирует преобразование *InvShiftRows*().

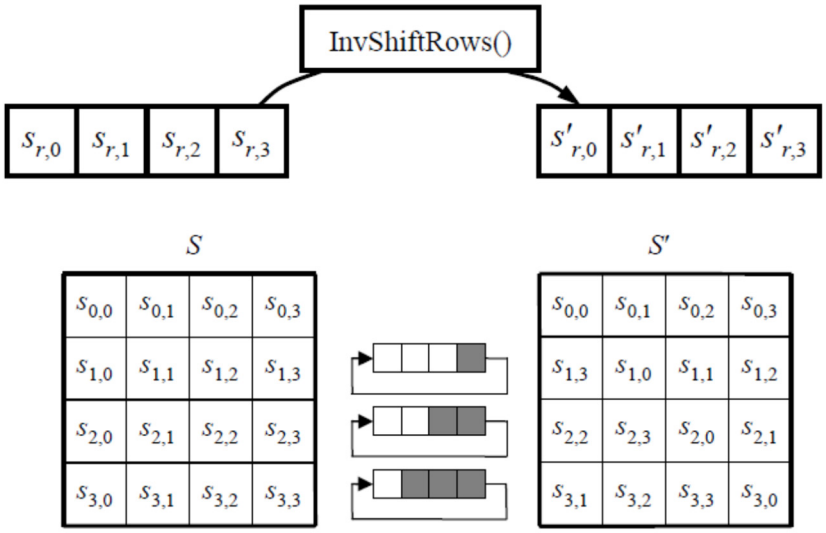

Рисунок 3.8. Преобразование *InvShiftRows*() циклически сдвигает три последние строки матрицы состояния

3.2.3. Преобразование *InvMixColumns()*

Преобразование *InvMixColumns*() является обратным к преобразованию *MixColumns*(). Преобразование *InvMixColumns*() обрабатывает каждый столбец матрицы состояния. Каждый столбец рассматривается как многочлен над полем $GF(2^8)$ и умножаются на многочлен $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$ по модулю $x^4 + 1$. Это может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}, \quad 0 \le c < Nb$$

В результате умножения байты столбца изменяются в соответствии со следующими выражениями:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$
$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

3.2.4. Преобразование *AddRoundKey()*

Преобразование *AddRoundKey*(), описанное в подразделе 3.1.4, является обратным само к себе, так как содержит только операцию *XOR*.

3.3. Табличная реализация криптосистемы

Один раунд процедуры шифрования можно представить в матричной форме: выходные данные, которые получаются на выходе одного раунда преобразований открытого текста можно представить в виде соотношений, записанных в матричной форме.

Пусть a – открытый текст, e – выходные данные (результат преобразования $AddRoundKey()$).

Пусть d – результат преобразования $MixColumns()$, c – результат преобразования $ShiftRows()$, k – ключ раунда. Запишем следующие соотношения для преобразования $MixColumns()$ и $AddRoundKey()$:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}, \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

$$j = \overline{0,3}.$$

Пусть b – результат преобразования $SubBytes()$ ($d_{i,j} = S[a_{i,j}]$, S – матрица состояния). Запишем соотношение для преобразований $ShiftRows()$ и $SubBytes()$:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ d_{2,j-2} \\ d_{3,j-3} \end{bmatrix}, \; j = \overline{0,3}.$$

Таким образом получим следующую формулу выходных данных:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}, \; j = \overline{0,3},$$

что можно переписать в следующем виде:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S[a_{0,j}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,j-1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,j-2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,j-3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}, \; j = \overline{0,3}.$$

Таким образом можно определить четыре таблицы для шифрования:

$$T_0[a]=\begin{bmatrix} S[a]\bullet 02 \\ S[a] \\ S[a] \\ S[a]\bullet 03 \end{bmatrix},\ T_1[a]=\begin{bmatrix} S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \\ S[a] \end{bmatrix},\ T_2[a]=\begin{bmatrix} S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \end{bmatrix},\ T_3[a]=\begin{bmatrix} S[a] \\ S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \end{bmatrix}.$$

Используя эти таблицы можно записать один раунд шифрования следующим образом:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-1}] \oplus T_2[a_{2,j-2}] \oplus T_3[a_{3,j-3}] \oplus k_j .$$

Таким образом, используя таблицы, один раунд шифрования можно осуществить всего за 16 обращений к *T*-таблицам: 4 обращения на каждый столбец входного массива данных и 4 обращения при выполнении операции сложения по модулю 2.

ЗАКЛЮЧЕНИЕ

В результате выполненной работы:

– выполнен анализ стандарта криптосистемы AES;

– рассмотрены элементы теории конечных полей, лежащие в основе AES-шифрования/дешифрования;

– выполнен детальный анализ процедур шифрования/дешифрования с целью определения структур данных и методов обработки при программной реализации криптосистемы AES;

– разработана и верифицирована статическая библиотека классов для выполнения процедур шифрования/дешифрования по стандарту AES.

Исходный код статической библиотеки, базирующийся на стандарте C++ допускает его использование на различных аппаратно-программных платформах.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Мао, Венбо Современная криптография = Modern Cryptography : теория и практика / Венбо Мао ; [пер. с англ. и ред. Д. А. Клюшина]. - Москва; Санкт-Петербург; Киев: Вильямс, 2005. - 764с.
2. Смарт, Н. Криптография / Н. Смарт; пер. с англ. С. А. Кулешова под ред. С. К. Ландо. - Москва: Техносфера, 2006. - 525 с.
3. Фергюсон, Нильс Практическая криптография = Practical Cryptography / Нильс Фергюсон, Брюс Шнайер ; [пер. с англ. Н. Н. Селиной ; под ред. А. В. Журавлева]. - Москва; Санкт-Петербург; Киев: Диалектика, 2005. - 422с.
4. Federal Information Processing Standards Publication 197 (FIPS 197), 2001.
5. Joan Daemen, Vincent Rijmen, The Rijndael Block Cipher AES Proposal, 1999.

ПРИЛОЖЕНИЯ

Приложение А. Файл Common.h

```
#pragma once
typedef unsigned char ui8;//переопределение типа unsigned char
typedef unsigned int ui32;//переопределение типа unsigned int

typedef struct
{
	ui8 byte0 : 8;
	ui8 byte1 : 8;
	ui8 byte2 : 8;
	ui8 byte3 : 8;
}SUI32AsStruct,*pSUI32AsStruct;

typedef union//объединение типов
{
	SUI32AsStruct _byteStr;
	ui32 _ui32;
}U_4Bytes;
```

Приложение Б. Файл объявления класса CEncoder.h

```
#pragma once
#include "Common.h"//подключение заголовочного файла с

class CEncoder
{
	int _Nb;//количество 32-битных слов (столбцов) в матрице состояния
	int _Nk;//количество 32-битных слов в ключе шифрования
	int _Nr;//количество раундов
	bool _fErrorLengthKey;//флаг ошибки
	U_4Bytes _stateMatrix[4];//объявление матрицы состояния
	ui32 _SBlock[256]= {
		0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76,
		0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0,
		0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15,
		0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75,
		0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84,
		0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf,
		0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8,
		0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2,
		0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73,
		0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb,
		0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79,
		0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
```

```
0xea, 0x65, 0x7a, 0xae, 0x08,
            0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
            0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e,
            0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf,
            0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16
      };//S-блок
      ui32 _InvSBlock[256] = {
            0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3,
0x9e, 0x81, 0xf3, 0xd7, 0xfb,
            0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
0x44, 0xc4, 0xde, 0xe9, 0xcb,
            0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
0x0b, 0x42, 0xfa, 0xc3, 0x4e,
            0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
0x49, 0x6d, 0x8b, 0xd1, 0x25,
            0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
0xcc, 0x5d, 0x65, 0xb6, 0x92,
            0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
0x57, 0xa7, 0x8d, 0x9d, 0x84,
            0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58,
0x05, 0xb8, 0xb3, 0x45, 0x06,
            0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
0x03, 0x01, 0x13, 0x8a, 0x6b,
            0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
0xce, 0xf0, 0xb4, 0xe6, 0x73,
            0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
0xe8, 0x1c, 0x75, 0xdf, 0x6e,
            0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
0x0e, 0xaa, 0x18, 0xbe, 0x1b,
            0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
0xfe, 0x78, 0xcd, 0x5a, 0xf4,
            0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
0x59, 0x27, 0x80, 0xec, 0x5f,
            0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
0x9f, 0x93, 0xc9, 0x9c, 0xef,
            0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
0x3c, 0x83, 0x53, 0x99, 0x61,
            0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0c, 0x7d
      };//обратный S-блок
      ui32 Rcon[11] = {
            0x8d000000, 0x01000000, 0x02000000, 0x04000000, 0x08000000,
0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000
      };//массив слов-констант раундов
      void ResetSMatrix();//метод, заполняющий матрицу состояния нулями
      ui8 GetByte(int nRow, int nColumn);//метод, который получает байт из
матрицы состояния
      void SetByte(int nRow, int nColumn, ui8 data);//метод, который заменяется
значение указанного байта
      U_4Bytes _key[8];//объявление ключа шифрования
      U_4Bytes _w[60];//объявления массива подключений
      void ExpansionKey();//метод расширения ключа шифрования
      void AddRoundKey(int param);//метод сложения матрицы состояния с раундовым
ключом
```

```
	void SubBytes();//метод замены байт в матрице состояния при помощи S-блока
	void InvSubBytes();//метод, обратный методу SubBytes(), заменяет байты
матрицы состояния при помощи обратного S-блока
	void ShiftRows();//метод циклического сдвига байт матрицы состояния влево
	void InvShiftRows();//метод, обратный методу ShiftRows(), циклически
сдвигает байты матрицы состояния вправо
	void MixColumns();//метод умножения каждого столбца матрицы состояния на
фиксированный многочлен a(x)
	void InvMixColumns();//метод, обратный методу MixColumns(), умножает
каждый столбец матрицы состояния на обратный к многочлену a(x)
	ui8 Multiply(ui8 a, ui8 b);//метод перемножения двух многочленов в поле
Галуа
	U_4Bytes SubWord(U_4Bytes w);//метод замены при помощи S-блока слова w
	U_4Bytes RotWord(U_4Bytes w);//метод циклического сдвига строк слова w

public:
	CEncoder();//конструктор без параметра, задает значения _Nb, _Nk, _Nr для
ключа равного 128 бит
	CEncoder(int lengthKey);//конструктор с параметром, задает значения _Nb,
_Nk, _Nr в зависимости от длины ключа
	void Receiver(ui32 *pArray);//заполнение матрицы состояния массивом данных
	void ReceiverKey(ui32 *pArray);//заполнение ключа шифрования массиовом
данных
	void Encrypt();//метод зашифровать
	void Decrypt();//метод дешифровать

	~CEncoder();
};
```

Приложение В. Файл реализации класса CEncoder.cpp

```
#include "CEncoder.h"

CEncoder::CEncoder()//конструктор без параметра, задает значения _Nb, _Nk, _Nr
для ключа равного 128 бит
{
	_Nb = 4;
	_Nk = 4;
	_Nr = 10;
	_fErrorLengthKey = false;
	ResetSMatrix();
}

CEncoder::CEncoder(int lengthKey)//конструктор с параметром, задает значения
_Nb, _Nk, _Nr в зависимости от длины ключа
{
	_Nb = 4;
	_fErrorLengthKey = false;
	switch (lengthKey)
	{
	case 128:
		_Nk = 4; _Nr = 10; break;
	case 192:
		_Nk = 6; _Nr = 12; break;
	case 256:
		_Nk = 8; _Nr = 14; break;
	default: _Nk = 4; _Nr = 10; _fErrorLengthKey = true;
```

```
	}
	ResetSMatrix();
}

void CEncoder::ResetSMatrix()//метод, заполняющий матрицу состояния нулями
{
	for (int i = 0; i<_Nb; i++)
	{
		_stateMatrix[i]._ui32 = 0;
	}
}

ui8 CEncoder::GetByte(int nRow, int nColumn)//метод, который получает байт из
матрицы состояния
{
	U_4Bytes tmp = _stateMatrix[nColumn];
	switch (nRow)
	{
	case 0:  return tmp._byteStr.byte0;
	case 1:  return tmp._byteStr.byte1;
	case 2:  return tmp._byteStr.byte2;
	case 3:  return tmp._byteStr.byte3;
	}
}

void CEncoder::SetByte(int nRow, int nColumn, ui8 data)//метод, который
заменяется значение указанного байта
{
	switch (nRow)
	{
	case 0: _stateMatrix[nColumn]._byteStr.byte0 = data; break;
	case 1: _stateMatrix[nColumn]._byteStr.byte1 = data; break;
	case 2: _stateMatrix[nColumn]._byteStr.byte2 = data; break;
	case 3: _stateMatrix[nColumn]._byteStr.byte3 = data; break;
	}
}

void CEncoder::ExpansionKey()//метод расширения ключа шифрования
{
	U_4Bytes temp;
	for (int i = 0; i<_Nk; i++)
	{
		_w[i]._ui32 = _key[i]._ui32;
	}
	for (int i = _Nk; i<_Nb*(_Nr + 1); i++)
	{
		temp = _w[i - 1];
		if (i % _Nk == 0)
		{
			temp._ui32 = SubWord(RotWord(temp))._ui32 ^ Rcon[i / _Nk];
		}
		else if ((_Nk > 6) && (i % _Nk == 4))
		{
			temp = SubWord(temp);
		}
		_w[i]._ui32 = _w[i - _Nk]._ui32 ^ temp._ui32;
	}
}
```

```
void CEncoder::AddRoundKey(int param)//метод сложения матрицы состояния с
раундовым ключом
{
	for (int i = 0; i<_Nb; i++)
	{
		_stateMatrix[i]._ui32 = _stateMatrix[i]._ui32 ^ _w[i +
param*_Nb]._ui32;
	}
}

void CEncoder::SubBytes()//метод замены байт в матрице состояния при помощи S-
блока
{
	int N = 0x00000010;
	ui8 tmp;
	for (int i = 0; i<_Nb; i++)
	{
		for (int j = 0; j<_Nb; j++)
		{
			tmp = GetByte(i, j);
			SetByte(i, j, _SBlock[tmp % 16 + N*(tmp / 16)]);
		}
	}
}

void CEncoder::InvSubBytes()//метод, обратный методу SubBytes(), заменяет байты
матрицы состояния при помощи обратного S-блока
{
	int N = 0x00000010;
	ui8 tmp;
	for (int i = 0; i<_Nb; i++)
	{
		for (int j = 0; j<_Nb; j++)
		{
			tmp = GetByte(i, j);
			SetByte(i, j, _InvSBlock[tmp % 16 + N*(tmp / 16)]);
		}
	}
}

void CEncoder::ShiftRows()//метод циклического сдвига байт матрицы состояния
влево
{
	int l;
	for (int i = _Nb - 1; i >= 0; i--)
	{
		l = _Nb - i - 1;
		ui8 *row = new ui8[_Nb];
		ui8 *rowCopy = new ui8[_Nb];
		for (int j = 0; j<_Nb; j++)
		{
			*(row + j) = GetByte(i, j);
			*(rowCopy + j) = GetByte(i, j);
		}
		for (int j = 0; j<_Nb - l; j++)
		{
			row[j] = rowCopy[j + l];
```

```
            }
            for (int j = 0; l >= 1; j++, l--)
            {
                  row[_Nb - l] = rowCopy[j];
            }
            for (int j = 0; j<_Nb; j++)
            {
                  SetByte(i, j, row[j]);
            }
      }
}

void CEncoder::InvShiftRows()//метод, обратный методу ShiftRows(), циклически
сдвигает байты матрицы состояния вправо
{
      int l;
      for (int i = _Nb - 1; i >= 0; i--)
      {
            l = _Nb - 1 - i;
            l = _Nb - l;
            ui8 *row = new ui8[_Nb];
            ui8 *rowCopy = new ui8[_Nb];
            for (int j = 0; j<_Nb; j++)
            {
                  *(row + j) = GetByte(i, j);
                  *(rowCopy + j) = GetByte(i, j);
            }
            for (int j = 0; j<_Nb - l; j++)
            {
                  row[j] = rowCopy[j + l];
            }
            for (int j = 0; l >= 1; j++, l--)
            {
                  row[_Nb - l] = rowCopy[j];
            }
            for (int j = 0; j<_Nb; j++)
            {
                  SetByte(i, j, row[j]);
            }
      }
}

void CEncoder::MixColumns()//метод умножения каждого столбца матрицы состояния
на фиксированный многочлен a(x)
{

      U_4Bytes _newstateMatrix[4];
      for (int i = 0; i<_Nb; i++)
      {
            U_4Bytes a = _stateMatrix[i];
            _newstateMatrix[i] = a;

            SetByte(3, i, Multiply(0x02, _newstateMatrix[i]._byteStr.byte3) ^
Multiply(0x03, _newstateMatrix[i]._byteStr.byte2) ^
_newstateMatrix[i]._byteStr.byte1 ^ _newstateMatrix[i]._byteStr.byte0);
            SetByte(2, i, _newstateMatrix[i]._byteStr.byte3 ^ Multiply(0x02,
_newstateMatrix[i]._byteStr.byte2) ^ Multiply(0x03,
_newstateMatrix[i]._byteStr.byte1) ^ _newstateMatrix[i]._byteStr.byte0);
```

```
            SetByte(1, i, _newstateMatrix[i]._byteStr.byte3
^_newstateMatrix[i]._byteStr.byte2 ^ Multiply(0x02,
_newstateMatrix[i]._byteStr.byte1) ^ Multiply(0x03,
_newstateMatrix[i]._byteStr.byte0));
            SetByte(0, i, Multiply(0x03, _newstateMatrix[i]._byteStr.byte3) ^
_newstateMatrix[i]._byteStr.byte2 ^ _newstateMatrix[i]._byteStr.byte1 ^
Multiply(0x02, _newstateMatrix[i]._byteStr.byte0));
      }
}

void CEncoder::InvMixColumns()//метод, обратный методу MixColumns(), умножает
каждый столбец матрицы состояния на обратный к многочлену a(x)
{
      U_4Bytes _newstateMatrix[4];
      for (int i = 0; i<_Nb; i++)
      {
            U_4Bytes a = _stateMatrix[i];
            _newstateMatrix[i] = a;

            SetByte(3, i, Multiply(0x0e, _newstateMatrix[i]._byteStr.byte3) ^
Multiply(0x0b, _newstateMatrix[i]._byteStr.byte2) ^ Multiply(0x0d,
_newstateMatrix[i]._byteStr.byte1) ^ Multiply(0x09,
_newstateMatrix[i]._byteStr.byte0));
            SetByte(2, i, Multiply(0x09, _newstateMatrix[i]._byteStr.byte3) ^
Multiply(0x0e, _newstateMatrix[i]._byteStr.byte2) ^ Multiply(0x0b,
_newstateMatrix[i]._byteStr.byte1) ^ Multiply(0x0d,
_newstateMatrix[i]._byteStr.byte0));
            SetByte(1, i, Multiply(0x0d, _newstateMatrix[i]._byteStr.byte3) ^
Multiply(0x09, _newstateMatrix[i]._byteStr.byte2) ^ Multiply(0x0e,
_newstateMatrix[i]._byteStr.byte1) ^ Multiply(0x0b,
_newstateMatrix[i]._byteStr.byte0));
            SetByte(0, i, Multiply(0x0b, _newstateMatrix[i]._byteStr.byte3) ^
Multiply(0x0d, _newstateMatrix[i]._byteStr.byte2) ^ Multiply(0x09,
_newstateMatrix[i]._byteStr.byte1) ^ Multiply(0x0e,
_newstateMatrix[i]._byteStr.byte0));
      }
}

ui8 CEncoder::Multiply(ui8 a, ui8 b)//метод перемножения двух многочленов в поле
Галуа
{
      ui8 p = 0;
      ui8 hi_bit_set;
      for (int i = 0; i < 8; i++)
      {
            if (b & 1)
                  p ^= a;
            hi_bit_set = (a & 0x80);
            a <<= 1;
            if (hi_bit_set)
                  a ^= 0x1b; /* x^8 + x^4 + x^3 + x + 1 */
            b >>= 1;
      }
      return p;
}

U_4Bytes CEncoder::SubWord(U_4Bytes w)//метод замены при помощи S-блока слова w
{
```

```
	int N = 0x00000010;
	w._byteStr.byte0 = _SBlock[w._byteStr.byte0 % 16 + N*(w._byteStr.byte0 /
16)];
	w._byteStr.byte1 = _SBlock[w._byteStr.byte1 % 16 + N*(w._byteStr.byte1 /
16)];
	w._byteStr.byte2 = _SBlock[w._byteStr.byte2 % 16 + N*(w._byteStr.byte2 /
16)];
	w._byteStr.byte3 = _SBlock[w._byteStr.byte3 % 16 + N*(w._byteStr.byte3 /
16)];
	return w;
}

U_4Bytes CEncoder::RotWord(U_4Bytes w)//метод циклического сдвига строк слова w
{
	ui8 k;
	k = w._byteStr.byte3;
	w._byteStr.byte3 = w._byteStr.byte2;
	w._byteStr.byte2 = w._byteStr.byte1;
	w._byteStr.byte1 = w._byteStr.byte0;
	w._byteStr.byte0 = k;
	return w;
}

void CEncoder::Receiver(ui32* pArray)//заполнение матрицы состояния массивом
данных
{
	for (int i = 0; i<_Nb; i++)
	{
		_stateMatrix[i]._ui32 = pArray[i];
	}
}

void CEncoder::ReceiverKey(ui32* pArray)//заполнение ключа шифрования массиовом
данных
{
	for (int i = 0; i<_Nk; i++)
	{
		_key[i]._ui32 = pArray[i];
	}
	ExpansionKey();
}

void CEncoder::Encrypt()//метод зашифровать
{
	AddRoundKey(0);
	for (int i = 1; i <= _Nr; i++)
	{
		while (i<_Nr)
		{
			SubBytes();
			ShiftRows();
			MixColumns();
			AddRoundKey(i);
			i++;
		}
		SubBytes();
		ShiftRows();
		AddRoundKey(i);
```

```
	}
}

void CEncoder::Decrypt()//метод дешифровать
{
	AddRoundKey(_Nr);
	for (int i = _Nr - 1; i >= 0; i--)
	{
		while (i>0)
		{
			InvShiftRows();
			InvSubBytes();
			AddRoundKey(i);
			InvMixColumns();
			i--;
		}
		InvShiftRows();
		InvSubBytes();
		AddRoundKey(i);
	}
}

CEncoder::~CEncoder()
{
}
```