ЛАБОРАТОРНАЯ РАБОТА № 4

СТАНДАРТ СИММЕТРИЧНОГО ШИФРОВАНИЯ AES RIJNDAEL

Цель работы: ознакомление с принципами шифрования, используемыми в алгоритме симметричного шифрования AES RIJNDAEL.

Описание лабораторной работы. ***Демонстрационная версия криптостойкого блочного алгоритма Rijndael. Состояние, ключ шифрования и число циклов.*** Rijndael — итеративный блочный шифр, имеющий переменную длину блоков и различные длины ключей. Длина ключа и длина блока могут независимо друг от друга составлять 128, 192 или 256 бит.

Разнообразные преобразования работают с промежуточным результатом, называемым состоянием (State). Состояние можно представить в виде прямоугольного массива байтов. Этот массив имеет четыре строки, а число столбцов обозначается Nb и равно длине блока, деленной на 32.

Ключ шифрования также представлен в виде прямоугольного массива с четырьмя строками. Число столбцов обозначено как Nk и равно длине ключа, деленной на 32 (рис. 1.21).

В некоторых случаях ключ шифрования изображается в виде линейного массива четырехбайтовых слов. Слова состоят из четырех байтов, которые находятся в одном столбце (при представлении в виде прямоугольного массива).

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Рис. 1.21. Пример представления состояния (Nb = 6) и ключа шифрования (Nk = 4)

Входные данные для шифра обозначаются как байты состояния в порядке $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, ... После завершения действия шифра выходные данные получаются из байтов состояния в том же порядке.

Число циклов, обозначенное Nr, зависит от значений Nb и Nk (табл. 1.7).

Таблица 1.7

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

Цикловое преобразование. Цикловое преобразование состоит из четырех различных преобразований. На языке псевдо-Си оно имеет следующий вид:

```
Round (State, RoundKey)
{
     ByteSub(State); // замена байт
     ShiftRow(State); // сдвиг строк
     MixColumn(State); // замешивание столбцов
     AddRoundKey(State, RoundKey); // добавление цикловогo ключа
}
```

Последний цикл шифра немного отличается:

```
FinalRound(State, RoundKey)
{
     ByteSub(State); // замена байт
     ShiftRow(State); // сдвиг строк
     AddRoundKey(State, RoundKey); // добавление циклового ключа
}
```

Отметим, что последний цикл отличается от простого цикла только отсутствием замешивания столбцов. Каждое из приведенных преобразований подробно рассмотрено далее.

Замена байт (*ByteSub*). Преобразование ByteSub — нелинейная замена байт, выполняемая независимо с каждым байтом состояния (рис. 1.22).

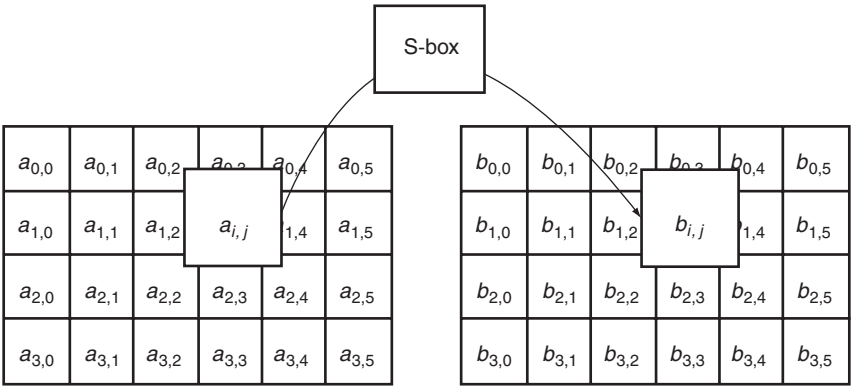

Рис. 1.22. ByteSub действует на каждый байт состояния

Замена происходит по массиву SboxE при шифровании и по массиву SboxD при расшифровании, причем SboxD[SboxE[a]] = a. На языке псевдо-Си это выглядит следующим облазом:

```
SboxE = {
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
};

SboxD = {
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
};
```

Преобразование сдвига строк (*ShiftRow*). Последние три строки состояния циклически сдвигаются на различное число байт. Строка 1 сдвигается на C1 байт, строка 2 — на C2 байт и строка 3 — на C3 байт. Значения сдвигов C1, C2 и C3 зависят от длины блока Nb. Их величины приведены в табл. 1.8.

Таблица 1.8

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Операция сдвига последних трех строк состояния на определенную величину обозначена ShiftRow (State). На рисунке 1.23 показано влияние преобразования на состояние.

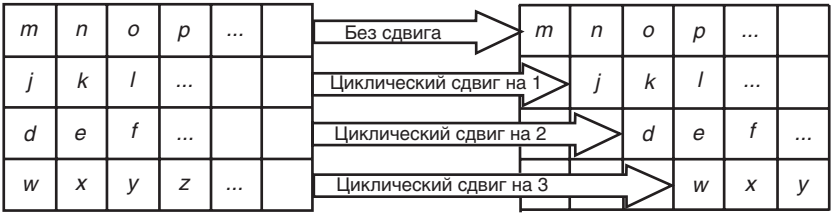

Рис. 1.23. Схема преобразования ShiftRow

При расшифровании происходит сдвиг на то же число элементов в обратном направлении.

Преобразование замешивания столбцов (MixColumn). Преобразование представляет собой умножение состояния на матрицу ME при шифровании или матрицу MD при расшифровании:

			ME	⊗	State			
			MD	⊗	State			
2	3	1	1	⊗	b1	b5	b9	b13
1	2	3	1		b2	b6	b10	b14
1	1	2	3		b3	b7	b11	b15
3	1	1	2		b4	b8	b12	b16

```
b1 = (b1 * 2) XOR (b2*3) XOR (b3*1) XOR (b4*1)
```

Умножение двух байт выполняется по следующему алгоритму:

- если один из байт равен 0, результатом будет 0;
- если один из байт равен 1, результатом будет другой байт;
- в остальных случаях происходит замена каждого байта по таблице L. Замененные байты складываются, при необходимости вычитается 255 для попадания в интервал [0, 255] и происходит замена по таблице E, что и дает результат. На языке псевдо-Си таблицы L и E имеют следующий вид:

```
L = {
    , 0x00, 0x19, 0x01, 0x32, 0x02, 0x1A, 0xC6, 0x4B,
0xC7, 0x1B, 0x68, 0x33, 0xEE, 0xDF, 0x03,
0x64, 0x04, 0xE0, 0x0E, 0x34, 0x8D, 0x81, 0xEF, 0x4C,
0x71, 0x08, 0xC8, 0xF8, 0x69, 0x1C, 0xC1,
0x7D, 0xC2, 0x1D, 0xB5, 0xF9, 0xB9, 0x27, 0x6A, 0x4D,
0xE4, 0xA6, 0x72, 0x9A, 0xC9, 0x09, 0x78,
0x65, 0x2F, 0x8A, 0x05, 0x21, 0x0F, 0xE1, 0x24, 0x12,
0xF0, 0x82, 0x45, 0x35, 0x93, 0xDA, 0x8E,
0x96, 0x8F, 0xDB, 0xBD, 0x36, 0xD0, 0xCE, 0x94, 0x13,
0x5C, 0xD2, 0xF1, 0x40, 0x46, 0x83, 0x38,
0x66, 0xDD, 0xFD, 0x30, 0xBF, 0x06, 0x8B, 0x62, 0xB3,
0x25, 0xE2, 0x98, 0x22, 0x88, 0x91, 0x10,
0x7E, 0x6E, 0x48, 0xC3, 0xA3, 0xB6, 0x1E, 0x42, 0x3A,
0x6B, 0x28, 0x54, 0xFA, 0x85, 0x3D, 0xBA,
0x2B, 0x79, 0x0A, 0x15, 0x9B, 0x9F, 0x5E, 0xCA, 0x4E,
0xD4, 0xAC, 0xE5, 0xF3, 0x73, 0xA7, 0x57,
0xAF, 0x58, 0xA8, 0x50, 0xF4, 0xEA, 0xD6, 0x74, 0x4F,
0xAE, 0xE9, 0xD5, 0xE7, 0xE6, 0xAD, 0xE8,
0x2C, 0xD7, 0x75, 0x7A, 0xEB, 0x16, 0x0B, 0xF5, 0x59,
0xCB, 0x5F, 0xB0, 0x9C, 0xA9, 0x51, 0xA0,
```

```
0x7F, 0x0C, 0xF6, 0x6F, 0x17, 0xC4, 0x49, 0xEC, 0xD8,
0x43, 0x1F, 0x2D, 0xA4, 0x76, 0x7B, 0xB7,
0xCC, 0xBB, 0x3E, 0x5A, 0xFB, 0x60, 0xB1, 0x86, 0x3B,
0x52, 0xA1, 0x6C, 0xAA, 0x55, 0x29, 0x9D,
0x97, 0xB2, 0x87, 0x90, 0x61, 0xBE, 0xDC, 0xFC, 0xBC,
0x95, 0xCF, 0xCD, 0x37, 0x3F, 0x5B, 0xD1,
0x53, 0x39, 0x84, 0x3C, 0x41, 0xA2, 0x6D, 0x47, 0x14,
0x2A, 0x9E, 0x5D, 0x56, 0xF2, 0xD3, 0xAB,
0x44, 0x11, 0x92, 0xD9, 0x23, 0x20, 0x2E, 0x89, 0xB4,
0x7C, 0xB8, 0x26, 0x77, 0x99, 0xE3, 0xA5,
0x67, 0x4A, 0xED, 0xDE, 0xC5, 0x31, 0xFE, 0x18, 0x0D,
0x63, 0x8C, 0x80, 0xC0, 0xF7, 0x70, 0x07
};

E = {
0x01, 0x03, 0x05, 0x0F, 0x11, 0x33, 0x55, 0xFF, 0x1A,
0x2E, 0x72, 0x96, 0xA1, 0xF8, 0x13, 0x35,
0x5F, 0xE1, 0x38, 0x48, 0xD8, 0x73, 0x95, 0xA4, 0xF7,
0x02, 0x06, 0x0A, 0x1E, 0x22, 0x66, 0xAA,
0xE5, 0x34, 0x5C, 0xE4, 0x37, 0x59, 0xEB, 0x26, 0x6A,
0xBE, 0xD9, 0x70, 0x90, 0xAB, 0xE6, 0x31,
0x53, 0xF5, 0x04, 0x0C, 0x14, 0x3C, 0x44, 0xCC, 0x4F,
0xD1, 0x68, 0xB8, 0xD3, 0x6E, 0xB2, 0xCD,
0x4C, 0xD4, 0x67, 0xA9, 0xE0, 0x3B, 0x4D, 0xD7, 0x62,
0xA6, 0xF1, 0x08, 0x18, 0x28, 0x78, 0x88,
0x83, 0x9E, 0xB9, 0xD0, 0x6B, 0xBD, 0xDC, 0x7F, 0x81,
0x98, 0xB3, 0xCE, 0x49, 0xDB, 0x76, 0x9A,
0xB5, 0xC4, 0x57, 0xF9, 0x10, 0x30, 0x50, 0xF0, 0x0B,
0x1D, 0x27, 0x69, 0xBB, 0xD6, 0x61, 0xA3,
0xFE, 0x19, 0x2B, 0x7D, 0x87, 0x92, 0xAD, 0xEC, 0x2F,
0x71, 0x93, 0xAE, 0xE9, 0x20, 0x60, 0xA0,
0xFB, 0x16, 0x3A, 0x4E, 0xD2, 0x6D, 0xB7, 0xC2, 0x5D,
0xE7, 0x32, 0x56, 0xFA, 0x15, 0x3F, 0x41,
0xC3, 0x5E, 0xE2, 0x3D, 0x47, 0xC9, 0x40, 0xC0, 0x5B,
0xED, 0x2C, 0x74, 0x9C, 0xBF, 0xDA, 0x75,
0x9F, 0xBA, 0xD5, 0x64, 0xAC, 0xEF, 0x2A, 0x7E, 0x82,
0x9D, 0xBC, 0xDF, 0x7A, 0x8E, 0x89, 0x80,
0x9B, 0xB6, 0xC1, 0x58, 0xE8, 0x23, 0x65, 0xAF, 0xEA,
0x25, 0x6F, 0xB1, 0xC8, 0x43, 0xC5, 0x54,
```

```
0xFC, 0x1F, 0x21, 0x63, 0xA5, 0xF4, 0x07, 0x09, 0x1B,
0x2D, 0x77, 0x99, 0xB0, 0xCB, 0x46, 0xCA,
0x45, 0xCF, 0x4A, 0xDE, 0x79, 0x8B, 0x86, 0x91, 0xA8,
0xE3, 0x3E, 0x42, 0xC6, 0x51, 0xF3, 0x0E,
0x12, 0x36, 0x5A, 0xEE, 0x29, 0x7B, 0x8D, 0x8C, 0x8F,
0x8A, 0x85, 0x94, 0xA7, 0xF2, 0x0D, 0x17,
0x39, 0x4B, 0xDD, 0x7C, 0x84, 0x97, 0xA2, 0xFD, 0x1C,
0x24, 0x6C, 0xB4, 0xC7, 0x52, 0xF6, 0x01
};
```

Добавление циклового ключа. Цикловой ключ добавляется к состоянию посредством простого EXOR (рис. 1.24). Цикловой ключ вырабатывается из ключа шифрования посредством алгоритма выработки ключей (key schedule). Длина циклового ключа равна длине блока Nb.

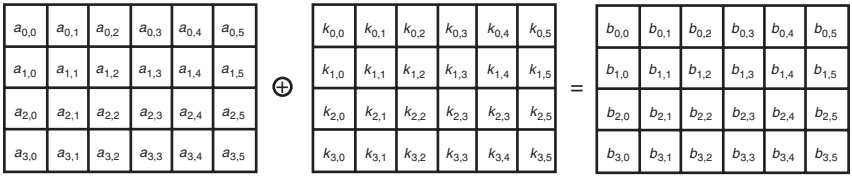

Рис. 1.24. Операция добавления циклового ключа

При шифровании части расширенного ключа выбираются от начала к концу, при расшифровании — от конца к началу.

Расширение ключа (*Key Expansion*). Расширенный ключ представляет собой линейный массив четырех байтовых слов и обозначается W[Nb*(Nr + 1)]. Первые Nk слов содержат ключ шифрования. Все остальные слова определяются рекурсивно из слов с меньшими индексами. Алгоритм выработки ключей зависит от величины Nk. Ниже приведена версия для Nk ≤ 6 и версия для Nk > 6:

- для Nk < 6 или Nk = 6

```
KeyExpansion(CipherKey,W)
{
     for (i = 0; i < Nk; i++) W[i] = CipherKey[i];
       for (j = Nk; j < Nb*(Nk+1); j+=Nk)
       {
          W[j] = W[j-Nk] ^ SubByte( Rotl( W[j-1] ) ) ^
          Rcon[j/Nk];
          for (i = 1; i < Nk && i+j < Nb*(Nr+1); i++)
```

```
          W[i+j] = W[i+j-Nk] ^ W[i+j-1];
        }
}
```

Можно заметить, что первые Nk слов заполняются ключом шифрования. Каждое последующее слово W[i] получается посредством EXOR предыдущего слова W[i – 1] и слова на Nk позиций ранее W[i – Nk]. Для слов, позиция которых кратна Nk, перед EXOR применяется преобразование к W[i – 1], а затем еще прибавляется цикловая константа. Преобразование содержит циклический сдвиг байтов в слове, обозначенный как Rotl, затем следует SubByte — применение замены байт;

- для Nk > 6

```
KeyExpansion(CipherKey,W)
{
     for (i=0; i<Nk; i++) W[i]=CipherKey[i];
        for (j=Nk; j<Nb*(Nk+1); j+=Nk)
        {
          W[j] = W[j-Nk] ^ SubByte(Rotl(W[j-1])) ^
          Rcon[j/Nk];
          for (i=1; i<4; i++) W[i+j] = W[i+j-Nk] ^
          W[i+j-1];
          W[j+4] = W[j+4-Nk] ^ SubByte(W[j+3]);
          for (i=5; i<Nk; i++) W[i+j] = W[i+j-Nk] ^
          W[i+j-1];
        }
}
```

Отличие для схемы при Nk > 6 состоит в применении SubByte для каждого четвертого байта из Nk.

Цикловая константа независит от Nk и определяется следующим образом:

```
Rcon[i] = ( RC[i], '00' , '00' , '00' ), где
RC[0]='01'
RC[i]=xtime(Rcon[i-1])
```

Шифрование. Шифр Rijndael включает следующие преобразования:

- начальное добавление циклового ключа;
- Nr – 1 циклов;

- заключительный цикл.

На языке псевдо-Си это выглядит следующим образом:

```
Rijndael (State, CipherKey)
{
     KeyExpansion(CipherKey, ExpandedKey); // Расши-
     рение ключа
     AddRoundKey(State, ExpandedKey); // Добавление
     циклового ключа
     For(i=1;i<Nr;i++)Round(State,ExpandedKey+Nb*i);
     // циклы
     FinalRound(State, ExpandedKey+Nb*Nr); // заклю-
     чительный цикл
}
```

Если предварительно выполнена процедура расширения ключа, то процедура будет иметь следующий вид:

```
Rijndael (State, CipherKey)
{
     AddRoundKey(State, ExpandedKey);
     For ( i=1 ; i<Nr ; i++) Round(State, ExpandedKey+Nb*i);
     FinalRound(State, ExpandedKey+Nb*Nr);
}
```

Описание демонстрационной программы. Программа выполнена на языке C# и состоит из двух элементов — файла Rijndael.dll, содержащего реализацию алгоритма шифрования, и демонстрационного приложения **RijndaelDemo.exe**. Для работы приложения необходима ОС Windows с установленным .NET Framework v1.1.

В основном окне демонстрационной программы задаются длина ключа, длина блока, а также расширенный ключ шифрования, вычисляемый в соответствии с заданным ключом шифрования (рис. 1.25, 1.26).

Можно подробно рассмотреть действие всех цикловых преобразований (ByteSub, ShiftRow, MixColumn, AddRoundKey) как при шифровании, так и при расшифровании (рис. 1.27, 1.28).

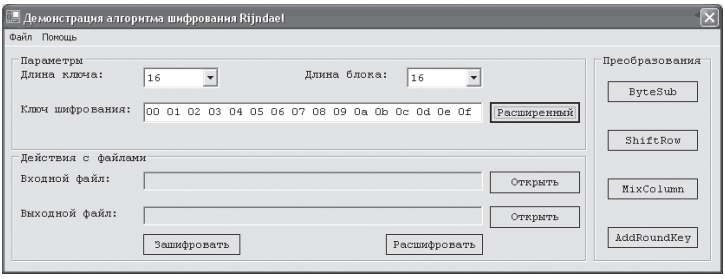

Рис. 1.25. Главное окно демонстрационной программы

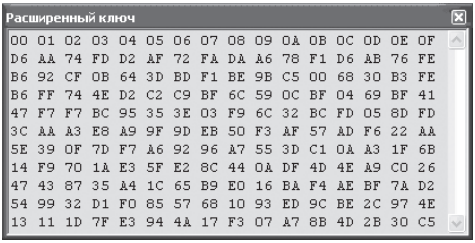

Рис. 1.26. Окно расширенного ключа

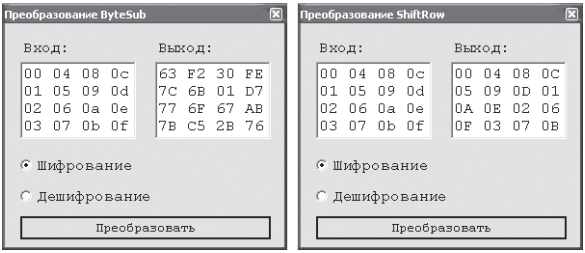

Рис. 1.27. Окна преобразований ByteSub и ShiftRow

При шифровании предлагается выбрать исходный файл и файл, куда будет помещен результат шифрования, при расшифровании — соответственно зашифрованный файл и файл, предназначенный для помещения результата расшифрования. В процессе используются указанные в главном окне программы ключ шифрования и длины ключа и блока.

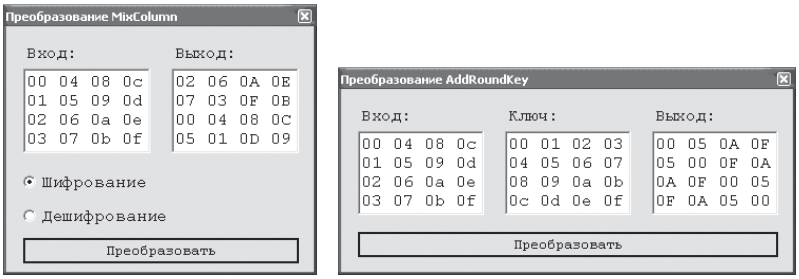

Рис. 1.28. Окна преобразований MixColumn и AddRoundKey

Задание

1. Ознакомиться со сведениями о программе RijndaelDemo. Запустить модуль **RijndaelDemo.exe**.

2. Изучить на примере обычных текстовых файлов способы шифрования и расшифрования с помощью алгоритма Rijndael. Подробно рассмотреть действие всех цикловых преобразований (ByteSub, ShiftRow, MixColumn, AddRoundKey) как при шифровании, так и расшифровании. Исходный текст для шифрования может быть подготовлен заранее и сохранен в файле *.txt.

3. Сохранить в отчете экранные формы, демонстрирующие процесс шифрования и расшифрования информации, проанализировать полученные результаты.

4. Включить в отчет о лабораторной работе ответы на контрольные задания, выбранные в соответствии с номером варианта, указанным преподавателем (табл. 1.9).

Таблица 1.9

Номер варианта	Контрольные задания
1, 5, 7, 26	Сравнить основные характеристики алгоритмов Rijndael и ГОСТ 28147—89
2, 4, 6	Сравнить основные характеристики алгоритмов Rijndael и DES
11, 13	Описать структуру сети Фейстеля
12, 14, 16	Привести обобщенные схемы шифрования данных с помощью алгоритма Rijndael и ГОСТ 28147—89. Дать их сравнительный анализ
3, 9, 18, 29	Сравнить один раунд шифрования данных с помощью алгоритма Rijndael и ГОСТ 28147—89