А.М. Голиков

ИЗУЧЕНИЕ СТАНДАРТА КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ AES (ADVANCED ENCRYPTION STANDART)

Методические указания по лабораторной работе для студентов специальностей 090106 «Информационная безопасность телекоммуникационных систем» и 210403 «Защищенные системы связи»

Томск - 2007

ИЗУЧЕНИЕ СТАНДАРТА КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ AES (ADVANCED ENCRYPTION STANDART)

1 Цель работы

Изучить криптографический стандарт шифрования AES и его особенности, познакомиться с различными режимами блочного шифрования и единственной атакой «Квадрат».

2 Краткие теоретические сведения

2.1 Историческая справка

В 1997 г. Национальный институт стандартов и технологий США (NIST) объявил о начале программы по принятию нового стандарта криптографической защиты - стандарта XXI в. для закрытия важной информации правительственного уровня на замену существующему с 1974 г. алгоритму DES, самому распространенному криптоалгоритму в мире.

Требования к кандидатам были следующие:

- криптоалгоритм должен быть открыто опубликован;
- криптоалгоритм должен быть симметричным блочным шифром, допускающим размеры ключей в 128, 192 и 256 бит;
- криптоалгоритм должен быть предназначен как для аппаратной, так и для программной реализации;
- криптоалгоритм должен быть доступен для открытого использования в любых продуктах, а значит, не может быть запатентован, в противном случае патентные права должны быть аннулированы;
- криптоалгоритм подвергается изучению по следующим па раметрам: стойкости, стоимости, гибкости, реализуемости в smart-картах.

В финал конкурса вышли следующие алгоритмы: MARS, TWOFISH и RC6 (США), RUNDAEL (Бельгия), SERPENT (Великобритания, Израиль, Норвегия). По своей структуре TWOFISH является классическим шифром Фейстеля; MARS и RC6 можно отнести к модифицированным шифрам Фейстеля, в них используется новая малоизученная операция циклического "прокручивания" битов слова на число позиций, изменяющихся в зависимости от шифруемых данных и секретного ключа; RUNDAEL и

SERPENT являются классическими SP-сетями. MARS и TWOFISH имеют самую сложную конструкцию, RUNDAEL и RC6 - самую простую.

В октябре 2000 г. конкурс завершился. Победителем был признан бельгийский шифр RIJNDAEL, как имеющий наилучшее сочетание стойкости, производительности, эффективности реализации и гибкости. Его низкие требования к объему памяти делают его идеально подходящим для встроенных систем. Авторами шифра являются Йон Дэмен (Joan Daemen) и Винсент Рюмен (Vincent Rijmen), начальные буквы фамилий которых и образуют название алгоритма - RIJNDAEL.

После этого NIST начал подготовку предварительной версии Федерального Стандарта Обработки Информации (Federal Information Processing Standart - FIPS) и в феврале 2001 г. опубликовал его на сайте http://csrc.nist.gov/encryption/aes/. В течение 90-дневного периода открытого обсуждения предварительная версия FIPS пересматривалась с учетом комментариев, после чего начался процесс исправлений и утверждения. Наконец 26 ноября 2001 г. была опубликована окончательная версия стандарта FIPS-197, описывающего новый американский стандарт шифрования AES. Согласно этому документу стандарт вступил в силу с 26 мая 2002 г.

2.2 Блочный криптоалгоритм RIJNDAEL и стандарт AES

Высочайшую надежность AES NIST подтверждает астрономическими числами. 128битный ключ обеспечивает 340 андециллионов (340*1036) возможных комбинаций, а 256битный ключ увеличивает это число до 11*1076. Для сравнения, старый алгоритм DES, дает общее число комбинаций в 72*1015 На их перебор у специально построенной машины "DES Cracker" уходит несколько часов. Но даже если бы она делала это всего за одну секунду, то на перебор 128битного ключа машина потратила бы 149 триллионов лет. Межу тем, возраст всей Вселенной ученые оценивают в менее, чем 20 миллиардов лет.

2.2.1 Математические предпосылки

Алгоритм оперирует байтами, которые рассматриваются как элементы конечного поля $GF(2^8)$.

Элементами поля $GF(2^8)$ являются многочлены степени не более 7, которые могут быть заданы строкой своих коэффициентов. Если представить байт в виде

$b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0,$

то элемент поля описывается многочленом с коэффициентами из {0, 1}:

$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0$

Например, байту {11001011} (или {cb} в шестнадцатеричной форме) соответствует многочлен $x^7 + x^6 + x^3 + x + 1$.

Для элементов конечного поля определены аддитивные и мультипликативные операции.

Сложение

Сложение суть операция поразрядного XOR и поэтому обозначается как ⊕. Пример выполнения операции сложения:

$(x^6 + x^4 + x^2 + x + 1) \oplus (x^7 + x + 1) = x^1 + x^6 + x^4 + x^2$ (в виде многочленов)

{01010111} 0 {10000011} = {11010100} (двоичное представление)

{57}⊕{83} = {d4} (шестнадцатеричное представление)

В конечном поле для любого ненулевого элемента **a** существует обратный элемент **-a**, при этом a + (-a) = 0, где нулевой элемент -это {00}. В $GF(2^8)$ справедливо a + a = 0, т. е. каждый ненулевой элемент является своей собственной аддитивной инверсией.

Умножение

Умножение, обозначаемое далее как •, более сложная операция. Умножение в $GF(2^8)$ - это операция умножения многочленов со взятием результата по модулю неприводимого многочлена m(x) восьмой степени и с использованием операции XOR при приведении подобных членов. В RIJNDAEL выбран $m(x) = x^8 + x^4 + x^3 + x + 1$, или в шестнадцатеричной форме l{1b} (такая запись обозначает, что присутствует «лишний» девятый бит). Пример операции умножения:

{57}·{83} = {cl},

или

(x6 + x4 + x2 + x + 1) (x7 + x + 1) =

= x13 + x11 + x9 + x8 + x7 + x7 + x5 + x3 + x2 + x + x6 + x4 + x2 + x + 1 =

$= x13 + x11 + x9 + x8 + x6 + x5 + x4 + x3 + 1$

Следовательно,

$x13 + x11 + x9 + x8 + x6 + x5 + x4 + x8 + 1 \bmod (x8 + x4 + x3 + x + 1) = x7 + x6 + 1$

Ясно, что результат является двоичным полиномом не выше 8 степени. В отличие от сложения, простой операции умножения на уровне байтов не существует.

Умножение, определенное выше, является ассоциативным, и существует единичный элемент ('01'). Для любого двоичного полинома b(x) не выше 8-й степени можно использовать расширенный алгоритм Евклида для вычисления полиномов a(x) и c(x) таких, что

$b(x)\, a(x) + m(x)\, c(x) = 1$

Следовательно,

$a(x) * b(x) \bmod m(x) = 1$

или

$b - 1(x) = a(x) \bmod m(x)$

Более того, можно показать, что

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

Из всего этого следует, что множество из 256 возможных значений байта образует конечное поле GF (2^8) с XOR в качестве сложения и умножением, определенным выше.

Умножение на x

Если умножить b(x) на полином x, мы будем иметь:

$b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x$

x * b(x) получается понижением предыдущего результата по модулю m(x). Если $b_7 = 0$, то данное понижение является тождественной операцией. Если $b_7 = 1$, m(x) следует вычесть (т.е. XORed). Из этого следует, что умножение на x может быть реализовано на уровне байта как левый сдвиг и последующий побитовый XOR с '1B'. Данная операция обозначается как b = xtime (a).

Полиномы с коэффициентами из GF (2^8)

Полиномы могут быть определены с коэффициентами из GF (2^8). В этом случае четырехбайтный вектор соответствует полиному степени 4.

Полиномы могут быть сложены простым сложением соответствующих коэффициентов. Как сложение в GF (2^8) является побитовым XOR, так и сложение двух векторов является простым побитовым XOR.

Умножение представляет собой более сложное действие. Предположим, что мы имеем два полинома в GF (2^8).

$$a(x) = a3\ x3 + a2\ x2 + a1\ x + a0$$
$$b(x) = b3\ x3 + b2\ x2 + b1\ x + b0$$

c(x) = a(x) b(x)

определяется следующим образом

$$c(x) = c6\ x6 + c5\ x5 + c4\ x4 + c3\ x3 + c2\ x2 + c1\ x + c0$$
$$c0 = a0 * b0$$
$$c1 = a1 * b0 \oplus a0 * b1$$
$$c2 = a2 * b0 \oplus a1 * b1 \oplus a0 * b2$$
$$c3 = a3 * b0 \oplus a2 * b1 \oplus a1 * b2 \oplus a0 * b3$$
$$c4 = a3 * b1 \oplus a2 * b12 \oplus a1 * b3$$
$$c5 = a3 * b2 \oplus a2 * b3$$
$$c6 = a3 * b3$$

Ясно, что в таком виде c(x) не может быть представлен четырехбайтным вектором. Понижая c(x) по модулю полинома 4-й степени, результат может быть полиномом степени ниже 4. В Rijndael это сделано с помощью полинома

M(x) = x4 + 1

так как

xj mod (x4 + 1) = xj mod 4

Модуль, получаемый из a(x) и b(x), обозначаемый d(x) = a(x) ⊗b(x), получается следующим образом:

$d0 = a0 * b0 \oplus a3 * b1 \oplus a2 * b2 \oplus a1 * b3$

$d1 = a1 * b0 \oplus a0 * b1 \oplus a3 * b2 \oplus a2 * b3$

$d2 = a2 * b0 \oplus a1 * b1 \oplus a0 * b2 \oplus a3 * b3$

$d3 = a3 * b0 \oplus a2 * b1 \oplus a1 * b2 \oplus a0 * b3$

Операция, состоящая из умножения фиксированного полинома a(x), может быть записана как умножение матрицы, где матрица является циклической. Мы имеем

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

2.2.2 Описание криптоалгоритма

Формат блоков данных и число раундов

RIJNDAEL - это итерационный блочный шифр, имеющий архитектуру "Квадрат". Шифр имеет переменную длину блоков и различные длины ключей. Длина ключа и длина блока могут быть равны независимо друг от друга 128, 192 или 256 битам. В стандарте AES определена длина блока данных, равная 128 битам.

Введем следующие обозначения:

- Nb - число 32-битных слов содержащихся во входном блоке, Nb=4;
- Nk - число 32-битных слов содержащихся в ключе шифрования, Nk=4,6,8;
- Nr - число раундов шифрования, как функция от Nb и Nk, Nr=10,12,14.

Входные (input), промежуточные (state) и выходные (output) результаты преобразований, выполняемых в рамках криптоалгоритма, называются состояниями (State). Состояние можно представить в виде прямоугольного массива байтов (рис. 1). При размере блока, равном 128 битам, этот 16-байтовый массив (рис. 2, а) имеет 4 строки и 4 столбца (каждая строка и каждый столбец в этом случае могут рассматриваться как 32-разрядные слова). Входные данные для шифра обозначаются как байты состояния в порядке S00, S10, S20, S30, S01, S11, S21, S31

После завершения действия шифра выходные данные получаются из байтов состояния в том же порядке. В общем случае число столбцов Nb равно длине блока, деленной на 32.

State

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

Рис. 1. Пример представления 128-разрядного блока данных в виде массива State, где a,- - байт блока данных, а каждый столбец - одно 32-разрядное слово

s_{00}	s_{01}	s_{02}	s_{03}
s_{10}	s_{11}	s_{12}	s_{13}
s_{20}	s_{21}	s_{22}	s_{23}
s_{30}	s_{31}	s_{32}	s_{33}

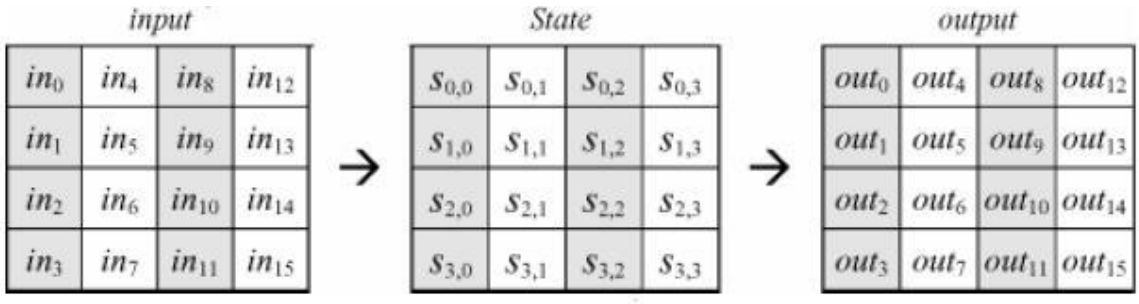
а

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

б

Рис. 2. Форматы данных: a - пример представления блока (Nb, = 4); 6 - ключа шифрования (Nk = 4), где s и k - соответственно байты массива State и ключа, находящиеся на пересечении i-й строки и j-то столбца

Рисунок 3 демонстрирует такое представление, носящее название архитектуры «Квадрат».



in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

→

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

→

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Рис. 3. Пример представления блока в виде матрицы 4Nb

Ключ шифрования также представлен в виде прямоугольного массива с четырьмя строками (рис. 2, б). Число столбцов этого массива равно длине ключа, деленной на 32. В стандарте определены ключи всех трех размеров - 128 бит, 192 бита и 256 бит, то есть соответственно 4, 6 и 8 32-разрядных слова (или столбца – в табличной форме представления). В некоторых случаях ключ шифрования рассматривается как линейный массив 4-байтовых слов. Слова состоят из 4 байтов, которые находятся в одном столбце (при представлении в виде прямоугольного массива).

Число раундов N_r в алгоритме RIJNDAEL зависит от значений Nb и Nk, как показано в табл. 1. В стандарте AES определено соответствие между размером ключа, размером блока данных и числом раундов шифрования, как показано в табл. 2.

Таблица 1. Число раундов N_r как функция от длины ключа Nk и длины блока Nb

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

Таблица 2. Соответствие между длиной ключа, размером блока данных и числом раундов в стандарте AES

Стандарт	Длина ключа (N_k слов)	Размер блока данных (N_b слов)	Число раундов (N_r)
AES-128	10	12	14
AES-192	12	12	14
AES-256	14	14	14

Раундовое преобразование

Раунд состоит из четырех различных преобразований:

- замены байтов SubBytes() - побайтовой подстановки в S-блоках с фиксированной таблицей замен размерностью 8 x 256;
- сдвига строк ShiftRows() - побайтового сдвига строк массива State на различное количество байт;
- перемешивания столбцов MixColumns() — умножения столбцов состояния, рассматриваемых как многочлены над $GF(2^8)$, на многочлен третьей степени g(x) по модулю $x^4 + 1$;
- сложения с раундовым ключом AddRoundKey() - поразрядного XOR с текущим

фрагментом развернутого ключа.

Замена байтов (SubBytes). Преобразование SubBytes() представляет собой нелинейную замену байтов, выполняемую независимо с каждым байтом состояния. Таблицы замены S-блока являются инвертируемыми и построены из композиции следующих двух преобразований входного байта:

1. получение обратного элемента относительно умножения в поле $GF(2^8)$, нулевой элемент {00} переходит сам в себя;
2. применение преобразования над $GF(2^8)$, определенного следующим образом:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Другими словами суть преобразования может быть описана уравнениями:

$$b'_I = b_i \oplus b_{(I+4)\bmod 8} \oplus b_{(I+5)\bmod 8} \oplus b_{(I+6)\bmod 8} \oplus b_{(I+7)\bmod 8} \oplus c_i$$

где $c_0=c_1=c_5=c_6= 0$, $c_2=c_3=c_4=c_7= 1$, b'_i и b_i - соответственно преобразованное и исходное значение i-го бита, i = 0,1,2….7.

Применение описанного S-блока ко всем байтам состояния обозначается как SubBytes(State). Рис. 4. иллюстрирует применение преобразования SubBytes() к состоянию. Логика работы S-блока при преобразовании байта {xy} отражена в табл. 3. Например, результат {ed} преобразования байта {53} находится на пересечении 5-й строки и 3-го столбца.

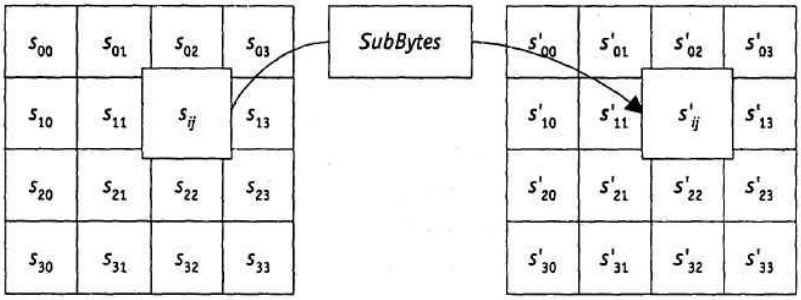


Рис. 4. SubBytes() действует на каждый байт состояния

Таблица 3. Таблица замен S-блока

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Преобразование сдвига строк (ShiftRows). Последние 3 строки состояния циклически сдвигаются влево на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 - на C2 байт, и строка 3 - на C3 байт. Значения сдвигов C1, C2 и C3 в RIJNDAEL зависят от длины блока N_b. Их величины приведены в табл..4.

Таблица 4. Величина сдвига для разной длины блоков

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

В стандарте AES, где определен единственный размер блока, равный 128 битам, C1 = 1, C2 = 2 и C3 = 3.

Операция сдвига последних трех строк состояния обозначена как ShiftRows(State). Рис. 5 показывает влияние преобразования на состояние.

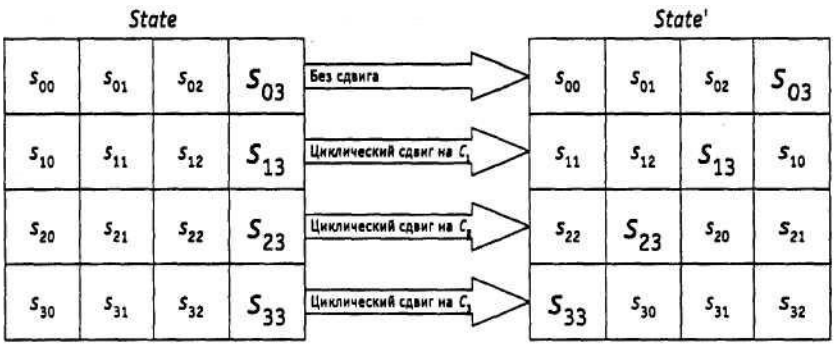


Рис. 5. ShiftRows() действует на строки состояния

Преобразование перемешивания столбцов (MixColumns).В этом преобразовании столбцы состояния рассматриваются как многочлены над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на многочлен g(x), выглядящий следующим образом:

$g(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Это может быть представлено в матричном виде следующим образом:

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

где c - номер столбца массива State

В результате такого умножения байты столбца S0c, S1c, S2c, S3c заменяются соответственно на байты

S'0c = ({02} • S0c) ⊕ ({03} • S1c) ⊕ S2c ⊕ S3c,

S'1c = S0c ⊕ ({02} • S1c) ⊕ ({03} • S2c) ⊕ S3c,

S'2c = S0c ⊕ S1c ⊕ ({02} • S2c) ⊕ ({03} • S3c),

S'3c = ({03} • S0c) ⊕ S1c © S2c ⊕ ({02} • S3c).

Применение этой операции ко всем четырем столбцам состояния обозначено как MixColumns(State). Рис. 6 демонстрирует применение преобразования MixColumnsQ к столбцу состояния.

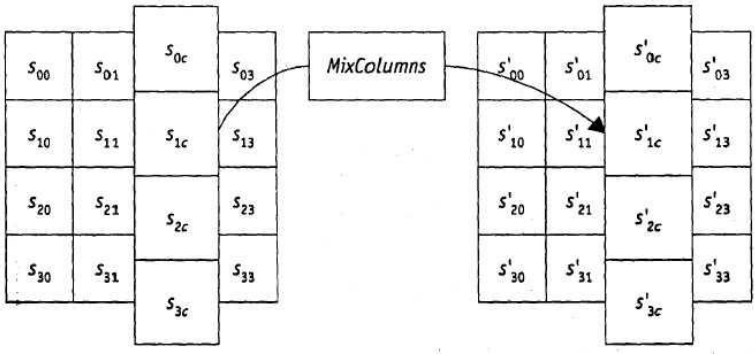


Рис. 6. MixColumnsQ действует на столбцы состояния

Добавление раундового ключа (AddRoundKey). В данной операции раундовый ключ добавляется к состоянию посредством простого поразрядного XOR. Раундовый ключ вырабатывается из ключа шифрования посредством алгоритма выработки ключей (key schedule). Длина раундового ключа (в 32-разрядных словах) равна длине блока Nb. Преобразование, содержащее добавление посредством XOR раундового ключа к состоянию (рис. 7), обозначено как AddRoundKey(State, RoundKey).

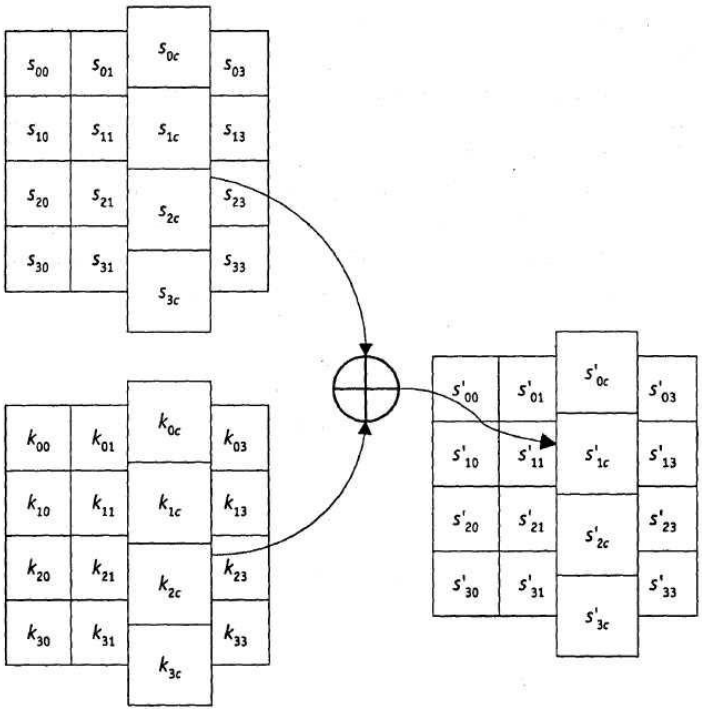


Рис. 7. При добавлении ключа раундовый ключ складывается посредством операции X0R с состоянием

Алгоритм выработки ключей (Key Schedule)

Раундовые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента: расширение ключа (Key Expansion) и выбор раундового ключа (Round Key Selection). Основополагающие принципы алгоритма выглядят следующим образом:

- общее число битов раундовых ключей равно длине блока, умноженной на число раундов, плюс 1 (например, для длины блока 128 бит и 10 раундов требуется 1408 бит раундовых ключей);
- ключ шифрования расширяется в расширенный ключ (Expanded Key);
- раундовые ключи берутся из расширенного ключа следующим образом: первый раундовый ключ содержит первые Nb слов, второй - следующие Nb слов и т. д.

Расширение ключа (Key Expansion). Расширенный ключ в RIJNDAEL представляет собой линейный массив w[i] из $Nb(N_r+ 1)$ 4-байтовых слов, $i = 0,1…Nb(N_r + 1)$. В AES массив w[i] состоит из $4(N_r +1)$ 4-байтовых слов, $i = 0,1…4(N_r + 1)$.

Примечание. Необходимо учитывать, что с целью полноты описания здесь приводится алгоритм для всех возможных длин ключей, на практике же полная его

реализация нужна не всегда.

Первые Nk слов содержат ключ шифрования. Все остальные слова определяются рекурсивно из слов с меньшими индексами. Алгоритм выработки ключей зависит от величины Nk.

Как можно заметить (рис. 8, а), первые Nk слов заполняются ключом шифрования. Каждое последующее слово w[i] получается посредством XOR предыдущего слова w[i-l] и слова на Nk позиций ранее w[i - Nk]'.

$w[i] = w[i - 1] \oplus w[i - N_k]$.

Для слов, позиция которых кратна N_k, перед XOR применяется преобразование к w[i-1], а затем еще прибавляется раундовая константа Rcon. Преобразование реализуется с помощью двух дополнительных функций: RotWord(), осуществляющей побайтовый сдвиг 32-разрядного слова по формуле {a0 a1 a2 a3} → {a1 a2 a3 a0}, и SubWord() осуществляющей побайтовую замену с использованием 5-блока функции SubBytes(). Значение Rcon[j] равно 2^{j-1}. Значение w[i] в этом случае равно

$w[i] - SubWord(RotWord(w[i - 1])) \oplus Rcon[i/N_k] \oplus w[i - N_k]$.

Рис. 8. Процедуры

а - расширения ключа (светло-серым цветом выделены слова расширенного ключа, которые формируются без использования функций SubWord() и RotWordQ; темно-серым цветом выделены слова расширенного ключа, при вычислении которых используются преобразования SubWordQ и RotWordQ);

б - выбора раундового ключа для Nk - 4

Выбор раундового ключа (Round Key Selection). Раундовый ключ i получается из слов массива раундового ключа от W[Nb i] и до $W[N_b (i + 1)]$, как показано на рис. 8.

Примечание. Алгоритм выработки ключей можно осуществлять и без использования массива w[i]. Для реализаций, в которых существенно требование к занимаемой памяти, раундовые ключи могут вычисляться "на лету" посредством использования буфера из Nk слов. Расширенный ключ должен всегда получаться из ключа шифрования и никогда не указывается напрямую. Нет никаких ограничений на выбор ключа шифрования.

Функция зашифрования

Шифр RIJNDAEL состоит (рис. 9):

- из начального добавления раундового ключа;
- Nr - 1 раундов;
- заключительного раунда, в котором отсутствует операция MixColumns().

На вход алгоритма подаются блоки данных State, в ходе преобразований содержимое блока изменяется и на выходе образуется шифротекст, организованный опять же в виде блоков State, как показано на рис. 26, где Nb - 4, in_m и out_m - m-е байты соответственно входного и выходного блоков, m =0,1…15, s_{ij} - байт, находящийся на пересечении i-й строки и j-го столбца массива State, i = j = 0,1..3.

Перед началом первого раунда происходит суммирование по модулю 2 с начальным ключом шифрования, затем - преобразование массива байтов State в течение 10, 12 или 14 раундов в зависимости от длины ключа. Последний раунд несколько отличается от предыдущих тем, что не задействует функцию перемешивания байт в столбцах MixColumns().

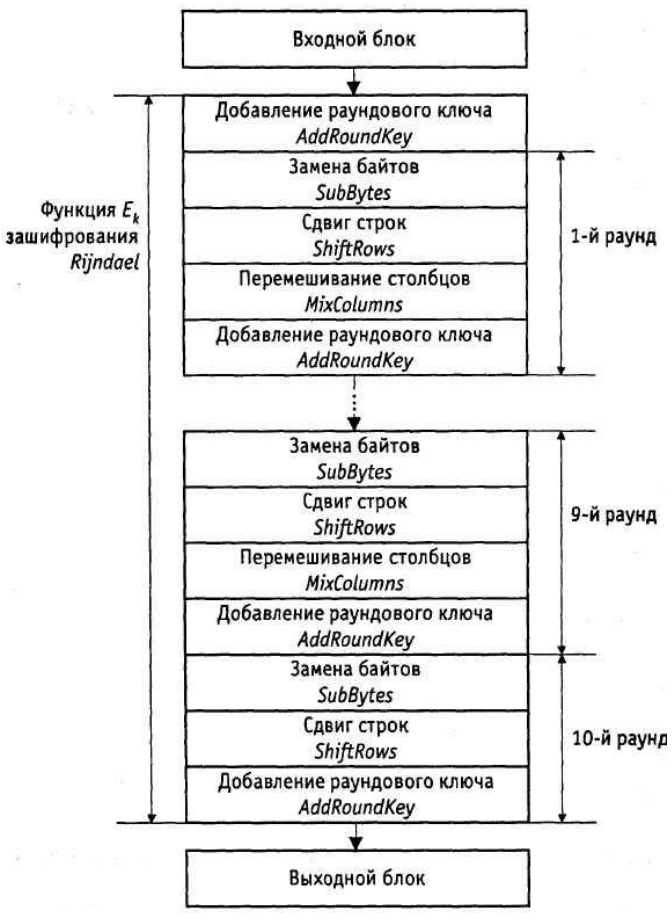


Рис. 9. Схема функции $E_к$ зашифрования криптоалгоритма RIJNDAEL при Nk = Nb = 4

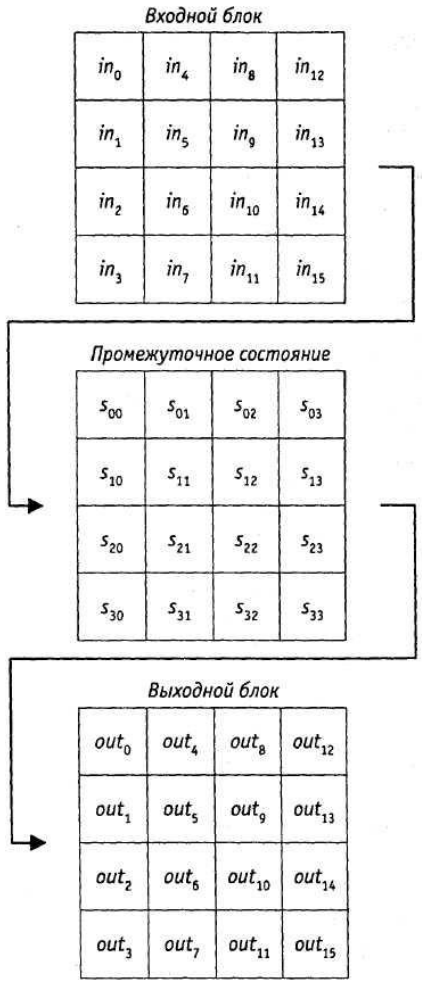


Рис. 10. Ход преобразования данных, организованных в виде блоков State

Рис. 11 демонстрирует и рассеивающие и перемешивающие свойства шифра. Видно, что два раунда обеспечивают полное рассеивание и перемешивание.

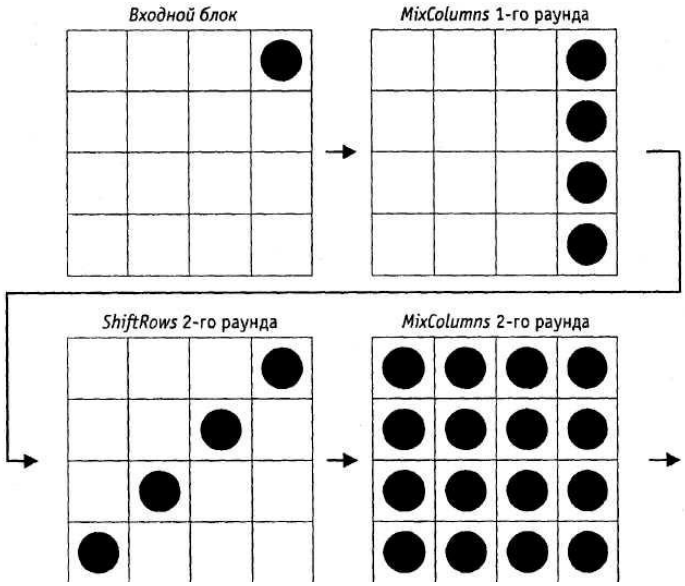


Рис. 11. Принцип действия криптоалгоритма RIJNDAEL; ◘ - измененный байт

Функция обратного расшифрования

Если вместо SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() в обратной последовательности выполнить инверсные им преобразования, можно построить функцию обратного расшифрования. При этом порядок использования раундовых ключей является обратным по отношению к тому, который используется при зашифровании.

Далее приводится описание функций, обратных используемым при прямом зашифровании.

Функция AddRoundKey() обратна сама себе, учитывая свойства используемой в ней операции XOR.

Преобразование InvSubBytes. Логика работы инверсного S-блока при преобразовании байта {xy} отражена в табл. 5.

Таблица 5. Таблица замен инверсного S-блока

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Преобразование InvShiftRows. Последние 3 строки состояния циклически сдвигаются вправо на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 - на C2 байт, и строка 3 - на C3 байт. Значения сдвигов C1, C2, C3 зависят от длины блока Nb,. Их величины приведены в табл. 8.

Преобразование InvMixColumns. В этом преобразовании столбцы состояния рассматриваются как многочлены над GF(2^8) и умножаются по модулю $x^4 + 1$ на многочлен $g^{-1}(x)$, выглядящий следующим образом:

$g^{-1}(x) = \{0b\}ч^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

Это может быть представлено в матричном виде следующим образом:

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

В результате на выходе получаются следующие байты

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

Функция прямого расшифрования

Алгоритм обратного расшифрования, описанный выше, имеет порядок приложения операций-функций, обратный порядку операций в алгоритме прямого шифрования, но использует те же параметры (развернутый ключ). Однако некоторые свойства алгоритма шифрования RIJNDAEL позволяют применить для расшифрования тот же порядок приложения функций (обратных используемым для зашифрования) за счет изменения некоторых параметров, а именно - развернутого ключа.

Два следующих свойства позволяют применить алгоритм прямого расшифрования.

Порядок приложения функций SubBytes() и ShiftRows() не играет роли. То же самое верно и для операций InvSubBytes() и InvShiftRows(). Это происходит потому, что функции SubBytes() и InvSubBytes() работают с байтами, а операции ShiftRows() и InvShifiRows() сдвигают целые байты, не затрагивая их значений.

Операция MixColumns() является линейной относительно входных данных, что означает

InvMixColumns(State XOR RoundKey) =

= lnvMixColumns(State) XOR InvMixColumns(RoundKey)

Эти свойства функций алгоритма шифрования позволяют изменить порядок применения функций InvSubBytes() и InvShiftRows(). Функции AddRounKey() и InvMixColwnns() также могут быть применены в обратном порядке, но при условии, что столбцы (32-битные слова) развернутого ключа расшифрования предварительно пропущены через функцию invMixColumns().

Таким образом, можно реализовать более эффективный способ расшифрования с тем же порядком приложения функций как и в алгоритме зашифрования.

При формировании развернутого ключа шифрования в процедуру развертывания ключа необходимо добавить следующий код

Примечание. В последнем операторе (в функции InvMixColumn()) происходит преобразование типа, так как развернутый ключ хранится в виде линейного массива 32-разрядных слов, в то время как входной параметр функции - двумерный массив байтов.

В табл. 6 приведена процедура зашифрования, а также два эквивалентных варианта процедуры расшифрования при использовании двухраундового варианта Rijndael. Первый вариант функции расшифрования суть обычная инверсия функции зашифрования. Второй вариант функции зашифрования получен из первого после изменения порядка следования операций в трех парах преобразований:

invShiftRows — InvSubBytes (дважды)

и AddRoundKey — InvMixColumns.

Таблица 6. Последовательность преобразований в двухраундовом варианте RIJNDAEL

Функция зашифрования двухраундового варианта *RIJNDAEL*	**Функция обратного расшифрования двухраундового варианта *RIJNDAEL***	**Эквивалентная функция прямого расшифрования двухраундового варианта *RIJNDAEL***
AddRoundKey	AddRoundKey	AddRoundKey
SubBytes	InvShiftRows	InvSubBytes
ShiftRows	InvSubBytes	InvShiftRows
MixColumns	AddRoundKey	InvMixColumns
AddRoundKey	InvMixColumns	AddRoundKey
SubBytes	InvShiftRows	InvSubBytes
ShiftRows	InvSubBytes	InvShiftRows
AddRoundKey	AddRoundKey	AddRoundKey

Очевидно, что результат преобразования при переходе от исходной к обратной последовательности выполнения операций в указанных парах не изменится.

Видно, что процедура зашифрования и второй вариант процедуры расшифрования совпадают с точностью до порядка использования раундовых ключей (при выполнении операций AddRoundKey), таблиц замен (при выполнении операций SubBytes и InvSubBytes) и матриц преобразования (при выполнении операций MixColumns и invMixColumns). Данный результат легко обобщить и на любое другое число раундов.

2.3 Атака “Квадрат”

Атака "Квадрат" была специально разработана для одноименного шифра SQUARE (авторы J. Daemen, L. Knudsen, V. Rijmen). Атака использует при своем проведении байт-

ориентированную структуру шифра. Учитывая, что RIJNDAEL унаследовал многие свойства шифра SQUARE, эта атака применима и к нему. Далее приведено описание атаки "Квадрат" применительно к RIJNDAEL.

Атака "Квадрат" основана на возможности свободного подбора атакующим некоторого набора открытых текстов для последующего их зашифрования. Она независима от таблиц замен блоков, многочлена функции MixColumns() и способа разворачивания ключа. Эта атака для 6-раундового шифра RIJNDAEL, состоящего из 6 раундов, эффективнее, чем полный перебор по всему ключевому пространству. После описания базовой атаки на 4-раундовый RUNDAEL, будет показано, как эту атаку можно продлить на 5 и даже 6 раундов. Но уже для 7 раундов “Квадрат” становится менее эффективным, чем полный перебор.

Предпосылки

Пусть L-набор - такой набор из 256 входных блоков (масивов State), каждый из которых имеет байты (назовем их активными), значения которых различны для всех 256 блоков. Остальные байты (будем называть их пассивными) остаются одинаковыми для всех 256 блоков из L-набора. То есть:

$$\forall x, y \in L \begin{cases} x_{ij} \neq y_{ij} & \text{, если байт с номером ij активный;} \\ x_{ij} = y_{ij} & \text{, в противном случае.} \end{cases}$$

Будучи подвергнутыми обработке функциями SubBytes() и AddRoundKey() блоки L-набора дадут в результате другой L-набор с активными байтами в тех же позициях, что и у исходного. Функция ShiftRows() сместит эти байты соответственно заданным в ней смещениям в строках массивов State. После функции MixColumns() L-набор в общем случае необязательно останется L-набором (т. е. результат преобразования может перестать удовлетворять определению L-набора). Но поскольку каждый байт результата функции MixColumns() является линейной комбинацией (с обратимыми коэффициентами) четырех входных байт того же столбца:

$$b_{ij} = 2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j}$$

столбец с единственным активным байтом на входе даст в результате на выходе столбец со всеми четырьмя байтами - активными.

Базовая атака “Квадрат” на 4 раунда

Рассмотрим L-набор, во всех блоках которого активен только один байт. Иначе говоря, значение этого байта различно во всех 256 блоках, а остальные байты одинаковы (скажем, равны нулю). Проследим эволюцию этого байта на протяжении трех раундов. В первом раунде функция MixColumns() преобразует один активный байт в столбец из 4

активных байт. Во втором раунде эти 4 байта разойдутся по 4 различным столбцам в результате преобразования функцией ShiftRows(). Функция же MixColumns() следующего, третьего раунда преобразует эти байты в 4 столбца, содержащие активные байты. Этот набор все еще остается L-набором до того момента, когда он поступает на вход функции MixColumns() третьего раунда.

Основное свойство L-набора, используемое здесь, то, что поразрядная сумма по модулю 2 всех блоков такого набора всегда равна нулю. Действительно, поразрядная сумма неактивных (с одинаковыми значениями) байт равна нулю по определению операции поразрядного XOR, а активные байты, пробегая все 256 значений, также при поразрядном суммировании дадут нуль. Рассмотрим теперь результат преобразования функцией MixColumns() в третьем раунде байтов входною массива данных a в байты выходного массива данных b. Покажем, что и в этом случае поразрядная сумма всех блоков выходного набора будет равна нулю, то есть:

$$\bigoplus_{b=mixcolumns(a),a\in L} b_{ij} = \bigoplus_{a\in L}(2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j}) =$$

$$= 2\bigoplus_{a\in L} a_{ij} \oplus 3\bigoplus_{a\in L} a_{(i+1)j} \bigoplus_{a\in L} a_{(i+2)j} \bigoplus_{a\in L} a_{(i+3)j} = 2\cdot 0 \oplus 3\cdot 0 \oplus 1\cdot 0 \oplus 1\cdot 0 = 0$$

Таким образом, вес данные на входе четвертого раунда сбалансированы (то есть их полная сумма равна нулю). Этот баланс в общем случае нарушается последующим преобразованием данных функцией SubBytes().

Мы предполагаем далее, что четвертый раунд является последним, то есть в нем нет функции MixColumns(). Тогда каждый байт выходных данных этого раунда зависит только от одного байта входных данных. Если обозначить через a байт выходных данных четвертого раунда, через b байт входных данных и через k - соответствующий байт раундового ключа, то можно записать:

$$a_{ij} = SubBytes(b_{ij}) \oplus k_{ij}$$

Отсюда, предполагая значение k_{ij} можно по известному a_{ij} вычислить b_{ij}, а затем проверить правильность догадки о значении k_{ij}; если значения байта b_{ij}, полученные при данном k_{ij} не будут сбалансированы по всем блокам (то есть не дадут при поразрядном суммировании нулевой результат), значит догадка неверна. Перебрав максимум 2^8 вариантов байта раундового ключа, мы найдем его истинное значение.

По такому же принципу могут быть определены и другие байты раундового ключа. За счет того, что поиск может производиться отдельно (читай - параллельно) для каждого байта ключа, скорость подбора всего значения раундового ключа весьма велика. А по значению полного раундового ключа, при известном алгоритме его развертывания, не составляет труда восстановить сам начальный ключ шифрования.

Добавление пятого раунда в конец базовой атаки “Квадрат”

Если будет добавлен пятый раунд, то значения b_{ij} придется вычислять уже на основании выходных данных не четвертого, а пятого раунда. И дополнительно кроме байта раундового ключа четвертого раунда перебирать еще значения четырех байт столбца раундового ключа для пятого раунда. Только так мы сможем выйти на значения сбалансированных байт b_{ij} входных данных четвертого раунда.

Таким образом, теперь нам нужно перебрать 2^{40} значений - 2^{32} вариантов для 4 байт столбца раундового ключа пятого раунда и для каждого из них 2^8 вариантов для одного байта четвертого раунда. Эту процедуру нужно будет повторить для всех четырех столбцов пятого раунда. Поскольку при подборе "верного" значения байта раундового ключа четвертого раунда количество "неверных" ключей уменьшается в 2^8 раз, то работая одновременно с пятью L-наборами, можно с большой степенью вероятности правильно подобрать все 2^{40} бита. Поиск может производиться отдельно (т. е. параллельно) для каждого столбца каждого из пяти L-наборов, что опять же гораздо быстрее полного перебора всех возможных значений ключа.

Добавление шестого раунда в начало базовой атаки “Квадрат”

Основная идея заключается в том, чтобы подобрать такой набор блоков открытого текста, который на выходе после первого раунда давал бы L-набор с одним активным байтом. Это требует предположения о значении четырех байт ключа, используемых функцией AddRoundKey() перед первым раундом.

Для того чтобы на входе второго раунда был только один активный байт достаточно, чтобы в первом раунде один активный байт оставался на выходе функции MixColumns(). Это означает, что на входе MixColumns() первого раунда должен быть такой столбец, байты а которого для набора из 256 блоков в результате линейного преобразования:

$$b_{ij} = 2a_i \oplus 3a_{i+1} \oplus a_{i+2} \oplus a_{i+3}, 0 \le i \le 3,$$

где i – номер строки, для одного определенного i давали 256 различных значений, в то время как для каждого из остальных трех значений i результат этого преобразования должен оставаться постоянным. Следуя обратно по порядку приложения функций преобразования в первом раунде, к ShiftRows() данное условие нужно применить к соответственно разнесенным по столбцам 4 байтам. С учетом применения функции SubBytes() и сложения с предполагаемым значением 4-байтового раундового ключа можно смело составлять уравнения и подбирать нужные значения байт открытого текста, подаваемых на зашифрование для последующего анализа результата:

$$b_{ij} = 2SubBytes(a_{ij} \oplus k_{ij}) \oplus 3SubBytes(a_{(i+1)(j+1)} \oplus k_{(i+1)(j+1)}) \oplus$$
$$\oplus SubBytes(a_{(i+2)(j+2)} \oplus k_{(i+2)(j+2)}) \oplus SubBytes(a_{(i+3)(j+3)} \oplus k_{(i+3)(j+3)}), 0 \le i, j \le 3$$

Таким образом, получаем следующий алгоритм взлома. Имеем всего 2^{32} различных значений a для определенных i и j. Остальные байты для всех блоков одинаковы (пассивные байты). Предположив значения четырех байт k ключа первого раунда, подбираем (исходя из вышеописанного условия) набор из 256 блоков. Эти 256 блоков станут L-набором после первого раунда. К этому L-наборе применима базовая атака для 4 раундов. Подобранный с ее помощью один байт ключа последнего раунда фиксируется. Теперь подбирается новый набор из 256 блоков для того же значения 4 байт k ключа первого раунда. Опять осуществляется базовая атака, дающая один байт ключа последнего раунда. Если после нескольких попыток значение этого байта не меняется, значит мы на верном пути. В противном случае нужно менять предположение о значении 4 байт k ключа первого раунда. Такой алгоритм действий достаточно быстро приведет к полному восстановлению всех байт ключа последнего раунда.

Таким образом, атака "Квадрат" может быть применена к 6 раундам шифра RIJNDAEL, являясь при этом более эффективной, чем полный перебор по всему ключевому пространству. Любое известное продолжение атаки "Квадрат" на 7 и более раундов становится более трудоемким, чем даже обычный полный перебор значений ключа.

3 Описание лабораторной установки и методики измерений

3.1 Интерфейс учебно-программного комплекса

Учебно-программный комплекс (в дальнейшем просто комплекс) был разработан на языке высокого уровня Visual Basic.

Главное окно

На рисунке 11 приведен интерфейс главного окна комплекса.

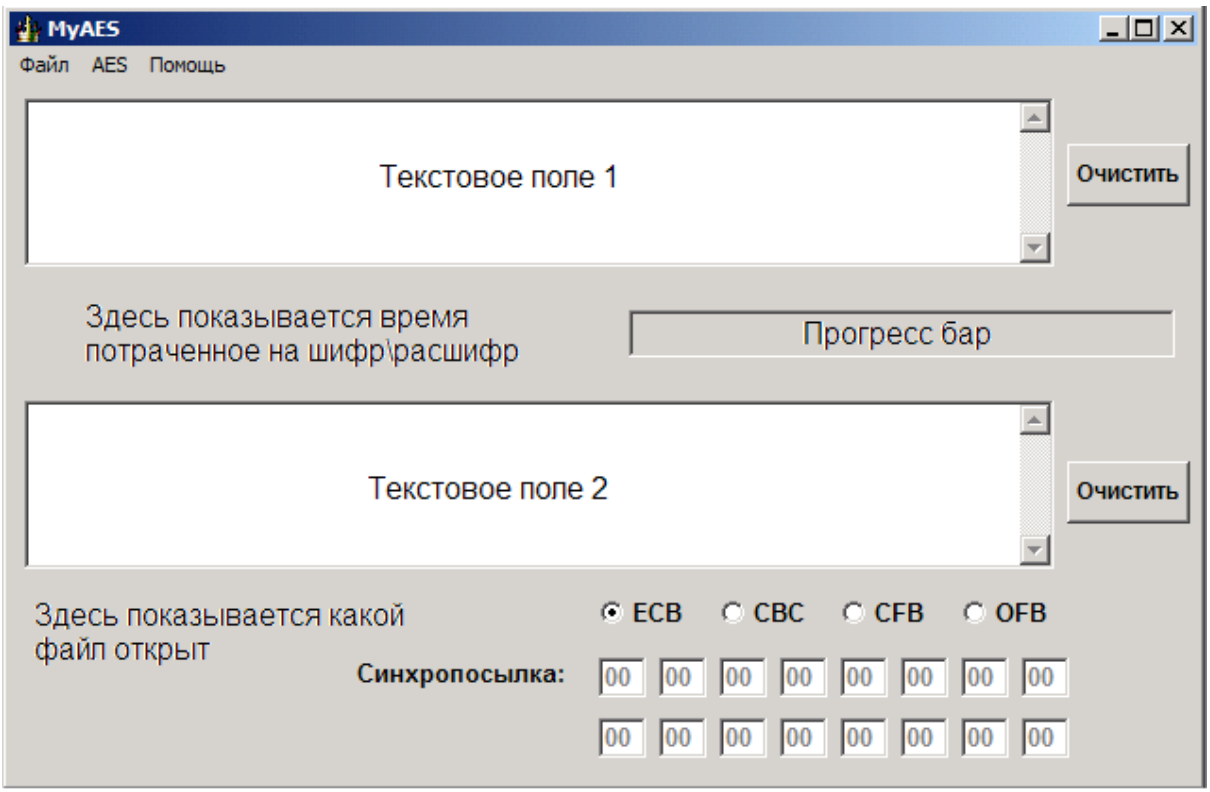


Рис. 11. Интерфейс главного окна комплекса

Главное окно состоит из:

1. Текстовое поле 1 – сюда вводится открытый текст, также если открывается файл с расширением txt, то его содержимое помещается сюда же.

2. Текстовое поле 2 – сюда помещается зашифрованный текст, также если открывается файл с расширением aes, то его содержимое помещается сюда же.

3. Прогресс бар – наглядно показывает время требуемое для шифрования \ расшифрования.

4. Кнопки “Очистить” – удаляют весь текст из текстовых полей.

5. Переключатели ECB, CBC, CFB, OFB – показывают, какой режим блочного шифрования используется.

6. Синхропосылка – задает синхропосылку для CBC, CFB и OFB.

В главном окне и происходит шифрование и расшифрование. Также в нем можно выбрать режимы симметричного шифрования (ECB, CBC, CFB, OFB). Про них сказано ниже.

Пункт меню “Файл”

На рисунке 12 приведен пункт меню “Файл”:

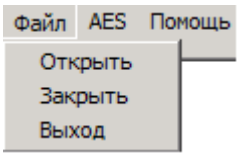


Рис. 12. Пункт меню “Файл”

Пункт меню “Файл” содержит следующие опции:

1. Открыть – открывает файл (*.txt, *.aes), но не больше 32kB для текстовых файлов и 64кВ для файлов типа aes.

2. Закрыть – закрывает выбранный файл.

3. Выход – завершает работу комплекса.

Пункт меню “AES”

На рисунке 13 приведен пункт меню “AES”:

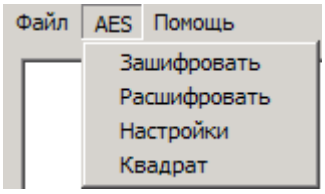


Рис. 13. Пункт меню “AES”

Пункт меню “AES” содержит следующие опции:

1. Зашифровать – шифрует текст, расположенный в текстовом поле 1, и помещает его в текстовое поле 2.

2. Расшифровать - расшифровывает текст, расположенный в текстовом поле 2, и помещает его в текстовое поле 1.

3. Настройки – позволяет выбрать свой ключ шифрования, а также включить \ выключить режим логирования. (рисунок 14)

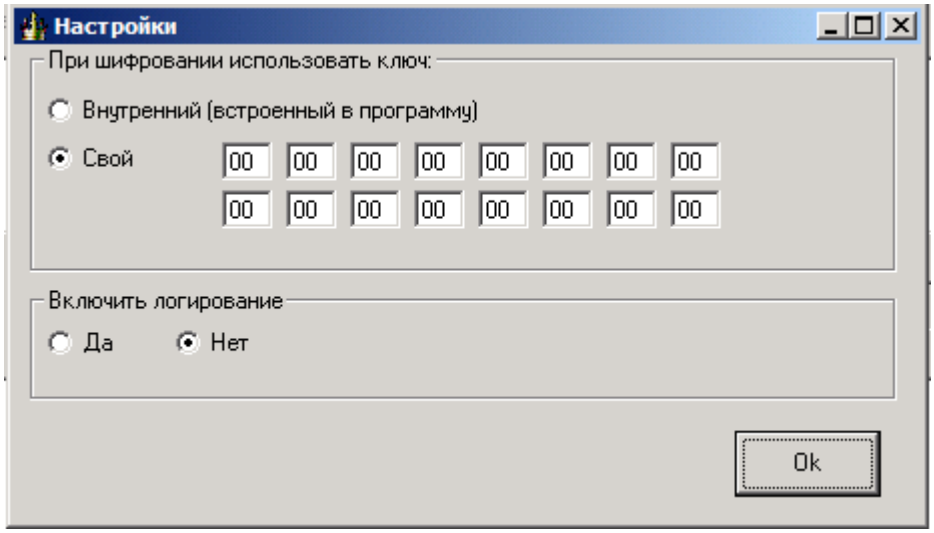


Рис. 14. Окно “Настройки”

В этом окне вы можете ввести свой ключ шифрования (например из задания по лабораторной работе) или же использовать внутренний. При использовании внутреннего ключа шифрования вы можете шифровать свои сообщения и отправлять их кому-либо. Причем получателю не обязательно знать ключ, главное чтобы у него была данная программа.

Здесь же вы можете включить логирование. При включенном логировании в папке программы создается Encrypt.txt (Decrypt.txt), в зависимости от того, что вы делаете шифруете или расшифровываете. В этом файле содержатся все преобразования, происходящие с открытым текстом. Пример такого файла приведен на рисунке 15:

```
Зашифрование
Входной блок CFF0EEE2E5F0EAE00101010101010101
Раундовый ключ: 00000000000000000000000000000000
Состояние после сложения с ключом: CFF0EEE2E5F0EAE00101010101010101
Раунд №1
Состояние после SubBytes: 8A8C2898D98C87E17C7C7C7C7C7C7C7C
Состояние после ShiftRows: 8A8C7C7CD97C7C987C7C28E17C8C877C
Состояние после MixColumns: 80717A8DC93DEE5BB51D68098C916177
Раундовый ключ: 62636363626363636263636362636363
Состояние после сложения с ключом: E21219EEAB5E8D38D77E0B6AEEF20214
Раунд №2
Состояние после SubBytes: 98C9D42862585D070EF32B02288977FA
Состояние после ShiftRows: 98582BFA62F377280E89D40728C95D02
Состояние после MixColumns: 12AF832F952E07724F673D414F445DE8
Раундовый ключ: 9B9898C9F9FBFBAA9B9898C9F9FBFBAA
Состояние после сложения с ключом: 89371BE66CD5FCD8D4FFA588B6BFA642
Раунд №3
Состояние после SubBytes: A79AAF8E5003B061481606C44E08242C
Состояние после ShiftRows: A703062C5016248E4808AF614E9AB0C4
Состояние после MixColumns: 7A87DCAF309E87C546D3A6BD5D6EF86B
Раундовый ключ: 90973450696CCFFAF2F457330B0FAC99
Состояние после сложения с ключом: EA10E8FF59F2483FB427F18E566154F2
```

Рис. 15. Зашифрование слова “Проверка”

Логирование работает только в режиме ECB, и автоматически отключается при открытии файла или использовании другого режима.

4. Квадрат – простенькая реализация единственной атаки на AES, под названием квадрат. Реализована для сокращенной версии AES, для 4 раундов. В стандарте же их 10.

Окно Атака «Квадрат»

Смысл атаки “Квадрат” описан выше.

В этом окне (рисунок 16) можно просмотреть ключи для любого раунда. При открытии этого окна в поля “Реальный ключ” загружается ключ, заданный в настройках. При нажатии кнопок “Up” или ”Down” будет меняться номер раунда, а также поля “Реальный ключ”. Они будут равны тому раундовому ключу, который указан в строке “Номер раунда”. Нас будет интересовать ключ 4 раунда поэтому, лучше поставить значение поля “Номер раунда” в 4.

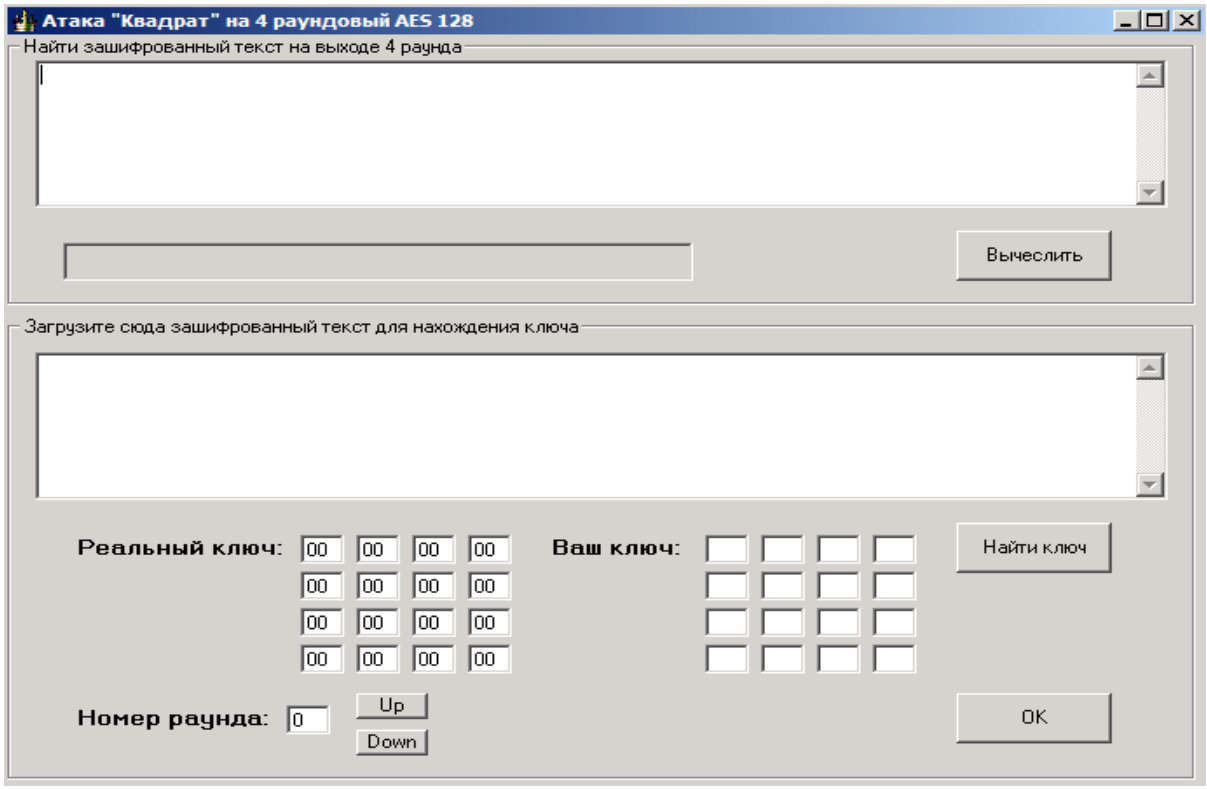


Рис. 16. Окно “Квадрат”

При нажатии кнопки “Вычислить” в текстовом поле 1 появляется зашифрованный L набор после 4 раунда. Прогресс бар наглядно показывает время требуемое для его вычисления. Пример L набора, зашифрованного ключом “00”{32}, показан на рисунке 17:

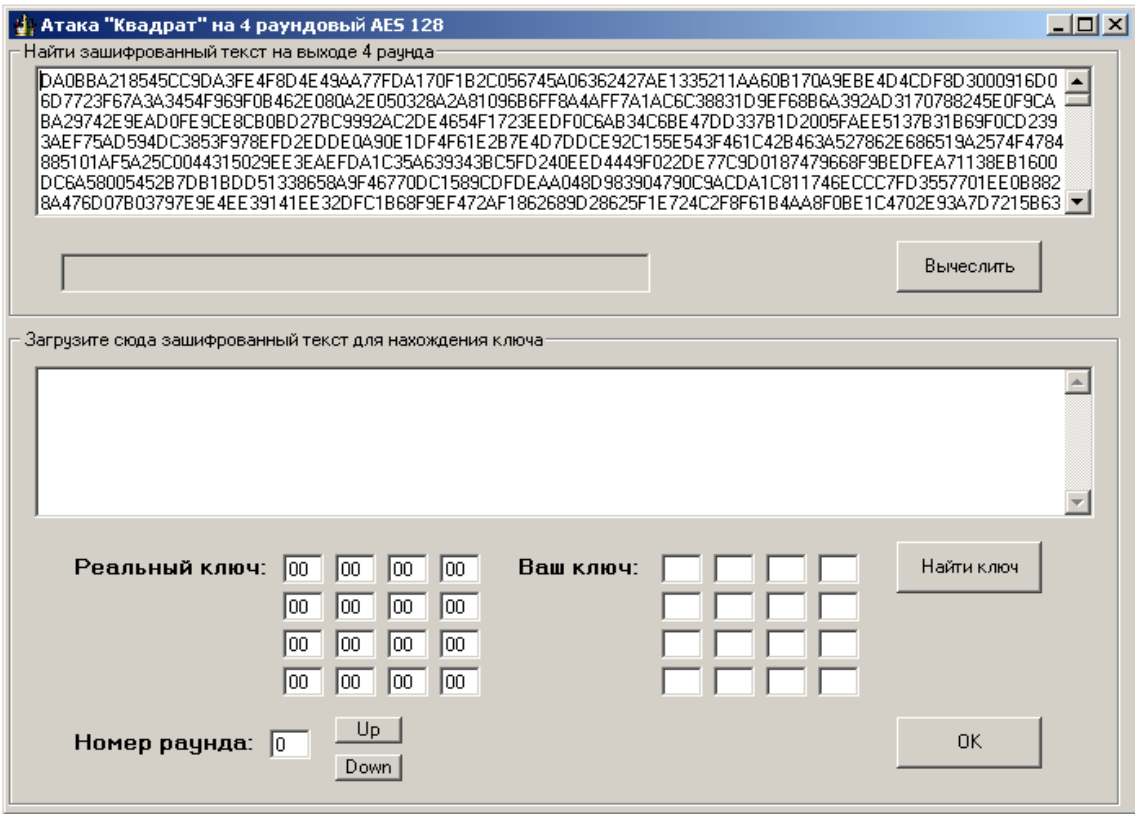


Рис. 17. Пример L набора, зашифрованного ключом “00”{32}

Этот текст нужно скопировать и вставить в текстовое поле 2.

При нажатии на кнопку “Найти ключ” начнется вычисление ключа 4 раунда на основе текста в текстовом поле 2. Прогресс бар наглядно показывает время, требуемое для его вычисления.

После вычисления ключа он появится в окошках “Ваш ключ”. Можно сравнить окна “Ваш ключ” и “Реальный ключ” и увидеть, что они практически совпадают (рисунок 18).

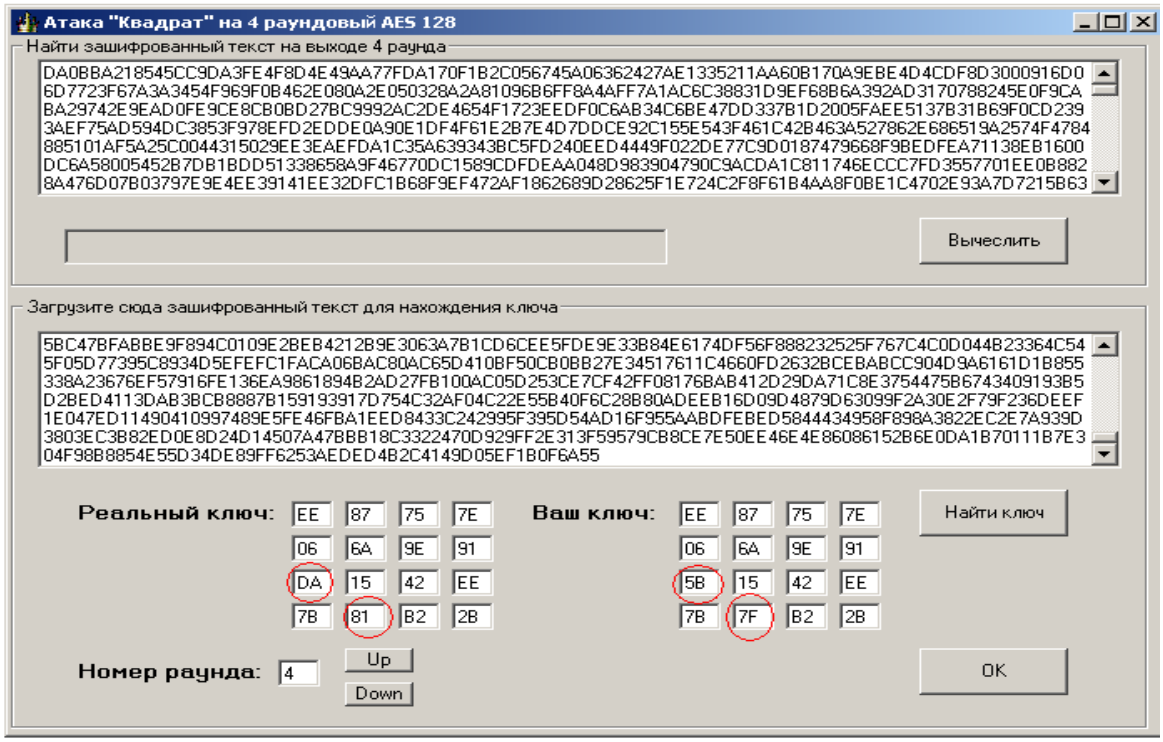


Рис. 18. Пример вычисления раундового ключа

Режимы ECB, CBC, CFB, OFB

Для различных ситуаций, встречающихся на практике, разработано значительное количество режимов шифрования.

Режим ECB (Electronic Code Book – режим электронной кодовой книги)

В режиме ECB каждый блок открытого текста заменяется блоком шифротекста. А так как один и тот же блок открытого текста заменяется одним и тем же блоком шифротекста, теоретически возможно создать кодовую книгу блоков открытого текста и соответствующих шифротекстов. Но если размер блока составляет n бит, кодовая книга будет состоять из 2n записей.

Режим ECB - простейший режим шифрования. Все блоки открытого текста шифруются независимо друг от друга. Это важно для шифрованных файлов с произвольным доступом, например, файлов баз данных. Если база данных зашифрована в режиме ECB, любая запись может быть добавлена, удалена, зашифрована или расшифрована независимо от любой другой записи (при условии, что каждая запись состоит из целого числа блоков шифрования). Кроме того, обработка может быть параллельной: если используются несколько шифровальных процессоров, они могут шифровать или расшифровывать различные блоки независимо друг от друга. Режим ECB показан на рисунке 19:

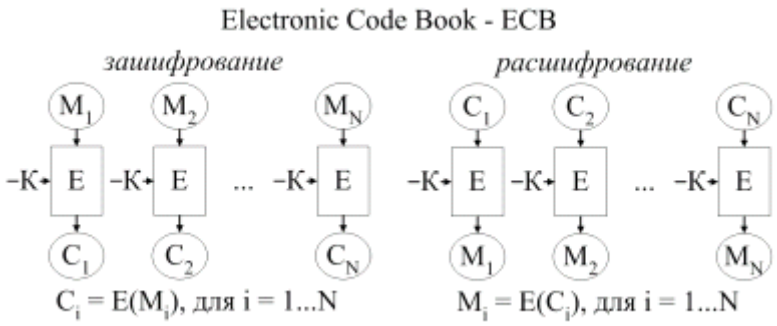


Рисунок 19 Режим ECB

К недостаткам режима ECB можно отнести то обстоятельство, что если у криптоаналитика есть открытый текст и шифротекст нескольких сообщений, он может, не зная ключа, начать составлять шифровальную книгу. В большинстве реальных ситуаций фрагменты сообщений имеют тенденцию повторяться. В различных сообщениях могут быть одинаковые битовые последовательности, В сообщениях, которые, подобно электронной почте, создаются компьютером, могут быть периодически повторяющиеся структуры. Сообщения могут быть высоко избыточными или содержать длинные строки нулей или пробелов.

К достоинствам режима ECB можно отнести возможность шифрования нескольких сообщений одним ключом без снижения надежности. По существу, каждый, блок можно рассматривать как отдельное сообщение, шифрованное тем же самым ключом. При расшифровании ошибки в символах шифротекста ведут к некорректному расшифрованию соответствующего блока открытого текста, однако не затрагивают остальной открытый текст. Но если бит шифротекста случайно потерян или добавлен, весь последующий шифротекст будет дешифрован некорректно, если только для выравнивания границ блоков не используется какое-нибудь выравнивание по границам блока. Большинство сообщений не делятся точно на n - битовые блоки шифрования – в конце обычно оказывается укороченный блок. Однако режим ECB требует использовать строго n - битовые блоки. Для решения этой проблемы используют дополнение (padding), Чтобы создать полный блок, последний блок дополняют некоторым стандартным шаблоном - нулями, единицами, чередующимися нулями и единицами.

Режим CBC (Ciphertext Block Chaining – режим сцепления блоков шифротекста)

Сцепление добавляет в блочный шифр механизм обратной связи: результаты шифрования предыдущих блоков влияют на шифрование текущего блока, т.е. каждый блок используется для модифицирования шифрования следующего блока. Каждый блок шифротекста зависит не только от шифруемого блока открытого текста, но и от всех предыдущих блоков открытого текста.

В режиме сцепления блоков шифротекста перед шифрованием над открытым текстом и предыдущим блоком шифротекста выполняется операция XOR. Процесс шифрования в режиме CBC. Когда блок открытого текста зашифрован, полученный шифротекст сохраняется в регистре обратной связи. Следующий блок открытого текста перед шифрованием подвергается операции XOR с содержимым регистра обратной связи. Результат операции XOR используется как входные данные для следующего этапа процедуры шифрования. Полученный шифротекст снова сохраняется в регистре обратной связи, чтобы подвергнуться операции XOR вместе со следующим блоком открытого текста, и так до конца сообщения. Шифрование каждого блока зависит от всех предыдущих блоков.

Расшифрование выполняется в обратном порядке. Блок шифротекста расшифровывается обычным путем, но сохраняется в регистре обратной связи. Затем следующий блок расшифровывается и подвергается операции XOR с содержимым регистра обратной связи. Теперь следующий блок шифротекста сохраняется в регистре обратной связи и т.д. до конца сообщения.

На рисунке 20 показан режим CBC (IV – синхропосылка):

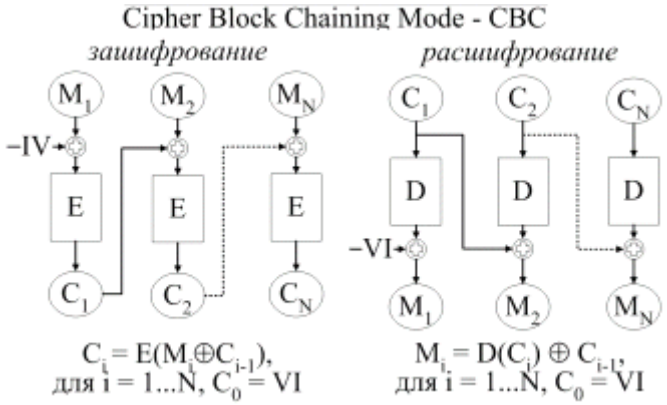


Рис. 20. Режим CBC

При шифровании в режиме CBC одинаковые блоки открытого текста превращаются в различающиеся друг от друга блоки шифротекста только в том случае, если различались какие-то предшествующие блоки открытого текста. Однако при шифровании двух 48 идентичных сообщений создается один и тот же шифротекст. Хуже того, два одинаково начинающихся сообщения будут шифроваться одинаково вплоть до первого различия.

Чтобы избежать этого, можно зашифровать в первом блоке какие-то произвольные данные. Этот блок случайных данных называют вектором инициализации (ВИ) (Initialization Vector, IV, русский термин - синхропосылка), инициализирующей переменной или начальным значением сцепления. Вектор ВИ не имеет какого-то

смыслового значения, он используется только для того, чтобы сделать каждое сообщение уникальным. Когда получатель расшифровывает этот блок, он использует его только для заполнения регистра обратной связи. В качестве вектора ВИ удобно использовать метку времени, либо какие-то случайные биты.

Если используется вектор инициализации, сообщения с идентичным открытым текстом после шифрования превращаются в сообщения с разными шифротекстами. Следовательно, злоумышленник не может попытаться повторить блок, и создание шифровальной книги затруднится. Хотя для каждого сообщения, шифруемого одним и тем же ключом, рекомендуется выбирать уникальный вектор ВИ, это требование необязательное. Вектор ВИ не обязательно хранить в секрете, его можно передавать открыто - вместе с шифротекстом.

Режим CFB (Ciphertext Feedback – обратная связь по шифротексту)

В режиме CBC начать шифрование до поступления полного блока данных невозможно. Для некоторых сетевых приложений это создает проблемы. Например, в защищенном сетевом окружении терминал должен иметь возможность передавать хосту каждый символ сразу после ввода. Если же данные нужно обрабатывать блоками в несколько байт, режим CBC просто не работает.

На рисунке 21 показан режим CFB (IV – синхропосылка):

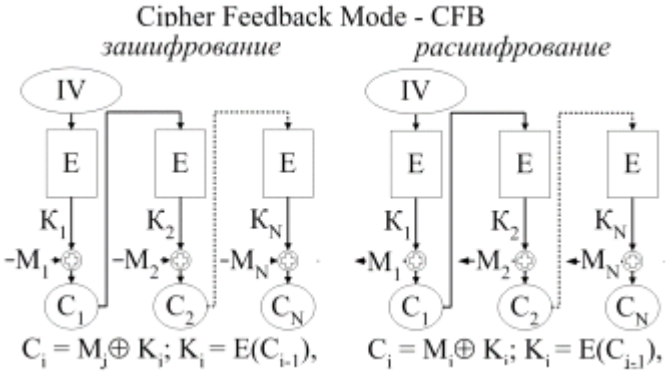


Рис. 21. Режим CFB

Как и в режиме CBC, первоначально очередь заполнена вектором инициализации ВИ. Очередь шифруется, затем выполняется операция XOR над *n* старшими (крайними левыми) битами результата и первым *n* -битовым символом открытого текста. В результате появляется первый *n* -битовый символ шифротекста. Теперь этот символ можно передать. Кроме того, полученные *n* битов попадают в очередь на место *n* младших битов, а все остальные биты сдвигаются на *n* позиций влево. Предыдущие *n* старших битов отбрасываются. Затем точно также шифруется следующие *n* битов открытого текста. Расшифрование выполняется в обратном порядке. Обе стороны - шифрующая и расшифровывающая - использует блочный алгоритм в режиме шифрования. Как и режим

CBC, режим CFB сцепляет символы открытого текста с тем, чтобы шифротекст зависел от всего предыдущего открытого текста.

Для инициализации процесса шифрования в режиме CFB в качестве входного блока алгоритма можно использовать вектор инициализации ВИ. Как и в режиме CBC, хранить в тайне вектор ВИ не нужно. Однако вектор ВИ должен быть уникальным. (В отличие от режима CBC, где уникальность вектора ВИ необязательна, хотя и желательна). Если вектор ВИ в режиме CFB не уникален, криптоаналитик может восстановить соответствующий открытый текст. Вектор инициализации должен меняться в каждом сообщении. Например, вектором ВИ может служить порядковый номер, возрастающий в каждом новом сообщении и не повторяющийся все время жизни ключа. Если данные шифруются с целью последующего хранения, вектор ВИ может быть функцией индекса, используемого для поиска данных.

Режим OFB (Output Feedback – режим обратной связи по выходу)

Режим OFB представляет собой метод использования блочного шифра в качестве синхронного потокового шифра. Этот режим подобен режиму CFB.

Блочный алгоритм работает в режиме шифрования как на шифрующей, так и на расшифровывающей сторонах. Такую обратную связь иногда называют внутренней, поскольку механизм обратной связи не зависит ни от потока открытого текста, ни от потока шифротекста.

К достоинствам режима OFB относится то, что большую часть работы можно выполнить оффлайново, даже когда открытого текста сообщения еще вовсе не существует. Когда, наконец, сообщение поступает, для создания шифротекста над сообщением и выходом алгоритма необходимо выполнить операцию XOR.

В сдвиговый регистр OFB сначала надо загрузить вектор ВИ. Вектор должен быть уникальным, но сохранять его в тайне не обязательно.

На рисунке 22 показан режим OFB (IV – синхропосылка):

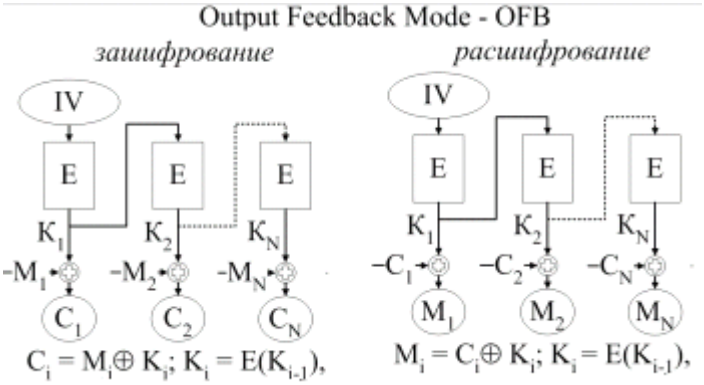


Рис. 22. Режим OFB

Анализ режима OFB показывает, что OFB целесообразно использовать только если разрядность обратной связи совпадает с размером блока.

В режиме OFB над гаммой и текстом выполняется операция XOR. Эта гамма, в конце концов, повторяется. Существенно, чтобы она не повторялась для одного и того же ключа, в противном случае секретность не обеспечивается ничем.

3.2 Порядок выполнения работы

1. Ознакомится с теорией по стандарту AES (из чего состоит раунд шифрования, как работают раундовые преобразования, с необходимым математическим аппаратом).

2. Взять у преподавателя задание, состоящие из:

- уникального ключа для шифрования/дешифрования текстовых сообщений
- набора файлов разного размера для исследования стандарта
- файла для исследования режимов шифрования
- вектора инициализации
- файла для проведения атаки «Квадрат»

3. Исследование стандарт шифрования

3.1. Открыть окно «Настройки» (AES>Настройки>), установить переключатель в положение «Свой ключ» и ввести посимвольное значение ключа (32 символа, 16 ячеек). Нажать «ОК».

3.2. В главном окне поставить переключатель в положение ECB.

3.3. Открыть файл из папки программы (Файл >Открыть)

3.4. Зашифровать и расшифровать его. (AES > Зашифровать (Расшифровать)).

3.5. Проделать пункты 3.2-3.4 для оставшихся трех файлов.

3.5. По полученным результатам заполнить таблицу.

Размер файла				
Время зашифрования				
Время расшифрования				

3.6. Построить графики зависимости размера файла от времени его шифрования \ расшифрования. На рисунке 23 приведен пример графика зависимости размера файла от времени его шифрования \ расшифрования при проведении эксперимента на Athlon 2600+.

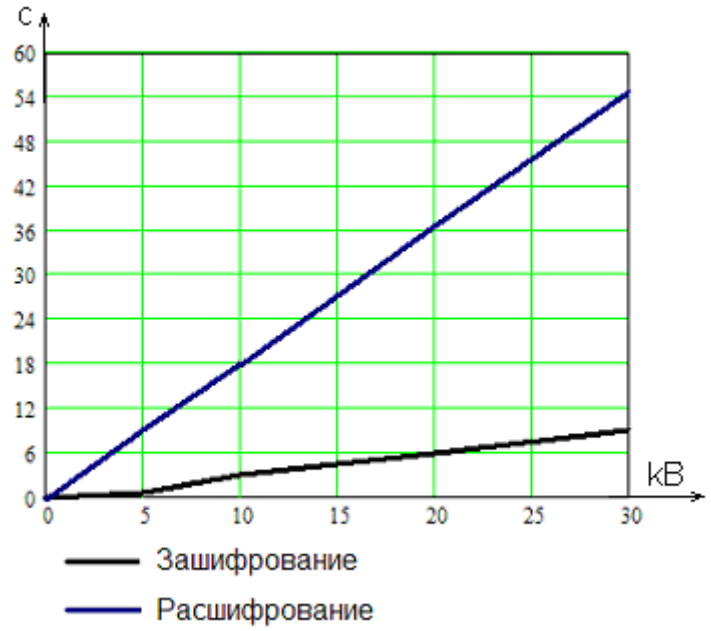


Рис. 23. Пример зависимости размера файла от времени его шифрования \ расшифрования при проведении эксперимента на Athlon 2600+

3.7. По полученным данным оценить примерное время взлома стандарта AES с помощью всего перебора ключей. Построить график.

Пример:

На расшифрование 5кВ (5120 байт) текста на Athlon 2600+ тратится 9с. Поэтому один блок (16 байт) программа расшифровывает за 9/(5120/16) = (9*16)/5120 = 28 мс. Таким образом, на перебор 1 ключа, а это 128 бит, потребуется 28 мс.

За 1 секунду программа способна перебрать 4571 бит или примерно 35 вариантов ключей.

За год непрерывной работы программа переберет (35*60*60*24*365) = 1 103 760 000 ключей.

Если поделить 2^{128} на 1 103 760 000 , то получим $3.08*10^{29}$ лет непрерывной работы программы.

3.8. Сделать выводы

4. Исследование быстродействия различных режимов шифрования

4.1. Поставить переключатель в режим CBC.

4.2. Открыть файл из папки программы (Файл >Открыть)

4.3. Зашифровать и расшифровать его (AES > Зашифровать (Расшифровать))

4.4. Повторить пункт 4.2-4.3 для оставшихся трех файлов

4.5. Поставить переключатель в режим CFB.

4.6. Повторить пункты 4.2 – 4.3.

4.7. Поставить переключатель в режим OFB.

4.8. Повторить пункты 4.2 – 4.3

4.9. По полученным результатам заполнить таблицу:

Размер файла				
ECB время заш/расш				
CBC время заш/расш				
CFB время заш/расш				
OFB время заш/расш				

4.10. Построить на одном графике зависимости размера файла от времени его шифрования \ расшифрования

4.11. Для каждого режима загрузить еще раз файл (любой). Зашифровать его. В текстовом поле 2 изменить какой-нибудь символ. Нажать AES > Расшифровать.

4.12. Сделать выводы.

5. Исследование свойств различных режимов шифрования

5.12. Загрузить файл «Свойства режимов».

5.13. Поочередно зашифровать текстовое сообщение во всех четырех режимах.

5.14. Сравнить полученные шифротексты и сделать выводы.

6. Исследование пораундовой работы алгоритма

6.1 Открыть любой из предоставленных для работы файлов

6.2. Включить функцию логирования (AES>Настройки>Включить логирование>Да).

6.3. В главном окне поставить переключатель в положение ECB.

6.4. Зашифровать и расшифровать.

6.5. Открыть с помощью текстового редактора файлы Encrypt.txt и Decrypt.txt из папки программы

6.6. Проанализировать полученный результат.

7. Знакомство с атакой «Квадрат»

7.1. В главном окне в текстовое поле 1 загрузить файл «Атака Квадрат» (Файл>Открыть)

7.2. Открыть окно Атака «Квадрат» (AES > Квадрат)

7.3. С помощью переключателя “Номер раунда” установить 4 раундовый ключ.

7.4. Нажать кнопку “вычислить”.

7.5. Поместить текст из поля 1 в поле 2.

7.6. Нажать кнопку найти ключ.

7.7. Сравнить значения текстовых полей “Ваш ключ” и значения текстовых полей “Реальный ключ”.

7.8. Сделать выводы.

8. Содержание отчета

8.1. Цель работы.

8.2. Описание действий по каждому пункту.

8.3. Результаты проделанной работы (таблицы, графики, расчетная часть).

8.4. Выводы.

4 Контрольные вопросы

1. Что такое симметричное шифрование?
2. Что представляет собой стандарт AES (длина ключа, размер входного блока)?
3. Какой алгоритм выбран в качестве стандарта AES?
4. Что собой представляет архитектура данного стандарта?
5. Из чего состоит один раунд?
6. Сколько раундов шифрования предусмотрено стандартом?
7. Что быстрее шифрование или расшифрование? Почему?
8. Какие режимы шифрования применяются в стандарте AES?
9. Какие режимы быстрее при расшифровании? Почему?
10. Какие режимы лучше восстанавливают зашифрованную информацию при ошибке в одном символе? Почему?
11. С какой целью используется синхропосылка или вектор инициализации?
12. Что представляет собой атака Квадрат? Какие ее особенности?

5. Рекомендуемая литература

1. Метлицкий Ю.В. Разработка программного комплекса для визуализации и анализа стандарта криптографической защиты AES, МИФИ 2003 год.
2. www.intuit.ru, Криптографические основы информации.
3. Зензин О.С., Иванов М.А. Стандарт криптографической защиты – AES. Конечные поля. – М: КУДИЦ-ОБРАЗ, 2002-176с.