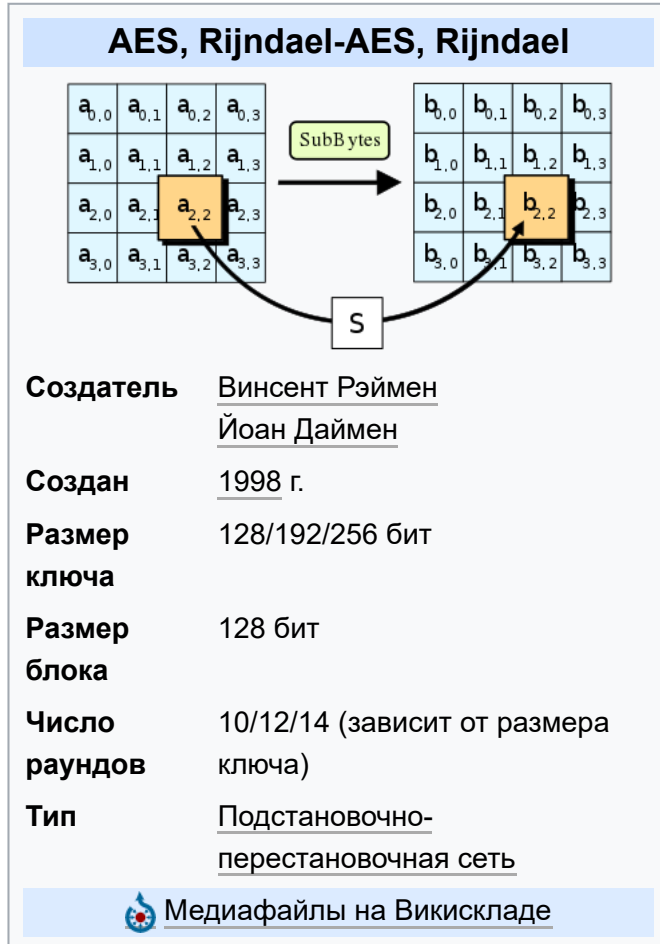


# AES (стандарт шифрования)

Материал из Википедии — свободной энциклопедии

**AES** (англ. *Advanced Encryption Standard*; также *Rijndael*, [ˈɹeɪndɑːl] — *рейнда́л*) — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником **DES**. Национальный институт стандартов и технологий США (англ. *National Institute of Standards and Technology*, NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года AES был объявлен стандартом шифрования. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования<sup>[1][2]</sup>. Поддержка ускорения AES была введена фирмой Intel в семейство процессоров x86 начиная с Arrandale в 2010 году, а затем на процессорах Sandy Bridge; фирмой AMD — в Bulldozer с 2011 года.



# Содержание

## История AES

## Описание AES

## Определения и вспомогательные процедуры

## Шифрование

## SubBytes()

## ShiftRows()

## MixColumns()

AddRoundKey()

## Алгоритм обработки ключа

## Алгоритм генерации раундовых ключей

## Расшифрование

## Алгоритм выбора раундового ключа

## Варианты алгоритма

## Криптостойкость

[XSL-атака](#)

[Атака по сторонним каналам](#)

[См. также](#)

[Примечания](#)

[Литература](#)

[Ссылки](#)

## История AES

2 января 1997 года NIST объявляет<sup>[3]</sup> о намерении выбрать преемника для DES, являвшегося американским стандартом с 1977 года. 2 октября 2000 года было объявлено, что победителем конкурса стал алгоритм Rijndael<sup>[4]</sup>, и началась процедура стандартизации. 28 февраля 2001 года был опубликован проект, а 26 ноября 2001 года AES был принят как FIPS 197. Историческую ретроспективу конкурса можно проследить на веб-сайте NIST<sup>[5]</sup>.

## Описание AES

### Определения и вспомогательные процедуры

#### Определения

<b>Block</b>	последовательность бит, из которых состоит input, output, State и Round Key. Также под Block можно понимать последовательность байтов
<b>Cipher Key</b>	секретный криптографический ключ, который используется Key Expansion процедурой, чтобы произвести набор ключей для раундов (Round Keys); может быть представлен как прямоугольный массив байтов, имеющий четыре строки и $Nk$ колонок
<b>Ciphertext</b>	выходные данные алгоритма шифрования
<b>Key Expansion</b>	процедура генерации Round Keys из Cipher Key
<b>Round Key</b>	Round Keys получаются из Cipher Key использованием процедуры Key Expansion. Они применяются к State при шифровании и расшифровании
<b>State</b>	промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов, имеющий 4 строки и $Nb$ колонок
<b>S-box</b>	нелинейная таблица замен, используемая в нескольких трансформациях замены байтов и в процедуре Key Expansion для взаимнооднозначной замены значения байта. Предварительно рассчитанный S-box можно увидеть ниже
<b>Nb</b>	число столбцов (32-битных слов), составляющих State. Для AES $Nb = 4$
<b>Nk</b>	число 32-битных слов, составляющих шифроключ. Для AES $Nk = 4, 6$ , или $8$
<b>Nr</b>	число раундов, которое является функцией $Nk$ и $Nb$ . Для AES $Nr = 10, 12, 14$
<b>Rcon[]</b>	массив, который состоит из битов 32-разрядного слова и является постоянным для данного раунда. Предварительно рассчитанный Rcon[] можно увидеть ниже

S-box

```
Sbox = array{
```

```

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
};

```

## Обратный S-box для процедуры InvSubBytes

```

InvSbox = array{
    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
};

```

## Rcon[]

```

Rcon = array{
    array{0x00, 0x00, 0x00, 0x00},
    array{0x01, 0x00, 0x00, 0x00},
    array{0x02, 0x00, 0x00, 0x00},
    array{0x04, 0x00, 0x00, 0x00},
    array{0x08, 0x00, 0x00, 0x00},
    array{0x10, 0x00, 0x00, 0x00},
    array{0x20, 0x00, 0x00, 0x00},
    array{0x40, 0x00, 0x00, 0x00},
    array{0x80, 0x00, 0x00, 0x00},
    array{0x1b, 0x00, 0x00, 0x00},
    array{0x36, 0x00, 0x00, 0x00}
};

```

## Вспомогательные процедуры

<b>AddRoundKey()</b>	трансформация при шифровании и обратном шифровании, при которой Round Key XOR'ится с State. Длина RoundKey равна размеру State (то есть если $Nb = 4$ , то длина RoundKey равна 128 бит или 16 байт)
<b>InvMixColumns()</b>	трансформация при расшифровании, которая является обратной по отношению к MixColumns()
<b>InvShiftRows()</b>	трансформация при расшифровании, которая является обратной по отношению к ShiftRows()
<b>InvSubBytes()</b>	трансформация при расшифровании, которая является обратной по отношению к SubBytes()
<b>MixColumns()</b>	трансформация при шифровании, которая берёт все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы
<b>RotWord()</b>	функция, используемая в процедуре Key Expansion, которая берёт 4-байтовое слово и производит над ним циклическую перестановку
<b>ShiftRows()</b>	трансформации при шифровании, которые обрабатывают State, циклически сдвигая последние три строки State на разные величины
<b>SubBytes()</b>	трансформации при шифровании, которые обрабатывают State, используя нелинейную таблицу замещения байтов (S-box), применяя её независимо к каждому байту State
<b>SubWord()</b>	функция, используемая в процедуре Key Expansion, которая берёт на входе четырёхбайтовое слово и, применяя S-box к каждому из четырёх байтов, выдаёт выходное слово

## Шифрование

AES является стандартом, основанным на алгоритме Rijndael. Для AES длина input (блока входных данных) и State (состояния) постоянна и равна 128 бит, а длина шифроключа **K** составляет 128, 192, или 256 бит. При этом исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин input, State и Cipher Key в 32-битных словах используется нотация  $Nb = 4$  для input и State,  $Nk = 4, 6, 8$  для Cipher Key соответственно для разных длин ключей.

Вначале зашифровывания input копируется в массив State по правилу  $state[r, c] = input[r + 4c]$ , для  $0 \leq r < 4$  и  $0 \leq c < Nb$ . После этого к State применяется процедура AddRoundKey(), и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу  $output[r + 4c] = state[r, c]$ , для  $0 \leq r < 4$  и  $0 \leq c < Nb$ .

Отдельные трансформации SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() — обрабатывают State. Массив  $w[]$  — содержит key schedule.

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)

```

```

        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

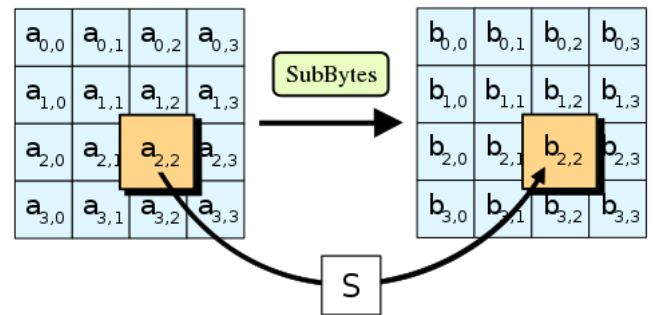
    out = state
end

```

Рис1. Псевдокод для Cipher

## SubBytes()

Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов, используя таблицу замен (S-box). Такая операция обеспечивает нелинейность алгоритма шифрования. Построение S-box состоит из двух шагов. Во-первых, производится взятие обратного числа в поле Галуа  $GF(2^8)$ . Во-вторых, к каждому байту  $b$ , из которых состоит S-box, применяется следующая операция:



В процедуре SubBytes, каждый байт в state заменяется соответствующим элементом в фиксированной 8-битной таблице поиска,  $S$ ;  $b_{ij} = S(a_{ij})$ .

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

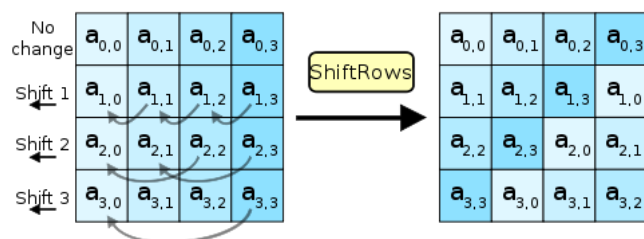
где  $0 \leq i < 8$ , и где  $b_i$  есть  $i$ -ый бит  $b$ , а  $c_i$  —  $i$ -ый бит константы  $c = 63_{16} = 99_{10} = 01100011_2$ . Таким образом обеспечивается защита от атак, основанных на простых алгебраических свойствах.

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

## ShiftRows()

ShiftRows работает со строками State. При этой трансформации строки состояния циклически сдвигаются на  $r$  байт по горизонтали в зависимости от номера строки. Для нулевой строки  $r = 0$ , для первой строки  $r = 1$  Б и т. д. Таким образом, каждая колонка выходного состояния после применения процедуры ShiftRows состоит из байтов из каждой

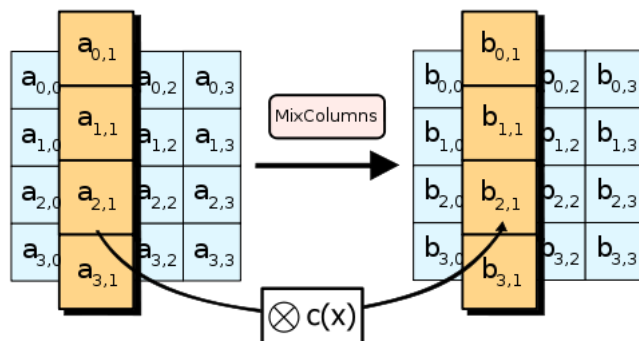
колонки начального состояния. Для алгоритма Rijndael паттерн смещения строк для 128- и 192-битных строк одинаков. Однако для блока размером 256 бит отличается от предыдущих тем, что 2-е, 3-и и 4-е строки смещаются на 1, 3 и 4 байта соответственно. Это замечание не относится к AES, так как он использует алгоритм Rijndael только с 128-битными блоками, независимо от размера ключа.



В процедуре ShiftRows байты в каждой строке state циклически сдвигаются влево. Размер смещения байтов каждой строки зависит от её номера

## MixColumns()

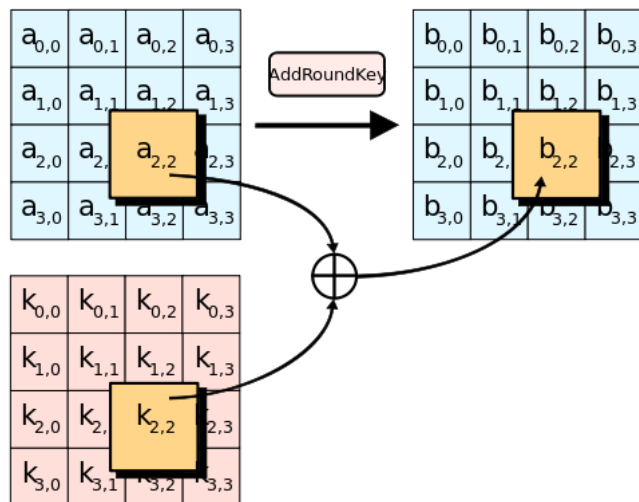
В процедуре MixColumns четыре байта каждой колонки State смешиваются, используя для этого обратимую линейную трансформацию. MixColumns обрабатывает состояния по колонкам, трактуя каждую из них как полином третьей степени. Над этими полиномами производится умножение<sup>[6]</sup> в  $GF(2^8)$  по модулю  $x^4 + 1$  на фиксированный многочлен  $c(x) = 3x^3 + x^2 + x + 2$ . Вместе с ShiftRows MixColumns вносит диффузию в шифр.



В процедуре MixColumns каждая колонка состояния перемножается с фиксированным многочленом  $c(x)$ .

## AddRoundKey()

В процедуре AddRoundKey RoundKey каждого раунда объединяется со State. Для каждого раунда Roundkey получается из CipherKey с помощью процедуры KeyExpansion; каждый RoundKey такого же размера, что и State. Процедура производит побитовый XOR каждого байта State с каждым байтом RoundKey.



В процедуре AddRoundKey каждый байт состояния объединяется с RoundKey, используя операцию XOR ( $\oplus$ ).

# Алгоритм обработки ключа

Алгоритм обработки ключа состоит из двух процедур:

- Алгоритм генерации раундовых ключей (алгоритм расширения ключа)
- Алгоритм выбора раундового ключа (ключа итерации)

## Алгоритм генерации раундовых ключей

Алгоритм AES, используя процедуру KeyExpansion() и подавая в неё Cipher Key, K, получает ключи для всех раундов. Всего получается  $Nb * (Nr + 1)$  слов: изначально для алгоритма требуется набор из Nb слов, и каждому из Nr раундов требуется Nb ключевых наборов данных. Полученный массив ключей для раундов обозначается как  $w[i]$ ,  $0 \leq i < Nb * (Nr + 1)$ . Алгоритм KeyExpansion() показан в псевдокоде ниже.

Функция SubWord() берёт четырёхбайтовое входное слово и применяет S-box к каждому из четырёх байтов. То, что получилось, подаётся на выход. На вход RotWord() подаётся слово  $[a_0, a_1, a_2, a_3]$ , которое она циклически переставляет и возвращает  $[a_1, a_2, a_3, a_0]$ . Массив слов, постоянный для данного раунда,  $Rcon[i]$ , содержит значения  $[x^{i-1}, 00, 00, 00]$ , где  $x = \{02\}$ , а  $x^{i-1}$  является степенью  $x$  в  $GF(2^8)$  ( $i$  начинается с 1).

Из рисунка можно видеть, что первые  $Nk$  слов расширенного ключа заполнены Cipher Key. В каждое последующее слово,  $w[i]$ , кладётся значение, полученное при операции XOR  $w[i - 1]$  и  $w[i - Nk]$ , те XOR'а предыдущего и на  $Nk$  позиций раньше слов. Для слов, позиция которых кратна  $Nk$ , перед XOR'ом к  $w[i-1]$  применяется трансформация, за которой следует XOR с константой раунда  $Rcon[i]$ . Указанная выше трансформация состоит из циклического сдвига байтов в слове (RotWord()), за которой следует процедура SubWord() — то же самое, что и SubBytes(), только входные и выходные данные будут размером в слово.

Важно заметить, что процедура KeyExpansion() для 256-битного Cipher Key немного отличается от тех, которые применяются для 128- и 192- битных шифроключей. Если  $Nk = 8$  и  $i - 4$  кратно  $Nk$ , то SubWord() применяется к  $w[i - 1]$  до XOR'а.

```

KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr+1)], Nk)
begin
    word temp
    i = 0;

    while(i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i + 1
    end while

    i = Nk

    while(i < Nb * (Nr+1))
        temp = w[i - 1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end

```

Псевдокод для Key Expansion

## Расшифрование

```

InvCipher(byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr+1)])

```

```

begin
  byte state[4, Nb]

  state = in

  AddRoundKey(state, w[Nr * Nb, Nb * (Nr+1) - 1])

  for round = Nr - 1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[Nb * round, Nb * (round+1) - 1])
    InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb - 1])

  out = state
end

```

Псевдокод для Inverse Cipher

## Алгоритм выбора раундового ключа

На каждой итерации  $i$  раундовый ключ для операции *AddRoundKey* выбирается из массива  $w[i]$ , начиная с элемента  $w[Nb * i]$  до  $w[Nb * (i + 1)]$ .

## Варианты алгоритма

На базе алгоритма Rijndael, лежащего в основе AES, реализованы альтернативные криптоалгоритмы. Среди наиболее известных — участники конкурса Nessie: Anubis на инволюциях, автором которого является Винсент Рэймен и усиленный вариант шифра — Grand Cru Йохана Борста.

## Криптостойкость

В июне 2003 года Агентство национальной безопасности США постановило, что шифр AES является достаточно надёжным, чтобы использовать его для защиты сведений, составляющих государственную тайну (англ. *classified information*). Вплоть до уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит<sup>[7]</sup>.

## XSL-атака

В отличие от большинства других шифров, AES имеет простое математическое описание. Это беспокоило в том числе и Нильса Фергюсона, который в своей работе отметил, что безопасность шифра основывается на новом непроверенном предположении о сложности решения определённых видов уравнений (англ. «*The security of Rijndael depends on a new and untested hardness assumption: it is computationally infeasible to solve equations of this type*»)<sup>[8][9]</sup>, а также Брюса Шнайера, который написал в совместной с Нильсом книге:



У нас есть одно критическое замечание к AES: мы не совсем доверяем его безопасности. Что беспокоит нас больше всего в AES, так это его простая алгебраическая структура... Ни один другой блочный шифр не имеет столь простого алгебраического представления. Мы понятия не имеем, ведёт это к атаке или нет, но незнание этого является достаточной причиной, чтобы скептически относиться к использованию AES.

Оригинальный текст (англ.) [\[показать\]](#)

— Niels Ferguson, Bruce Schneier Practical Cryptography — 2003 — pp. 56—

57

Николя Куртуа (англ. Nicolas Courtois) и Йозеф Пепшик (англ. Josef Pieprzyk) в 2002 году опубликовали статью, в которой описали теоретическую атаку, названную ими XSL-атакой (англ. *eXtended Sparse Linearization*), которая могла бы позволить вскрыть шифры AES и Serpent<sup>[10][11]</sup>. Тем не менее, результаты работы не всеми были восприняты оптимистично:

Я считаю, что в работе Куртуа-Пепшика есть ошибка. Они переоценили число линейно-независимых уравнений. В результате у них нет достаточного количества линейных уравнений для решения системы, и [указанный] метод не может взломать Rijndael. Он имеет определённые достоинства и заслуживает изучения, но не взламывает Rijndael в его нынешнем виде.

Оригинальный текст (англ.) [\[показать\]](#)

— Дон Копперсмит, комментарий к записи (<http://www.schneier.com/cryptogram-0210.html#8>) в блоге Брюса Шнайера

На странице, посвящённой обсуждению конкурса NESSIE, в конце 2002 года один из авторов шифра, Винсент Рэймен, заявил, что XSL-атака является всего лишь мечтой (англ. *The XSL attack is not an attack. It is a dream*) (данная точка зрения позже была повторена в 2004 году на 4-й конференции AES в Бонне). На это Куртуа ответил, что данная мечта может стать для автора AES кошмаром (англ. *It may also be a very bad dream and turn into a nightmare*)<sup>[12]</sup> (игра слов: *dream* переводится и как *мечта* и как *сновидение*. *Nightmare* переводится как *кошмарный сон*, *ночной кошмар*).

В 2003 году Шон Мёрфи и Мэтт Робшоу (англ. Matt Robshaw) опубликовали работу, в которой (в предположении, что результаты Куртуа и Пепшика верны) обосновали возможность атаки на алгоритм AES, сокращающей количество операций для взлома с  $2^{128}$  до  $2^{100}$ . Однако на 4-й конференции AES Илья Толи (англ. Ilya Toli) и Альберто Дзанони (англ. Alberto Zanoni) показали, что работа Мёрфи и Робшоу неверна<sup>[13]</sup>. Позже, в 2007 году, Чу-Ви Лим (англ. Chu-Wee Lim) и Хунгминг Ху (англ. Khoongming Khoo) также показали, что данная атака не может работать в том виде, как она была описана<sup>[14]</sup>.

## Атака по сторонним каналам

Атаки по сторонним каналам не связаны с математическими особенностями шифра, но используют определённые особенности реализации систем, использующих данные шифры, с целью раскрыть частично или полностью секретные данные, в том числе ключ. Известно несколько подобных атак на системы, использовавшие алгоритм AES.

В апреле 2005 года Дэниел Бернштейн (англ. Daniel J. Bernstein) опубликовал работу с описанием атаки, использующей для взлома информацию о времени выполнения каждой операции шифрования<sup>[15]</sup>. Данная атака потребовала более 200 миллионов выбранных

шифротекстов для нахождения ключа<sup>[16]</sup>.

В октябре 2005 года Даг Арне Освик, Ади Шамир и Эран Трумер представили работу с описанием нескольких атак, использующих время выполнения операций для нахождения ключа. Одна из представленных атак получала ключ после 800 операций шифрования. Атака требовала от криптоаналитика возможности запускать программы на той же системе, где выполнялось шифрование<sup>[17]</sup>.

В декабре 2009 года была опубликована работа, в которой использование дифференциального анализа ошибок (англ. *Differential Fault Analysis*), искусственно создаваемых в матрице состояния на 8-м раунде шифрования, позволило восстановить ключ за  $2^{32}$  операций<sup>[18]</sup>.




## См. также


---

- Тестирование алгоритмов, участвовавших в конкурсе AES
- TKIP
- WPA
- CCMP

## Примечания


---

1. *Лаборатория Чеканова*. Intel Core i5 (Clarkdale): анализ аппаратного ускорения шифрования AES ([http://www.thg.ru/cpu/aes\\_clarkdale/index.html](http://www.thg.ru/cpu/aes_clarkdale/index.html)). THG (19 января 2010). — «наиболее популярный стандарт симметричного шифрования в мире ИТ». Дата обращения: 14 ноября 2010.
2. *Biryukov, Alex and Khovratovich, Dmitry*. Related-key Cryptanalysis of the Full AES-192 and AES-256 (<http://www.impic.org/papers/Aes-192-256.pdf>)  (англ.) // Advances in Cryptology – ASIACRYPT 2009. — Springer Berlin / Heidelberg, 2009. — Vol. 5912. — P. 1—18. — doi:10.1007/978-3-642-10366-7\_1 ([https://dx.doi.org/10.1007%2F978-3-642-10366-7\\_1](https://dx.doi.org/10.1007%2F978-3-642-10366-7_1)).
3. Архивированная копия ([http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes\\_9701.txt](http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes_9701.txt)) (недоступная ссылка). Дата обращения: 7 декабря 2006. Архивировано ([https://web.archive.org/web/20061106004319/http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes\\_9701.txt](https://web.archive.org/web/20061106004319/http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes_9701.txt)) 6 ноября 2006 года.
4. NIST Error Page ([http://www.nist.gov/public\\_affairs/releases/g00-176.htm](http://www.nist.gov/public_affairs/releases/g00-176.htm)) Архивировано ([https://web.archive.org/web/20100928154133/http://www.nist.gov/public\\_affairs/releases/g00-176.htm](https://web.archive.org/web/20100928154133/http://www.nist.gov/public_affairs/releases/g00-176.htm)) 28 сентября 2010 года.
5. Bounce to index.html (<http://csrc.nist.gov/CryptoToolkit/aes/>) Архивировано (<https://web.archive.org/web/20140717094809/http://csrc.nist.gov/CryptoToolkit/aes/>) 17 июля 2014 года.
6. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>  «5.1.3 MixColumns() Transformation .. The columns are considered as polynomials over GF(2^8) and multiplied modulo  $x^4 + 1$  with a fixed polynomial  $a(x)$ , given by  $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ .»
7. National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information (<http://cryptome.org/aes-natsec.htm>) (англ.). Committee on National Security Systems (июнь 2003). Дата обращения: 27 октября 2010. Архивировано (<https://www.webcitation.org/65Yixk1FT?url=http://cryptome.org/aes-natsec.htm>) 19 февраля 2012 года.
8. *James McLaughlin*. The XSL controversy // A survey of block cipher cryptanalysis techniques ([https://www-users.cs.york.ac.uk/~jmclaugh/XSL\\_chapter.pdf](https://www-users.cs.york.ac.uk/~jmclaugh/XSL_chapter.pdf)) . — preprint. — York: University of York, 2009. (недоступная ссылка)

9. *Niels Ferguson, Richard Schroeppe, and Doug Whiting. A simple algebraic representation of Rijndael* (<http://www.macfergus.com/pub/rdalgeq.html>) (англ.) // Selected Areas in Cryptography, Proc. SAC 2001, Lecture Notes in Computer Science #2259. — Springer Verlag, 2001. — P. 103—111. Архивировано (<https://web.archive.org/web/20160116051445/http://www.macfergus.com/pub/rdalgeq.html>) 16 января 2016 года.
10. *Bruce Schneier. Crypto-Gram Newsletter* (<http://www.schneier.com/crypto-gram-0209.html>) (англ.). *Schneier on Security* (15 сентября 2002). Дата обращения: 27 октября 2010. Архивировано (<https://www.webcitation.org/65YiyUTE?url=http://www.schneier.com/crypto-gram-0209.html>) 19 февраля 2012 года.
11. *Nicolas Courtois, Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations* (<https://eprint.iacr.org/2002/044>) (англ.) // Advances in Cryptology — ASIACRYPT 2002 8th International Conference on the Theory Application of Cryptology and Information Security Queenstown, New Zealand, December 1—5, 2002 Proceedings. Lecture Notes in Computer Science (2501). — Springer, 2002. — P. 267—287. — doi:10.1007/3-540-36178-2 (<https://dx.doi.org/10.1007%2F3-540-36178-2>).
12. NESSIE Discussion Forum (<https://web.archive.org/web/20031108050830/http://www.cosic.esat.kuleuven.ac.be/nessie/forum/read.php?f=1&i=82&t=82>)
13. *Ilia Toli, Alberto Zanzi. An Algebraic Interpretation of AES-128* (<http://www.springerlink.com/content/3q790mfnr6bk70pfl/>) (англ.) // Proc. of AES Conference. — 2005. — Vol. 2005. — P. 84—97. — doi:10.1007/11506447\_8 ([https://dx.doi.org/10.1007%2F11506447\\_8](https://dx.doi.org/10.1007%2F11506447_8)). (недоступная ссылка)
14. *Chu-wee Lim, Khoongming Khoo. An Analysis of XSL Applied to BES* (<http://www.springerlink.com/content/6x35t1u383824353/>) (англ.) // Fast Software Encryption. — Heidelberg: Springer Berlin / Heidelberg, 2007. — Vol. 4593. — P. 242—253. — doi:10.1007/978-3-540-74619-5\_16 ([https://dx.doi.org/10.1007%2F978-3-540-74619-5\\_16](https://dx.doi.org/10.1007%2F978-3-540-74619-5_16)). (недоступная ссылка)
15. *Daniel J. Bernstein. Cache-timing attacks on AES* (<http://cr.yp.to/papers.html#cachetiming>) (англ.). — 2004.
16. *Bruce Schneier. AES Timing Attack* ([http://www.schneier.com/blog/archives/2005/05/aes\\_timing\\_atta\\_1.html](http://www.schneier.com/blog/archives/2005/05/aes_timing_atta_1.html)) (англ.). *Schneier on Security* (17 мая 2005). Дата обращения: 27 октября 2010. Архивировано ([https://www.webcitation.org/65YizbRPr?url=http://www.schneier.com/blog/archives/2005/05/aes\\_timing\\_atta\\_1.html](https://www.webcitation.org/65YizbRPr?url=http://www.schneier.com/blog/archives/2005/05/aes_timing_atta_1.html)) 19 февраля 2012 года.
17. *Dag Arne Osvik, Adi Shamir and Eran Tromer. Cache Attacks and Countermeasures: the Case of AES* (<https://eprint.iacr.org/2005/271.pdf>)  // Topics in Cryptology — CT-RSA 2006, The Cryptographers' Track at the RSA Conference. — Springer-Verlag, 2005. — P. 1—20.
18. *Dhiman Saha, Debdeep Mukhopadhyay, Dipanwita Roy Chowdhury. A Diagonal Fault Attack on the Advanced Encryption Standard* (<https://eprint.iacr.org/2009/581>) (англ.) // Cryptology ePrint Archive. — 2009.

## Литература

---

- Federal Information Processing Standards Publication 197 November 26, 2001 Specification for the ADVANCED ENCRYPTION STANDARD (AES) (<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>)  (англ.)
- *Баричев С. Г., Гончаров В. В., Серов Р. Е. 2.4.2. Стандарт AES. Алгоритм Rijndael // Основы современной криптографии — 3-е изд. — М.: Диалог-МИФИ, 2011. — С. 30—35. — 176 с. — ISBN 978-5-9912-0182-7*

## Ссылки

---

- Подробная анимация про реализацию и устройство AES ([https://web.archive.org/web/20140529233954/http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael\\_ingles2004.swf](https://web.archive.org/web/20140529233954/http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf))
- О процессе принятия AES (<http://byrd.narod.ru/aes/aes2.htm>)

- *Jeff Moser*. A Stick Figure Guide to the Advanced Encryption Standard (AES) (<http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>) (англ.) (22 сентября 2009). Дата обращения: 22 ноября 2010. Архивировано (<https://www.webcitation.org/65Yj0G6mK?url=http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>) 19 февраля 2012 года. — AES в картинках (русский перевод (<http://www.pgpru.com/biblioteka/statji/aesvkartinkah>))
- 

Источник — [https://ru.wikipedia.org/w/index.php?title=AES\\_\(стандарт\\_шифрования\)&oldid=119213318](https://ru.wikipedia.org/w/index.php?title=AES_(стандарт_шифрования)&oldid=119213318)

---

Эта страница в последний раз была отредактирована 7 января 2022 в 23:29.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike; в отдельных случаях могут действовать дополнительные условия.

Wikipedia® — зарегистрированный товарный знак некоммерческой организации Wikimedia Foundation, Inc.