Новый стандарт AES – Алгоритм шифрования Rijndael

История проведения конкурса AES

- В 1997 году правительство США объявило на базе института стандартизации NIST (the National Institute of Standards and Technology) открытый конкурс на новый стандарт блочного шифра США. Победитель конкурса получал статус нового стандарта шифрования AES (Advanced Encryption Standard) и рекомендовался к повсеместному использованию на территории США.

Требования, которые предъявлялись к новому стандарту:

- криптоалгоритм должен быть открыто опубликован;
- криптоалгоритм должен быть симметричным блочным шифром, поддерживающим 128-, 192- и 256-битные ключи.
- 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 параметрам: стойкости, стоимости, гибкости, реализуемости в smart-картах.

Претенденты конкурса AES

Алгоритм	Криптографические преимущества	Криптографические недостатки
CAST_256	----------	●Высокие требования к ресурсам для хранения S-блоков. ●Средняя производительность.
CRYPTON	----------	● Очень невысокий запас стойкости криптоанализу по количеству раундов
DEAL	----------	●Низкая производительность; ●Отсутствие повышения стойкости 192-битного ключа по сравнению со 128-битным.
DFC	----------	●Невысокий запас по количеству раундов. ●Использование 64-битного умножения (сложности с реализацией в 8-битовых архитектурах, атака по времени и потребляемой мощности).
E2	----------	●Низкая производительность на некоторых архитектурах. ●Высокие требования к ресурсам для хранения S-блоков.

Претенденты конкурса AES

Алгоритм	Криптографические преимущества	Криптографические недостатки
FROG	----------	•Высокие требования к объему оперативной памяти. •Очень медленная процедура расширения ключа
HPC	----------	•Наличие эквивалентных ключей; •Высокие требования к объему оперативной памяти. •Очень медленная процедура расширения ключа.
LOKI97	----------	•Высокие требования к объему памяти. •Наличие криптоатаки по известным 256 парам «открытый/зашифрованный текст».
MAGENTA	----------	•Низкая производительность на некоторых архитектурах; •Наличие криптоатаки по выбранным 264 парам «открытый/зашифрованный текст».
MARS	•Высокий уровень защищенности. •Высокая эффективность на 32-разрядных платформах, особенно поддерживающих операции умножения и циклического сдвига.	•Сложность алгоритма затрудняет анализ его возможности. •Снижение эффективности на платформах без необходимых операций. •Сложность защиты от временного анализа и анализа мощности.

Претенденты конкурса AES

Алгоритм	Криптографические преимущества	Криптографические недостатки
RC6	●Высокая эффективность на 32-битовых платформах, особенно поддерживающих операции умножения и циклического сдвига.	●Относительно низкий уровень защищенности. ●Снижение эффективности на платформах, не имеющих необходимых операций. ●Сложность защиты от временного анализа и анализа мощности. ●Невозможность генерации раундовых ключей «на лету»
Rijndael	●Высокая эффективность на любых платформах. ●Высокий уровень защищенности. ●Хорошо подходит для реализации в smart-картах из-за низких требований к памяти. ●Быстрая процедура формирования ключа. ●Хорошая поддержка параллелизма на уровне инструкций; поддержка разных длин ключа с шагом 32 бита.	●Уязвим к анализу мощности.

Претенденты конкурса AES

Алгоритм	Криптографические преимущества	Криптографические недостатки
Safer+	----------	●Низкая производительность на некоторых архитектурах. ●Возможность криптоатаки класса «встреча посередине».
Serpent	●Высокий уровень защищенности. ●Хорошо подходит для реализации в smart-картах из-за низких требований к памяти.	●Самый медленный алгоритм среди финалистов. ●Уязвим к анализу мощности.
Twofish	●Высокий уровень защищенности. ●Хорошо подходит для реализации в smart-картах из-за низких требований к памяти. ●Высокая эффективность на любых платформах, в том числе на ожидаемых в будущем 64-разрядных архитектурах фирм Intel и Motorola. ●Поддерживает вычисление раундовых ключей “на лету”. ●Поддерживает распараллеливание на уровне инструкций. ●Допускает произвольную длину ключа до 256 бит.	●Особенности алгоритма затрудняют его анализ. ●Высокая сложность алгоритма. ●Применение операции сложения делает алгоритм уязвимым к анализу мощности и временному анализу.

Формат представления блока входных данных

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}

Формат данных ключа шифрования

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}

Число раундов Nr как функция от длины ключа Nk и длины блока Nb

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

Соответствие между длиной ключа, размером блока данных и числом раундов

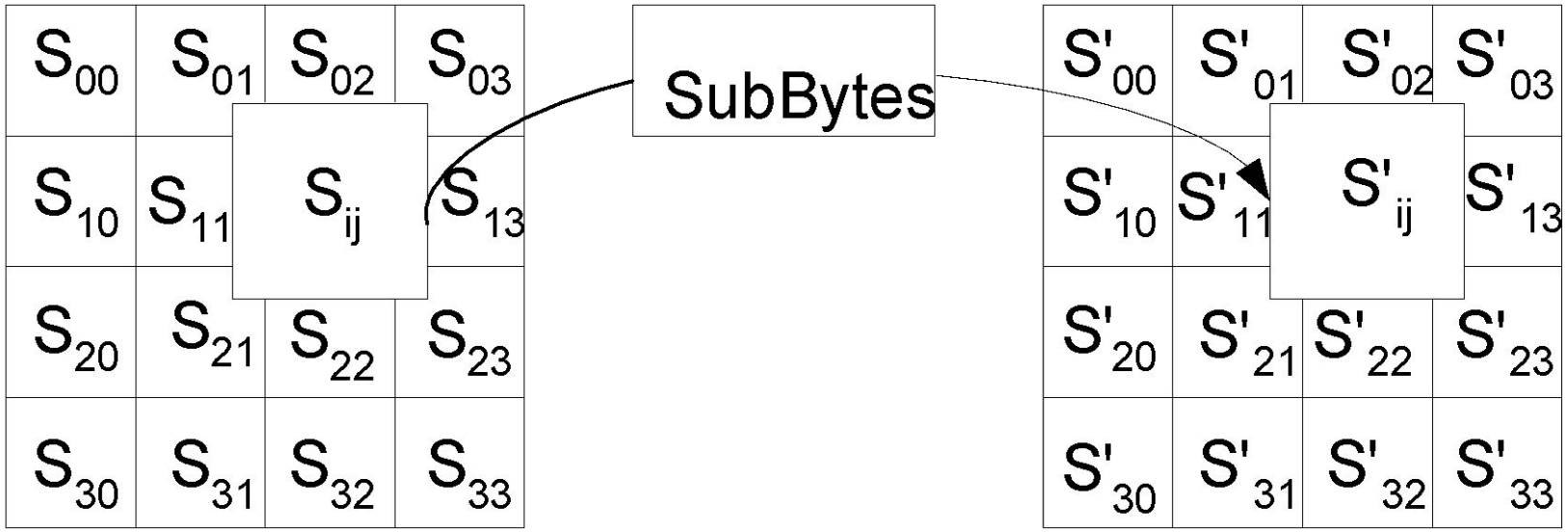
Стандарт	Длина ключа (Nk слов)	Размер блока данных (Nb слов)	Число раундов (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Раундовое преобразование

Раунд состоит из четырех различных преобразований:

- замена байтов SubBytes() – побайтовой подстановки в S-блоках с фиксированной таблицей замен размерностью 8x256;
- сдвига строк ShiftRows() – побайтового сдвига строк массива State на различное количество байт;
- перемешивание столбцов MixColumns() – умножение столбцов состояния, рассматриваемых как многочлены над $GF(2^8)$, на многочлен третьей степени g(x) по модулю x^4+1;
- сложение с раундовым ключом AddRoundKey() – поразрядного XOR с текущим фрагментом развернутого ключа.

Применение преобразования 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 преобразования входного байта:

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

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

Суть преобразования может быть описана уравнением

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

где $c_0=c_1=c_5=c_6=1$, $c_2=c_3=c_4=c_7=0$, b_i и b_i'-соответственно исходное и преобразованное значение i-го бита, i меняется от 0 до 7.

Преобразование SubBytes()

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¤

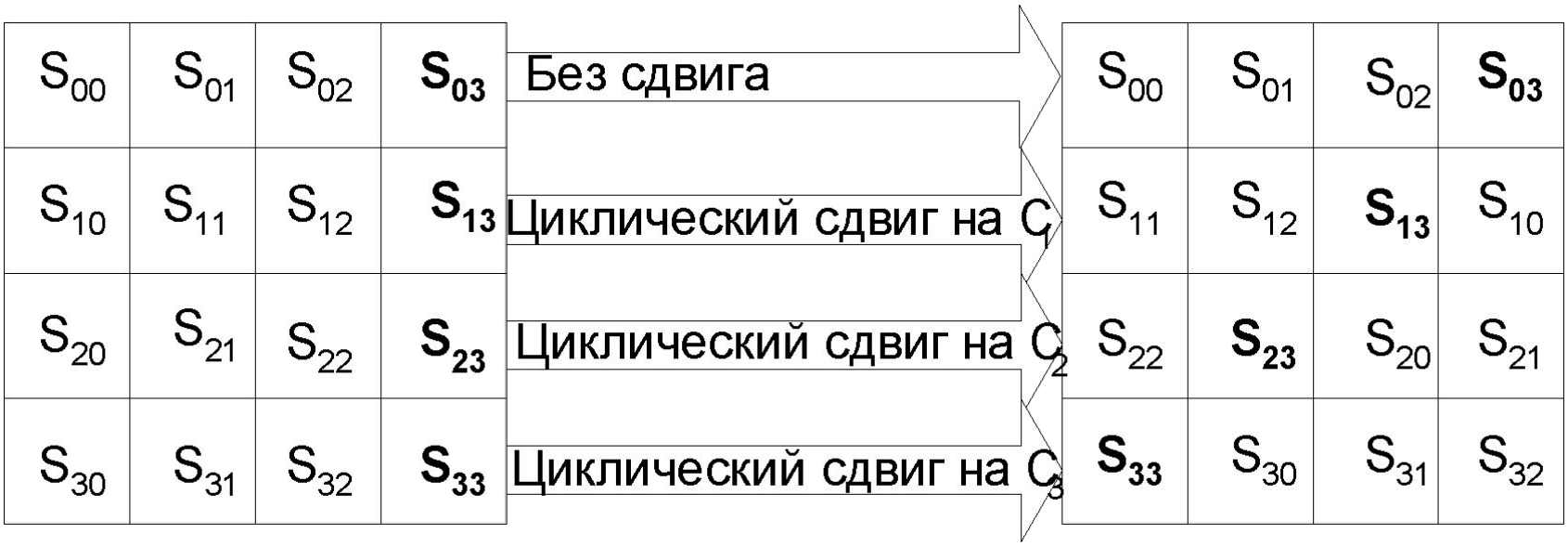
- Логика работы S-блока при преобразовании байта {xy} отражена в таблице. Например, результат {26} преобразования байта {23} находится на пересечении 3-й строки и 4-го столбца.

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

- Логика работы S-блока при преобразовании байта {xy} отражена в таблице. Например, результат {26} преобразования байта {23} находится на пересечении 3-й строки и 4-го столбца.

Преобразование сдвига строк (ShiftRows)

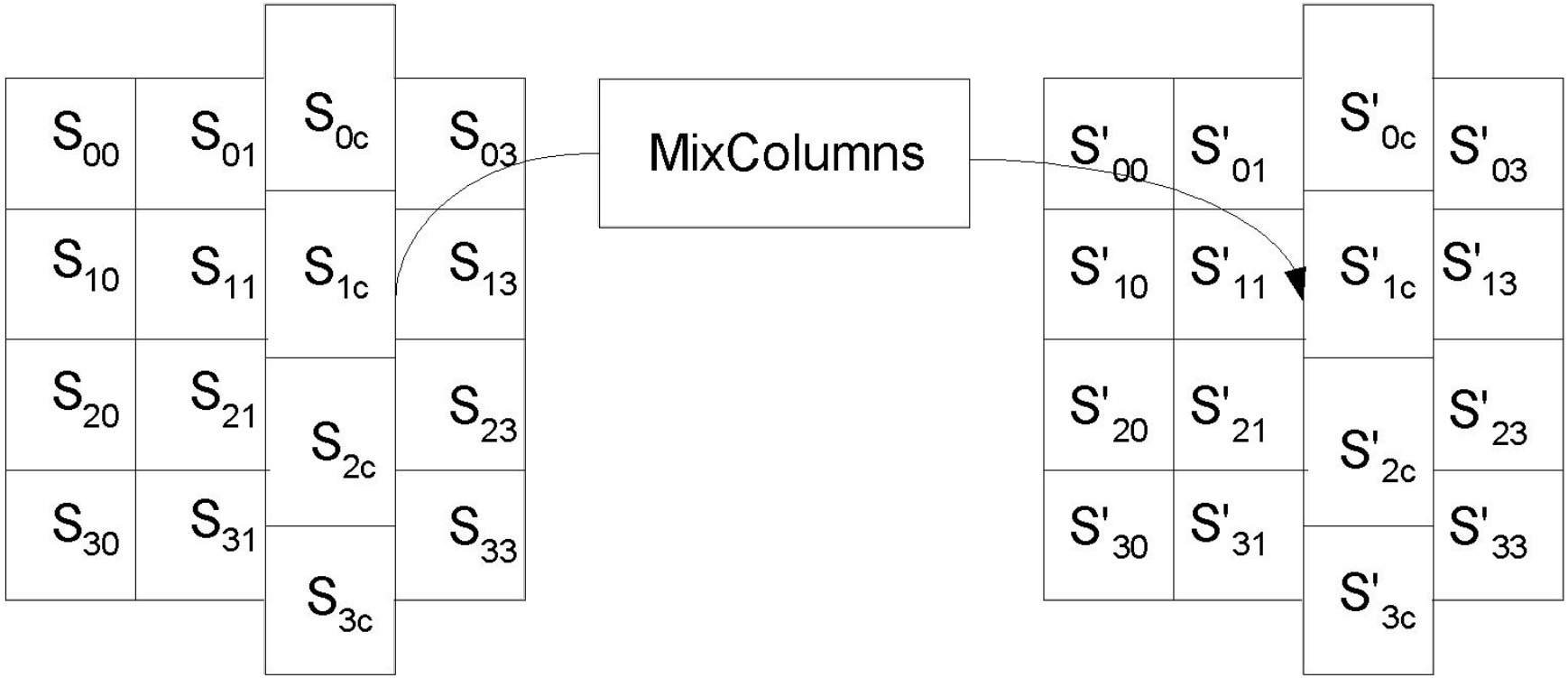


Величина сдвига для разной длины блоков

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

В стандарте AES, где определен единственный размер блока, равный 128 битам, $C_1 = 1$, $C_2 = 2$, $C_3 = 3$.

Преобразование перемешивания столбцов (MixColumns)



Преобразование перемешивания столбцов (MixColumns)

- Преобразование перемешивания столбцов (MixColumns) это такое преобразование, при котором 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:

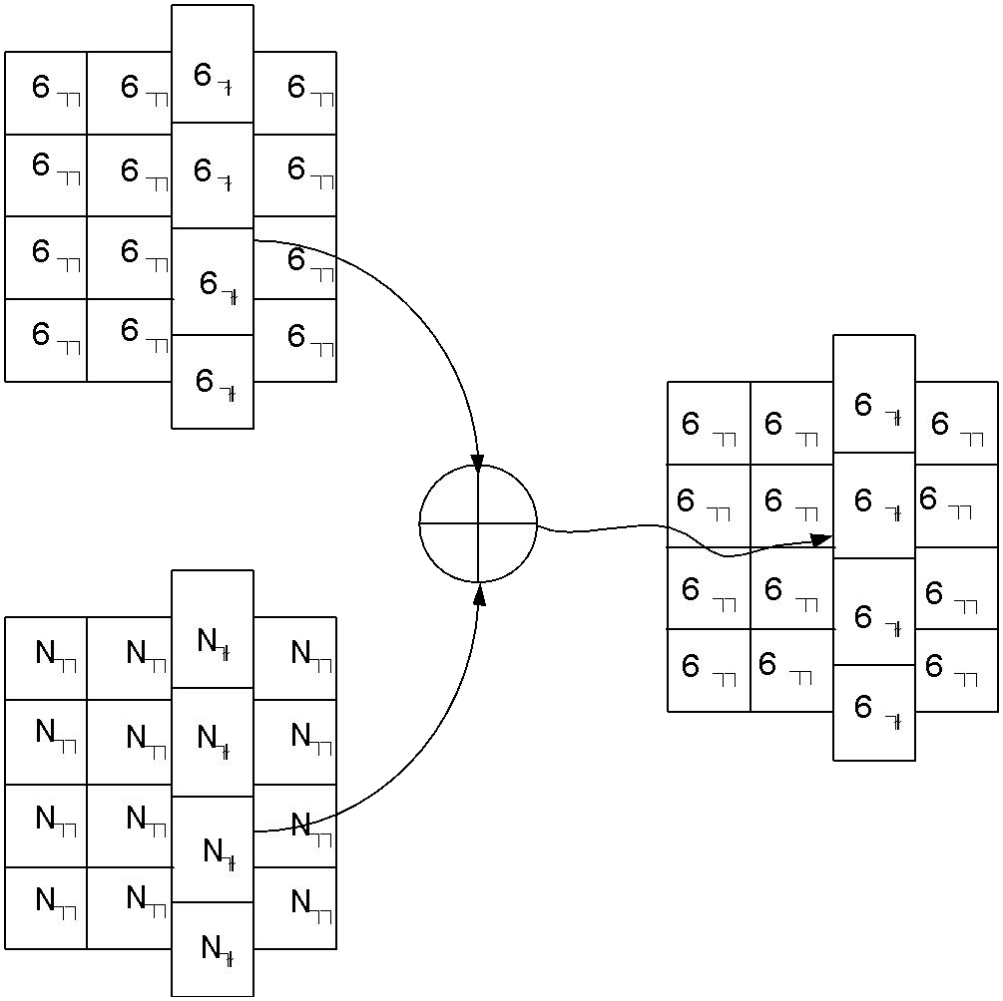
$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, 0 \le c \le 3$$

где c – номер столбца массива State.

Преобразование перемешивания столбцов (MixColumns)

- В результате такого умножения байты столбца s_{0c}, s_{1c}, s_{2c}, s_{3c} заменяются соответственно на байты:
- $s'_{0c}=(\{02\}*s_{0c})\oplus(\{03\}*s_{1c})\oplus s_{2c}\oplus s_{3c}$,
- $s'_{1c}=s_{0c}\oplus(\{02\}*s_{1c})\oplus(\{03\}*s_{2c})\oplus s_{3c}$,
- $s'_{2c}=s_{0c}\oplus s_{1c}\oplus(\{02\}*s_{2c})\oplus(\{03\}*s_{3c})$,
- $s'_{3c}=(\{03\}*s_{0c})\oplus s_{1c}\oplus s_{2c}\oplus(\{02\}*s_{3}c)$.

Добавление раундового ключа (AddRoundKey)



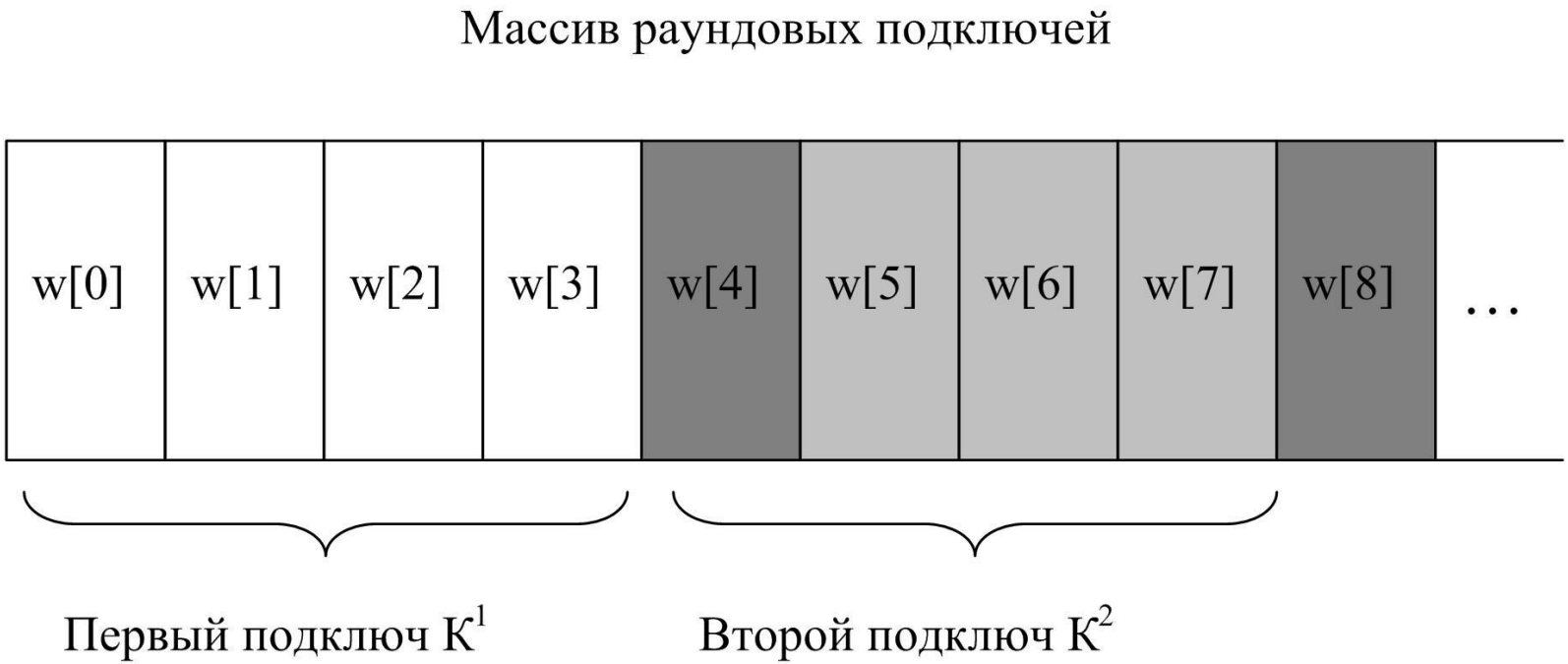
Алгоритм выработки ключей

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 преобразуется 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.
- Расширенный ключ (Key Expansion) в Rijndael представляет собой линейный массив w[i] из Nb(Nr+1) 4-байтовых слов.
- Первые Nk слов содержат ключ шифрования. Все остальные слова определяются рекурсивно из слов с меньшими индексами. Алгоритм выработки подключей зависит от величины Nk.
- Первые Nk слов заполняются ключом шифрования. Каждое последующее слово w[i] получается посредством сложения по модулю два предыдущего слова w[i-1] и слова на Nk позиций ранее, то есть w[i-Nk]:

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

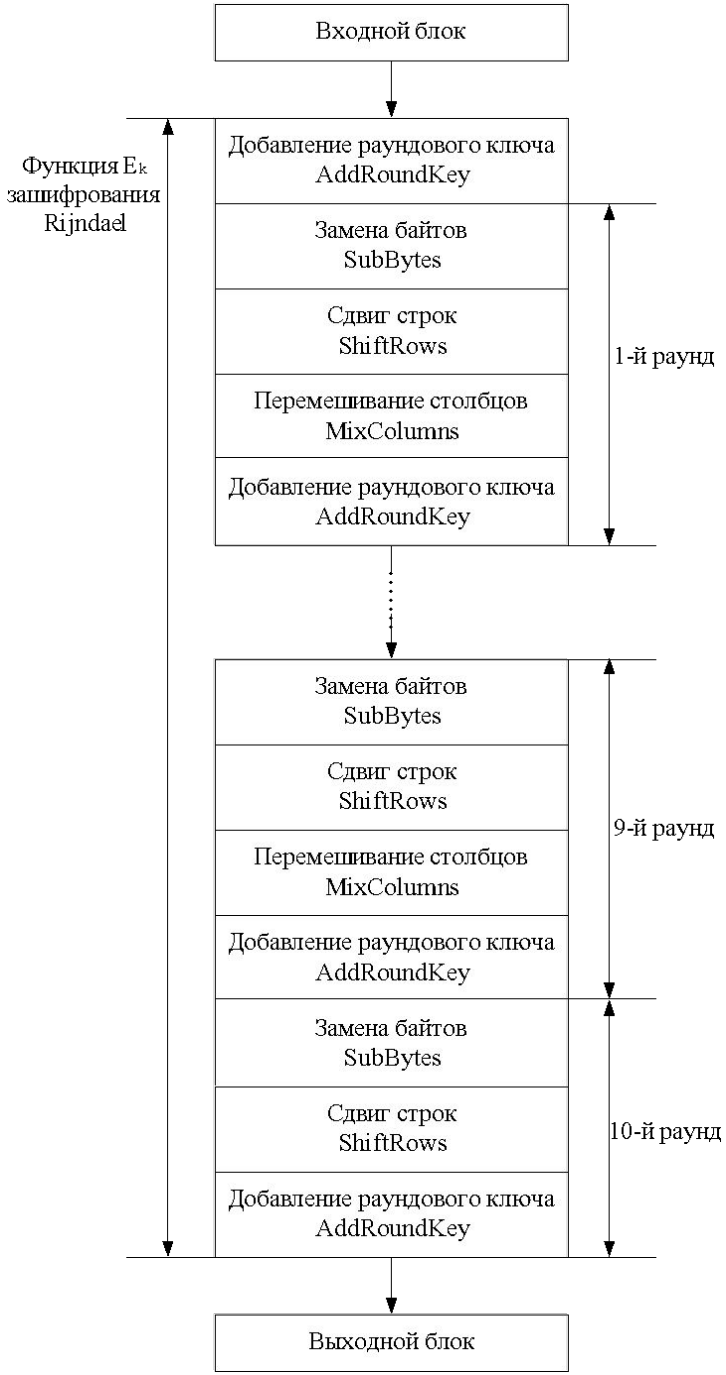
Алгоритм выработки ключей



- Для слов, позиция которых кратна Nk перед операцией сложения по модулю два применяется преобразование к w[i-1], а затем еще прибавляется раундовая константа Rconst. 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 S-блока функции SubBytes(). Значение Rconst[j] равно2^{j-1}. Значение w[i] в этом случае равно:

  $$w[i] = SubWord(RotWord(w[i-1])) \oplus Rconst[i/Nk] \oplus w[i-Nk].$$

- Раундовый ключ i получается из слов массива раундового ключа от W[Nbi] и до W[Nb(i+1)].



Функция зашифрования

- Шифр Rijndael состоит:
  из начального добавления раундового ключа;
  N_{r-1} раундов;
  заключительного раунда, в котором отсутствует операция MixColumns().

Функция обратного дешифрования

- Если вместо SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() в обратной последовательности выполнить инверсные им преобразования, можно построить функцию обратного дешифрования. При этом порядок использования раундовых ключей является обратным по отношению к тому, который используется при зашифровании.
- Функция AddRoundKey() обратна сама себе, учитывая свойства используемой в ней операции XOR.
- Для преобразования байта {xy} используется инверсный S-блок InvSubBytes
- В преобразовании InvShiftRows последние 3 строки состояния сдвигаются вправо на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 – на C2 байт, и строка 3 – на C3 байт. Значение сдвигов C1, C2 и C3 зависят от длины блока Nb.

Функция обратного дешифрования

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

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

- Это может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 9d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}$$

- В результате на выходе получаются следующие байты:

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

Последовательность преобразований в двухраундовом варианте Rijndael

Функция зашифрования двухраундового варианта Rijndael	Функция обратного дешифрования двухраундового варианта Rijndael	Эквивалентная функция прямого дешифрования 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

Основные особенности Rijndael

Основные особенности Rijndael:

- новая архитектура «Квадрат», обеспечивающая быстрое рассеивание и перемешивание информации, при этом за один раунд преобразованию подвергается весь входной блок;
- байт ориентированная структура, удобная для реализации на 8-разрядных МК;
- все раундовые преобразования представляют собой операции в конечных полях, допускающие эффективную аппаратную и программную реализацию на различных платформах.