Лекция №8
Симметричное шифрование
AES 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.

Претендентов конкурса AES было множество, но многие имели свои криптографические недостатки

Некоторые 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.

Некоторые 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.

Некоторые 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 **Winner**	• 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.

Основные массивы данных

Входными данными для операций шифрования есть массив из 16 байт. Перед началом шифрования байты этого массива размещаются последовательно в столбцы матрицы InputBlock (**сверху вниз**). Внутри алгоритма операции выполняются над матрицей байт, называемой матрицей состояний State или просто **состоянием**. Конечное значение матрицы состояния OutputBlock является выходом алгоритма и преобразуется в последовательность байтов шифротекста. Аналогично в столбцы матрицы InputKey попадают и 16 байтов ключа шифра.

InputBlock → State = OutputBlock

in_0	in_4	in_0	in_0
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_0	out_0
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Байты ключа
Input*Key*

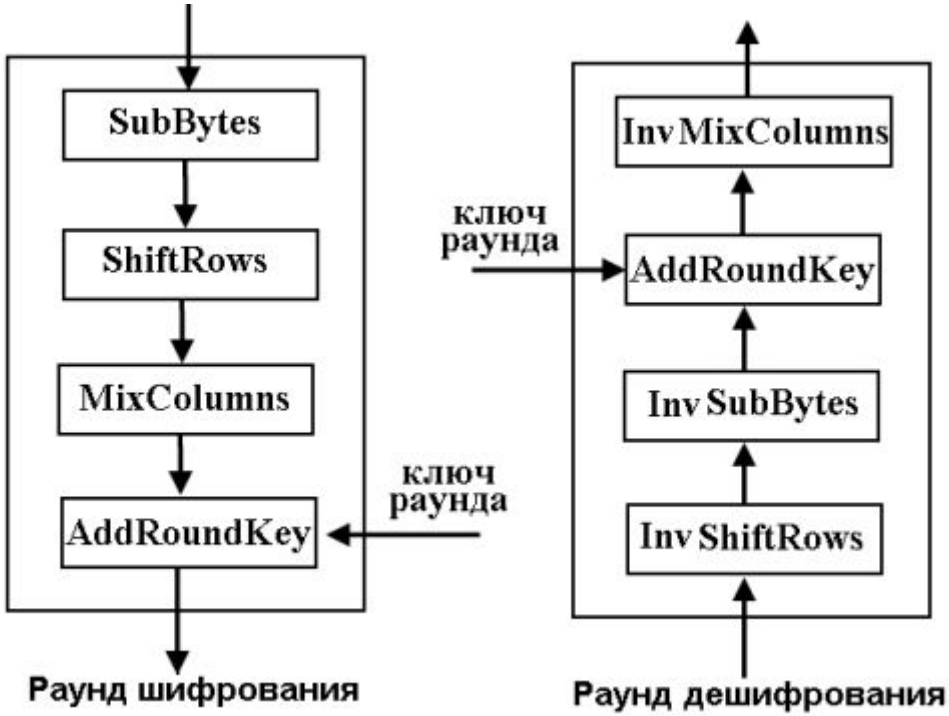
k_0	k_4	k_0	k_0
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

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.

Раундовое преобразование



Операции в поле GF(2^8)

Операции в поле $GF(2^8)$. Для описания алгоритма используется конечное поле Галуа $GF(2^8)$, построенное как расширение поля $GF(2)=\{0,1\}$ по модулю неприводимого многочлена $m(x)=x^8+x^4+x^3+x+1$. Элементами поля $GF(2^8)$ являются многочлены вида

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0,$$

степень которых меньше 8, а коэффициенты $b_7,b_6,...,b_0\in\{0,1\}$. Операции в поле выполняются по модулю $m(x)$. Всего в поле $GF(2^8)$ насчитывается $2^8=256$ многочленов.

Представление двоичного числа $b_7b_6b_5b_4b_3b_2b_1b_0$ в виде многочлена с коэффициентами $b_7,b_6,...,b_0$ позволяет интерпретировать байт как битовый многочлен в конечном поле $GF(2^8)$:

$$b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0. \quad (1)$$

Например, байт 63 задает последовательность битов 01100011 и определяет конкретный элемент поля

$$01100011\leftrightarrow x^6+x^5+x+1.$$

Соответствие между длиной ключа, размером блока данных и числом раундов (циклов)

Стандарт	Длина ключа (N_k слов)	Размер блока данных (N_b слов)	Число раундов (N_r)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Применение преобразования SubBytes()

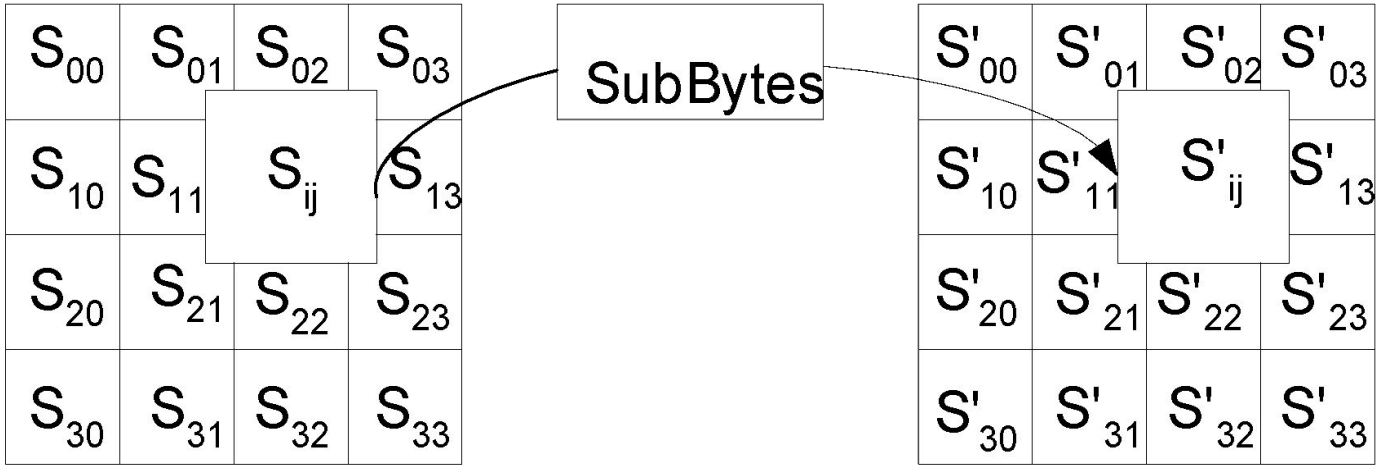
Преобразование представляет собой **нелинейную** замену байт, выполняемую независимо с каждым байтом состояния. Таблицы замены (или S-блоки) являются инвертируемыми и построены из композиции двух преобразований:

1. Получение обратного элемента относительно умножения в поле $GF(2^8)$, принято считать, что комбинация '00' переходит сам в себя.
2. Применение афинного преобразования (над GF(2)), определенного как:

y_0	=	1	1	1	1	1	0	0	0	*	x_0	+ 0
y_1		0	1	1	1	1	1	0	0		x_1	1
y_2		0	0	1	1	1	1	1	0		x_2	1
y_3		0	0	0	1	1	1	1	1		x_3	0
y_4		1	0	0	0	1	1	1	1		x_4	0
y_5		1	1	0	0	0	1	1	1		x_5	0
y_6		1	1	1	0	0	0	1	1		x_6	1
y_7		1	1	1	1	0	0	0	1		x_7	1

через x обозначены входные биты, а через y – выходные

Преобразования SubBytes()

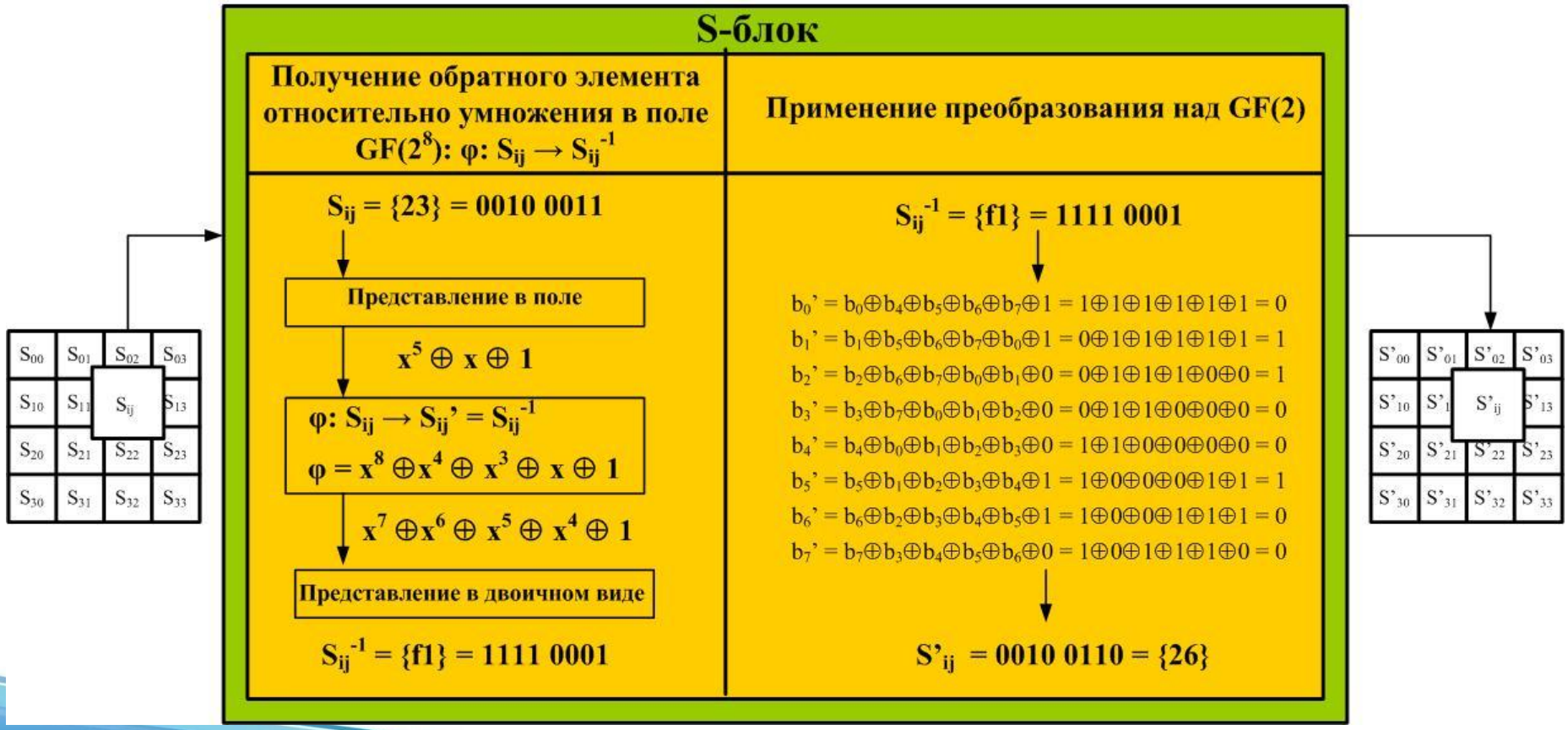


Суть преобразования может быть описана уравнением

$$y_i = x_i \oplus x_{(i+4)mod8} \oplus x_{(i+5)mod8} \oplus x_{(i+6)mod8} \oplus x_{(i+7)mod8} \oplus c_i,$$

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.

Преобразования SubBytes() - пример



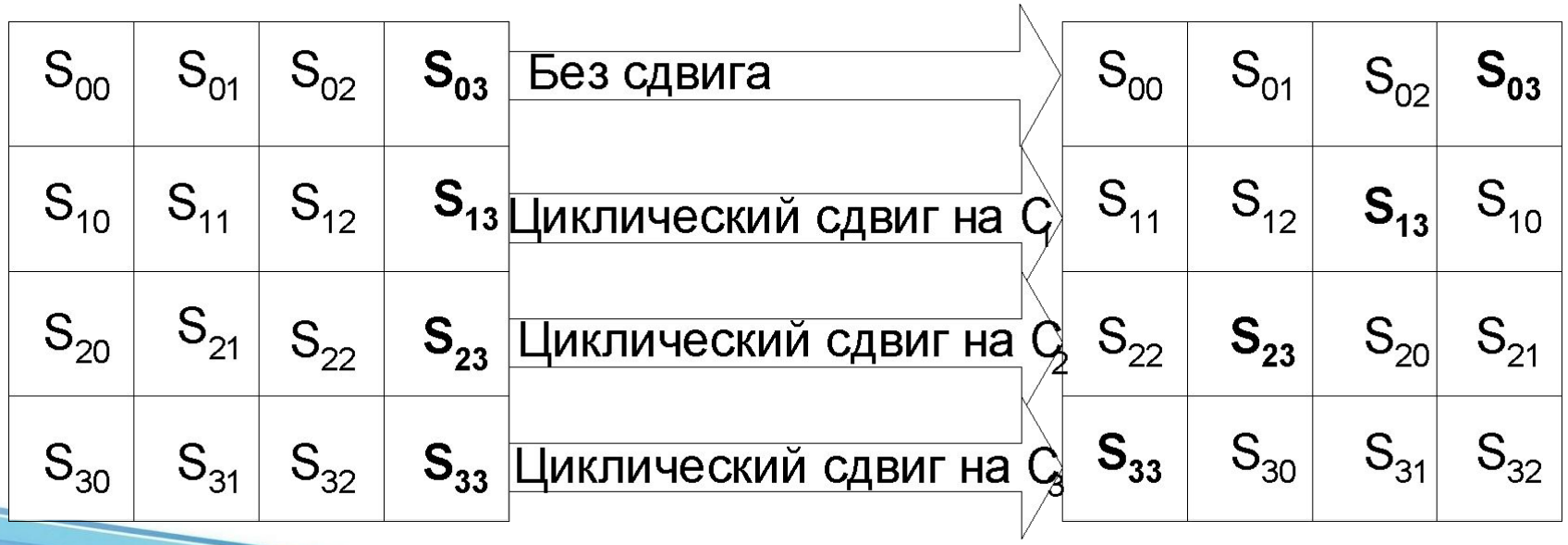
S-блок

	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

Например, если $s_{1,1}$={8A}, то результат замены этого байта следует искать на пересечении строки с индексом 8 и столбца с индексом A, т.е. SubBytes(8A)={7E}.

Преобразование сдвига строк (ShiftRows)

Операция применяется к строкам матрицы State – ее первая строка неподвижна, а элементы нижних трех строк циклически сдвигаются вправо на 1, 2 и 3 байта соответственно.

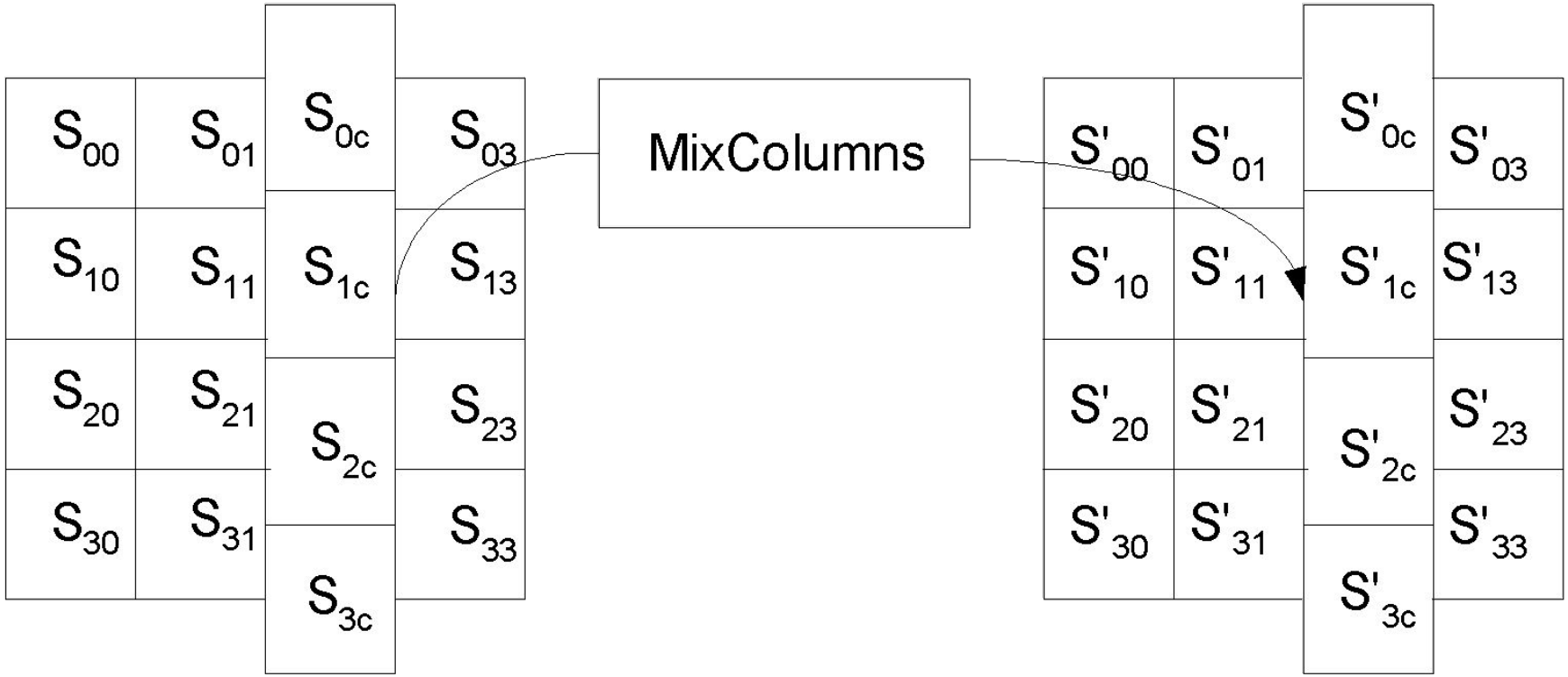


Величина сдвига для разной длины блоков

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 c(x), выглядящий следующим образом:

$$c(x) = c_3x^3 + c_2x^2 + cx + c_0 = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Это может быть представлено в матричном виде следующим образом:

$$\begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{pmatrix}.$$

где c – номер столбца массива State.

Преобразование перемешивания столбцов (MixColumns)

В результате такого умножения байты столбца s_0, s_1, s_2, s_3 заменяются соответственно на байты:

$s'_0=(\{02\}*s_0)\oplus(\{03\}*s_1)\oplus s_2\oplus s_3$,

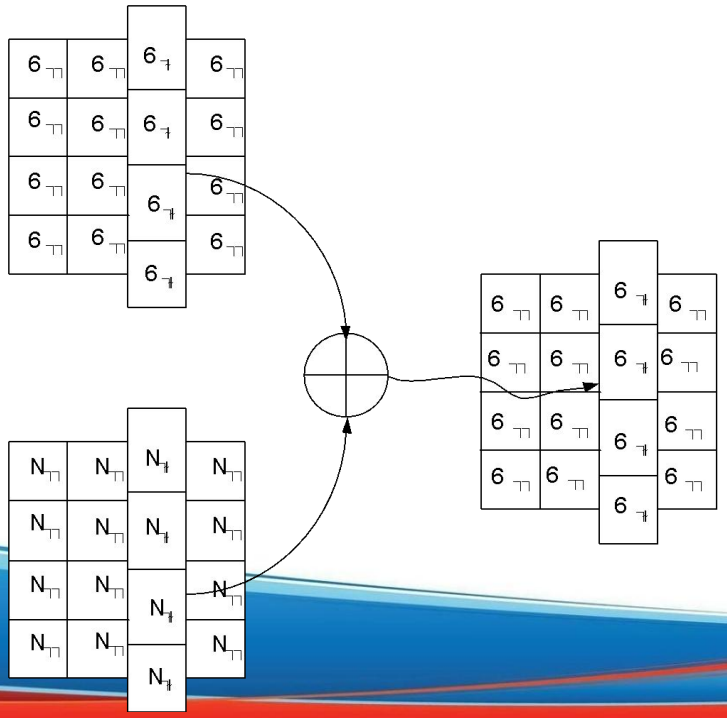
$s'_1=s_0\oplus(\{02\}*s_1)\oplus(\{03\}*s_2)\oplus s_3$,

$s'_2=s_0\oplus s_1\oplus(\{02\}*s_2)\oplus(\{03\}*s_3)$,

$s'_3=(\{03\}*s_0)\oplus s_1\oplus s_2\oplus(\{02\}*s_3)$.

Добавление раундового ключа (AddRoundKey)

AddRoundKey(State, RoundKey) побитово складывает элементы переменной RoundKey и элементы переменной State по принципу: i-й столбец данных (i = 0, 1, 2, 3) складывается с определенным 4-байтовым фрагментом расширенного ключа W=[4r+1], где r – номер поточного раунда алгоритма. При шифровании первое сложение ключа раунда происходит до первого выполнения операции SubBytes.



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 N_b слов, второй – следующие N_b слов и т. д.

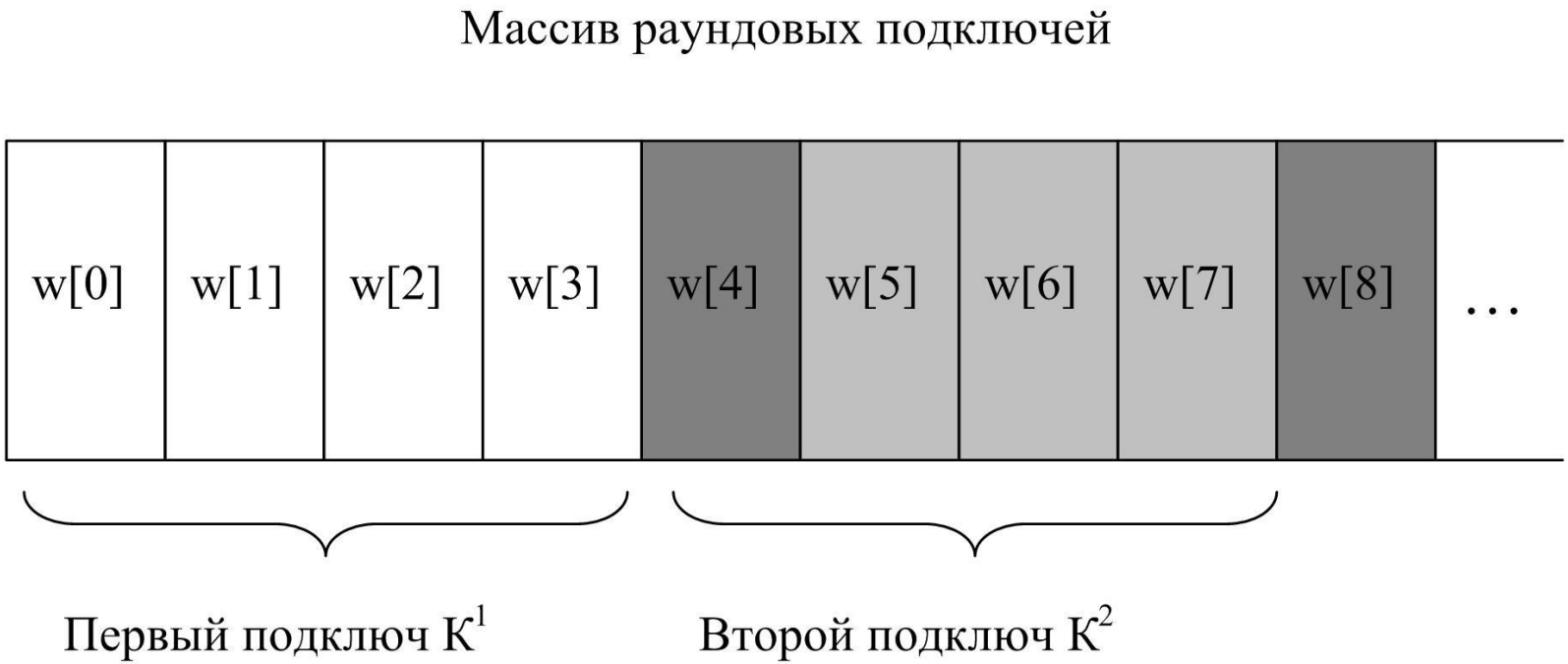
Расширенный ключ (Key Expansion) в Rijndael представляет собой линейный массив w[i] из Nb(Nr+1) 4-байтовых слов.

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 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 C_1 байт, строка 2 – на C_2 байт, и строка 3 – на C_3 байт. Значение сдвигов C_1, C_2 и C_3 зависят от длины блока N_b.

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:

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

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.

МИРЭА
Российский технологический университет

Спасибо за внимание