авторов). Стандарт начал действовать с 2002 г.

AES – симметричный итеративный блоковый алгоритм;
AES – не шифр Фейстеля, базируется на принципах новой сети подстановок-перестановок. Имеет новую архитектуру SQUARE (КВАДРАТ), для которой характерно: 1) представление шифруемого блока в виде двумерного байтового массива; 2) шифрование за один раунд всего блока данных (***байт-ориентированная структура***); 3) выполнение криптографических преобразований, как над отдельными байтами массива, так и над его строками и столбцами. Это обеспечивает диффузию данных одновременно в двух направлениях - по строкам и по столбцам. Архитектура SQUARE присуща, кроме шифра AES(RIJNDAEL), шифрам SQUARE (его название и дало имя всей архитектуре), CRYPTON (один из кандидатов на AES). Второе место в конкурсе AES занял другой SP-шифр, SERPENT. По-видимому, SP-сети и, в частности, архитектура SQUARE, в ближайшем будущем станут безраздельно доминировать.

Общие характеристики AES

- AES зашифровывает и расшифровывает 128-битовые блоки данных.
- AES позволяет использовать три различных ключа длиной 128, 192 или 256 бит (в зависимости от длины ключа версии шифра обозначают AES-128, AES-192 или AES-256).
- От размера ключа зависит число раундов шифрования:
  - длина 128 бит – 10 раундов;
  - длина 192 бита – 12 раундов;
  - длина 256 бит – 14 раундов.

Напоминание: 1 байт$=$8 битов, 128 битов=16$\times$8 битов=16 байт.

Основным элементом, которым оперирует алгоритм AES, является байт – последовательность 8 бит, обрабатываемых как единое целое. Для формирования байтов 128 битов блока открытого текста, выходного блока шифротекста и ключа шифра делятся на группы из 8-ми рядом стоящих бит так, чтобы в целом получился массив байт. Ниже представлена принятая нумерация бит в пределах каждого байта:

№ бита на входе	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	…
№ байта	0								1								2								…
№ бита в байте	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…

Задавать значение байта удобно в шестнадцатеричной системе исчисления. Для этого байт делится на две группы из 4-х бит: группа старших бит в байте представляется первым шестнадцатеричным символом, а группа младших бит – вторым. Например, для байта 10101100 получим

$$10101100 = 1010\ \ 1100 = AC.$$

Обозначим

$in_0, in_1, \ldots, in_{15}$ – 16 байт блока открытого текста;

$k_0, k_1, \ldots, k_{15}$ – 16 байт ключа шифра;

– 16 байт блока шифротекста.

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}

$\rightarrow$

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

$=$

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

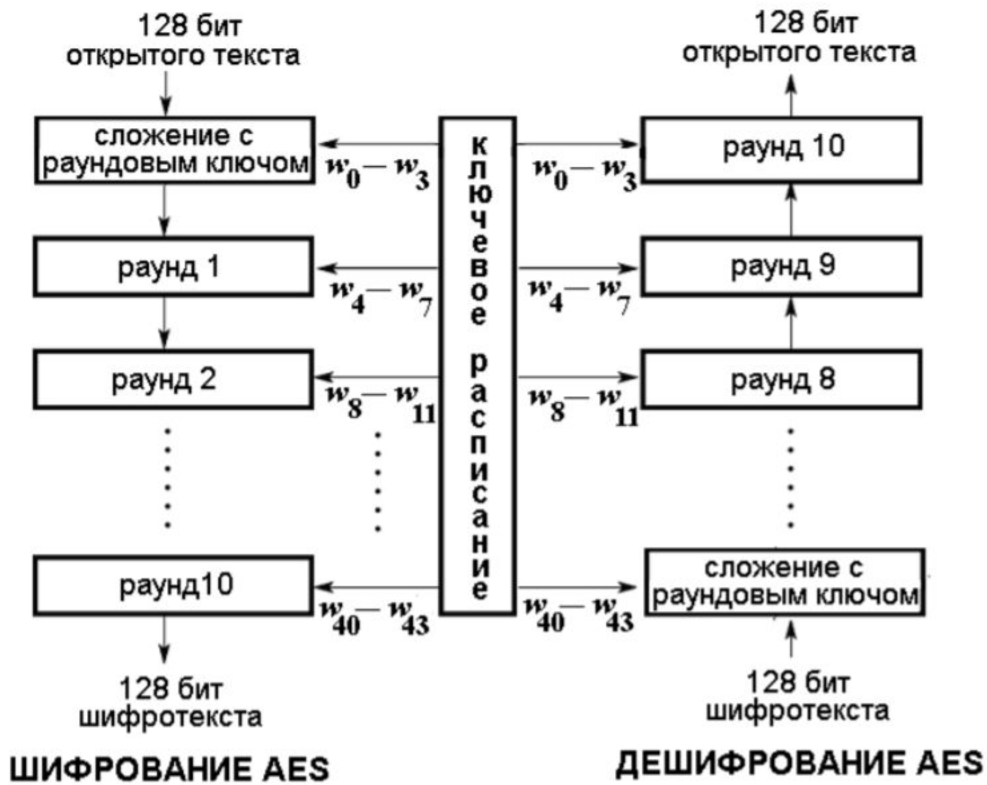
Input*Key*

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}

$$= (15\ \ 0E\ \ 0F\ 00\ \ 05\ \ 11\ 0B\ \ 15\ \ 16\ \ 1E\ \ 09\ \ 00\ \ 05\ \ 00\ \ 1B\ \ 0B\ )_{16}$$

$$\text{Input}Block = \begin{pmatrix} 15 & 05 & 16 & 05 \\ 0E & 11 & 1E & 00 \\ 0F & 0B & 09 & 1B \\ 00 & 15 & 00 & 0B \end{pmatrix}.$$

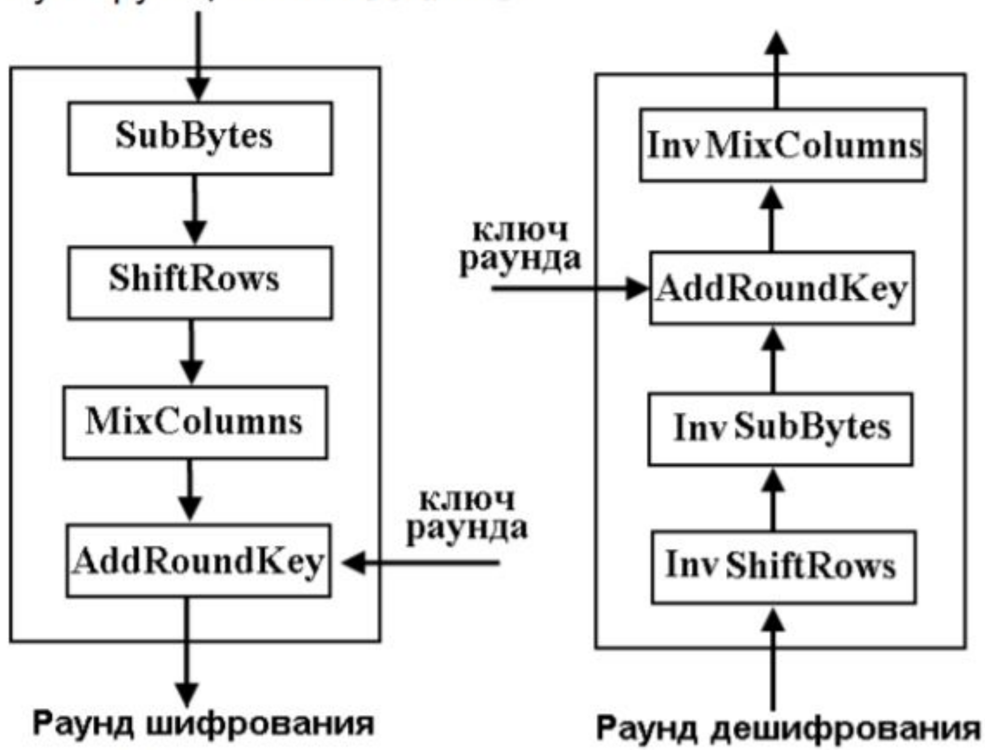
Схема преобразования данных показана на рисунке.



Раунд состоит из 4 различных преобразований:

- SubBytes – побайтовая подстановка в S-боксе с фиксированной таблицей замен;
- ShiftRows – побайтовый сдвиг строк матрицы *State* на различное количество байт;
- MixColumns – перемешивание байт в столбцах;
- AddRoundKey – сложение с раундовым ключом (операция *XOR*).

Последний раунд несколько отличается от предыдущих тем, что не задействует функцию MixColumns.



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

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

Рассмотрим основные математические операции в поле $GF(2^8)$.

1. ***Сложение байт*** можно выполнить любым из трех способов:
   - представить байты битовыми многочленами и сложить их по обычному правилу суммирования многочленов с последующим приведением коэффициентов суммы по модулю 2 (операция XOR над коэффициентами);
   - суммировать по модулю 2 соответствующие биты в байтах;
   - сложить байты в шестнадцатеричной системе исчисления.

Например, следующие три записи эквивалентны:

- представление в виде многочленов

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2;$$

- битовое представление

$$\{01010111\} \oplus \{10000011\} = \{11010100\};$$

- шестнадцатеричное представление

$$\{57\} \oplus \{83\} = \{D4\}.$$

2. ***Умножение байт*** выполняется с помощью представления их многочленами и перемножения по обычным алгебраическим правилам. Полученное произведение необходимо привести по модулю многочлена $m(x) = x^8 + x^4 + x^3 + x + 1$ (результат приведения равен остатку от деления произведения на $m(x)$).

Перемножение многочленов в поле можно упростить, введя операцию умножения битового многочлена (1) на x:

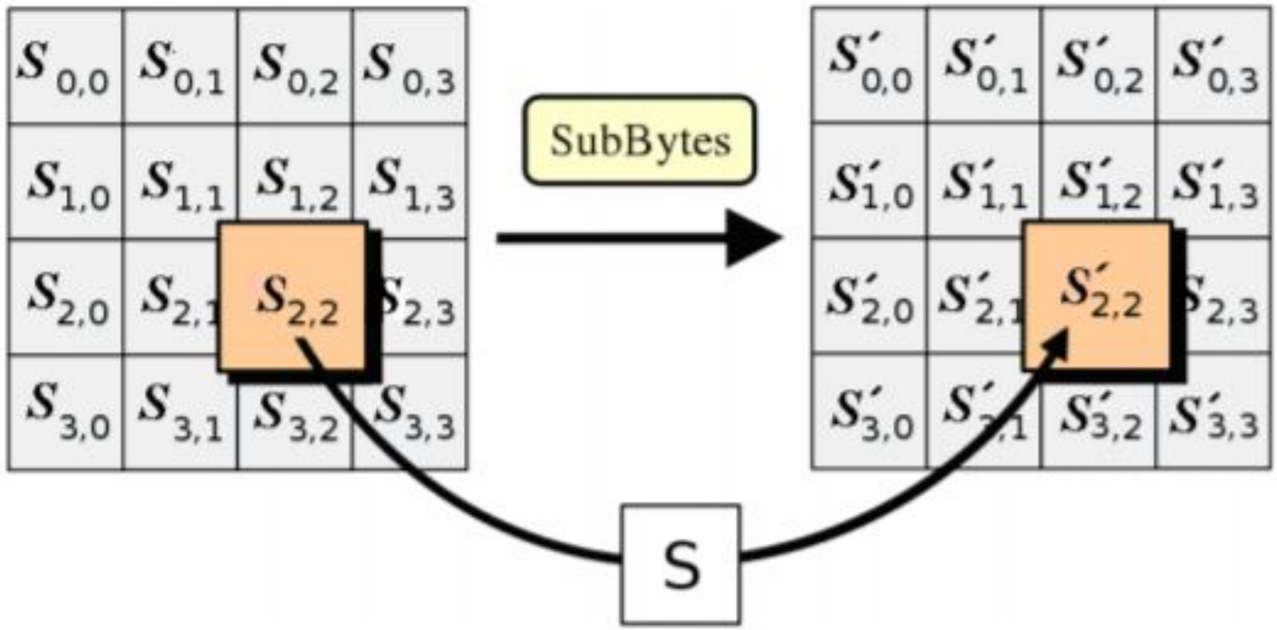
$$x(b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0) =$$
$$= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x\,.$$

3. Для любого ненулевого битового многочлена $b(x)$ в поле $GF(2^8)$ существует многочлен $b^{-1}(x)$, ***обратный*** к нему по умножению, т.е. $b(x)b^{-1}(x) \equiv 1 \bmod m(x)$. Для нахождения обратного элемента используют расширенный алгоритм Эвклида, с помощью которого находят такие многочлены $a(x)$ и $c(x)$, что $a(x)b(x) + c(x)m(x) = 1$. Следовательно, $b^{-1}(x) \equiv a(x) \bmod m(x)$.

1. **Операция** SubBytes.

Операция выполняет нелинейную замену байтов, выполняемую независимо с каждым байтом матрицы *State*. Замена обратима и построена путем комбинации двух преобразований над входным байтом:

- нахождение обратного (инвертированного) элемента относительно умножения в поле $GF(2^8)$ (считается, что нулевой байт $\{00\}$ переходит сам в себя);
- выполнение некого аффинного преобразования: умножение инвертированного байта на многочлен $a(x) = x^4 + x^3 + x^2 + x + 1$ и суммирование с многочленом $b(x) = x^6 + x^5 + x + 1$ в поле $F_2[x]/x^8 + 1$.



S0,0 S0,1 S0,2 S0,3
S1,0 S1,1 S1,2 S1,3
S2,0 S2,1 S2,2 S2,3
S3,0 S3,1 S3,2 S3,3
SubBytes
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
S

	Таблица преобразования SubBytes															
	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	FO	AD	D4	A2	AF	9C	A4	72	CO
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	AO	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	DO	EF	AA	FB	43	4D	33	85	45	F9	02	F7	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	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	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	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	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	OD	BF	E6	42	68	41	99	2D	OF	BO	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. $\mathrm{SubBytes}(8A)=\{7E\}$.

Лабораторная SubBytes.
Задание 1
Найти частное и остаток от деления a на b , где a и b – многочлены с коэффициентами из Z2.

Пример.
a = [110101000001110111001001111001 01]
b=[1111100110101100]

q = [10111010110111011]
r=[10101100110001]
test
q*b=[11010100000111011110001011010100]
q*b+r=[11010100000111011100100111100101]
a=[11010100000111011100100111100101]

Лабораторная SubBytes.
Задание 2
Найти НОД(а,в) и коэффициенты Безу многочленов a и b из F2

a=[11110100000001001111111000010010111
10011111100011110110110110001]
b=[1011110010001010]

GCD = [11]
x=[1100010101]
y=[10111110111110101101011100010111110
00001101000010101001111]
test
ax+by=[11]
GCD=[11]

Лабораторная SubBytes.
Задание 3
Используя расширенный алгоритм Евклида, найти обратный элемент к многочлену a по модулю p, если такой элемент существует.

a=[0010110011110100]
p=[0111101011100111]

GCD = [1]
inv=[010101100010111]
test
a*inv=[1001000111000110101100001100]
a*inv mod p=[1]
GCD=[1]

a=[1111010010010000]
p=[0100001011110010]

GCD = [10]

Лабораторная SubBytes.
Задание 4
Вычислить SubByte(byte).
Byte=[11101101]

Step 1/ Step1= Inverse(byte) mod m(x) =?
(m(x)=[1 0 0 0 1 1 0 1 1])
step1=[01010000]

Step 2/ Step2=Step1 * a__ =?
(a__=[0 0 0 1 1 1 1 1])

Step 2_/ step2_=Step2 mod m2 =?
(m2(x)=[1 0 0 0 0 0 0 0 1])
step2=[11000110000]

Step 3/ step3=Step2_ +b__ =?
(b__=[0 1 1 0 1 0 0 1])
step2_=[00110110]

Step 3_/ step3_=Step3 mod m2 =?
step3_=[01010101]