ПРАКТИЧНА РОБОТА 07

Забезпечення безпеки в операційних системах

Система оцінювання

№	Тема	К-ть балів
1.	*Захист принаймні одного завдання з роботи*	1
2.	Практичні завдання	3,6*
3.	*Здача звіту*	0,4
	Всього	**5**

* – діє бонусна система

AES – симетричний ітеративний блочний алгоритм шифрування, який дозволяє зашифровувати та розшифровувати 128-бітні блоки даних. Загалом алгоритм дозволяє використовувати три ключі різної довжини: 128 (AES-128) бітів, 192 (AES-192) біти чи 256 (AES-256) бітів. Від розміру ключа залежить кількість раундів шифрування:

- 128 бітів – 10 раундів;
- 192 біти – 12 раундів;
- 256 бітів – 14 раундів.

Усі раунди, крім останнього, ідентичні.

128-бітні блоки зручно задавати побайтово (16 байтів), записуючи значення байтів у шістнадцятковій системі числення. Позначимо:

$in_0, in_1, \dots, in_{15}$ – 16 байтів відкритого тексту;

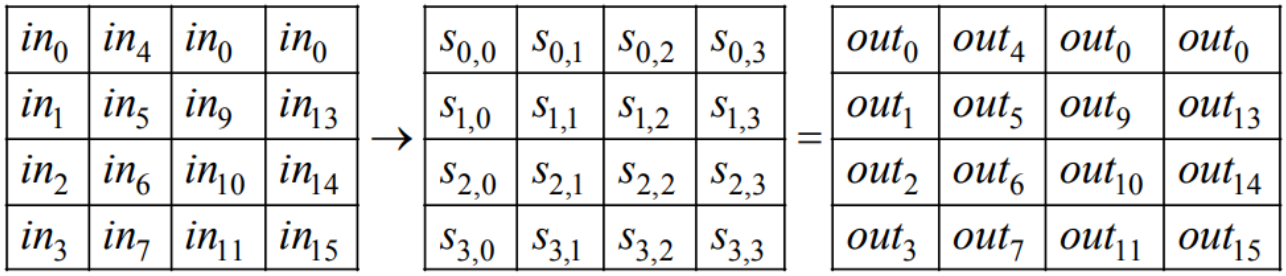
$k_0, k_1, \dots, k_{15}$ – 16 байтів ключа шифру;

$out_0, out_1, \dots, out_{15}$ – 16 байтів зашифрованого тексту.

Вхідними даними для операцій шифрування є масив з 16 байтів $in_0, in_1, \dots, in_{15}$. До початку шифрування байти цього масиву розміщуються послідовно в *стовпці* матриці *InputBlock*. Всередині алгоритму операції виконуються над матрицею байтів State, яку називають ***матрицею станів*** або просто ***станом***. Кінцеве значення матриці *OutputBlock* є виходом алгоритму та перетворюється в послідовність байтів зашифрованого тексту $out_0, out_1, \dots, out_{15}$. Аналогічно, в стовпці матриці InputKey потрапляють 16 байтів $k_0, k_1, \dots, k_{15}$ ключа шифру. Розмірність усіх матриць – 4x4 (рис. 1).

Чотири байти в кожному *стовпці* матриці станів або ключа можна розглядати як одне 32-бітне ***слово***. Тому матриця станів – це масив з 4 слів:

$$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0}; \quad w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1};$$
$$w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}; \quad w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}.$$



Вхідні байти InputBlock | State | Вихідні байти OutputBlock

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}

Матриця InputKey

Рис. 1. Матриці алгоритму

Матрицю, яка надходить на вхід кожного раунду, називається матрицею *InputState* (для першого раунду *InputState* = *InputBlock*), а на виході раунду отримується матриця OutputState (на виході останнього раунду *OutputState* = *OutputBlock*). Наприклад, сформуємо матрицю *InputBlock* для вхідного тексту

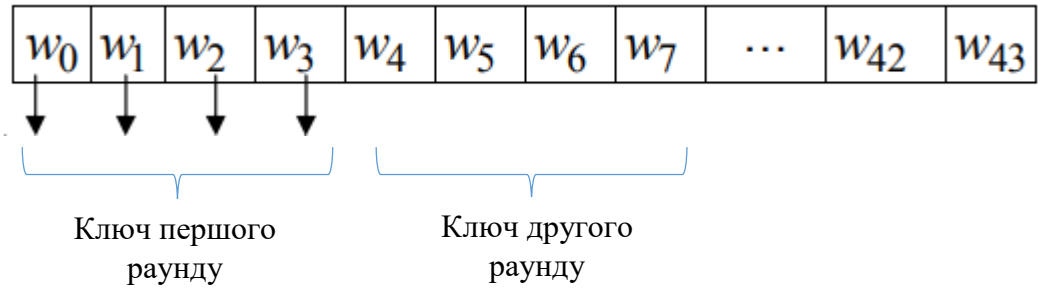
СКЛАДНІСТЬЗАДАЧІ = (21 14 15 00 05 17 11 21 22 30 09 00 05 00 27 11)$_{10}$= = (15 0E 0F 00 05 11 0B 15 16 1E 09 00 05 00 1B 0B)$_{16}$.

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

Ключ шифру також фактично складається з 4 слів:

$w_0 = k_0k_1k_2k_3$; $w_1 = k_4k_5k_6k_7$; $w_2 = k_8k_9k_{10}k_{11}$; $w_3 = k_{12}k_{13}k_{14}k_{15}$.

З цих слів за допомогою спеціального алгоритму формується послідовність з 44 слів:



На кожний раунд шифрування подаються по 4 слова з даної послідовності, які на відповідних етапах будуть грати роль раундового ключа (рис. 2).

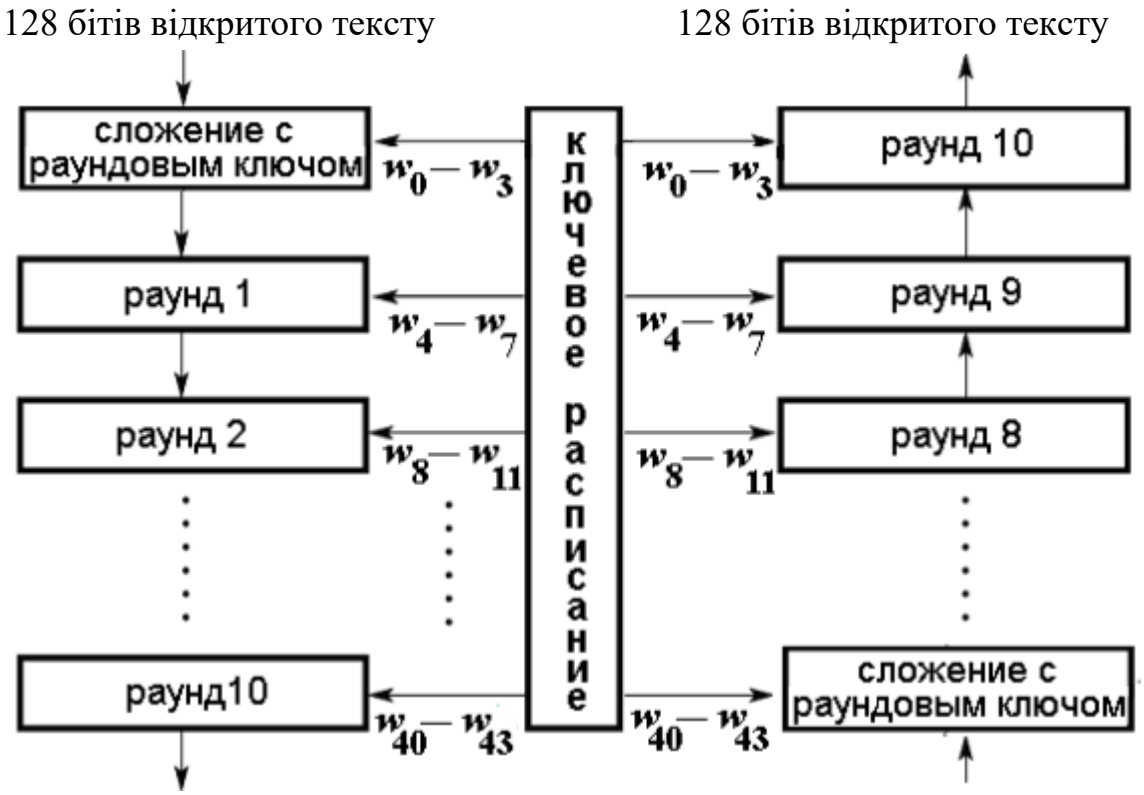


Рис. 2. Схема перетворення даних

Перед першим раундом виконується операція AddRoundKey (додавання за модулем 2 з початковим ключем шифру). Перетворення, виконані в одному раунді, позначають

Round (*State*, *RoundKey*),

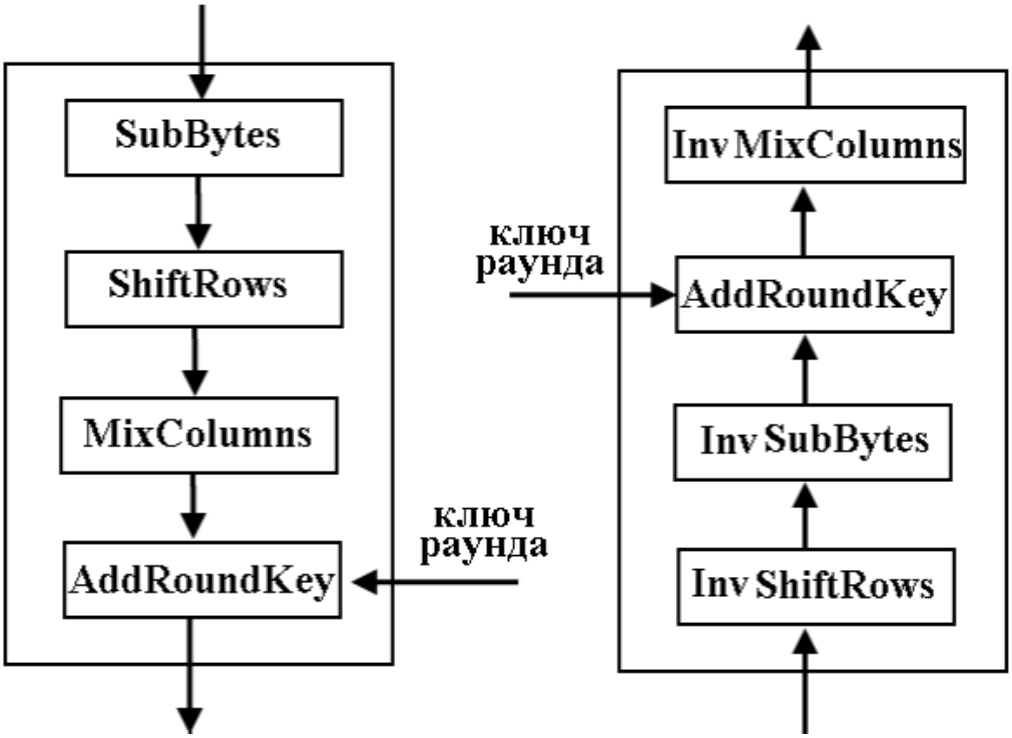
де *State* – матриця, яка описує дані на вході раунду та на його виході після шифрування; змінна *RoundKey* – матриця, яка містить раундовий ключ.

Раунд складається з 4 різних перетворень:

- *SubBytes* – побайтова підстановка в S-боксі з фіксованою таблицею замін;
- *ShiftRows* – побайтовий зсув рядків матриці State на різну кількість байтів;
- *MixColumns* – перемішування байтів у стовпцях;
- *AddRoundKey* – додавання з раундовим ключем (операція XOR).

Останній раунд відрізняється від попередніх тим, що не задіює функцію *MixColumns*. Загальна схема раунду зображена на рис. 3. При дешифруванні в кожному раунді виконуються обернені операції InvShiftRows, InvSubBytes,

AddRoundKey та InvMixColumns. Зверніть увагу, що порядок виконання операцій при шифруванні та дешифруванні відрізняється.



Раунд шифрування Раунд дешифрування

Рис. 3. Схема операцій у раундах

§18. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ШИФРА AES

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, $b^{-1}(x) \equiv a(x) \bmod m(x)$.

Многочлены с коэффициентами, принадлежащими полю $GF(2^8)$. Многочлены третьей степени с коэффициентами из конечного поля $a_i \in GF(2^8)$ имеют вид:
$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0. \qquad (2)$$

Таким образом, в этих многочленах в роли коэффициентов при неизвестных задействованы байты вместо бит. Далее многочлены (2) будем представлять в форме слова $[a_0, a_1, a_2, a_3]$. В стандарте AES при умножении многочленов вида (2) используется приведение по модулю другого многочлена $x^4 + 1$.

Для изучения арифметики рассматриваемых многочленов введем дополнительно многочлен $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, где $b_i \in GF(2^8)$.

Тогда

1. ***Сложение.***

$$a(x)+b(x)=(a_3\oplus b_3)x^3+(a_2\oplus b_2)x^2+(a_1\oplus b_1)x+(a_0\oplus b_0).$$

2. ***Умножение.***

Это более сложная операция. Пусть, мы перемножаем два многочлена $a(x)=a_3x^3+a_2x^2+a_1x+a_0$ и $b(x)=b_3x^3+b_2x^2+b_1x+b_0$. Результатом умножения будет многочлен

$$c(x)=a(x)\cdot b(x)=c_6x^6+c_5x^5+c_4x^4+c_3x^3+c_2x^2+c_1x+c_0,$$

где

$$\begin{aligned}
c_0&=a_0b_0;\\
c_1&=a_1b_0\oplus a_0b_1;\\
c_2&=a_2b_0\oplus a_1b_1\oplus a_0b_2;\\
c_3&=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_0b_3;\\
c_4&=a_3b_1\oplus a_2b_2\oplus a_1b_3;\\
c_5&=a_3b_2\oplus a_2b_3;\\
c_6&=a_3b_3.
\end{aligned}$$

Чтобы результат можно было представить четырехбайтовым словом, необходимо взять результат по модулю многочлена степени не более 4. Авторы шифра выбрали для этой цели многочлен x^4+1, для которого справедливо $x^i \bmod (x^4+1)\equiv x^{i \bmod 4}$.

Поэтому в произведении коэффициенты при степенях x^i, $i=0,1,2,3$, равны сумме произведений a_jb_k по индексам, для которых $j+k=i(\bmod 4)$, $j,k=0,1,2,3$.

Таким образом, после приведения по модулю x^4+1 получим

$$d(x)=a(x)\cdot b(x)=d_3x^3+d_2x^2+d_1x+d_0x,$$

где

$$\begin{aligned}
d_0&=a_0b_0\oplus a_3b_1\oplus a_2b_2\oplus a_1b_3;\\
d_1&=a_1b_0\oplus a_0b_1\oplus a_3b_2\oplus a_2b_3;\\
d_2&=a_2b_0\oplus a_1b_1\oplus a_0b_2\oplus a_3b_3;\\
d_3&=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_0b_3.
\end{aligned}$$

или в матричной форме:

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

Детально розглянемо перетворення раунду шифрування.

1) ***Операція SubBytes*** незалежно виконує нелінійну заміну байтів з кожним байтом матриці State. Заміна оборотна та побудована шляхом комбінації двох перетворень над вхідним байтом:
   - Знаходження зворотного (інфертованого) елементу відносно множення в полі Галуа $GF(2^8)$ (вважається, що байт {00} переходить сам у себе);
   - Виконання деякого афінного перетворення: множення інвертованого байту на многочлен $a(x) = x^4 + x^3 + x^2 + x + 1$ та додавання з многочленом $b(x) = x^6 + x^5 + x + 1$ у полі $\frac{F_2[x]}{x^8} + 1$. Слід зауважити, що $a^{-1}(x) = x^6 + x^3 + x$, а $a^{-1}(x)b(x) = x^2 + 1$. У матричній формі процедура SubBytes записується як

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

   де x – вхідні біти, а y – вихідні. Якщо на вхід подається нульовий байт, то результатом заміни буде число $y = b$. Процес заміни байтів за допомогою таблиці підстановки проілюстровано на рис. 4. Нелінійність перетворення зумовлена нелінійністю інверсії x^{-1}, а оборотність – оборотністю матриці.

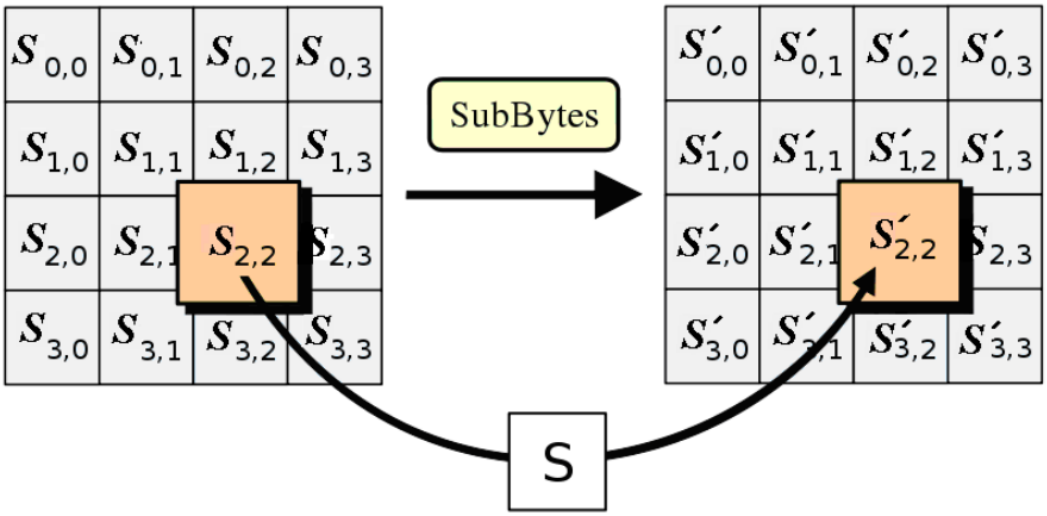


Рис. 4. Заміна байтів за допомогою таблиці підстановки

Утворену на базі цієї операції спеціальну таблицю замін байтів у шістнадцятковій системі називають ***S-боксом***.

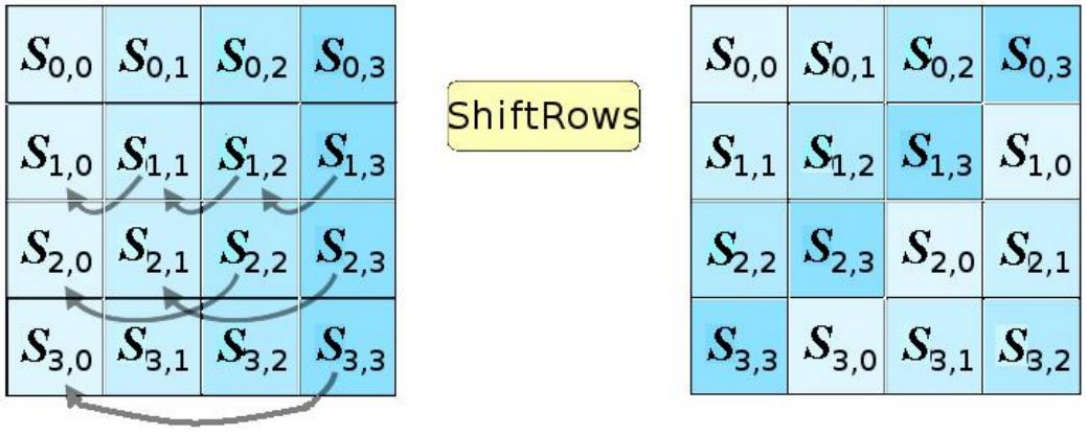
Наприклад, якщо $s_{1,1} = \{8A\}$, то результат заміни цього байту слід шукати на перетині рядки з індексом 8 та стовпця з індексом A, тобто SubBytes(8A)={7E}.

	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

2) ***Операція ShiftRows*** застосовується до рядків матриці *State* – її перший рядок нерухомий, а елементи трьох нижніх рядків циклічно зсуваються вправо на 1, 2 і 3 байти відповідно.

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

За своєю суттю це перестановка елементів матриці, в якій беруть участь тільки елементи рядків, тому таке перетворення оборотне.

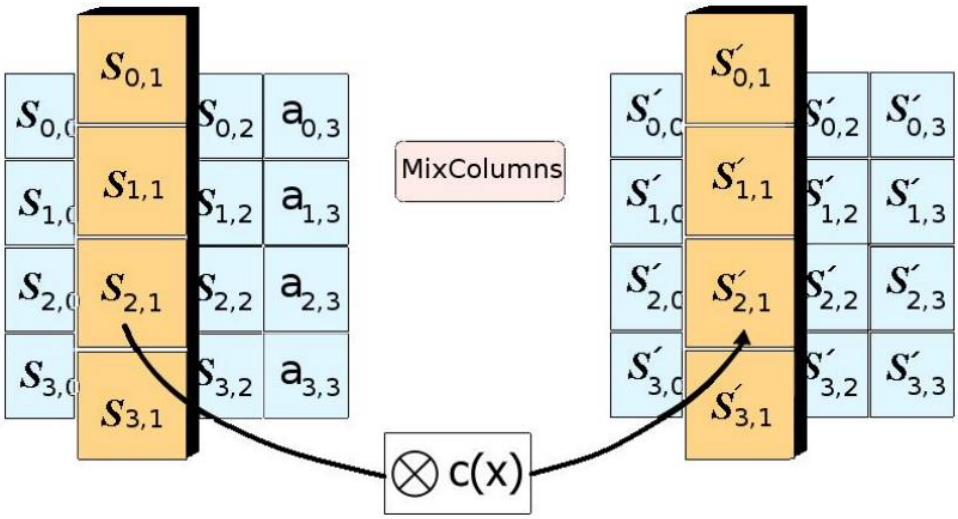


3) ***Операція MixColumns*** використовується для перемішування байтів у стовпцях матриці *State*. Кожний стовпець цієї матриці приймається за многочлен над полем $GF(2^8)$ та множиться на фіксований многочлен

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

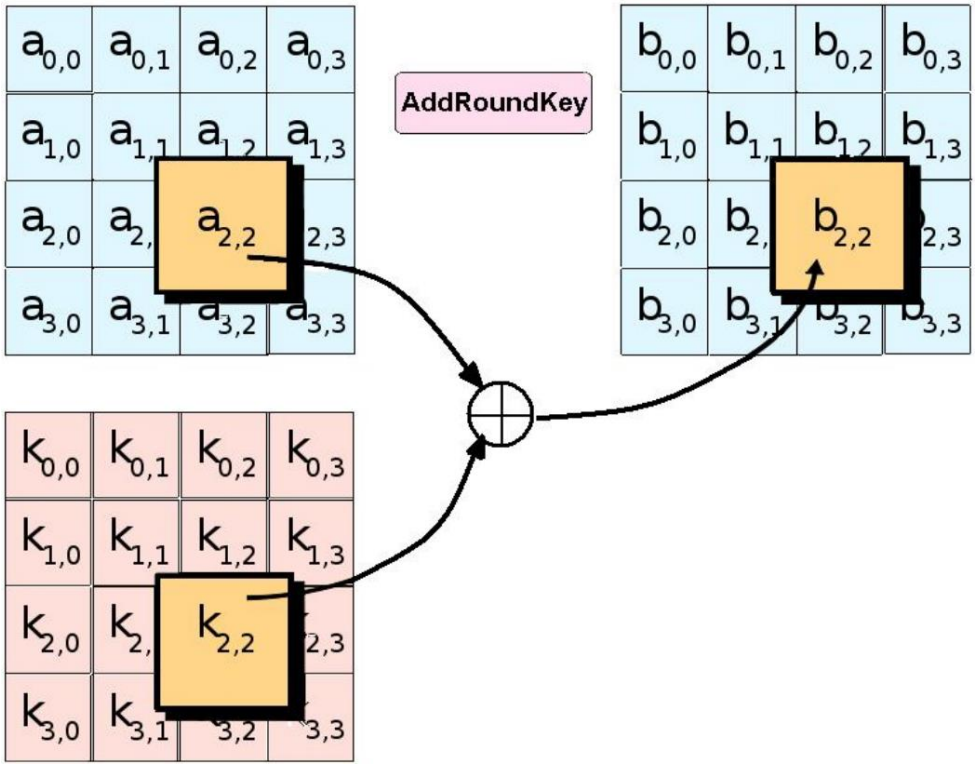
за модулем многочлена $x^4 + 1$ (пам'ятайте, що всі коефіцієнти над полем $GF(2^8)$ – байти). Як показано вище, таку операцію можна записати в матричному вигляді як

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

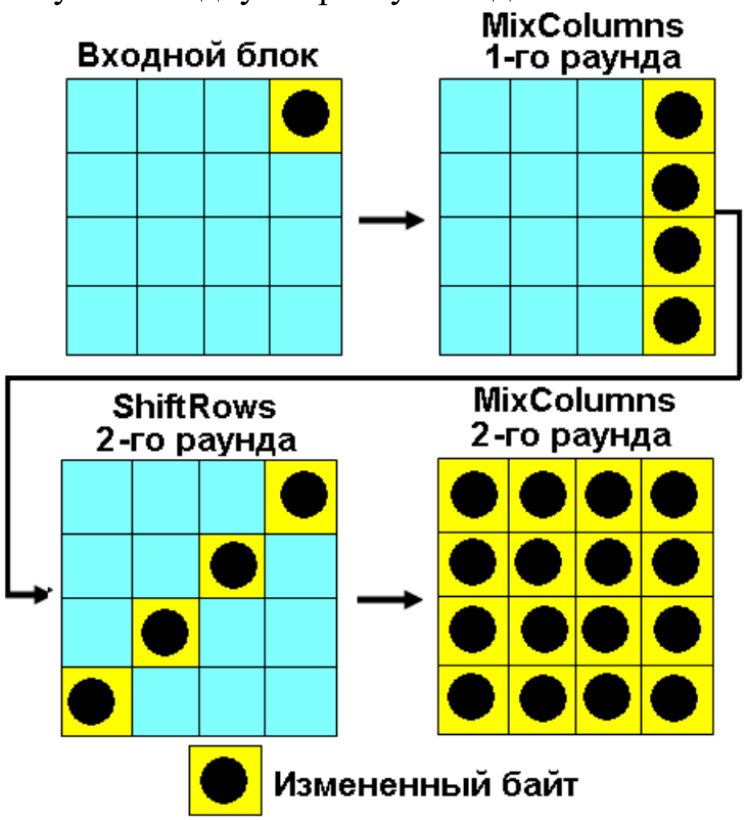


Многочлен $c(x)$ – взаємно простий з многочленом $x^4 + 1$ над полем $GF(2)$, тому в полі існує обернений многочлен $c^{-1}(x)(mod\ x^4 + 1)$. Матриця в цій формулі – оборотна.

4) ***Операція AddRoundKey***. Функція AddRoundKey(*State*, *RoundKey*) побітово додає елементи змінної RoundKey та елементи змінної State за принципом: i-тий стовпець даних ($i = 0,1,2,3$) додається з певним 4-байтним фрагментом розширеного ключа $W[4r + 1]$, де r – номер поточного раунду алгоритму. При шифруванні перше додавання ключа раунду відбувається до першого виконання операції *SubBytes*.



Наступний рисунок демонструє властивості розсіювання та перемішування інформації під час шифрування алгоритмом AES. Видно, що 2 раунди забезпечують повне розсіювання та перемішування інформації. Це досягається за рахунок функцій ShiftRows та MixColumns. Операція SubBytes надає шифруванню стійкість проти диференціального криптоаналізу, а операція AddRoundKey забезпечує необхідну секретну випадковість.



Раундові ключі виробляються з ключа шифру K за допомогою процедури ***розширення ключа***, у результаті чого формується масив раундових ключів, з якого потім безпосередньо обирається необхідний раундовий ключ.

Кожний раундовий ключ має довжину 128 бітів, а довжина в бітах усіх раундових ключів дорівнює 128 бітів · (10 раундів + 1) = 1408 бітів (44 чотирибайтових слова $w_0, w_1, w_2, \ldots, w_{42}, w_{43}$). Перші 4 слова w_0, w_1, w_2, w_3 у ключовому масиві заповнені ключем шифру, з решти напрацьованих 40 слів обирається по 4 слова для ключа раунду.

Нові слова $w_{i+4}, w_{i+5}, w_{i+6}, w_{i+7}$ наступного раундового ключа визначаються зі слів $w_i, w_{i+1}, w_{i+2}, w_{i+3}$ попереднього ключа на основі рівнянь

$$w_{i+5} = w_{i+4} \oplus w_{i+1};$$
$$w_{i+6} = w_{i+5} \oplus w_{i+2};$$
$$w_{i+7} = w_{i+6} \oplus w_{i+3}.$$

Перше слово w_{i+4} у кожному раундовому ключі змінюється по-іншому:

$$w_{i+4} = w_i \oplus g(w_{i+3}),$$

Тут дія функції g зводиться до послідовного виконання 3 кроків, що відображають слово в слово:

- Циклічний зсув чотирибайтного слова вліво на 1 байт (операція RotWord);
- Заміна кожного байту слова, отриманого на попередньому кроці, у відповідності з таблицею SubBytes (операція SubWord);
- Додавання за модулем 2 байтів, отриманих на попередньому кроці, до несекретної та унікальної для кожного раундового ключа K_i раундової константи $R_{con}[i] = (RC[i], 0,0,0)$. Три байти найбільш справа в цій константі – нульові, а ненульовий лівий байт змінюється відповідно до відомого закону рекурсії: $RC[1] = 1,\ RC[i] = 2 \cdot RC[i-1], i = 1,2, \dots ,10.$

Мета додавання раундових констант – зруйнувати будь-яку симетрію, що може виникнути на різних етапах розгортання ключа та призвести до появи слабких ключів, як в алгоритмі DES. Робота алгоритму розширення ключа продемонстрована на рис. 5.

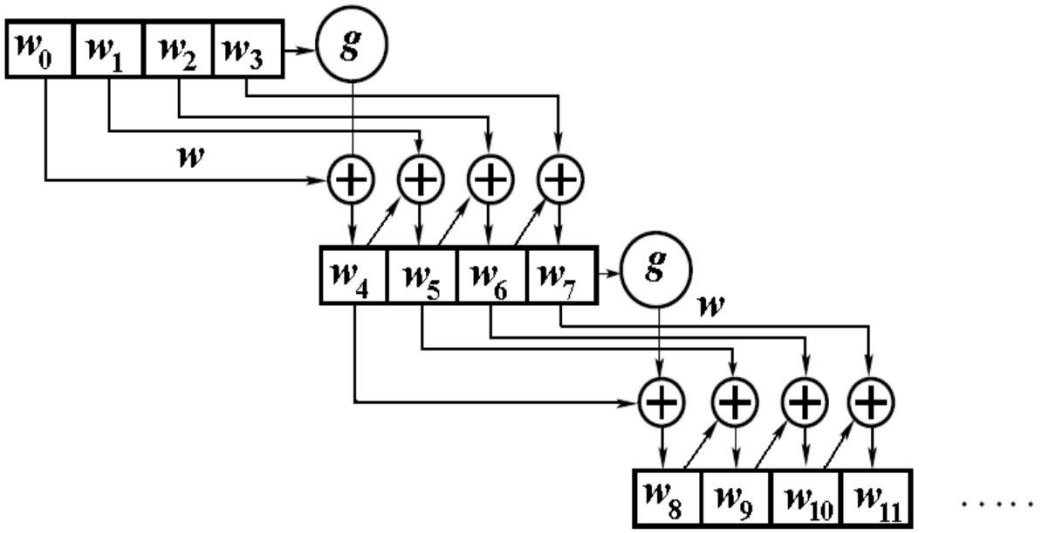


Рис. 5. Алгоритм розширення ключа

Приклад. Початок шифрування.

Ключ шифру 0F 15 71 C9 47 D9 E8 59 0C B7 AD DF AF 7F 67 98

Раундовые ключи	Функция $g(w)$
w_0 =0F 15 71 C9 w_1 =47 D9 E8 59 w_2 =0C B7 AD DF w_3 =AF 7F 67 98	RotWord(w_3) = 7F 67 98 AF = x_1 SubWord(x_1) = D2 85 46 79 = y_1 R_{con} [1] = 01 00 00 00 y_1 + R_{con} [1] = D3 85 46 79 = z_1
w_4 = w_4 + z_1 = DC 90 37 B0 w_5 = w_4 + w_1 = 9B 49 DF E9 w_6 = w_5 + w_2 = 97 FE 72 3F w_7 = w_6 + w_3 = 38 81 15 A7	RotWord(w_7) = 81 15 A7 38 = x_2 SubWord(x_4) = 0C 59 5C 07 = y_2 R_{con} [2] = 02 00 00 00 y_2 + R_{con} [2] = 0E 59 5C 07 = z_2
w_8 = w_4 + z_2 = D2 C9 6B B7 w_9 = w_8 + w_5 = 49 80 B4 5E w_{10} = w_9 + w_6 = DE 7E C6 61 w_{11} = w_{10} + w_7 = E6 FF D3 C6	RotWord(w_{11}) = FF D3 C6 E6 = x_3 SubWord(x_2) = 16 66 B4 8E = y_3 R_{con} [3] = 04 00 00 00 y_3 + R_{con} [3] = 12 66 B4 8E = z_3

Практичні завдання

1. ***1,6 бала*** Реалізуйте безпечну передачу трафіку в TCP-тунелі, використовуючи *власну* реалізацію алгоритму Advanced Encryption Standard (AES-128). Для прикладу можете розглянути реалізацію мовою Python на основі бібліотеки PyCrypto.
2. ***0.8 бала*** Розробіть програмне забезпечення для автентифікації користувача, використовуючи хешування паролю відповідно до хеш-функції згідно з Вашим варіантом:

№	Хеш-функція	Сіль
1	SHA-256	ytggy345bfs
2	SHA-1	p3o4fre7fg4
3	SHA-384	nbvr5o64e11
4	SHA-512	87934fdghcx
5	MD5	63gfaq5qqq5
6	SHA-224	mbcygffd093
7	SHA-256	fdf34jv0bwe
8	SHA-1	ooptet56ghs

9	SHA-384	bvcrty7gg6f
10	SHA-512	7kk8re2vcxz
11	MD5	09tgfio63gdm
12	SHA-224	m823fsdjzzza
13	SHA-256	zxcvbnm0111
14	SHA-1	987654321gtv
15	SHA-384	memoopcpp2
16	SHA-512	lkwevbttyd30
17	MD5	aaaaaaaaaaa
18	SHA-224	zcdpok65w2

Базу даних з паролями можете зберігати у вигляді текстового файлу. Окремо постачайте конфігураційний файл, який міститиме сіль. При реалізації мовою Python рекомендується використовувати бібліотеку hashlib. Доповніть зміст вище згаданими файлами.

3. ***(Завдання для всіх)*** *1,2 бала* Виконайте хешування MAC-адреси свого комп'ютера за допомогою алгоритму MD5 та створіть зведену таблицю з хешами для груп. Оберіть один з методів отримання MAC-адреси за допомогою мови Python. Реалізуйте клієнт-серверне програмне забезпечення, яке триматиме на «сервері» скрипт з фільтруванням MAC-адрес по групах. Наприклад, при доречності MAC-адреси для своєї групи, буде виводитись повідомлення про перехід у зону групи. Але при зверненні в зону чужої групи доступ буде заблоковано (виведено спеціальне повідомлення). Можливо, забажаєте спробувати створити простий веб-сервер.