Розділ 21. Симетричні криптосистеми

21.1. Модель симетричної криптосистеми.

21.2. Класичні методи шифрування: Шифр Цезаря, Вернама. Квадрат Полібія. Шифр гамування.

21.3. Блокові шифри: DES. ДСТУ ГОСТ 28147:2009. AES. ДСТУ 7624:2014 (режими роботи, довжина ключів, довжина блоку вхідного тексту, кількість раундів, крипостійкість згідно з ДСТУ ISO/IEC 10116:2019).

21.4. Потокові шифри: RC4, STRUMOK. (ДСТУ 8845:2019) (довжина ключів, криптостійкість).

21.1. Модель симетричної криптосистеми

Розглянемо загальну схему ***симетричної, або традиційної, криптографії***.

Рисунок 21.1 – Загальна схема симетричного шифрування

У процесі шифрування використовується певний алгоритм шифрування, на вхід якому подаються вихідне незашифроване повідомлення, називане також ***plaintext***, і ключ. Виходом алгоритму є зашифроване повідомлення, називане також ***ciphertext***. Ключ є значенням, що не залежить від шифруємого повідомлення. Зміна ключа повинна приводити до зміни зашифрованого повідомлення.

Зашифроване повідомлення передається одержувачеві. Одержувач перетворить зашифроване повідомлення у вихідне незашифроване повідомлення за допомогою алгоритму дешифрування й того ж самого ключа, що використовувався при шифруванні, або ключа, легко одержуваного із *ключа шифрування*.

Незашифроване повідомлення будемо позначати P або M, від слів *plaintext* і message. Зашифроване повідомлення будемо позначати C, від слова ciphertext.

Безпека, забезпечувана традиційною *криптографією*, залежить від декількох факторів:

– Криптографічний алгоритм повинен бути досить сильним, щоб передане зашифроване повідомлення неможливо було розшифрувати без ключа, використовуючи тільки різні статистичні закономірності зашифрованого повідомлення або які-небудь інші способи його аналізу.

– Безпека переданого повідомлення повинна залежати від таємності ключа, але не від таємності алгоритму. Алгоритм повинен бути проаналізований фахівцями, щоб виключити наявність слабких місць, при яких погано схований взаємозв'язок між незашифрованим і зашифрованим повідомленнями. До того ж при виконанні цієї умови виробники можуть створювати дешеві апаратні чипи й вільно розповсюджувані програми, що реалізують даний алгоритм шифрування.

– Алгоритм повинен бути таким, щоб не можна було довідатися ключ, навіть знаючи досить багато пар (зашифроване повідомлення, незашифроване повідомлення), отриманих при шифруванні з використанням даного ключа.

Клод Шеннон увів поняття *дифузії* й *конфузії* для опису *стійкості алгоритму* шифрування.

Дифузія – це розсіювання статистичних особливостей незашифрованого тексту в широкому діапазоні статистичних особливостей зашифрованого тексту. Це досягається тим, що значення кожного елемента незашифрованого тексту впливає на значення багатьох елементів зашифрованого тексту або, що той же саме, будь-який елемент зашифрованого тексту залежить від багатьох елементів незашифрованого тексту.

Конфузія – це знищення статистичного взаємозв'язку між зашифрованим текстом і ключем.

Якщо X – це вихідне повідомлення й K – криптографічний ключ, то зашифрований переданий текст можна записати у вигляді

$$Y = E_K[X].$$

Одержувач, використовуючи той же ключ, розшифровує повідомлення

$$X = D_K[Y]$$

Супротивник, не маючи доступу до K і X, повинен спробувати довідатися X, K або й те, і інше.

Алгоритми симетричного шифрування розрізняються способом, яким обробляється вихідний текст. **Можливе шифрування блоками або шифрування потоком.**

21.2. Класичні методи шифрування: Шифр Цезаря, Вернама. Квадрат Полібія. Шифр гамування

Шифр Цезаря

Теоретичні відомості. Розглянемо задачу, що полягає в захисті інформації від несанкціонованого доступу. Розглянемо алфавіт укр. мови та занумеруємо літери так, щоб А мала номер 0, Б – номер 1 і т.д. та "розташуємо" їх в рядок згідно номерів. Внизу випишемо ще такий самий рядок, зсунувши його вправо на одну літеру та переставимо літеру Я на початок другого рядка:

А Б В Г Д Е Є Ж З І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я
Я А Б В Г Д Е Є Ж З І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю

Цим самим виконаємо зсув на одну літеру. Аналогічним чином визначаємо зсув на N, або що те саме-циклічну перестановку літер алфавіту. Використаємо цей другий рядок для кодування тексту з метою конфіденційності (захисту від несанкціонованого використання).

Розглянемо обернену задачу – декодування тексту, закодованого при певному зсуві алфавіту. Для того щоб взламати текст закодований методом цезаря застосовується наступний підхід:

Використаємо статистичні методи для визначення величини зсуву наступним чином:

1. Побудуємо таблицю 1 ймовірності всіх літер алфавіту на базі величезного текстового файлу.
2. Побудуємо таблицю 2 частот появи літер в закодованому текстовому файлі.
3. Визначаємо величину зсуву алфавіту:

а) Підрахувати суму різниць частоти літери із кожного рядка та ймовірності появи відповідної літери. Наприклад для уявного зсуву на 1 маємо таку суму:

$$S_1=|V_2\,(А)-V_1\,(Я)+V_2\,(Б)-V_1\,(А)+V_2\,(В)-V_1\,(Б)+\ldots+V_2\,(Я)-V_1\,(Ю)|$$

де $V_2\,(А)$ – частота появи літер А в декодованому тексті, $V_1\,(Я)$ – ймовірність появи літери Я в укр. тексті.

б) Якщо позначити через Si-суму, побудовану в а) для уявного зсуву на *i* всіх літер алфавіту, то можливо побудувати множину $\{S_i\}$, $i = 0,\ldots,33$ в якій найменший елемент буде відповідати величині зсуву літер алфавіту.

4.Таким чином статистичним методом знайдемо можливість для декодування тексту, про який відомий спосіб кодування.

Шифр Вернама

Шифр Вернама (інша назва: англ. *one-time pad* – схема одноразових блокнотів) – у криптографії, система симетричного шифрування, винайдена в 1917 році співробітниками AT&T Мейджором Джозефом Моборном і Гільбертом Вернамом. Шифр Вернама є єдиною системою шифрування, для якої доведена абсолютна криптографічна стійкість.

Для утворення шифротексту повідомлення об'єднується операцією XOR з ключем (названим одноразовим блокнотом або шифроблокнотом). При цьому ключ повинен мати три критично важливі властивості:

1. Бути справді випадковим.
2. Збігатися за розміром з заданим відкритим текстом.
3. Застосовуватися тільки один раз.
4. Повинен зберігатися в повній таємниці сторонами, що спілкуються.

Шифр названий на честь телеграфіста AT&T Гільберта Вернама, який у 1917 році побудував телеграфний апарат, що виконував цю операцію автоматично – треба було тільки подати на нього стрічку з ключем. Не будучи шифрувальником, Вернам, тим не менш, помітив важливу властивість свого шифру – кожна стрічка повинна використовуватися тільки один раз і після цього знищуватися. Це було важко застосувати на практиці – тому апарат був перероблений на кілька зациклених стрічок із взаємно простими періодами.

Квадрат Полібія

У криптографії квадрат Полібія (англ. *Polybius square*), також відомий як шахова дошка Полібія – оригінальний код простої заміни, одна з найдавніших систем кодування, запропонована Полібієм (грецький історик, полководець, державний діяч, III століття до н.е.). Цей спосіб кодування спочатку застосовувався для грецької абетки, але потім поширився на інші мови.

Спосіб шифрування

Квадрат спочатку створювався для кодування, з його допомогою можна успішно шифрувати. Для того, щоб зашифрувати текст квадратом Полібія, потрібно зробити кілька кроків:

Крок 1: Формування таблиці шифрування

До кожної мови окремо складається таблиця шифрування з однаковою (не обов'язково) кількістю пронумерованих рядків та стовпців, параметри якої залежать від її потужності (кількості букв в абетці). Беруться два цілих числа, добуток яких найближче до кількості букв у мові – отримуємо потрібну кількість рядків і стовпців. Потім вписуємо в таблицю всі букви алфавіту поспіль – по одній на кожну клітину. При нестачі клітин можна вписати в одну дві букви (рідко вживані або схожі за вживанням).

Латинський алфавіт

У сучасній латинській абетці 26 букв, отже таблиця повинна складатися з 5 рядків і 5 стовпців, оскільки 25=5*5 найбільш близьке до 26 число. При цьому літери I, J не розрізняються (J ототожнюється з буквою I), оскільки не вистачає однієї комірки:

	1	**2**	**3**	**4**	**5**
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Українська абетка

Ідею формування таблиці шифрування проілюструємо для української мови. Число букв в українській абетці відрізняється від числа букв у грецькій абетці, тому розмір таблиці вибрано інший (квадрат 6*6=36, оскільки 36 найбільш близьке число до 33):

	1	**2**	**3**	**4**	**5**	**6**
1	А	Б	В	Г	Ґ	Д
2	Е	Є	Ж	З	И	І
3	Ї	Й	К	Л	М	Н
4	О	П	Р	С	Т	У
5	Ф	Х	Ц	Ч	Ш	Щ
6	Ю	Я	Ь	—	—	—

Можливий також інший варіант складання, що передбачає об'єднання букв Г і Ґ, И і Й, видання І і Ї. В даному випадку отримуємо такий результат:

	1	**2**	**3**	**4**	**5**	**6**
1	А	Б	В	Г/Ґ	Д	Е
2	Є	Ж	З	И/Й	І/Ї	К
3	Л	М	Н	О	П	Р
4	С	Т	У	Ф	Х	Ц
5	Ч	Ш	Щ	Ю	Я	Ь

Використовуючи подібний алгоритм, таблицю шифрування можна задати для будь-якої мови. Щоб розшифрувати закритий текст, необхідно знати, таблицею шифрування якого алфавіту він зашифрований.

Крок 2: Принцип шифрування

Існує кілька методів шифрування з допомогою квадрата Полібія. Нижче наведено три з них.

Метод 1

Зашифруємо слово «SOMETEXT»:

Для шифрування на квадраті знаходили букву тексту і вставляли в шифровку нижню від неї в тому ж стовпці. Якщо буква була в нижньому рядку, то брали верхню з стовпця.

Таблиця координат								
Буква тексту:	**S**	**O**	**M**	**E**	**T**	**E**	**X**	**T**
Буква шифротексту:	**X**	**T**	**R**	**K**	**Y**	**K**	**C**	**Y**

Таким чином після шифрування отримуємо:

Результат	
До шифрування:	**SOMETEXT**
Після шифрування:	**XTRKYKCY**

Метод 2

Повідомлення перетвориться в координати по квадрату Полібія, координати записуються вертикально:

Таблиця координат								
Буква:	**S**	**O**	**M**	**E**	**T**	**E**	**X**	**T**
Координата горизонтальна:	**3**	**4**	**2**	**5**	**4**	**5**	**3**	**4**
Координата вертикальна:	**4**	**3**	**3**	**1**	**4**	**1**	**5**	**4**

Потім координати зчитують по рядках: 34 25 45 34 43 31 41 54 (*)

Далі координати перетворюються в літери з цього ж квадрату:

Таблиця координат								
Координата горизонтальна:	**3**	**2**	**4**	**3**	**4**	**3**	**4**	**5**
Координата вертикальна:	**4**	**5**	**5**	**4**	**3**	**1**	**1**	**4**
Буква:	**S**	**W**	**Y**	**S**	**O**	**C**	**D**	**U**

Таким чином після шифрування отримуємо:

Результат	
До шифрування:	**SOMETEXT**
Після шифрування:	**SWYSOCDU**

Метод 3

Ускладнений варіант, який полягає в наступному: отриманий первинний шифротекст (*) шифрується вдруге. При цьому він виписується без розбиття на пари: 3425453443314154 Отримана послідовність цифр зсувається циклічно вліво на один крок (непарна кількість кроків): 4254534433141543 Ця послідовність знову розбивається в групи по два: 42 54 53 44 33 14 15 43 і за таблицею замінюється на остаточний шифротекст:

Таблиця координат								
Координата горизонтальна:	**4**	**5**	**5**	**4**	**3**	**1**	**1**	**4**
Координата вертикальна:	**2**	**4**	**3**	**4**	**3**	**4**	**5**	**3**
Буква:	**I**	**U**	**P**	**T**	**N**	**Q**	**V**	**O**

Таким чином після шифрування отримуємо:

Результат	
До шифрування:	**SOMETEXT**
Після шифрування:	**IUPTNQVO**

Додавання ключа

На перший погляд шифр здається дуже нестійким, але для його реальної оцінки слід враховувати два фактори:

1. Можливість заповнити квадрат Полібія буквами довільно, а не тільки строго за алфавітом.

2. Можливість періодично замінювати квадрати.

Тоді розбір попередніх повідомлень нічого не дає, оскільки до моменту розкриття шифру він може бути замінений.

Букви можуть вписуватися в таблицю в довільному порядку – заповнення таблиці в цьому випадку і є ключем. Для латинської абетки в першу клітку можна вписати одну з 25 букв, у другу – одну з 24, в третю – одну з 23 і т. д. Отримуємо максимальну кількість ключів для шифру на таблиці латинської абетки:

$$N=25*24*23*...*2*1=25!$$

Відповідно для дешифрування повідомлення буде потрібно не тільки знання абетки, але і ключа, за допомогою якого складалася таблиця шифрування. Але довільний порядок букв важко запам'ятати, тому користувачеві шифру необхідно постійно мати при собі ключ – квадрат. з'являється небезпека таємного ознайомлення з ключем сторонніх осіб. Як компромісне рішення був запропонований ключ – пароль. Пароль виписується без повторів букв в квадрат; в клітини, що залишилися, в абетковому порядку виписуються букви абетки, відсутні в паролі.

Приклад

Зашифруємо слово «SOMETEXT», використовуючи ключ «DRAFT». Складемо попередньо таблицю шифрування з даним ключем, записуючи символи ключа по порядку в таблицю, після них абетки:

	1	**2**	**3**	**4**	**5**
1	D	R	A	F	T
2	B	C	E	G	H
3	I	K	L	M	N
4	O	P	Q	S	U
5	V	W	X	Y	Z

Перетворимо повідомлення в координати по квадрату Полібія:

Таблиця координат								
Буква:	**S**	**O**	**M**	**E**	**T**	**E**	**X**	**T**
Координата горизонтальна:	**4**	**1**	**4**	**3**	**5**	**3**	**3**	**5**
Координата вертикальна:	**4**	**4**	**3**	**2**	**1**	**2**	**5**	**1**

Рахуємо координати по рядках: 41 43 53 35 44 32 12 51

Перетворимо координати в літери з цього ж квадрату:

Таблиця координат								
Координата горизонтальна:	**4**	**4**	**5**	**3**	**4**	**3**	**1**	**5**
Координата вертикальна:	**1**	**3**	**3**	**5**	**4**	**2**	**2**	**1**
Буква:	**F**	**M**	**N**	**X**	**S**	**E**	**B**	**T**

Таким чином після шифрування отримуємо:

Результат	
До шифрування:	**SOMETEXT**
Після шифрування:	**FMNXSEBT**

Шифр гамування

Принцип шифрування гаммуванням полягає в генерації гамми шифру за допомогою датчика псевдовипадкових чисел і накладення отриманої гамми шифру на відкриті дані зверненим способом (наприклад, за допомогою операції складання за модулем 2). Процес дешифрування здійснюється для повторної генерації гамми шифру при відомому ключі та накладенні такої ж гами на зашифровані дані.

Отриманий зашифрований текст є достатньо трудомістким для розкриття в тому випадку, якщо гамма-шифр не містить повторюваних бітових послідовностей і змінюється випадковим чином для кожного зашифрованого слова. Якщо період гамми перевищує довжину всього зашифрованого тексту і невідома жодна частина вихідного тексту, то шифр можна розкрити тільки прямим перебором (підбором ключа). В цьому випадку криптостійкість визначається розміром ключа.

Метод гамування стає бессильним, якщо відомий фрагмент вихідного тексту і відповідна йому шифрограма. В цьому випадку простим відніманням за модулем 2 отримується відрізок псевдовипадковой послідовності й за ним відновлюється вся ця послідовність.

21.3. Блокові шифри: DES. ДСТУ ГОСТ 28147:2009. AES. ДСТУ 7624:2014 (режими роботи, довжина ключів, довжина блоку вхідного тексту, кількість раундів, крипостійкість згідно з ДСТУ ISO/IEC 10116:2019)

Блокові шифри

Блок тексту розглядається як ненегативне ціле число, або як кілька незалежних ненегативних цілих чисел. Довжина блоку завжди вибирається рівною ступеню двійки. У більшості блокових алгоритмів симетричного шифрування використовуються наступні типи операцій:

– Таблична підстановка, при якій група біт відображається в іншу групу біт. Це так звані S-***box***.

– Переміщення, за допомогою якого біти повідомлення переупорядковуються.

– Операція додавання за модулем 2, позначувана XOR або ⊕.

– Операція додавання за модулем 2^{32} або за модулем 2^{16}.

– Циклічне зрушення на деяке число біт.

Ці операції циклічно повторюються в алгоритмі, створюючи так звані ***раунди***. Входом кожного *раунду* є вихід попереднього *раунду* й ключ, що отриманий по певному алгоритму із ключа шифрування K. Ключ *раунду* називається ***підключем***. Кожний алгоритм шифрування може бути представлений у такий спосіб:

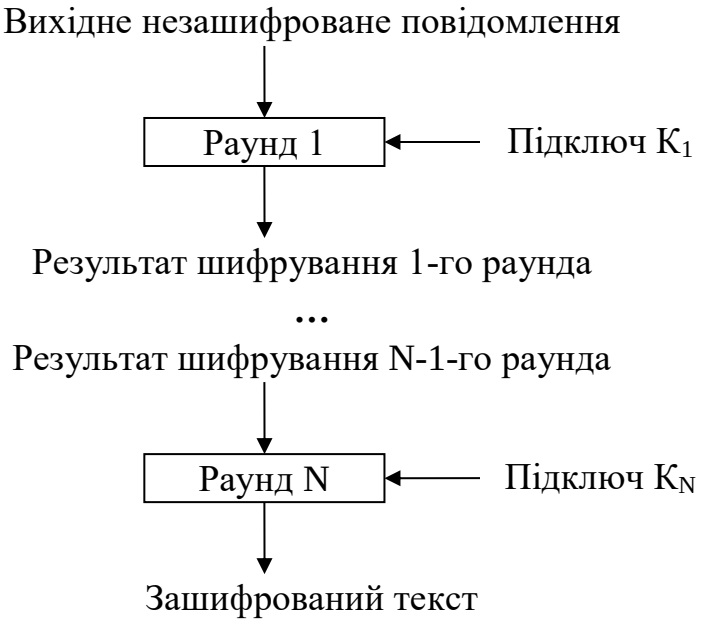


Рисунок 21.2 – Структура алгоритму симетричного шифрування

Області застосування

Стандартний алгоритм шифрування повинен бути застосуємо в багатьох додатках:

– Шифрування даних. Алгоритм повинен бути ефективний при шифруванні файлів даних або великого потоку даних.

– Створення випадкових чисел. Алгоритм повинен бути ефективний при створенні певної кількості випадкових біт.

– Ґешування. Алгоритм повинен ефективно перетворюватися в однобічну ґеш-функцію.

Платформи

Стандартний алгоритм шифрування повинен бути реалізований на різних платформах, які, відповідно, висувають різні вимоги.

– Алгоритм повинен ефективно реалізовуватися на спеціалізованій апаратурі, призначеної для виконання шифрування/дешифрування.

– Великі процесори. Хоча для найбільш швидких додатків завжди використовується спеціальна апаратура, програмні реалізації застосовуються частіше. Алгоритм повинен допускати ефективну програмну реалізацію на 32-бітних процесорах.

– Процесори середнього розміру. Алгоритм повинен працювати на мікроконтролерах і інших процесорах середнього розміру.

– Малі процесори. Повинна існувати можливість реалізації алгоритму на смарт-картах, нехай навіть із урахуванням твердих обмежень на використовувану пам'ять.

Додаткові вимоги

Алгоритм шифрування повинен, по можливості, задовольняти деяким додатковим вимогам.

– Алгоритм повинен бути простим для написання коду, щоб мінімізувати ймовірність програмних помилок.

– Алгоритм повинен мати плоский простір ключів і допускати будь-який випадковий рядок біт потрібної довжини як можливий ключ. Наявність слабких ключів небажано.

– Алгоритм повинен легко модифікуватися для різних рівнів безпеки й задовольняти як мінімальним, так і максимальним вимогам.

– Всі операції з даними повинні здійснюватися над блоками, кратними байту або 32-бітному слову.

Мережа Фейштеля

Блоковий алгоритм перетворює n-бітний блок незашифрованого тексту в n-бітний блок зашифрованого тексту. Число блоків довжини n дорівнює 2^n. Для того щоб перетворення було оборотним, кожний з таких блоків повинен перетворюватися у свій унікальний блок зашифрованого тексту. При маленькій довжині блоку така підстановка погано приховує статистичні особливості незашифрованого тексту. Якщо блок має довжину 64 біта, то він уже добре приховує статистичні особливості вихідного тексту. Але в цьому випадку перетворення тексту не може бути довільним у силу того, що ключем буде саме перетворення, що виключає ефективну як програмну, так і апаратну реалізації.

Найбільш широке поширення одержали ***мережі Фейштеля***, тому що, з одного боку, вони задовольняють всім вимогам до алгоритмів симетричного шифрування, а з іншого боку, досить прості й компактні.

Мережа Фейштеля має наступну структуру. Вхідний блок ділиться на трохи рівної довжини підблоків, називаних гілками. У випадку, якщо блок має довжину 64 біта, використовуються дві гілки по 32 біта кожна. Кожна гілка обробляється незалежно від інший, послу чого здійснюється циклічне зрушення всіх гілок уліво. Таке перетворення виконується кілька циклів або *раундів*. У випадку двох гілок кожний *раунд* має структуру, показану на рисунку:

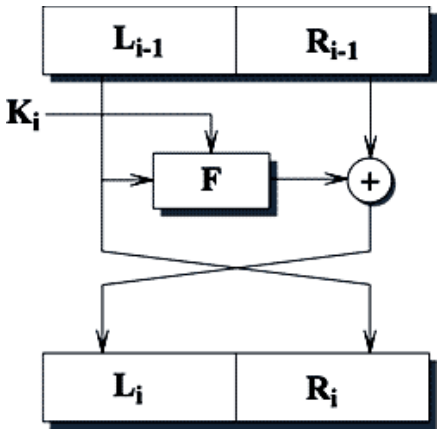


Рисунок 21.3 – і-ий раунд мережі Фейштеля

Функція F називається утворюючою. Кожний *раунд* складається з обчислення функції F для однієї гілки й побітового виконання операції XOR результату F з іншою гілкою. Після цього гілки міняються місцями. Вважається, що оптимальне число *раундів* – від 8 до 32. Важливо те, що збільшення кількості *раундів* значно збільшує криптостійкість алгоритму. Можливо, ця особливість і вплинула на настільки активне поширення *мережі Фейштеля*, тому що для більшої криптостійкости досить просто збільшити кількість *раундів*, не змінюючи сам алгоритм. Останнім часом кількість *раундів* не фіксується, а лише вказуються припустимі межі.

Мережа Фейштеля є оборотною навіть у тому випадку, якщо функція F не є такою, тому що для дешифрування не потрібно обчислювати F^{-1}. Для дешифрування використовується той же алгоритм, але на вхід подається зашифрований текст, і ключі використовуються у зворотному порядку.

Останнім часом все частіше використовуються різні різновиди *мережі Фейштеля* для 128-бітного блоку із чотирма гілками. Збільшення кількості галузей, а не розмірності кожної гілки пов'язане з тим, що найбільш популярними дотепер залишаються процесори з 32-розрядними словами, отже, оперувати 32-розрядними словами ефективніше, ніж з 64-розрядними.

Основною характеристикою алгоритму, побудованого на основі *мережі Фейштеля*, є функція F. Різні варіанти стосуються також початкового й кінцевого перетворень. Подібні перетворення, називані забілюванням (whitening), здійснюються для того, щоб виконати початкову рандомизацію вхідного тексту.

Алгоритм DES

Принципи розробки

Найпоширенішим і найбільш відомим алгоритмом симетричного шифрування є ***DES*** (Data Encryption Standard). Алгоритм був розроблений в 1977 році, в 1980 році був прийнятий NIST (National Institute of Standards and Technolody США) як стандарт (FIPS PUB 46).

DES є класичною *мережею Фейштеля* із двома гілками. Дані шифруються 64-бітними блоками, використовуючи 56-бітний ключ. Алгоритм перетворить за кілька *раундів 64*-бітний вхід в 64-бітний вихід. Довжина ключа дорівнює 56 бітам. Процес шифрування складається із чотирьох етапів. На першому з них виконується початкова перестановка (IP) 64-бітного вихідного тексту (забілювання), під час якої біти з у відповідності зі стандартною таблицею. Наступний етап складається з 16 *раундів* однієї й тої ж функції, що використовує операції зрушення й підстановки. На третьому етапі ліва й права половини виходу останньої (16-й) ітерації міняються місцями. Нарешті, на четвертому етапі виконується перестановка IP^{-1} результату, отриманого на третьому етапі. Перестановка IP^{-1} інверсна початковій перестановці.

Рисунок 21.4 – Загальна схема DES

Праворуч на рисунку показаний спосіб, яким використовується 56-бітний ключ. Спочатку ключ подається на вхід функції перестановки. Потім для кожного з 16 *раундів підключ* K_i є комбінацією лівого циклічного зрушення й перестановки. Функція перестановки та сама для кожного *раунду*, але *підключи* K_i для кожного *раунду* виходять різні внаслідок повторюваного зрушення біт ключа.

Потрійний DES

У цей час основним недоліком *DES* вважається маленька довжина ключа, тому вже давно почали розроблятися різні альтернативи цьому алгоритму шифрування. Один з підходів полягає в тому, щоб розробити новий алгоритм, і успішний тому приклад – **IDEA**. Інший підхід припускає повторне застосування шифрування за допомогою *DES* з використанням декількох ключів.

Недоліки подвійного DES

Найпростіший спосіб збільшити довжину ключа складається в повторному застосуванні *DES* із двома різними ключами. Використовуючи незашифроване повідомлення P і два ключі K_1 і K_2, зашифроване повідомлення C можна одержати в такий спосіб:

$$C = E_{k2} [E_{k1} [P]]$$

Для дешифрування потрібно, щоб два ключі застосовувалися у зворотному порядку:

$$P = D_{k1} [D_{k2} [C]]$$

У цьому випадку довжина ключа дорівнює 56 * 2 = 112 біт.

Потрійний DES із двома ключами

Очевидна протидія атаці "зустріч посередині", що буде розглядатися пізніше, складається у використанні третьої стадії шифрування із трьома різними ключами. Це піднімає вартість лобової атаки до 2^{168}, що на сьогоднішній день вважається вище практичних можливостей. Але при цьому довжина ключа дорівнює 56 * 3 = 168 біт, що іноді буває громіздко.

Як альтернатива пропонується метод потрійного шифрування, що використовує тільки два ключі. У цьому випадку виконується послідовність **зашифрування-розшифрування-зашифрування (EDE).**

$$C = E_{K1} [D_{K2} [E_{K1} [P]]]$$

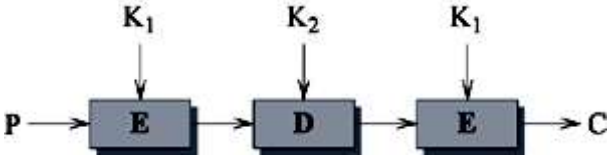


Рисунок 21.5 – Шифрування потрійним DES

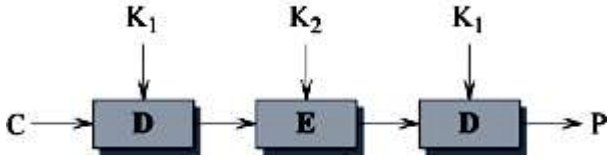


Рисунок 21.6 – Дешифрування потрійним DES

Не має великого значення, що використовується на другій стадії: шифрування або дешифрування. У випадку використання дешифрування існує тільки та перевага, що можна *потрійний DES* звести до звичайного одиночного *DES*, використовуючи $K_1 = K_2$:

$$C = E_{K1} [D_{K1} [E_{K1} [P]]] = E_{K1} [P]$$

Потрійний DES є досить популярною альтернативою *DES* і використовується при керуванні ключами в стандартах ANSI X9.17 і ISO 8732 і в PEM (Privacy Enhanced Mail).

Відомих криптографічних атак на *потрійний DES* не існує. Ціна підбора ключа в *потрійному DES* дорівнює 2^{112}.

Алгоритм Blowfish

Blowfish є мережею Фейштеля, у якої кількість ітерацій дорівнює 16. Довжина блоку дорівнює 64 бітам, ключ може мати будь-яку довжину в межах 448 біт. Хоча перед початком будь-якого шифрування виконується складна фаза ініціалізації, саме шифрування даних виконується досить швидко.

Алгоритм призначений в основному для додатків, у яких ключ міняється нечасто, до того ж існує фаза початкового рукостискання, під час якої відбувається автентифікація сторін і узгодження загальних параметрів і секретів. Класичним прикладом подібних додатків є мережна взаємодія. При реалізації на 32-бітних мікропроцесорах з великим кешем даних *Blowfish* значно швидше DES.

Алгоритм складається із двох частин: розширення ключа й шифрування даних. Розширення ключа перетворить ключ довжиною, принаймні, 448 біт у кілька масивів *підключей* загальною довжиною 4168 байт.

В основі алгоритму лежить мережа Фейштеля з 16 ітераціями. Кожна ітерація складається з перестановки, що залежить від ключа, і підстановки, що залежить від ключа й даних. Операціями є XOR і додавання 32-бітних слів.

Blowfish використовує велику кількість *підключей*. Ці ключі повинні бути обчислені заздалегідь, до початку будь-якого шифрування або дешифрування даних.

Алгоритм IDEA

IDEA (International Data Encryption Algorithm) є блоковим симетричним алгоритмом шифрування, розробленим Сюдзя Гавкіт (Xuejia Lai) і Джеймсом Массей (James Massey) зі швейцарського федерального інституту технологій. Первісна версія була опублікована в 1990 році. Переглянута версія алгоритму, посилена засобами захисту від диференціальних криптографічних атак, була представлена в 1991 році й докладно описана в 1992 році.

IDEA є одним з декількох симетричних криптографічних алгоритмів, якими спочатку передбачалося замінити DES.

Принципи розробки

IDEA є блоковим алгоритмом, що використовує 128-бітовий *ключ для шифрування* даних блоками по 64 біта.

Метою розробки *IDEA* було створення щодо стійкого криптографічного алгоритму з досить простою реалізацією.

Криптографічна стійкість

Наступні характеристики *IDEA* характеризують його криптографічну стійкість:

1. **Довжина блоку**: довжина блоку повинна бути достатньою, щоб сховати всі статистичні характеристики вихідного повідомлення. З іншого боку, складність реалізації криптографічної функції зростає експоненціально відповідно до розміру блоку. Використання блоку розміром в 64 біта в 90-і роки означало достатню силу. Більше того, використання режиму шифрування CBC говорить про подальше посилення цього аспекту алгоритму.

2. **Довжина ключа**: довжина ключа повинна бути досить великою для того, щоб запобігти можливості простого перебору ключа. При довжині ключа 128 біт *IDEA* вважається досить безпечним.

3. **Конфузія**: зашифрований текст повинен залежати від ключа складним і заплутаним способом.

4. **Дифузія**: кожний біт незашифрованого тексту повинен впливати на кожний біт зашифрованого тексту. Поширення одного незашифрованого біта на велику кількість зашифрованих біт приховує статистичну структуру незашифрованого тексту. Визначити, як статистичні характеристики зашифрованого тексту залежать від статистичних характеристик незашифрованого тексту, повинне бути непросто. *IDEA* із цього погляду є дуже ефективним алгоритмом.

В *IDEA* два останніх пункти виконуються за допомогою трьох операцій. Це відрізняє його від DES, де все побудовано на використанні операції XOR і маленьких нелінійних S-*boxes*.

Алгоритм ДСТУ ГОСТ 28147:2009

Алгоритм ДСТУ ГОСТ 28147:2009 є вітчизняним стандартом для алгоритмів симетричного шифрування. ***ДСТУ ГОСТ 28147:2009*** розроблений в 1989 році, є блоковим алгоритмом шифрування, довжина блоку дорівнює 64 бітам, довжина ключа дорівнює 256 бітам, кількість *раундів* дорівнює 32. Алгоритм являє собою класичну мережу Фейштеля.

$$L_i = R_{i-1}$$

$$R_i = L_i \oplus f\,(R_{i-1}, K_i)$$

Функція F проста. Спочатку права половина й і-ий *підключ* складаються за модулем 2^{32}. Потім результат розбивається на вісім 4-бітових значень, кожне з яких подається на вхід S-*box*. *ДСТУ ГОСТ 28147:2009* використовує вісім різних S-*boxes*, кожний з яких має 4-бітовий вхід і 4-бітовий вихід. Виходи всіх S-*boxes* поєднуються в 32-бітне слово, що потім циклічно зрушується на 11 біт уліво. Нарешті, за допомогою XOR результат поєднується з лівою половиною, у результаті чого виходить нова права половина.

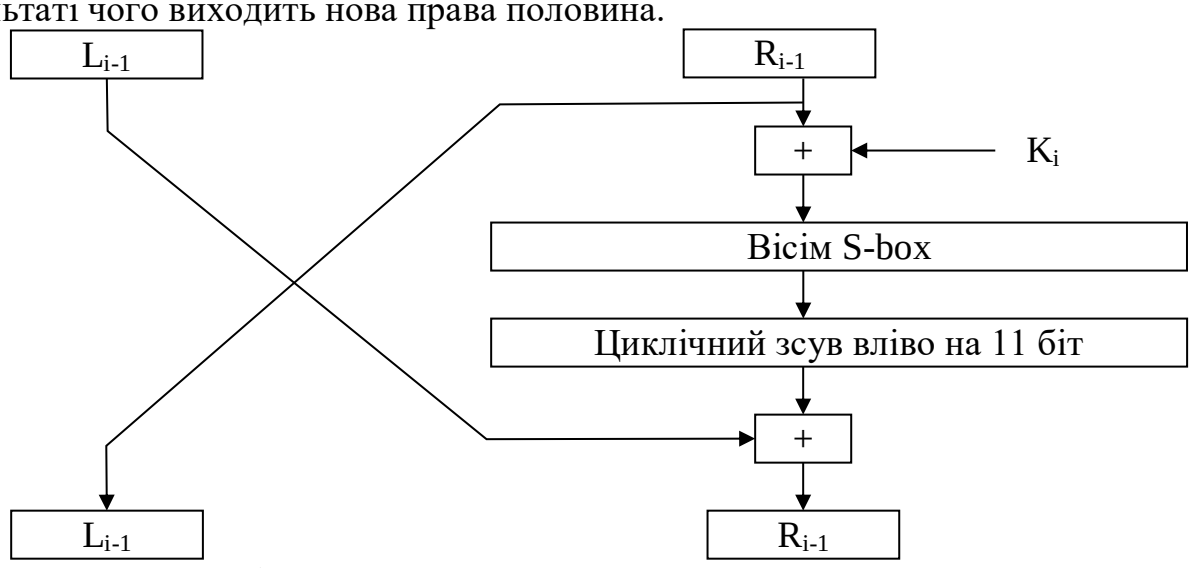


Рисунок 17.7 – і-ий раунд ДСТУ ГОСТ 28147:2009

Режими виконання алгоритмів симетричного шифрування

Для будь-якого симетричного блокового алгоритму шифрування визначено чотири режими виконання.

ECB – Electronic Codebook – кожний блок з 64 біт незашифрованого тексту шифрується незалежно від інших блоків, із застосуванням того самого *ключа шифрування*. Типові додатки – безпечна передача одиночних значень (наприклад, криптографічного ключа).

CBC – Cipher Block Chaining – вхід криптографічного алгоритму є результатом застосування операції XOR до наступного блоку незашифрованого тексту й попередньому блоку зашифрованого тексту. Типові додатки – загальна блокоорієнтована передача, автентифікація.

CFB – Cipher Feedback – при кожному виклику алгоритму обробляється J біт вхідного значення. Попередній зашифрований блок використовується як вхід в алгоритм; до J биток виходу алгоритму й наступному незашифрованому блоку з J біт застосовується операція XOR, результатом якої є наступний зашифрований блок з J біт. Типові додатки – потокоорієнтована передача, автентифікація.

OFB – Output Feedback – аналогічний *CFB*, за винятком того, що на вхід алгоритму при шифруванні наступного блоку подається результат шифрування попереднього блоку; тільки після цього виконується операція XOR із черговими J бітами незашифрованого тексту. Типові додатки – потокоорієнтована передача по зашумленому каналу (наприклад, супутниковий зв'язок).

Advanced Encryption Standard (AES)

Опис симетричного алгоритму шифрування AES

Поліпшений стандарт шифрування (AES) – це нова симетрична схема шифрування прийнята американським Національним інститутом Стандартів і Технології (NIST). Стандарт описаний в NIST-197 і визначає симетричний блоковий шифр, що називається Rijndael, сконструйований Дж. Даеменом і В. Риджменом. Цей шифр був відібраний в 2001 році після процесу публічної оцінки протягом більш ніж 2 роки й призначений для заміни існуючого Цифрового Стандарту Шифрування (DES). Алгоритм AES не був формально представлений на NESSIE, але був включений в оцінювання як широко використовуваний і затверджений FIPSом алгоритм для симетричного шифрування.

Опис примітива. Блоковий шифр Rijndael ліг у основу AES. Він був спочатку створений для обробки широкого спектра розмірів блоків і довжин ключів. Однак тільки три певні комбінації розмірів блоків і довжин ключів підтримуються стандартом FIPS. Ці три варіанти **AES, AES-128, AES-192 і AES-256**, мають фіксований розмір блоку в 128 біти і приймають ключі довжиною **128, 192 і 256** бітів відповідно.

Блоковий шифр AES – це підстановочно-перестановочна система (Substitution Permutation Network – SPN), що включає **10, 12 або 14** циклів (раундів) залежно від довжини ключа. Шифр бере 128-бітний блок відкритого тексту P і шифрує його в 128-бітний блок шифрованого тексту C

відповідно до секретного ключа K. Всі операції під час цього ітеративного процесу виконуються над масивом байт **4×4**, які розглядаються як елементи кінцевого поля GF (2^8).

Один цикл (раунд) включає:

– нелінійну підстановку байтів (SubBytes);

– перетворення зсуву у рядках (ShiftRows);

– перетворення змішування колонок (MixColumns) (у 10-му раунді пропускається);

– додаток ключа циклу (AddRoundKey). Ключі циклу, кожний зі 128 біт, походять від оригінального 128-бітового, 192-бітового або 256-бітового секретного ключа за допомогою процедури розширення окремого ключа.

Зворотний процес, що дозволяє відкрити зашифрований відкритий текст для даного секретного ключа, дуже схожий з алгоритмом шифрування, але різні перетворення злегка модифіковані.

При цьому, вихідний алгоритм Rijndael допускає довжину ключа й розмір блоку від 128 до 256 біт із кроком в 32 біта. Для позначення обраних довжин input, State і Cipher Key у байтах використовується нотація ***Nb* = 4** для input і State, ***Nk* = 4, 6, 8** для Cipher Key відповідно для різних довжин ключів.

Визначення й допоміжні процедури

Визначення:

– Block – послідовність біт, з яких складається input, output, State і Round Key. Також під Block можна розуміти послідовність байт.

– Cipher Key – секретний, криптографічний ключ, що використовується Key Expansion процедурою, щоб зробити набір ключів для раундів (Round Keys); може бути представлений як прямокутний масив байтів, що має чотири рядки й *Nk* колонок.

– Ciphertext – вихідні дані алгоритму шифрування, з метою збереження конфіденційності.

– Key Expansion – процедура використовувана для генерації Round Keys з Cipher Key.

– Round Key – Round Keys виходять із Cipher Key використовуючи процедуру Key Expansion. Вони застосовуються до State при шифруванні й розшифруванні.

– State – проміжний результат шифрування, з метою збереження конфіденційності, що може бути представлений як прямокутний масив байтів що має 4 рядки й *Nb* колонок.

– S-box – нелінійна таблиця замін, що використовується в декількох трансформаціях заміни байт і в процедурі Key Expansion для взаємнооднозначної заміни значення байта.

– *Nb* – число стовпців (32-ух бітних слів), що становлять *State*. Для, AES *Nb* = 4.

– *Nk* – число 32-ух бітних слів, що становлять шифроключ. Для AES, *Nk* = 4,6, або 8.

– *Nr* – число раундів, що є функцією *Nk* і *Nb*. Для AES, *Nr* = 10, 12, 14.

– Rcon[] – масив, що складається з біт 32-х розрядного слова і є постійним для даного раунду.

Допоміжні процедури:

– AddRoundKey () – трансформація при шифруванні й зворотному шифруванні, при якуму Round Key XOR *c* State. Довжина RoundKey дорівнює розміру State (ті, якщо *Nb* = 4, то довжина RoundKey дорівнює 128 біт або 16 байт).

– InvMixColumns () – трансформація при розшифруванні яка є зворотною стосовно MixColumns ().

– InvShiftRows () – трансформація при розшифруванні яка є зворотною стосовно ShiftRows ().

– InvSubBytes () – трансформація при розшифруванні яка є зворотною стосовно SubBytes ().

– MixColumns () – трансформація при шифруванні яка бере всі стовпці State і змішує їх дані (незалежно друг від друга), щоб одержати нові стовпці.

– RotWord () – функція, що використовується в процедурі Key Expansion, що бере 4-х байтне слово й робить над ним циклічну перестановку.

– ShiftRows () – трансформації при шифруванні, які обробляють State, циклічно зміщаючи останні три рядки State на різні величини.

– SubBytes () – трансформації при шифруванні які обробляють State використовуючи нелінійну таблицю заміщення байтів (S-box), застосовуючи її незалежно до кожного байта State.

– SubWord () – функція, використовувана в процедурі Key Expansion, що бере на вході чотирьохбайтне слово й застосовуючи S-box до кожного із чотирьох байтів видає вихідне слово.

На початку шифрування, з метою збереження конфіденційності, input копіюється в масив State за правилом:

$$s[r,c] = in[r + 4c],$$

для $0 \le r < 4$ й $0 \le c < Nb$.

Після цього до State застосовується процедура AddRoundKey () і потім State проходить через процедуру трансформації (раунд) **10, 12,** або **14** разів (залежно від довжини ключа), при цьому треба врахувати, що останній раунд трохи відрізняється від попередніх. У підсумку, після завершення останнього раунду трансформації, State копіюється в output за правилом:

$out[r + 4c] = s[r,c]$,

для $0 \le r < 4$ й $0 \le c < Nb$.

Операція шифрування, з метою збереження конфіденційності, описана в псевдокоді. Окремі трансформації SubBytes (), ShiftRows (), MixColumns (), і AddRoundKey () – обробляють State. Масив *w*[] – містить key schedule.

Функція шифрування в псевдокоді

```
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
```

SubBytes ()

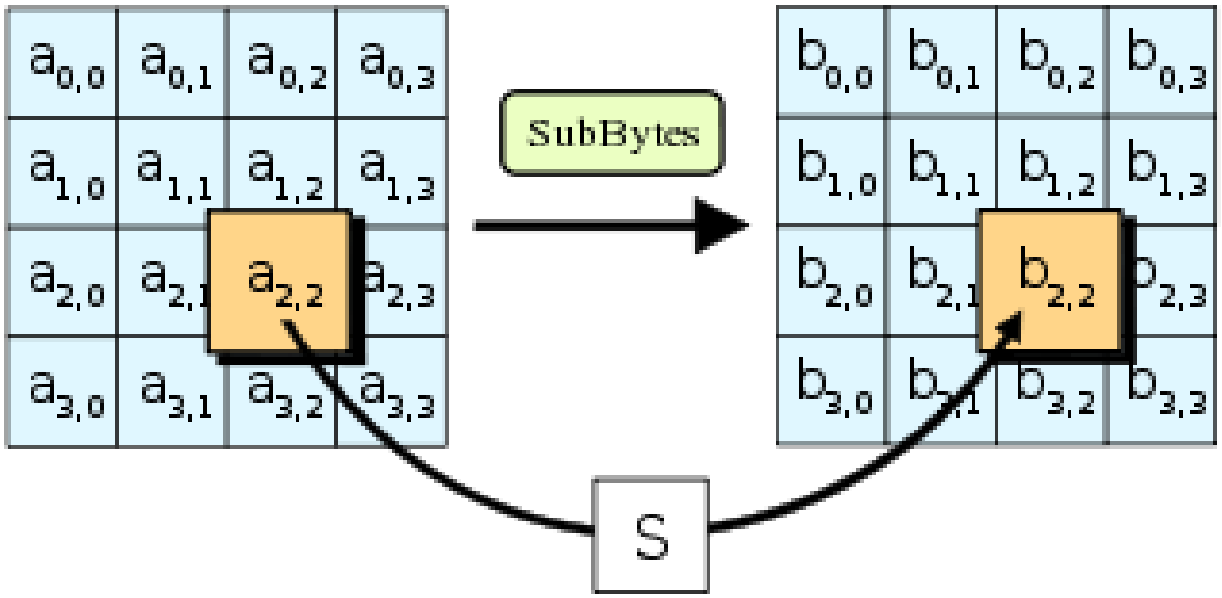


Рисунок 21.8 – Опис процедури SubBytes ()

У процедурі SubBytes, кожний байт в state заміняється відповідним елементом у фіксованої 8-бітній таблиці пошуку, *S*; $b_{ij} = S(a_{ij})$.

Процедура SubBytes () обробляє кожний байт стану, незалежно роблячи нелінійну заміну байтів використовуючи таблицю замін (S-box). Така операція забезпечує нелінійність алгоритму шифрування, з метою збереження конфіденційності. Побудова S-box складається із двох кроків. По-перше, виробляється узяття оберненого числа в GF$\{2^8\}$. По-друге, до кожного байта *b* з яких складається S-box застосовується наступна операція:

$$b_i' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i,$$

де:

– $0 \leq i < 8$;

– b_i є *i-ий* біт *b*,

– c_i – *i-ий* біт *c* = {63} або {01100011}.

Таким чином, забезпечується захист від атак, заснованих на простих алгебраїчних властивостях.

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

S-box можна відобразити таблицею простої підстановки:

S-box																
\	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

Наприклад, на вході **19** на виході отримаємо **d4**.

Фактично це звичайний шифр простої підстановки.

ShiftRows ()

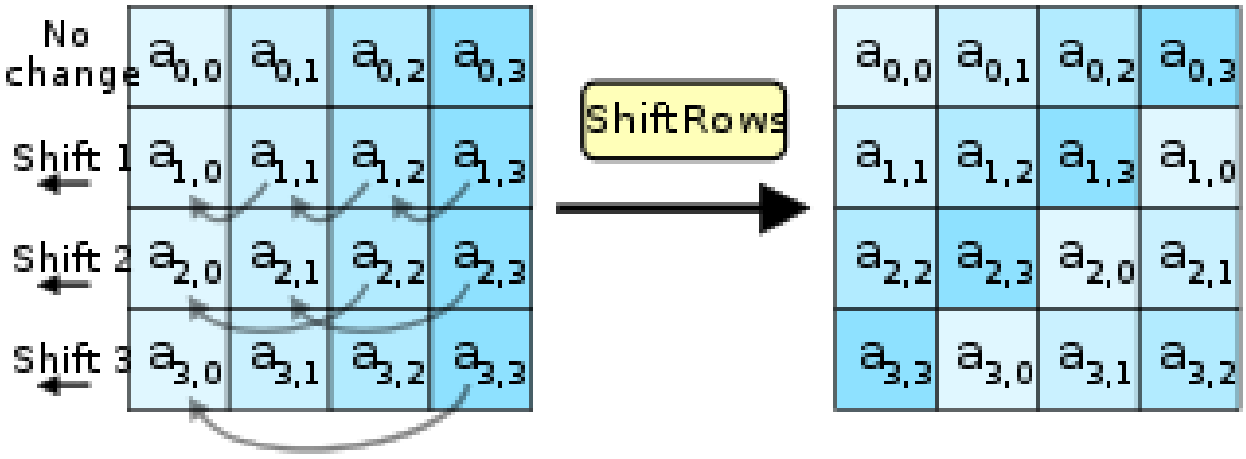


Рисунок 21.9 – Опис процедури ShiftRows ()

У процедурі ShiftRows, байти в кожному рядку state циклічно зрушуються вліво. Розмір зсуву байтів кожного рядка залежить від її номера.

ShiftRows працює з рядками State. При цій трансформації рядка стани циклічно зрушуються на *r* байт по горизонталі, залежно від номера рядка. Для нульового рядка $r = 0$, для першого рядка $r = 1$ і т.д.. У такий спосіб кожна колонка вихідного стану після застосування процедури ShiftRows складається з байтів з кожної колонки початкового стану. Для алгоритму Rijndael паттерн зсуву рядків для 128 і 192-х бітних рядків однаковий. Однак для блоку розміром 256 біт відрізняється від попередніх тим, що 2, 3, і 4-і рядки зміщаються на 1, 3, і 4 байти відповідно.

MixColumns ()

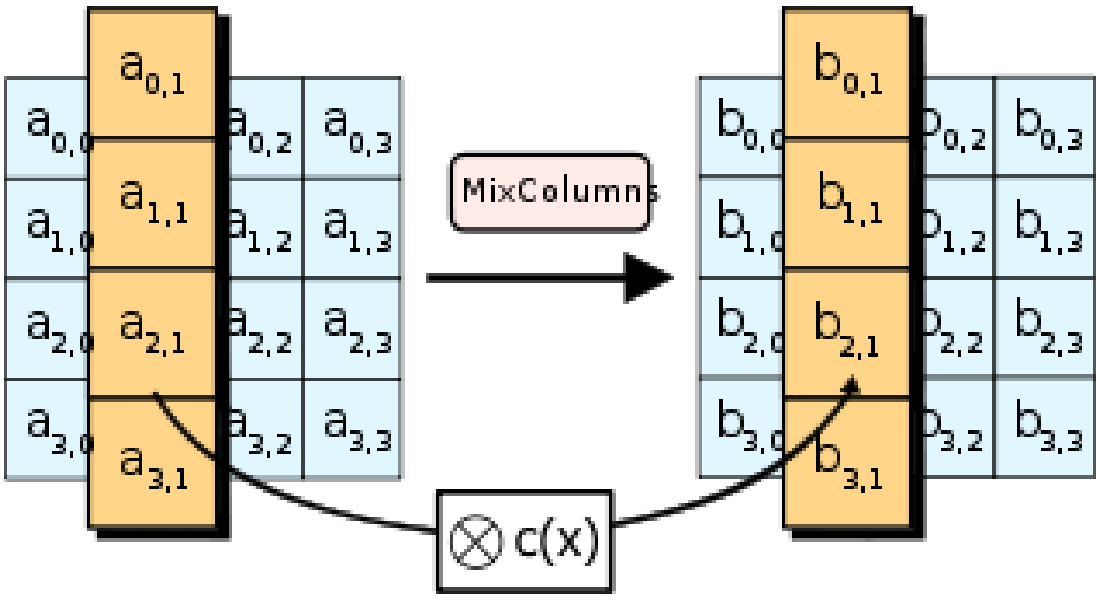


Рисунок 21.10 – Опис процедури MixColumns ()

У процедурі MixColumns, кожна колонка стану перемножується з фіксованим багаточленом *c (x)*.

У процедурі MixColumns, чотири байти кожної колонки State змішуються використовуючи для цього оборотну лінійну трансформацію. MixColumns обробляє стану по колонках, трактуючи кожну з них як поліном четвертого ступеня. Над цими поліномами виробляється множення в $GF(2^8)$ за модулем $x^4 + 1$ на фіксований багаточлен $c(x) = 3x^3 + x^2 + x + 2$. Разом з ShiftRows, MixColumns вносить дифузію в шифр.

AddRoundKey ()

У процедурі AddRoundKey, кожний байт стану поєднується з RoundKey використовуючи XOR operation (⊕).

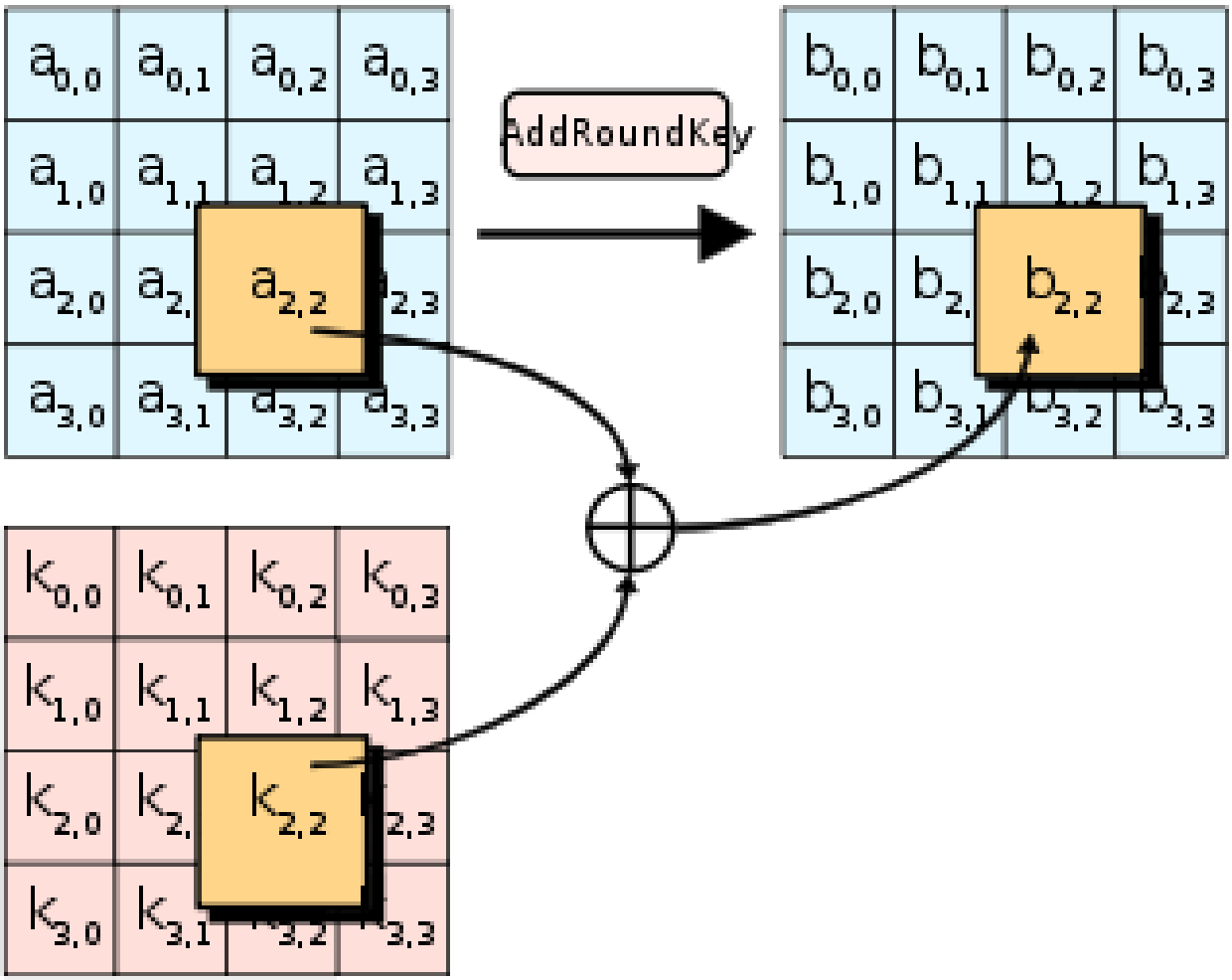


Рисунок 21.11 – Опис процедури AddRoundKey ()

У процедурі AddRoundKey, RoundKey кожного раунду поєднується з State. Для кожного раунду Roundkey виходить із CipherKey використовуючи процедуру KeyExpansion; кожний RoundKey такого ж розміру, що й State. Процедура робить побітовий XOR кожного байта State з кожним байтом RoundKey.

Опис симетричного алгоритму шифрування ДСТУ 7624:2014 (Калина) згідно з ДСТУ ISO/IEC 10116:2019

«Калина» (англ. Kalyna) – блочний симетричний шифр описаний у національному стандарті України ДСТУ 7624:2014 «Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення». Стандарт набрав чинності з 1 липня 2015 року наказом Мінекономрозвитку від 2 грудня 2014 року №1484. Текст стандарту у вільному доступі.

Стандарт ДСТУ 7624:2014 розроблено у співпраці Держспецзв'язку та провідних українських науковців і враховує досвід та результати проведення міжнародних і відкритого національного конкурсу криптографічних алгоритмів. Він призначений для поступової заміни міждержавного стандарту ДСТУ ГОСТ 28147:2009.

Згідно чинного наказу Мінцифри від 30 вересня 2020 року №140/614 після 1 січня 2022 року разом з функцією ґешування Купина є обов'язковим для використання при накладанні та перевірці електронного цифрового підпису за ДСТУ 4145-2002 замість криптографічного перетворення за ДСТУ ГОСТ 28147:2009.

Розмір блоку може складати 128, 256, 512 біт. (відповідно 10, 14, 30 раундів шифрування).

Використовує 8 S-боксів 8*8 біт. (8*256 = 2048 байт).

Основні відмінності “Калини” від Rijndael (AES):

– збільшена кількість циклів шифрування (запас стійкості);

– використання додавання за модулем 2^{32} і за модулем 2 для введення ключової інформації (захист від алгебраїчних атак, лінійного та диференціального криптоаналізів, інтерполяційної атаки тощо);

– використання 8 блоків нелінійного перетворення (S-блоків) замість одного (додатковий захист від алгебраїчних атак, поліпшення властивостей розсіювання шифру – поліпшені статистичні властивості, відповідно більш високий рівень стійкості до диференціального та лінійного криптоаналізів тощо);

– використання випадково сформованих S-блоків, відібраних за критеріями стійкості до диференціального, лінійного криптоаналізів та степені нелінійності булевих функцій (на відміну від S-блоку Rijndael/Camellia та інших шифрів, що використовують звернення в полі та, відповідно, квадратичні залежності між входом і виходом – захист від алгебраїчних атак);

– принципово нова схема створення підключів (захист від всіх відомих атак на схеми створення підключів;

– досить висока продуктивність;

– високе відновлення майстер-ключа за окремим підключем – додатковий захист від атак, що виконують відновлення підключів).

Всі поліпшення спрямовані на збільшення стійкості та перекриття потенційних вразливостей відносно Rijndael, що виявлені в останні роки.

Режими роботи та їх позначення

У стандарті описані наступні режими роботи криптографічного алгоритму, їх позначення та послуги безпеки, які забезпечує відповідний режим

Таблиця 17.1 – Режими роботи алгоритму

#	**Назва режиму**	**Позначення**	**Послуга безпеки**
1	Проста заміна (базове перетворення)	ECB	Конфіденційність
2	Гамування	CTR	Конфіденційність
3	Гамування зі зворотним зв'язком за шифротекстом	CFB	Конфіденційність
4	Вироблення імітовставки	CMAC	Цілісність
5	Зчеплення шифроблоків	CBC	Конфіденційність
6	Гамування зі зворотним зв'язком за шифрогамою	OFB	Конфіденційність
7	Вибіркове гамування із прискореним виробленням імітовставки	GCM, GMAC	Конфіденційність і цілісність (GCM), тільки цілісність (GMAC)
8	Вироблення імітовставки і гамування	CCM	Цілісність і конфіденційність
9	Індексована заміна	XTS	Конфіденційність
10	Захист ключових даних	KW	Конфіденційність і цілісність

Режим роботи криптографічного алгоритму, позначають так: **«Калина-I/k-позначення режиму-параметри режиму»** (для деяких режимів параметрів немає), де I – розмір блока базового перетворення, k – довжина ключа. Наприклад, **«Калина-256/512-CCM-32,128»** визначає використання базового перетворення з розміром блока 256 бітів, довжиною ключа 512 бітів, застосування у режимі вироблення імітовставки і гамування, довжина конфіденційної (та відкритої) частини повідомлення завжди менше ніж 232 байтів, довжина імітовставки дорівнює 128 бітів. Режим простої заміни збігається з базовим перетворенням, тому крім позначення **«Калина-I/k-ECB»** можна використовувати позначення **«Калина-I/k»**.

Загальна структура алгоритму

Загальна структура – SPN, square-type, байт-орієнтований шифр. Структура алгоритму аналогічна структурі Rijndael, забезпечує гарне розсіювання та перемішування (відмінні криптографічні та статистичні властивості, висока продуктивність).

Використовуються цикли перетворення 2-х типів: із введенням ключа за модулем 2 і за модулем 2^{32}, що збільшує нелінійність шифру, вводить додаткові залежності між результуючими значеннями.

Значно збільшує стійкість до алгебраїчних атак, диференціального та лінійного криптоаналізів тощо.

Таблиці підстановки (вузли нелінійного перетворення)

Застосування випадково сформованих таблиць підстановок, відібраних за критеріями стійкості до диференціального, лінійного криптоаналізів та ступеня нелінійності булевих функцій.

S-блоки Rijndael/Camellia та інших шифрів використовують перетворення в полі. Такий S-блок забезпечує найкращі показники з погляду диференційного та лінійного криптоаналізів (ДК/ЛК), але підстановка (S-блок) має яскраво виражену математичну структуру та просте алгебраїчне подання. Відповідно, досить серйозну загрозу наявності вразливостей шифру до алгебраїчних атак.

Маючи багаторазовий запас стійкості до ДК/ЛК (як в Rijndael, так і в "Калини"), можливе використання S-блоків з небагато гіршими показниками S-блоку, ніж в Rijndael, але в той же час позбутися від явної математичної структури, що дозволяє побудувати квадратичні залежності між входом і виходом.

Використання випадкових S-блоків дозволяє позбутися від таких залежностей (детермінованих), і перейти до ймовірнісних рівнянь опису підстановки в алгебраїчних атаках.

Застосування 8 підстановок додатково знижує можливість побудови та ймовірність знаходження правильного рішення для такої системи рівнянь.

Крім того, використання 8 S-блоків замість одного – поліпшення статистичних властивостей, відповідно більш високий рівень стійкості до диференціального та лінійного криптоаналізів тощо.

У цілому до формування S-блоків шифру "Калина" були задані такі вимоги:

– випадкова генерація, що забезпечує мінімізацію ймовірності одержання строгих математичних залежностей між вхідними та вихідними бітами;

– обмеження максимального значення ймовірність проходження різниці через підстановку (для "Калини" – 2^{-5}, для Rijndael – теоретично досяжний мінімум – 2^{-6});

– обмеження максимального значення ймовірності лінійної апроксимації підстановки (для "Калини" – 2^{-5}, для Rijndael – теоретично досяжний мінімум – 2^{-3});

– нелінійний порядок підстановки (для "Калини" теоретично досяжний максимум).

Можливе використання довільних S-блоків (відмінних від наведених у специфікації), що задовольняють даним вимогам, при цьому всі криптографічні властивості та стійкість шифру зберігаються.

Крім того, допускається використання S-блоків як додатково встановлюваного секретного параметра.

Блок лінійного перетворення

Як блок лінійного перетворення використовується добре перевірене МДР-перетворення (MixColumns), що дає найкращі властивості розсіювання (поширення різниці).

Завдяки використанню 64-бітного МДР-коду, забезпечується повна

залежність кожного біта від входу вже на 2-х циклах шифрування, незалежно від розміру блоку. МДР-код – це лінійний код, для якого виконується рівність: k = n – d + 1. Відомим представником таких кодів, є коди Ріда –Соломона.

Характеристики кращі, ніж в Rijndael 256/256, де потрібна більша кількість циклів для поширення різниці на весь блок.

Недолік – менш ефективна реалізація на 32-бітових процесорах, що поки використовуються.

Проте на 64-бітових процесорах, що одержують все більше поширення, показники продуктивності близькі до Rijndael.

Розширення ключа

Недоліки схеми створення підключів Rijndael:

– можливість відновлення майстер-ключа за одним підключем (!);

– досить прості залежності між підключами (уразливість до атак на зв'язаних ключах);

– перший з підключів – майстер-ключ (!);

– слабкий вплив змін бітів майстер-ключа на біти перших підключів (задовільні результати – тільки 5-7 підключів)

– використання в схемі розгортання іншої конструкції, відмінної від циклової функції;

– різна складність генерації послідовності підключів для шифрування та дешифрування.

У зв'язку з наявністю істотних недоліків було прийнято використовувати **принципово нову схему розгортання ключів**. Вона розроблена на основі таких вимог:

– нелінійна залежність кожного біта кожного підключа від кожного біта майстер-ключа, відповідно забезпечення всіх необхідних лавинних властивостей і відсутність "проміжних точок";

– забезпечення стійкості до всіх відомих атак на схеми створення підключів;

– відсутність слабких ключів, на яких може відбутися погіршення криптографічних властивостей шифру;

– неможливість (висока обчислювальна складність) відновлення майстер-ключа за одним або декількома підключами;

– простота реалізації, використання циклового перетворення шифру;

– обчислювальна складність генерації всіх підключів не перевищує складності однієї операції шифрування;

– можливість генерації підключів у будь-якому порядку (як для шифрування, так і для дешифрування).

Вимоги до констант C^j_{len}:

– різні константи для різних ключових станів (видалення симетрії між циклами);

– відмінності 64-бітних блоків, що попадають на MixColumn, у межах

однієї константи (видалення симетрії перетворення, що шифрує, у межах циклу).

Можливе використання довільних значень, що задовольняють цим вимогам, при цьому всі криптографічні властивості та стійкість шифру зберігаються. У специфікації наведені найбільш очевидні значення.

Режими роботи симетричного блокового шифру

На основі аналізу встановлено, що доцільно використовувати стандартні режими роботи, які визначені в NIST SP 800-38A (режими ГОСТ 28147-89, крім створення імітовставки, є підмножиною цих режимів).

Ці режими використовуються більше 20 років (специфікація DES modes of operation), існують рекомендації з їх вибору для конкретних умов застосування, і використання цих режимів забезпечує високий рівень захисту переданих повідомлень, властивості режимів добре досліджені.

Через недостатнє дослідження на даний момент вважаємо недоцільним використання режиму, що одночасно забезпечує конфіденційність і цілісність повідомлень.

Режим забезпечення цілісності

Недолік стандартного CBC: неможливість обробки повідомлення, якщо його довжина не кратна розміру блоку шифру (потрібне доповнення повідомлення, причому імітовставка доповненого повідомлення та співпадаючого з ним недоповненого, такої ж довжини, залишається деякий ризик нав'язування).

Вимоги до режиму забезпечення цілісності:

– відповідність всім вимогам стійкості, пропонованих до ґеш-функцій;

– можливість обробки повідомлень довільної довжини;

– використання одного ключа шифрування;

– забезпечення стійкості.

Більшість відомих альтернативних схем використовують 2 або 3 ключі шифрування, що є істотним недоліком.

З відомих режимів один ключ використовується в OMAC, при цьому забезпечується захист механізму доповнення повідомлення, чия довжина не кратна розміру блоку.

Краща атака на OMAC є атакою з обраними відкритими текстами, коли криптоаналітик виконує статистичний пошук колізії ($2^{n/2}$ шифрувань, де n – розмір блоку), тобто фактично теоретичний максимум стійкості.

Схема є стійкою, має низьку обчислювальну складність, проста в реалізації та зручна у використанні.

Детальний опис алгоритму «Калина»

Допустимі комбінації розміру блока та довжини ключа шифрування наведено у таблиці 17.2.

Таблиця 21.2 – Параметри алгоритму “Калина”

№ з/п	Розмір блоку (l)	Довжина ключа (k)	Кількість ітерацій перетворення (t)	Кількість стовпців в матриці (c)
1	128	128	10	2
2		256	14	
3	256	256	14	4
4		512	18	
5	512	512	18	8

Алгоритм шифрування є процедурою, що складається з попереднього і прикінцевого забілювання та ітеративного циклового перетворення. На вхід кожного циклового перетворення подається поточний стан, а також необхідна кількість ключової інформації (цикловий ключ). Відкритий текст копіюється в поточний стан перед початком зашифрування, а після його завершення в поточному стані знаходиться шифртекст. Кількість циклів шифрування (N r) залежить від довжини ключа; її наведено у таблиці 17.2. На початку та у кінці процедури зашифрування/розшифрування виконуються додаткові операції забілювання.

У стандарті визначені обсяги повідомлень, після обробки яких потрібна обов’язкова зміна ключа. Крім того, наводяться рекомендації розробникам, що звертають увагу на необхідність запобігання атак з використанням особливостей реалізації засобів шифрування. Зокрема, враховані особливості, які дозволяли організацію атак BEAST і CRIME/BREACH в протоколах SSL/TLS, повторний прийом повідомлень, відновлення конфіденційних параметрів на основі залежності часу шифрування від даних (через промахи кешу при табличній реалізації) та інших.

До вхідних та вихідних даних алгоритму “Калина” належать відкритий текст та шифртекст відповідно. Ці параметри представляються у вигляді рядків заданої довжини $8 \times Nb$ байт ($64 \times Nb$ біт). Додатковим вхідним параметром є ключ, розмір якого дорівняє $8 \times Nk$ байт ($64\times Nk$ біт). Байтовий рядок довжиною $n = 8 \times Nb$ байт представляється у наступній формі: $B_0\ B_1\ B_2\ \ldots\ B_{n-1}$.

До початку зашифрування відкритий текст копіюється в поточний стан шифру. Після завершення процедури зашифрування шифртекст копіюється з поточного стану.

Відкритий текст представлено байтовим рядком $in_0, in_1, \ldots in_{31}$. Отриманий шифртекст представлено як послідовність байт $out_0, out_1, \ldots out_{31}$.

Заповнення початкового стану шифру перед початком зашифрування та після його закінчення для блоку 128 біт представлено на на рис.17.12.

Вхідна послідовність		→	Поточний стан		→	Вихідна послідовність	
in_0	in_8		$s_{0,0}$	$s_{0,1}$		out_0	out_8
in_1	in_9		$s_{1,0}$	$s_{1,1}$		out_1	out_9
in_2	in_{10}		$s_{2,0}$	$s_{2,1}$		out_2	out_{10}
in_3	in_{11}		$s_{3,0}$	$s_{3,1}$		out_3	out_{11}
in_4	in_{12}		$s_{4,0}$	$s_{4,1}$		out_4	out_{12}
in_5	in_{13}		$s_{5,0}$	$s_{5,1}$		out_5	out_{13}
in_6	in_{14}		$s_{6,0}$	$s_{6,1}$		out_6	out_{14}
in_7	in_{15}		$s_{7,0}$	$s_{7,1}$		out_7	out_{15}

Рисунок 21.12 – Заповнення початкового стану для довжини блоку 128 біт

Аналогічно, при розшифруванні шифртекст позначається байтовою послідовністю in_0, in_1,… $in_{31;}$ отриманий відкритий текст представлено послідовністю байт $out_{0,}$ out_1, …out_{31}; заповнення відповідає рис.17.12.

Операція лінійного розсіювання (перемішування в стовбці) алгоритму “Калина” використовує поліноміальне представлення байт в полі GF (2^8), сформованому незвідним поліномом. Для алгоритму шифрування “Калина” у якості незвідного полінома використовується $f(x) = x^8 + x^4 + x^3 + x^2 + 1$.

Зашифрування у режимі простої заміни передбачає те, що на вхід процедури подається відкритий текст та циклові ключі. На початку зашифрування відкритий текст представляється у вигляді блоку даних, що описує поточний стан шифру. Після закінчення зашифрування отриманий шифртекст формується у вигляді байтової послідовності.

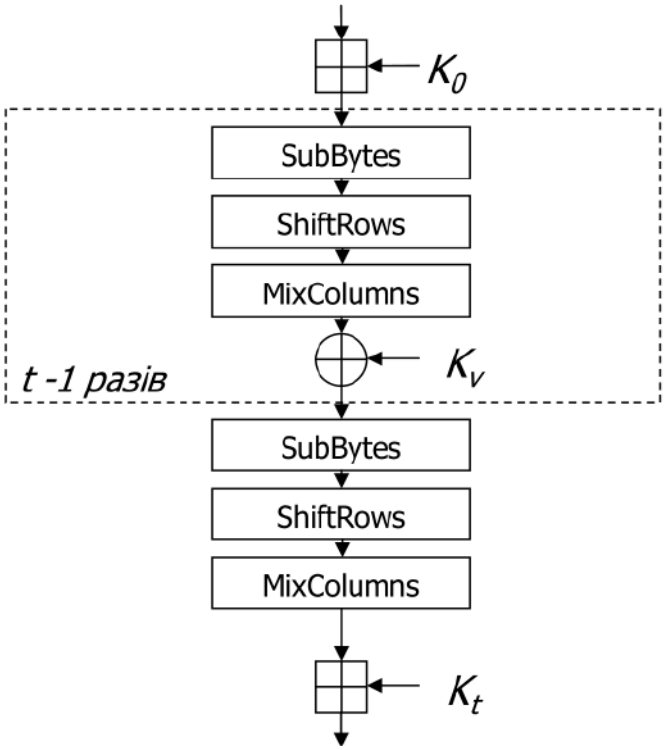


Рисунок 21.13 – Алгоритм шифрування «Калина»

При здійсненні перетворення XORRoundKey (функція $K_l^{(Kv)}$ в ДСТУ 7624:2014) виконується побітове складання за модулем 2 (XOR) поточного стану та циклового ключа. Після виконання операції результат записується на місце першого аргументу.

Представлення циклового ключа як матриці віповідного розміру є аналогічним представленню відкритого тексту у вигляді поточного стану шифра.

Для поточного стану $A=(a_{i,j})$ та циклового ключа $k=(k_{i,j})$ результат перетворення $B=(b_{i,j})=A \oplus k$ обчислюється за формулою $b_{i,j} = a_{i,j} \oplus k_{i,j}$, $0 \le i < 8$, $0 \le j < N_b$. Приклад здійснення перетворення для розміру блока 128 біт представлено на рис. 17.14.

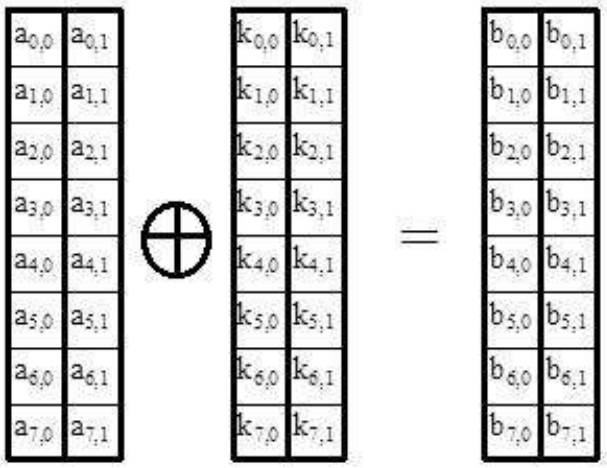


Рисунок 21.14 – Перетворення XORRoundKey для розміру блока 128 біт

При здійсненні Add64RoundKey виконується складання за модулем 2^{64} 64-бітних слів поточного стану та циклового ключа. Після виконання операції результат записується на місце першого аргументу (рис 17.15).

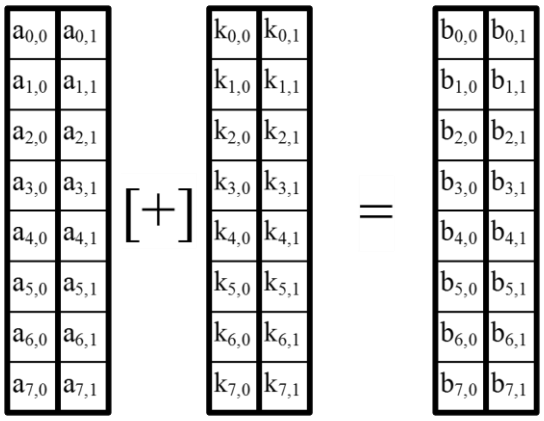


Рисунок 21.15 – Перетворення Add64RoundKey для розміру блока 128 біт

Представлення циклового ключа як матриці віповідного розміру є аналогічним представленню відкритого тексту у вигляді поточного стану шифру. При розбитті поточного стану та циклового ключа на 64-бітні блоки молодшим бітом 64-бітного блоку буде молодший біт байту рядка з найменшим номером, старшим бітом 64-бітного блоку – старший біт байта рядка з найбільшим номером (формат little endian).

Перетворення Kalyna_S_boxes виконує заміну кожного байта поточного стану у відповідності з заданою таблицею підстановки. Використовуються 4 різні підстановки “байт-в-байт”, причому для байтів одного рядка поточного стану шифру використовується одна і та ж підстановка:

для байт 0-го рядка (елементи s 0,i) – підстановка S0;
для байт 1-го рядка (елементи s 1,i) – підстановка S1;
для байт 2-го рядка (елементи s 2,i) – підстановка S2;
для байт 3-го рядка (елементи s 3,i) – підстановка S3;
для байт 4-го рядка (елементи s 4,i) – підстановка S0;
для байт 5-го рядка (елементи s 5,i) – підстановка S1;
для байт 6-го рядка (елементи s 6,i) – підстановка S2;
для байт 7-го рядка (елементи s 7,i) – підстановка S12.

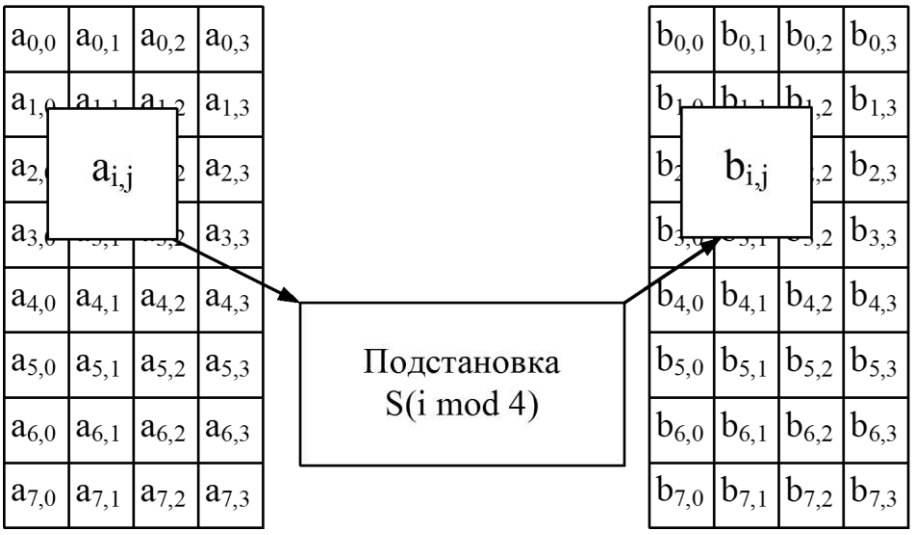


Рисунок 21.16 – Приклад підстановки байта для розміра блоку 256 біт

Заміна одного байту полягає у виборі з таблиці підстановки нового значення за адресою із зсувом, що задає поточне значення байту. Нове вибране значення і є результатом здійснення підстановки для одного байту.

Приклад підстановки для таблиці S0 байта з шістнадцятковим значенням 5A наведено у табл. 21.17.

Таблиця 21.3 – Приклад підстановки для таблиці S0

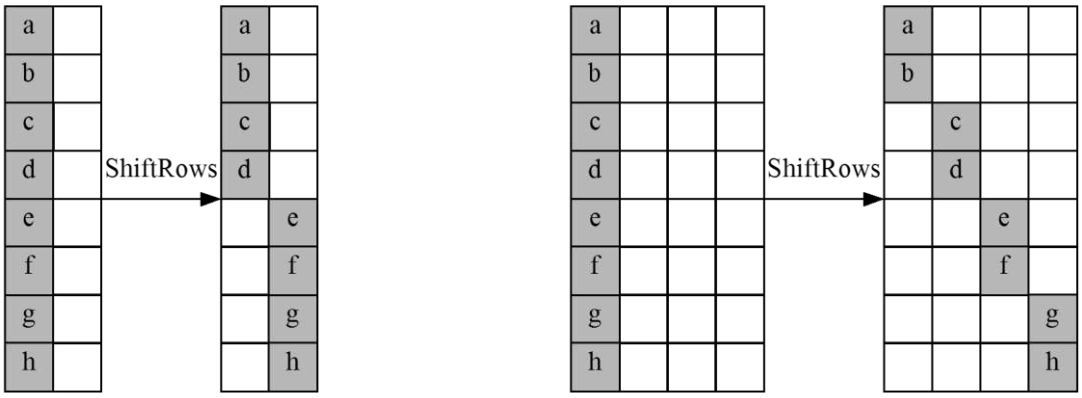
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	A8	43	5F	06	6B	75	6C	59	71	DF	87	95	17	F0	D8	09
1	6D	F3	1D	CB	C9	4D	2C	AF	79	E0	97	FD	6F	4B	45	39
2	3E	DD	A3	4F	B4	B6	9A	0E	1F	BF	15	E1	49	D2	93	C6
3	92	72	9E	61	D1	63	FA	EE	F4	19	D5	AD	58	A4	BB	A1
4	DC	F2	83	37	42	E4	7A	32	9C	CC	AB	4A	8F	6E	04	27
5	2E	E7	E2	5A	96	16	23	2B	C2	65	66	0F	BC	A9	47	41
6	34	48	FC	B7	6A	88	A5	53	86	F9	5B	DB	38	7B	C3	1E
7	22	33	24	28	36	C7	B2	3B	8E	77	BA	F5	14	9F	08	55
8	9B	4C	FE	60	5C	DA	18	46	CD	7D	21	B0	3F	1B	89	FF
9	EB	84	69	3A	9D	D7	D3	70	67	40	B5	DE	5D	30	91	B1
A	78	11	01	E5	00	68	98	A0	C5	02	A6	74	2D	0B	A2	76
B	B3	BE	CE	BD	AE	E9	8A	31	1C	EC	F1	99	94	AA	F6	26
C	2F	EF	E8	8C	35	03	D4	7F	FB	05	C1	5E	90	20	3D	82
D	F7	EA	0A	0D	7E	F8	50	1^a	C4	07	57	B8	3C	62	E3	C8
E	AC	52	64	10	D0	D9	13	0C	12	29	51	B9	CF	D6	73	8D
F	81	54	C0	ED	4E	44	A7	2^a	85	25	E6	CA	7C	8B	56	80

Старші 4 біти визначають рядок, молодші 4 біти – стовбець. Результат підстановки для значення 5A – це число в шістнадцятковому представленні 66, що знаходиться в таблиці на перетині 6-го рядка (з індексом 5) та 11-го стовбця.

Під час виконання перетворення ShiftRows (перестановка елементів τ_i в ДСТУ 7624:2014) здійснюється рівномірне розподілення байт кожного 64-бітного стовбця серед інших стовбців. Це досягається шляхом циклічного зсуву рядків стану вправо на різну кількість байт. Значення зсувів залежать від розміру блока шифрування і представлені у табл. 17.4. Рис. 17.17 пояснює порядок виконання перетворення ShiftRows для різних розмірів блоку.

Таблиця 21.4 – Значення циклічних зсувів рядків для різних розмірів блоку

Номер рядка	Значення зсуву, байти		
	Довжина блоку 128 біт	Довжина блоку 256 біт	Довжина блоку 512 біт
0	0	0	0
1	0	0	1
2	0	1	2
3	0	1	3
4	1	2	4
5	1	2	5
6	1	3	6
7	1	3	7



а) 128-бітний блок б) 256-бітний блок

Рисунок 21.17 – Порядок розподілення байт першого стовбця при здійсненні перетворення ShiftRows

Під час здійснення перетворення MixColumns (лінійне перетворення ψ_l в ДСТУ 7624:2014) виконується послідовна обробка всіх стовбців поточного стану. Кожний 8-байтний стовбець розглядається як поліном над полем GF (2^8), що складається з суми 8 одночленів (кожний байт представляється у вигляді елемента поля GF (2^8) і є коефіцієнтом перед змінною степеня, що дорівнює індексу байту у стовбці). При перетворенні виконується множення цього полінома за модулем x^8+1 на фіксований поліном C (*x*), де

$$C(x) = \{01\}x^7 + \{05\}x^6 + \{01\}x5 + \{08\}x^4 + \{06\}x^3 + \{07\}x^2 + \{04\}x + \{01\}.$$

Ця операція еквівалентна матричному множенню над GF (2^8) початкового 8-байтового вектору на фіксовану матрицю, результат зберігається у 8 байтний вектор b (див. рис. 21.18).

Порядок обчислення елементів результуючого вектору *b* пояснюється на рис. 21.19 при цьому всі операції множення на байт виконуються у полі GF (2^8). шифру.

На рис. 21.20 пояснюється порядок виконання перетворення MixColumns для поточного стану шифру.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 01 & 01 & 05 & 01 & 08 & 06 & 07 & 04 \\ 04 & 01 & 01 & 05 & 01 & 08 & 06 & 07 \\ 07 & 04 & 01 & 01 & 05 & 01 & 08 & 06 \\ 06 & 07 & 04 & 01 & 01 & 05 & 01 & 08 \\ 08 & 06 & 07 & 04 & 01 & 01 & 05 & 01 \\ 01 & 08 & 06 & 07 & 04 & 01 & 01 & 05 \\ 05 & 01 & 08 & 06 & 07 & 04 & 01 & 01 \\ 01 & 05 & 01 & 08 & 06 & 07 & 04 & 01 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}$$

Рисунок 21.18 – Матричне представлення перемішування у стовбці

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 01\cdot a_0 \oplus 01\cdot a_1 \oplus 05\cdot a_2 \oplus 01\cdot a_3 \oplus 08\cdot a_4 \oplus 06\cdot a_5 \oplus 07\cdot a_6 \oplus 04\cdot a_7 \\ 04\cdot a_0 \oplus 01\cdot a_1 \oplus 01\cdot a_2 \oplus 05\cdot a_3 \oplus 01\cdot a_4 \oplus 08\cdot a_5 \oplus 06\cdot a_6 \oplus 07\cdot a_7 \\ 07\cdot a_0 \oplus 04\cdot a_1 \oplus 01\cdot a_2 \oplus 01\cdot a_3 \oplus 05\cdot a_4 \oplus 01\cdot a_5 \oplus 08\cdot a_6 \oplus 06\cdot a_7 \\ 06\cdot a_0 \oplus 07\cdot a_1 \oplus 04\cdot a_2 \oplus 01\cdot a_3 \oplus 01\cdot a_4 \oplus 05\cdot a_5 \oplus 01\cdot a_6 \oplus 08\cdot a_7 \\ 08\cdot a_0 \oplus 06\cdot a_1 \oplus 07\cdot a_2 \oplus 04\cdot a_3 \oplus 01\cdot a_4 \oplus 01\cdot a_5 \oplus 05\cdot a_6 \oplus 01\cdot a_7 \\ 01\cdot a_0 \oplus 08\cdot a_1 \oplus 06\cdot a_2 \oplus 07\cdot a_3 \oplus 04\cdot a_4 \oplus 01\cdot a_5 \oplus 01\cdot a_6 \oplus 05\cdot a_7 \\ 05\cdot a_0 \oplus 01\cdot a_1 \oplus 08\cdot a_2 \oplus 06\cdot a_3 \oplus 07\cdot a_4 \oplus 04\cdot a_5 \oplus 01\cdot a_6 \oplus 01\cdot a_7 \\ 01\cdot a_0 \oplus 05\cdot a_1 \oplus 01\cdot a_2 \oplus 08\cdot a_3 \oplus 06\cdot a_4 \oplus 07\cdot a_5 \oplus 04\cdot a_6 \oplus 01\cdot a_7 \end{bmatrix}$$

Рисунок 21.19 – Порядок обчислення байт при виконанні перемішування

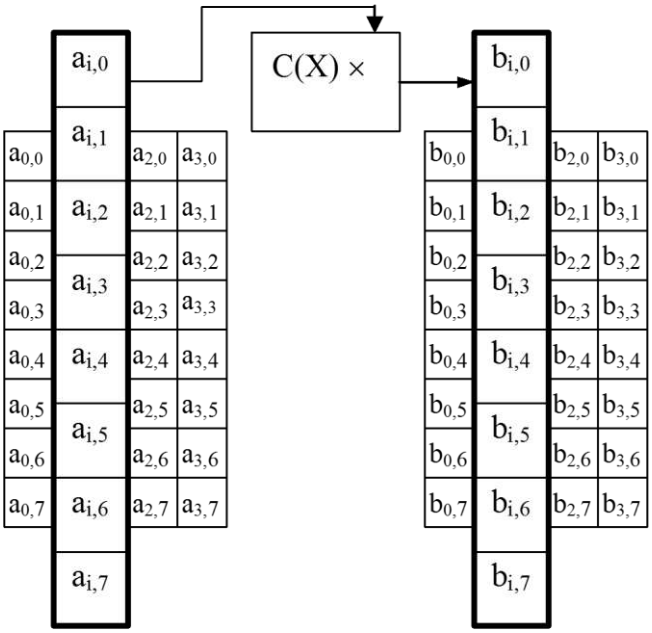


Рисунок 21.20 – Порядок виконання перетворення Mix Columns для поточного стану шифру з розміром блоку 256 біт

Процедура розшифрування є зворотною зашифруванню. На вхід подається шифртекст та циклові ключі. Після закінчення розшифрування отриманий відкритий текст формується у вигляді байтового рядка.

Операція XORRoundKey є зворотною до себе (подвійне застосування дає початкове значення), відповідно, якщо до стану $B=A \oplus k$ додати цикловий ключ k, то буде отримано початковий стан A.

При зашифруванні та розшифруванні використовується одна й та сама операція XORRoundKey.

Зворотне для Add64RoundKey перетворення Sub64RoundKey полягає в аналогічному розбитті стану та циклового ключа на 64-бітні блоки та відніманні за модулем 2^{64} від блоків стану $B = (b_i)$ відповідних блоків циклового ключа $K = (k_i)$ для отримання стану (результату).

Після виконання операції результат записується на місце першого параметра. Правила переходу від байт поточного стану до 64-бітних блоків та навпаки повністю ідентичні з перетвореннями, що виконуються під час Add64RoundKey.

ISO/IEC 10116:2017. Інформаційні технології. Методи захисту. Режими роботи n-розрядного блокового шифру

Цей документ визначає режими роботи для *n* - розрядного блочного шифру. Ці режими забезпечують методи шифрування та дешифрування даних за допомогою блокового шифру.

Це четверте видання ISO/IEC 10116 визначає п’ять режимів роботи:

– Електронна кодова книга (ECB).
– Ланцюг блоків шифру (CBC).
– Зворотній зв'язок по шифру (CFB).
– Вихідний зворотний зв'язок (OFB).
– Лічильник (CTR).

Цей документ встановлює п'ять режимів роботи для застосувань *n* - розрядного блокового шифру (наприклад, захист даних під час передачі або зберігання). Визначені режими забезпечують лише захист конфіденційності даних. Захист цілісності даних не входить до сфери застосування цього документа. Крім того, більшість режимів не захищають конфіденційність інформації про довжину повідомлення.

21.4. Потокові шифри: RC4, STRUMOK (ДСТУ 8845:2019) (довжина ключів, крипостійкість).)

RC4

RC4 (від англ. *Rivest cipher 4* або *Ron's code*), також відомий як **ARC4** або **ARCFOUR** (*alleged RC4*) – потоковий шифр, що широко застосовується в різних системах захисту інформації в комп'ютерних мережах (наприклад, в протоколах SSL і TLS, алгоритмах забезпечення безпеки бездротових мереж WEP та WPA).

Шифр розроблений компанією RSA Security і для його використання потрібна ліцензія.

Алгоритм RC4, як і будь-який потоковий шифр, будується на основі генератора псевдовипадкових бітів. На вхід генератора записується ключ, а на виході читаються псевдовипадкові біти. Довжина ключа може становити від 40 до 2048 біт. Генеровані біти мають рівномірний розподіл.

Основні **переваги шифру**:

– висока швидкість роботи;
– змінний розмір ключа.

RC4 досить вразливий, якщо:

– використовуються не випадкові чи пов'язані ключі;
– один ключовий потік використовується двічі.

Ці фактори, а також спосіб використання можуть зробити криптосистему небезпечною (наприклад, WEP).

Алгоритм шифрування RC4 застосовується в деяких широко поширених стандартах і протоколах шифрування (наприклад, WEP, WPA, SSL та TLS).

RC4 став популярним завдяки:

- простоті його апаратної та програмної реалізації;
- високої швидкості роботи алгоритму обох випадках.

У довжина ключа, рекомендована до використання у країні, дорівнює 128 бітам. Угода, укладена між «SPA» (англ. *software publishers association)* і урядом США, дозволила експортувати шифри RC4 з довжиною ключа до 40 біт. 56 бітні ключі дозволено використовувати закордонним відділенням американських компаній.

Опис алгоритму

Ядро алгоритму потокових шифрів складається з функції – генератора псевдовипадкових бітів (гами), який видає потік бітів ключа (ключовий потік, гаму, послідовність псевдовипадкових бітів).

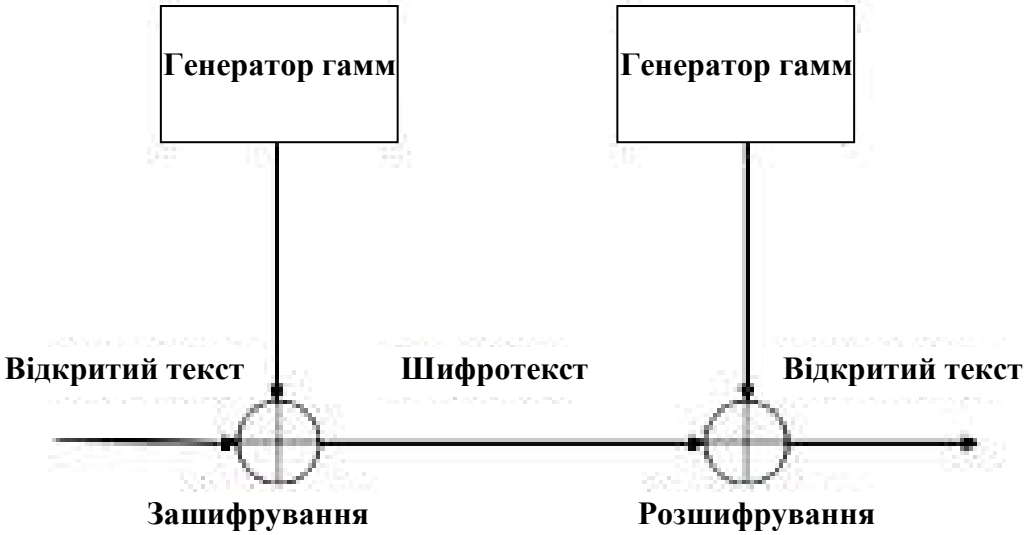


Рисунок 21.21 – Режим гамування для потокових шифрів

Алгоритм шифрування.

1. Функція генерує послідовність бітів (k_i).

2. Потім послідовність бітів за допомогою операції «додавання за модулем два» (xor) поєднується з відкритим текстом (m_i). В результаті виходить шифрограма (c_i):

$$c_i = m_i \oplus k_i$$

Алгоритм розшифрування.

1. Повторно створюється (регенерується) потік бітів ключа (ключовий потік) (k_i).

2. Потік бітів ключа складається із шифрограмою (c_i) Операцією «xor». В силу властивостей операції xor на виході виходить вихідний (незашифрований) текст (m_i):

$$m_i = c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i$$

RC4 – фактично клас алгоритмів, що визначаються розміром блоку (надалі S-блоку). Параметр n є розміром слова алгоритму і визначає довжину *S-блока*. Зазвичай $n = 8$, але з метою аналізу можна зменшити його. Проте підвищення безпеки необхідно збільшити цю величину. У алгоритмі немає протиріч збільшення розміру *S-блока*. При збільшенні n, припустимо, до 16 біт, елементів у *S-блоці* стає 65536 і відповідно час початкової ітерації буде збільшено. Однак швидкість шифрування зросте.

Внутрішній стан RC4 представляється як масиву розміром 2^n і двох лічильників. Масив відомий як *S-блок* і далі буде позначатися як S. Він завжди містить перестановку 2^n можливих значень слова. Два лічильники позначені через *i* та *j*.

Ініціалізація RC4 складається із двох частин:

1. Ініціалізація *S-блоку*.
2. Генерація псевдовипадкового слова K.

Ініціалізація *S-блоку*

Алгоритм також відомий як «key-scheduling algorithm» або «KSA». Цей алгоритм використовує ключ, який подається на вхід користувачем, збережений в Key, і має довжину L байт. Ініціалізація починається із заповнення масиву S, далі цей масив перемішується шляхом перестановок, що визначаються ключем. Так як тільки одна дія виконується над S, то повинно виконуватися твердження, що S завжди містить один набір значень, який був наданий при початковій ініціалізації (*S[i]:=i*).

for i **from** 0 **to** 255
  S[i]:= i
endfor
j:= 0
for i **from** 0 **to** 255
  j: = (j + S [i] + Key [i mod L]) mod 256 // $n = 8$; $2^8 = 256$
  поміняти місцями S[i] та S[j]
endfor

Генерація псевдовипадкового слова *K*

Ця частина алгоритму називається генератором псевдовипадкової послідовності (англ. ***p**sudo-**r**andom **g**eneration **a**lgorithm, PRGA*). Генератор ключового потоку RC4 переставляє значення, що зберігаються у S. В одному циклі RC4 визначається одне n-бітне слово K із ключового потоку. Надалі ключове слово буде складено за модулем два з вихідним текстом, який користувач хоче зашифрувати, та отримано зашифрований текст.

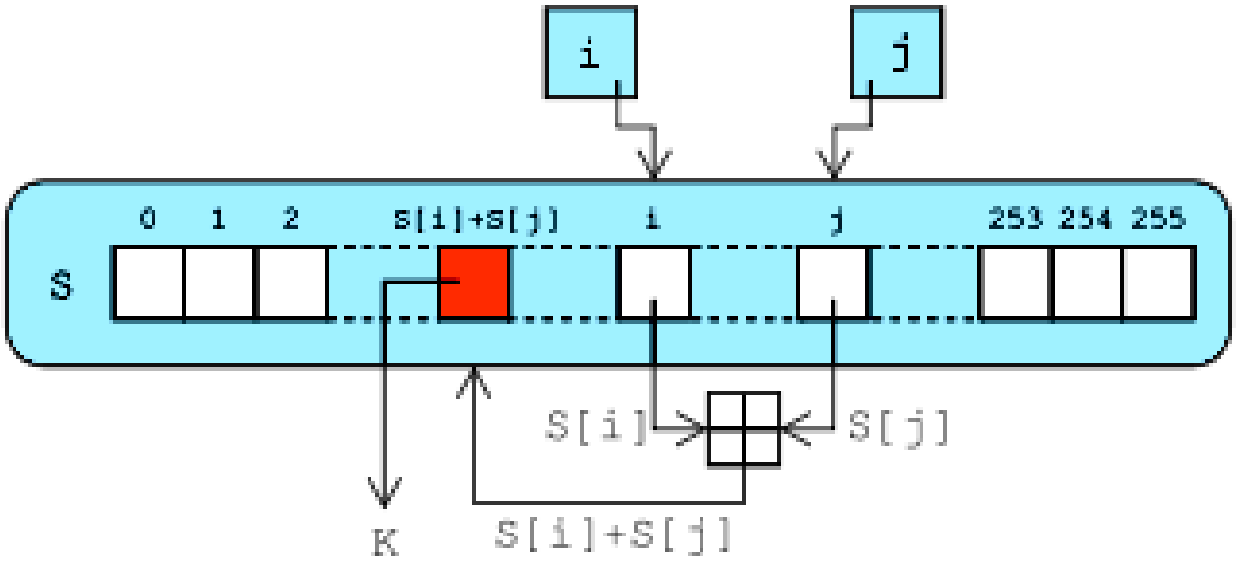


Рисунок 21.22 – Генератор ключового потоку RC4

i:= 0
j:= 0
while Цикл генерації:
  i:= (i + 1) mod 256
  j:= (j + S [i]) mod 256
  поміняти місцями S[i] та S[j]
  t:= (S[i] + S[j]) mod 256
  K:= S[t]
  згенеровано псевдовипадкове слово K (для *n* = 8 буде згенеровано один байт)
endwhile

Безпека

На відміну від сучасних шифрів (таких, як eSTREAM), RC4 не використовує nonce (від англ. *nonce* – Number that can only be used once – число, яке може бути використано один раз) поряд з ключем. Це означає, що якщо один ключ повинен використовуватися протягом тривалого часу для шифрування кількох потоків, сама криптосистема, що використовує RC4, повинна комбінувати наказ і довгостроковий ключ для отримання потокового ключа для RC4. Один з можливих виходів – генерувати **новий** ключ для RC4 за допомогою ґеш-функції від довгострокового ключа та nonce. Однак багато програм, що використовують RC4, просто конкатенують ключ і nonce. Через цей і слабкий розклад ключів, що використовується в RC4, додаток може стати вразливим. Тому він був визнаний застарілим багатьма софтверними компаніями, такими як Microsoft. Наприклад, у.NET Framework від Microsoft відсутня реалізація RC4.

Далі будуть розглянуті деякі атаки на шифр та методи захисту від них.

Дослідження Руза та відновлення ключа з перестановки

У 1995 році Андрю Руз (англ. *Andrew Roos*) експериментально поспостерігав, що перший байт ключового потоку корельований з першими трьома байтами ключа, а перші кілька байт перестановки після алгоритму розкладу ключів (англ. *KSA*) корельовані з деякою лінійною комбінацією байт. Ці усунення не були доведені до 2007 року, коли Пол, Рафі та Мейтре довели корелювання ключа та ключового потоку. Також Пол і Мейтре довели корелювання перестановки та ключа. Остання робота також використовує корелювання ключа та перестановки для того, щоб створити перший алгоритм повного відновлення ключа з останньої перестановки після KSA, не роблячи припущень про ключ і вектор ініціалізації (англ. *IV, **i**nitial **v**ector*). Цей алгоритм має постійну ймовірність успіху залежно від часу, що відповідає квадратному кореню зі складності повного перебору. Пізніше було зроблено багато робіт про відновлення ключа із внутрішнього стану RC4.

Атака Флурера, Мантіна та Шаміра (ФМШ)

У 2001 році Флурер, Мантін та Шамір опублікували роботу про вразливість ключового розкладу RC4. Вони показали, що перші байти ключового потоку серед усіх можливих ключів є невипадковими. З цих байтів можна з високою ймовірністю отримати інформацію про використовуваний шифр ключа. І якщо довготривалий ключ і nonce просто склеюються для створення ключа шифру RC4, цей довготривалий ключ може бути отриманий за допомогою аналізу досить великої кількості повідомлень, зашифрованих з використанням даного ключа. Ця вразливість та деякі пов'язані з нею ефекти були використані при зламі шифрування WEP у бездротових мережах стандарту IEEE 802.11. Це показало необхідність якнайшвидшої заміни WEP, що спричинило розробку нового стандарту безпеки бездротових мереж WPA.

Криптосистему можна зробити несприйнятливою до цієї атаки, якщо відкидати початок ключового потоку. Таким чином, модифікований алгоритм називається «RC4-drop[n]», де n кількість байтів з початку ключового потоку, які слід відкинути. Рекомендовано використовувати n = 768 консервативна оцінка становить n = 3072.

Атака базується на слабкості вектора ініціалізації. Знаючи перше псевдовипадкове слово K та m байтів вхідного ключа Key, використовуючи слабкість в алгоритмі генерації псевдовипадкового слова K, можна отримати m + 1 байт вхідного ключа. Повторюючи кроки, видобувається повний ключ. При атаці на WEP, n = 8 IV має вигляд (B; 255; N), де B – від 3 до 8, а N будь-яке число. Для визначення близько 60 варіантів *N* потрібно перехопити приблизно 4 мільйони пакетів.

Атака Кляйна

У 2005 році Андреас Кляйн представив аналіз шифру RC4, в якому він вказав на сильну корелювання ключа та ключового потоку RC4. Кляйн проаналізував атаки на першому раунді (подібні до атаки ФМШ), на другому раунді та можливі їх поліпшення. Він також запропонував деякі зміни алгоритму посилення стійкості шифру. Зокрема, він стверджує, що якщо змінити напрямок

циклу на зворотний в алгоритмі ключового розкладу, то можна зробити шифр більш стійким до атак типу ФМШ

Комбінаторна проблема

У 2001 році Аді Шамір та Іцхак Мантін першими поставили комбінаторну проблему, пов'язану з кількістю всіляких вхідних та вихідних даних шифру RC4. Якщо з різних 256 елементів внутрішнього стану шифру відомо хелементів зі стану ($x \leq 256$), то, якщо припустити, що інші елементи нульові, максимальна кількість елементів, які можуть бути отримані детермінованим алгоритмом за наступні 256 раундів, також дорівнює x. У 2004 році це припущення було доведено Сорадюті Полом (англ. *Souradyuti Paul*) і Бартом Пренелем (англ. *Bart Preneel*).

Атака Ванхофа та Пісенса (2015)

Влітку 2015 року Меті Ванхоф (Mathy Vanhoef) і Франк Пісенс (Frank Piessens) з університету Левена в Бельгії продемонстрували реальну атаку на протокол TLS, який використовує RC4 для шифрування даних. Ідея злому базується на принципі MITM. Вбудовавшись у канал передачі, атакуюча сторона генерує серверу велику кількість запитів, змушуючи їх у відповідь повертати куки, зашифровані одним і тим самим ключем. Маючи у розпорядженні близько $9x2^{27}$~2^{30} пар {відкритий текст, шифротекст}, атакуюча сторона отримала можливість на основі статистичних методів Флюрер-Макгрю та ABSAB з ймовірністю 94% відновити ключ і, отже, зашифровані куки. Практичні часові витрати становили близько 52 годин, верхня ж оцінка потрібного часу на момент демонстрації склала близько 72 годин.

STRUMOK (ДСТУ 8845:2019)

ДСТУ 8845:2019 Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного потокового перетворення

Зміст алгоритму

Сфера застосування

Цей стандарт установлює криптографічний алгоритм симетричного потокового перетворення для забезпечення конфіденційності та цілісності (як додаткової послуги) інформації під час її оброблення.

У стандарті описано алгоритм симетричного потокового криптографічного перетворення, який використовує ключовий потік для шифрування відкритого тексту побітово чи поблоково. Ключовий потік генерують лише із секретного ключа та вектора ініціалізації (синхропосилки), отже, стандарт визначає синхронний потоковий шифр (за класифікацією згідно з ДСТУ ISO/IEC 18033-4).

Стандарт використовують під час розроблення засобів криптографічного захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах, а також у разі модернізації наявних систем для заміни потокових режимів згідно з ДСТУ ГОСТ 28147.

Для забезпечення конфіденційності та цілісності послідовностей двійкових символів можна застосовувати цей стандарт сумісно з ДСТУ ISO/IEC 18033-4 та ДСТУ 7564.

Нормативні посилання

У цьому стандарті є посилання на такі стандарти:

– ДСТУ 7564:2014 Інформаційні технології. Криптографічний захист інформації. Функція ґешування

– ДСТУ 7624:2014 Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення

– ДСТУ ГОСТ 28147:2009 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования (ГОСТ 28147–89, IDT)

– ДСТУ ISO/IEC 18033-4:2015 (ISO/IEC 18033-4:2011, IDT) Інформаційні технології. Методи захисту. Алгоритми шифрування. Частина 4. Потокові шифри.

Опис алгоритму

В основі алгоритму *Струмок* лежить класична схема підсумовуючого генератора, подібна генератору *SNOW-2.0*, який визначено в ISO/IEC 18033-4:2011. Алгоритм *Струмок* використовує 256-бітний вектор

ініціалізації *IV* та 256-бітний або 512-бітний секретний ключ *K* і забезпечує високий та надвисокий рівень стійкості із врахуванням можливого застосування квантового криптографічного аналізу. Криптоалгоритм орієнтований на 64-розрядні обчислювальні системи, отже розмір слова визначено рівним 64 бітам.

Основними структурними компонентами генератору є регістр зсуву з лінійним зворотнім зв'язком (linear feedback shift register, *LFSR*) та скінченний автомат (finite-state machine, *FSM*), в якому виконується нелінійне перетворення. Вхідні дані використовуються для ініціалізації змінної стану Si $(i \geq 0)$, яка складається з вісімнадцяти 64-бітових блоків, до складу яких входить дві компоненти:

– 16 змінних $s^{(i)}$ – комірок регістра зсуву з лінійним зворотнім зв'язком:

$$s^{(i)} = (s_{15}{}^{(i)}, s_{14}{}^{(i)}, ..., s_0{}^{(i)});$$

– два регістри скінченного автомату

$$r\,(^{i}): r^{(i)} = (r_2{}^{(i)}, r_1\,(^{i}).$$

На виході отримуємо ключовий потік (гаму шифру), який формується з 64-бітових слів Z_i. Схематичне зображення функціонування генератора ключових потоків *Струмок* в довільний момент часу *i* наведено на рис. 17.23. Змінну часової залежності *i* не наведено.

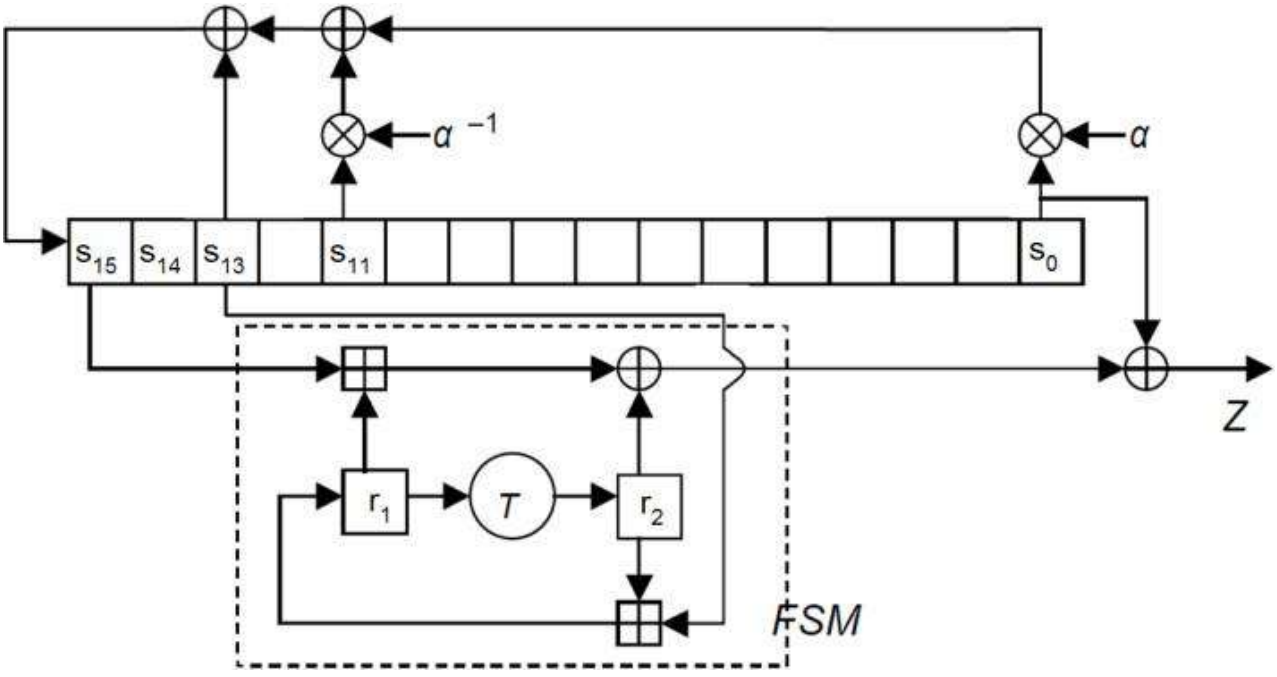


Рис. 21.23. Схематичне зображення генератору ключових потоків

Струмок у режимі генерації гами шифру (ключового потоку)

Відводи зворотного зв’язку у LFSR будується за примітивним над полем $GF\,(2^{64})$ поліномом:

$$f\,(x) = x^{16} + x^{13} + \alpha^{-1}x^{11} + \alpha,$$

де α є коренем примітивного над полем $GF\,(2^8)$ поліному:

$$g(z) = z^8 + \beta^{170}z^7 + \beta^{166}z^6 + \beta^2 z^5 + \beta^{224}z^4 + \beta^{70}z^3 + \beta^2.$$

В свою чергу поле $GF(2^8)$ будується за примітивним над полем $GF(2)$ поліномом:

$$p(y) = y^8 + y^4 + y^3 + y^2 + 1,$$

а коефіцієнти поліному $g(z)$ подаються через ступінь примітивного елементу β поля $GF(2^8)$, тобто β – корінь поліному $p(y)$.

Таким чином, маємо вежу полів:

$$GF(2) \subset GF(2^8) \subset GF(2^{64}) \subset GF(2^{1024}),$$

де:

– поле $GF(2^{1024})$ задається відводами зворотного зв'язку LFSR як фактор кільце $GF(2^{64})[x] / (f(x))$,

– поле $GF(2^{64})$ задається як факторкільце $GF(2^8)[z] / (g(z))$,

– поле $GF(2^8)$ задається як факторкільце $GF(2)[y]/ (p(y))$.

Отже період вихідної послідовності *LFSR* є максимальним і дорівнює $2^{1024}-1$.

Структурно в алгоритмі *Струмок* можна виділити три основні функції:

– функція ініціалізації *Init*, яка приймає в якості вхідних даних ключ *K* (256 біт або 512 біта) і вектор ініціалізації *IV* (256 біт), і виробляє початкове значення змінної стану:

$$S_0 = (s^{(0)}, r^{(0)});$$

– функція наступного стану *Next*, яка приймає на вхід змінну стану:

$$S_i = (s^{(i)}, r^{(i)})$$

і виробляє наступне значення змінної стану

$$S_{i+1} = (s^{(i+1)}, r^{(i+1)}).$$

Функція *Next* може виконуватися в двох режимах, в залежності від способу виконання ітерації – як частини реалізації або як частини нормального режиму генерації вихідних даних;

– функція ключового потоку *Strm*, що приймає на вході i змінну стану $S_i = (s^{(i)}, r^{(i)})$ і виробляє на виході 64-бітний ключовий потік Z_i.

Функція ініціалізації внутрішнього стану *Init*

Функція ініціалізації внутрішнього стану *Init* описується наступним чином.

Вхід: 256 або 512-бітний ключ *K*, 256-бітний вектор ініціалізації *IV*. *Вихід*: початкове значення змінної стану $S_0 = (s^{(0)}, r^{(0)})$.

Ключ для версії потокового шифру *Струмок-256* можна представити у вигляді чотирьох 64-бітних слів:

$$K = (K_3, K_2, K_1, K_0),$$

а для 512-бітного ключа – у вигляді восьми 64-бітних слів

$$K = (K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0),$$

де K_3 та K_7, відповідно для 256 и 512 біт, найбільш значущі слова, а K_0 – найменш значущі.

Вектор ініціалізації можна представити у вигляді чотирьох 64-бітних слів

$$IV = (IV_3, IV_2, IV_1, IV_0),$$

де IV_3 – найбільш значуще слово, а IV_0 – найменш значуще.

1. В 16 комірок LFSR заноситься значення ключа.

Для версії з 256-бітним ключем виконуються операції:

$$s_{15}^{(-33)} = \neg K_0,\ s_{14}^{(-33)} = K_1,\ s_{13}^{(-33)} = \neg K_2,\ s_{12}^{(-33)} = K_3,\ s_{11}^{(-33)} = K_0,\ s_{10}^{(-33)} = \neg K_1,\ s_9^{(-33)} = K_2,\ s_8^{(-33)} = K_3,$$
$$s_7^{(-33)} = \neg K_0,\ s_6^{(-33)} = \neg K_1,\ s_5^{(-33)} = K_2 \oplus IV_3,\ s_4^{(-33)} = K_3,\ s_3^{(-33)} = K_0 \oplus IV_2,\ s_2^{(-33)} = K_1 \oplus IV_1,$$
$$s_1^{(-33)} = K_2,\ s_0^{(-33)} = K_3 \oplus IV_0.$$

Для версії з 512-бітним ключем K виконуються операції:

$$s_{15}^{(-33)} = K_0,\ s_{14}^{(-33)} = \neg K_1,\ s_{13}^{(-33)} = K_2,\ s_{12}^{(-33)} = K_3,\ s_{11}^{(-33)} = \neg K_7,\ s_{10}^{(-33)} = K_5,\ s_9^{(-33)} = \neg K_6,$$
$$s_8^{(-33)} = K_4 \oplus IV_3,\ s_7^{(-33)} = \neg K_0,\ s_6^{(-33)} = K_1,\ s_5^{(-33)} = K_2 \oplus IV_2,\ s_4^{(-33)} = K_3,\ s_3^{(-33)} = K_4 \oplus IV_1,$$
$$s_2^{(-33)} = K_5,\ s_1^{(-33)} = K_6,\ s_0^{(-33)} = K_7 \oplus IV_0.$$

2. Виконуються 32 ініціюючих такти без генерації ключового потоку, тобто чотири пов них циклів. Формально це подається наступним чином:

$$S_{-1} = Next^{32}(S_{-33}, INIT),$$

що означає 32 ітерації з виконання функції *Next* у режимі ініціалізації *INIT*, $S_{-33} = (s^{(-33)}, r^{(-33)})$ – обраховані на попередньому кроці значення змінної стану.

3. Розраховується початкове значення змінної стану $S_0 = (s^{(0)}, r^{(0)})$ за правилом: $S_0 = Next\ (S_{-1})$, тобто шляхом виконання функції *Next* у звичайному режимі.

4. Виводиться вихідне значення $S_0 = (s^{(0)}, r^{(0)})$.

Функція наступного стану *Next*

Функція стану *Next* описується наступним чином.

Вхід: Змінна стану $S_i = (s^{(i)}, r^{(i)})$, обраний режим (звичайний, або режим ініціалізації).

Вихід: Наступне значення змінної стану $S_{i+1} = (s^{(i+1)}, r^{(i+1)})$.

1. Виконується нелінійна підстановка для оновлення значення регістру $r_2^{(i+1)}$ скінченного автомату. Для цього розраховується значення функції T: $r_2^{(i+1)} = T\ (r_1^{(i)})$.

2. Оновлюється значення регістру значення $r_1^{(i+1)}$ скінченного автомату. Для цього розраховується значення:

$$r_1^{(i+1)} = r_2^{(i+1)} +_{64} s_{13}^{(i)},$$

де $+_{64}$ позначає операцію додавання цілих чисел за модулем 2^{64} (у схемі шифру на рис. 17.23 цю операцію позначено як ⊞).

3. Оновлюється значення 15 комірок *LFSR*

$$s_j^{(i+1)} = s_{j+1}^{(i)}$$

для всіх $j = 0, 1, \ldots, 14$.

4. Оновлюється значення 16-ї комірки *LFSR*. Якщо встановлено звичайний режим функції *Next*, значення цієї комірки обчислюється за правилом:

$$s_{15}^{(i+1)} = (s_0^{(i)} \otimes \alpha) \oplus (s_{11}^{(i)} \otimes \alpha^{-1}) \oplus s_{13}^{(i)}.$$

Якщо встановлено режим ініціалізації *INIT* функції *Next*, значення обчислюється за правилом:

$$s_{15}^{(i+1)} = FSM(s_{15}^{(i)}, r_1^{(i)}, r_2^{(i)}) \oplus (s_0^{(i)} \otimes \alpha) \oplus (s_{11}^{(i)} \otimes \alpha^{-1}) \oplus s_{13}^{(i)}.$$

Операції множення $\otimes$ на α та на α^{-1} та сутність функції *FSM* пояснюються далі.

5. Обчислюється та виводиться значення змінної стану $S = (s^{(i)}, r^{(i)})$.

Схематичне зображення генератору ключових потоків *Струмок* при виконанні функції *Next* у режимі ініціалізації представлено на рис. 17.24.

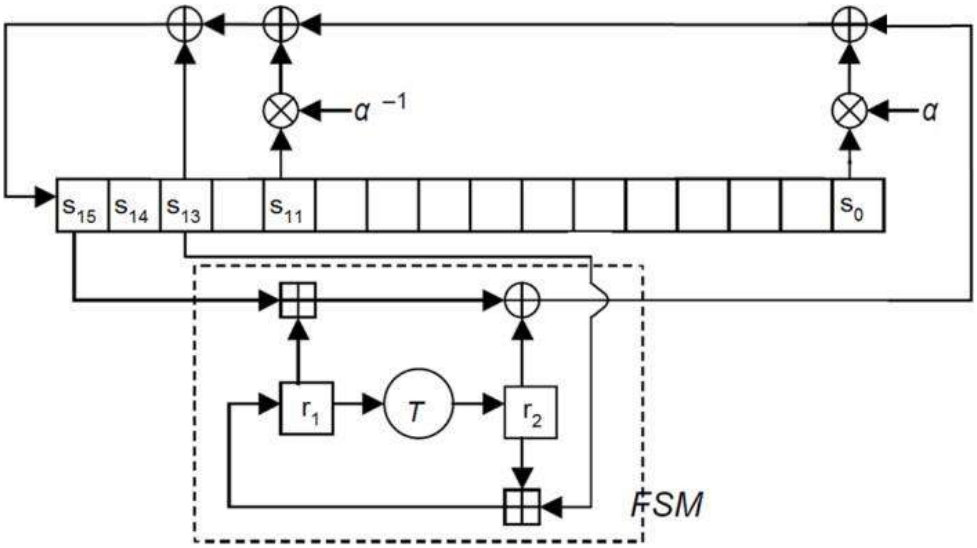


Рис. 21.24. Схематичне зображення генератору ключових потоків Струмок у режимі ініціалізації функції *Next*

Функція ключового потоку *Strm*

Функція ключового потоку *Strm* описується наступним чином.

Вхід: Змінна стану $S_i = (s^{(i)}, r^{(i)})$.

Вихід: 64-бітовий ключовий потік

1. Обчислюється значення

$$Z_i = FSM(s_{15}^{(i)}, r_1^{(i)}, r_2^{(i)}) \oplus s_0^{(i)}.$$

2. Виводиться вихідне значення *Zi*.

Функція скінченного автомату *FSM*

Функція скінченного автомату позначається як *FSM* (*x*, *y*, *z*) та описується наступним чином.

Вхід: три 64-бітових рядка *x*, *y* і *z*. *Вихід*: 64-бітовий рядок *q*.

1. Обчислюється значення $q = (x +_{64} y) \oplus z$.

2. Виводиться вихідне значення *q*.

Функція нелінійної підстановки *T*

Функція нелінійної підстановки *T* реалізує перестановку елементів скінченного поля *GF* (2^{64}) за допомогою компонентів національного стандарту блокового симетричного криптоперетворення ДСТУ 7624:2014.

Вхід: 64-бітовий рядок *w*.

Вихід: 64-бітовий рядок $T = T(w)$.

1. Вхідний 64-бітовий рядок *w* розбивається на підблоки w_j по 8 біт:

$$w = (w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0),$$

2. Для кожного підблоку w_j виконується підстановка з алгоритму ДСТУ 7624:2014 за допомогою чотирьох табличних перетворень π_0, π_1, π_2, π_3 (див. додаток Б). Виконання функції T за допомогою цих перетворень схематично (на прикладі підблоків w_j, що подано у шіснадцятковому вигляді)

зображено на рис. 21.25.

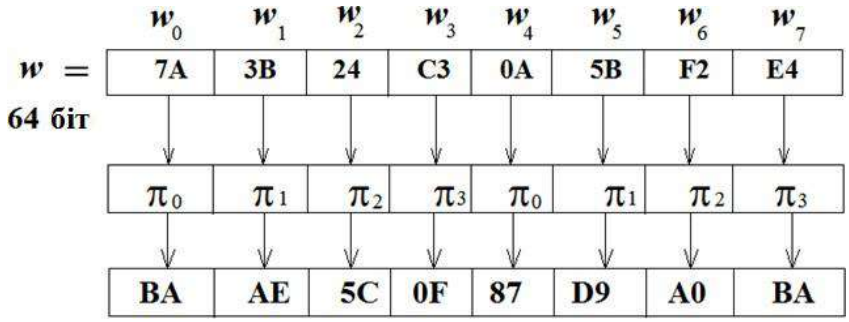


Рис. 21.25. Схематичне зображення виконання процедури підстановки $T = T(w)$

У результаті формується вихідний вектор $r = (r_7, r_6, r_5, r_4, r_3, r_2, r_1, r_0)$:

$$r_j = \pi_{j \bmod 4}\left[w_j\right],$$

де $j = 0, 1, \ldots, 7$.

3. Обчислюється вектор

$$q = (q_7, q_6, q_5, q_4, q_3, q_2, q_1, q_0)$$

за правилом:

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \end{pmatrix} = \begin{pmatrix} 01 & 01 & 05 & 01 & 08 & 06 & 07 & 04 \\ 04 & 01 & 01 & 05 & 01 & 08 & 06 & 07 \\ 07 & 04 & 01 & 01 & 05 & 01 & 08 & 06 \\ 06 & 07 & 04 & 01 & 01 & 05 & 01 & 08 \\ 08 & 06 & 07 & 04 & 01 & 01 & 05 & 01 \\ 01 & 08 & 06 & 07 & 04 & 01 & 01 & 05 \\ 05 & 01 & 08 & 06 & 07 & 04 & 01 & 01 \\ 01 & 05 & 01 & 08 & 06 & 07 & 04 & 01 \end{pmatrix} \cdot \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{pmatrix},$$

де елементи матриці (подано у шістнадцятковому вигляді) та векторів r і q інтерпретуються як елементи скінченного поля $GF(2^8)$, яке задане як факторкільце $GF(2)[y]/(p(y))$.

Цю операцію можна записати у скороченому вигляді (як у ДСТУ 7624:2014):

$$q_i = (\upsilon >>> i) r^T,$$

де $\upsilon = (01, 01, 05, 01, 08, 06, 07, 04)$, $>>> i$ – операція циклічного зсуву на i розрядів праворуч, $i = 0,1,..., 7$,

$$r^T = (r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7)^T.$$

4. Виводиться вихідне значення q, яке інтерпретується як 64-бітовий рядок. Швидке обчислення вектору $(q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7) = Q$ реалізується за правилом

$$Q^T = T_0[w_0] \oplus T_1[w_1] \oplus T_2[w_2] \oplus T_3[w_3] \oplus T_4[w_4] \oplus T_5[w_5] \oplus T_6[w_6] \oplus T_7[w_7],$$

де

$$T_0[a] = \begin{pmatrix} 01 \\ 04 \\ 07 \\ 06 \\ 08 \\ 01 \\ 05 \\ 01 \end{pmatrix} \cdot \pi_0[a],\ T_1[a] = \begin{pmatrix} 01 \\ 01 \\ 04 \\ 07 \\ 06 \\ 08 \\ 01 \\ 05 \end{pmatrix} \cdot \pi_1[a],\ T_2[a] = \begin{pmatrix} 05 \\ 01 \\ 01 \\ 04 \\ 07 \\ 06 \\ 08 \\ 01 \end{pmatrix} \cdot \pi_2[a],\ T_3[a] = \begin{pmatrix} 01 \\ 05 \\ 01 \\ 01 \\ 04 \\ 07 \\ 06 \\ 08 \end{pmatrix} \cdot \pi_3[a],$$

$$T_4[a] = \begin{pmatrix} 08 \\ 01 \\ 05 \\ 01 \\ 01 \\ 04 \\ 07 \\ 06 \end{pmatrix} \cdot \pi_0[a],\ T_5[a] = \begin{pmatrix} 06 \\ 08 \\ 01 \\ 05 \\ 01 \\ 01 \\ 04 \\ 07 \end{pmatrix} \cdot \pi_1[a],\ T_6[a] = \begin{pmatrix} 07 \\ 06 \\ 08 \\ 01 \\ 05 \\ 01 \\ 01 \\ 04 \end{pmatrix} \cdot \pi_2[a],\ T_7[a] = \begin{pmatrix} 04 \\ 07 \\ 06 \\ 08 \\ 01 \\ 05 \\ 01 \\ 01 \end{pmatrix} \cdot \pi_3[a].$$

Застосування таблиць-констант T_i $[a]$, $i = 0,1,..., 7$ дозволяє значно зменшити кількість операцій, зокрема функція нелінійної підстановки обчислюється за сім операцій XOR над 64-бітовими рядками.

Множення на α в арифметиці поля *GF* (2^{64})

Множення на α в арифметиці поля *GF* (2^{64}) реалізується за допомогою таблиці передобчислень Mul_α з 256 рядків по 64 бітів в кожному.

Вхід: 64-бітовий рядок *w*, що представляє елемент поля *GF* (2^{64}).

Вихід: 64-бітовий рядок $w' = w \otimes \alpha$, що представляє елемент поля *GF* (2^{64}).

1. Обчислюється значення

$$w' = (w << 8) \oplus Mul_\alpha[w >> 56], \qquad (21.1)$$

де *w* << 8 є результатом зсуву ліворуч (в бік старших розрядів) 64-бітового рядка *w* на вісім розрядів із заповненням молодших розрядів нульовими значеннями;

w >> 56 є результатом зсуву праворуч (в бік молодших розрядів) 64-бітового рядка *w* на 56 розрядів із заповненням старших розрядів нульовими значеннями. Вісім молодших розрядів вектору *w* >> 56 інтерпретуються як елемент поля *GF* (2^8) для індексації таблиці передобчислень Mul_α;

Mul_α – таблиця-константа з 256 рядків по 64 біта в кожному (таблиця передобчислень);

$Mul_\alpha[c]$ – 64-бітне значення таблиці передобчислень у рядку з індексом *c*, де *c* представляє елемент поля *GF* (2^8), $Mul_\alpha[c]$ представляє елемент поля *GF* (2^{64}). 2. Виводиться вихідне значення *w*'.

Множення на α^{-1} в арифметиці поля *GF* (2^{64})

Множення на α^{-1} в арифметиці поля *GF* (2^{64}) реалізується за допомогою таблиці перед обчислень $Mul_{\alpha^{-1}}$ з 256 рядків по 64 бітів в кожному.

Вхід: 64-бітовий рядок *w*, що представляє елемент поля *GF* (2^{64}).

Вихід: 64-бітовий рядок $w' = w \otimes \alpha^{-1}$, що представляє елемент поля *GF* (2^{64}).

1. Обчислюється значення

$$w' = (w >> 8) \oplus Mul_{\alpha^{-1}}[w \,\&\, \gamma], \tag{21.2}$$

де:

w >> 8 є результатом зсуву праворуч (в бік молодших розрядів) 64-бітового рядка *w* на 8 розрядів із заповненням старших розрядів нульовими значеннями;

w & γ є результатом побітової кон'юнкції 64-бітового рядка *w* та 64-бітового рядка γ, який у шістнадцятковому поданні має вигляд γ = 00000000000000FF. Вісім молодших розрядів вектору *w* & γ інтерпретуються як елемент поля *GF* (2^8) для індексації таблиці перед обчислень $Mul_{\alpha^{-1}}$;

2. Виводиться вихідне значення *w*'.

Значення таблиць-констант Mul_{α}, $Mul_{\alpha^{-1}}$

Для швидкого шифрування застосовуються таблиці передобчислень Mul_{α}, $Mul_{\alpha^{-1}}$, Це дозволяє значно зменшити кількість операцій для обробки блоку вхідних даних.

Поліном, що задає зворотний зв'язок LFSR має вигляд:

$$f(x) = x^{16} + x^{13} + \alpha^{-1}x^{11} + \alpha,$$

де α та α^{-1} належать полю *GF* (2^{64}), причому $\alpha = z$ є коренем примітивного над полем *GF* (2^8) поліному *g* (*z*). Таким чином, у кожній комірці LFSR зберігається 64-бітна послідовність *w*, яку представимо у вигляді восьми підблоків w_j по вісім біт у кожному:

$$w = (w_7, w_6, w_5, w_4, w_3, w_2, w_1, w_0),$$

які інтерпретуються як коефіцієнти поліному

$$w(z) \in GF(2^8)[z]/(g(z)).$$

Якщо $\alpha = z$ є коренем примітивного над *GF* (2^8) поліному:

$$g(z) = z^8 + g_7 z^7 + \ldots + g_1 z + g_0$$

тоді маємо:

$$\begin{aligned} w(z)\cdot\alpha &= (w_7 z^8 + w_6 z^7 + \ldots + w_1 z^2 + w_0 z) \bmod g(z) \equiv \\ &\equiv (w_6 + w_7 g_7) z^7 + (w_5 + w_7 g_6) z^6 + \ldots + (w_0 + w_7 g_1) z + (w_7 g_0) z^0 = \\ &= w_{<<8}(z) + w_{>>56}(z)\cdot g'(z), \end{aligned}$$

де поліноми

$$w_{<<8}(z),\ w_{>>56}(z) \text{ та } g'(z)$$

мають вигляд:

$$w_{<<8}(z) = w_6 z^7 + w_5 z^6 + \ldots + w_0 z,$$

$$w_{>>56}(z) = w_7,$$

$$g'(z) = g_7 z^7 + g_6 z^6 + \ldots + g_1 z + g_0.$$

Двійкове подання коефіцієнтів поліномів

$$w_{<<8}(z) \quad \text{і} \quad w_{>>56}(z)$$

утворює розглянуті у (17.1) двійкові послідовності $w << 8$ і $w >> 56$. Отже, обчислення $w(z)\cdot\alpha$ в арифметиці поля $GF(2^{64})$ відповідає формулі (17.1), де 256 значень таблиці $Mul_\alpha[w_7]$ розраховуються як 64-бітні послідовності при двійковому поданні коефіцієнтів (w_7g_7, w_7g_6,..., w_7g_1, w_7g_0) поліному:

$$w_{>>56}(z)\cdot g'(z) = w_7(g_7 z^7 + g_6 z^6 + \ldots + g_1 z + g_0)$$

для кожного з 256 можливих значень $w_7 \in GF(2^8)$.

Якщо $\alpha = z$ є коренем примітивного над $GF(2^8)$ поліному

$$g(z) = z^8 + g_7 z^7 + \ldots + g_1 z + g_0,$$

тоді маємо:

$$\alpha^8 = g_7\alpha^7 + \ldots + g_1\alpha + g_0\alpha^0,$$

або

$$\alpha^7 = g_7\alpha^6 + \ldots + g_1\alpha^0 + g_0\alpha^{-1},$$

отже

$$g_0^{-1}\alpha^7 + g_0^{-1}g_7\alpha^6 + \ldots + g_0^{-1}g_1\alpha^0 = \alpha^{-1} = z^{-1}.$$

Тоді

$$w(z)\alpha^{-1} = w_7 z^6 + w_6 z^5 + \ldots + w_1 z^0 + w_0 z^{-1} =$$
$$w_7 z^6 + w_6 z^5 + \ldots + w_1 z^0 + w_0(g_0^{-1}z^7 + g_0^{-1}g_7 z^6 + \ldots + g_0^{-1}g_1 z^0) =$$
$$= (w_0 g_0^{-1})z^7 + (w_7 + w_0 g_0^{-1} g_7)z^6 + \ldots + (w_1 + w_0 g_0^{-1} g_1)z^0 =$$
$$= w_{>>8}(z) + w_0(z)\cdot g''(z),$$

де поліноми $w_{>>8}(z)$, $w_0(z)$ **та** $g''(z)$ **мають вигляд:**

$$w_{>>8}(z) = w_7 z^6 + w_6 z^5 + \ldots + w_2 z + w_1,$$

$$w_0(z) = w_0,$$

$$g''(z) = g_0^{-1}z^7 + g_0^{-1}g_7 z^6 + \ldots + g_0^{-1}g_2 z + g_0^{-1}g_1.$$

Двійкове подання коефіцієнтів поліномів

$$w_{>>8}(z) \quad \text{і} \quad w_0(z)$$

утворює розглянуті у (17.2) двійкові послідовності $w >> 8$ і γ. Таким чином, обчислення $w(z)\cdot\alpha^{-1}$ в арифметиці поля $GF(2^{64})$ відповідає формулі (17.2), де 256 значень таблиці $Mul_{\alpha^{-1}}[w]$ розраховуються як 64-бітні послідовності при двійковому поданні коефіцієнтів:
$(w_0 g_0^{-1}, w_0 g_0^{-1} g_7, \ldots, w_0 g_0^{-1} g_2, w_0 g_0^{-1} g_1)$ **поліному**

$$w_0(z)\cdot g''(z) = w_0(g_0^{-1}z^7 + g_0^{-1}g_7 z^6 + \ldots + g_0^{-1}g_2 z + g_0^{-1}g_1)$$

для кожного з 256 можливих значень $w \in GF(2^8)$.

Висновки

Генератор ключових потоків *Струмок* дозволяє формувати псевдовипадкові послідовності із швидкістю понад 10 Гбіт/с. За цим показником він випереджає майже всі найбільш поширені шифри, зокрема і алгоритм *SNOW2.0*.

Генератор ключового потоку *Струмок* у своїй концептуальній схемі подібний до SNOW2.0. Але розробники SNOW2.0 зосереджувалися на використанні 32-розрядних обчислювальних систем, тоді як *Струмок* призначений для використання в більш потужних 64-розрядних обчислювальних системах. У зв'язку з цим в алгоритмі *Струмок* підвищується швидкість формування псевдовипадкової послідовності, що використовується 64-розрядними словами, для зберігання потоку ключів шифрування. Проведені порівняльні тести показали, що алгоритм *Струмок* на 32-розрядних обчислювальних системах також демонструє хороші результати роботи. Використання попереднього обчислення збільшує швидкість алгоритму, оскільки в процесі генерації ключові потоку немає необхідності в складних калькуляціях. В алгоритмі *Струмок* збільшені, в порівнянні з *SNOW2.0*, довжини секретного ключа та вектору ініціалізації. Це дозволяє надійно застосовувати потоковий шифр навіть з із врахуванням квантових методів криптографічного аналізу.