Інститут спеціального зв’язку та захисту інформації
Національного технічного університету України
“Київський політехнічний інститут імені Ігоря Сікорського”

І. В. Самойлов, А. А. Матійко, А.С. Сторчак

КРИПТОГРАФІЯ

Навчальний посібник

Рекомендовано до друку Вченою радою ІСЗЗІ КПІ ім. Ігоря Сікорського
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності 125 Кібербезпека та захист інформації

Київ
КПІ ім. Ігоря Сікорського
2023

УДК 004.056.55

Рекомендовано Вченою радою
ІСЗЗІ КПІ ім. Ігоря Сікорського
(протокол № 3 від 30. 10. 2023)

Рецензент: В. Є. Чевардін, доктор технічних наук, старший науковий співробітник, начальник кафедри Кібербезпеки ВІТІ імені Героїв Крут

Самойлов І. В, Матійко А. А, Сторчак А. С Криптографія: навчальний посібник. Київ: ІСЗЗІ КПІ ім. Ігоря Сікорського, 2023. 372 с.

Навчальний посібник містить основні поняття криптології, наведено класифікацію шифрів, описано класичні криптосистеми та методи їх криптоаналізу, розглянуто основні алгоритми симетричної та асиметричної криптографії, приділено увагу національним стандартам криптографічного захисту інформації.

Посібник призначений для підготовки здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 125 Кібербезпека та захист інформації.

© ІСЗЗІ КПІ ім. Ігоря Сікорського, 2023

ЗМІСТ

ВСТУП

Криптографія (наука про шифри) довгий час була засекречена, оскільки застосовувалася, в основному, для захисту державних і військових секретів. В даний час методи та засоби криптографії використовуються для забезпечення інформаційної безпеки не тільки держави, а й приватних осіб та організацій. Справа тут зовсім не обов'язково у секретах. Занадто багато різних відомостей гуляє по всьому світу у цифровому вигляді. І над цими відомостями висять загрози недружнього ознайомлення, накопичення, підміни, фальсифікації тощо. Найбільш надійні методи захисту від таких загроз дає саме криптографія.

Криптографія займається розробкою алгоритмів перетворення повідомлень, в тому числі шляхом шифрування з використанням спеціальних даних. Формування загальних принципів побудови систем криптографічного захисту інформації на основі використання сучасних алгоритмів симетричного й асиметричного шифрування для забезпечення конфіденційності даних в інформаційно-комунікаційних системах є метою даного навчального посібника.

Посібник містить десять розділів, кожен з яких доповнений контрольними питаннями для самостійного опрацювання. Перший розділ містить загальні відомості про основи криптології. У ньому розкрито основи захисту інформації в інформаційно-комунікаційних системах, роль криптографічних протоколів у загальній задачі забезпечення інформаційної безпеки, основи теорії засекреченого зв'язку К. Шеннона, наведено класифікацію методів шифрування повідомлень та методів криптографічного аналізу. Другий розділ присвячений історичним шифрам підстановки (моноалфавітні, багатоалфавітні) та перестановки. Третій розділ визначає принципи побудови сучасних блокових систем шифрування, що використовують блокові та складені шифри з криптографічним аналізом атаки на блокові шифри. Четвертий розділ розкриває суть стандарту

блокового симетричного шифрування Data Encription Standard (DES): історія розробки, принципи побудови, структура, приклади шифрування та криптографічний аналіз. П’ятий розділ містить режими роботи блокових систем шифрування. У шостому розділі детально розглядається стандарт криптографічного перетворення даних ДСТУ ГОСТ 28147:2009: у режимі простої заміни, режимі гамування, режимі гамування зі зворотним зв’язком та режимі вироблення імітовставки. Сьомий розділ присвячений стандарту блокового симетричного шифрування даних Advanced Encryption Standard (AES): структура стандарту, раундові перетворення тощо. У восьмому розділі детально розглядається національний стандарт симетричного блокового перетворення даних ДСТУ 7624:2014: формат даних, алгоритми шифрування, розшифрування та формування раундових ключів, режими роботи. У дев’ятому розділі розкрито потокові системи шифрування (A5, RC4, національний алгоритм симетричного потокового перетворення ДСТУ 8845:2019) та генератори псевдовипадкових чисел. У десятому розділі розглянуто асиметричні криптографічні системи: Діффі-Хеллмана, RSA, Ель-Гамаля, Рабіна.

Посібник призначений для здобувачів вищих навчальних закладів, які навчаться в галузі знань 12 “Інформаційна безпека” за спеціальністю 125 “Кібербезпека та захист інформації” та спеціалістів з експлуатації й застосування криптографічних систем, комплексів і засобів захисту інформації в інформаційно-комунікаційних системах, а також може бути використаний аспірантами й науковцями при проведенні наукових досліджень.

РОЗДІЛ 1
ОСНОВИ КРИПТОЛОГІЇ

1.1. Загальні відомості про захист інформації в інформаційно-комунікаційних системах

Стрімкий розвиток засобів обчислювальної техніки і відкритих мереж передачі даних зумовило їх широке поширення в повсякденному житті й підприємницькій діяльності. Потужні обчислювальні можливості і оперативність передачі інформації не лише вплинули на принципи ведення бізнесу, що склалися в більшості традиційних галузей, але й відкрили нові напрями розвитку підприємницької діяльності [23].

Однак останні досягнення людської думки в галузі комп'ютерних технологій пов'язані з появою не лише персональних комп'ютерів, мереж передачі даних і електронних грошей, але й таких понять, як хакер, інформаційна зброя, комп'ютерні віруси тощо.

На практиці загрози інформаційно-комунікаційним системам (ІКС) можуть бути реалізовані безпосереднім впливом на інформацію, що становить інтерес для кінцевих користувачів подібних систем, так і на інформаційні ресурси та телекомунікаційні служби, що забезпечуються в межах цієї ІКС [3, 6, 18, 19]. Наприклад, існує поширений вид атаки через Internet – шторм помилкових запитів на TCP (Transmission Control Protocol – протокол управління передачею) – з'єднання, що призводить до того, що система тимчасово припиняє обслуговування віддалених користувачів.

Під інформаційною безпекою будемо розуміти стан захищеності даних, які оброблюються, зберігаються і передаються в ІКС, від незаконного ознайомлення, перетворення та знищення (як крайній випадок модифікації), а також стан захищеності інформаційних ресурсів від дій, спрямованих на порушення їх працездатності [3, 4, 5, 6, 18, 19].

Основними завданнями захисту інформації, яка призначена для

користувача, є забезпечення [3, 4, 5, 6, 18, 22, 24, 26]:

– конфіденційності інформації;

– цілісності інформації;

– достовірності інформації;

– оперативності доступу до інформації;

– неспростовності інформації;

– юридичної значущості інформації, наданої у вигляді електронного документа;

– невідстежуваності дій клієнта.

Конфіденційність інформації – це її властивість бути доступною тільки обмеженому колу користувачів ІКС, в якій циркулює ця інформація.

Під *цілісністю* розуміють властивість інформації або програмного забезпечення зберігати свою структуру та вміст у процесі передачі та зберігання. Розглядаючи питання передачі інформації у вигляді повідомлення через мережу, можна дійти висновку, що кожне повідомлення за своїм змістом утворює певний клас. Іншими словами, сенс кінцевого повідомлення залишиться таким самим, як і початкового, навіть якщо форма подання інформації в електронному вигляді суттєво зміниться. Таким чином, кожне повідомлення на будь-якій мові матиме свій клас еквівалентності, і для даного випадку властивість збереження цілісності можна сформулювати наступним чином: передане повідомлення M вважається таким, що зберегло цілісність, якщо отримане в результаті передачі повідомлення M' належить класу еквівалентності повідомлення M.

Достовірність інформації – це властивість, що виражається в строгій приналежності до об’єкта, який є її джерелом, або того об’єкта, від якого ця інформація прийнята.

Оперативність – це здатність інформації або деякого інформаційного ресурсу бути доступним для кінцевого користувача відповідно до його тимчасових потреб.

Неспростовність – властивість запобіганню можливості заперечення реальними користувачами фактів повного або часткового взяття участі в інформаційному обміні або інформаційній взаємодії [8].

Юридична значущість означає, що документ має юридичну силу. З цією метою суб'єкти, які потребують підтвердження юридичної значущості повідомлення, що передається, домовляються про загальне прийняття деяких атрибутів інформації, що виражають її здатність бути юридично значущою. Ця властивість інформації особливо актуальна в системах електронних платежів, де здійснюється операція з переказу грошових коштів. Виходячи зі сказаного, можна сформулювати деякі вимоги до атрибутів інформації, що виражають її властивість бути юридично значущою. Інформацію необхідно сформувати таким чином, щоб із формального погляду було достовірно зрозуміло, що тільки відправник, якому належить цей платіжний документ, міг його створити.

Невідстежуваність – це здатність здійснювати деякі дії в ІКС непомітно для інших об'єктів. Актуальність цієї вимоги стала очевидною завдяки появі таких понять, як електронні гроші і Internet-banking. Так, для авторизації доступу до електронної платіжної системи користувач повинен надати деякі відомості, що однозначно його ідентифікують. У міру стрімкого розвитку таких систем може з'явитися реальна небезпека у тому, що, наприклад, усі платіжні операції будуть контролюватися, тим самим виникнуть умови для тотального стеження за користувачами ІКС.

Існує декілька шляхів вирішення проблеми невідстежуваності:

– заборона за допомогою законодавчих актів будь-якого тотального стеження за користувачами ІКС;

– застосування криптографічних методів для підтримки невідстежуваності.

Як вже зазначалося, інформаційна безпека може розглядатися не лише у відношенні до деяких конфіденційних відомостей, але й відносно здатності ІКС виконувати задані функції.

Основні завдання, що вирішуються в рамках інформаційної безпеки відносно працездатності ІКС, повинні забезпечувати захист від:

– порушення функціонування ІКС, що виражається у впливі на інформаційні канали, канали сигналізації, управління і віддаленого завантаження баз даних комутаційного обладнання, системне та прикладне програмне забезпечення;

– несанкціонованого доступу до інформаційних ресурсів і від спроб використання ресурсів мережі, що призводять до витоку даних, порушення цілісності мережі та інформації, зміни функціонування підсистеми розподілу інформації, доступності баз даних;

– руйнування вбудованих і зовнішніх засобів захисту;

– неправомірних дій користувачів і обслуговуючого персоналу мережі.

Пріоритети серед перерахованих завдань інформаційної безпеки визначаються індивідуально для кожної конкретної ІКС і залежать від вимог, що пред'являються безпосередньо до інформаційних систем.

Слід врахувати, що з погляду державних структур заходи захисту, в першу чергу, повинні забезпечувати конфіденційність, цілісність і доступність інформації. Зрозуміло, що для режимних державних організацій на першому місці завжди стоїть конфіденційність відомостей, а цілісність розуміється виключно як їх незмінність. Комерційним структурам, ймовірно, найважливішими є цілісність і доступність даних та послуг щодо їх обробки. У порівнянні з державними комерційні організації є більш відкритими й динамічними, тому ймовірні загрози для них відрізняються не лише кількістю, але й якістю.

Для вирішення завдання інформаційної безпеки в ІКС необхідно забезпечити:

– захист інформації під час її зберігання, обробки та передачі по каналах зв'язку;

– підтвердження достовірності об'єктів даних і користувачів

(автентифікація сторін, що встановлюють зв’язок);

– виявлення і попередження порушення цілісності об’єктів даних;

– захист конфіденційної інформації від її можливого витоку по каналах та пристроях негласного отримання цієї інформації;

– захист програмних продуктів від впровадження програмних закладок і вірусів;

– захист від несанкціонованого доступу до інформаційних ресурсів і технічних засобів каналу передачі даних, зокрема, до засобів управління, щоб запобігти зниження рівня захищеності інформації й самого каналу в цілому;

– організація необхідних заходів, спрямованих на забезпечення збереження конфіденційних даних;

– захист технічних пристроїв та приміщень.

Конкретна реалізація загальних принципів забезпечення інформаційної безпеки може полягати в організаційних або технічних заходах захисту інформації.

Слід зазначити, що обсяг заходів із захисту даних, які обробляються й передаються, залежить передусім від величини можливого збитку. Ця величина може визначатися в прямій (наприклад, витрати на купівлю нового програмного забезпечення у разі порушення його цілісності) або в опосередкованій (наприклад, витрати від простою інформаційної системи банку) формі. Правда, у деяких ситуаціях розрахувати величину збитку складно (наприклад, у випадку витоку державної таємниці).

1.2. Роль криптографічних протоколів у загальній задачі забезпечення інформаційної безпеки

Основу забезпечення інформаційної безпеки в ІКС складають криптографічні методи й засоби захисту інформації. Слід врахувати, що найбільш надійний захист можна забезпечити тільки за допомогою комплексного підходу, тобто вирішення задачі за допомогою сукупності

організаційно-технічних та криптографічних заходів.

В основі криптографічних методів лежить поняття криптографічного перетворення інформації, встановленого за певними математичними законами, з метою виключення несанкціонованого доступу до інформації та забезпечення неможливості безконтрольної її зміни сторонніми користувачами.

Застосування криптографічних методів захисту забезпечує вирішення основних завдань інформаційної безпеки. Цього можна досягнути шляхом реалізації наступних криптографічних методів захисту: як інформації з обмеженим доступом, так і інформаційних ресурсів у цілому:

– шифрування всього інформаційного потоку, що передається через відкриті мережі передачі даних, і окремих повідомлень;

– криптографічної автентифікації різнорівневих об’єктів, що встановлюють зв’язок (мається на увазі рівні моделі взаємодії відкритих систем);

– захисту інформаційного потоку, що несе дані засобами імітозахисту (захисту від нав’язування помилкових повідомлень) і електронного цифрового підпису (ЕЦП) з метою забезпечення цілісності й достовірності інформації, що передається [13];

– шифрування даних, що надані у вигляді файлів або зберігаються в базі даних;

– застосування ЕЦП для забезпечення юридичної значущості платіжних документів;

– застосування сліпого ЕЦП для забезпечення невідстежуваності дій клієнта в платіжних системах, заснованих на понятті електронних грошей.

При реалізації більшості з наведених методів криптографічного захисту виникає необхідність обміну деякою інформацією. Наприклад, автентифікація об’єктів ІКС супроводжується обміном ідентифікуючої і автентифікуючої інформації.

У загальному випадку взаємодія об’єктів (суб’єктів) подібних систем завжди супроводжується дотриманням деяких домовленостей, що називаються *протоколом*. Формально протоколом будемо вважати послідовність дій об’єктів (суб’єктів) для досягнення певної мети, яка в даному випадку визначає структуру і специфіку застосування протоколу.

У свою чергу, *криптографічними протоколами* будемо називати ті, в яких учасники для досягнення певної мети використовують криптографічні перетворення інформації.

Перерахуємо основні завдання забезпечення інформаційної безпеки, які вирішуються за допомогою криптографічних протоколів:

– обмін ключовою інформацією з подальшим встановленням захищеного обміну даними; при цьому не існує ніяких припущень, чи спілкувалися заздалегідь між собою сторони, що обмінюються ключами (наприклад, без використання криптографічних протоколів неможливо було б створити системи розподілу ключової інформації в розподілених мережах передачі даних);

– автентифікація сторін, що встановлюють зв’язок;

– авторизація користувачів при доступі до ІКС.

На сьогодні завдяки широкому застосуванню відкритих мереж передачі даних, таких як Internet, і побудованих на її основі мереж – intranet і extranet – криптографічні протоколи набули широкого застосування для вирішення різноманітного кола завдань і забезпечення, послуг, що постійно розширюються і надаються користувачам таких мереж.

Окрім розглянутих класичних сфер застосування протоколів існує широке коло специфічних завдань, які також вирішуються за допомогою відповідних криптографічних протоколів. Це, передусім, встановлення частини відомостей без повного або часткового розкриття самого секрету в його справжньому об’ємі. Так, наприклад, учасники можуть для досягнення певної спільної мети повідомити один одному частину своєї інформації або об’єднати зусилля для розкриття секрету, невідомого кожному з них окремо.

Стрімке удосконалення криптографічних протоколів більшою мірою стимулюється розвитком систем електронних платежів, інтелектуальних карток, появою електронних грошей тощо.

Оскільки на сьогодні основним криптографічним засобом захисту інформації в Internet є протоколи, то можна констатувати, що удосконалення подібних засобів захисту інформації з обмеженим доступом буде тривати і в кількісному, і в якісному відношенні.

Багатогранність застосування криптографічних протоколів у вирішенні задачі забезпечення інформаційної безпеки, як в локальних, так і в розподілених інформаційних системах, приводить до необхідності детального розгляду їх основних типів, питань практичного застосування таких протоколів і побудови на їх основі спеціальних інформаційних систем.

1.3. Загальні відомості про криптологію

1.3.1. Криптологія як наука

Криптологія – це галузь знань, що вивчає тайнопис (*криптографія*) і методи її розкриття (*криптографічний аналіз*), яка за влучним висловлюванням Рональда Рівеста (професора Массачусетського технологічного інституту і одного з авторів знаменитої криптографічної системи RSA) є “повитухою усієї computer science взагалі” [19, 38]. Назва криптології пішла від грецької мови (*криптос* – таємний, *логос* – наука чи слово).

Побудова сучасної криптології як науки ґрунтується на сукупності фундаментальних понять і фактів математики, фізики, теорії інформації та складності обчислень, навіть для всебічного та глибокого осмислення професіоналами. Однак, незважаючи на органічно притаманну їй складність, багато теоретичних досягнень криптології наразі широко використовуються в нашому насиченому інформаційними технологіями житті, наприклад: у

пластикових smart-картах, електронній пошті, у системах банківських платежів, електронної торгівлі через Internet, у системах електронного документообігу, під час ведення баз даних, у системах електронного голосування та ін. Подібне співвідношення загальної внутрішньої складності та практичної застосовності для теоретичної науки, напевно, унікальне.

Криптографія – це розділ прикладної математики, що вивчає моделі, методи, алгоритми, програмні й апаратні засоби перетворення інформації (шифрування) з метою приховування її змісту, запобігання зміні або несанкціонованого використання. Іншими словами, криптографія намагається знайти методи забезпечення секретності і/чи автентичності (достовірності) повідомленя.

Без криптографії принципово неможливо обійтися під час захисту даних, що передаються по відкритих електронних каналах зв'язку, а також там, де необхідно підтверджувати цілісність електронної інформації або доводити її авторство.

Криптографічний аналіз об'єднує математичні методи порушення конфіденційності й автентичності інформації без знання ключів.

Існує ряд суміжних, але таких, що не входять у криптологію, галузей знань. Так, забезпеченням приховання інформації в інформаційних масивах займається *стеганографія* [25]. Забезпечення цілісності інформації в умовах випадкового впливу знаходиться у віданні *теорії завадостійкого кодування* [30]. Нарешті, суміжною областю відносно до криптології є математичні методи стиснення інформації.

У даному навчальному посібнику будуть розглядатися тільки основи криптографії та частково криптографічний аналіз для визначення стійкості систем, що використовують криптографічні протоколи.

1.3.2. Історія розвитку криптології

В історії криптології умовно можна виділити чотири етапи [2, 6, 21, 24, 36]: наївний, формальний, науковий, комп'ютерний.

Для *наївної* криптології (до початку XVI ст.) характерне використання будь-яких, зазвичай примітивних, способів заплутування супротивника відносно змісту шифрованих повідомлень. На початковому етапі для захисту інформації використовувалися методи кодування і стеганографії, які споріднені, але не тотожні криптології.

Більшість із використовуваних шифрів зводилися до перестановки або моноалфавітної підстановки. Одним із перших зафіксованих прикладів є шифр Цезаря, який полягає у заміні кожної букви вихідного повідомлення на іншу, віддалену від неї в алфавіті на певне число позицій. Інший шифр, полібійський квадрат, авторство якого приписується грецькому письменникові Полібію, є загальною моноалфавітною підстановкою, яка проводиться за допомогою випадково заповненої алфавітом квадратної таблиці (для грецького алфавіту розмір становить 5×5). Кожна літера вихідного повідомлення замінюється на букву, що стоїть у квадраті знизу від неї.

Етап *формальної* криптології (кінець XV – початок XX ст.) пов'язаний із появою формалізованих і відносно стійких до ручного криптографічного аналізу шифрів. В європейських країнах це відбулося в епоху Відродження, коли розвиток науки і торгівлі викликав попит на надійні способи захисту інформації. Важлива роль на цьому етапі належить Леону Баттісті Альберті, італійському архітекторові, який одним із перших запропонував багатоалфавітну підстановку. Цей шифр, що отримав ім'я дипломата Блеза Віженера, полягав у послідовному "складанні" букв вихідного повідомлення з ключем (процедуру можна полегшити за допомогою спеціальної таблиці). Його робота "Трактат про шифр" (1466 р.) вважається першою науковою працею з криптології.

Однією з перших друкованих робіт, в якій узагальнено та сформульовано відомі на той момент алгоритми шифрування, є праця “Поліграфія” (1508 р.) німецького абата Йоганна Трисемуса. Йому належать два невеликих, але важливих відкриття: спосіб заповнення полібійського квадрата (перші позиції заповнюються за допомогою ключового слова, що легко запам’ятовується, інші – літерами алфавіту, що залишилися) і шифрування пар букв (біграм).

Простим, але стійким способом багатоалфавітної заміни (підстановки біграм) є шифр Плейфера, який був відкритий на початку XIX ст. Чарльзом Уітстоном. Уїтстону належить і важливе удосконалення – шифрування “подвійним квадратом”. Шифри Плейфера і Уітстона використовувалися аж до Першої світової війни, оскільки важко піддавалися ручному криптографічному аналізу.

У XIX ст. голландець Огюст Керкгоффс сформулював головну вимогу до криптографічних систем, яка залишається актуальною дотепер: *секретність шифрів має бути заснована на секретності ключа, але не алгоритму*.

Нарешті, останнім словом у донауковій криптології, яке забезпечило ще вищу криптографічну стійкість, а також дозволило автоматизувати (тобто механізувати) процес шифрування, стали роторні криптографічні системи.

Однією з перших подібних систем стала винайдена в 1790 році Томасом Джефферсоном, майбутнім президентом США, механічна машина. Багатоалфавітна підстановка за допомогою роторної машини реалізується варіацією взаємного положення обертових роторів, що обертаються, кожний з яких здійснює “прошиту” в ньому підстановку.

Практичного поширення роторні машини набули тільки на початку XX ст. Однією з перших використовуваних машин стала німецька Enigma, розроблена в 1917 році Едвардом Хеберном і вдосконалена Артуром Кірхом. Роторні машини активно використовувалися під час Другої світової війни. Окрім німецької машини Enigma використовувалися також пристрої Sigaba

(США), Typex (Великобританія), Red, Orange і Purple (Японія). Отже, роторні системи – вершина формальної криптографії, оскільки відносно просто реалізовували стійкі шифри. Успішні криптографічні атаки на роторні системи стали можливі тільки з появою електронних обчислювальних машин (ЕОМ) на початку 40-х рр. XX ст.

Головна відмінна риса *наукової* криптології (1930-60-і рр.) – поява криптографічних систем зі строгим математичним обґрунтуванням криптографічної стійкості. До початку 30-х років остаточно сформувалися розділи математики, що є науковою основою криптології: теорія ймовірності і математична статистика, загальна алгебра, теорія чисел, почали активно розвиватися теорія алгоритмів, теорія інформації, кібернетика. Своєрідним кроком стала робота Клода Шеннона “Теорія зв’язку в секретних системах” (1949 р.), яка підвела наукову базу під криптографію і криптографічним аналізом. Із того часу почали говорити про криптологію – як про науку перетворення інформації для забезпечення її секретності. Етап розвитку криптографії та криптографічного аналізу до 1949 року почали називати донауковою криптологією. Шеннон ввів поняття “розсіювання” та “перемішування”, обґрунтував можливість створення скільки завгодно стійких криптографічних систем.

У 1960-х роках провідні криптографічні школи підійшли до створення блокових шифрів, ще стійкіших у порівнянні з роторними криптографічними системами, проте вони допускали практичну реалізацію тільки у вигляді цифрових електронних пристроїв.

Комп’ютерна криптологія (з 1970-х рр.) зобов’язана своєю появою обчислювальним засобам із продуктивністю, достатньою для реалізації криптографічних систем, що забезпечують за великої швидкості шифрування на декілька порядків вищу криптографічну стійкість, ніж ручні й механічні шифри.

Першим класом криптографічних систем, практичне застосування яких стало можливо із появою потужних і компактних обчислювальних засобів,

стали блокові шифри. У 70-і роки XX ст. був розроблений американський стандарт шифрування DES – Data Encryption Standard (Стандарт шифрування даних), прийнятий в 1978 році. Один із його авторів, Хорст Фейстель (співробітник IBM), описав модель блокових шифрів, на основі якої було побудовано інші, стійкіше симетричні криптосистеми, зокрема стандарт шифрування ГОСТ 28147-89, який був розроблений ще в СРСР у 1989 році.

Із появою DES збагатився і криптографічний аналіз. Для атак на американський алгоритм було створено декілька нових видів криптографічного аналізу (лінійний, диференціальний тощо), практична реалізація яких знову ж таки стала можливою тільки з появою потужних обчислювальних систем.

У середині 70-х роках XX століття стався справжній прорив у сучасній криптографії – поява асиметричних криптографічних систем, які не вимагали передачі секретного ключа між сторонами. Тут, відправною точкою прийнято вважати працю, опубліковану Уітфілдом Діффі та Мартіном Хеллманом у 1976 році під назвою “Нові напрями в сучасній криптографії”, де вперше сформульовано принципи обміну шифрованою інформацією без обміну секретним ключем. Незалежно до ідеї асиметричних криптографічних систем підійшов Ральф Меркл. Декількома роками пізніше Рон Рівест, Аді Шамір і Леонард Адлеман відкрили систему RSA – першу практичну асиметричну криптографічну систему, стійкість якої була заснована на проблемі факторизації великих простих чисел. Асиметрична криптографія відкрила відразу декілька нових прикладних напрямів, зокрема системи електронного цифрового підпису та електронних грошей.

У 1980-90-х роках з’явилися зовсім нові напрями криптографії: ймовірнісне шифрування, квантова криптографія та інші. Усвідомлення їх практичної цінності ще попереду. Актуальним залишається і задача вдосконалення симетричних криптографічних систем. У цей самий період були розроблено нефейстелеві шифри (SAFER, RC6 та інші), а в 2000 році після відкритого міжнародного конкурсу прийнято новий Національний

стандарт шифрування США – AES (Advanced Encryption Standard – Удосконалений стандарт шифрування).

В Україні постійно проводиться робота щодо розробки національних криптографічних стандартів. Наразі розроблені, прийняті в якості національних та впроваджуються наступні стандарти криптографічного захисту інформації:

– ДСТУ 7624:2014 – Алгоритм симетричного блокового перетворення [7];

– ДСТУ 7564:2014 – Функція хешування [14];

– ДСТУ 8845:2019 – Алгоритм симетричного потокового перетворення [9];

– ДСТУ 8961:2019 – Алгоритми асиметричного шифрування та інкапсуляції ключів [15];

– ДСТУ 9041:2020 – Алгоритм шифрування коротких повідомлень, що ґрунтується на скручених еліптичних кривих Едвардса [16].

За оцінками фахівців, ці стандарти можуть розглядатися як квантово-захищені і можуть бути використані у постквантовий період.

1.3.3. Основні визначення

Способи та методи перетворення (шифрування) інформації з метою її захисту від незаконних користувачів (від несанкціонованого доступу) називаються *шифрами* [2, 24, 26, 29, 38, 40].

Шифрування (*зашифровування*) – це процес застосування шифру до інформації, що захищається, тобто перетворення інформації (відкритого тексту, даних) у шифроване повідомлення (шифрований текст, дані) за допомогою певних правил, що містяться в шифрі.

Розшифрування – це процес, обернений шифруванню, тобто перетворення шифрованого повідомлення в інформацію, що захищається, за допомогою певних правил, що містяться в шифрі.

Ключ – це найважливіший компонент шифру, що відповідає за вибір перетворення, яке використовується для зашифрування конкретного повідомлення. Зазвичай ключ є деякою буквеною або числовою послідовністю. Ця послідовність наче “налаштовує” алгоритм шифрування.

Дешифрування – це процес відновлення вихідного повідомлення на основі шифрованого без знання ключа, тобто методами криптографічного аналізу. У деяких джерелах поняття розшифрування і дешифрування вважаються синонімами.

Протокол – це послідовність кроків, які роблять дві або більше сторін для спільного вирішення завдання. Усі кроки слідують у порядку строгої послідовності, і жодний із них не можна зробити перш, ніж закінчиться попередній. Крім того, будь-який протокол передбачає участь принаймні, двох сторін. І, нарешті, протокол обов’язково призначений для досягнення певної мети.

Протокол, в основі якого лежить криптографічний алгоритм називається *криптографічним протоколом*. Проте метою криптографічного протоколу часто є не лише збереження інформації в таємниці від сторонніх. Учасники криптографічного протоколу можуть бути близькими друзями, у яких немає один від одного секретів, але й можуть бути настільки непримиренними ворогами, що кожен із них відмовляється повідомити іншому, наприклад, яке сьогодні число. Проте, їм може знадобитися поставити свої підписи під спільним договором або засвідчити свою особу. У цьому випадку криптографія потрібна, щоб запобігти або виявити підслуховування сторонніми особами, що не є учасниками протоколу, а також не допустити шахрайства. Тому, часто потрібно, щоб криптографічний протокол забезпечував наступне: його учасники не можуть зробити або дізнатися більше того, що визначено протоколом [4, 5, 26, 34, 37, 40].

Основою будь-якого криптографічного протоколу є так звані *криптографічні алгоритми*. Кожне перетворення однозначно визначається ключем і описується деяким криптографічним алгоритмом. Той самий

криптографічний алгоритм може застосовуватися для шифрування в різних режимах. Так реалізуються різні способи шифрування. Кожний режим шифрування має як свої переваги, так і недоліки. Тому вибір режиму залежить від конкретної ситуації. Під час розшифрування використовується криптографічний алгоритм, який у загальному випадку може відрізнятися від алгоритму, який застосовувався для шифрування повідомлення. Відповідно можуть відрізнятися *ключі шифрування* й *розшифрування*. Пару алгоритмів шифрування й розшифрування зазвичай називають *криптографічною системою*, а пристрої, що їх реалізують, – шифрувальними пристроями.

1.3.4. Узагальнена схема криптографічної системи

Криптографічна система – це система, яка реалізована програмно, апаратно або програмно-апаратно і здійснює криптографічне перетворення інформації з метою її захисту [3, 4, 21, 31, 40].

Припустимо, що відправник бажає надіслати повідомлення отримувачу. Більш того, цей відправник хоче надіслати своє повідомлення безпечно: він бажає бути впевнений, що зловмисник, який перехопив носій, не зможе дізнатися зміст повідомлення. Саме повідомлення, що передається, називається *відкритим* повідомленням (текстом, даними). Зміна виду повідомлення з метою приховання його суті називається *шифруванням*. Зашифроване повідомлення називається *шифротекстом* (*шифрограмою*, *криптограмою*). Процес перетворення зашифрованого повідомлення у відкрите повідомлення називається *розшифруванням* (*дешифруванням* під час криптографічного аналізу).

Узагальнену схему криптографічної системи, що забезпечує шифрування інформації і її розшифрування, показано на рис. 1.1.

Позначимо відкритий текст як *M* (від *message* – повідомлення) або *P* (від *plaintext* – відкритий текст). Це може бути потік бітів, текстовий файл, бітове зображення, оцифрований звук, цифрове відеозображення та ін.

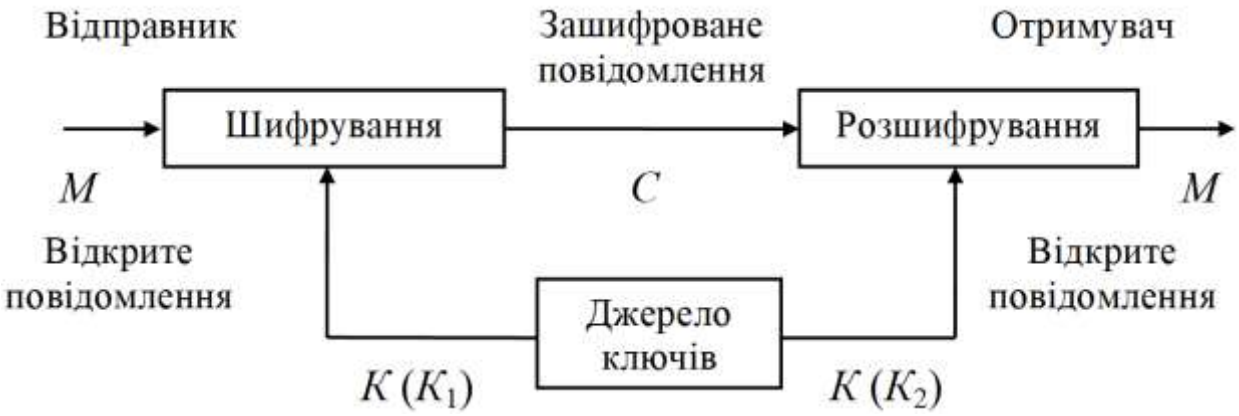


Рис. 1.1. Узагальнена схема криптографічної системи

Позначимо шифротекст як *C* (від *ciphertext* – зашифрований текст). Це також двійкові дані, іноді того самого розміру, що й *M*, іноді більшого. Якщо шифрування супроводжується стисненням, *C* може бути менше *M*. Однак саме шифрування не забезпечує стиснення інформації. *Функція шифрування E* діє на відкритий текст, створюючи зашифрований текст:

$$C = E(M). \tag{1.1}$$

Процес відновлення відкритого тексту із зашифрованого тексту називається *розшифруванням* і виконується за допомогою функції розшифрування *D*:

$$M = D(C). \tag{1.2}$$

Оскільки змістом шифрування і подальшого розшифрування повідомлення є відновлення первинного відкритого тексту, то повинна виконуватися така рівність:

$$M = D(E(M)).$$

Криптографічний алгоритм, що також називається шифром, є математичною функцією (наприклад, функції *E* і *D* у виразах (1.1) і (1.2)), яка використовується для шифрування й розшифрування.

Якщо безпека алгоритму заснована на збереженні самого алгоритму в таємниці, це *обмежений алгоритм*. Обмежені алгоритми становлять тільки історичний інтерес, але вони абсолютно не відповідають сьогоднішнім стандартам. Велика або така, що змінюється, група користувачів не може використовувати такі алгоритми, оскільки кожного разу, коли користувач залишає групу, її члени повинні переходити на інший алгоритм. Алгоритм повинен бути замінений, якщо будь-хто із зовні випадково дізнається секрет.

Сучасна криптографія вирішує ці проблеми за допомогою ключа *K*. *Ключ* – це конкретний секретний стан деяких параметрів алгоритму криптографічного перетворення даних, що забезпечує вибір тільки одного варіанту з усіх можливих для цього алгоритму. Множину можливих ключів називають *простором ключів*.

З урахуванням використання ключа, функції шифрування (1.1) і розшифрування (1.2) будуть мати такий вигляд:

$$C = E_K(M) \tag{1.3}$$

та

$$M = D_K(C), \tag{1.4}$$

при цьому має виконуватися:

$$M = D_K(E_K(M)). \tag{1.5}$$

Для деяких алгоритмів під час шифрування й розшифрування використовуються різні ключі. У цьому випадку вирази (1.3) і (1.4) будуть мати вигляд:

$$C = E_{K_1}(M),\ M = D_{K_2}(C), \tag{1.6}$$

а рівність (1.5):

$$M = D_{K_2}(E_{K_1}(M)).$$

В якості інформації, що підлягає шифруванню й розшифруванню, будемо розглядати тексти (повідомлення), побудовані на деякому алфавіті. Під цими термінами розуміється наступне: *алфавіт* – кінцева множина знаків, що використовується для шифрування інформації. *Текст* (*повідомлення*) – упорядкований набір з елементів алфавіту.

Як приклади алфавітів, використовуваних у сучасних інформаційних системах, можна навести такі:

– алфавіт Z_{26} – 26 букв англійського алфавіту (виключаючи пробіл);

– алфавіт Z_{33} – 33 літери українського алфавіту (виключаючи пробіл);

– алфавіт Z_{256} – 256 символів, що входять у стандартний код ASCII;

– алфавіт Z_{16} – 16 символів $\{0, 1, \ldots, F\}$ шістнадцяткового алфавіту;

– алфавіт Z_2 – 2 символу $\{0, 1\}$ двійкового алфавіту.

Безпека алгоритмів, що описуються виразами (1.3), (1.4) і (1.6), повністю заснована на ключах, а не на деталях алгоритму. Це означає, що алгоритм може бути опублікований і проаналізований. Продукти, що використовують цей алгоритм, можуть широко тиражуватися. Як було визначено раніше фундаментальне правило криптографічного аналізу, уперше сформульоване голландцем О. Керкгоффсом ще в XIX столітті, яке полягає в тому, що стійкість шифру (криптографічної системи) повинна визначатися тільки ступенем секретності ключа. Іншими словами, *правило Керкгоффса* полягає в тому, що весь алгоритм шифрування й розшифрування, крім секретного ключа, відомий криптографічному аналітику супротивника. Це обумовлено тим, що криптографічна система, яка реалізує сімейство криптографічних перетворень, зазвичай розглядається як відкрита система. Такий підхід відображає дуже важливий принцип технології захисту інформації: захищеність системи не повинна залежати від секретності чого-небудь такого, що неможливо швидко змінити в разі витоку секретної (конфіденційної) інформації. Зазвичай криптографічна система є сукупністю апаратних і програмних засобів, яку можна змінити лише за

значних витратах часу і коштів, тоді як ключ є легко змінюваним об’єктом. Саме тому стійкість криптографічної системи повинна визначатися тільки ступенем секретності ключа.

1.4. Основи теорії засекреченого зв’язку

Перш ніж перейти до розгляду криптографічних алгоритмів, необхідно приділити увагу питанням, які в рамках криптографії давно визнаються класичними, а саме основам побудови систем засекреченого зв’язку.

Під *системою засекреченого зв’язку* будемо розуміти систему передачі даних, у якій зміст інформації, що передається, приховується за допомогою криптографічних перетворень [41]. При цьому сам факт передачі інформації не приховується. В основі кожної системи засекреченого зв’язку – використання алгоритмів шифрування як основного засобу збереження конфіденційності.

Клод Шеннон розглянув модель (рис. 1.2), в якій джерело повідомлень породжує відкритий текст *M*. Джерело ключів генерує ключ *K*. Пристрій шифрування перетворює відкритий текст *M* за допомогою ключа *K* в зашифрований текст *C*: $C = E_K(M)$. Пристрій розшифрування, отримавши зашифроване повідомлення *C*, виконує обернену операцію: $M = D_K(C)$.

Завданням криптографічного аналітика противника є отримання відкритого тексту і/або ключа на основі аналізу зашифрованого тексту.

Шеннон розглянув питання теоретичної та практичної секретності. Для визначення теоретичної секретності Шеннон сформулював наступні питання:

1. Наскільки стійкою є система, якщо криптографічний аналітик противника не обмежений у часі та має всі необхідні засобами для аналізу шифрограми?

2. Чи має шифрограма єдиний розв’язок?

3. Який обсяг зашифрованого тексту необхідно перехопити криптографічному аналітику, щоб розв’язок став єдиним?

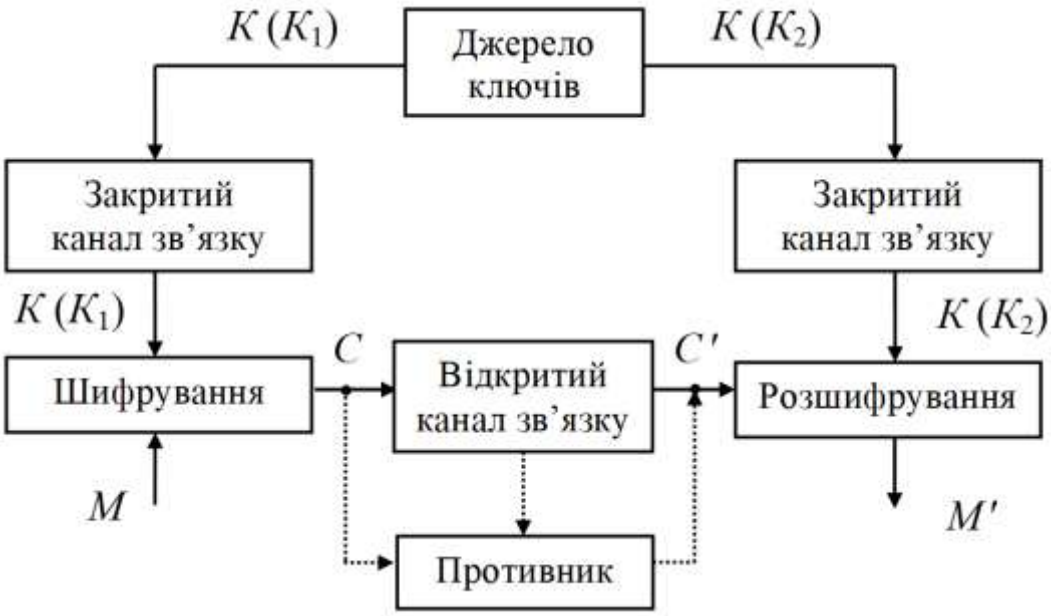


Рис. 1.2. Модель системи передачі зашифрованих повідомлень

Для відповіді на ці питання Шеннон увів поняття досконалої секретності за допомогою наступної умови: для усіх C апостеріорна ймовірність дорівнює апріорній ймовірності, тобто перехоплення зашифрованого повідомлення не дає криптографічному аналітику противника ніякої інформації. За теоремою Байеса:

$$p(M, C) = p(M) \cdot p(C / M) = p(C) \cdot p(M / C), \qquad (1.7)$$

де $p(M)$ – апріорна ймовірність повідомлення M;

$p(C / M)$ – умовна ймовірність шифрограми C за умови, що обрано повідомлення M, тобто сума ймовірностей усіх тих ключів, які приводять повідомлення M у шифрограму C;

$p(C)$ – ймовірність отримання шифрограми C;

$p(M / C)$ – апостеріорна ймовірність повідомлення M за умови, що перехоплено шифрограму C.

Із виразу (1.7) випливає, що:

$$p(M/C) = \frac{p(M)\cdot p(C/M)}{p(C)} \qquad (1.8)$$

Для досконалої секретності криптографічної системи значення $p\,(M/C)$ і $p\,(M)$ повинні бути рівні для всіх M і C, тобто $p\,(M/C) = p\,(M)$. Отже, з аналізу (1.8), повинна виконуватися одна з рівностей:

– $p\,(M) = 0$ (цей розв'язок слід відкинути, оскільки потрібно, щоб рівність здійснювалася за будь-яких значень $p\,(M)$);

– $p\,(C/M) = p\,(C)$ для будь-яких M і C.

Навпаки, якщо $p\,(C/M) = p\,(C)$, то з (1.8) випливає, що $p\,(M/C) = p\,(M)$, і система досконало секретна. Отже, можна сформулювати наступне: необхідна й достатня умова для досконалої секретності полягає у тому, що $p\,(C/M) = p\,(C)$ для усіх M і C, тобто $p\,(C/M)$ не повинна залежати від M.

Іншими словами, повна ймовірність усіх ключів, що переводять повідомлення M_i у дане зашифроване повідомлення C, дорівнює повній ймовірності усіх ключів, що переводять повідомлення M_j у те саме зашифроване повідомлення C для всіх M_i, M_j і C.

Далі, має існувати принаймні стільки само зашифрованих повідомлень C, скільки й повідомлень M, оскільки для фіксованого i відображення E_i дає взаємно однозначну відповідність між усіма M і деякими з C. Для досконало секретних систем для кожного із цих C і будь-якого M

$$p\,(C/M) = p\,(C) \neq 0.$$

Отже, знайдеться принаймні один ключ, що відображує вибране M у будь-яке з C. Але всі ключі, що відображують фіксоване M у різні C, повинні бути різними. Тому, кількість різних ключів повинна бути не менше кількості повідомлень M.

Як показує наступний приклад, можна отримати досконалу секретність,

коли кількість повідомлень точно дорівнює кількості ключів. Нехай M_i пронумеровані числами від 1 до n, так само як і C_i, і нехай використовуються n ключів. Тоді:

$$E_i\,(M_j) = C_s$$

де $s = (i + j) \bmod n$.

У цьому випадку справедливою є рівність $p\,(M/C) = 1/n = p\,(C)$ і система є досконало секретною. Один приклад такої системи показано на рис. 1.3, де $s = (i + j - 1) \bmod 5$.

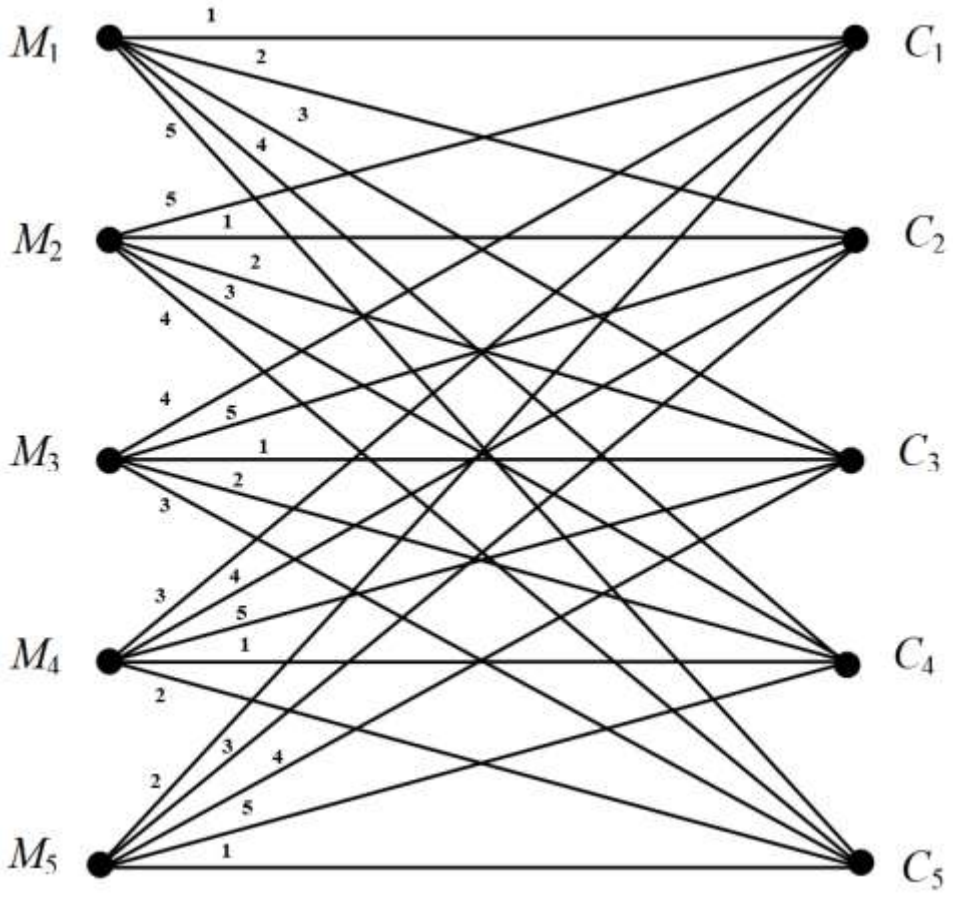


Рис. 1.3. Досконало секретна криптографічна система

Досконало секретні системи, у яких кількість зашифрованих повідомлень дорівнює кількості вихідних повідомлень та кількості ключів, характеризуються наступними двома властивостями:

– кожне M зв’язується з кожним C тільки однією лінією;

– всі ключі рівноймовірні.

Таким чином, матричне представлення такої системи є “латинський квадрат”.

У роботі Клода Шеннона “Математична теорія зв’язку” показано, що кількісно інформацію зручно вимірювати за допомогою ентропії. Якщо є деяка сукупність можливостей з ймовірностями $p_1, p_2, \ldots, p_n$, то ентропія представляється виразом

$$H = -\sum p_i \cdot \log p_i.$$

Секретна система включає в себе два статистичних вибори: вибір повідомлення і вибір ключа. Можна вимірювати кількість інформації, яка створюється при виборі повідомлення, через $H(M)$

$$H(M) = -\sum P(M) \cdot \log P(M),$$

де додавання виконується за всіма можливими повідомленнями.

Аналогічно, невизначеність, яка пов’язана з вибором ключа, представляється виразом

$$H(K) = -\sum P(K) \cdot \log P(K).$$

У досконало секретних системах описаного вище типу кількість інформації в повідомленні дорівнює щонайбільше $\log n$ (ця величина досягається для рівноймовірних повідомлень). Ця інформація може бути прихована повністю лише тоді, коли невизначеність ключа не менше $\log n$. Це є першим прикладом загального принципу: існує межа, яку не можна перевершити при заданій невизначеності ключа – кількість невизначеності, яке може бути введене у розв’язок, не може бути більше, ніж невизначеність ключа.

Становище дещо ускладнюється, якщо число повідомлень нескінченне. Припустимо, наприклад, що повідомлення породжуються відповідним марківським процесом у вигляді нескінченної послідовності букв. Ясно, що ніякий кінцевий ключ не дає досконалої секретності. Припустимо тоді, що джерело ключів породжує ключ аналогічним чином, тобто як нескінченну послідовність символів. Також припустимо, що для шифрування і розшифрування повідомлення довжиною L_M потрібно тільки певна довжина ключа L_K. Нехай логарифм числа букв у алфавіті повідомлень дорівнює R_M, а той самий логарифм для ключа – R_K. Тоді з міркувань для кінцевого випадку, очевидно, випливає, що для досконалої секретності необхідно виконання нерівності

$$R_M \cdot L_M \leq R_K \cdot L_K.$$

Ці висновки зроблені у припущенні, що апріорні ймовірності повідомлень невідомі або довільні. У цьому випадку ключ, необхідний для того, щоб мала місце досконала секретність, залежить від повного числа можливих повідомлень. Можна було б очікувати, що якщо у просторі повідомлень є фіксовані відомі статистичні зв'язки, так що є визначена швидкість створення повідомлень R у сенсі, прийнятому в "Математичній теорії зв'язку", то необхідний обсяг ключа можна було б знизити у середньому в R/R_M разів, і це дійсно вірно. Насправді, повідомлення можна пропустити через перетворювач, який усуває надмірність і зменшує середню довжину повідомлення якраз у стільки разів. Потім до результату можна застосувати шифр Вернама. Очевидно, що обсяг ключа, що використовується на букву повідомлення, статистично зменшується на множник R/R_M, і в цьому випадку джерело ключів і джерело повідомлень в точності узгоджені – один біт ключа повністю приховує один біт інформації повідомлення. За допомогою методів, що використовуються у праці "Математична теорія зв'язку", легко показати, що це найкраще, чого можна досягти.

1.5. Класифікація методів шифрування повідомлень

Класична або одноключова криптографія спирається на використання симетричних алгоритмів шифрування, у яких шифрування й розшифрування відрізняються тільки порядком виконання і напрямком деяких кроків. Ці алгоритми використовують однаковий секретний елемент (ключ), і друга дія (розшифрування) є простим оберненням першої (шифрування). Тому, зазвичай кожний з учасників обміну може як зашифрувати, так і розшифрувати повідомлення. Схематичну структуру такої системи показано на рис. 1.1.

На передавальній стороні є джерело повідомлень *M* і джерело ключів *K*. Джерело ключів вибирає конкретний ключ *K* серед усіх можливих ключів даної системи. Цей ключ *K* передається деяким способом приймаючій стороні, причому передбачається, що його неможливо перехопити, наприклад, ключ передається спеціальним кур'єром. Джерело повідомлень формує деяке повідомлення *M*, яке потім зашифровується з використанням вибраного ключа *K*. В результаті процедури шифрування виходить зашифроване повідомлення *C* (зване також *криптограмою*). Далі криптограма *C* передається по каналу зв'язку. Оскільки канал зв'язку є відкритим (незахищеним), наприклад, радіоканал або комп'ютерна мережа, то передане повідомлення може бути перехоплено противником. На приймаючій стороні криптограму *C* за допомогою ключа *K* розшифровують і отримують вхідне повідомлення М.

Через велику надмірність природних мов безпосередньо в зашифроване повідомлення надзвичайно складно внести осмислену зміну, тому класична криптографія забезпечує також захист від нав'язування хибних даних. Якщо ж природної надмірності виявляється недостатньо для надійного захисту повідомлення від модифікації, надмірність може бути штучно збільшена шляхом додавання до повідомлення спеціальної контрольної комбінації, яку називають *імітовставкою*.

Відомі різні методи шифрування (рис. 1.4). На практиці часто використовуються алгоритми перестановки, підстановки, а також комбіновані методи.

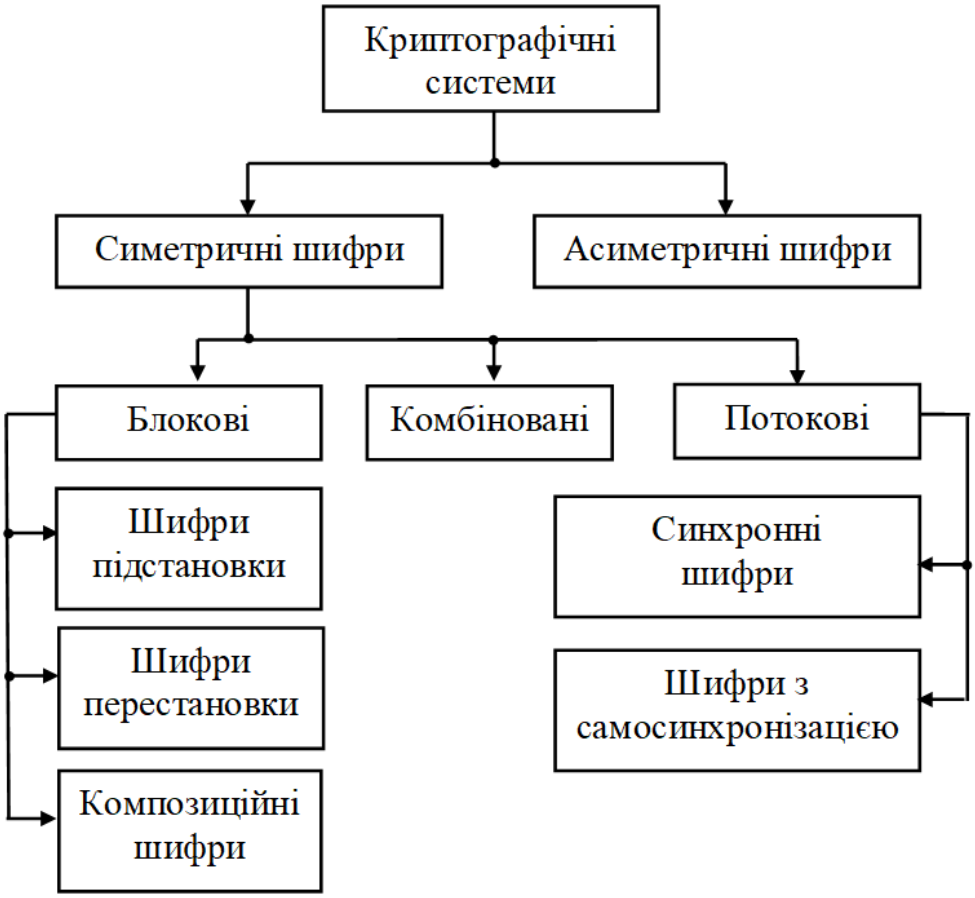


Рис. 1.4. Класифікація методів шифрування повідомлень

Криптографічні системи за типом алгоритму шифрування поділяються на дві великі групи (рис. 1.4): *симетричні* й *асиметричні*. Крім того, алгоритми діляться за типом перетворення, за способом обробки інформації.

У симетричних криптографічних системах шифрування й розшифрування проводяться за допомогою однакового ключа. І відповідно цей ключ необхідно зберігати в таємниці (звідси інша назва симетричних криптографічних систем – криптографічні *системи із секретним ключем*).

В асиметричних криптографічних системах існують два різні ключі: один використовується для шифрування, який ще називають *відкритим*, інший – для розшифрування, який називають *закритим* [12]. Головна

відмінність асиметричних криптосистем полягає у тому, що навіть той, хто за допомогою відкритого ключа зашифрував повідомлення, не зможе його самостійно розшифрувати без секретного ключа. Тому ці системи називаються асиметричними, або *системами з відкритим ключем*.

Гібридними прийнято називати криптографічні системи, що поєднують обидва типи криптографічних систем, у них, як правило, текст повідомлення зашифровується з використанням симетричної криптографічної системи, а секретний ключ, використаний симетричною криптографічною системою, зашифровується з використанням асиметричної криптографічної системи.

За типом обробки вхідної інформаційної послідовності криптографічні системи діляться на *потокові,* у яких кожен символ або біт відкритого тексту перетворюється в символ зашифрованого тексту потоком, і *блокові,* у яких повідомлення обробляються у вигляді блоків певної довжини.

У методах *перестановки* символи вхідного тексту міняються місцями один з одним за певним правилом. У методах *підстановки* (або заміни) символи відкритого тексту замінюються деякими еквівалентами зашифрованого тексту. По суті, шифри перестановки й підстановки є цеглинками, з яких будуються різні інші більш стійкі шифри.

З метою підвищення надійності шифрування текст, зашифрований за допомогою одного методу, може бути ще раз зашифрований за допомогою іншого методу. У такому випадку виходить *комбінований* або *композиційний шифр*. Застосовувані на практиці в теперішній час блокові або потокові симетричні шифри також належать до комбінованих, оскільки в них використовується декілька операцій для шифрування повідомлення.

Основна відмінність сучасної криптографії від докомп’ютерної полягає у тому, що раніше криптографічні алгоритми оперували символами природних мов, наприклад, буквами англійської або української мови. Ці букви переставлялися або замінювалися іншими за певним правилом. У сучасних криптографічних алгоритмах використовуються операції над двійковими знаками, тобто над нулями й одиницями. У даний час основними

операціями під час шифрування також є перестановка або підстановка, причому для підвищення надійності шифрування ці операції застосовуються разом (комбінуються) і багато разів циклічно повторюються.

Ідея, що лежить в основі *складених*, або *композиційних шифрів*, полягає в побудові криптографічно стійкої системи шляхом багаторазового застосування відносно простих криптографічних перетворень. К. Шеннон запропонував використовувати в якості перетворення підстановки (substitution) і транспозиції (permutation). Багаторазове використання цих перетворень дозволяє забезпечити дві властивості, які повинні бути притаманні стійким шифрами*: розсіювання* (diffusion) і *перемішування* (confusion).

Розсіювання передбачає поширення впливу одного знака відкритого тексту, а також одного знака ключа на значну кількість знаків зашифрованого повідомлення. Наявність у шифрі цієї властивості, з одного боку, дозволяє приховувати статистичну залежність між знаками відкритого тексту, інакше кажучи, перерозподілити надмірність вхідної мови за допомогою поширення її на весь текст, а з іншого – не дозволяє відновити невідомий ключ частинами. Наприклад, звичайна перестановка символів дозволяє приховати частоти появи біграм, триграм, тощо.

Мета перемішування – зробити якомога складнішою залежність між ключем і зашифрованим повідомленням. Криптографічний аналітик на основі статистичного аналізу перемішаного тексту не повинен отримати будь-яку кількість інформації про використаний ключ. Зазвичай перемішування здійснюється за допомогою підстановки. Застосування розсіювання й перемішування окремо не забезпечує необхідну стійкість, стійка криптографічна система виходить тільки внаслідок їх спільного використання.

1.6. Криптографічний аналіз

Як уже зазначалося, криптографія – наука й мистецтво створення секретних кодів, криптографічний аналіз – наука й мистецтво зламу цих кодів. На додаток до вивчення методів криптографії необхідно також вивчити методи криптографічного аналізу. Це необхідно не для того, щоб зламувати коди інших людей, а щоб оцінити вразливі місця своїх криптографічних систем. Вивчення криптографічного аналізу допоможе створювати кращі секретні коди. Є чотири загальні типи атак криптографічного аналізу, які показані на рис. 1.5 [4, 19, 24, 36].

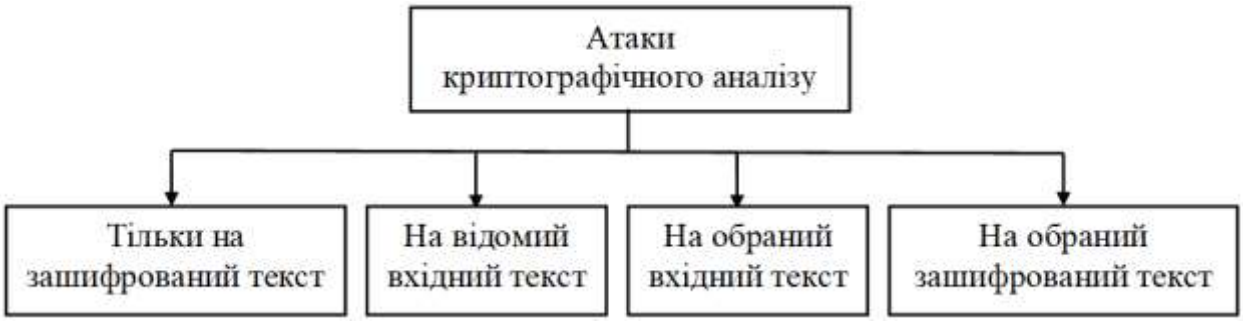


Рис. 1.5. Типи атак криптографічного аналізу

Атака тільки на зашифрований текст

В атаці тільки на зашифрований текст криптографічний аналітик має доступ тільки до деякого зашифрованого тексту. Він намагається знайти відповідний ключ і вхідний текст. При цьому, згідно з припущенням, криптографічний аналітик знає алгоритм і може перехопити зашифрований текст. Атака тільки на зашифрований текст – найімовірніша, тому що криптографічному аналітику для неї потрібен тільки сам зашифрований текст. Шифр повинен серйозно перешкоджати цьому типу атаки і не дозволити дешифрування повідомлення противником. Рис. 1.6 ілюструє процес атаки.

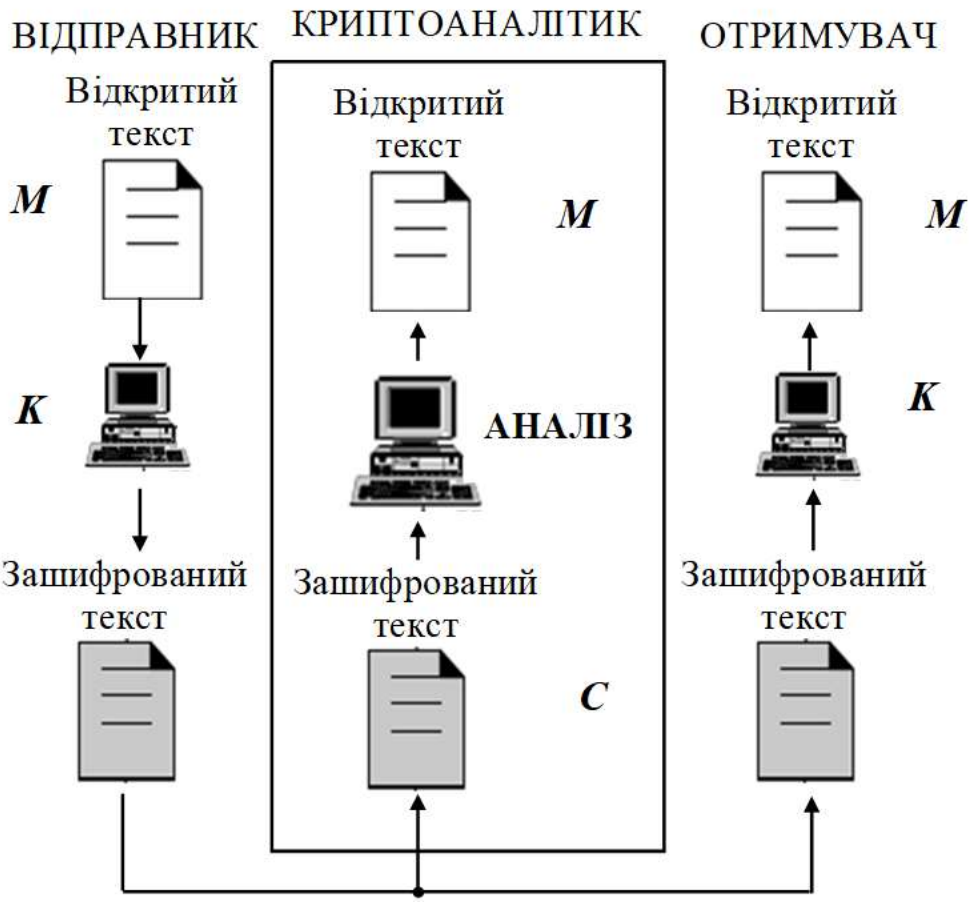


Рис. 1.6. Пояснення процесу атаки тільки на зашифрований текст

В атаці тільки на зашифрований текст можуть використовуватися різні методи. Розглянемо деякі з них.

***Атака* "*грубої сили*"**

За допомогою методу "грубої сили", або методу вичерпного ключового пошуку, криптографічний аналітик намагається використовувати всі можливі ключі. Припускаємо, що він знає алгоритм і знає множину ключів (список можливих ключів). Шляхом використання цього методу перехоплюється зашифрований текст і задіються всі можливі ключі, поки не вийде вхідний текст. Проведення атаки "грубої сили" було в минулому важким завданням; сьогодні за допомогою комп'ютера це стало простіше. Щоб запобігти атакам цього типу, число можливих ключів повинне бути дуже великим.

Статистична атака

Криптографічний аналітик може отримати вигоду з деяких властивих мові вхідного тексту характеристик, щоб почати статистичну атаку. Наприклад, відомо, що буква *E* – найбільш часто використовувана буква в англійському тексті. Криптографічний аналітик знаходить найбільш часто використовуваний символ у зашифрованому тексті та приймає, що це відповідний символ вхідного тексту – *E*. Після визначення декількох пар аналітик може знайти ключ і розшифрувати повідомлення. Щоб запобігти атакам цього типу, шифр повинен приховувати характеристики мови.

Атака за зразком

Деякі шифри приховують характеристики мови, але створюють деякі зразки в зашифрованому тексті. Криптографічний аналітик може використовувати атаку за зразком, щоб зламати шифр. Тому важливо використовувати шифри, які зробили б зашифрований текст, що проглядається, невизначеним (абстрактним) наскільки це можливо.

Атака на відомий вхідний текст

При атаці на відомий вхідний текст криптографічний аналітик має доступ до деяких пар “відкритий/зашифрований текст” на додаток до перехопленого зашифрованого тексту, який він хоче зламати, як показано на рис. 1.7.

Пари відкритого/зашифрованого тексту були зібрані раніше. Наприклад, відправник передав секретне повідомлення отримувачу, але пізніше він відкрив зміст повідомлення стороннім. Криптографічний аналітик зберігав і зашифрований текст, і відкритий текст, щоб використовувати їх, коли знадобиться зламати наступне секретне повідомлення від відправника до отримувача, припускаючи, що відправник не змінить свій ключ. Криптографічний аналітик використовує відношення між попередньою парою, щоб аналізувати поточний зашифрований текст.

Ті самі методи, що використовуються в атаці тільки на зашифрований текст, можуть бути застосовані і тут. Але цю атаку здійснити простіше, тому що криптографічний аналітик має більше інформації для аналізу. Однак може статися, що відправник змінив свій ключ або не розкривав змісту будь-яких попередніх повідомлень, – тоді подібна атака стане неможливою.

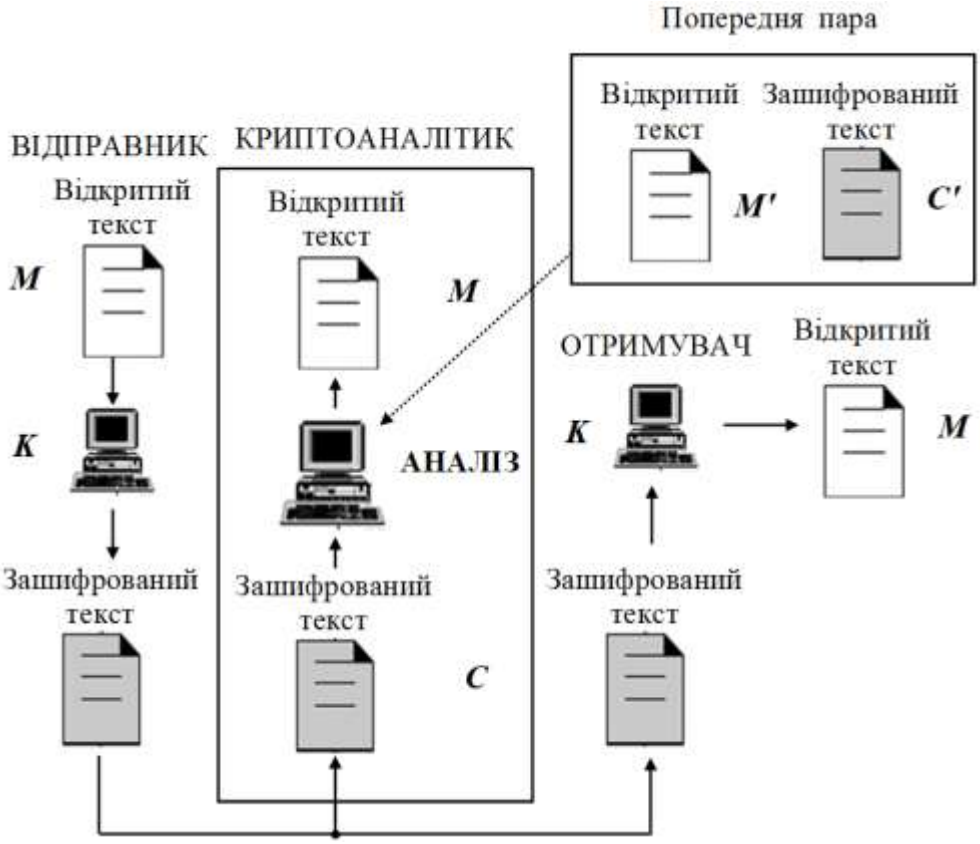


Рис. 1.7. Пояснення процесу атаки на відомий вхідний текст

Атака на обраний вхідний текст

Атака на обраний вхідний текст подібна атаці на відомий вхідний текст, але пари “відкритий/зашифрований текст” були обрані і виготовлені самим нападником. Цей процес ілюструє рис. 1.8.

Це може статися, наприклад, якщо криптографічний аналітик має доступ до комп’ютера відправника. Він вибирає певний відкритий текст і створює за допомогою комп’ютера зашифрований текст. Звичайно, він не має ключа, тому що ключ зазвичай міститься в програмному забезпеченні, що

використовується відправником. Цей тип атаки набагато простіше здійснити, але він найменш імовірний, оскільки має на увазі занадто багато “*якщо*”.

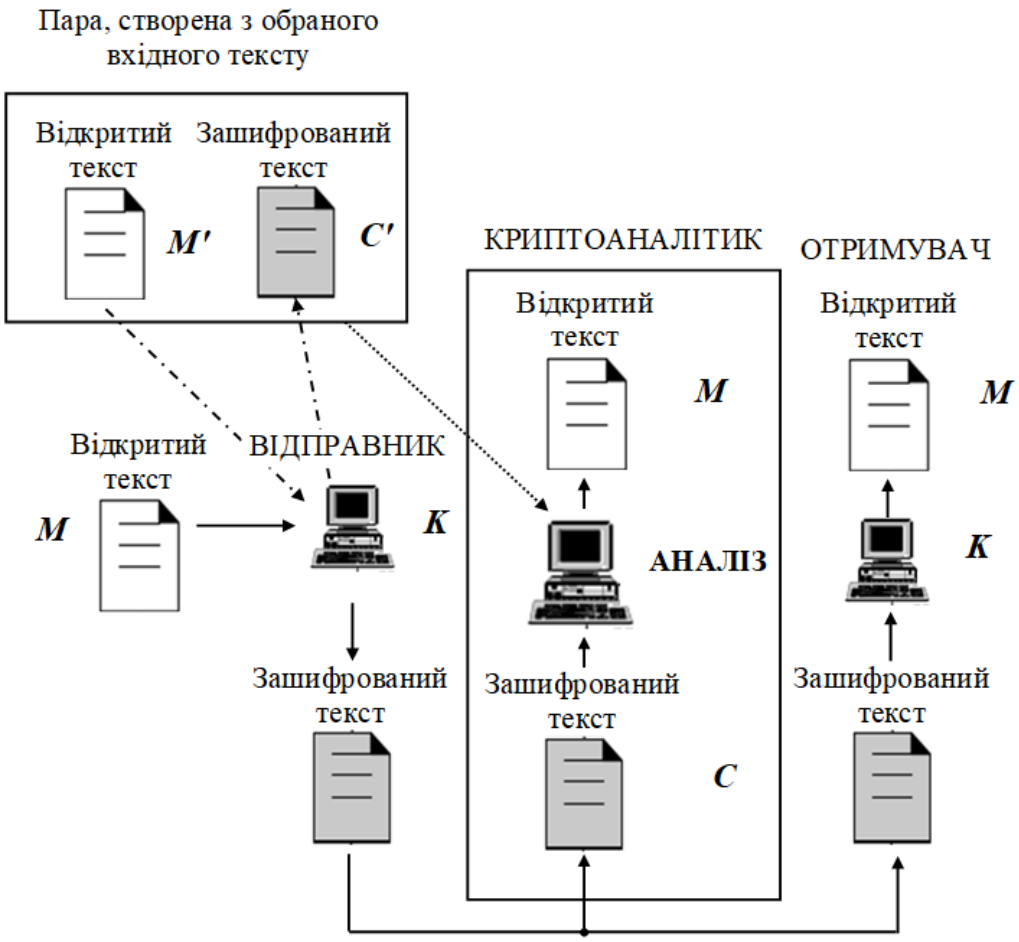


Рис. 1.8. Пояснення процесу атаки на обраний вхідний текст

Атака на обраний зашифрований текст

Атака на обраний зашифрований текст подібна атаці на обраний вхідний текст, за винятком того, що криптографічний аналітик вибирає певний зашифрований текст і розшифровує його, щоб сформувати пару “зашифрований/відкритий текст” (це трапляється, коли аналітик має доступ до комп’ютера отримувача). Рис. 1.9 показує цей процес.

Криптографічні атаки за використання сучасних криптографічних систем буде розглянуто під час їх вивчення в наступних розділах.

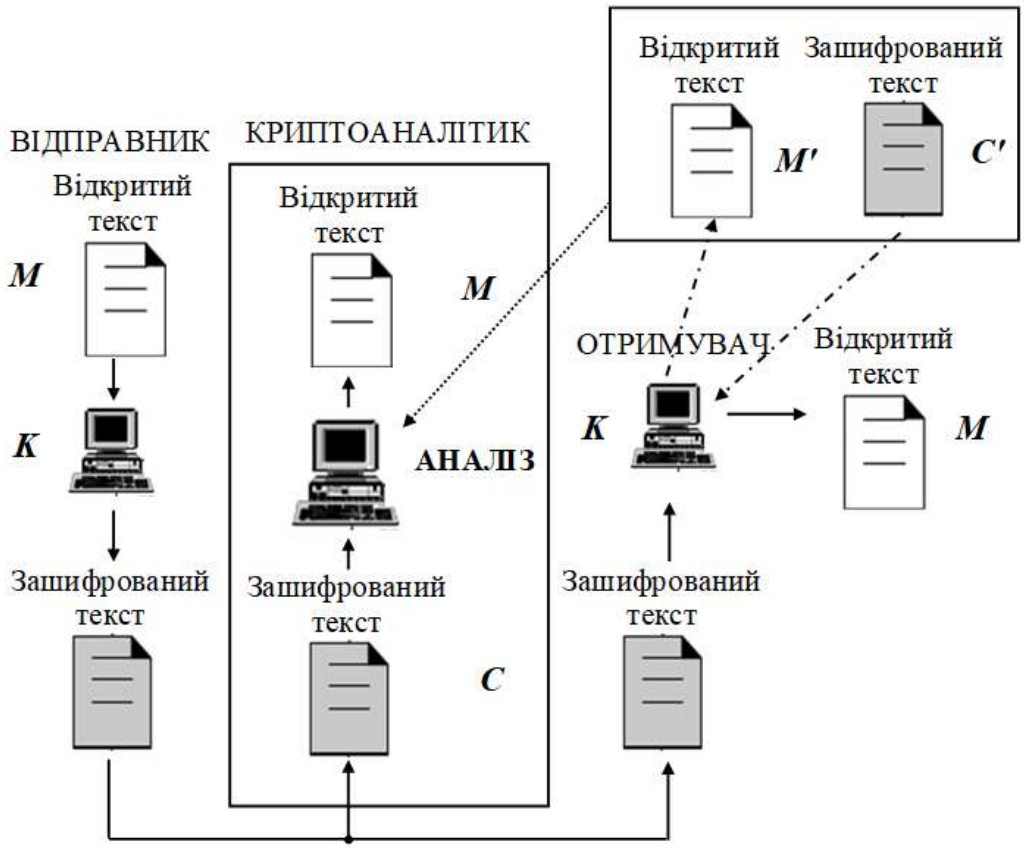


Рис. 1.9. Пояснення процесу атаки на обраний зашифрований текст

Контрольні питання до розділу 1

1. Назвіть проблеми, для вирішення яких можуть використовуватися криптографічні методи.

2. У чому полягає забезпечення конфіденційності, цілісності та достовірності інформаційних ресурсів?

3. Дайте визначення: криптологія, криптографія та криптоаналіз.

4. Дайте визначення: шифр, ключ шифрування, розшифрування, дешифрування, криптографічний алгоритм, криптографічна система.

5. Дайте визначення криптографічного протоколу.

6. Поясніть правило Керкгоффса для криптографії.

7. Сформулюйте необхідну й достатню умови для досконалої секретності криптографічної системи.

8. Дайте класифікацію методів шифрування повідомлень.

9. Чим відрізняється симетрична криптосистема від асиметричної?

10. Дайте пояснення сутності розсіювання даних у процесі їх шифрування.

11. Що є метою перемішування даних у процесі їх шифрування?

12. Що таке криптографічна атака?

13. Які типи криптографічних атак існують?

14. Дайте характеристику атаки тільки на зашифрований текст. Поясніть сутність атаки “грубої сили”.

15. Дайте характеристику атаки тільки на зашифрований текст. Поясніть сутність статистичної атаки.

16. Дайте характеристику атаки тільки на зашифрований текст. Поясніть сутність атаки за зразком.

17. Дайте характеристику атаки на відомий вхідний текст.

18. Дайте характеристику атаки на обраний вхідний текст.

19. Дайте характеристику атаки на обраний зашифрований текст.

РОЗДІЛ 2
ІСТОРИЧНІ ШИФРИ

Традиційні шифри з симетричним ключем можна розділити на дві великі категорії: *шифри підстановки* та *шифри перестановки*. У шифрі підстановки один символ у зашифрованому тексті замінюється на інший символ; у шифрі перестановки – міняються місцями позиції символів у початковому тексті.

2.1. Шифри підстановки

Шифр підстановки замінює один символ іншим. Якщо символи в початковому тексті – символи алфавіту, то одна буква замінюється іншою. Наприклад, можна замінити букву *B* буквою *W* використовуючи англійський алфавіт, а букву *M* – буквою *H*. Якщо символи – цифри (від 0 до 9), то можна, наприклад, замінити 5 на 1, а 3 на 8. Шифри підстановки можуть бути розбиті на дві категорії: *моноалфавітні* (*одноалфавітні*) і *багатоалфавітні* шифри.

Для шифрів підстановки довжина алфавіту відкритого тексту (даних) (n_M) і довжина алфавіту зашифрованого тексту (даних) (n_C) однакова, тобто $n_M = n_C$.

При використанні цих шифрів букви алфавіту зручно ототожнювати з їх порядковими номерами.

2.1.1. Моноалфавітні шифри підстановки

У моноалфавітних шифрах підстановки буква (або символ) у початковому тексті завжди змінюється на ту саму букву (або символ) у зашифрованому тексті незалежно від її (його) позиції в тексті. Наприклад, якщо алгоритм визначає, що буква *B* в початковому тексті змінюється на

букву *W*, то при цьому кожна буква *B* змінюється на букву *W*. Іншими словами, букви в початковому тексті й зашифрованому тексті перебувають у відношенні один до одного.

Приклад 2.1. Початковий текст *hello* в результаті зашифрування перетворився на *KHOOR*. Визначити, яким шифром забезпечувалося шифрування. Використовуємо рядкові букви, щоб показати початковий текст, і заголовні букви (символи верхнього регістру), щоб отримати зашифрований текст.

Рішення. Оскільки обидві букви *l* зашифровані як *O*, то використовувався моноалфавітний шифр.

Приклад 2.2. Початковий текст *hello* в результаті зашифрування перетворився на *ABNZF*. Визначити, яким шифром забезпечувалося шифрування.

Рішення. Шифр не є моноалфавітним, тому що кожна буква *l* зашифрована різними символами. Перша буква *l* зашифровано як *N*; друга – як *Z*.

Адитивні шифри підстановки

Найпростіший моноалфавітний шифр – *адитивний шифр*, його іноді називають шифром *зсуву*, а іноді – шифром *Цезаря*, але термін адитивний шифр краще показує його математичний сенс. Припустимо, що початковий текст складається з маленьких букв (від *a* до *z*), а зашифрований текст складається з заголовних букв (від *A* до *Z*). Щоб забезпечити застосування математичних операцій до вихідного та зашифрованого текстів, надамо кожній букві числове значення (для нижнього і верхнього регістру), як це показано на рис. 2.1 для англійської мови (26 букв) і на рис. 2.2 для української мови (33 букви).

На рис. 2.1 кожному символу (нижній регістр або верхній регістр) поставлено у відповідність ціле число з Z_{26}. Ключ засекречування між відправником і отримувачем – також ціле число в Z_{26}. Алгоритм шифрування

додає ключ до символу вихідного тексту; алгоритм розшифрування віднімає ключ із символу зашифрованого тексту. Усі операції проводяться в кінцевому полі Z_{26}.

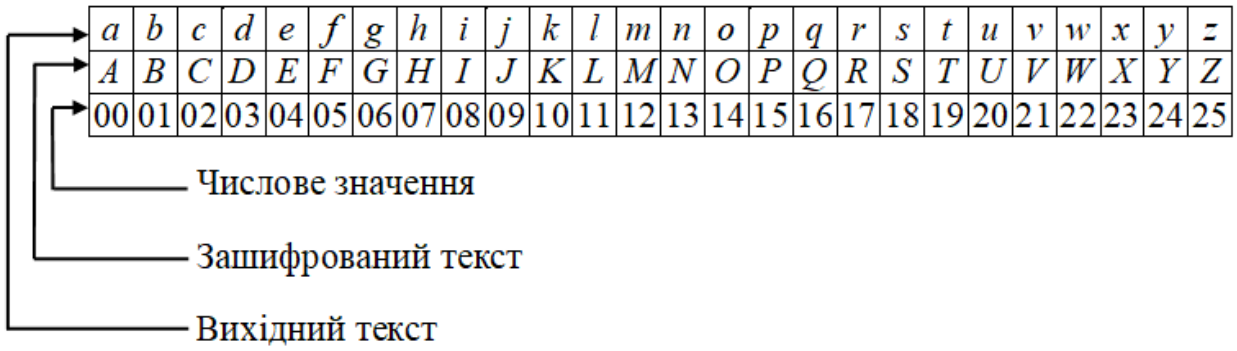


Рис. 2.1. Представлення букв вихідного і зашифрованого тексту в Z_{26} для англійської мови

а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м
А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Рис. 2.2. Представлення букв вихідного і зашифрованого тексту в Z_{33} для української мови

На рис. 2.3 показано процес шифрування і розшифрування повідомлень за допомогою адитивного шифру підстановки для алфавіту довжиною n.

Шифрування текстових повідомлень за допомогою адитивного шифру підстановки здійснюється за формулою:

$$C = (M + K) \bmod n, \tag{2.1}$$

де n – довжина алфавіту (для англійського алфавіту $n = 26$); M і C – вихідний і зашифрований текст відповідно; K – ключ шифрування (розшифрування), а розшифрування

$$M = (C - K) \bmod n. \tag{2.2}$$

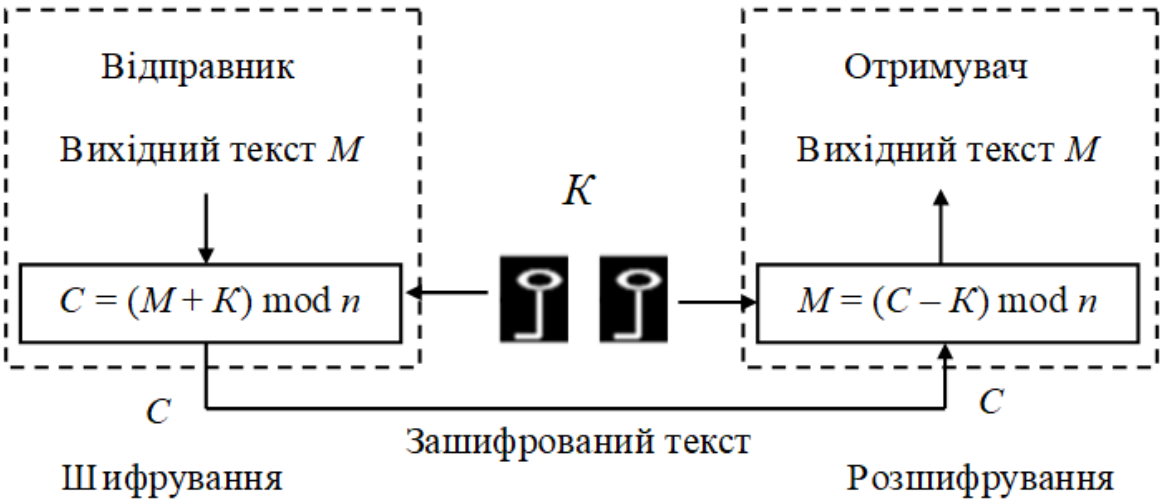


Рис. 2.3. Шифрування і розшифрування повідомлень за допомогою адитивного шифру підстановки для алфавіту довжиною n

Можна легко показати, що процеси шифрування і розшифрування є інверсними один одному, тому що створений отримувачем (M') вихідний текст той самий, що і переданий відправником (M):

$$M' = (C - K) \bmod n = (M + K - K) \bmod n = M.$$

Приклад 2.3. Зашифрувати повідомлення *hello* з використанням адитивного шифру підстановки з ключем $K = 15$.

Рішення. Застосовуємо формулу (2.1) до вихідного тексту, буква за буквою:

$m_1 = h \rightarrow 07$; $c_1 = (07 + 15) \bmod 26 = 22 \rightarrow W$;
$m_2 = e \rightarrow 04$; $c_2 = (04 + 15) \bmod 26 = 19 \rightarrow T$;
$m_3 = l \rightarrow 11$; $c_3 = (11 + 15) \bmod 26 = 00 \rightarrow A$;
$m_4 = l \rightarrow 11$; $c_4 = (11 + 15) \bmod 26 = 00 \rightarrow A$;
$m_5 = o \rightarrow 14$; $c_5 = (14 + 15) \bmod 26 = 03 \rightarrow D$,

в результаті отримаємо зашифроване повідомлення – *WTAAD*.

Зверніть увагу, що шифр моноалфавітний, оскільки два відображення тієї самої букви вихідного тексту (*l*) зашифровані як той самий символ (*A*).

Приклад 2.4. Розшифрувати повідомлення *WTAAD*, використовуючи адитивний шифр з ключем $K = 15$.

Рішення. Застосовуємо формулу розшифрування (2.2) до зашифрованого тексту, буква за буквою:

$c_1 = W \rightarrow 22$; $m_1 = (22 - 15) \bmod 26 = 07 \rightarrow h$;

$c_2 = T \rightarrow 19$; $m_2 = (19 - 15) \bmod 26 = 04 \rightarrow e$;

$c_3 = A \rightarrow 00$; $m_3 = (00 - 15) \bmod 26 = 11 \rightarrow l$;

$c_4 = A \rightarrow 00$; $m_4 = (00 - 15) \bmod 26 = 11 \rightarrow l$;

$c_5 = D \rightarrow 03$; $m_5 = (03 - 15) \bmod 26 = 14 \rightarrow o$,

в результаті отримаємо вихідне повідомлення – *hello*.

Зверніть увагу, що операції проводяться за модулем 26, негативний результат повинен бути відображений у Z_{26} (наприклад, –15 стає 11).

Історично адитивні шифри називалися шифрами *зсуву*, з тієї причини, що алгоритм шифрування може інтерпретуватися як "клавіша зсуву букви вниз", а алгоритм розшифрування може інтерпретуватися як "клавіша зсуву букви вгору". Наприклад, якщо ключ $K = 15$, то алгоритм шифрування зсуває букву на 15 букв вниз (до кінця алфавіту), а алгоритм розшифрування зсуває букву на 15 букв вгору (до початку алфавіту). Звісно, коли досягається кінець або початок алфавіту, потрібно рухатися по кільцю до початку або кінця відповідно (властивості операції за модулем n).

Юлій Цезар використовував адитивний шифр, щоб зв'язатися зі своїм оточенням. З цієї причини адитивні шифри згадуються іноді як шифри *Цезаря*. Цезар для свого зв'язку використовував в якості ключа цифру 3 ($K = 3$).

Адитивні шифри підстановки вразливі до атак тільки на зашифрований текст, коли використовується вичерпний перебір ключів (атака "грубої сили"). Множина ключів адитивного шифру підстановки дуже мала – їх тільки 26 (для англійського алфавіту) і 33 (для українського алфавіту). Одним

із недоцільних ключів є нульовий (зашифрований текст буде просто відповідати початковому тексту). Отже, залишається тільки 25 можливих ключів для англійського та 32 для українського алфавіту. Зловмисник може легко почати атаку “грубої сили” на зашифрований текст.

Приклад 2.5. Нехай зловмисник перехопив зашифрований текст *UVACLYFZLJBYL*. Він знає, що для шифрування використовувався адитивний шифр. Покажемо, як він може зламати шифр, використовуючи атаку “грубої сили”.

Рішення. Зловмисник намагається розкрити текст і послідовно перебирає ключі, починаючи з першого. За допомогою ключа номер 7 ($K = 7$) він отримує осмислений текст “*not very secure*” (не дуже безпечний).

Зашифрований текст: *UVACLYFZLJBYL*

$K = 1$	Вихідний текст: *tubkxeykiaxk*
$K = 2$	Вихідний текст: *styajwdxjhzwj*
$K = 3$	Вихідний текст: *rsxzivewigyvi*
$K = 4$	Вихідний текст: *qrwyhubvhfhuh*
$K = 5$	Вихідний текст: *pqvxgtaugewtg*
$K = 6$	Вихідний текст: *opuwfsztfdvst*
$K = 7$	Вихідний текст: *notverysecure*.

Адитивні шифри підстановки також можуть бути об’єктами статистичних атак. Це особливо реально, якщо противник перехопив зашифрований текст великої довжини. Він може скористатися знаннями про частоту появи символів у конкретній мові. У табл. 2.1 та 2.2 показано частоту появи певних букв відповідно для англійського та українського текстів довжиною в 100 символів [31, 36].

Таблиця 2.1

Частота появи букв в англійському тексті

Буква	Частота	Буква	Частота	Буква	Частота
E	12,7	*D*	4,3	*B*	1,0
T	9,1	*L*	4,0	*V*	0,9
A	8,2	*C*	2,8	*K*	0,8
O	7,5	*U*	2,8	*J*	0,2
I	7,0	*W*	2,3	*Q*	0,1
N	6,7	*F*	2,2	*X*	0,1
S	6,3	*G*	2,0	*Z*	0,1
H	6,1	*Y*	1,9		
R	6,0	*P*	1,5		

Таблиця 2.2

Частота появи букв в українському тексті

Буква	Частота	Буква	Частота	Буква	Частота
О	9,4	*Л*	3,6	*Х*	1,2
А	7,2	*Д*	3,5	*Ш*	1,2
Н	6,5	*К*	3,5	*Ж*	0,9
И	6,1	*М*	3,1	*Є*	0,8
І	5,7	*П*	2,9	*Й*	0,8
Т	5,5	*Ь*	2,9	*Ї*	0,6
В	5,2	*Я*	2,9	*Ц*	0,6
Е	4,7	*З*	2,3	*Ю*	0,4
Р	4,7	*Ч*	1,8	*Ф*	0,1
С	4,1	*Б*	1,7	*Щ*	0,1
У	4,0	*Г*	1,6	*Ґ*	0,0

Однак інформації про частоту єдиного символу недостатньо, і це ускладнює аналіз зашифрованого тексту, заснованого на аналізі частоти появи букв. Важливо знати частоту появи комбінацій символів, а також необхідно знати частоту появи в зашифрованому тексті комбінацій із двома або трьома символами та порівнювати її з частотою в мові, якою написаний вихідний документ.

Найбільш використовувані групи з двох символів (біграми) і групи з трьох символів (триграми) для англійського тексту показано в табл. 2.3.

Таблиця 2.3

Групи біграм і триграм, засновані на частоті їх появи в англійському тексті

Біграма	*TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI, OF*
Триграма	*THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR, DTH*

Приклад 2.6. Зловмисник перехопив наступний зашифрований текст

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX LQSVILYVVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW.

Використовуючи статистичну атаку, необхідно знайти вихідний текст.

Рішення. Коли зловмисник складе таблицю частоти букв у цьому зашифрованому тексті, він отримає: $I = 14$, $V = 13$, $S = 12$ і так далі. Найчастіший символ – I – має 14 появ. Це показує, що символ I в зашифрованому тексті, ймовірно, відповідає символу *e* в початковому тексті. Тим самим, ключ шифрування $K = 4$. Тому зловмисник, використовуючи ключ $K = 4$, розшифровує текст і отримує:

the house is now for sale for four million dollars it is worth more hurry before the seller receives more offers

(*будинок тепер продається за чотири мільйони доларів, варто поспішити, поки продавець не отримав більше пропозицій*).

Мультиплікативні шифри підстановки

У мультиплікативному шифрі підстановки алгоритм шифрування застосовує операцію множення вихідного тексту на ключ, а алгоритм розшифрування застосовує операцію ділення зашифрованого тексту на ключ, як показано на рис. 2.4. Оскільки операції в мультиплікативному шифрі підстановки проводяться в кінцевому полі Z_{26}, розшифрування тут означає множення на мультиплікативно обернене значення ключа. Зверніть увагу, що

ключ повинен належати до набору Z_n^*, це гарантує, що алгоритми шифрування й розшифрування будуть обернені один одному.

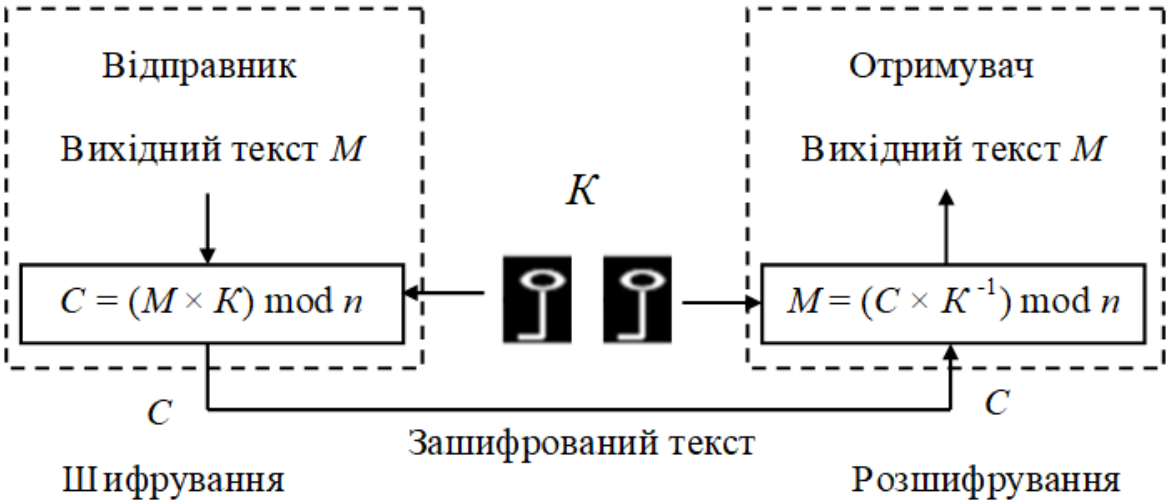


Рис. 2.4. Шифрування і розшифрування повідомлень за допомогою мультиплікативного шифру підстановки для алфавіту довжиною n

Шифрування текстових повідомлень за допомогою мультиплікативного шифру підстановки здійснюється за формулою:

$$C = (M \times K) \bmod n, \tag{2.3}$$

а розшифрування:

$$M = (C \times K^{-1}) \bmod n, \tag{2.4}$$

де K^{-1} – мультиплікативно обернене значення ключа K.

Приклад 2.7. Визначити, яка множина ключів шифрування (розшифрування) для мультиплікативного шифру за використання англійського алфавіту?

Рішення. Ключ повинен бути в кінцевому полі Z_{26}^*. Ця множина має тільки 12 елементів: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23 і 25. Пояснюється це тим, що для значень 2, 4, 6, 8, 10, 12, 13, 14, 16, 18, 20, 22 і 24 не існує мультиплікативно оберненого значення набору Z_{26}^*.

Приклад 2.8. Використовуючи мультиплікативний шифр, зашифрувати повідомлення *hello* з ключем $K = 7$.

Рішення. Застосовуємо формулу (2.3) до вихідного тексту, буква за буквою:

$m_1 = h \rightarrow 07$; $c_1 = (07 \times 07) \bmod 26 = 23 \rightarrow X$;
$m_2 = e \rightarrow 04$; $c_2 = (04 \times 07) \bmod 26 = 02 \rightarrow C$;
$m_3 = l \rightarrow 11$; $c_3 = (11 \times 07) \bmod 26 = 25 \rightarrow Z$;
$m_4 = l \rightarrow 11$; $c_4 = (11 \times 07) \bmod 26 = 25 \rightarrow Z$;
$m_5 = o \rightarrow 14$; $c_5 = (14 \times 07) \bmod 26 = 20 \rightarrow U$,

отримаємо зашифроване повідомлення – *XCZZU*.

Афінні шифри підстановки

Можна комбінувати адитивні та мультиплікативні шифри підстановки, щоб отримати те, що названо афінним шифром – комбінацією обох шифрів із парою ключів. Перший ключ використовується мультиплікативним шифром, другий – адитивним шифром. Рис. 2.5 показує, що афінний шифр – фактично два шифри, що застосовуються один за одним.

При використанні афінного шифру підстановки відношення між вихідним M і зашифрованим C текстами, а також ключами k_1 і k_2 визначається, як це показано нижче:

$$C = (M \times k_1 + k_2) \bmod n, \tag{2.5}$$

і

$$M = ((C + (-k_2)) \times k_1^{-1}) \bmod n, \tag{2.6}$$

де k_1^{-1} – мультиплікативна інверсія k_1, а $(-k_2)$ – адитивна інверсія k_2.

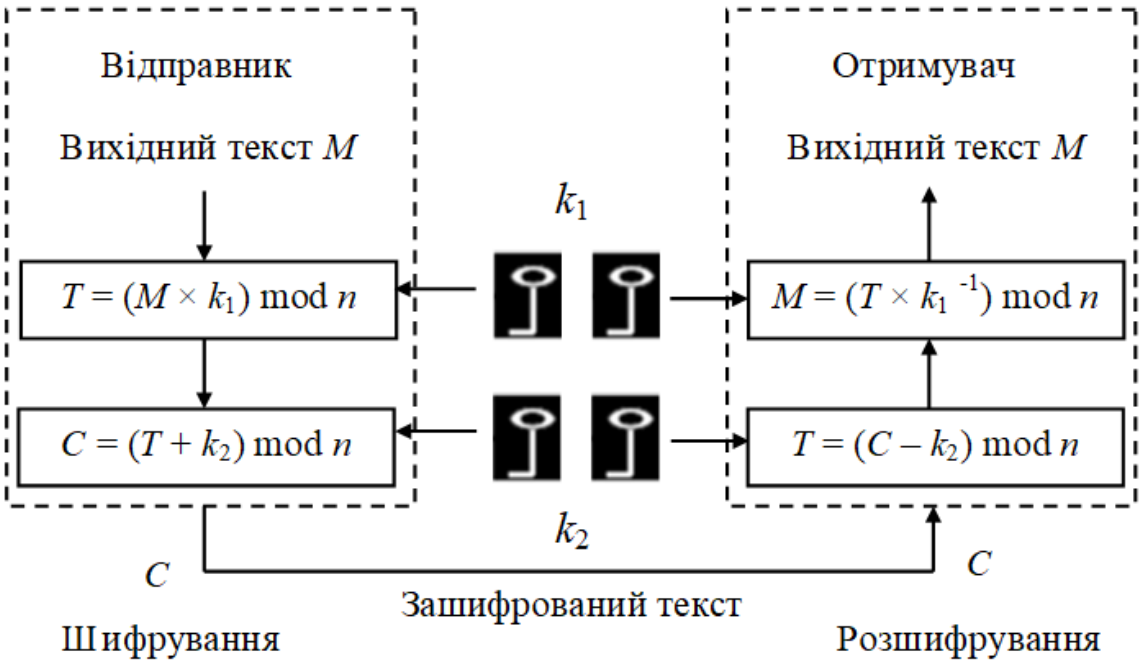


Рис. 2.5. Шифрування і розшифрування повідомлень за допомогою афінного шифру підстановки для алфавіту довжиною n

Можна було б показати тільки одну комплексну операцію для шифрування або розшифрування на рис. 2.5, таку, як (2.5). Однак на рис. 2.5 використовується проміжний результат (T) і зазначено дві окремі операції, показуючи тим самим, що кожного разу, коли використовується комбінація шифрів, потрібно переконатися, що кожний із них має інверсію на іншій стороні і що вони використовуються у зворотному порядку під час шифрування і розшифрування. Якщо додавання – остання дія під час шифрування, то віднімання має бути першою дією під час розшифрування.

Приклад 2.9. Афінний шифр підстановки використовує пару ключів, з яких перший ключ з Z_{26}^*, а другий – з Z_{26}. Визначити множину ключів для афінного шифру підстановки при використанні англійського алфавіту.

Рішення. У прикладі 2.7 визначено, що для англійського алфавіту множина ключів k_1^{-1} дорівнює 12. Тому область існування ключів для афінного шифру буде дорівнювати

$$K = Z_{26}^* \times Z_{26} = 25 \times 12 = 300.$$

Приклад 2.10. Використовуючи афінний шифр підстановки, зашифрувати повідомлення *hello* з ключовою парою $k_1 = 7$ і $k_2 = 2$.

Рішення. Використовуємо $k_1 = 7$ для мультиплікативного шифру та $k_2 = 2$ для адитивного шифру, а також застосовуючи алгоритм шифрування (2.5) до повідомлення, буква за буквою:

$m_1 = h \rightarrow 07$; $c_1 = (07 \times 7 + 2) \bmod 26 = 25 \rightarrow Z$;
$m_2 = e \rightarrow 04$; $c_2 = (04 \times 7 + 2) \bmod 26 = 04 \rightarrow E$;
$m_3 = l \rightarrow 11$; $c_3 = (11 \times 7 + 2) \bmod 26 = 01 \rightarrow B$;
$m_4 = l \rightarrow 11$; $c_4 = (11 \times 7 + 2) \bmod 26 = 01 \rightarrow B$;
$m_5 = o \rightarrow 14$; $c_5 = (14 \times 7 + 2) \bmod 26 = 22 \rightarrow W$,

отримаємо зашифроване повідомлення *ZEBBW*.

Приклад 2.11. Використовуючи афінний шифр підстановки, розшифрувати повідомлення *ZEBBW* із ключовою парою $k_1 = 7$ і $k_2 = 2$.

Рішення. Щоб знайти символи вихідного тексту, додамо адитивну інверсію від $(-k_2) \bmod 26 = (-2) \bmod 26 = 24$ до отриманого зашифрованого тексту. Потім помножимо результат на мультиплікативну інверсію від $k_1^{-1} \bmod 26 = 7^{-1} \bmod 26 = 15$. Оскільки k_2 має адитивну інверсію в Z_{26} і k_1 має мультиплікативну інверсію в Z_{26}^*, вихідний текст – точно такий, що використовувався в прикладі 2.10:

$c_1 = Z \rightarrow 25$; $m_1 = ((25 + 24) \times 15) \bmod 26 = 07 \rightarrow h$;
$c_2 = E \rightarrow 04$; $m_2 = ((04 + 24) \times 15) \bmod 26 = 04 \rightarrow e$;
$c_3 = B \rightarrow 01$; $m_3 = ((04 + 24) \times 15) \bmod 26 = 11 \rightarrow l$;
$c_4 = B \rightarrow 01$; $m_4 = ((01 + 24) \times 15) \bmod 26 = 11 \rightarrow l$;
$c_5 = W \rightarrow 22$; $m_5 = ((22 + 24) \times 15) \bmod 26 = 14 \rightarrow o$.

Отже, отримаємо вихідне повідомлення *hello*.

Адитивний шифр підстановки – окремий випадок афінного шифру для якого $k_1 = 1$. Мультиплікативний шифр підстановки – окремий випадок афінного шифру, де $k_2 = 0$.

Хоча методи “грубої сили” і статистичної атаки можуть використовуватися тільки для зашифрованого тексту, спробуємо атаку на обраний вхідний текст. Припустимо, що зловмисник перехоплює наступний зашифрований текст англійською мовою:

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG.

Зловмисник також ненадовго (тимчасово) отримує доступ до комп’ютера відправника повідомлень і час, достатній лише для того, щоб надрукувати вхідний текст з двома символами: *et*. Тоді він намагається зашифрувати короткий вхідний текст, використовуючи два різних алгоритми, тому що невпевнений, який із них є афінним шифром підстановки. В результаті він отримує перший набір даних “*вхідний текст/зашифрований текст*” – $et \rightarrow WC$ і другий набір даних $et \rightarrow WF$.

Для того, щоб знайти ключ, зловмисник використовує наступну стратегію:

1. Зловмисник знає, якщо перший алгоритм є афінним шифром підстановки, то він може скласти наступні рівняння, засновані на першому наборі даних:

$m_1 = e \rightarrow 04$; $\quad c_1 = (04 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W$;

$m_2 = t \rightarrow 19$; $\quad c_2 = (19 \times k_1 + k_2) \bmod 26 = 02 \rightarrow C$.

Ці два рівняння можна розв’язати (знайти значення k_1 і k_2)

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 04 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 02 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 02 \end{pmatrix} \bmod 26 = \begin{pmatrix} 16 \\ 10 \end{pmatrix}.$$

Звідки $k_1 = 16$, а $k_2 = 10$. Проте ця відповідь неприйнятна, тому що $k_1 = 16$ не може бути першою частиною ключа. Його значення 16 не має мультиплікативної інверсії в Z_{26}^*.

2. Зловмисник тепер намагається використовувати результат другого набору даних:

$m_1 = e \rightarrow 04$; $c_1 = (04 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W$;

$m_2 = t \rightarrow 19$; $c_2 = (19 \times k_1 + k_2) \bmod 26 = 05 \rightarrow F$.

Квадратна матриця та її інверсія такі самі, що й у попередньому прикладі:

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 04 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 05 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 05 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 04 \end{pmatrix}.$$

Тепер зловмисник отримує $k_1 = 11$ і $k_2 = 4$, ця пара є прийнятною, тому що k_1 має мультиплікативну інверсію в Z_{26}^*. Він використовує пару ключів (19, 22), які є інверсією пари (11, 4):

$c_1 = W \rightarrow 22$; $m_1 = ((22 + 22) \times 19) \bmod 26 = 04 \rightarrow e$;

$c_2 = F \rightarrow 05$; $m_2 = ((05 + 22) \times 19) \bmod 26 = 19 \rightarrow t$,

і розшифровує повідомлення

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG.

Вихідний текст при цьому буде

Best time of the year is spring when flower bloom

(*Найкраща пора року – весна, коли цвітуть квіти*).

Одноалфавітні шифри підстановки

Оскільки адитивні, мультиплікативні й афінні шифри підстановки мають малу множину ключів, вони вразливі до атаки “грубої сили”. Відправник і отримувач погоджують єдиний ключ, який вони використовують, щоб зашифрувати кожну букву початкового тексту або розшифрувати кожну букву в зашифрованому тексті. Іншими словами, ключ незалежний від переданих букв.

Вдале рішення полягає в тому, щоб створити відображення кожної букви початкового тексту у відповідну букву зашифрованого тексту. Відправник і отримувач можуть домовитися про відображення для кожної букви й записати його у вигляді таблиці. Рис. 2.6 показує приклад такого відображення для англійського алфавіту.

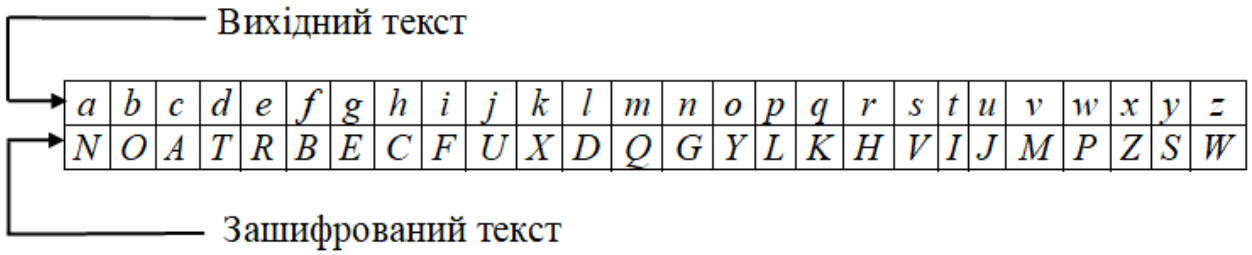


Рис. 2.6. Приклад ключа для одноалфавітного шифру підстановки англійського алфавіту

Приклад 2.12. Використовуючи ключ, показаний на рис. 2.6, зашифрувати повідомлення

This message is easy to encrypt but hard to find the key

(*Це повідомлення просто зашифрувати, але важко знайти ключ, яким зашифрований текст*).

Рішення. Зашифроване повідомлення має вигляд

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS.

Розмір ключового простору для одноалфавітного шифру підстановки для англійського алфавіту – число перестановок із 26, тобто 26! (майже $4 \cdot 10^{26}$). Це робить атаку “грубої сили” надзвичайно складною для зловмисника, навіть при використанні потужного комп’ютера. Однак він може застосувати статистичну атаку, засновану на частоті появи символів, тому що даний шифр не змінює частоту появи символів.

2.1.2. Багатоалфавітні шифри підстановки

Використання багатоалфавітного шифру підстановки призводить до появи символу, який може мати різну заміну. Відношення між символом у початковому тексті й символом у зашифрованому тексті – “*один до багатьох*”. Наприклад, буква *a* може бути зашифрована як *D* на початку тексту, але як *N* – у середині. Багатоалфавітні шифри підстановки мають перевагу: вони приховують частоту появи символу основної мови. Зловмисник не може використовувати статистичну частоту появи окремого символу, щоб зламати зашифрований текст.

Щоб створити багатоалфавітний шифр підстановки, потрібно зробити кожний символ зашифрованого тексту залежним від відповідного символу вихідного тексту та його позиції в повідомленні. Це передбачає, що ключ повинен бути потоком підключів, в якому кожний підключ так чи інакше залежить від позиції символу вихідного тексту, який використовується для вибору підключа шифрування. Іншими словами, потрібно мати ключовий потік $K = (k_1, k_2, \ldots)$, в якому k_i застосовується для того, щоб зашифрувати i-й символ у початковому тексті та створити i-й символ у зашифрованому тексті.

Автоключовий шифр підстановки

Для визначення залежності ключа від позиції використовується простий багатоалфавітний шифр підстановки, який називається автоключовим. У цьому шифрі ключ – потік підключів, в якому кожний

підключ використовується для того, щоб зашифрувати відповідний символ у початкового тексті. Перший підключ – визначене заздалегідь значення, яке таємно погоджене відправником і отримувачем. Другий підключ – значення першого символу вихідного тексту (між 0 і 25). Третій – значення другого символу вихідного тексту й так далі. Нехай вихідний текст: $M = m_1, m_2, m_3, \ldots$. Зашифрований текст: $C = c_1, c_2, c_3, \ldots$. Ключ $K = (k_1, k_2 = m_1, k_3 = m_2, \ldots)$.

Шифрування текстових повідомлень за допомогою автоключового шифру підстановки здійснюється за формулою:

$$c_i = (m_i + k_i) \bmod n, \tag{2.7}$$

а розшифрування:

$$m_i = (c_i - k_i) \bmod n. \tag{2.8}$$

Назва шифру автоключовий означає, що підключі створюються автоматично залежно від символів шифру вихідного тексту у процесі шифрування.

Приклад 2.13. Припустимо, що відправник і отримувач погодилися використовувати автоключовий шифр із початковим ключовим значенням $k_1 = 12$. Тепер відправник хоче передати отримувачу повідомлення *Attack is today* (*Атака сьогодні*).

Рішення. Шифрування проводиться символ за символом. Кожний символ у початковому тексті спочатку замінюється його числовим значенням, як показано на рис. 2.1, перший підключ додається, щоб створити перший символ зашифрованого тексту. Інша частина ключа створюється в міру читання символів вихідного тексту. Результат шифрування наведено в табл. 2.4.

Таблиця 2.4

Результат шифрування для прикладу 2.13

Вихідний текст	*a*	*t*	*t*	*a*	*c*	*k*	*i*	*s*	*t*	*o*	*d*	*a*	*y*
Значення *M*	00	19	19	00	02	10	08	18	19	14	03	00	24
Потік ключів *K*	12	00	19	19	00	02	10	08	18	19	14	03	00
Значення *C*	12	19	12	19	02	12	18	00	11	07	17	03	24
Зашифрований текст	*M*	*T*	*M*	*T*	*C*	*M*	*S*	*A*	*L*	*H*	*R*	*D*	*Y*

Зауважимо, що шифр є багатоалфавітним, тому що три появи символу *a* у початковому тексті зашифровані по-різному. Три появи символу *t* також зашифровані по-різному.

Автоключовиий шифр підстановки дійсно приховує статистику частоти появи окремого символу, однак залишається вразливий до атаки “грубої сили”, як і адитивний шифр підстановки. Перший підключ може бути тільки одним із 25 значень (1 – 25). Існує потреба в багатоалфавітних шифрах підстановки, які не тільки приховують характеристики мови, але і мають велику множину ключів.

Шифр підстановки Плейфера

Інший приклад багатоалфавітного шифру підстановки – шифр Плейфера, що використовувався британською армією протягом Першої світової війни. Ключ засекречування в цьому шифрі зроблений із 25 букв алфавіту, розміщених у матриці 5×5 для латинського алфавіту (для кириличного алфавіту – в матриці 6×6).

Для створення матриці й використання шифру достатньо запам’ятати ключове слово і чотири прості правила. Щоб скласти ключову матрицю, у першу чергу потрібно заповнити порожні комірки матриці буквами ключового слова (не записуючи повторювані символи). Потім заповнити порожні комірки матриці, які залишилися вільними, символами алфавіту, що не зустрічаються в ключовому слові, по порядку (в англійських текстах

зазвичай опускається символ *q*, щоб зменшити алфавіт, в інших версіях *i* і *j* об’єднуються в одну комірку). Ключове слово може бути записано у верхньому рядку матриці зліва направо, або по спіралі з лівого верхнього кута до центру. Ключове слово, доповнене символами алфавіту, складає матрицю 5×5 і є ключем шифру. За допомогою різних домовленостей про розміщення букв у матриці можна створити багато різних ключів засекречування.

Для того, щоб зашифрувати повідомлення необхідно розбити його на біграми (групи з двох символів), наприклад, *hello world* стає *he ll ow or ld*, і відшукати ці біграми у таблиці. Два символи біграми відповідають кутам прямокутника в ключовій матриці. Визначаємо положення кутів цього прямокутника відносно один одного. Потім, керуючись наступними чотирма правилами, зашифровуємо пари символів вихідного тексту:

1. Якщо два символи біграми збігаються, додаємо після першого символа фіктивний символ *x*, зашифровуємо нову пару символів і продовжуємо. У деяких варіантах шифру Плейфера замість символу *x* використовується символ *q* або _ .

2. Якщо символи біграми вихідного тексту зустрічаються в одному рядку, то вони замінюються на символи того самого рядка, що знаходяться праворуч від них. Якщо символ є останнім у рядку, то він замінюється на перший символ цього самого рядка.

3. Якщо символи біграми вихідного тексту зустрічаються в одному стовпці, то вони замінюються на символи того самого стовпця, що знаходяться безпосередньо під ними. Якщо символ є нижнім у стовпці, то він замінюється на перший символ цього самого стовпця.

4. Якщо символи біграми вихідного тексту знаходяться в різних стовпцях і різних рядках, то вони замінюються на символи, що знаходяться в тих самих рядках, але відповідають іншим кутам прямокутника в ключовій матриці.

Для розшифрування зашифрованого повідомлення необхідно

використовувати інверсію цих чотирьох правил, відкидаючи символи *x* (або *q*), якщо вони не мають сенсу у вихідному повідомленні.

Шифр Плейфера відповідає критеріям для багатоалфавітного шифру. Ключ – потік підключів, у якому вони створюються по два одночасно. У шифрі Плейфера потік ключів і потік шифру – такі самі. Це означає, що вищезазначені правила можна представити як правила для створення потоку ключів. Алгоритм шифрування бере пару символів з вихідного тексту та створює пару підключів, згідно вищезазначених правил. Можна сказати, що потік ключів залежить від позиції символу у початковому тексті. Ця залежність від позиції має різну інтерпретацію: підключ для кожного символу вихідного тексту залежить від наступного або попереднього. Отже, розглядаючи шифр Плейфера, можна сказати, що зашифрований текст – це фактично потік ключів.

Нехай вихідний текст: $M = m_1, m_2, m_3, \ldots$. Зашифрований текст: $C = c_1, c_2, c_3, \ldots$. Ключ $K = [(k_1, k_2), (k_3, k_4), \ldots)]$. Шифрування текстових повідомлень за допомогою шифру Плейфера відбувається за виразом:

$$c_i = k_i$$

а розшифрування:

$$m_i = k_i$$

Приклад 2.14. Зашифрувати вихідний текст *hello world* (*Привіт світ*) за допомогою шифру Плейфера з ключовим словом *playfairexample*. У такому випадку матриця засекречування (або секретний ключ) набуде вигляду, як показано на рис. 2.7.

Секретний ключ =

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

Рис. 2.7. Приклад секретного ключа шифру Плейфера

Рішення. Коли згрупуємо букви по парам, то отримаємо *he ll ow or ld*. Необхідно вставити *x* між двома буквами *l*, після чого отримуємо *he lx lo wo rl d*. Для забезпечення останнього символу парою потрібно додати символ *x* за символом *d*, після чого отримаємо *he lx lo wo rl dx*.

Перетворюючи вихідний текст за допомогою секретного ключа (рис. 2.7), отримаємо:

1. Біграма *he* формує прямокутник, замінюємо її на *DM*.
2. Біграма *lx* формує прямокутник, замінюємо її на *YR*.
3. Біграма *lo* формує прямокутник, замінюємо її на *YK*.
4. Біграма *wo* розташована в одному стовпці, замінюємо її на *YW*.
5. Біграма *rl* розташована в одному стовпці, замінюємо її на *CR*.
6. Біграма *dx* формує прямокутник, замінюємо її на *GE*.

Отже, зашифрований текст буде представлений у вигляді:

DM YR YK YW CR GE.

Цей приклад показує, що шифр Плейфера – фактично багатоалфавітний шифр: однакові символи зашифровані різними символами (три появи символу *l* у вихідному тексті зашифровані як *Y* та *R* відповідно; дві появи символу *o* у вихідному тексті зашифровані як *K* та *W* відповідно).

Для використання кириличного алфавіту необхідно збільшити розмір матриці до 6×6. Використовуються 33 букви алфавіту та додатково три символи: . (крапка); , (кома); _ (знак підкреслення). Приклад матриці засекречування для українського алфавіту (без використання ключового слова) наведено на рис. 2.8.

Символ _ (знак підкреслення) використовується у випадку коли в біграмах з’являються однакові символи (він вставляється між ними) або для забезпечення останнього символу парою.

Очевидно, атака “грубої сили” на шифр Плейфера є надто складною, адже розмір домену – *n*! (*n* факторіал). Крім того, зашифрований текст

приховує частоту появи окремих букв. Однак частота появи двобуквених комбінацій (біграм) зберігається до деякої міри через вставки наповнювача так, що криптографічний аналітик для знаходження ключа може використовувати атаку тільки на зашифрований текст, засновану на випробуванні частоти появи біграм.

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

Рис. 2.8. Приклад матриці засекречування для українського алфавіту

Шифр підстановки Віженера

Ще одним цікавим видом багатоалфавітного шифру підстановки є шифр Віженера, створений Блезом де Віженером, французьким математиком шістнадцятого століття [2, 24, 31, 36]. Шифр Віженера використовує різну стратегію створення потоку ключів. Потік ключів – повторення початкового потоку ключа засекречування довжини m. Шифр може бути описаний таким чином: (k_1, k_2, …, k_m) – секретний ключ засекречування, узгоджений відправником і отримувачем.

Нехай вихідний текст: $M = m_1, m_2, m_3, \ldots$. Зашифрований текст: $C = c_1, c_2, c_3, \ldots$. Потік ключів $K = k_1, k_2, k_3, \ldots$.

Шифрування текстових повідомлень за допомогою шифру Віженера відбувається за допомогою виразу:

$$c_i = (m_i + k_i) \bmod n, \tag{2.11}$$

а розшифрування:

$$m_i = (c_i - k_i) \bmod n. \tag{2.12}$$

Однією із важливих відмінностей між шифром Віженера та іншими розглянутими багатоалфавітними шифрами є те, що потік ключів шифру підстановки Віженера не залежить від символів вихідного тексту; він залежить тільки від позиції символу у вихідному тексті. Іншими словами, потік ключів може бути створений без знання суті вихідного тексту.

Приклад 2.15. Зашифрувати повідомлення *She is listening* (*Вона слухає*), використовуючи ключове слово із шести символів *PASCAL*.

Рішення. Початковий потік ключів – це (15, 0, 18, 2, 0, 11). Потоком ключів є повторення цього початкового потоку ключів (стільки разів, скільки необхідно). Процес шифрування представлено у табл. 2.5.

Таблиця 2.5

Процес шифрування для прикладу 2.15

Вихідний текст	*s*	*h*	*e*	*i*	*s*	*l*	*i*	*s*	*t*	*e*	*n*	*i*	*n*	*g*
Значення *M*	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Потік ключів *K*	15	00	18	02	00	11	15	00	18	02	00	11	15	00
Значення *C*	07	07	22	10	18	22	23	18	11	06	13	9	02	06
Зашифрований текст	*H*	*H*	*W*	*K*	*S*	*W*	*X*	*S*	*L*	*G*	*N*	*T*	*C*	*G*

Таким чином, зашифрований текст буде мати такий вигляд: *HHWKSWXSLGNTCG*.

Шифр підстановки Віженера може розглядатися як комбінація адитивних шифрів. Рис. 2.9 показує, що вихідний текст попереднього прикладу можна розглядати як такий, що складається з декількох частин по шість елементів у кожному (хоча в одному не вистачило букв вихідного тексту), де кожний з елементів зашифрований окремо. Рисунок допоможе пізніше зрозуміти криптографічний аналіз шифру підстановки Віженера. Є m частин вихідного тексту, кожна з яких зашифрована різним ключем, щоб розділити зашифрований текст на m частин. Адитивний шифр – окремий випадок шифру підстановки Віженер, в якому m =1.

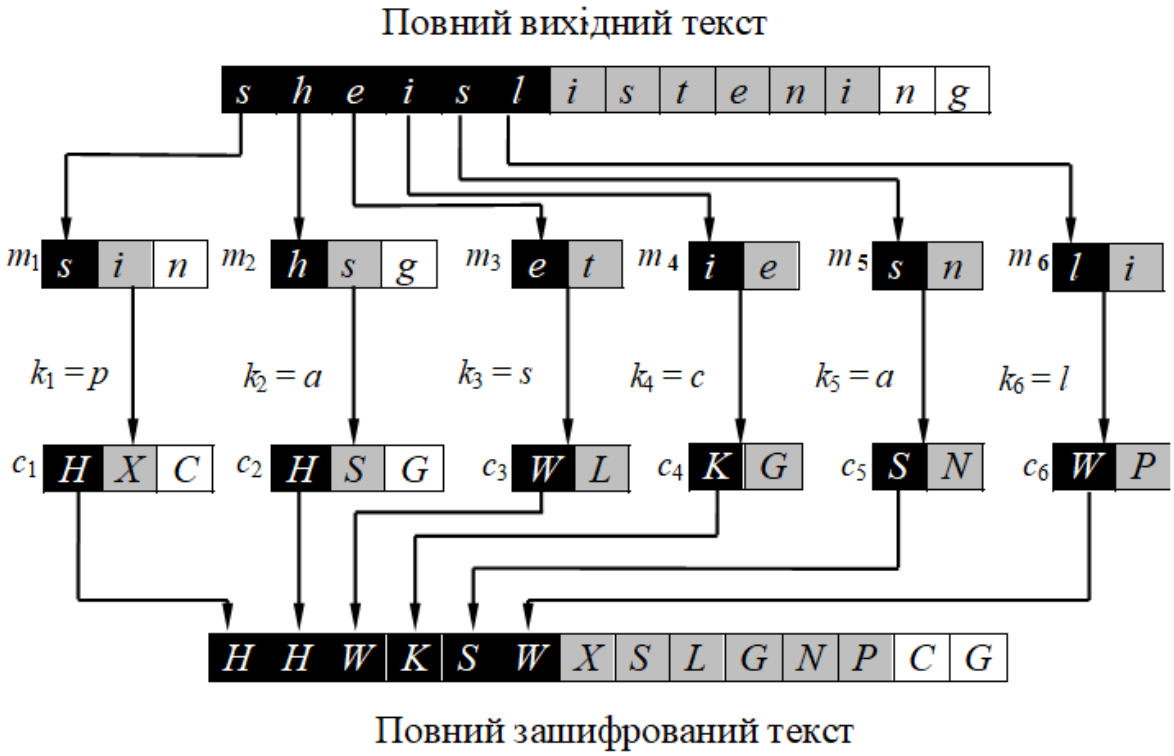


Рис. 2.9. Шифр підстановки Віженера, як комбінація адитивних шифрів

Інший спосіб розгляду шифрів Віженера – за допомогою відомого списку або таблиці Віженера, що показано на рис. 2.10. Перший рядок показує символи вихідного тексту, які будуть зашифровані. Перший стовпець містить символи, які використовуються в якості ключа. Інша частина таблиці показує символи зашифрованого тексту. Щоб знайти зашифрований текст для вихідного *she listening*, використовуючи слово *PASCAL* як ключ, необхідно знайти символ *s* з вихідного тексту в першому рядку, символ ключового слова *P* у першому стовпці; а на їх перетині знаходиться символ *H* зашифрованого тексту. Далі знаходимо *h* у першому рядку і *A* у першому стовпці, на перетині рядка та стовпця знаходиться символ *H* із зашифрованого тексту. Повторюємо ті самі дії, поки всі символи зашифрованого текст не будуть знайдені.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Рис. 2.10. Список або таблиця Віженера

Щоб знайти вихідний текст для зашифрованого *HHWKSWXSLGNTCG*, використовуючи слово *PASCAL* як ключ, потрібно знайти символ *P* із ключового слова у першому стовпці; далі, рухаючись вправо по рядку, знаходимо символ *H* зашифрованого тексту; на перетині стовпця, в якому знаходиться символ *H* і першого рядка – символ вихідного тексту *s*. Далі знаходимо символ *A* з ключового слова у першому стовпці; і, рухаючись вправо по рядку, знаходимо символ *H* зашифрованого тексту; на перетині стовпця, в якому знаходиться символ *H* і першого рядка – символ вихідного тексту *h*. Повторюємо ті самі дії, поки всі символи вихідного тексту не будуть знайдені.

Шифр підстановки Віженера, подібно до всіх багатоалфавітних шифрів, не зберігає частоту появи символів. Однак зловмисник може використовувати деякі методи для того, щоб розшифрувати перехоплений зашифрований текст. Криптографічний аналіз у даному випадку складається з двох частин: знаходять довжину ключа та потім безпосередньо знаходять ключ:

1. Було винайдено кілька методів, щоб знайти довжину ключа. Один з них розглянемо нижче. У так званому *тесті Казіскі* криптографічний аналітик у зашифрованому тексті шукає повторні сегменти принаймні з трьох символів [4, 19, 27, 39]. Припустимо, що знайдено два сегмента, і відстань між ними – d. Криптографічний аналітик припускає, що $m|d$, де m – довжина ключа. Якщо можна знайти більше повторних сегментів із відстанню $d_1, d_1, \ldots, d_n$, тоді $m|\text{НСД}(d_1, d_1, \ldots, d_n)$ (найбільший спільний дільник). Це припущення логічне, тому що якщо два символи однакові й $k \times m$ ($k = 1, 2, \ldots$) – символи, виділені в початковому тексті, то однакові також $k \times m$ символи, виділені в зашифрованому тексті. Криптографічний аналітик використовує сегменти принаймні з трьох символів, щоб уникнути випадків, де символи мають той самий ключ.

2. Після того як довжину ключа знайдено, криптографічний аналітик використовує адитивний шифр – окремий випадок шифру підстановки Віженера, в якому $m = 1$. Зашифрований текст поділяється на m різних частин і застосовується метод, який використовується у криптографічному аналізі адитивного шифру, включаючи статистичну атаку частоти появи символів. Кожна з m частин зашифрованого тексту може бути розшифрована та з'єднана з іншими m частинами, щоб створити цілісний вихідний текст. Весь зашифрований текст не зберігає частоту появи окремих букв вихідного тексту, але кожна m частина робить це.

Наступний приклад допоможе зрозуміти ці міркування.

Приклад 2.16. Припустимо, що зловмисник перехопив наступний зашифрований текст:

LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEVLWPCZV
GTHVTSGXQOVGCSVETQLTJSUMVWVEFRLXKJHGKDKITKLXJHKI
MSBWUSWXQHKVGSHEEVFLCFDGVSUMPHKIRZDMPHHBVWVWJWIX
VVUIGQKETOFXGFVGTSDXGQLTJSUXGLW

Зловмиснику необхідно дешифрувати цей зашифрований текст.

Рішення. Тест Казіскі на повторення сегментів у три символи приводить до результатів, показаних у табл. 2.6.

Таблиця 2.6

Тест Казіскі для прикладу 2.16

Комбінація	Перша відстань	Друга відстань	Різниця
JSU	69	169	100
SUM	70	122	52
VWV	73	137	64
MPH	124	132	8

Найбільший спільний дільник – 4, що означає довжину ключа, пропорційну чотирьом. Спочатку зловмисник пробує $m = 4$. Ділить зашифрований текст на чотири частини. Частина c_1 буде складатися з 1-го, 5-го, 9-го, ... символів; частина c_2 буде складатися з 2-го, 6-го, 10-го, … символів і так далі. Зловмисник використовує статистичну атаку кожної частини окремо. Він перебирає розшифровані частини по одному символу одночасно, щоб отримати цілісний вихідний текст як показано у табл. 2.7.

Якщо вихідний текст не має сенсу, то зловмисник пробує з іншим значенням m. У даному випадку вихідний текст має сенс (читаємо по стовпцях):

Julius Caesar used a cryptosystem in his war, which is now referred to as Caesar cipher. It is additive cipher with the keys set to three. Each character in the plaintext is shift three characters create cipher text.

Переклад цього тексту:

Юлій Цезар використовував у своїх війнах криптографічну систему, яка згадується тепер як шифр Цезаря. Це адитивний шифр з ключем, встановленим на три. Кожний символ у початковому тексті зсунутий на три символи, щоб створити зашифрований текст.

Таблиця 2.7

Процес дешифрування зашифрованого тексту для прикладу 2.16 за допомогою тесту Казіскі

c_1	LWGW	CRAO	KTEP	GTQC	TJVF	KKTJ	MUQG	ECVP	RPVJ	VGTG	TGJG
m_1	J u e u	a p y m	i r c n	e r o a	r h t d	i i r h	k s o e	c a t n	p n t h	t e r e	r e h e

c_2	IGGG	QHGW	GKVC	TSOS	QSWR	JDKH	SSHS	VFSH	ZHWW	VQOF	SQSL
m_2	u s s s	c t s i	s w h o	f e a e	c e i d	v p w t	e e t e	h r e t	l t i i	h c a r	e c e x

c_3	OFDH	URWQ	ZKLZ	HGVV	LUVL	HKLK	BWKH	FDUK	DHVI	UKFV	DLUW
m_3	l C a e	r o t n	w h i w	e d s s	i r s i	e h i h	y t h E	c a r h	a e s f	r h c s	a i r t

c_4	MEVH	CWIL	EMWV	VXGE	TMEX	GIXI	WXVE	LGMI	MBWX	IEXG	XTX
m_4	i a r d	y s e h	a i s r	r t C a	p I a t	c e t e	s t r a	h c I e	i x s t	e a t c	t p t

Для використання шифрування і розшифрування повідомлень із використанням українського алфавіту можна скористатися табл. 2.8.

Таблиця 2.8

Список Віженера для українського алфавіту

	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
А	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я
Б	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А
В	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б
Г	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В
Ґ	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г
Д	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ
Е	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д
Є	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е
Ж	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є
З	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж
И	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З
І	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И
Ї	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І
Й	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї
К	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й
Л	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К
М	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л
Н	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М
О	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н
П	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О
Р	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П
С	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р
Т	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С
У	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т
Ф	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У
Х	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф
Ц	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х
Ч	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц
Ш	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч
Щ	Щ	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш
Ь	Ь	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ
Ю	Ю	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь
Я	Я	А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю

Шифр Хілла

Інший цікавий приклад багатоалфавітного шифру – шифр Хілла, винайдений Лестером С. Хіллом [4, 19, 24, 31, 36, 39]. На відміну від розглянутих багатоалфавітних шифрів, в ньому вихідний текст розділений на блоки рівного розміру. Блоки зашифровані за одним таким способом, що

кожний символ у блоці робить внесок у шифрування інших символів у блоці. З цієї причини шифр Хілла належить до категорії шифрів, названих *блоковими шифрами*.

Інші шифри, які були вивчені до цього часу, належать до категорії так званих *шифрів потоку*. Відмінність між блоковими шифрами і шифрами потоку буде розглянуто в наступних розділах 3 та 9.

У шифрі Хілла ключ – квадратна матриця розміру $r \times r$, в якому r є розміром блоку. Якщо викликається ключова матриця K, то кожний елемент $k_{i,j}$ визначається матрицею, як показано на рис. 2.11.

$$\begin{pmatrix} k_{11} & k_{12} & \cdots & k_{1r} \\ k_{21} & k_{22} & \cdots & k_{2r} \\ \cdots & \cdots & \cdots & \cdots \\ k_{r1} & k_{r2} & \cdots & k_{rr} \end{pmatrix}$$

Рис. 2.11. Ключ у шифрі Хілла

Якщо позначити символи блоку вихідного тексту $m_1, m_2, \ldots, m_r$, а відповідні символи у блоці зашифрованого тексту будуть $c_1, c_2, \ldots, c_r$, тоді будемо мати один блок зашифрованого тексту:

$$c_1 = m_1 \cdot k_{11} + m_2 \cdot k_{21} + \ldots + m_r \cdot k_{r1};$$
$$c_2 = m_1 \cdot k_{12} + m_2 \cdot k_{22} + \ldots + m_r \cdot k_{r2};$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$c_r = m_1 \cdot k_{1r} + m_2 \cdot k_{2r} + \ldots + m_r \cdot k_{rr}.$$

Рівняння показують, що кожний символ зашифрованого тексту, такий, як c_1, залежить від символів всього вихідного тексту у блоці ($m_1, m_2, \ldots, m_r$). Однак необхідно знати, що не всі квадратні матриці мають мультиплікативні інверсії в Z_n, тому відправник і отримувач повинні бути обережні у виборі ключа. Отримувач не зможе розшифрувати зашифрований текст, переданий відправником, якщо матриця не має мультиплікативної інверсії. Отже,

ключова матриця в шифрі Хілла повинна мати мультиплікативну інверсію.

Використання матриць дозволяє відправнику зашифрувати весь вихідний текст. У цьому випадку вихідний текст – $l \times r$ – матриця, в якій l є номером блоків. Шифрування текстових повідомлень за допомогою шифру Хілла здійснюється за правилом:

$$C(l \times r) = M(l \times r) \times K(r \times r) \bmod n, \tag{2.13}$$

а розшифрування:

$$M(l \times r) = C(l \times r) \times K^{-1}(r \times r) \bmod n. \tag{2.14}$$

Приклад 2.17. Нехай вихідний текст *code is ready* (*код готовий*) необхідно зашифрувати, а потім розшифрувати зашифрований текст за допомогою шифру Хілла, використовуючи ключову матрицю K (*GYBNQKURP* у буквеному вигляді):

$$K = \begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}.$$

Рішення. Вихідний текст може бути представлений як матриця розміром 4×3 з додаванням додаткового фіктивного символу *z* до останнього блока та видалення пропусків між словами. Вихідний текст для шифрування в такому випадку буде: *codeisreadyz*. Числовий еквівалент даного повідомлення представляється у вигляді: 02, 14, 03, 04, 08, 18, 17, 04, 00, 03, 24, 25.

Шифрування даних за допомогою шифру Хілла показано на рис. 2.12.

$$\underset{C}{\begin{pmatrix} 20 & 11 & 05 \\ 20 & 10 & 16 \\ 24 & 04 & 05 \\ 24 & 23 & 20 \end{pmatrix}} = \underset{M}{\begin{pmatrix} 02 & 14 & 03 \\ 04 & 08 & 18 \\ 17 & 04 & 00 \\ 03 & 24 & 25 \end{pmatrix}} \times \underset{K}{\begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}}$$

Рис. 2.12. Шифрування повідомлень за допомогою шифру Хілла для прикладу 2.17

Як бачимо, в результаті шифрування отримано зашифроване повідомлення: 20, 11, 05, 20, 10, 16, 24, 04, 05, 24, 23, 20. Це зашифроване повідомлення відповідає *ULFUKQYEFYXU* з використанням англійського алфавіту.

Отримувач може розшифрувати повідомлення, використовуючи мультиплікативно інверсну (обернену) матрицю-ключ K^{-1}, таку, що $(K \times K^{-1}) \bmod n = I$, де I – одинична матриця. Для нашого прикладу множення матриці-ключа K на мультиплікативно обернену матрицю-ключ K^{-1} буде дорівнювати:

$$K \times K^{-1} = \begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} \times \begin{pmatrix} 08 & 05 & 10 \\ 21 & 08 & 21 \\ 21 & 12 & 08 \end{pmatrix} = \begin{pmatrix} 01 & 00 & 00 \\ 00 & 01 & 00 \\ 00 & 00 & 01 \end{pmatrix} = I.$$

Розшифрування зашифрованого повідомлення у прикладі 2.17 за допомогою шифру Хілла показано на рис. 2.13.

$$\underset{M}{\begin{pmatrix} 02 & 14 & 03 \\ 04 & 08 & 18 \\ 17 & 04 & 00 \\ 03 & 24 & 25 \end{pmatrix}} = \underset{C}{\begin{pmatrix} 20 & 11 & 05 \\ 20 & 10 & 16 \\ 24 & 04 & 05 \\ 24 & 23 & 20 \end{pmatrix}} \times \underset{K^{-1}}{\begin{pmatrix} 08 & 05 & 10 \\ 21 & 08 & 21 \\ 21 & 12 & 08 \end{pmatrix}}$$

Рис. 2.13. Розшифрування зашифрованого в прикладі 2.17 повідомлення за допомогою шифру Хілла

З рис. 2.13 витікає, що в результаті розшифрування отримано вихідне повідомлення: 02, 14, 03, 04, 08, 18, 17, 04, 00, 03, 24, 25, яке відповідає *codeisreadyz* із використанням англійського алфавіту. Відкидаючи останній символ, як результат отримуємо *codeisready*, що відповідає вихідному тексту.

Криптографічний аналіз тільки на зашифрований шифром Хілла текст є складним. По-перше, атака “грубої сили” на шифр Хілла є надзвичайно складною через матрицю-ключ – $r \times r$. Кожний вхід може мати одне з 26

значень англійського алфавіту. Це означає, що розмір ключа $26^{r \times r}$. Однак не всі матриці мають мультиплікативно обернену матрицю-ключ. Тому область існування ключів все ж не така величезна. По-друге, шифр Хілла не зберігає статистику звичайного тексту. Зловмисник не може провести статистичну атаку частоти появи окремих блоків з двох або трьох букв. Аналіз частоти слів розміру r міг би спрацювати, але дуже рідко вихідний текст має багато однакових рядків розміру r. Однак, зловмисник може провести атаку на відомий вхідний текст, якщо він знає значення r і пари вихідний текст/зашифрований текст, принаймні r блоків. Блоки можуть належати до того самого повідомлення або різних повідомлень, але повинні бути різними. Зловмисник може створити дві $r \times r$ матриці, M (вихідний текст) і C (зашифрований текст), в яких відповідні рядки представляють відомі пари вихідний текст/зашифрований текст. Оскільки $C = M \times K$, то зловмисник може використовувати відношення $K = M^{-1} \times C$, щоб знайти ключ, якщо M є оберненою. Якщо M не є оберненою, то зловмисник повинен задіяти різні набори r пар вихідний текст/зашифрований текст. Якщо зловмисник не знає значення r, то він може спробувати різні значення за умови, що r не є дуже великим.

Приклад 2.18. Припустимо, що зловмисник знає, що $r = 3$. Він перехопив три пари блоків вихідний текст/зашифрований текст (не обов'язково з того самого повідомлення), як показано на рис. 2.14.

$$M \qquad\qquad C$$
$$(05\ \ 07\ \ 10) \leftrightarrow (03\ \ 06\ \ 00)$$
$$(13\ \ 17\ \ 07) \leftrightarrow (14\ \ 16\ \ 19)$$
$$(00\ \ 05\ \ 04) \leftrightarrow (03\ \ 17\ \ 11)$$

Рис. 2.14. Формування зашифрованого тексту за допомогою шифру Хілла для прикладу 2.18

Рішення. Зловмисник формує матриці M і C із пар, представлених на рис. 2.14. Оскільки в даному випадку матриця M є оберненою, то вона обертається та множиться на C, що дає матрицю ключів K, як це показано на рис. 2.15.

$$\underset{K}{\begin{pmatrix}02 & 03 & 07\\ 05 & 07 & 09\\ 01 & 02 & 11\end{pmatrix}} = \underset{M^{-1}}{\begin{pmatrix}21 & 14 & 01\\ 00 & 08 & 25\\ 13 & 03 & 08\end{pmatrix}} \times \underset{C}{\begin{pmatrix}03 & 06 & 00\\ 14 & 16 & 09\\ 03 & 17 & 11\end{pmatrix}}$$

Рис. 2.15. Отримання матриці ключів K шифру Хілла для прикладу 2.18

Тепер він має ключ і може зламати будь-який зашифрований текст, де застосовано цей ключ.

Одноразова система шифрування

Одна із цілей криптографії – ідеальна секретність. Майже всі застосовувані на практиці шифри характеризуються як умовно надійні, оскільки вони можуть бути розкриті за наявності необмежених обчислювальних можливостей. Абсолютно надійні шифри неможливо зруйнувати навіть за використання необмежених обчислювальних можливостей.

Існує єдиний такий шифр, що винайдений в 1917 році американцями Г. Вернамом і Д. Моборном [6, 18] і використовуваний на практиці, – *одноразовий блокнот*. Характерною особливістю такого шифру є одноразове використання ключової послідовності. У літературі цю систему шифрування часто називають шифром *Вернама*. Ключ даного шифру має таку саму довжину, що і вхідний текст, і обирається абсолютно випадково.

У 1949 році було опубліковано роботу К. Шеннона [36, 41], в якій доведено абсолютну стійкість шифру Вернама. Робота Шеннона показує, що не існує інших шифрів із подібними властивостями. Це й призвело до

висновку з таким твердження: шифр Вернама – найбезпечніша криптографічна система з усіх наявних.

Дослідження К. Шеннона показали, що ідеальна секретність може бути досягнута за виконання наступних правил [36, 41]:

– ключ для шифрування вибирається випадково;

– довжина ключа повинна дорівнювати довжині тексту, що шифрується;

– ключ повинен використовуватися тільки один раз.

Відомо, що адитивний шифр можна легко зламати, якщо використовувати той самий ключ. Але, навіть і цей шифр може бути ідеальним, якщо для шифрування кожного символу застосовувати ключ, який обирається випадково з множини ключів (00, 01, 02, ..., n–1): якщо перший символ зашифрований за допомогою ключа 04, другий символ – ключа 02, третій – ключа 21 і так далі. Тоді атака тільки на зашифрований текст стає неможливою. У разі, якщо відправник змінює ключ, використовуючи кожного разу іншу випадкову послідовність цілих чисел, то і інші типи атак також будуть неможливі.

Для реалізації такої системи підстановок іноді використовують одноразовий блокнот. Цей блокнот складається з відривних аркушів, на кожному з яких надруковано таблицю з випадковими числами (ключами) k_i. Блокнот виконується у двох примірниках: один використовується відправником, а інший – отримувачем. Для кожного символу m_i повідомлення використовується свій ключ k_i з таблиці тільки один раз. Після того як таблиця використана, вона повинна бути вилучена з блокнота та знищена. Шифрування нового повідомлення починається з нового аркуша.

Цей шифр є абсолютно надійним, якщо набір ключів K дійсно випадковий і непередбачуваний. Навіть, якщо криптографічний аналітик спробує використовувати всі можливі набори ключів, заданих для зашифрованих даних, і відновити варіанти вхідних даних, то вони виявляться рівноймовірними. Тобто немає можливості вибрати вхідні дані, які були

надіслані.

Здавалося б, що завдяки такій перевазі одноразові системи слід застосовувати в всіх випадках, які вимагають абсолютної інформаційної безпеки. Проте можливості застосування одноразової системи обмежені суто практичними аспектами. Суттєвою особливістю є вимога одноразового використання випадкової ключової послідовності. Ключова послідовність із довжиною, не меншою за довжину повідомлення, повинна передаватися отримувачу повідомлення заздалегідь або окремо по деякому закритому каналу. Ця вимога не є занадто обтяжливою для передачі дійсно важливих одноразових повідомлень. Однак така вимога майже неможлива для сучасних систем обробки інформації, де потрібно шифрувати багато мільйонів символів.

У деяких варіантах одноразового блокнота вдаються до більш простого управління ключовою послідовністю, але це призводить до деякого зниження надійності шифру. Наприклад, ключ визначається зазначенням місця в книзі відомого відправнику і отримувачу повідомлення. Ключова послідовність починається з вказаного місця цієї книги й використовується так само, як у системі Віженера. Іноді такий шифр називають шифром з біжучим ключем. Управління ключовою послідовністю у такому варіанті шифру є набагато простіше, оскільки довга ключова послідовність може бути представлена в компактній формі. Але з іншого боку, ці ключі не будуть випадковими. Тому, у криптографічного аналітика з’являється можливість використовувати інформацію про частоту символів вхідної природної мови і реалізувати відповідну атаку.

Система шифрування Вернама є, по суті, окремим випадком системи шифрування Віженера зі значенням модуля $n = 2$. Конкретна версія цього шифру, запропонована у 1926 році Г. Вернамом, співробітником фірми *AT&T* (США), використовує двійкове представлення символів вхідних даних.

Кожний символ вихідного відкритого тексту з англійського алфавіту $\{A, B, C, D, ..., Z\}$, розширеного шістьма допоміжними символами (пробіл,

повернення каретки тощо), спочатку кодувався в п’ятибітовий блок (b_0, b_1, b_2, b_3, b_4) телеграфного коду Бодо. Випадкова послідовність двійкових ключів {k_0, k_1, k_2, …} попередньо записувалася на паперовій стрічці. Схему передачі повідомлення з використанням шифрування методом Вернама показано на рис. 2.16.

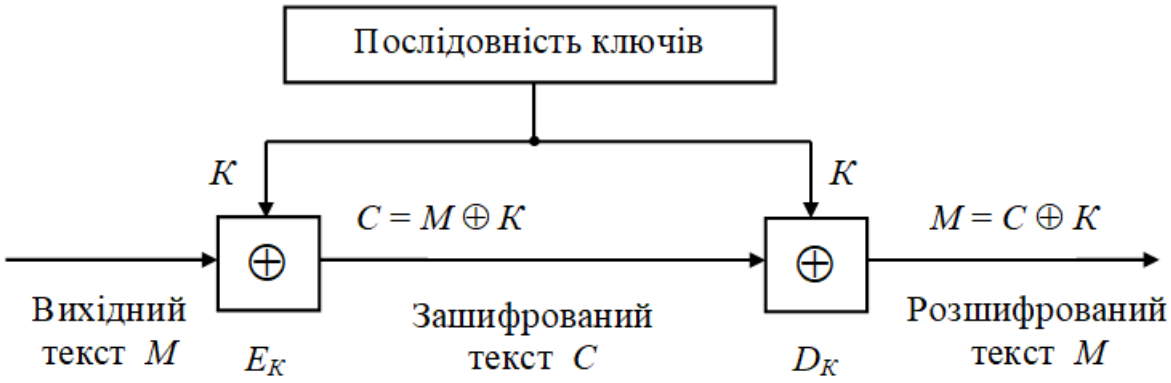


Рис. 2.16. Схема шифрування і розшифрування повідомлень методом Вернама

Шифрування вихідних даних, попередньо перетворених у послідовність двійкових символів m_i, здійснюється шляхом додавання за модулем два символів m_i з послідовністю ключів k_i.

Символи зашифрованих даних перетворюються наступним чином:

$$c_i = m_i \oplus k_i, \qquad i = 1, 2. 3, \ldots, n. \tag{2.15}$$

Розшифрування полягає в додаванні за модулем два символів зашифрованих даних c_i з тією самою послідовністю ключів k_i:

$$m_i = c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i, \qquad i = 1, 2. 3, \ldots, n. \tag{2.16}$$

При цьому послідовності ключів, що використовуються під час шифрування й розшифрування, компенсують один одного (при додаванні за модулем два), і внаслідок цього відновлюються символи вихідних даних m_i.

Розробляючи свою систему, Вернам перевіряв її за допомогою закільцьованих стрічок, встановлених на передавальній і приймальній сторонах для того, щоб використовувалася однакова послідовність ключів (рис. 2.17).

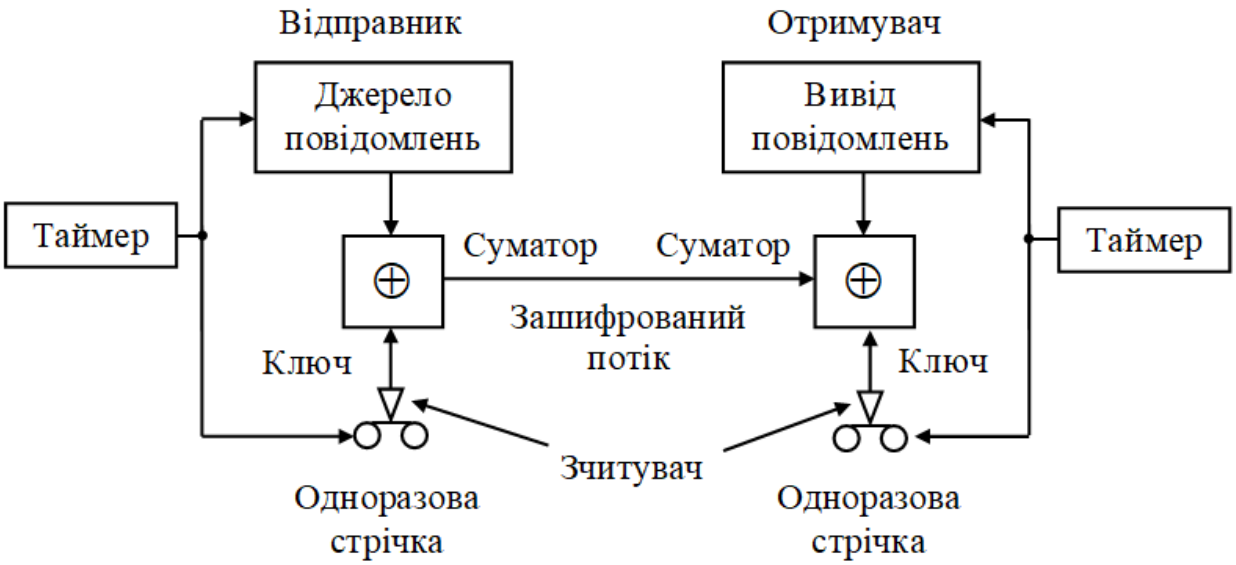


Рис. 2.17. Система шифрування Вернама, яка використовує закільцьовані стрічки

У реальних системах спочатку готують дві однакові стрічки з випадковими цифрами ключа. Одна залишається у відправника, а інша – передається законному отримувачу так, щоб ніхто не перехопив, наприклад, кур'єром з охороною. Коли відправник хоче передати повідомлення, він спочатку перетворює його у двійкову форму і поміщає в пристрій, який до кожної цифри повідомлення додає за модулем два цифри, визначені в ключовій стрічці. На приймаючій стороні зашифроване повідомлення записується та пропускається через машину, схожу на пристрій, який використовується для шифрування. Цей пристрій до кожної двійкової цифри повідомлення додає (віднімає, оскільки операції додавання й віднімання за модулем два еквівалентні) за модулем два цифри, визначені в ключовій стрічці, отримуючи, таким чином, відкритий текст. При цьому ключова стрічка повинна просуватися абсолютно синхронно із своїм дублікатом, що використовується для шифрування.

Головним недоліком цієї системи є те, що для кожного біта переданої інформації повинен бути заздалегідь підготовлений біт ключової інформації, причому ці біти повинні бути випадковими. Для шифрування великого обсягу даних це є серйозним обмеженням. Тому дана система використовується тільки для передачі повідомлень найвищої секретності.

Щоб обійти проблему попередньої передачі секретного ключа великого об'єму, інженери й винахідники придумали багато схем генерації дуже довгих потоків псевдовипадкових чисел із декількох коротких потоків відповідно до деякого алгоритму. Отримувача зашифрованого повідомлення при цьому необхідно забезпечити точно таким самим генератором, як і у відправника. Але такі алгоритми додають регулярність в зашифрований текст, виявлення якої може допомогти криптографічному аналітику дешифрувати повідомлення. Один з основних методів побудови подібних генераторів полягає у використанні двох або більше бітових стрічок, в яких визначені дані побітно додаються для отримання змішаного потоку (рис. 2.18).

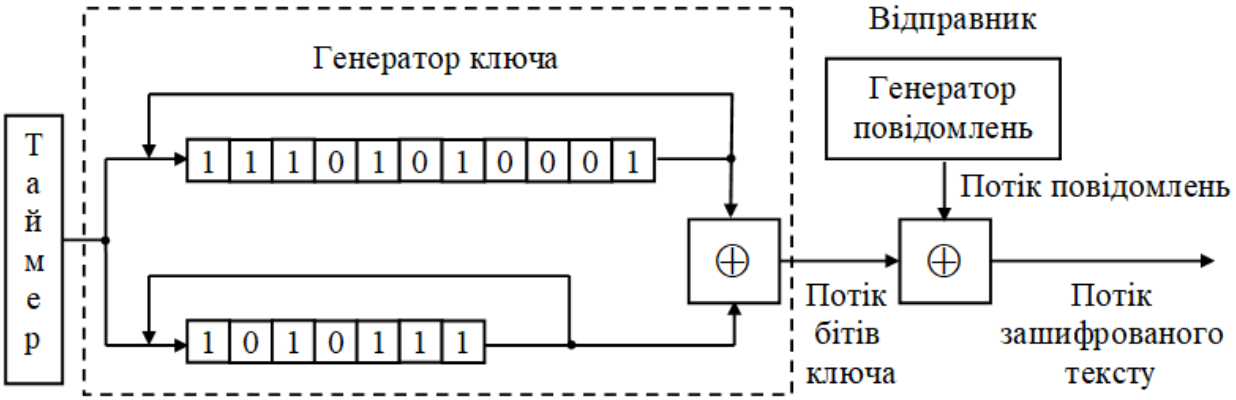


Рис. 2.18. Система шифрування Вернама, яка використовує змішаний потік

Слід зазначити, що метод Вернама не залежить від довжини послідовності ключів і, крім того, дозволяє використовувати випадкову послідовність ключів. Однак за реалізації методу Вернама виникають серйозні проблеми, пов'язані з необхідністю доставки отримувачу тієї самої

послідовності ключів як у відправника, або з необхідністю безпечного зберігання ідентичних послідовностей ключів у відправника й отримувача. Ці недоліки системи шифрування Вернама усуваються при використанні шифрування методом гамування.

Шифрування методом гамування

Суть шифрування методом гамування полягає в тому, що символи даних, які шифруються, послідовно додаються із символами деякої спеціальної послідовності, яка називається гамою. Іноді такий метод представляють як накладення гами на вихідні дані, звідси і назва "*гамування*".

Процедуру накладення гами на вихідні дані можна здійснити двома способами.

У першому способі символи вихідного тексту й гами замінюються цифровими еквівалентами, які потім додаються за модулем n, де n – кількість символів в алфавіті, тобто:

$$c_i = (m_i + g_i) \bmod n, \qquad i = 1, 2. 3, \ldots, n, \tag{2.17}$$

де c_i, m_i, g_i – символи відповідно зашифрованого тексту, вихідного тексту й гами.

Приклад 2.19. Нехай відкритий текст "ГАМБІТ", числовий еквівалент якого відповідно до рис. 2.2 відповідає: 03 00 16 01 11 22. Гама: "МОДЕЛЬ" – 16 18 05 06 15 30. Для українського алфавіту n = 33.

Рішення. Для шифрування відкритого тексту скористаємося співвідношенням (2.17) і отримаємо:

$c_1 = (03 + 16) \bmod 33 = 19;$ $c_2 = (00 + 18) \bmod 33 = 18;$

$c_3 = (16 + 05) \bmod 33 = 21;$ $c_4 = (01 + 06) \bmod 33 = 07;$

$c_5 = (11 + 15) \bmod 33 = 26;$ $c_6 = (22 + 30) \bmod 33 = 19.$

Отже, вийшов зашифрований текст: 19 18 21 07 26 19, який з урахуванням рис. 2.2 представляється у вигляді: “ПОСЄЦП”.

За використання другого способу символи вихідного тексту й гами представляються у вигляді двійкового коду, а потім відповідні розряди додаються за модулем два:

$$c_{ij} = m_{ij} \oplus g_{ij}, \qquad i = 1, 2. 3, \ldots, n; \quad j = 0, 1, 2, \ldots, l, \tag{2.18}$$

де $\oplus$ – операція додавання за модулем два; c_{ij}, m_{ij}, g_{ij} – двійкові символи зашифрованого тексту, вихідного тексту й гами відповідно; n – кількість символів даних, які шифруються (кількість символів зашифрованих даних; кількість символів гами); l – кількість двійкових бітів вихідних символів, гами та зашифрованих символів.

Приклад 2.20. Нехай відкритий текст і гама як у наведеному вище прикладі.

Рішення. Для шифрування скористаємося співвідношенням (2.18):

$$\begin{array}{ll} & 00011 \quad 00000 \quad 10000 \quad 00001 \quad 01011 \quad 10110 \\ \oplus & 10000 \quad 10010 \quad 00101 \quad 00110 \quad 01111 \quad 11110 \\ \hline & 10011 \quad 10010 \quad 10101 \quad 00111 \quad 00100 \quad 01000 \end{array}$$

Маємо зашифроване повідомлення: 19 18 21 07 04 08, яке відповідає “ПОСЄҐЖ”.

Замість додавання за модулем два при гамуванні можна використовувати інші логічні операції, наприклад, перетворення за правилами логічної еквівалентності або логічної нееквівалентності. Така заміна рівнозначна введенню ще одного ключа, яким є вибір правила формування символів зашифрованого повідомлення із символів вихідного тексту й гами.

Стійкість шифрування методом гамування визначається, головним

чином, властивостями гами – тривалістю періоду й рівномірністю статистичних характеристик. Остання властивість забезпечує відсутність закономірностей за появи різних символів у межах періоду.

Розшифрування здійснюється шляхом застосування до символів зашифрованого тексту і гами оберненої операції:

$$m_i = (c_i - g_i) \bmod n, \qquad i = 1, 2. 3, \ldots, n,$$

або

$$m_{ij} = c_{ij} \oplus g_{ij}, \qquad i = 1, 2. 3, \ldots, n; \quad j = 0, 1, 2, \ldots, l.$$

Приклад 2.21. Розшифруємо зашифровані тексти із попередніх прикладів (2.19 та 2.20).

Рішення. Для першого способу:

$m_1 = (19 - 16) \bmod 33 = 03;$ $m_2 = (18 - 18) \bmod 33 = 00;$

$m_3 = (21 - 05) \bmod 33 = 16;$ $m_4 = (00 - 06) \bmod 33 = 01;$

$m_5 = (26 - 15) \bmod 33 = 11;$ $m_6 = (19 - 30) \bmod 33 = 22.$

Отже, вийшов відкритий текст: 03 00 16 01 11 22, який відповідає повідомленню “ГАМБІТ”.

Для другого способу:

$$\begin{array}{rcccccc} & 10011 & 10010 & 10101 & 00111 & 00100 & 01000 \\ \oplus & 10000 & 10010 & 00101 & 00110 & 01111 & 11110 \\ \hline & 00011 & 00000 & 10000 & 00001 & 01011 & 10110 \end{array}$$

Як бачимо, вийшов відкритий текст: 03 00 16 01 11 22, який також відповідає відкритому тексту “ГАМБІТ”.

Стійкість систем шифрування, заснованих на гамуванні, залежить від характеристик гами – її довжини й рівномірності розподілу ймовірності появи знаків гами.

Розділяють два різновиди шифрування методом гамування – з скінченною і нескінченною гамою. За кращих статистичних властивостей гами стійкість шифрування визначається тільки довжиною її періоду. При цьому, якщо довжина періоду гами перевищує довжину зашифрованого тексту, то такий шифр теоретично є абсолютно стійким. Однак це не означає, що дешифрування такого тексту взагалі неможливе: за наявності деякої додаткової інформації вхідний текст може бути частково або повністю відновлений навіть за використання нескінченної гами.

В якості нескінченної гами може бути використана будь-яка послідовність випадкових символів, наприклад, послідовність цифр числа π або e. За шифрування електронно-обчислювальною машиною послідовність гами формується за допомогою датчика псевдовипадкових чисел. У даний час розроблені алгоритми роботи таких датчиків, які забезпечують необхідні характеристики [4, 5, 36].

Роторний шифр підстановки

Хоча шифри одноразового блокнота не застосовуються на практиці, один крок від нього до більш захищеного шифру – роторний шифр підстановки. Він повертається до ідеї моноалфавітної підстановки, але змінює принцип відображення вихідного тексту в символи зашифрованого тексту для кожного символу вихідного тексту. Рис. 2.19 показує спрощений приклад роторного шифру.

Ротор, показаний на рис. 2.19, застосовано лише для 6 букв, але реальні ротори використовують 26 букв англійського алфавіту. Ротор постійно зв’язує символи вихідного й зашифрованого текстів, але підключення забезпечується щітками. Зверніть увагу, що з’єднання символів вихідного й зашифрованого текстів показано так, якби ротор був прозорий і можна було побачити внутрішню частину. Початкова установка (позиція) ротора – ключ засекречування між відправником і отримувачем – це зашифрований перший символ вихідного тексту. Використовуючи початкову установку, другий

символ зашифрований після того, як проведено перше обертання (на рис. 2.19 – це поворот на 1/6 кола, на реальній установці – поворот на 1/26), і так далі.

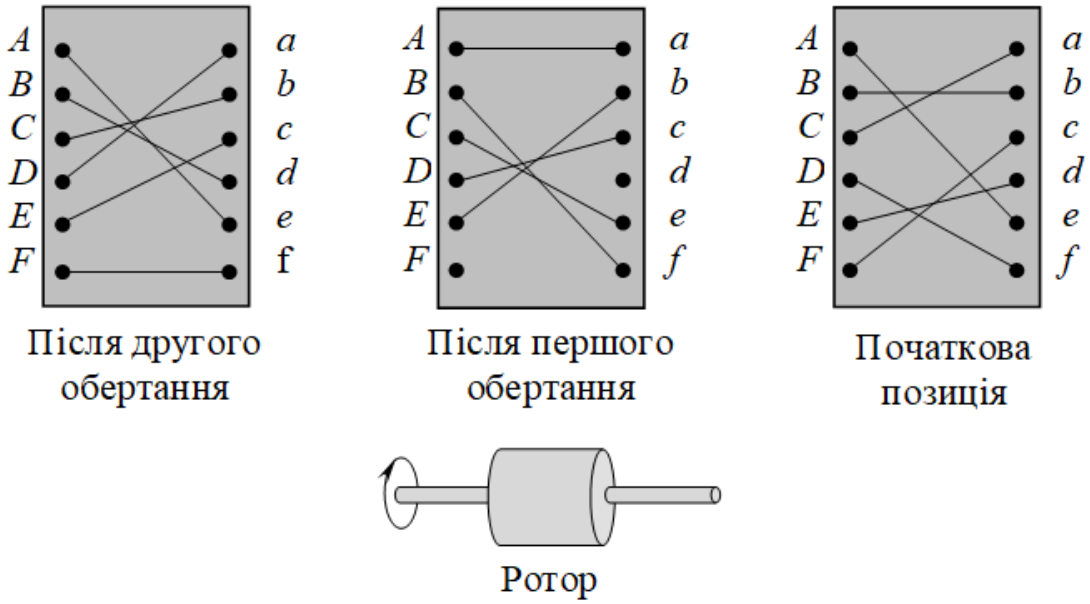


Рис. 2.19. Роторний шифр підстановки

Слово з трьома буквами, такими як *bee*, зашифровано як *BAA*, якщо ротор нерухомий (моноалфавітний шифр підстановки), але воно буде зашифровано як *BCA*, якщо він обертається (роторний шифр підстановки). Це показує, що роторний шифр підстановки – багатоалфавітний шифр, тому що дві появи того самого символу вихідного тексту зашифровані як різні символи.

Роторний шифр підстановки є стійким до атаки “грубої сили”, як одноалфавітний шифр підстановки, тому що зловмисник повинен знайти першу множину відображень серед можливих $n!$. Роторний шифр підстановки є набагато стійкішим до статистичної атаки, ніж одноалфавітний шифр підстановки, тому що в ньому не зберігається частота використання букв алфавіту.

Роторна машина підстановки “Енігма” спочатку була винайдена в Сербії, але фахівці німецької армії змінили її і інтенсивно використовували

протягом Другої світової війни [2, 24, 36]. Машина базувалася на принципі роторних шифрів. Рис. 2.20 показує спрощену схему побудови машини “Енігма”.

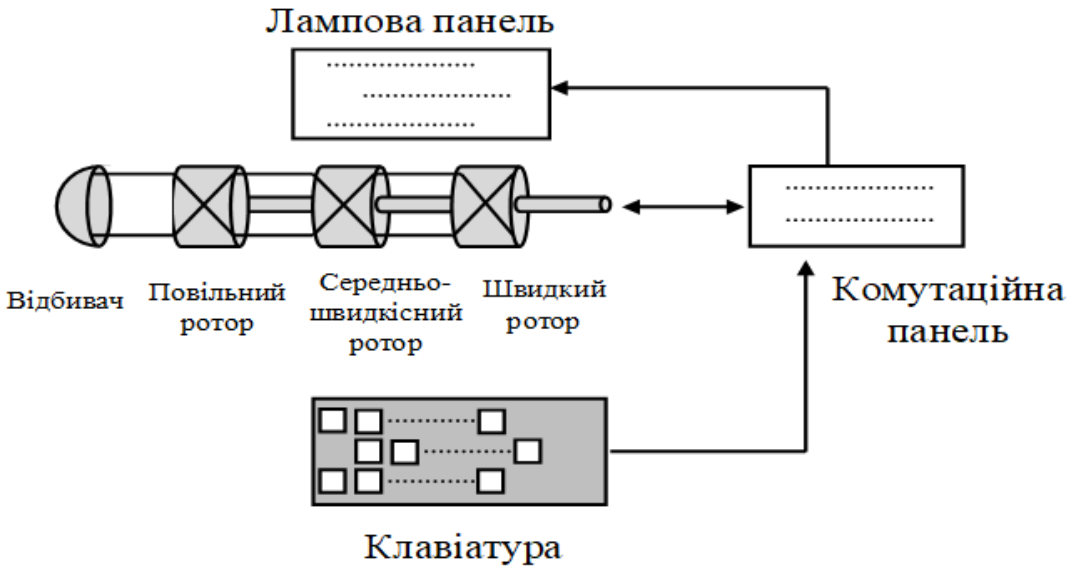


Рис. 2.20. Спрощена схема машини “Енігма”

Нижче перераховано основні компоненти машини:

1. Клавіатура з 26-ма клавішами, використовуваними для того, щоб вводити вихідний текст під час шифрування, і для того, щоб вводити зашифрований текст під час розшифрування.

2. Лампова панель з 26-ма лампами, яка показує символи зашифрованого тексту під час шифрування або символи вихідного тексту під час розшифрування.

3. Комутаційна панель з 10-ю парами комутаційних проводів. Конфігурація змінюється кожного дня, щоб забезпечити різне скремблювання.

4. Три вбудовані ротори, такі самі, як розглянуті на попередньому рисунку. Ці три ротори вибираються щодня з п’яти доступних роторів. Швидкий ротор обертається на 1/26 повороту для кожного символу, уведеному за допомогою клавіатури. Середній ротор робить 1/26 повороту для кожного повного повороту швидкого ротора. Повільний ротор робить

1/26 повороту для кожного повного повороту середнього ротора.

5. Відбивач, який є постійним і попередньо вбудованим.

Щоб використовувати машину “Енігма”, було видано кодову книгу, яка протягом кожного дня дає кілька параметрів налаштування, включаючи:

- три ротора, які повинні бути обрані з п’яти доступних;
- порядок, в якому ці ротори повинні бути встановлені;
- параметри установок для комутаційної панелі;
- код з трьома буквами дня.

Відомо, що машину “Енігма” під час війни було зламано, хоча німецька армія й інша частина світу не знала про цей факт ще кілька десятиліть після того [36]. Хоча німецька мова є надто складною, союзники, так чи інакше, отримали деякі копії машин. Наступним кроком був пошук параметрів установки для кожного дня і коду, переданого для ініціалізації роторів для кожного повідомлення. Винахід першого комп’ютера допоміг союзникам подолати й ці труднощі.

2.2. Шифри перестановки

Шифр перестановки не замінює один символ на інший, натомість він змінює місце розташування символів в зашифрованому тексті. Символ у першій позиції вихідного тексту може з’явитися в одинадцятій позиції зашифрованого тексту. Символ, який знаходиться у п’ятій позиції вихідного тексту, може з’явитися в першій позиції зашифрованого тексту. Іншими словами, шифр перестановки ставить символи в іншому порядку (переміщує).

2.2.1. Шифри перестановки без використання ключа

Прості шифри перестановки, які застосовувалися в минулому, не використовували ключ. Існує два методи для перестановки символів. У

першому методі текст записується в таблицю стовпець за стовпцем і потім передається рядок за рядком. У другому методі навпаки, текст записується в таблицю рядок за рядком і потім передається стовпець за стовпцем.

Приклад 2.22. Хороший приклад шифру без використання ключа – *шифр огорожі*. У цьому шифрі вихідний текст розміщений на двох лініях як зигзагоподібний шаблон (що може розглядатися як стовпець за стовпцем таблиці, яка має два рядки); зашифрований текст складається під час читання шаблона рядок за рядком. Наприклад, щоб передати повідомлення *Meet me at the park* (*Зустрічай мене в парку*), відправник пише отримувачу (рис. 2.21).

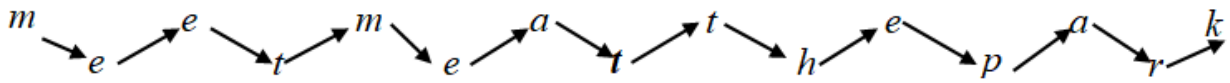


Рис. 2.21. Приклад шифрування повідомлення *Meet me at the park* за допомогою шифру огорожі

Відправник створює зашифрований текст *MEMATEAKETETHPR*, посилаючи перший рядок, супроводжуваний другим рядком. Отримувач отримує зашифрований текст і поділяє його навпіл (у цьому випадку друга половина має на один символ менше). Перша половина форми – перший рядок; друга половина – другий рядок. Отримувач читає результат по зигзагу. Оскільки немає ніякого ключа і встановлено два рядків, криптографічний аналіз зашифрованого тексту був би дуже простий для зловмисника. Усе, що він повинен знати, – це те, що використовується шифр огорожі.

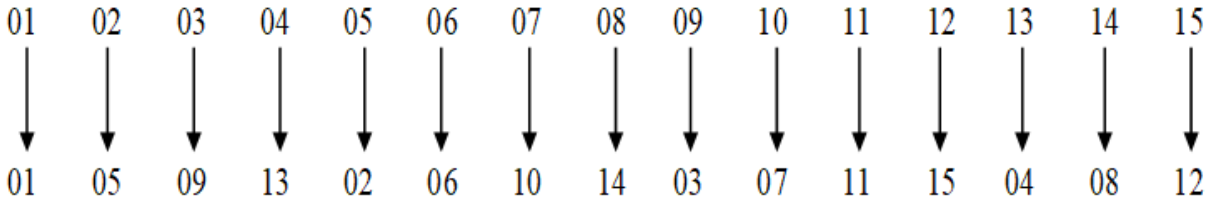
Приклад 2.23. Відправник і отримувач можуть домовитися про кількість стовпців і використовувати другий метод. Відправник пише той самий вихідний текст, рядок за рядком, у таблицю із чотирьох стовпців (рис. 2.22).

m	*e*	*e*	*t*
m	*e*	*a*	*t*
t	*h*	*e*	*p*
a	*r*	*k*	

Рис. 2.22. Приклад шифрування повідомлення *Meet me at the park* за допомогою таблиці

Відправник створює зашифрований текст *MMTAEEHREAEKTTP*, передаючи символи стовпець за стовпцем. Отримувач отримує зашифрований текст і застосовує обернений процес. Він пише отримане зашифроване повідомлення стовпець за стовпцем і читає його рядок за рядком як вихідний текст. Зловмисник може легко розшифрувати повідомлення, якщо він знає число стовпців.

Шифр у прикладі 2.23 – реальний шифр перестановки. Далі покажемо перестановку кожної літери початкового тексту в зашифрований текст, базуючись на номерах їх позицій:



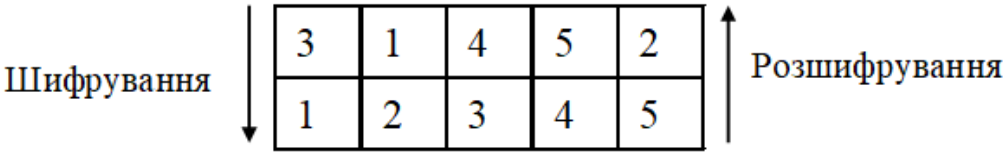
Другий символ у початковому тексті пересунувся на п’яту позицію в зашифрованому тексті; третій символ пересунувся на дев’яту позицію; і так далі. Хоча символи й переставлені, вони самі і є шаблонами: (01, 05, 09, 13), (02, 06, 10, 14), (03, 07, 11, 15) і (04, 08, 12). У кожній секції різниця між двома суміжними номерами – 4.

2.2.2. Шифри перестановки з використанням ключа

Безключові шифри перестановки переставляють символи, використовуючи запис вихідного тексту одним способом (наприклад, рядок за рядком) і передачу цього тексту в іншому порядку (наприклад, стовпець за стовпцем). Перестановка відбувається в усьому початковому тексті, щоб створити весь зашифрований текст. Інший метод полягає в тому, щоб розділити вихідний текст на групи заздалегідь визначеного розміру, звані блоками, а потім використовувати ключ, щоб переставити символи в кожному блоці окремо.

Приклад 2.24. Відправник повинен передати отримувачу повідомлення *Enemy attacks tonight* (*Ворожі атаки сьогодні ввечері*). Відправник і отримувач домовились розділити текст на групи по п’ять символів і потім переставити символи в кожній групі. Нижче показано групування після додавання фіктивного символу в кінці, щоб зробити останню групу однаковою за розміром з іншими: *Enemy attac kston ightz.*

Ключ, використовуваний для шифрування і розшифрування, – ключ перестановки, який показує, як переставляти символи. Для цього повідомлення приймемо, що відправник і отримувач використовували наступний ключ:



Третій символ у блоці вихідного тексту стає першим символом у блоці зашифрованого тексту, перший символ у блоці вихідного тексту стає другим символом у блоці зашифрованого тексту і так далі. Результати перестановки наступні:

EEMYN TAACT TKONS HITZG.

Відправник передає зашифрований текст *EEMYNTAACTTKONSHITZG* отримувачу. Отримувач ділить зашифрований текст на групи по п’ять символів і, використовуючи ключ в оберненому порядку, знаходить вихідний текст.

Шифри перестановки стовпців за ключа

Сучасні шифри перестановки, щоб досягти кращого скремблювання, об’єднують два підходи. Шифрування і розшифрування відбувається за три кроки.

Перший крок: вихідний текст записується в таблицю рядок за рядком.

Другий крок: робиться перестановка, змінюючи порядок слідування стовпців (переставляються стовпці в таблиці по заданому ключу).

Третій крок: зашифрований текст читається з нової таблиці стовпець за стовпцем.

Перший і третій кроки забезпечують безключову глобальну зміну порядку слідування; другий крок забезпечує блокову ключову перестановку. Ці типи шифрів згадуються часто як *шифри перестановки стовпців за ключем*.

Приклад 2.25. Припустимо, що відправник знову зашифровує повідомлення *Enemy attacks tonight*, на цей раз використовуючи об’єднаний підхід. Шифрування і розшифрування показано на рис. 2.23.

Перша таблиця, створена відправником, містить вихідний текст, записаний рядок за рядком. Стовпці переставлені з використанням того самого ключа, що і у попередньому прикладі. Зашифрований текст створений за допомогою читання символів з другої таблиці стовпець за стовпцем. Отримувач робить ті самі три кроки, але в оберненому порядку. Він записує зашифрований текст у першу таблицю стовпець за стовпцем, переставляє стовпці за ключем, а потім читає символи з другої таблиці рядок за рядком.

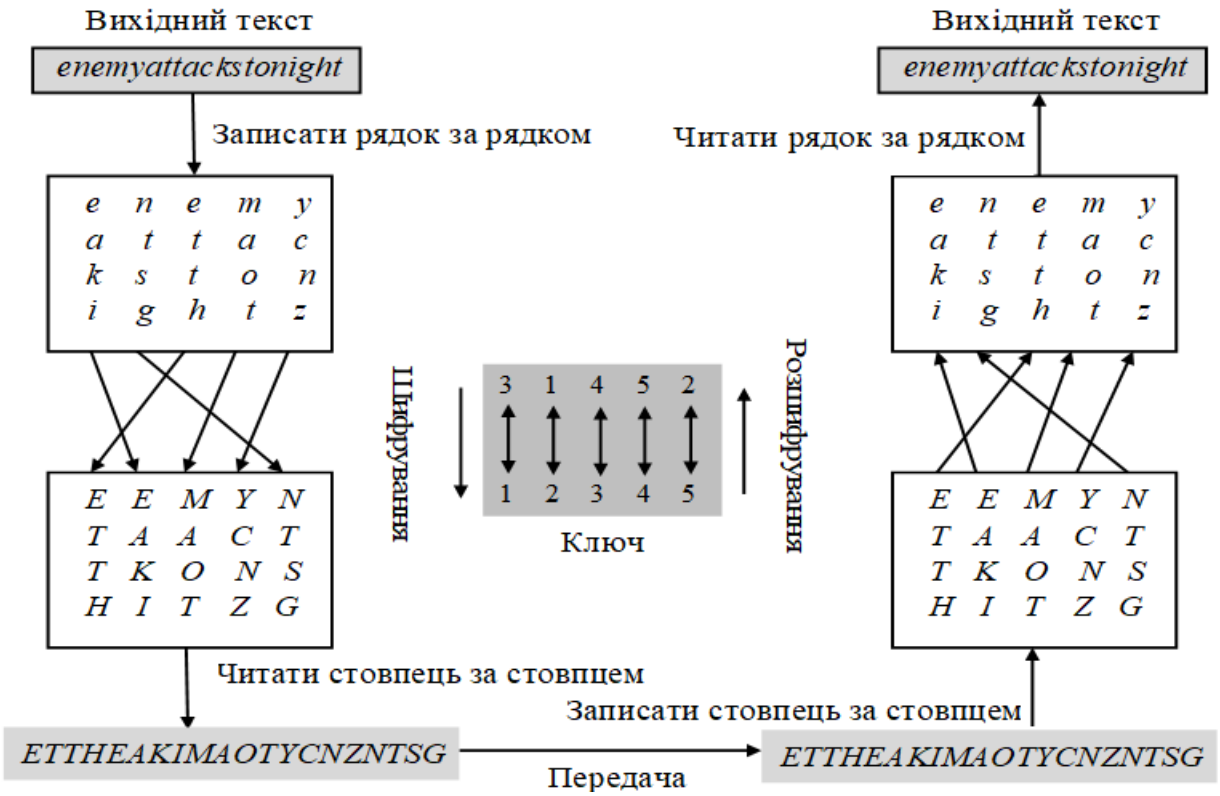


Рис. 2.23. Приклад шифрування і розшифрування шляхом перестановки з використанням ключа

У прикладі 2.25 єдиний ключ використовувався у двох напрямках для зміни порядку слідування стовпців – вниз для шифрування, вгору для розшифрування. Зазвичай прийнято створювати два ключа для цього графічного представлення: один для шифрування й один для розшифрування. Ключі накопичуються в таблицях, які мають одну адресу (вхід) для кожного стовпця. Рис. 2.24 показує, як ці дві таблиці можуть бути створені за допомогою графічного представлення ключа.

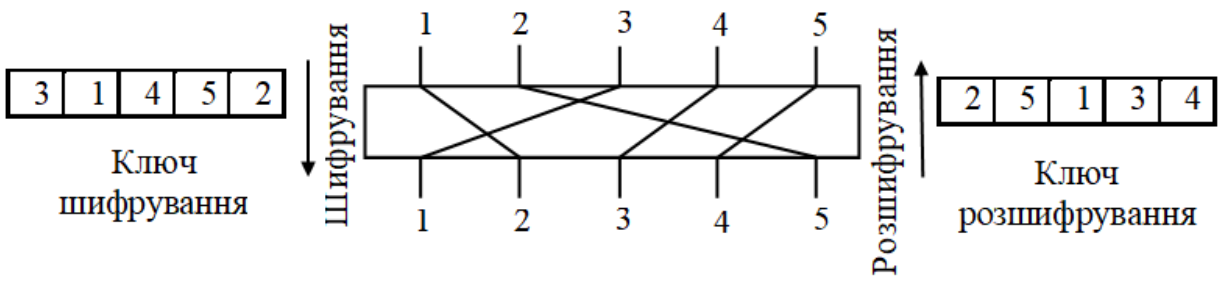


Рис. 2.24. Графічне представлення ключів шифрування/розшифрування в шифрі перестановки

Ключ шифрування – (3 1 4 5 2). Перше значення “3” показує, що 3-ій стовпець (зміст) в джерелі стає 1-им стовпцем (положення або індекс входу) в пункті призначення. Ключ розшифрування – (2 5 1 3 4). Перше значення “2” показує, що 2-ий стовпець в джерелі стає 1-им стовпцем у пункті призначення.

Як знайти ключ розшифрування, якщо дано ключ шифрування або, навпаки, дано ключ розшифрування, а потрібно знайти ключ шифрування? Процес може бути виконаний вручну за кілька кроків, як це показано на рис. 2.25.

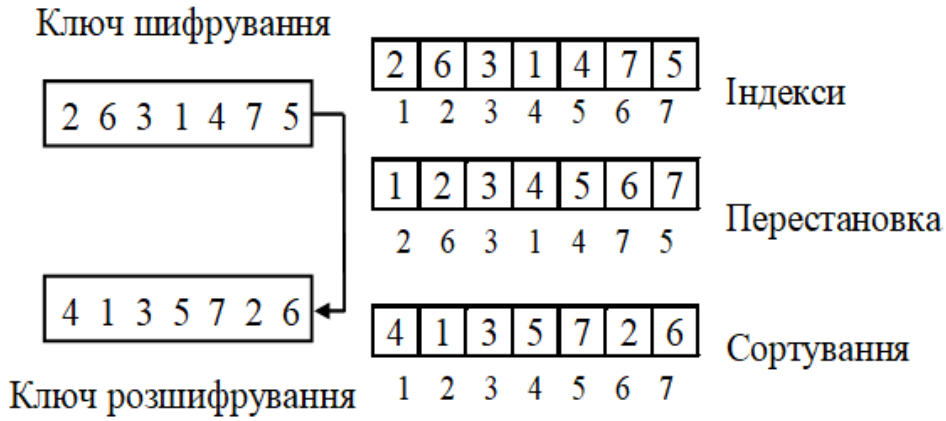


Рис. 2.25. Пояснення процесу інверсії ключа в шифрі перестановки

Спочатку додамо індекси до таблиці ключів, потім зробимо перестановку індексів та значень ключа, і, нарешті, проведемо сортування пар відповідно до індексу.

Можна використовувати матриці, щоб показати процес шифрування/розшифрування для шифру перестановки. Вихідний текст $M\,(l\times r)$ і зашифрований $C\,(l\times r)$ – матриці розміром $(l\times r)$, що представляють числові значення символів; ключі $K\,(r\times r)$ – квадратні матриці перестановки розміром $(r\times r)$. У такому випадку алгоритм шифрування даних можна представити у вигляді:

$$C\,(l\times r) = M\,(l\times r) \times K_{\text{Ш}}\,(r\times r), \qquad (2.19)$$

а розшифрування

$$M\,(l\times r) = C\,(l\times r) \times K_{\text{Р}}\,(r\times r), \qquad (2.20)$$

де $K_{\text{Ш}}\,(r\times r)$ і $K_{\text{Р}}\,(r\times r)$ – ключові матриці, які використовуються для шифрування і розшифрування відповідно. Ключові матриці є оберненими, тобто:

$$K_{\text{Ш}}\,(r\times r) = K_{\text{Р}}^{-1}\,(r\times r) \qquad \text{та} \qquad K_{\text{Р}}\,(r\times r) = K_{\text{Ш}}^{-1}\,(r\times r).$$

У даному алгоритмі ключові матриці описують процес перестановки стовпців, тому їх частіше називають матрицями перестановки. У матрицях перестановки кожний рядок або стовпець мають строго одну одиницю (1), а інша частина значень – нулі (0). Шифрування виконується множенням матриці вихідного тексту на ключову матрицю шифрування (пряма перестановка), щоб отримати матрицю зашифрованого тексту; розшифрування виконується множенням матриці зашифрованого тексту на ключову матрицю розшифрування (обернена перестановка), після чого отримуємо матрицю вихідного тексту. Цікаво, що ключова матриця розшифрування у цьому випадку, як і завжди, – обернена ключова матриця шифрування. Однак немає ніякої необхідності шукати обернену матрицю розшифрування – ключова матриця шифрування може просто бути переставлена зсувом рядків і стовпців, щоб отримати ключову матрицю розшифрування.

Приклад 2.26. Повідомлення, яке передається: *Enemy attacks tonight*. Зашифрувати повідомлення з використанням перестановки стовпців матриці вихідних даних за ключем шифрування: (3 1 4 5 2).

Рішення. Повідомлення *Enemy attacks tonight* в цифровому еквіваленті має вигляд 04, 13, 04, 12, 24, 00, 19, 19, 00, 02, 10, 18, 19, 14, 13, 08, 06, 07, 19, 25. Рис. 2.26 показує процес шифрування повідомлення.

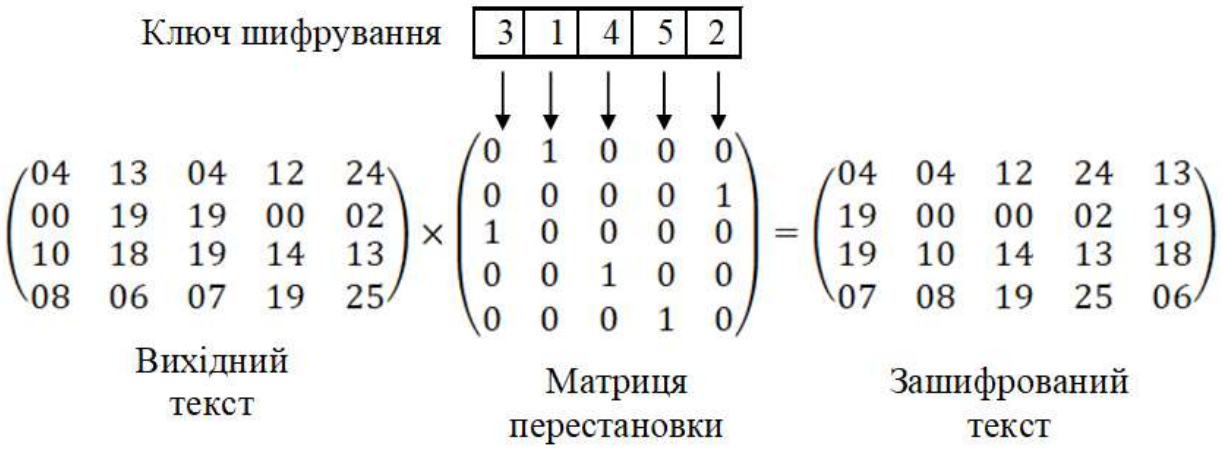


Рис. 2.26. Пояснення процесу шифрування даних для прикладу 2.26 з використанням шифру перестановки

Множення матриці вихідного тексту M (4×5) на ключову матрицю шифрування $K_{Ш}$ (5×5) дає матрицю зашифрованого тексту C (4×5). Матрична маніпуляція вимагає заміни символів у прикладі 2.26 до їх числових значень (від 00 до 25). Зверніть увагу, що матричне множення забезпечує тільки перестановку стовпців; читання і запис у матрицю повинні бути забезпечені іншою частиною алгоритму.

Отже, зашифроване повідомлення набуде вигляду 04, 19, 19, 07, 04, 00, 10, 08, 12, 00, 14, 19, 24, 02, 13, 25, 13, 19, 18, 06, що відповідає: *ETTHEAKIMAOTYCNZNTSG*.

Приклад 2.27. Розшифрувати повідомлення отримане у прикладі 2.26 із використанням перестановки стовпців матриці зашифрованих даних за ключем (2 5 1 3 4), яка є оберненою до матриці шифрування (див. приклад 2.26).

Рішення. Множення матриці зашифрованого тексту C (4×5) на ключову матрицю розшифрування $K_{Р}$ (5×5), дає матрицю вихідного тексту M (4×5).

Рис. 2.27 показує процес розшифрування повідомлення.

Отже, розшифрований текст набуде вигляду: 04, 13, 04, 12, 24, 00, 19, 19, 00, 02, 10, 18, 19, 14, 13, 08, 06, 07, 19, 25, що відповідає: *Enemy attacks tonightz*.

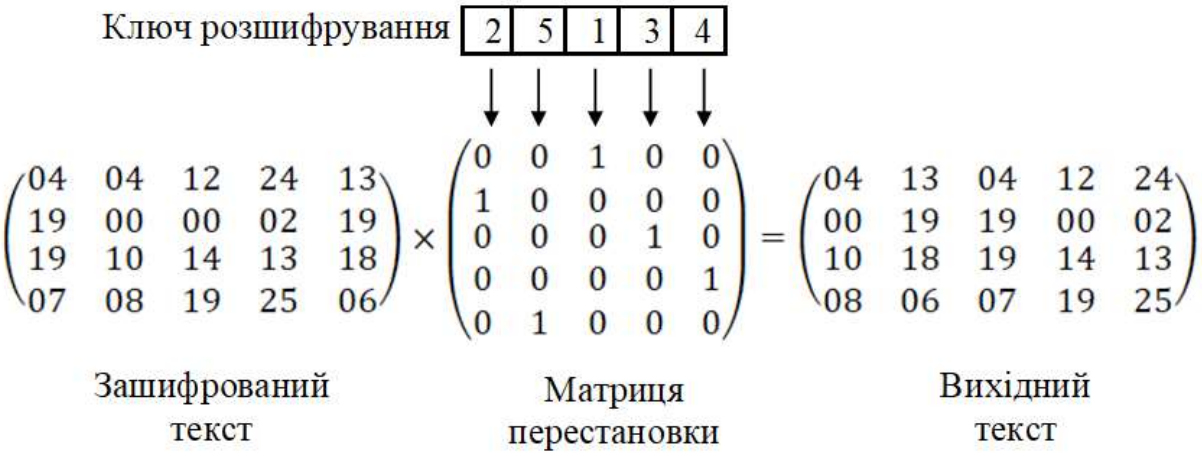


Рис. 2.27. Пояснення процесу розшифрування даних для прикладу 2.27 з використанням шифру перестановки

Криптографічний аналіз шифрів перестановки

Шифри перестановки вразливі до декількох видів атак тільки на зашифрований текст.

Статистична атака

Шифр перестановки не змінює частоту букв у зашифрованому тексті; а тільки переставляє їх. Так, перша атака, яка може бути застосована, – аналіз частоти появи в зашифрованому тексті окремих букв. Цей метод може бути корисний, якщо довжина зашифрованого тексту досить велика. Приклад такої атаки вже розглядався раніше. Однак шифри перестановки не зберігають частоту появи біграм і триграм. Це означає, що зловмисник не може використовувати такі інструментальні засоби. Фактично можна вважати, що якщо шифр не зберігає частоту появи біграм і триграм, але зберігає частоту появи окремих букв, то найімовірніше, що це і є шифр перестановки.

Атака "грубої сили"

Зловмисник, щоб розшифрувати повідомлення, може спробувати всі можливі ключі. Однак число ключів може бути величезним $1!+2!+3!+\ldots+L!$, де L – довжина зашифрованого тексту. Кращий підхід полягає в тому, щоб спробувати відгадати число стовпців. Зловмисник знає, що довжина

зашифрованого тексту (L) ділиться на число стовпців. Наприклад, якщо довжина зашифрованого тексту – 20 символів, то $20 = 1 \times 2 \times 2 \times 5$. Це означає, що номером стовпців може бути комбінація цих коефіцієнтів (1, 2, 4, 5, 10, 20). Однак варіанти тільки одного стовпця або тільки одного рядка – малоймовірні [4, 36].

Приклад 2.28. Припустимо, що зловмисник перехопив повідомлення зашифрованого тексту *ETTHEAKIMAOTYCNZNTSG*, яке отримане під час виконання прикладу 2.26. Дешифрувати цей шифротекст.

Рішення. Довжина повідомлення $L = 20$, число стовпців може бути 1, 2, 4, 5, 10 або 20. Зловмисник ігнорує перше значення, тому що це означає тільки один стовпець і воно малоймовірне.

1. Якщо число стовпців – 2, то маємо дві перестановки – (1 2) і (2 1). Перша означає, що перестановки не було. Зловмисник пробує другу комбінацію. Він ділить зашифрований текст на дві частини (два стовпця) і переставляє їх місцями, отримуючи текст *oe tt yt ch ne za nk ti sm*, який не має сенсу.

2. Якщо число стовпців – 4, то маємо 4! = 24 можливих перестановок. Перша перестановка (1 2 3 4) означає, що не було ніякої перестановки. Зловмисник повинен спробувати інші. Після випробування всіх 23 можливих зловмисник робить висновок, що будь-який вихідний текст за таких перестановках не має сенсу.

3. Якщо число стовпців – 5, то маємо 5! = 120 перестановок. Перша (1 2 3 4 5) означає відсутність перестановки. Зловмисник повинен спробувати інші. Перестановка (2 5 1 3 4) дає результат – вихідний текст *enemyattackstonightz*, який має сенс після видалення фіктивної букви *z* і додавання пробілів (*Enemy attacks tonight*).

Атака за зразком

Інша атака шифру перестановки може бути названа атакою за зразком. Зашифрований текст, створений за допомогою ключового шифру

перестановки, має деякі повторювані зразки. Наступний приклад показує зашифрований текст, відносно якого відомо, що кожний символ у зашифрованому тексті в прикладі 2.26 виходить із вихідного тексту за таким правилом:

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
03	08	13	18	01	06	11	16	04	09	14	19	05	10	15	20	02	07	12	17

1-й символ у зашифрованому тексті виходить з 3-го символу вихідного тексту. 2-й символ у зашифрованому тексті виходить з 8-го символу вихідного тексту. 20-й символ у зашифрованому тексті виходить з 17-го символу вихідного тексту, і так далі. Виходячи з вищезазначеного списку, маємо п’ять груп зразків: (3, 8, 13, 18), (1, 6, 11, 16), (4, 9, 14, 19), (5, 10, 15, 20) і (2, 7, 12, 17). У всіх групах різниця між двома суміжними номерами – 5. Ця регулярність може використовуватися криптографічним аналітиком, щоб зламати шифр. Якщо зловмисник знає або може припустити кількість стовпців (в цьому випадку вона дорівнює 5), тоді він може перетворити зашифрований текст у групи по чотири символи. Перестановка груп може забезпечити ключ до знаходження вихідного тексту.

Шифри з подвійною перестановкою

Шифри з подвійною перестановкою можуть ускладнити роботу криптографічного аналітика. Прикладом такого шифру було б повторення двічі алгоритму, який використовувався для шифрування і розшифрування в прикладі 2.26. На кожному кроці може застосовуватися різний ключ, але зазвичай ключ використовується той самий.

Приклад 2.29. Повторимо приклад 2.26, де використовується подвійна перестановка. Рис. 2.28 показує процес.

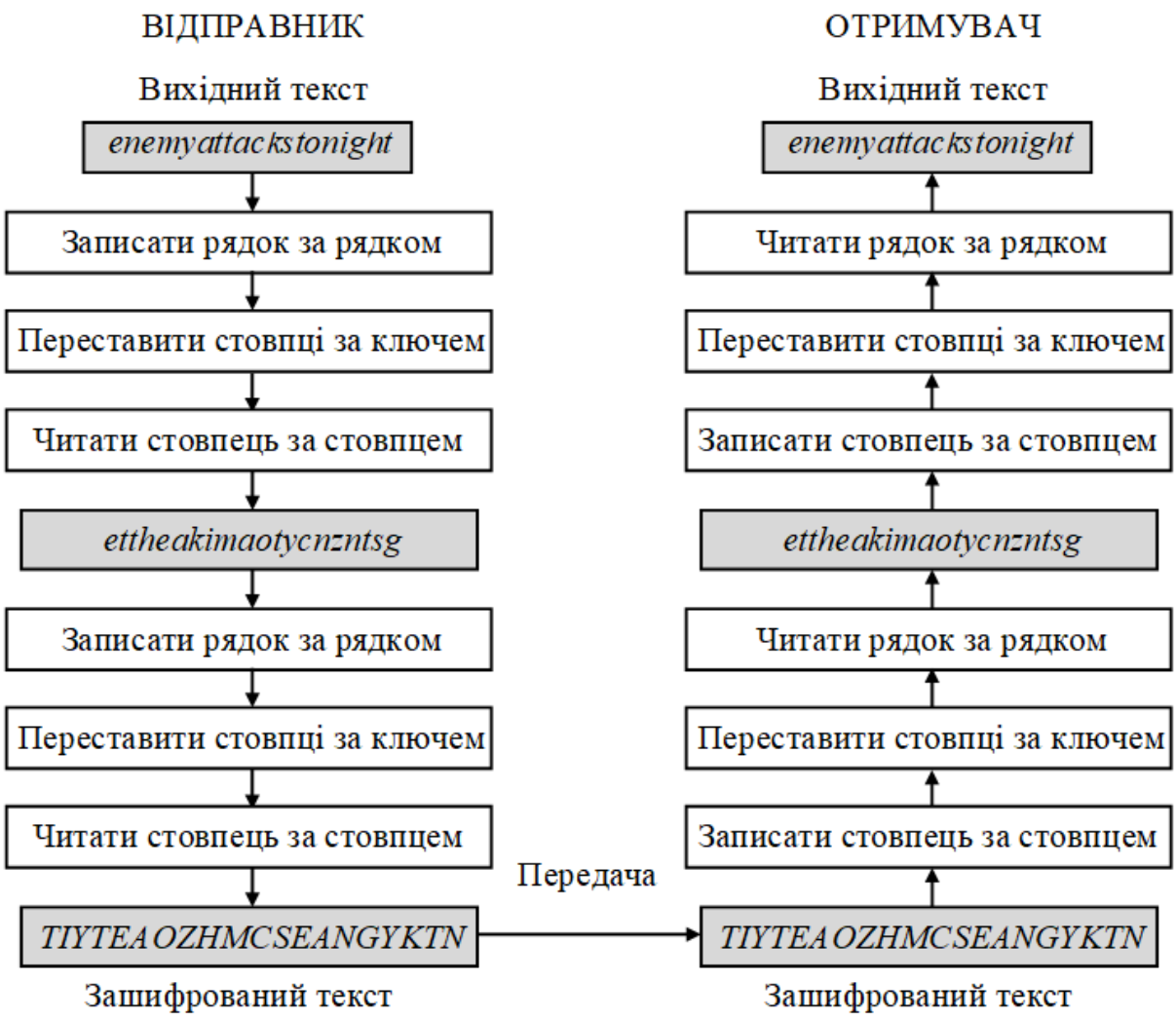


Рис. 2.28. Приклад шифрування і розшифрування шляхом подвійної ключової перестановки

Хоча криптографічний аналітик може ще використовувати частоту появи окремих символів для статистичної атаки на зашифрований текст, але атака за зразком тепер ускладнена:

13 16 05 07 03 06 14 20 18 04 10 12 01 09 15 17 08 11 19 02.

Порівнявши наведений результат і результат прикладу 2.26, бачимо, що тепер немає повторюваних зразків. Подвійна перестановка видалила ту регулярність, яка була раніше.

Контрольні питання до розділу 2

1. Поясніть загальну схему симетричного шифрування. Що спільного мають усі методи шифрування із закритим ключем?

2. Дайте визначення шифрів підстановки. Сформулюйте загальні принципи для методів шифрування підстановкою.

3. Поясніть сутність шифрів моноалфавітної підстановки.

4. Поясніть сутність шифрів багатоалфавітної підстановки.

5. Поясніть сутність адитивного шифру підстановки.

6. Поясніть сутність мультиплікативного шифру підстановки.

7. Поясніть сутність афінного шифру підстановки.

8. Поясніть сутність автоключового шифру підстановки.

9. Поясніть сутність шифру підстановки Плейфера.

10. Поясніть сутність шифру підстановки Віженера.

11. Поясніть можливі методи криптоаналізу шифру Віженера.

12. Поясніть сутність шифру Хілла та можливі методи його криптоаналізу.

13. Поясніть сутність одноразової системи шифрування.

14. Поясніть сутність шифрування методом гамування.

15. Дайте визначення шифру перестановки.

16. Поясніть сутність шифрів перестановки без використання та з використанням ключа.

17. Зашифрувати з використанням адитивного шифру підстановки (K = 3) повідомлення українською мовою “*бакалавр*” (див. рис. 2.2).

18. Розшифрувати з використанням адитивного шифру підстановки (K = 3) зашифроване повідомлення “*VBVWHP*” при шифруванні якого використовували англійську мову (див. рис. 2.1).

19. За допомогою частотного криптоаналізу відновіть текст ВЬСЄРУХЗТДЇЕЯЯХЧХВЙУХЗШСЄЧЖНГХЇЄЗФХМЄЖЬЩО, зашифрований алгоритмом Цезаря.

20. Використовуючи мультиплікативний шифр підстановки, зашифрувати повідомлення українською мовою “*магістр*” з ключем $K = 13$ (див. рис. 2.2).

21. Розшифрувати з використанням мультиплікативного шифру підстановки ($K = 11$) зашифроване повідомлення “*FRQWURO*”, при шифруванні якого використовували англійську мову (див. рис. 2.1).

22. Використовуючи афінний шифр ($k_1 = 13$ і $k_2 = 17$), зашифрувати повідомлення українською мовою “*захист*” (див. рис. 2.2).

23. За допомогою методу Казіскі доведіть, що наступний текст TCKQZANRAEGDNJAIWOVCJNBZVBOCWNRCPUSNFHKUSHCHDRAPH FJIWOVCJVBPBFANZEGMCESWGZANRAEGYESWGSIBFAYSWQSNFBK GTKYZKJSNFUNROPYSWQSNFVWISRVGEBBOUONRJEFWKAOJQWJFD EESKGVAEGPBQNROPRHDRWNBKJ, зашифрований алгоритмом Віженера з довжиною ключа 3. Знайдіть літери ключового слова та відновіть повідомлення.

24. Використовуючи автоключовий шифр підстановки ($K = 19$), зашифрувати повідомлення українською мовою “*кіберзахист*” (див. рис. 2.2).

25. Використовуючи шифр Плейфера (див. рис. 2.8), зашифрувати повідомлення українською мовою “*криптографія*”.

26. Шифротекст “*QOOFQMDFTFELTQDKBFFP*” розшифруйте за допомогою шифру Плейфера, використовуючи ключ “*work*”.

27. Використовуючи шифр Віженера і ключове слово “*клас*”, зашифрувати повідомлення українською мовою “*криптоаналіз*”.

28. Використовуючи шифр Віженера з ключовим словом “*time*”, зашифрвати відкритий текст “*like cures like*”.

29. Використовуючи шифр Хілла і ключову матрицю K із прикладу 2.18, зашифрувати повідомлення англійською мовою “*internet*”.

30. Розшифрувати шифротекст “*CVJRKDXAHIBS*”, при шифруванні якого використовували: шифр Хілла, ключову матрицю K^{-1} з прикладу 2.18 і англійську мову.

31. Дано вихідний текст “*letusmeetnow*” і відповідний зашифрований текст “*HBCDFNOPIKLB*”. Відомо, що використовувався шифр Хілла. Знайти ключову матрицю.

32. У криптосистемі Хілла з матрицею $\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}$ зашифруйте текст “*fly a kite*”.

33. У криптосистемі Хілла з матрицею $\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$ зашифруйте текст “*live and learn*”.

34. У криптосистемі Хілла з матрицею $K^{-1} = \begin{pmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{pmatrix}$ розшифруйте текст “*WRHSHXLASLQV*”.

35. Ключ шифрування в шифрі перестановки – (3 2 6 1 5 4). Знайти ключ розшифрування.

36. Покажіть матричне представлення ключа шифрування перестановки з ключем (7 3 2 1 6 4 5). Знайдіть матричне представлення ключа розшифрування.

37. Використовуючи шифр перестановки стовпців за ключем та ключ (5 4 2 3 6 1), зашифрувати повідомлення “*Криптологія поєднує у собі два взаємозалежні напрями – криптографію та криптоаналіз*”.

РОЗДІЛ 3
ПРИНЦИПИ ПОБУДОВИ СУЧАСНИХ БЛОКОВИХ СИСТЕМ ШИФРУВАННЯ

Розглянуті вище традиційні шифри із симетричним ключем орієнтуються на символи. З появою комп'ютерних систем необхідними стали шифри, орієнтовані на біти, тому що інформація, яку потрібно зашифровувати, не завжди може складатися тільки з тексту, але й також із чисел, графіки, аудіо - та відеоданих. Для шифрування зручно перетворити ці типи даних у потік бітів і потім вже передавати зашифрований текст. Крім того, коли текст оброблено на розрядному рівні, кожний символ замінено на 8 (або 16) бітів, а це означає, що кількість символів стає у 8 (або 16) разів більше. Саме змішування більшої кількості символів збільшує безпеку [11, 36, 38, 41].

3.1. Сучасні блокові шифри

Сучасні блокові шифри із симетричними ключами зашифровують *n*-бітовий блок вихідного тексту або розшифровують *n*-бітовий блок зашифрованого тексту. Алгоритми шифрування або розшифрування використовують *k*-бітовий ключ. Алгоритм розшифрування повинен бути інверсією алгоритму шифрування, і обидва в роботі використовують той самий секретний ключ так, щоб отримувач мав змогу відновити повідомлення, передане відправником. Рис. 3.1 показує загальну ідею шифрування і розшифрування у сучасному блоковому шифрі.

Якщо повідомлення має розмір менше, ніж *n* біт, потрібно додати заповнення, щоб створити цей *n*-розрядний блок; якщо повідомлення має більше, ніж *n* біт, воно повинно бути розділене на *n*-розрядні блоки, і у разі потреби додати до останнього блоку відповідне заповнення. Загальні значення для *n* зазвичай 64, 128, 256 або 512 бітів [11, 22, 24, 36].

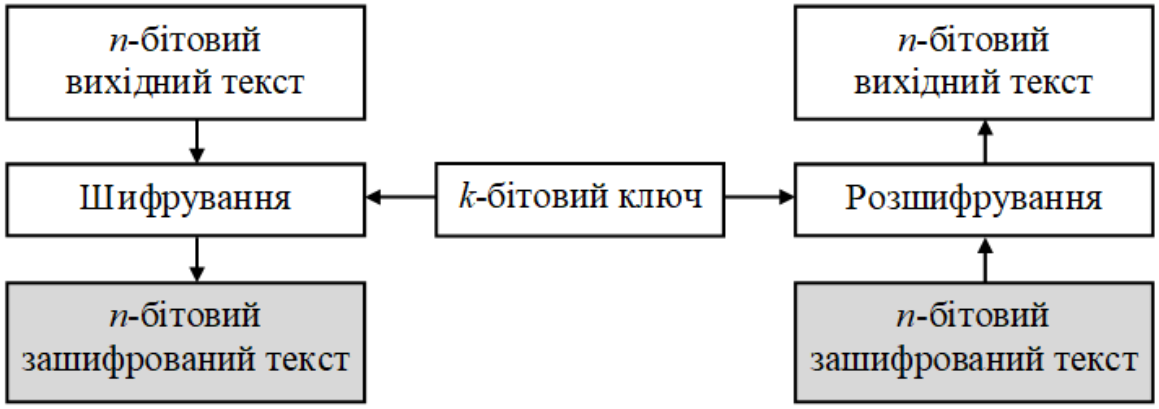


Рис. 3.1. Сучасний блоковий шифр

Приклад 3.1. Визначити кількість додаткових біт, які потрібно додати до повідомлення, що складається з 100 символів, якщо для кодування використовується *ASCII* коди по 8 бітів і блоковий шифр приймає блоки довжиною 64 біта.

Рішення. Кодуємо 100 символів, використовуючи *ASCII* коди по 8 бітів, тоді довжина повідомлення буде містити 800 біт. Вихідні дані повинні ділитися без залишку на 64. Якщо $|M|$ і $|Pad|$ – відповідно довжина повідомлення і довжина заповнення, то

$$(|M| + |Pad|)\text{mod}64 = 0.$$

Звідси випливає, що

$$|Pad| = -|M|\text{mod}64 = -800 \text{ mod}64 = -32\text{mod}64 = 32.$$

Це означає, що до повідомлення потрібно додати 32 біта заповнення (наприклад, нулів). Вихідні дані тоді будуть складатися з 832 бітів або тринадцяти 64-розрядних блоків. Зауважимо, що тільки останній блок містить заповнення. Шифратор використовує алгоритм шифрування тринадцять разів, щоб створити тринадцять блоків зашифрованих даних.

3.1.1. Шифри підстановки і транспозиції

Сучасний блоковий шифр може бути спроектований так, щоб діяти як шифр підстановки (заміни) або як шифр транспозиції (перестановки). Така сама ідея використовується і у традиційних шифрах, за винятком того, що замість символів, які будуть замінені або переміщені, містять біти.

Якщо шифр спроектований як шифр підстановки, значення біта 1 або 0 у вихідних даних можуть бути замінені або на 0, або на 1. Це означає, що вихідні та зашифровані дані можуть мати різне число одиниць. Наприклад, блок вихідних даних на 64 біта, який містить 12 нулів і 52 одиниці, може бути представлений у зашифрованих даних 34 нулями і 30 одиницями. Якщо шифр спроектований як шифр перестановки (транспозиції), біти тільки змінюють порядок проходження (переміщуються), зберігаючи те ж саме число символів у вихідних і зашифрованих даних. У будь-якому випадку, число можливих n-бітових вихідних або зашифрованих даних дорівнює 2^n, тому що кожний з n бітів, використаних у блоці, може мати одне з двох значень – 0 або 1.

Сучасні блокові шифри спроектовані як шифри підстановки, тому що властивості транспозиції (збереження числа одиниць або нулів) роблять шифр уразливим до атак вичерпного пошуку, як це показують нижченаведені приклади.

Приклад 3.2. Припустимо, є блоковий шифр, де $n = 64$ Нехай у зашифрованих даних міститься 10 одиниць, а решта даних – нулі (54 нулі). Скільки випробувань типу “проб і помилок” повинен зробити зловмисник, щоб отримати вихідні дані шляхом перехоплення зашифрованих даних у кожному з наступних випадків:

а) шифр спроектований як шифр підстановки;

б) шифр спроектований як шифр транспозиції.

Рішення. У першому випадку (підстановка) зловмиснику невідомо скільки одиниць знаходиться у вихідних даних. Тоді він повинен спробувати

усі можливі 2^{64} блоки по 64 біта, щоб знайти один, який має сенс. Перебір одного мільярда блоків за секунду міг би дати бажаний результат тільки через століття.

У другому випадку (транспозиція) зловмисник знає, що у вихідних даних є точно 10 одиниць, тому що транспозиція не змінює числа одиниць (або нулів) у зашифрованих даних. Зловмисник може почати атаку вичерпного пошуку, використовуючи тільки ті 64-бітові блоки, які мають точно 10 одиниць. Є тільки

$$\frac{n!}{(m_1)!\cdot(n-\mathrm{m}_1)!} = \frac{64!}{10!\cdot 54!} = 151473214816 \qquad (3.1)$$

блоків з 2^{64} слів по 64 біта, які мають точно 10 одиниць. У виразі (3.1) m_1 – число одиниць у зашифрованих даних. Зловмисник може перевірити їх усі менше ніж за три хвилини, якщо він може робити один мільярд випробувань за секунду.

Тому, сучасний блоковий шифр, спроектований як шифр підстановки, повинен бути стійкий до атаки вичерпного пошуку.

3.1.2. Блокові шифри як групові математичні перестановки

Необхідно визначити, чи є сучасний блоковий шифр математичною групою. Припустимо спочатку, що ключ у блоковому шифру досить довгий, щоб створити відображення будь-яких можливих вихідних даних у зашифровані. Такі блокові шифри називаються *повнорозмірними ключовими шифрами*. Практично, ключ менший; довгий ключ можна застосовувати тільки для деяких відображень вхідної інформації у вихідну. Хоча блоковий шифр повинен мати ключ, який є секретним при обміні між відправником і отримувачем, у шифрі використовуються також компоненти, що не залежать від ключа [5, 19, 23].

Повнорозмірні ключові шифри

Хоча повнорозмірні ключові шифри практично не використовуються, спочатку обговоримо їх, щоб зробити більш зрозумілим обговорення шифрів з ключем часткового розміру.

Повнорозмірний ключовий блоковий шифр *транспозиції* переміщує біти, не змінюючи їх значення, тому може бути змодельований як шифр перестановки n-мірного об'єкта з множиною $n!$ таблиць перестановки, в яких ключ визначає, яка таблиця використовується відправником і отримувачем. Для цього потрібно мати $n!$ можливих ключів, і такий ключ повинен мати довжину $[\log_2 n!]$ біт.

Приклад 3.3. Побудувати модель і множину таблиць перестановки для блокового шифру транспозиції на три біта, де розмір блоку – $n = 3$ біта.

Рішення. Множина таблиць перестановки має $n! = 3! = 6$ елементів, як показано на рис. 3.2.

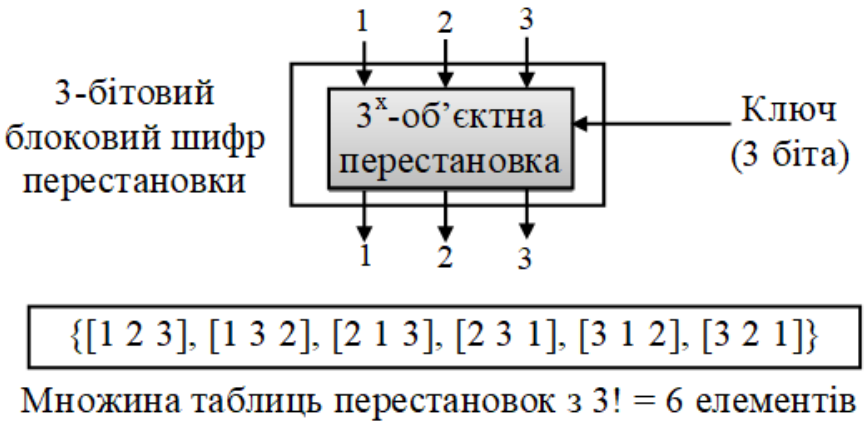


Рис. 3.2. Блоковий шифр транспозиції у вигляді перестановки

Ключ повинен бути довжиною $[\log_2 n!] = 3$ біта. Зауважимо, що хоч ключ на три біти може визначати $2^3 = 8$ різних відображень, використовуються тільки шість із них.

Повнорозмірні ключові блокові шифри *підстановки* на перший погляд здається, не можуть бути змодельовані як перестановка. Однак можна

застосувати модель перестановки і для шифру підстановки, якщо декодувати вхідну інформацію і кодувати вихідну. Таке декодування означає перетворення n-розрядного цілого числа в рядок із 2^n бітів з єдиною одиницею і 2^n-1 нулями [22, 36]. Позиція єдиної одиниці вказує на значення цілого числа у впорядкованій послідовності позицій рядка від 0 до 2^n-1. Оскільки нова вхідна інформація має завжди єдину одиницю, шифр може бути змодельований як перестановка 2^n! об'єктів.

Приклад 3.4. Побудувати модель і множину таблиць перестановки для блокового шифру підстановки блоку на три біти.

Рішення. Три вхідні блоки даних можуть бути позначені цілими числами від 0 до 7. Їх можна закодувати як рядок, що містить 8 бітів з єдиною одиницею. Наприклад, комбінація 000 може бути закодована як 00000001 (перша одиниця справа); комбінація 101 може бути закодована як 00100000 (шоста одиниця справа). Рис. 3.3 показує модель і множину таблиць перестановки.

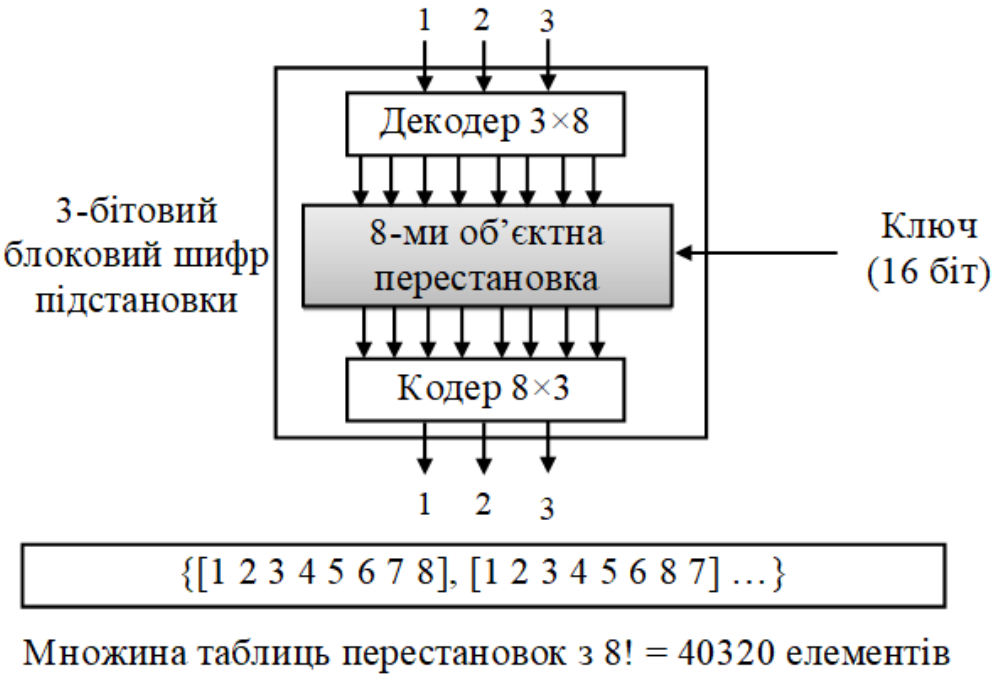


Рис. 3.3. Модель блокового шифру підстановки як шифру перестановки

Зауважимо, що число елементів у закодованій множині є набагато більшою, ніж число елементів у шифрі транспозиції (8! = 40320). Ключ – також набагато довший $[\log_2 40320] = 16$ біт. Хоча ключ на 16 бітів може визначити 65536 різних відображень, використовуються тільки 40320.

Повнорозмірний ключ – це *n*-розрядний шифр транспозиції або шифр підстановки. Такі шифри можуть бути змодельовані як шифр перестановки, але розміри їх ключів різні: для шифру транспозиції довжина ключа становить – $[\log_2 n!]$ біт, для шифру підстановки – $[\log_2 2^n!]$ біт.

Повнорозмірна ключова транспозиція або шифр підстановки/перестановки показує, що якщо шифрування або розшифрування використовує більше ніж одну будь-яку комбінацію із цих шифрів, результат еквівалентний операції групової перестановки. Відомо [20, 36], що дві або більше каскадні перестановки можуть завжди бути замінені однією перестановкою. Це означає, що марно мати більше ніж один каскад повнорозмірних ключових шифрів, тому що ефект той самий, як і за наявності одного кроку.

Шифри ключа часткового розміру

Фактичні шифри не можуть використовувати повнорозмірні ключі, тому що розмір ключа стає надто великим, особливо для блокового шифру підстановки. Наприклад, загальний шифр підстановки DES застосовує 64-розрядний блоковий шифр. Якби проектувальники DES використовували повнорозмірний ключ, він був би $\log_2 2^{64}! = 2^{2^{70}}$ бітів. На практиці ключ для DES – тільки 56 бітів, що є дуже маленьким фрагментом повнорозмірного ключа. Це означає, що *DES* використовує тільки 2^{56} відображень із приблизно $2^{2^{70}}$ можливих.

Шифри без ключа

Використання окремо шифру без ключа є фактично марним, але можливо його застосування в якості компонентів ключових шифрів.

Шифри транспозиції без ключа (або з фіксованим ключем) можна розглядати як шифр транспозиції, реалізований в апаратних засобах. Фіксований ключ (єдине правило перестановки) може бути представлений таблицею у разі реалізації шифру в програмному забезпеченні. Далі будуть розглянуті шифри транспозиції без ключа, названі *P*-блоками перестановки, які використовуються як стандартні блоки сучасних блокових шифрів.

Шифр підстановки без ключа (або з фіксованим ключем) можна уявити собі як заздалегідь визначене відображення вхідної інформації у вихідну. Відображення може бути представлене таблицею, як математична функція, а також іншими способами. Далі будуть розглянуті шифри підстановки без ключів, названі *S*-блоками заміни, які застосовуються як стандартні блоки сучасних блокових шифрів.

3.1.3. Компоненти сучасного блокового шифру

Сучасні блокові шифри зазвичай є ключовими шифрами підстановки, в яких ключ дозволяє тільки часткові відображення можливих входів інформації в можливі виходи. Однак ці шифри зазвичай не проектуються як єдиний модуль. Щоб забезпечувати необхідні властивості сучасного блокового шифру, такі як розсіювання і перемішування інформації, цей шифр формується як комбінація модулів транспозиції (*P*-блоківи перестановки), модулів підстановки (*S*-блоків заміни) і деяких інших модулів.

P-блок перестановки подібний традиційному шифру транспозиції символів. Він переміщає біти. У сучасних блокових шифрах можна знайти три типи *P*-блоків перестановки: прямі *P*-блоки; *P*-блоки стиснення і *P*-блоки розширення, які показані на рис. 3.4.

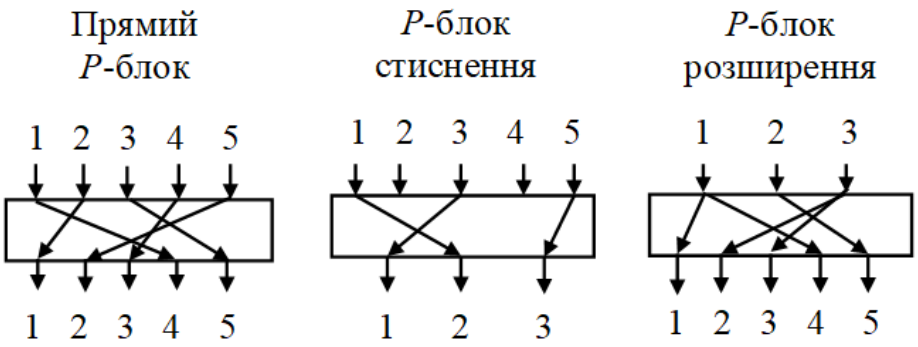


Рис. 3.4. Три типи *P*-блоків

На рис. 3.4 показано прямий *P*-блок 5×5 (5 входів та 5 виходів), *P*-блок стиснення 5×3 (5 входів та 3 виходу) і *P*-блок розширення 3×5 (3 входу та 5 виходів). Розглянемо кожен з них більш детально.

Прямі P-блоки

Прямий *P*-блок з *n* входами і *n* виходами – це перестановка з *n*! можливими відображеннями.

Приклад 3.5. На рис. 3.5 показано усі шість можливих відображень прямого *P*-блоку 3×3.

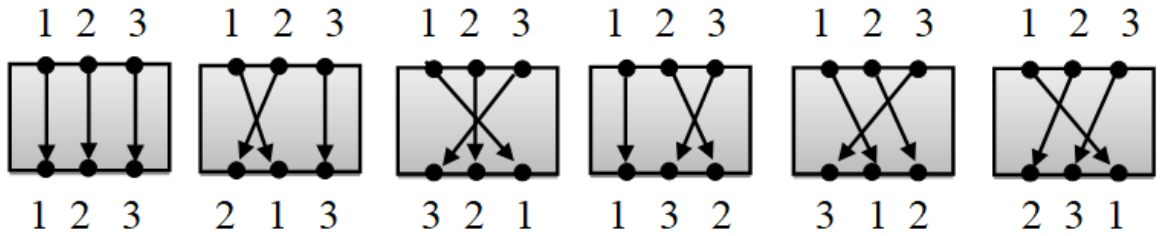


Рис. 3.5. Можливі відображення прямого *P*-блока 3×3

Хоча прямий *P*-блок може використовувати ключ, щоб визначити одне з *n*! відображень, зазвичай прямі *P*-блоки – без застосування ключа, тобто відображення задано заздалегідь. Якщо прямий *P*-блок заданий заздалегідь і реалізований в апаратних засобах, або якщо він реалізований в програмному забезпеченні, таблиці перестановок задають правило відображення. У табл. 3.1 показано приклад таблиці перестановок для прямого *P*-блоку, коли *n* дорівнює 32.

Таблиця 3.1

Приклад таблиці перестановки для прямого *P*-блоку

	Номера бітів															
Вхід	26	18	10	02	28	20	12	04	30	22	14	06	32	24	16	08
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	25	17	09	01	27	19	11	03	29	21	13	05	31	23	15	07
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Табл. 3.1 має 32 табличні виходи, які фіксують відповідність 32 інформаційним входам. Позиція (індекс) входу відповідає виходу. Наприклад, перший табличний вхід містить номер 26. Це означає, що значення на першому виході буде відповідати значенню з 26-го входу. Оскільки останній табличний вхід – 7, це означає, що значення на 32-му виході буде відповідати значенню сьомого інформаційного входу, і так далі.

Прямі *P*-блоки є оберненими. Це означає, що можна використовувати прямий *P*-бітовий блок для шифрування, а потім для розшифрування. Таблиці перестановки, проте, повинні бути оберненими у відношенні одна до одної.

Приклад 3.6. Необхідно скласти таблицю перестановки для прямого *P*-блоку 8×8, яка переміщує два середніх біта (біти 4 і 5) у вхідному слові до двох крайніх біт (біти 1 і 8) вихідного слова. Відносні позиції інших бітів не змінюються.

Рішення. Потрібно створити прямий *P*-блок з таблицею [4, 1, 2, 3, 6, 7, 8, 5]. Відносні позиції біт 1, 2, 3, 6, 7 і 8 не змінюються, але перший вихід пов’язаний з четвертим інформаційним входом, восьмий вихід – з п’ятим інформаційним входом. Перестановки для такого прямого *P*-блоку наведено в табл. 3.2.

Таблиця 3.2

Таблиця перестановки для прямого *P*-блоку для прикладу 3.5

	Номера бітів							
Вхід	04	01	02	03	06	07	08	05
Вихід	01	02	03	04	05	06	07	08

P-блоки стиснення

P-блок стиснення – це *P*-блок з *n* входами і *m* виходами, де $m < n$. Деякі з інформаційних входів блоковані і не пов'язані з виходом (див. рис. 3.4). *P*-блоки стиснення, що використовуються в сучасних блокових шифрах, зазвичай є безключовими з таблицею перестановки, яка визначає правила перестановки біт. Необхідно враховувати, що таблиця перестановок для *P*-блоку стиснення має *n* табличних входів – від 1 до *n*, але в змісті кожного табличного виходу деякі з них можуть бути відсутні (ті інформаційні входи, які блоковані). Приклад таблиці перестановок для *P*-блоку стиснення 32×24, у яких входи 7, 8, 9, 16, 23, 24 і 25 блоковані, показано у табл. 3.3.

Таблиця 3.3

Приклад таблиці перестановки для *P*-блоку стиснення 32×24

	Номера бітів											
Вхід	01	02	03	21	22	26	27	28	29	13	14	17
Вихід	01	02	03	04	05	06	07	08	09	10	11	12
Вхід	18	19	20	04	05	06	10	11	12	30	31	32
Вихід	13	14	15	16	17	18	19	20	21	22	23	24

P-блоки стиснення використовуються, коли потрібно переставити біти і водночас зменшити число бітів для наступного ступеня.

P-блоки стиснення є необерненими. У *P*-блоках стиснення окремі входи можуть бути відкинуті у процесі шифрування і при цьому алгоритм розшифрування не має ключа, щоб відновити відкинуті біти. Рис. 3.6 демонструє випадок використання *P*-блока стиснення для шифрування і розшифрування.

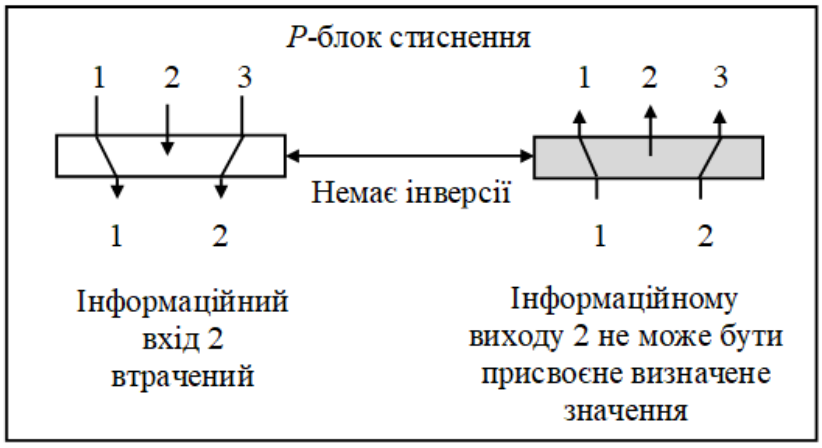


Рис. 3.6. Р-блок стиснення як необернений компонент

P-блоки розширення

P-блок розширення – *P*-блок з *n* входами і *m* виходами, де $m > n$. Деякі з входів пов'язані більше ніж з одним виходом (див. рис. 3.4). *P*-блоки розширення, використовувані в сучасних блокових шифрах, зазвичай без ключа. Правила перестановки біт вказуються в таблиці. Таблиця перестановки для *P*-блоку розширення має *m* табличних виходів, але *m–n* входів (входи, які пов'язані більше ніж з одним інформаційним виходом). Приклад таблиці перестановок для *P*-блоку розширення 12×16 показано у табл. 3.4. Зверніть увагу, що кожний з 1, 3, 9 і 12 входів з'єднаний з двома виходами.

Таблиця 3.4

Приклад таблиці перестановки для Р-блоку розширення 12×16

	Номера бітів															
Вхід	01	09	10	11	12	01	02	03	03	04	05	06	07	08	09	12
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

P-блоки розширення використовуються, коли потрібно переставити біти і водночас збільшити число бітів для наступного каскаду шифрування.

P-блоки розширення також є необернеными. У *P*-блоці розширення деякі входи в процесі шифрування можуть бути відображені більш ніж в один вихід, але при цьому алгоритм розшифрування не має ключа і не може

визначити, які з кількох входів відображені в даному виході. Рис. 3.7 демонструє випадок використання *P*-блоку розширення для шифрування і розшифрування.

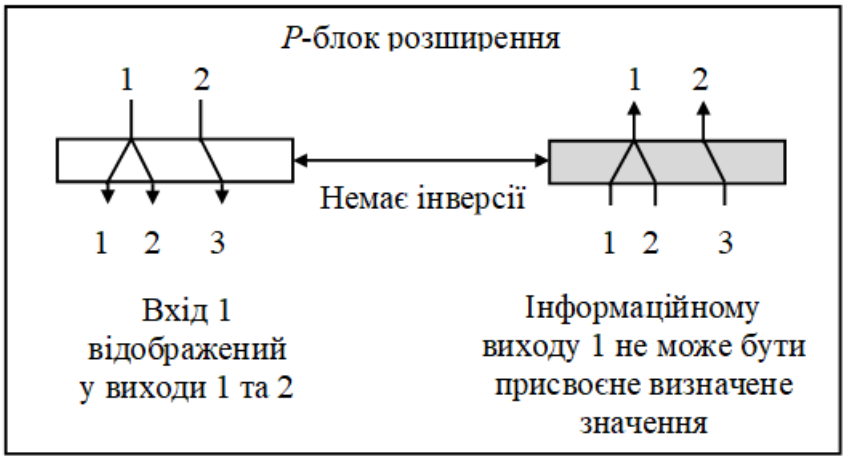


Рис. 3.7. *P*-блок розширення як необернений компонент

S-блоки підстановки

S-блок підстановки (заміни) можна представити собі як мініатюрний шифр підстановки. Цей блок може мати різне число входів і виходів. Іншими словами, вхід до *S*-блоку підстановки може бути *n*-бітових словом, а вихід може бути *m* розрядних словом, де *m* і *n* – не обов'язково однакові числа. Хоча *S*-блок підстановки може бути ключовим або без ключа, сучасні блокові шифри зазвичай використовують *S*-блоки підстановки без ключів, де відображення від інформаційних входів до інформаційних виходів заздалегідь визначене.

В *S*-блоці підстановки з *n* входами і *m* виходами позначимо входи як $x_1, x_2, \ldots, x_n$ і виходи як $y_1, y_2, \ldots, y_m$. Співвідношення між входами і виходами можуть бути представлені системою рівнянь:

$$y_1 = f_1(x_1, x_2, \ldots, x_n);$$
$$y_2 = f_2(x_1, x_2, \ldots, x_n);$$
$$\ldots$$
$$y_m = f_m(x_1, x_2, \ldots, x_n).$$

У лінійному *S*-блоці підстановки вищезгадані співвідношення можуть бути виражені як:

$$y_1 = a_{1,1} \cdot x_1 \oplus a_{1,2} \cdot x_2 \oplus \ldots \oplus a_{1,n} \cdot x_n;$$
$$y_2 = a_{2,1} \cdot x_1 \oplus a_{2,2} \cdot x_2 \oplus \ldots \oplus a_{2,n} \cdot x_n;$$
$$\ldots$$
$$y_m = a_{m,1} \cdot x_1 \oplus a_{m,2} \cdot x_2 \oplus \ldots \oplus a_{m,n} \cdot x_n.$$

У нелінійному *S*-блоці підстановки також можна завжди визначити для кожного виходу зазначені вище співвідношення.

Приклад 3.7. Нехай у *S*-блоці з трьома входами і двома виходами маємо

$$y_1 = x_1 \oplus x_2 \oplus x_3;$$
$$y_2 = x_1.$$

S-блок підстановки є лінійним, тому що

$$a_{1,1} \cdot = a_{1,2} \cdot = a_{1,3} \cdot = a_{2,1} = 1 \quad \text{та} \quad a_{2,2} \cdot = a_{2,3} \cdot = 0.$$

Ці співвідношення можуть бути представлені матрицями, як показано нижче:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Приклад 3.8. У *S*-блоці з трьома входами і двома виходами маємо

$$y_1 = (x_1)^3 + x_2;$$
$$y_2 = (x_1)^2 + x_1 \cdot x_2 + x_3,$$

де множення і додавання проводиться в полі Галуа *GF*(2). *S*-блок підстановки нелінійний, тому що немає лінійних співвідношень між входами і виходами.

Приклад 3.9. Наступна таблиця (див. рис. 3.8) визначає залежність між входами/виходами для *S*-блоку підстановки розміру 3×2. Крайній лівий біт входу визначає рядок; два правих біта входу визначають стовпець. Два біта виходу – це значення на перетині вибраного рядка і стовпця.

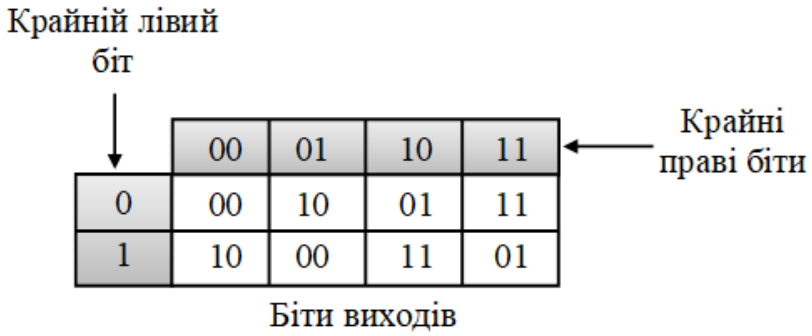


Рис.3.8. Таблиця *S*-блока підстановки для прикладу 3.9

S-блоки підстановки – це шифри підстановки, в яких відносини між входом і виходом визначені таблицею або математичним співвідношенням. *S*-блок може бути або не бути оборотним. В оборотному *S*-блоці підстановки число вхідних бітів повинно дорівнювати числу біт виходу.

Приклад 3.10. Рис. 3.9 показує приклад оберненого *S*-блоку підстановки. Одна з таблиць використовується в алгоритмі шифрування; інша таблиця – в алгоритмі розшифрування.

У кожній таблиці крайній лівий біт входу визначає рядок; наступні два біта визначають стовпець. Вихід – це значення на перетині рядка і стовпця таблиці. Наприклад, якщо вхід до лівого блоку – 001, то вихід – 101. Вхід 101 в правій таблиці дає вихід 001. Це показує, що ці дві таблиці дозволяють отримати обернений результат у відношенні один до одного.

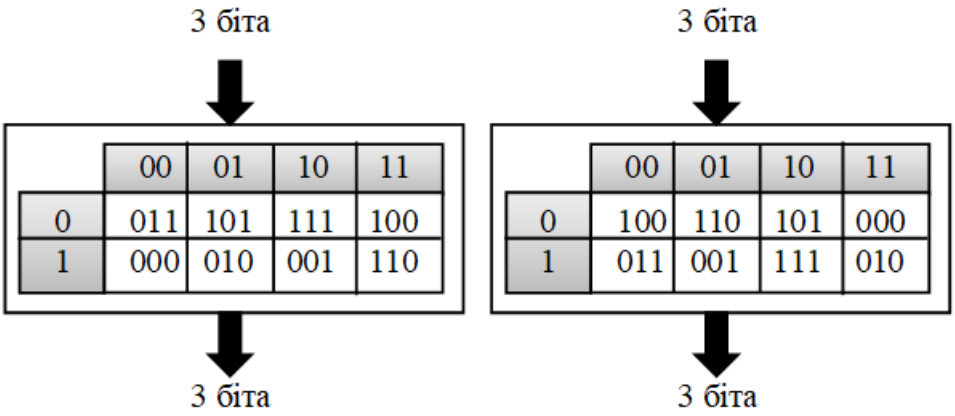


Рис. 3.9. Таблиці *S*-блока підстановки для прикладу 3.10

Виключне або (xor)

Важливим компонентом у більшості блоків шифрів є операція *виключне або* (*xor*). Операції додавання і віднімання в полі Галуа $GF(2^n)$ виконуються за допомогою тієї самої операції, яку називають виключним або (*xor*).

П'ять властивостей операції *xor* у полі Галуа $GF(2^n)$ роблять цю операцію дуже зручною для використання в блоковому шифрі:

1. *Замкнутість*. Ця властивість гарантує, що в результаті цієї операції два *n*-бітових слова дають інше *n*-бітове слово.

2. *Асоціативність*. Ця властивість дозволяє використовувати більше ніж одне *xor*, які можна обчислювати в будь-якому порядку:

$$x \oplus (y \oplus z) \leftrightarrow (x \oplus y) \oplus z.$$

3. *Комутативність*. Ця властивість дозволяє міняти місцями оператори (вхідну інформацію), не змінюючи результат (вихідну інформацію):

$$x \oplus y \leftrightarrow y \oplus x.$$

4 *Існування нульового елемента*. Нульовий елемент для операції *xor* – це слово, яке складається з усіх нулів. Це означає, що існує слово з нейтральними елементами, яке під час операції не змінює слово:

$$x \oplus (00\ldots00) = x.$$

Ця властивість використовується у шифрі Фейстеля, який розглянемо далі.

5. *Існування інверсії*. У полі Галуа $GF(2^n)$ кожне слово є адитивною інверсією самого себе. Це означає, що проведення операції *xor* слова із самим собою призводить до нульового елементу:

$$x \oplus x = (00\ldots00).$$

Ця властивість також використовується у шифрі Фейстеля.

6. *Доповнення*. Операція доповнення – одномісна операція (один інформаційний вхід і один інформаційний вихід), яка інвертує кожний біт у

слові. Нульовий біт змінюється на одиничний біт, а одиничний біт – на нульовий.

Під час шифруванні викликають інтерес операції з доповненням відносно операції *xor*. Якщо $\bar{x}$ – доповнення x, то вірні наступні два співвідношення:

$$x \oplus \bar{x} = (11 \ldots 11) \quad \text{та} \quad x \oplus (11 \ldots 11) = \bar{x}.$$

Скористаємося цією властивістю далі, коли буде обговорюватися безпека деяких шифрів.

7. *Інверсія*. Інверсія компонента в шифрі має сенс, якщо компонент представляє одномісну операцію (один вхід і один вихід).

Операція *xor* – бінарна операція. Інверсія операції *xor* може мати сенс тільки тоді, коли один з входів зафіксований (той самий під час шифрування і розшифрування). Наприклад, якщо один з входів – ключ, який зазвичай є однаковим під час шифрування і розшифрування, тоді операція *xor* є оберненою, як показано на рис. 3.10.

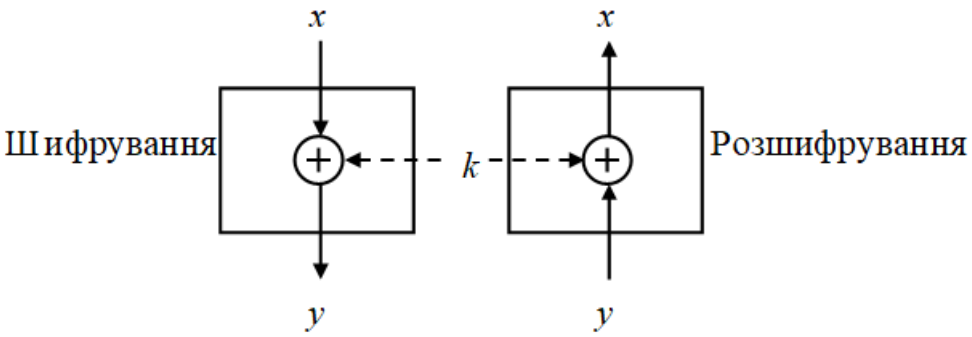


Рис. 3.10. Оберненість операції *xor*

На рис. 3.10 властивість адитивної інверсії означає, що

$$y = x \oplus k \quad \text{та} \quad x = k \oplus y.$$

Скористаємося цією властивістю, коли будемо обговорювати структуру блокових шифрів.

Циклічний зсув

Наступний компонент, який застосовується у деяких сучасних блокових шифрах, – *операція циклічного зсуву*. Зсув може бути вліво або вправо. Циклічна операція лівого зсуву зсуває кожний біт у n-бітовому слові на k позицій вліво; крайні ліві k-біти видаляються зліва і стають крайніми правими бітами. Циклічна операція правого зсуву зсуває кожний біт у n-бітовому слові на k позицій вправо; крайні праві k-біти видаляються і стають крайніми лівими бітами. Рис. 3.11 показує як і ліві, так і праві операції у випадку, коли $n = 8$ і $k = 3$.

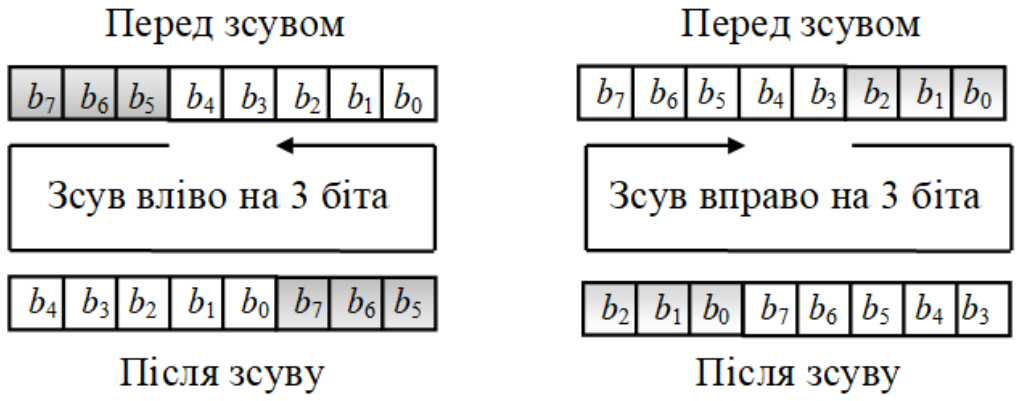


Рис. 3.11. Приклад циклічного зсуву 8 бітового слова на 3 біта вліво чи вправо

Циклічна операція зсуву змішує біти в слові і допомагає приховати зразки в початковому слові. Хоча число позицій, на яке будуть зсунуті біти, може використовуватися як ключ, циклічна операція зсуву зазвичай без ключа; значення k встановлюється і задається заздалегідь.

Циклічна операція лівого зсуву – інверсія операції правого зсуву, тобто є оберненою. Якщо одна з них використовується для шифрування, то інша може застосовуватися для розшифрування.

Операція циклічного зсуву має дві властивості.

Перша – це зміщення за модулем n. Іншими словами, якщо $k = 0$ або $k = n$, то ніякого зсуву не відбувається. Якщо k є більшим, ніж n, то вхідна інформація зсувається на $k \bmod n$ біт.

Друга властивість, операція циклічного зсуву над з'єднанням операцій – є група. Це означає, що якщо зміщення робиться неодноразово, то те саме значення може з'явитися кілька разів.

Заміна

Операція заміни – спеціальний випадок операції циклічного зсуву, де $k = n/2$. Це означає, що ця операція можлива, тільки якщо n – парне число. Оскільки зсув вліво на $n/2$ – те саме, що і зсув на $n/2$ вправо, то ця операція є оберненою. Операція заміни для шифрування може бути повністю розкрита операцією заміни під час розшифрування. Рис. 3.12 ілюструє операцію заміни для слова з *8* бітів.

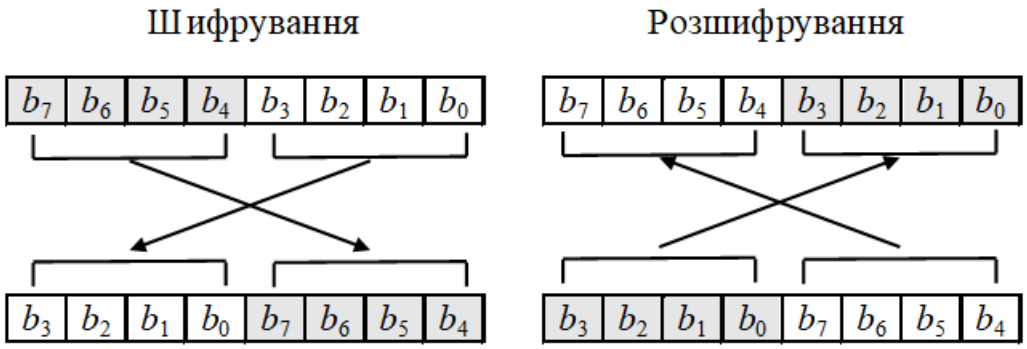


Рис. 3.12. Операція заміни для 8 бітового слова

Розбиття і об'єднання

Наступні операції, що застосовуються в деяких блокових шифрах, – *розбиття і об'єднання*. Розбиття зазвичай розділяє n-бітове слово по середині, створюючи два слова рівної довжини. Об'єднання зв'язує два слова рівної довжини, щоб створити n-бітове слово. Ці дві операції інверсні одна одній і можуть використовуватися як пара, щоб врівноважити одна одну. Якщо одна використовується для шифрування, то інша – для розшифрування. Рис. 3.13 показує ці дві операції для випадку коли $n = 8$.

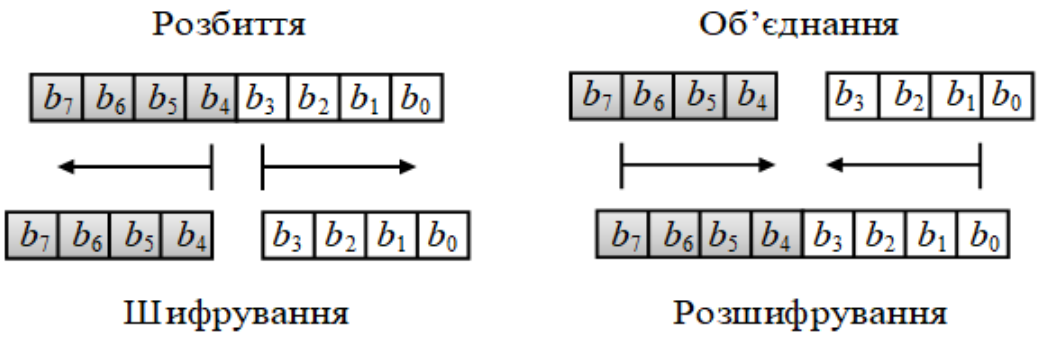


Рис. 3.13. Операції розбиття і об’єднання для 8-бітового слова

3.2. Складені шифри

Шеннон ввів поняття складених шифрів. Складений шифр – комплекс, який об’єднує підстановку, перестановку і інші компоненти, розглянуті вище.

Розсіювання і перемішування

Ідея Шеннона в поданні складеного шифру повинна була дати можливість блоковим шифрам мати дві важливі властивості: розсіювання та перемішування.

Розсіювання має приховати відношення між зашифрованими і вихідними даними. Це дезорієнтує зловмисника, який використовує статистику зашифрованих даних, щоб знайти вихідні дані. Під розсіюванням розуміється, що кожний біт (символ) у зашифрованих даних залежить від одного або усіх бітів (символів) у вихідних даних. Іншими словами, якщо один єдиний біт у вихідних даних змінений, то декілька або усі біти в зашифрованих даних будуть також змінені.

Ідея щодо перемішування полягає в тому, що воно повинно приховати залежність між зашифрованими даними і ключем. Це також дезорієнтує зловмисника, який прагне використовувати зашифровані дані, щоб знайти ключ. Іншими словами, якщо один єдиний біт у ключі буде змінений, то усі біти в зашифрованих даних будуть також змінені.

Раунди

Розсіювання і перемішування можна досягнути використанням повторення складених шифрів, де кожна ітерація – комбінація *S*-блоків, *P*-блоків та інших компонентів. Кожна ітерація називається раундом. Блоковий шифр використовує ключовий список, або генератор ключів, який створює різні ключі для кожного раунду з ключа шифру. В *N*-раундовому шифрі, щоб створити зашифровані дані, вихідні дані зашифровуються *N* разів; відповідно, зашифровані дані розшифровуються *N* разів. Дані, створені на проміжних рівнях (між двома раундами), називаються середніми даними. Рис. 3.14. показує простий складений шифр з двома раундами.

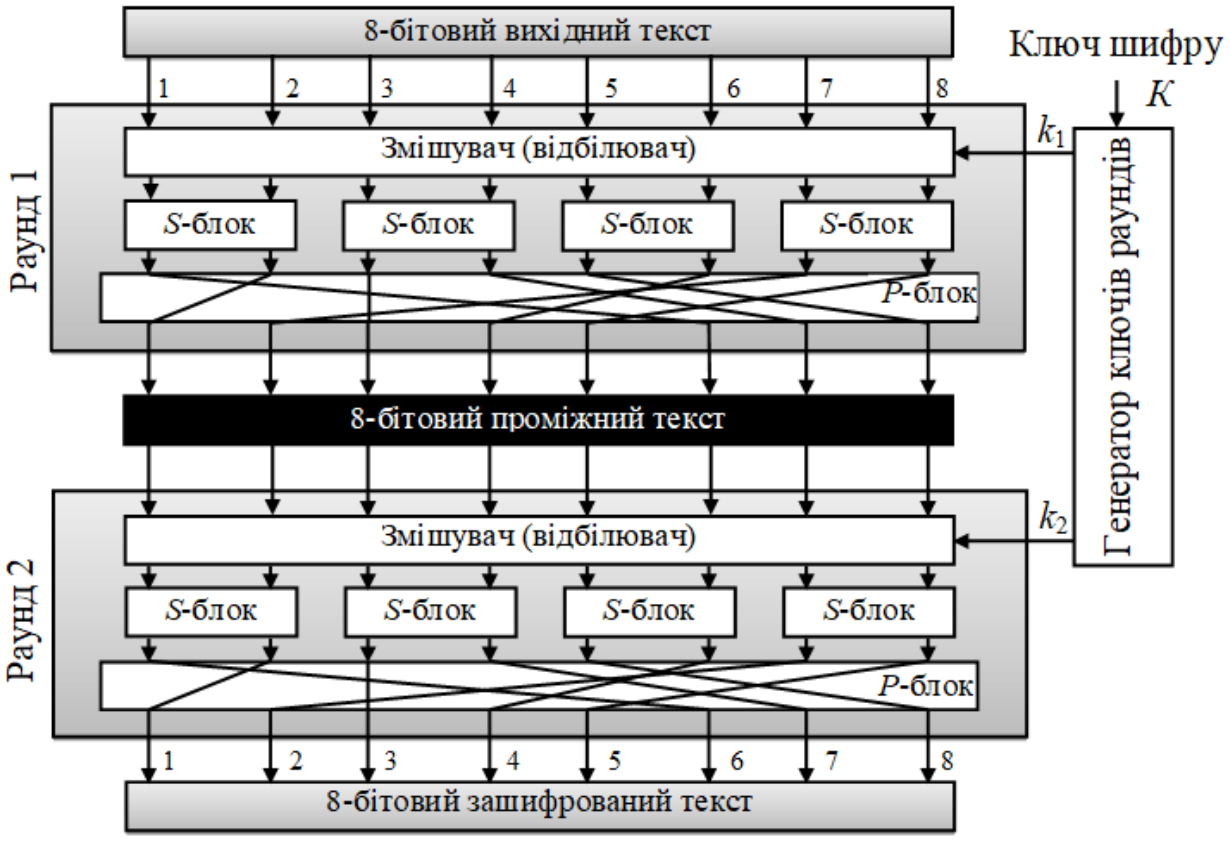


Рис. 3.14. Складений шифр, що складається з двох раундів

На практиці складові шифри мають більше, ніж два раунди. На рис. 3.14 у кожному раунді проводяться три перетворення:

1. 8-бітові дані змішуються з 8-бітовим ключем раунду, щоб зробити біти даних рівноймовірними (приховати біти, використовуючи ключ) – “відбілити” дані. Це зазвичай робиться за допомогою операції *xor*.

2. Дані з виходів “відбілювача” розбиваються на чотири групи по 2 біти і подаються у чотири *S*-блоки заміни. Значення бітів змінюються відповідно до побудови *S*-блоків заміни у цьому перетворенні.

3. З виходів *S*-блоків заміни дані надходять у *P*-блок перестановки, при цьому біти повинні бути переставлені так, щоб у наступному раунді результат з виходу кожного блоку надійшов на різні входи.

На рис. 3.15 показано, як зміна одного єдиного біта у вихідних даних викликає зміну багатьох бітів у зашифрованих даних.

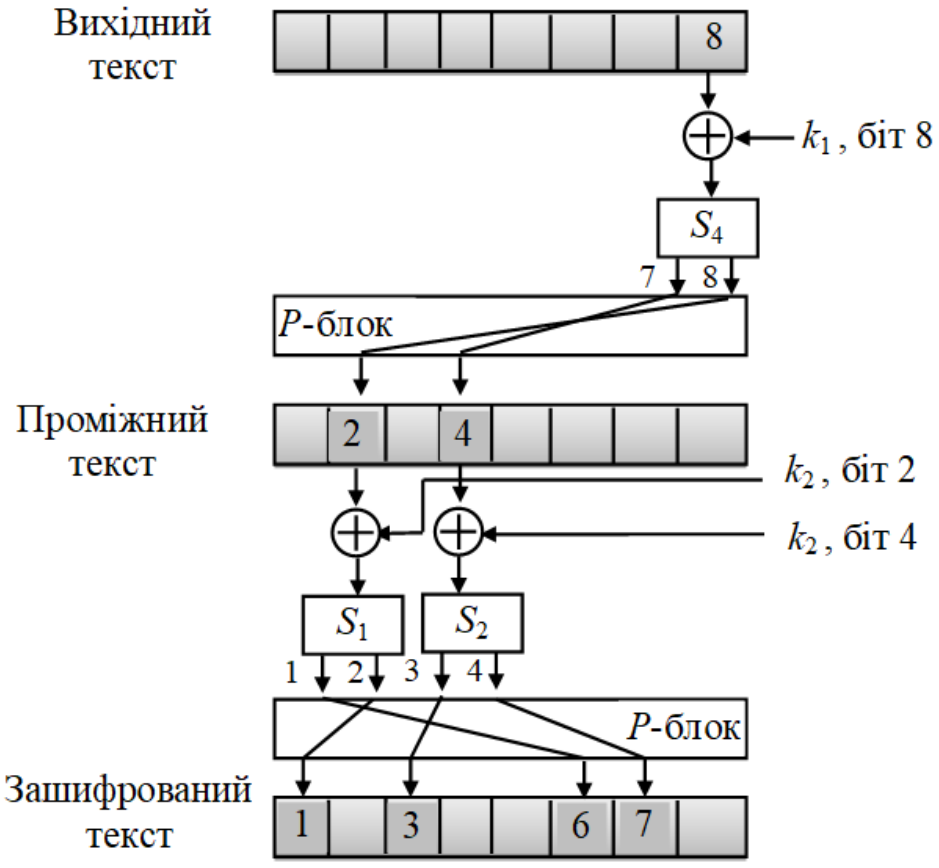


Рис. 3.15. Розсіювання і перемішування у блоковому шифрі

У першому раунді восьмий вихідний біт, після проведення операції *xor* з відповідним бітом ключа раунду k_1, змінює два біти (7 та 8) через *S*-блок 4.

Біт 7 після перестановки стає бітом 4, а біт 8 стає бітом 2. Таким чином, після першого раунду восьми вихідний біт змінює біти 2 і 4. У другому раунді біт 2 після проведення операції *xor* з відповідним бітом ключа k_2 змінює два біта (1 та 2) через *S*-блок 1. Біт 1 після перестановки стає бітом 6, а біт 2 стає бітом 1. Біт 4 після проведення операції *xor* з відповідним бітом ключа k_2 змінює біти 3 та 4 через *S*-блок 2. Біт 3 залишається 3 бітом, а біт 4 після перестановки стає бітом 7. Таким чином, після другого раунду восьмий вихідний біт змінює біти 1, 3 , 6 і 7.

Рис. 3.15 також доводить, що властивість перемішування може бути отримана за допомогою складеного шифру. Чотири біта зашифрованих даних (біти 1, 3, 6 і 7) перетворені за допомогою трьох бітів в ключах раундів (8 біта у k_1 і 2 та 4 бітів у k_2).

Щоб поліпшити розсіювання і перемішування, практичні шифри використовують великі блоки даних, більше *S*-блоків заміни і більше раундів. Очевидно, що деяке збільшення числа раундів при використанні великого числа *S*-блоків заміни може створити кращий шифр, в якому зашифровані дані виглядають усе більш як випадкове *n*-бітове слово. Таким чином, взаємозв'язки між зашифрованими і вихідними даними будуть повністю приховані (розсіяні). Збільшення числа раундів збільшує число ключів раундів, що краще приховує взаємозалежність між зашифрованими даними і ключем.

Два класи складених шифрів

Сучасні блокові шифри – усі складені, але вони розділені на два класи. Шифри першого класу використовують і обернені, і необернені компоненти. Ці шифри згадуються зазвичай як шифри Фейстеля. Шифри другого класу застосовують тільки обернені компоненти. Слід звернути увагу, що шифри цього класу це шифри не-Фейстеля (через відсутність іншої назви).

Фейстель розробив дуже інтелектуальний і цікавий шифр, який використовувався протягом багатьох десятиліть. Шифр Фейстеля має три типи компонентів: самообернений, обернений і необернений.

Шифр Фейстеля містить необернені компоненти і використовує один і той модуль в алгоритмах шифрування і розшифрування. Питання в тому, як алгоритми шифрування і розшифрування дозволяють інвертувати відкриті і закриті дані один до одного, якщо кожен містить необернені компоненти. Фейстель показав, що вони можуть бути збалансовані [4, 29, 34, 36].

Щоб краще зрозуміти шифр Фейстеля, подивимося, як можна використовувати той самий необернений компонент в алгоритмах шифрування і розшифрування. Ефекти необерненого компонента в алгоритмі шифрування можуть бути скасовані в алгоритмі розшифрування, якщо використовувати операцію *xor*, як показано на рис. 3.16.

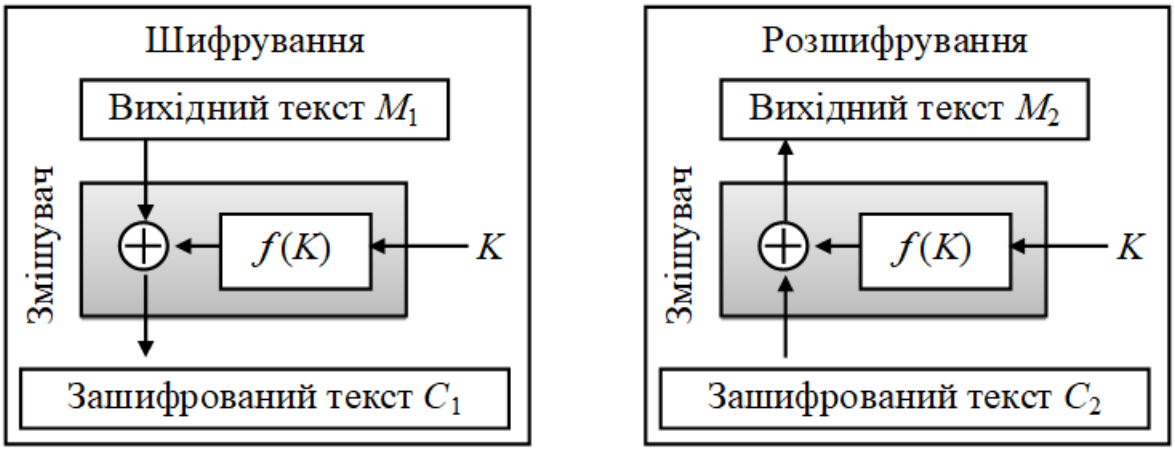


Рис. 3.16. Використання необерненого компоненту в алгоритмах шифрування і розшифрування Фейстеля

Під час шифрування ключ надходить на вхід необерненої функції $f(K)$, яка є одним з доданків оператора *xor* з вихідними даними. В результаті отримаємо зашифровані дані. Будемо називати комбінацію функції $f(K)$ і операції *xor* – змішувачем. Змішувач відіграє важливу роль у пізніших варіантах шифру Фейстеля.

Оскільки ключ під час шифрування і розшифрування такий самий, можна довести, що два алгоритми інверсні один одному. Іншими словами, якщо $C_2 = C_1$ (в зашифрованих даних протягом передачі не відбулося змін), то $M_2 = M_1$.

Шифрування:

$$C_1 = M_1 \oplus f(K).$$

Розшифрування:

$$M_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = M_1 \oplus f(K) \oplus f(K) = M_1 \oplus (00\ldots0) = M_1.$$

Зверніть увагу, що для обчислення використовувалися дві властивості операції *xor* (існування інверсії і існування нульового елементу).

Показані вище рівняння доводять, що хоча змішувач має необернений компонент, сам змішувач є самооберненим.

Приклад 3.11. Нехай є вихідні і зашифровані дані, кожні завдовжки 4 біта, і ключ завдовжки 3 біта. Припустимо, що функція $f(K)$ використовує перший і третій біти ключа, інтерпретує біти як десятковий номер, знаходить квадрат цього числа і інтерпретує результат як 4-бітову двійкову послідовність. Необхідно показати результати шифрування і розшифрування даних, якщо початкові вихідні дані – 0111, і ключ – 101.

Рішення. Функція $f(K)$ використовує перший і третій біти ключа. В результаті виходить 11 у двійковому вигляді або 3 у десятковому відображенні. Результат піднесення у другу ступінь (квадрат) – 9, у двійковому відображенні 1001.

Шифрування: $C = M \oplus f(K) = 0111 \oplus 1001 = 1110.$

Розшифрування: $M = C \oplus f(K) = 1110 \oplus 1001 = 0111.$

Як видно, результат розшифрування співпаде з вихідними даними M.

Функція $f(101) = 1001$ є необернеою, але операція *xor* дозволяє використовувати функцію $f(K)$ у алгоритмах шифрування й розшифрування. Іншими словами, функція $f(K)$ є необернеою, але змішувач, в який входить ця функція, буде самооберненим.

Щоб наблизитися до шифру Фейстеля, необхідно удосконалити шифр, представлений на рис. 3.16. Відомо, що для цього треба застосувати вхід до необерненої функції, але при цьому необхідно використовувати не лише ключ. Задіємо також вхід до функції $f(K)$, щоб застосувати її для шифрування частини вихідних даних і розшифрування частини зашифрованих даних. Ключ може використовуватися як другий вхід до функції. Завдяки цьому способу функція $f(K)$ стає складним елементом з деякими неключовими і ключовими елементами. Щоб досягти мети, розділимо вихідні та зашифровані дані на два блоки однакової довжини – лівий (L) і правий (R). Правий блок вводиться у функцію $f(K)$ і набуває вигляду $f(R_1, K)$, а лівий блок складається за допомогою операції *xor* з виходом функції $f(R_1, K)$. Необхідно запам'ятати, що входи до функції $f(R_1, K)$ повинні точно співпадати під час шифрування і розшифрування. Це означає, що права секція R_1 вихідних даних до шифрування і права секція R_1 зашифрованих даних після розшифрування будуть співпадати. Іншими словами, секція R_1 повинна увійти до шифрування і вийти з розшифрування незміненою. Рис. 3.17 ілюструє цю ідею.

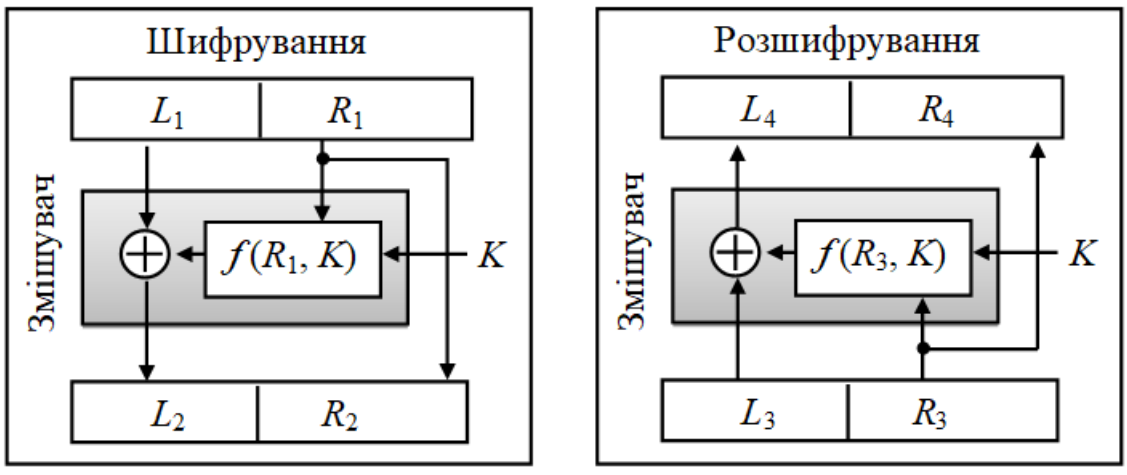


Рис. 3.17. Удосконалення попередньої схеми Фейстеля

Алгоритми шифрування і розшифрування інверсні один одному. Припустимо, що $L_3 = L_2$ і $R_3 = R_2$ (у зашифрованих даних протягом передачі

не відбулося змін). Тоді

$$R_4 = R_3 = R_2 = R_1,$$

$$L_4 = L_3 \oplus f(R_3, K) = L_2 \oplus f(R_2, K) = L_1 \oplus f(R_1, K) \oplus f(R_1, K) = L_1.$$

Вихідні дані, що використовувались в алгоритмі шифрування, – це дані, правильно відновлені алгоритмом розшифрування.

Попереднє удосконалення має один недолік: права половина вихідних даних ніколи не змінюється. Зловмисник може одразу знайти праву половину вихідних даних, розбиваючи на частини зашифровані дані і розпаковуючи його праву половину. Проект потребує подальших кроків удосконалення: по-перше, необхідно збільшити число раундів; по-друге, необхідно додати новий елемент до кожного раунду – пристрій заміни. Ефект пристрою заміни в раунді шифрування компенсується ефектом пристрою заміни в раунді розшифрування. Однак це дозволяє змінювати ліві і праві половини в кожному раунді. Рис. 3.18 ілюструє новий варіант шифру Фейстеля з двома раундами. Зверніть увагу, що є два ключа раундів: k_1 і k_2. Ключі використовуються у зворотному порядку під час шифрування і розшифрування.

Оскільки два змішувача і пристрої заміни інверсні один одному, очевидно, що шифрування і розшифрування також інверсні один одному. Можна довести цей факт, використовуючи відношення між лівими і правими секціями в кожному шифрі. Іншими словами, якщо $L_6 = L_1$ і $R_6 = R_1$, припустимо, що $L_4 = L_3$ і $R_4 = R_3$ (зашифровані дані не змінилися при передачі). Спочатку доведемо це для проміжних даних:

$$L_5 = R_4 \oplus f(L_4, k_2) = R_3 \oplus f(L_3, k_2) = L_2 \oplus f(R_2, k_2) \oplus f(R_2, k_2) = L_2,$$

$$R_5 = L_4 = L_3 = R_2.$$

Тоді просто довести рівність для двох блоків вихідних даних:

$$L_6 = R_5 \oplus f(L_5, k_1) = R_2 \oplus f(L_2, k_1) = L_1 \oplus f(R_1, k_1) \oplus f(R_1, k_1) = L_1,$$

$$R_6 = L_5 = L_2 = R_1.$$

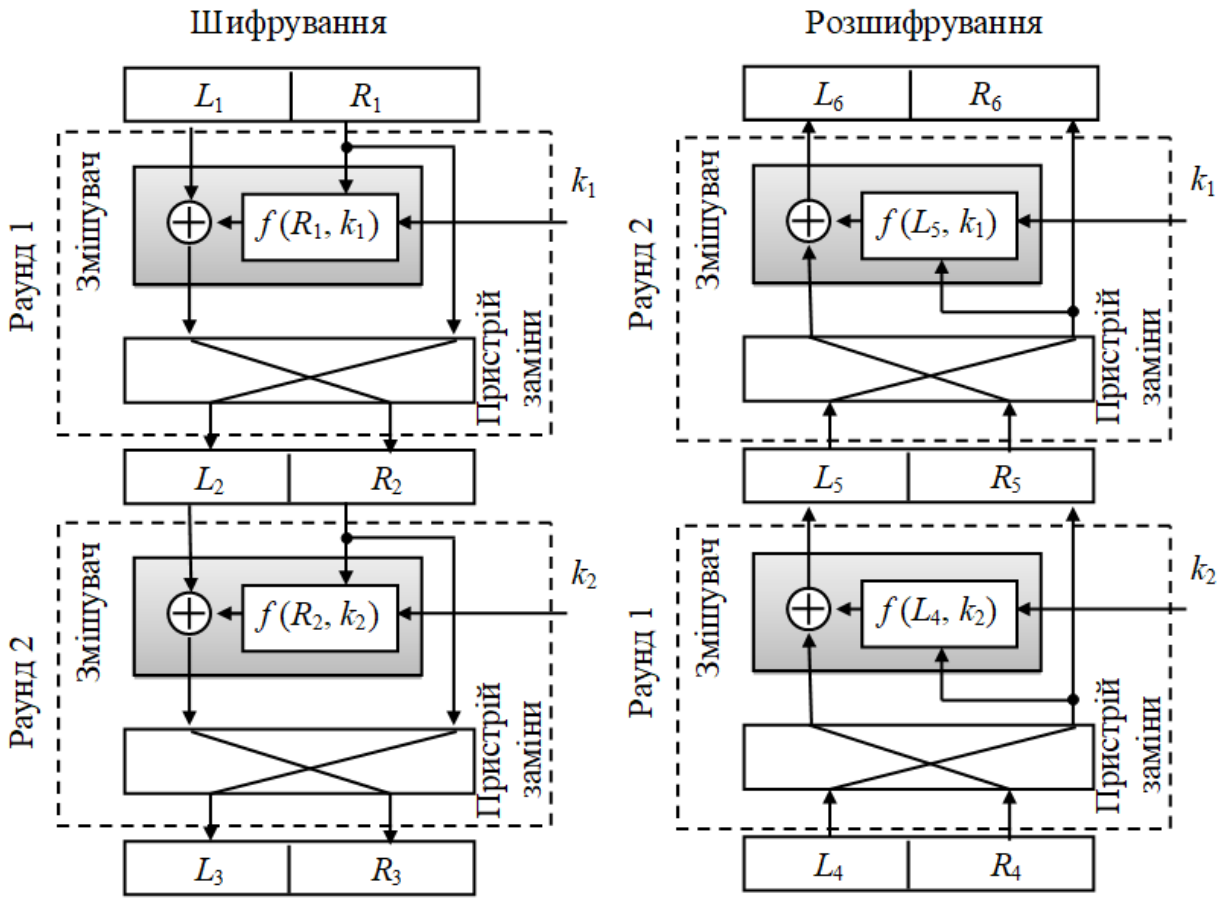


Рис. 3.18. Кінцевий варіант шифру Фейстеля с двома раундами

Шифри не-Фейстеля використовують тільки обернені компоненти. Наприклад, *S*-блоки заміни повинні мати однакове число входів і виходів, щоб бути сумісними (оберненими). Не дозволяється у таких шифрах ніяке стиснення або розширення *P*-блоків, тому що вони стануть необернеными. У шифрах не-Фейстеля немає потреби ділити вихідний текст на дві половини.

Рис. 3.14 можна розглядати як графічну ілюстрацію принципу шифру не-Фейстеля, тому що єдині компоненти в кожному раунді – самообернені операції *xor*, *S*-блоки 2×2, які можуть бути побудовані так, щоб бути оберненими, і прямі *P*-блоки, які обернені, якщо використана відповідна таблиця перестановки. Оскільки кожен компонент є оберненим, то можна показати, що і кожний раунд є оберненими. Необхідно тільки застосовувати ключі раундів у зворотному порядку. Шифрування використовує ключі раундів k_1 і k_2. Алгоритм розшифрування повинен користуватися ключами раундів k_2 і k_1.

3.3. Атаки на блокові шифри

Атаки традиційних шифрів можуть також використовуватися для сучасних блокових шифрів, але теперішні блокові шифри успішно протистоять більшості атак. Наприклад, атака “грубої сили” ключа, як правило, нездійсненна, тому що ключі зазвичай мають дуже велику довжину. Однак були винайдені деякі нові види атак на блокові шифри, які засновані на структурі сучасних блокових шифрів. Ці атаки використовують методи диференціального і лінійного криптографічного аналізу.

3.3.1. Диференціальний криптографічний аналіз

Ідею щодо диференціального криптографічного аналізу запропонували Елі Біхам і Аді Шамір. Це – атака на обраний вихідний текст [4, 18, 27, 36]. Зловмисник може яким-небудь чином отримати доступ до комп’ютера відправника і заволодіти вибірково частиною вихідних даних і відповідним їм зашифрованих даних. Мета полягає в тому, щоб знайти ключ шифру відправника.

Алгоритм аналізу. Перед тим як зловмисник зробить атаку на обраний вихідний текст, він повинен проаналізувати алгоритм шифрування, щоб зібрати деяку інформацію про залежність зашифрованих і вихідних даних. Очевидно, зловмисник не знає ключ шифру. Однак деякі шифри мають уразливі місця в структурах, які можуть дозволити зловмиснику знайти відмінності вихідних даних і відмінності зашифрованих даних, не знаючи ключ.

Приклад 3.12. Припустимо, що шифр складається тільки з однієї операції *xor*, як показано на рис. 3.19.

Не знаючи значення ключа, зловмисник може легко знайти відношення між різницями вихідних і різницями зашифрованих даних.

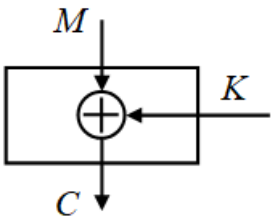


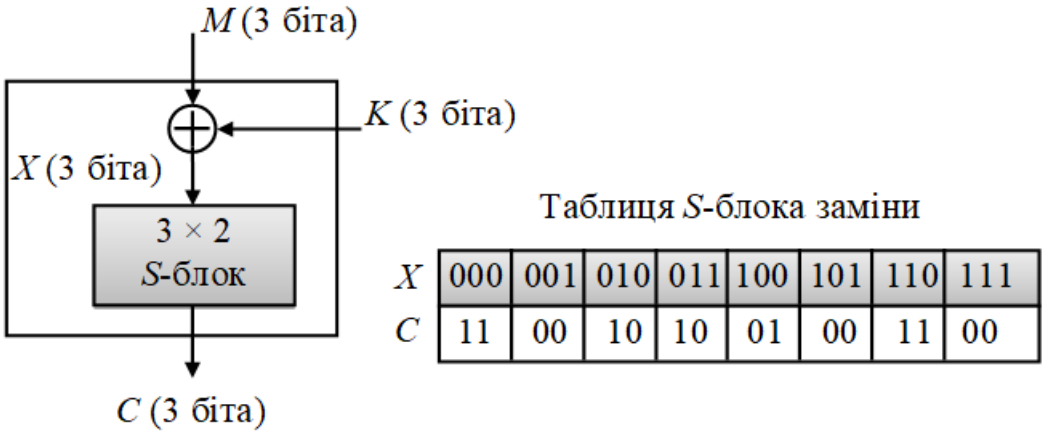
Рис. 3.19. Діаграма для прикладу 3.12

Якщо різницю вихідних даних позначити $M_1 \oplus M_2$, а різницю зашифрованих даних позначити $C_1 \oplus C_2$, то наведені наступні перетворення доводять, що $C_1 \oplus C_2 = M_1 \oplus M_2$:

$$C_1 = M_1 \oplus K; \quad C_2 = M_2 \oplus K;$$
$$C_1 \oplus C_2 = M_1 \oplus K \oplus M_2 \oplus K = M_1 \oplus M_2.$$

Однак цей приклад нереалістичний; сучасні блокові шифри не настільки прості.

Приклад 3.13. У прикладі 3.12 необхідно додати один *S*-блок заміни, як показано на рис. 3.20.



Таблиця *S*-блока заміни

X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

Рис. 3.20. Діаграма для прикладу 3.13

Хоча ефект шифрування ключем не діє, коли використовуються різниці між двома *X* і двома *M* ($X_1 \oplus X_2 = M_1 \oplus M_2$), існування *S*-блоку заміни заважає зловмиснику знайти певні відношення між різницями вихідних даних і різницями зашифрованих даних. Однак можливо встановити ймовірні

відносини. Зловмисник може скласти таблицю (див. табл. 3.5), яка показує скільки можна створити різниць зашифрованих даних для визначеної різниці вихідних даних. Зверніть увагу, що таблиця складена за інформацією, яка створена з урахуванням таблиці входу-виходу *S*-блоку заміни на рис. 3.20, тому що $X_1 \oplus X_2 = M_1 \oplus M_2$.

Таблиця 3.5

Диференціальна таблиця входів і виходів для шифру в прикладі 3.13

		$C_1 \oplus C_2$			
		00	01	10	11
$M_1 \oplus M_2$	000	8			
	001	2	2		4
	010	2	2	4	
	011		4	2	2
	100	2	2	4	
	101		4	2	2
	110	4		2	2
	111			2	6

Оскільки розмір ключа – три біта, то може бути вісім випадків для кожної різниці у введенні. Таблиця показує, що якщо вхідна різниця – $(000)_2$, то різниця виходу – завжди $(00)_2$. З іншого боку, таблиця показує, що якщо вхідна різниця – $(100)_2$, то є два випадки різниць виходу $(00)_2$, два випадки різниць виходу $(01)_2$ і чотири випадки різниць виходу $(10)_2$.

Приклад 3.14. Евристичний результат прикладу 3.13 може створити ймовірнісну інформацію для зловмисника, як показано в табл. 3.6. Значення в таблиці відповідають ймовірностям появи відповідних різниць. Різниці з нульовою ймовірністю ніколи не будуть виникати. Як буде зазначено пізніше, зловмисник тепер має в своєму розпорядженні достатню кількість інформації, щоб почати атаку. Табл. 3.6 показує, що ймовірності розподілені неоднорідно через уразливість у структурі *S*-блоку заміни. Табл. 3.6 згадується іноді як диференціальна таблиця розподілу або профайл *xor*.

Таблиця 3.6

Диференціальна таблиця входів і виходів для шифру в прикладі 3.14

	00	01	10	11
000	1	0	0	0
001	0,25	0,25	0	0,50
010	0,25	0,25	0,50	0
011	0	0,50	0,25	0,25
100	0,25	0,25	0,50	0
101	0	0,50	0,25	0,25
110	0,50	0	0,25	0,25
111	0	0	0,25	0,75

Запуск атаки на обрані вихідні дані. Після того, як евристичний аналіз одного разу зроблений, він може бути збережений для майбутнього використання, поки структура шифру не зміниться. Зловмисник може вибрати для атак вихідні дані. Диференціальна таблиця розподілу ймовірності (табл. 3.6) допоможе зловмиснику їх вибирати – використовуються ті, які мають найвищу ймовірність у таблиці.

Передбачуване значення ключа. Після запуску деяких атак з відповідно обраними вихідними даними зловмисник може знайти деяку пару “вихідні дані/зашифровані дані”, яка дозволяє йому припустити деяке значення ключа. Процес починається від C і просувається до M.

Приклад 3.15. Розглядаючи табл. 3.6, зловмисник знає, що якщо $M_1 \oplus M_2 = 001$, то $C_1 \oplus C_2 = 11$ з ймовірністю 0,5. Він пробує взяти $C_1 = 00$ і отримує $M_1 = 010$ (атака з обраним зашифрованим текстом). Він ще пробує взяти $C_2 = 11$ і отримує $M_2 = 011$ (інша атака з обраним зашифрованим текстом). Тепер зловмисник пробує повернутися до аналізу, заснованому на першій парі, M_1 і C_1 (див. рис. 3.20):

$$C_1 = 00 \rightarrow X_1 = 001 \quad \text{або} \quad X_1 = 101 \quad \text{або} \quad X_1 = 111.$$

Якщо $X_1 = 001$, то $K = X_1 \oplus M_1 = 011$.

Якщо $X_1 = 101$, то $K = X_1 \oplus M_1 = 111$.

Якщо $X_1 = 111$, то $K = X_1 \oplus M_1 = 101$.

Використовуючи пару M_2 і C_2 отримаємо

$$C_2 = 11 \rightarrow X_2 = 000 \quad \text{або} \quad X_2 = 110.$$

Якщо $X_2 = 000$, то $K = X_2 \oplus M_2 = 011$.

Якщо $X_2 = 110$, то $K = X_2 \oplus M_2 = 101$.

Два випробування показують, що $K = 011$ або $K = 101$. Хоча зловмисник не впевнений, яке з них точне значення ключа, але він знає, що крайній правий біт – 1 (загальний біт між двома значеннями). Продовжуючи атаку, можна вже враховувати, що крайній правий біт у ключі – 1. Таким чином, можна визначити і інші біти у цьому ключі.

Загальна процедура. Сучасні блокові шифри мають більшу складність, ніж та, яка обговорювалася в цьому підрозділі. Крім того, вони можуть містити різну кількість раундів. Зловмисник може використовувати наступну стратегію.

1. Оскільки кожний раунд містить ті самі операції, то зловмисник може створити таблицю диференціальних розподілів (профайл *xor*) для кожного *S*-блоку заміни і комбінувати їх, щоб створити розподіл для кожного раунду.

2. Припустимо, що кожний раунд незалежний (справедливе припущення). Зловмисник може створити таблицю розподілу для усього шифру, помноживши на відповідну ймовірність.

3. Зловмисник може тепер робити список вихідних даних для атак, заснованих на таблиці розподілів, створеної на другому кроці. Зауважимо, що таблиця, створена за другим кроком, допомагає зловмиснику вибирати меншу кількість пар “вихідні дані/зашифровані дані”.

4. Зловмисник вибирає зашифровані дані і знаходить відповідні вихідні дані. Потім він аналізує результат, щоб знайти деякі біти у ключі.

5. Зловмисник повторює дію четвертого кроку, щоб знайти більше бітів у ключі.

6. Після знаходження достатньої кількості бітів у ключі зловмисник може використовувати атаку “грубої сили”, щоб знайти увесь ключ.

Диференціальний криптографічний аналіз базується на таблиці неоднорідних диференціальних розподілів *S*-блоків заміни в блоковому шифрі.

3.3.2. Лінійний криптографічний аналіз

Лінійний криптоаналіз був представлений Міцурі Мацуї у 1993 році. Аналіз використовує атаки на відомі вихідні дані (на відміну від атак на обрані вихідні дані в диференціальному криптографічному аналізі). Повне обговорення цієї атаки базується на деяких поняттях теорії ймовірності. Щоб розглянути головну ідею цієї атаки, припустимо, що шифр складається з одного раунду, як показано на рис. 3.21, де c_0, c_1, і c_2 представляють собою три біта на виході і x_0, x_1, і x_2 – три біта на вході *S*-блоку підстановки (заміни).

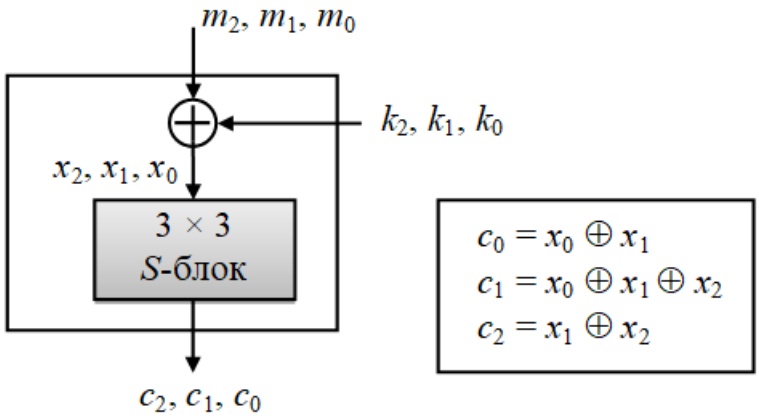


Рис. 3.21. Простий шифр з лінійним S-блоком підстановки (заміни)

S-блок підстановки (заміни) – лінійне перетворення, в якому кожний з виходів є лінійною функцією входів, яке було розглянуте вище. З цим лінійним компонентом можна створити три лінійних рівняння між вихідними даними і бітами зашифрованих даних, як показано нижче:

$$c_0 = m_0 \oplus k_0 \oplus m_1 \oplus k_1,$$

$$c_1 = m_0 \oplus k_0 \oplus m_1 \oplus k_1 \oplus m_2 \oplus k_2,$$

$$c_2 = m_1 \oplus k_1 \oplus m_2 \oplus k_2.$$

Розв’язавши систему рівнянь для трьох невідомих, отримуємо:

$$k_1 = (m_1) \oplus (c_0 \oplus c_1 \oplus c_2),$$
$$k_2 = (m_2) \oplus (c_0 \oplus c_1),$$
$$k_0 = (m_3) \oplus (c_1 \oplus c_2).$$

Це означає, що три атаки на відомі вихідні дані дають можливість знайти значення k_0, k_1, і k_2. Однак реальні блокові шифри не такі прості, як цей, вони мають більше компонентів і *S*-блоки заміни нелінійні.

Лінійна апроксимація. У деяких сучасних блокових шифрах деякі *S*-блоки заміни не повністю нелінійні; тоді вони можуть бути в імовірнісному сенсі апроксимовані деякими лінійними функціями. Взагалі, задаючи вихідні та зашифровані дані у n біт і ключ m біт, шукають деякі рівняння, що мають вигляд:

$$(k_0 \oplus k_1 \oplus \ldots \oplus k_x) = (m_0 \oplus m_1 \oplus \ldots \oplus m_y) \oplus (c_0 \oplus c_1 \oplus \ldots \oplus c_z),$$

де $1 \le x \le m$, $1 \le y \le n$ і $1 \le z \le n$. Біти у перехоплених вихідних та зашифрованих текстах можуть бути використані для того щоб знайти біти ключа.

Контрольні питання до розділу 3

1. Вкажіть відмінності між сучасними і традиційними шифрами з симетричним ключем.

2. Поясніть, чому сучасні блокові шифри спроектовані як шифри підстановки замість того, щоб застосовувати шифри транспозиції.

3. Перерахуйте компоненти сучасного блокового шифру.

4. Визначте *P*-блок і назвіть три його варіанти. Який варіант є оберненим?

5. Визначте *S*-блок і покажіть необхідну умову оберненості *S*-блоків.

6. Визначте складений шифр і назвіть два класи складених шифрів.

7. Поясніть відмінність між розсіюванням і перемішуванням.

8. Поясніть відмінність між блоковим шифром Фейстеля і не-Фейстеля.

9. Яка різниця між лінійним і диференціальним криптографічним аналізом? Який криптографічний аналіз використовує атаку на обраний вихідний текст, а який – атаку на відомі вихідні дані?

10. Повідомлення має 1500 символів для представлення, яких використовуються *ASCII* коди. Це повідомлення буде зашифровано блоковим шифром довжиною 64 біта. Знайдіть розмір доповнення і кількість зашифрованих блоків.

11. Блок транспозиції має 4 входи та 4 виходи. Який порядок групи перестановки і розмір ключової послідовності в бітах?

12. Блок підстановки має 4 входи та 4 виходи. Який порядок групи перестановки і розмір ключової послідовності в бітах?

13. Використовуючи слово $(11011010)_2$, покажіть результат циклічного зсуву вправо на 3 біти.

14. Використовуючи слово $(11011010)_2$, покажіть результат циклічного зсуву вліво на 5 біт.

15. Здійсніть перестановку бітів для прямого *P*-блоку, показаного на рис. 3.4, якщо на його вході діє послідовність: $(10011)_2$.

16. Здійсніть перестановку бітів для *P*-блоку стиснення, показаного на рис. 3.4, якщо на його вході діє послідовність: $(10011)_2$.

17. Здійсніть перестановку бітів для *P*-блоку розширення, показання на рис. 3.4, якщо на його вході діє послідовність: $(101)_2$.

18. *P*-блоки, визначені наступними таблицями:

1	4	1	2	4	3

8	1	3	2	4	5	7	6

1	7	6	2	4

До якого типу блоків належать показані *P*-блоки?

19. Крайній лівий біт *S*-блоку підстановки розміром 4×3 визначає зміщення інших трьох біт. Якщо крайній лівий біт дорівнює 0, то три інших біта переміщуються вправо на один біт. Якщо крайній лівий біт – 1, то три інших біта переміщуються вліво на один біт. Якщо вхід – 0011, то який результат буде на виході? Якщо вхід – 1110, то який результат буде на виході?

РОЗДІЛ 4
СТАНДАРТ БЛОКОВОГО СИМЕТРИЧНОГО ШИФРУВАННЯ ДАНИХ DES

4.1. Історія розробки алгоритму шифрування даних DES

Стандарт шифрування даних DES (Data Encryption Standard) – блоковий шифр із симетричними ключами, розроблений Національним Інститутом Стандартів і Технологій (National Institute of Standards and Technology – NIST) [5, 18, 24, 35, 36].

У 1973 році NIST видав запит на розробку національної криптографічної системи із симетричними ключами. Запропонована IBM модифікація проекту, названого “Люцифер” (Lucifer), була прийнята як DES. DES опубліковано у ескізному вигляді у Федеральному Реєстрі в березні 1975 року як Федеральний Стандарт Обробки Інформації (Federal Information Processing Standard – FIPS) [18, 36]. Після публікації ескіз строго критикувався з двох причин: перша – критикувалась сумнівно невелика довжина ключа (тільки 56 біт), що могло зробити шифр уразливим до атаки “грубої сили”, друга причина – критики були занепокоєні деякою прихованою побудовою внутрішньої структури DES. Вони підозрювали, що деяка частина структури (*S*-блоки заміни) може мати приховану “лазівку”, яка дозволить дешифровувати повідомлення без ключа. В подальшому, проектувальники IBM повідомили, що внутрішня структура була доопрацьована, щоб запобігти криптографічному аналізу. DES був нарешті виданий як FIPS 46 у Федеральному Реєстрі в січні 1977 року [35]. Однак FIPS оголосив DES як стандарт для використання в неофіційних додатках.

Алгоритм, покладений в основу стандарту, поширювався достатньо швидко, і вже у 1980 році був схвалений NIST США. З цього моменту DES перетворюється на стандарт не тільки за назвою, але і фактично. З’являється програмне забезпечення і спеціалізовані мікроЕОМ, призначені для

шифрування і розшифрування інформації в мережах передачі даних. DES – назва Федерального Стандарту Обробки інформації (FIPS 46-3), який описує алгоритм шифрування даних (Data Encryption Algorithm – DEA). У документах NIST криптографічна система DES називається алгоритмом шифрування даних, а міжнародна організація по стандартизації, посилаючись на шифр DES, користується абревіатурою DEA-1 [18, 36].

Цей алгоритм був світовим стандартом протягом більш ніж двадцяти років і затвердився як перший, доступний усім бажаючим, офіційний алгоритм. Тому його варто відмітити як найважливішу віху на шляху криптографії від суто військового використання до широкомасштабного застосування. DES був найбільш широко використовуваним блоковим шифром із симетричними ключами, починаючи з його публікації. Пізніше NIST запропонував новий стандарт – FIPS 46-3, який рекомендує використання потрійного DES (трикратно використаний DES) для майбутніх додатків.

Основні вимоги, які були поставлені розробникам алгоритму DES:

– алгоритм повинен забезпечувати високий рівень безпеки;

– алгоритм повинен бути повністю визначений і легко зрозумілий;

– безпека алгоритму повинна ґрунтуватися на ключі і не повинна залежати від збереження в таємниці самого алгоритму;

– алгоритм повинен бути доступний усім користувачам;

– алгоритм повинен дозволяти адаптацію до різних застосувань;

– алгоритм повинен дозволяти економічну реалізацію у вигляді електронних пристроїв;

– алгоритм повинен бути ефективним у використанні;

– алгоритм повинен забезпечувати можливість перевірки;

– алгоритм повинен бути дозволений для експорту.

Основні переваги алгоритму DES:

– використовується тільки один ключ завдовжки 64 біти (56 біт довжина ключа і 8 контрольних розрядів);

– зашифрувавши повідомлення за допомогою одного пакету програм, для розшифрування можна використовувати будь-який інший пакет програм, що відповідає стандарту DES;

– відносна простота алгоритму забезпечує високу швидкість обробки;

– достатньо висока стійкість алгоритму.

Спочатку метод, що лежить в основі стандарту DES, був розроблений фірмою IBM для своїх цілей і реалізований у вигляді системи “Люцифер”. Система “Люцифер” заснована на комбінуванні методів підстановки і перестановки та складається з послідовності блоків перестановки та підстановки, що чергуються між собою. В ній використовувався ключ завдовжки 128 бітів, що керував станами блоків перестановки і підстановки. Система “Люцифер” виявилася достатньо складною для практичної реалізації через відносно малу швидкість шифрування (2190 байт/с – програмна реалізація і 96970 байт/с – апаратна реалізація).

4.2. Принципи побудови алгоритму шифрування даних DES

Як показано на рис. 4.1, DES – криптографічна система блокового симетричного шифрування і розшифрування даних.

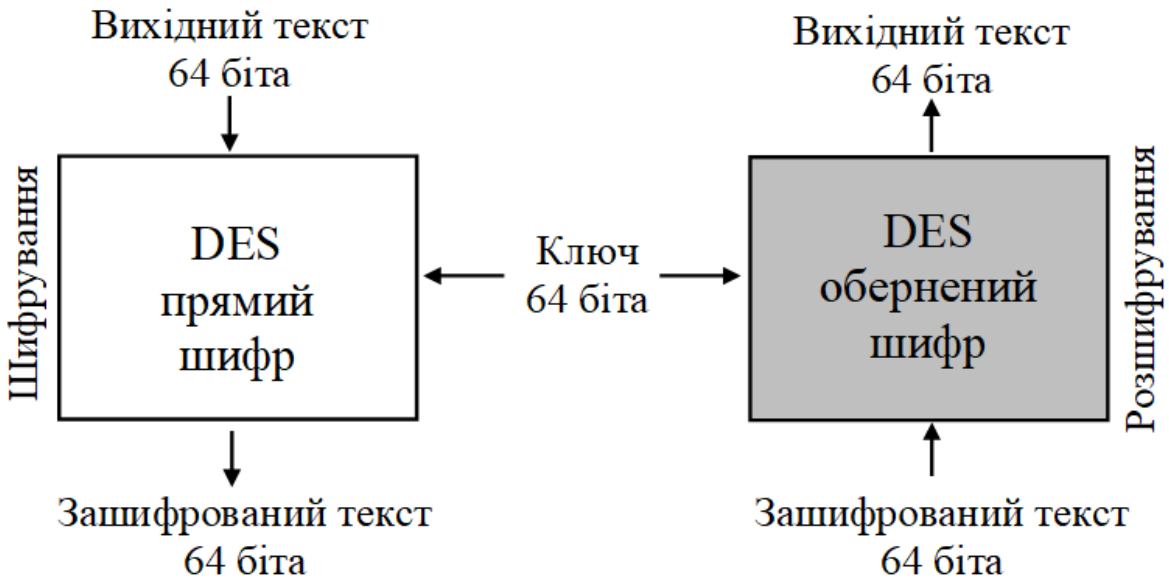


Рис. 4.1. Шифрування і розшифрування даних з використанням стандарту DES

На стороні шифрування DES приймає 64-бітовий вихідний блок даних і породжує 64-бітовий зашифрований блок даних; на стороні розшифрування DES приймає 64-бітовий зашифрований блок даних і породжує 64-бітовий вихідний блок даних. На обох сторонах для шифрування і розшифрування застосовується один і той же 64-бітовий ключ.

Алгоритм DES також використовує комбінацію підстановок і перестановок. DES здійснює шифрування 64-бітових блоків даних за допомогою 64-бітового ключа, в якому значущими є 56 біт (кожний восьмий біт використовується для контролю парності інших біт ключа) [29, 36]. Розшифрування в DES є операцією, оберненою шифруванню, і виконується шляхом повторення операцій шифрування у зворотній послідовності.

4.3. Структура алгоритму шифрування даних DES

Узагальнена схема процесу шифрування в алгоритмі DES показана на рис. 4.2.

Процес шифрування полягає в початковій перестановці бітів 64-бітового блоку, шістнадцяти раундах шифрування і, нарешті, в кінцевій перестановці бітів.

Використання шістнадцяти раундів шифрування обумовлено наступним:

– дванадцять раундів є мінімально необхідними для забезпечення достатнього рівня криптографічного захисту;

– при апаратній реалізації використання шістнадцяти раундів дозволяє повернути перетворений ключ у початковий стан для подальших перетворень;

– ця кількість раундів також необхідна, щоб виключити можливість проведення атаки на блок зашифрованих даних з обох сторін.

Кожний раунд використовує різні (шістнадцять) згенеровані 48-бітові ключі.

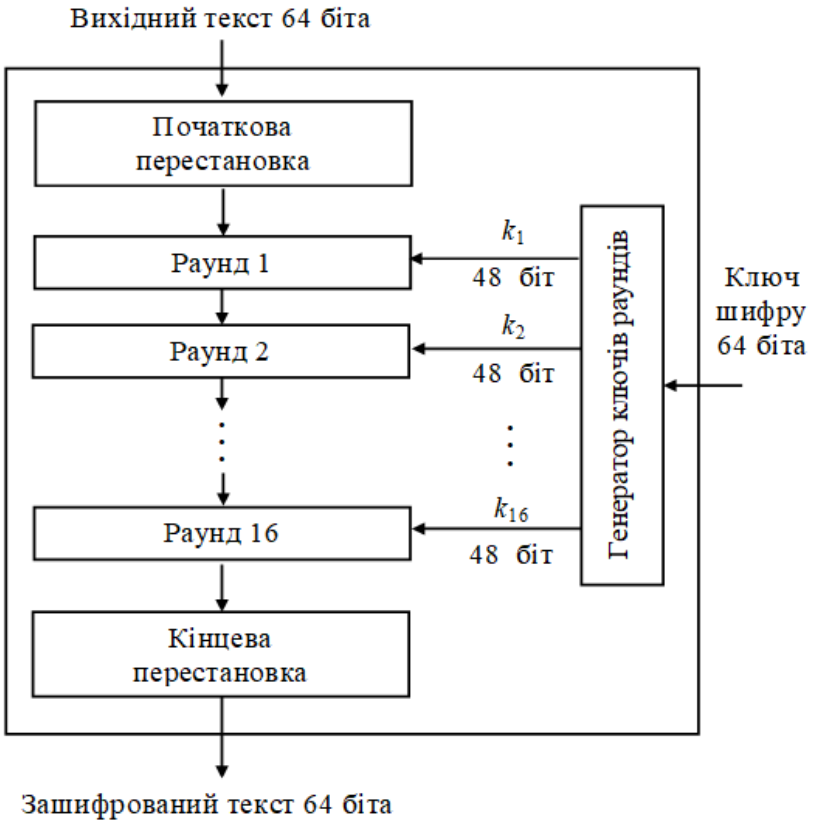


Рис. 4.2. Узагальнена схема процесу шифрування в алгоритмі DES

У деяких реалізаціях DES блоки відкритого повідомлення, перед тим, як вони будуть завантажені в регістри самого пристрою шифрування, проходять процедуру початкової перестановки. Дана процедура застосовується для того, щоб здійснити початкове розсіювання статистичної структури повідомлення [29, 36].

У випадку використання початкової перестановки (*Initial Permutation – IP*), після завершення шістнадцяти раундів, до отриманого блоку застосовується кінцева перестановка (IP^{-1}).

IP перестановка бітів даних представляє собою провідну комутацію з інверсією IP^{-1}, тобто кінцева перестановка обернена початковій. Використані дві перестановки не мають ніякого значення для криптографії в DES. Обидві перестановки – без ключів і визначені наперед. Це дозволяє використовувати таке саме програмне чи апаратне забезпечення для двох сторін процесу: шифрування і розшифрування. Рис. 4.3 показує початкові і кінцеві перестановки.

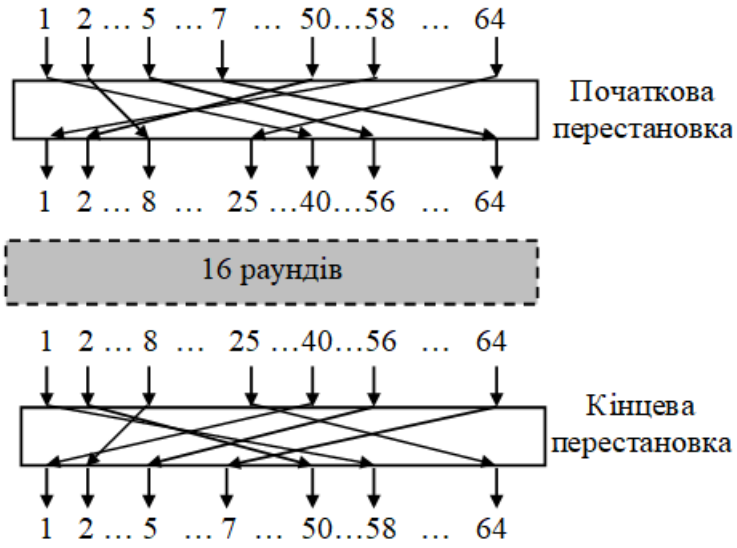


Рис. 4.3. Початкові і кінцеві перестановки DES

Кожна з перестановок приймає на вхід 64-бітові блоки даних і переставляє його елементи за заданим правилом. На рис. 4.3 показано тільки невелику кількість вхідних портів і відповідних вихідних портів. Ці перестановки – прямі перестановки без ключів, які є інверсні одна одній. Наприклад, у початковій перестановці 58-й біт на вході переходить у перший біт на виході. Аналогічно, у кінцевій перестановці перший вхідний біт переходить у 58-й біт на виході. Іншими словами, якщо між цими двома перестановками не існує раунду, 58-й біт, що надійшов на вхід пристрою початкової перестановки, буде доставлений на 58-й вихід кінцевої перестановки.

Правила початкової *IP* та кінцевої IP^{-1} перестановок для 64-бітового блоку даних показані в табл. 4.1 та 4.2. Ця заміна (*IP* та IP^{-1}) ніяк не впливає на стійкість шифру, і користувачі часто задавалися питанням: навіщо її взагалі робити? Один з членів творчого колективу розробників DES, стверджував, що вона полегшує апаратну реалізацію процедури шифрування [36, 40]. Усі перестановки і коди в таблицях підібрані розробниками таким чином, щоб максимально ускладнити процес дешифрування шляхом підбору ключа. Слід відразу зазначити, що всі наведені таблиці є стандартними і їх слід включати в реалізацію алгоритму DES у незмінному вигляді.

Таблиця 4.1

Правила початкової перестановки *IP*

	Номера біт															
Вхід	58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Вхід	57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
Вихід	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Вхід	61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07
Вихід	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Таблиця 4.2

Правила кінцевої перестановки IP^{-1}

	Номера біт															
Вхід	40	08	48	16	56	24	64	32	39	07	47	15	55	23	63	31
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	38	06	46	14	54	22	62	30	37	05	45	13	53	21	61	29
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Вхід	36	04	44	12	52	20	60	28	35	03	43	11	51	19	59	27
Вихід	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Вхід	34	02	42	10	50	18	58	26	33	01	41	09	49	17	57	25
Вихід	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Приклад 4.1. Необхідно визначити результат на виході початкового блоку перестановки, коли на його вхід надходить наступна шістнадцяткова послідовність:

$$(0002\ 0000\ 0000\ 0001)_{16}.$$

Рішення. Вхід має тільки дві одиниці (біт 15 і біт 64); вихід повинен також мати тільки дві одиниці, так як використовується пряма перестановка. Використовуючи табл. 4.1, можна знайти вихід, пов'язаний з цими двома бітами. Біт 15 на вході стає бітом 63 на виході, а біт 64 на вході стає бітом 25 на виході. На виході матимемо тільки дві одиниці – біт 25 і біт 63.

Отже, результат у шістнадцятковому обчисленні:

$$(0000\ 0080\ 0000\ 0002)_{16}.$$

Приклад 4.2. Довести, що початкова і кінцева перестановки, отримані в прикладі 4.1, інверсні одна одній.

Рішення. Перетворимо отриману вихідну послідовність $(0000\ 0080\ 0000\ 0002)_{16}$ у вхідну. Одиничні біти – 25 і 63, інші біти дорівнюють нулю. У кінцевій перестановці 25-й біт переходить у 64-й, а 63-й – у 15-й. Результат:

$$(0002\ 0000\ 0000\ 0001)_{16}.$$

Як видно з результату, початкова і кінцева перестановки – це прямі блоки перестановки, які інверсні один одному.

4.4. Генерація раундових ключів в DES

Генератор ключів створює шістнадцять ключів по 48 біт з ключа шифру завдовжки 56 біт. Однак ключ шифру зазвичай дається як ключ з 64-х біт, в якому 8 додаткових біт є бітами перевірки. Вони фактично відкидаються перед процесом генерації раундових ключів, який показаний на рис. 4.4.

Видалення біт перевірки

Попередній процес перед генерацією ключів – перестановка стиснення, який називають видаленням біт перевірки ключа шифру. Він видаляє біти парності (біти 08, 16, 24, 32, 40, 48, 56, 64) з 64-бітового ключа. Ці біти не впливають на шифрування. Перераховані біти ключа відповідають за те, щоб кожний байт ключа складався з непарного числа одиничних бітів.

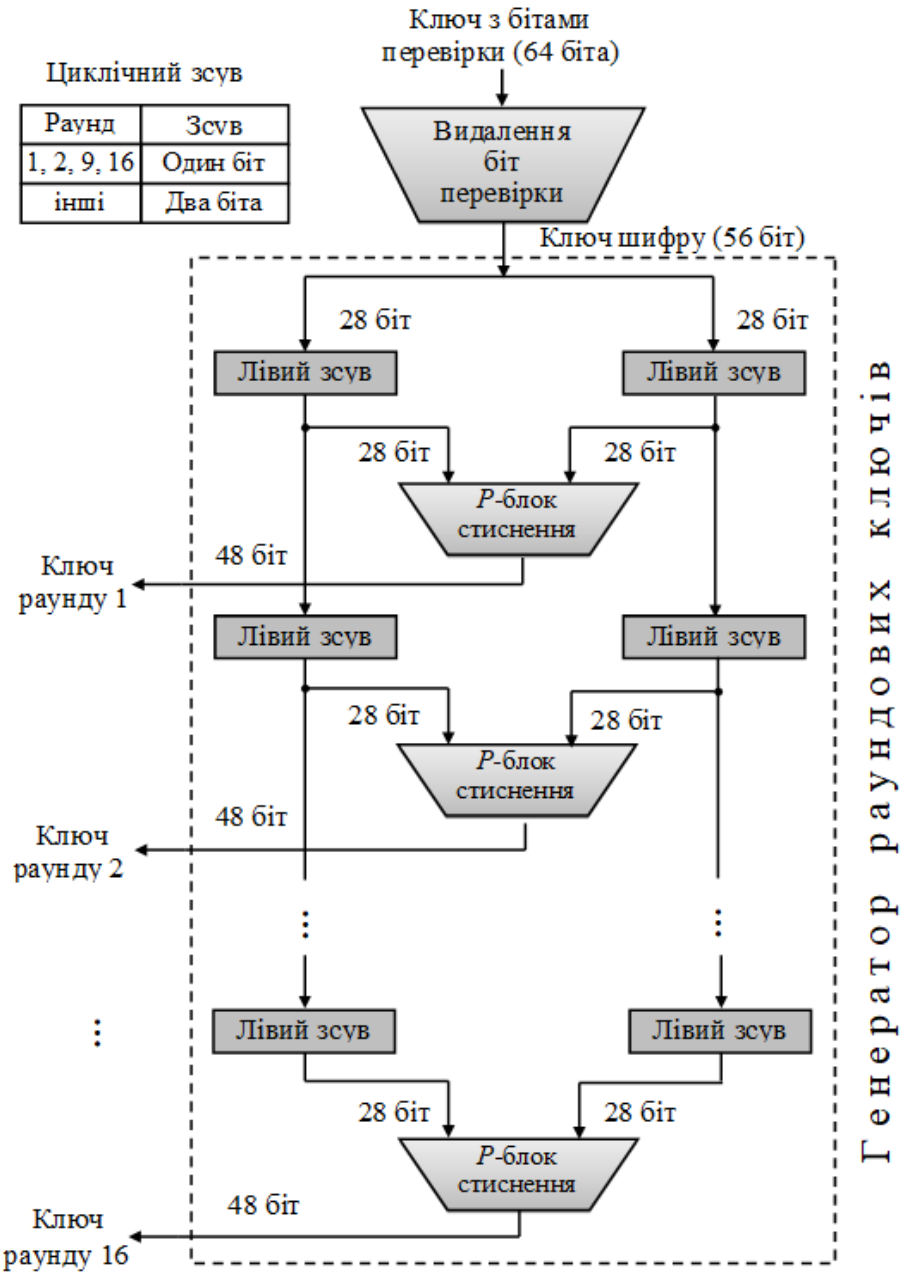


Рис. 4.4. Структурна схема генерації раундових ключів

Отже, в дійсності ключ шифру є 56-бітовим. Для видалення контрольних біт і підготовки ключа до роботи використовується перестановка (*P*-блок стиснення), яка показана в табл. 4.3.

Приклад 4.3. Необхідно визначити результат на виході блока видалення перевірочних біт, якщо ключ шифру дорівнює:

$$K = (ABAB19283746CDCD)_{16}.$$

Рішення. Замінюючи значення ключа шифру на двійкове

$(1010\ 1011\ 1010\ 1011\ 0001\ 1001\ 0010\ 1000\ 0011\ 0111\ 0100\ 0110\ 1100\ 1101\ 1100\ 1101)_2$

і використовуючи таблицю видалення перевірочних біт (табл. 4.3), отримаємо послідовність:

$(1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010\ 0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010)_2$

або $(C3C033A33F0CFA)_{16}$.

Таблиця 4.3

Таблиця видалення перевірочних біт

	Номера біт													
Вхід	57	49	41	33	25	17	09	01	58	50	42	34	26	18
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Вхід	10	02	59	51	43	35	27	19	11	03	60	52	44	36
Вихід	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Вхід	63	55	47	39	31	23	15	07	62	54	46	38	30	22
Вихід	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Вхід	14	06	61	53	45	37	29	21	13	05	28	20	12	04
Вихід	43	44	45	46	47	48	49	50	51	52	53	54	55	56

Циклічний зсув вліво

Після перестановки ключ розбивається на дві частини по 28 біт. Кожна частина циклічно зсувається вліво на один або два біта (в раундах 1, 2, 9 і 16 зсув – на один біт, в інших раундах – на два біта). Після зсуву частини об'єднуються, щоб створити послідовність на 56 біт. В табл. 4.4 показана кількість біт циклічного зсуву вліво для кожного раунду.

Операції циклічного зсуву вліво виконуються для кожної з частин незалежно. Загальна кількість циклічних зсувів у схемі генерації раундових ключів складає 28 біт і вибрано так, щоб після формування останнього раундового ключа значення послідовності дорівнювали значенням послідовності, яка була перед першим зсувом.

Таблиця 4.4

Кількість біт циклічного зсуву вліво для кожного раунду

Номера раундів															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Кількість біт, що зсувається вліво															

Приклад 4.4. Необхідно визначити результат після операції циклічного зсуву вліво послідовності, яка була отримана у прикладі 4.3 для формування другого раундового ключа k_2.

Ліва частина – $(1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010)_2 \rightarrow (C3C033A)_{16}$;

Права частина – $(0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010)_2 \rightarrow (33F0CFA)_{16}$.

Рішення. Для формування другого раундового ключа k_2, відповідно до табл. 4.4, ліва і права частини послідовності повинні бути циклічно зсунуті вліво на два розряди. В результаті отримаємо:

ліва частин – $(0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011)_2 \rightarrow (0F00CEB)_{16}$;

права частин – $(1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000)_2 \rightarrow (CFC33E8)_{16}$.

Об'єднання та перестановка стиснення

Після циклічного зсуву ліва і права частини послідовності об'єднуються, щоб створити блок на 56 біт. Наступний блок перестановки (*P*-блок стиснення) змінює 56 біт на 48 біт, які і використовуються як раундові ключі. Правило перестановки *P*-блоку стиснення показано в табл. 4.5.

Приклад 4.5. Необхідно визначити результат об'єднання лівої та правої частин, які були отримані у прикладі 4.4.

Рішення. Об'єднання можна здійснити у такий спосіб:

$$(0F00CEB)_{16} \,||\, (CFC33E8)_{16} = (0F00CEBCFC33E8)_{16},$$

де || – знак математичної операції конкатенації (об'єднання).

Таблиця 4.5

Таблиця перестановки *P*-блоку стиснення

	Номера біт															
Вхід	14	17	11	24	01	05	03	28	15	06	21	10	23	19	12	04
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	26	08	16	07	27	20	13	02	41	52	31	37	47	55	30	40
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Вхід	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32
Вихід	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Приклад 4.6. Необхідно згенерувати другий раундовий ключ k_2, якщо на вхід *P*-блоку стиснення надійшла послідовність, яка була отримана у прикладі 4.5.

Рішення. Використовуючи двійкове значення отриманої послідовності $(0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011\ 1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000)_2$ і зробивши перестановку за допомогою табл. 4.5, отримаємо другий раундовий ключ k_2:

$$k_2 = (0100\ 0101\ 0110\ 1000\ 0101\ 1000\ 0001\ 1010\ 1011\ 1100\ 1100\ 1110)_2$$

$$\text{або} \qquad k_2 = (4568581ABCCE)_{16}.$$

4.5. Шифрування даних в DES

Після здійснення початкової перестановки *IP* над вхідним блоком даних *m*, отримана 64-бітова послідовність $m_0 = IP(m)$ представляється у вигляді:

$$m_0 = L_0 \,||\, R_0,$$

де L_0, R_0 – відповідно ліва і права частини послідовності бітів, які мають однакову довжину, що дорівнює 32 бітам.

Безпосередньо процес шифрування в DES складається з шістнадцяти раундів, тому блок даних $m_0 = L_0 \| R_0$ перетвориться далі шістнадцять разів за так званою схемою Фейстеля. Кожний раунд DES використовує схему Фейстеля, як це показано на рис. 4.5.

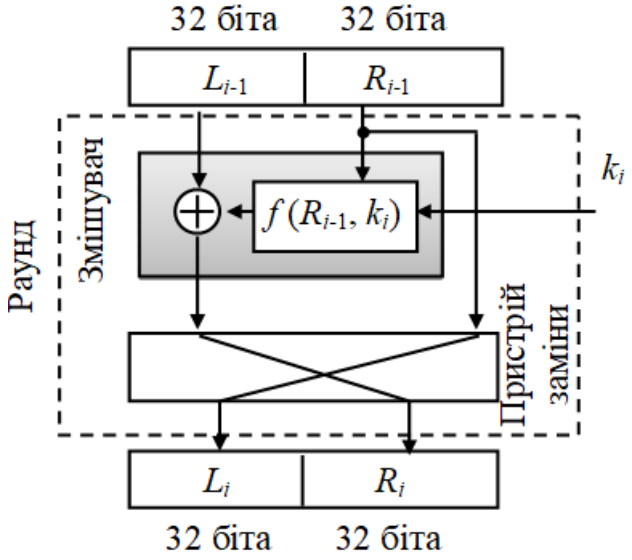


Рис. 4.5. Один раунд шифрування даних в DES

Процес шифрування даних з використанням шістнадцяти раундів можна уявити математично:

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

$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i), \quad i = 1, 2, \ldots, 16,$$

де $f(\bullet)$ – функція шифрування; k_i – i-ий раундовий ключ; $\oplus$ – операція побітового додавання даних за модулем два (*xor*).

Після виконання шістнадцяти раундів шифрування блок даних $m_{16} = L_{16} \| R_{16}$, зазнає оберненої перестановки IP^{-1}. В результаті отримуємо 64-бітовий блок зашифрованих даних c:

$$c = IP^{-1}(m_{16}) = IP^{-1}(L_{16} \| R_{16}).$$

Кожний поточний раунд приймає L_{i-1} і R_{i-1} від попереднього раунду (або початкового блоку перестановки) і створює для наступного раунду L_i і R_i, які надходять на наступний раунд (або кінцевий блок перестановки). Як було вище зазначено, можна прийняти, що кожний раунд має два елементи шифру: змішувач і пристрій заміни. Кожний з цих елементів є оберненим. Пристрій заміни очевидно обернений, він міняє місцями ліву половину даних з правою половиною. Змішувач є оберненим, тому що операція *xor* обернена. Усі необернені елементи зосереджені у функції Фейстеля $f(R_{i-1}, k_i)$.

Функція змішувача f

Основою схеми Фейстеля в DES є функція *f*. Функція *f* за допомогою 48-бітового ключа шифрує 32 крайні праві біти (R_{i-1}), щоб отримати на виході 32-бітове слово. Ця функція містить, як показано на рис. 4.6, чотири операції: *P*-блок розширення; порозрядне додавання за модулем два (⊕); групу *S*-блоків заміни; прямий *P*-блок.

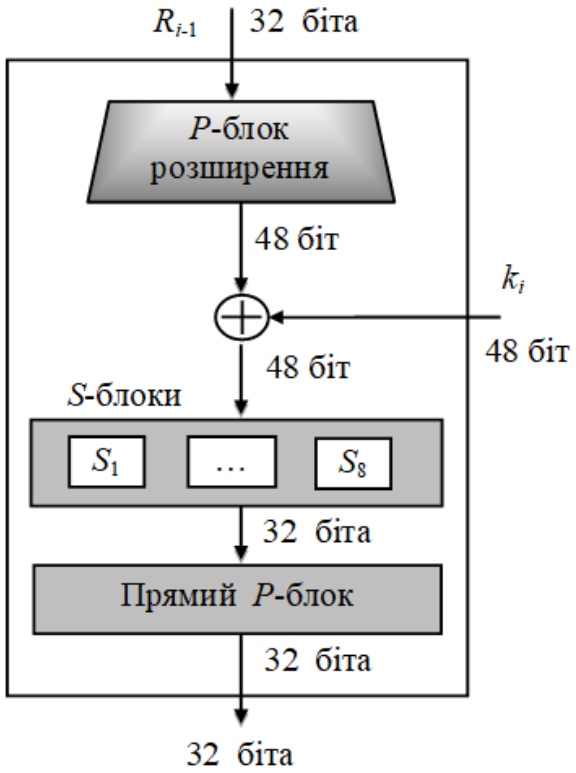


Рис. 4.6. Функція змішувача *f*

P-блок розширення

Так як на вході R_{i-1} має довжину 32 біта, а ключ k_i – довжину 48 біт, то спочатку потрібно розширити R_{i-1} до 48 біт. Для цього R_{i-1} розділяється на 8 секцій по 4 біта. Потім кожна секція на 4 біта розширюється до 6 біт. Ця перестановка з розширенням відбувається за заздалегідь визначеними правилами. Для секції значення вхідних біт 1, 2, 3 і 4 присвоюються бітам 2, 3, 4 і 5 відповідно на виході. Вихідний біт 1 формується на основі вхідного біта 4 з попередньої секції; біт виходу 6 формується з біта 1 у наступній секції. Якщо секції 1 і 8 розглядати як сусідні секції, то такі самі правила застосовуються до бітів 1 і 32. На рис. 4.7 показано входи і виходи *P*-блоку розширення.

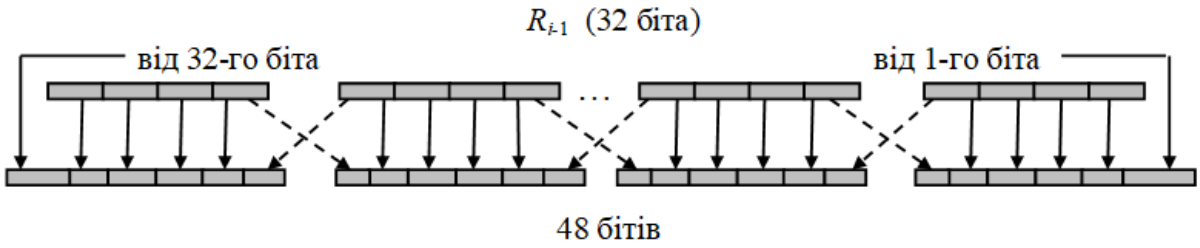


Рис. 4.7. Пояснення процесу розширення *P*-блоку

Хоча відношення між входами і виходами можуть бути визначені математично, але щоб визначити цей *P*-блок, DES використовує табл. 4.6.

Таблиця 4.6

Таблиця перестановки *P*-блоку розширення

Вхід	32	01	02	03	04	05	04	05	06	07	08	09	08	09	10	11
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	12	13	12	13	14	15	16	17	16	17	18	19	20	21	20	21
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Вхід	22	23	24	25	24	25	26	27	28	29	28	29	30	31	32	01
Вихід	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Зверніть увагу, що число виходів 48, а входів – тільки 32. Деякі з входів надходять до більш ніж одного виходу. Наприклад, значення вхідного біта 5 стає значенням бітів виходу 6 і 8.

Приклад 4.7. Необхідно визначити послідовність R'_0 на виході P-блоку розширення, якщо на його вхід надходить послідовність $R_0 = (18CA18AD)_{16}$.

Рішення. Використовуючи двійкове значення R_0

$$R_0 = (18CA18AD)_{16} = (0001\ 1000\ 1100\ 1010\ 0001\ 1000\ 1010\ 1101)_2$$

і зробивши перестановку за допомогою табл. 4.6, отримаємо:

$$R'_0 = (1000\ 1111\ 0001\ 0110\ 0101\ 0100\ 0000\ 1111\ 0001\ 0101\ 0101\ 1010)_2 = (8F16540F155A)_{16}.$$

Порозрядне додавання за модулем два

Після розширення послідовності R_{i-1} до 48 біт використовується операція порозрядного додавання за модулем два (*xor*) над отриманою частиною правої секції R_{i-1} і ключем раунду k_i, який має довжину 48 біт. Зауважимо, що ключ раунду використовується тільки в цій операції.

Приклад 4.8. Необхідно визначити послідовність на виході суматора за модулем два, якщо на його вхід надходить послідовність $R'_0 = (8F16540F155A)_{16}$ і раундовий ключ $k_1 = (194CD072DE8C)_{16}$.

Рішення. Представляючи значення R'_0 і k_1 у двійковому вигляді і використовуючи правило порозрядного додавання за модулем два, отримаємо:

$$R'_0 = (1000\ 1111\ 0001\ 0110\ 0101\ 0100\ 0000\ 1111\ 0001\ 0101\ 0101\ 1010)_2$$
$$k_1 = (0001\ 1001\ 0100\ 1100\ 1101\ 0000\ 0111\ 0010\ 1101\ 1110\ 1000\ 1100)_2$$
$$R'_0 \oplus k_1 = (1001\ 0110\ 0101\ 1010\ 1000\ 0100\ 0111\ 1101\ 1100\ 1011\ 1101\ 0110)_2 = (965A847DCBD6)_{16}.$$

Група S-блоків заміни

DES використовує вісім *S*-блоків, кожний з 6 вхідними бітами і 4 вихідними (рис. 4.8).

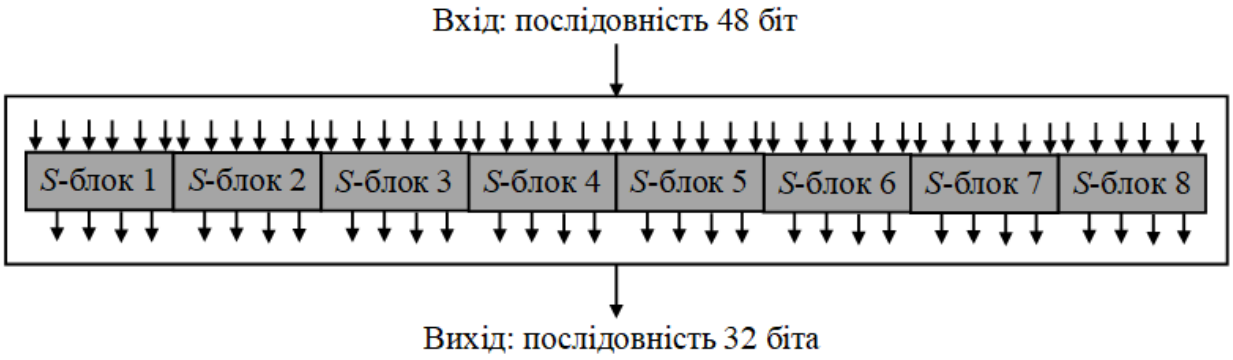


Рис. 4.8. *S*-блоки заміни

Послідовність із 48 біт після операції *xor* розділяється на вісім секцій по 6 біт, і кожна секція потрапляє у відповідний *S*-блок заміни. Результат кожного *S*-блоку заміни – дані завдовжки 4 біта; коли вони об'єднуються, то результат буде представлений послідовністю у 32 біта. Підстановка в кожному блоці відбувається за заздалегідь визначеними правилами, що основані на таблицях з 4 рядків і 16 стовпців (матриця розміром 4×16 з цілими елементами в діапазоні від 0 до 15). Комбінація першого і шостого бітів на вході, якщо їх розглядати як двійковий запис числа, визначають номер рядка таблиці *S*-блоку, а чотири інших вхідних біта (2, 3, 4 і 5) визначають номер стовпця. На перетині визначеного рядка і стовпця знаходиться елемент матриці. Його двійковий запис і буде виходом *S*-блоку (рис. 4.9).

Оскільки для кожного *S*-блоку є власна таблиця, необхідно мати вісім таблиць, наприклад таких, як це показано в табл. 4.7 – 4.14. Значення входу (номер рядка і номер стовпця) і значення виходу даються як десяткові номери, щоб заощадити місце на сторінці. У реальності вони можуть бути замінені двійковими числами.

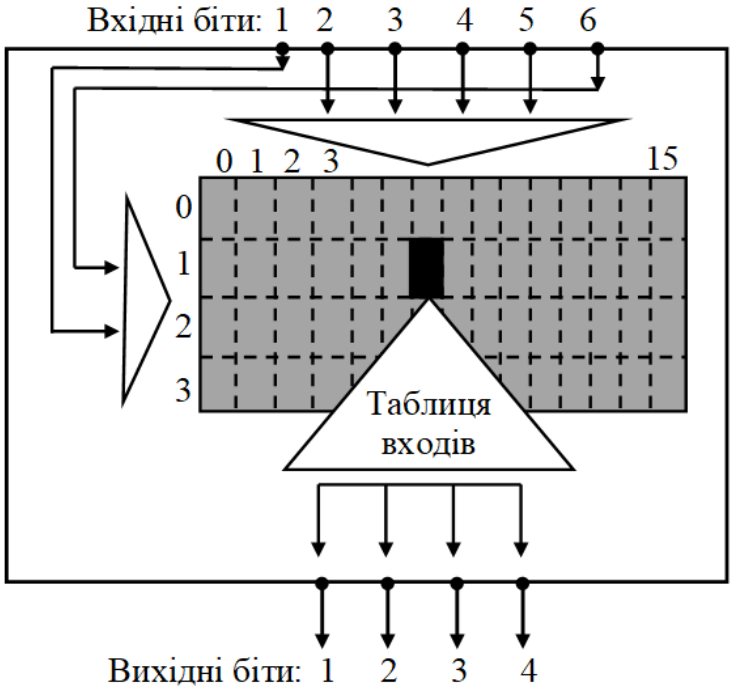


Рис. 4.9. Правила для *S*-блоку заміни

Таблиця 4.7

S_1-блок заміни

		Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
	01	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
	02	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
	03	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Таблиця 4.8

S_2-блок заміни

		Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
	01	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
	02	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
	03	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Таблиця 4.9

S_3-блок заміни

		Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
	01	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
	02	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
	03	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

Таблиця 4.10

S_4-блок заміни

		Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
	01	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
	02	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
	03	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Таблиця 4.11

S_5-блок заміни

		Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
	01	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
	02	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
	03	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Таблиця 4.12

S_6-блок заміни

	Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
	01	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
	02	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
	03	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

Таблиця 4.13

S_7-блок заміни

	Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
	01	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
	02	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
	03	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Таблиця 4.14

S_8-блок заміни

	Номери стовпців															
		00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**
Номери рядків	**00**	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
	01	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
	02	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
	03	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Аналітична складність розшифрування DES залежить від математичних властивостей *S*-блоків заміни, оскільки саме в них реалізуються нелінійні перетворення. Усі інші операції в цьому алгоритмі

носять лінійний характер. Аналітичне обчислення лінійного перетворення не становить складності для криптографічного аналітика.

Нелінійність перетворень, що здійснюються в DES, визначається тільки *S*-блоками заміни. Їх вибір не має достатньо детального обґрунтування. Висловлювалися думки про те, що *S*-блоки заміни мають деяку "лазівку", що дозволяє здійснювати контроль за шифрованим обміном повідомленнями. Офіційна ж версія така: у 1976 році Національне бюро стандартів США заявило, що вибір *S*-блоків заміни визначений наступними вимогами [36, 38, 40]:

– кожний рядок табличного завдання кожного *S*-блоку повинен бути перестановкою множини {0, 1, …, 15};

– *S*-блоки не повинні бути лінійними або афінними функціями своїх входів;

– зміна одного біта входу *S*-блоку повинна приводити до зміни принаймні двох бітів виходу;

– для кожного *S*-блоку і будь-якого входу x значення $S(x)$ та $S(x \oplus (0, 0, 1, 1, 0, 0))$ повинні розрізнятися мінімум двома бітами.

Пояснимо процеси заміни за допомогою *S*-блоків на прикладах.

Приклад 4.9. Вхідна послідовність S_1-блоку дорівнює 101110. Визначити, яка послідовність буде на виході цього блоку?

Рішення. Якщо записати перший і шостий біти разом, то отримаємо у двійковому обчисленні $(10)_2$, яке виражається як число $(2)_{10}$ у десятковому обчисленні. Частина бітів, що залишились, $(0111)_2$ у двійковому обчисленні виглядає як $(7)_{10}$ у десятковому обчисленні. На перетині рядка 2 і стовпця 7 в табл. 4.7 (S_1-блок) знаходиться значення результату – $(11)_{10}$ у десятковому обчисленні або $(1011)_2$ у двійковому обчисленні. Таким чином значення входу 101110 замінюється на значення 1011 на виході S_1-блоку.

Приклад 4.10. Вхідна послідовність S_8-блоку дорівнює 000000. Визначити, яка послідовність буде на виході цього блоку?

Рішення. Якщо записати перший та шостий біти разом, то отримаємо у

двійковому обчисленні $(00)_2$, яке виражається як число $(0)_{10}$ у десятковому обчисленні. Частина бітів, що залишилась, $(0000)_2$ у двійковому обчисленні також є $(0)_{10}$ у десятковому обчисленні. На перетині рядку 0 та стовпця 0 в табл. 4.14 (S_8-блок) знаходиться значення результату – $(13)_{10}$ у десятковому обчисленні або $(1101)_2$ – у двійковому. Таким чином, значення входу 000000 замінюється на значення 1101 на виході S_8-блоку.

Прямий P-блок

Прямий *P*-блок – остання операція у функції *f*. Відношення "вхід-вихід" для цієї операції показані в табл. 4.15. Вони здійснюються за тими самими загальними правилами, як в попередніх таблицях перестановки. Наприклад, сьомий біт входу стає другим бітом виходу.

Таблиця 4.15

Правила *P*-блоку прямої перестановки

Вхід	16	07	20	21	29	12	28	17	01	15	23	26	05	18	31	10
Вихід	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
Вхід	02	08	24	14	32	27	03	09	19	13	30	06	22	11	04	25
Вихід	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Приклад 4.11. Вхідна послідовність *P*-блоку прямої перестановки становить $(EC5965B5)_{16}$. Визначити, яка послідовність буде на виході цього блоку?

Рішення. Використовуючи двійкове значення послідовності

$$(EC5965B5)_{16} = (1110\ 1100\ 0101\ 1001\ 0110\ 0101\ 1011\ 0101)_2$$

і зробивши перестановку за допомогою табл. 4.15, отримаємо:

$$(1000\ 0110\ 1000\ 1101\ 1010\ 1110\ 1111\ 1001)_2 = (868DAEF9)_{16}.$$

Порозрядне додавання в змішувачі

Результат перетворення прямого *P*-блоку додається за модулем два (*xor*) з 32-бітовою послідовністю L_{i-1} і виходить 32-бітова послідовність (див. рис. 4.5).

Приклад 4.12. Необхідно визначити послідовність на виході суматора за модулем два раунду DES, якщо на його вхід надходить послідовність $f(R_0, k_1) = (868DAEF9)_{16}$ та $L_0 = (ACF014A7)_{16}$.

Рішення. Представляючи значення $f(R_0, k_1)$ та L_0 у двійковому вигляді і використовуючи правило порозрядного додавання за модулем два, отримаємо:

$$f(R_0, k_1) = (1000\ 0110\ 1000\ 1101\ 1010\ 1110\ 1111\ 1001)_2$$

$$L_0 = (1010\ 1100\ 1111\ 0000\ 0001\ 0100\ 1010\ 0111)_2$$

$$f(R_0, k_1) \oplus L_0 = (0010\ 1010\ 0111\ 1101\ 1011\ 1010\ 0101\ 1110)_2 = (2A7DBA5E)_{16}.$$

Пристрій заміни раунду DES

Пристрій заміни призначений для завершення операцій одного раунду DES, тобто він формує значення послідовностей L_i та R_i для наступного раунду. Послідовності L_i та R_i будуть визначатися наступним чином (рис. 4.5):

$$L_i = R_{i-1}; \qquad R_i = L_{i-1} \oplus f(R_{i-1}, k_i), \quad i = 1, 2, \ldots, 16,$$

Приклад 4.13. Необхідно визначити послідовності L_1 та R_1 для другого раунду шифрування даних, якщо на вхід пристрою заміни надходять послідовності $R_0 = (305A18CA)_{16}$ та $f(R_0, k_1) \oplus L_0 = (2A7DBA5E)_{16}$.

Рішення. Використовуючи вище наведені вирази отримаємо:

$$L_1 = R_0 = (305A18CA)_{16}; \qquad R_1 = f(R_0, k_1) \oplus L_0 = (2A7DBA5E)_{16}.$$

4.6. Прямий та обернений шифри DES

Використовуючи змішувач і пристрій заміни, можна створити прямий і обернений шифр для кожного з 16 раундів. Прямий шифр використовується на стороні шифрування; обернений шифр – на стороні розшифрування.

Один з методів досягнення поставленої мети (шифрування і розшифрування) полягає в тому, щоб зробити останній раунд відмінним від інших; він буде містити тільки змішувач і не буде містити пристрою заміни, як це показано на рис. 4.10.

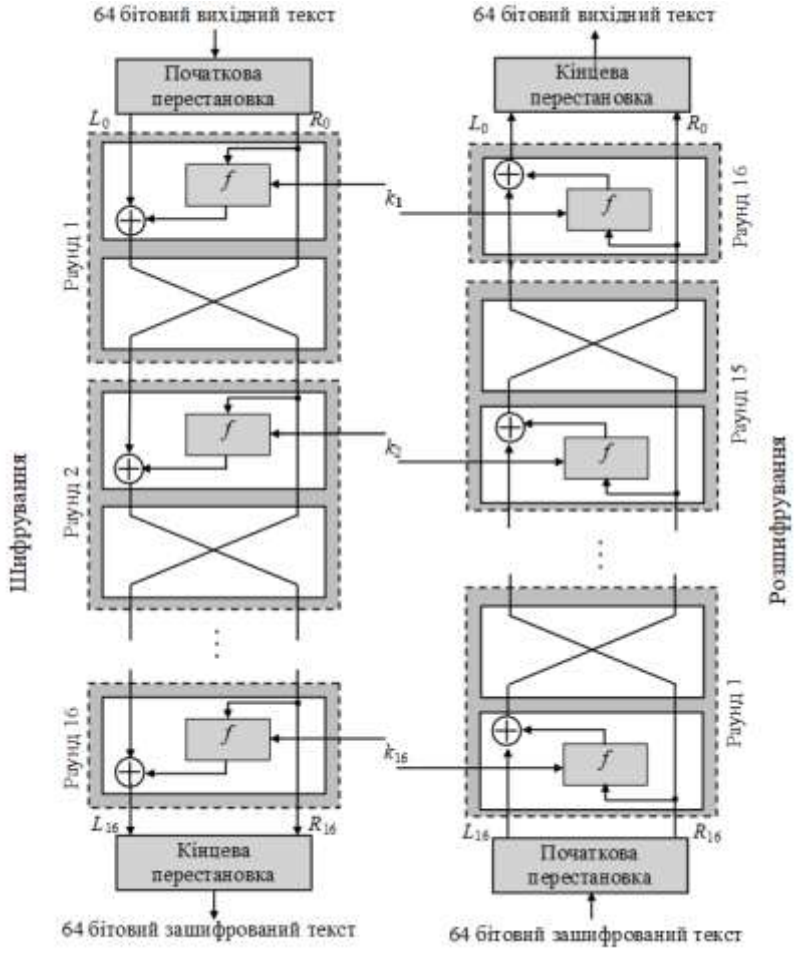


Рис. 4.10. Перший метод прямого і оберненого шифру DES

У розділі 3 було доведено, що змішувач і пристрій заміни самоінверсні. Кінцева і початкова перестановки також інверсні одна одній. Ліва секція вихідних даних на стороні шифрування шифрується: L_0 як L_{16}, а L_{16} розшифровується на стороні розшифрування як L_0. Аналогічна ситуація з R_0 і R_{16}. Треба пам’ятати дуже важливе положення, яке стосується шифрів: ключі

раундів (k_1, k_2, …, k_{16}) використовуються при шифруванні та розшифруванні в оберненому порядку. На стороні шифрування в першому раунді використовується ключ k_1, а в раунді 16 – k_{16}; при розшифруванні в першому раунді використовується ключ k_{16}, а в раунді 16 – k_1. У цьому методі останній раунд не має пристрою заміни.

У першому методі раунд 16 відрізняється від інших раундів тим, що у ньому не застосовується пристрій заміни. Це необхідно, щоб зробити останній і перший раунди в шифрі однаковими.

У другому методі можна зробити всі 16 раундів однаковими, додаючи до 16-го раунду додатковий пристрій заміни (два пристрої заміни дозволяють нейтралізувати один одного). Ця схема прямого і зворотного шифру показана на рис. 4.11.

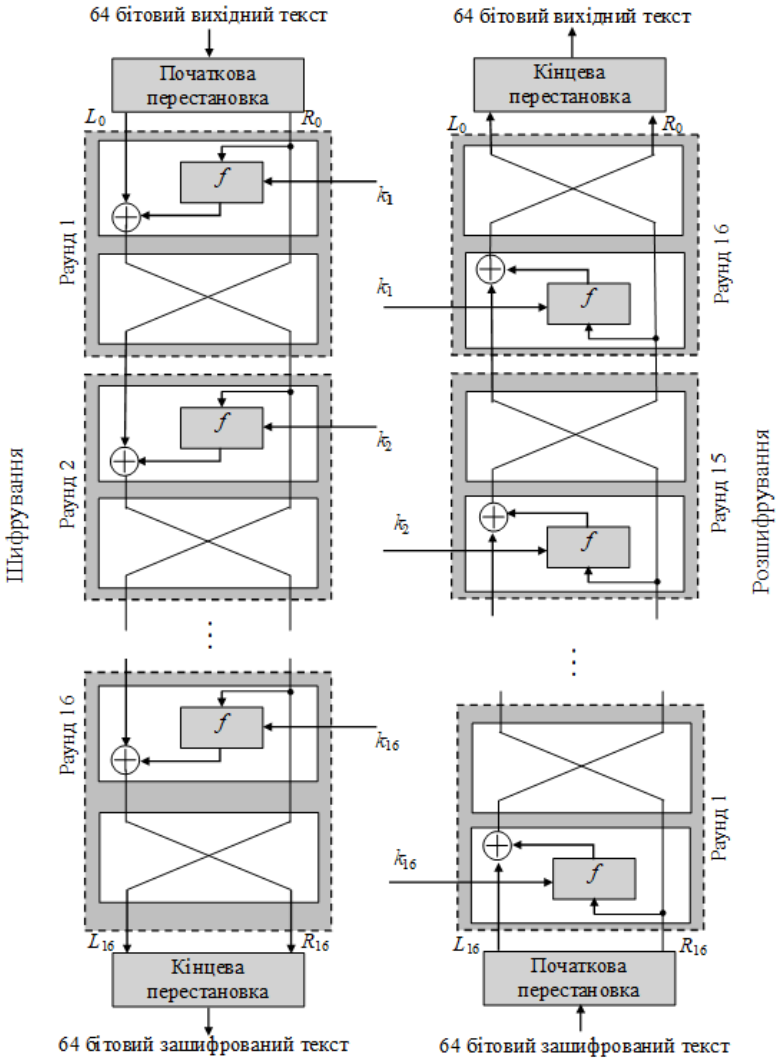


Рис. 4.11. Другий метод прямого і оберненого шифру DES

4.7. Приклади шифрування даних в DES

Перед аналізом DES розглянемо декілька прикладів, щоб зрозуміти, як шифрування і розшифрування змінюють значення бітів у кожному раунді.

Приклад 4.14. Виберемо випадковий блок вихідних даних і випадковий ключ шифру і визначимо, яким має бути блок зашифрованих даних при використанні другого способу шифрування:

– вихідні дані: $(123456ABCD132536)_{16}$;

– ключ шифру: $(ABBA08192637CDDC)_{16}$.

У табл. 4.16 показано результати для кожного раунду.

Таблиця 4.16

Результати 16 раундів для прикладу 4.14

Раунди	Ліві частини L_i	Праві частини R_i	Раундові ключі k_i
1	$18CA18AD_{16}$	$5A78E394_{16}$	$194CD072DE8C_{16}$
2	$5A78E394_{16}$	$4A1210F6_{16}$	$4568581ABCCE_{16}$
3	$4A1210F6_{16}$	$B8089591_{16}$	$06EDA4ACF5B5_{16}$
4	$B8089591_{16}$	$236779C2_{16}$	$DA2D032B6EE3_{16}$
5	$236779C2_{16}$	$A15A4B87_{16}$	$69A629FEC913_{16}$
6	$A15A4B87_{16}$	$2E8F9C65_{16}$	$C1948E87475E_{16}$
7	$2E8F9C65_{16}$	$A9FC20A3_{16}$	$708AD2DDB3C0_{16}$
8	$A9FC20A3_{16}$	$308BEE97_{16}$	$34F822F0C66D_{16}$
9	$308BEE97_{16}$	$10AF9D37_{16}$	$84BB4473DCCC_{16}$
10	$10AF9D37_{16}$	$6CA6CB20_{16}$	$02765708B5BF_{16}$
11	$6CA6CB20_{16}$	$FF3C485F_{16}$	$6D5560AF7CA5_{16}$
12	$FF3C485F_{16}$	$22A5963B_{16}$	$C2C1E96A4BF3_{16}$
13	$22A5963B_{16}$	$387CCDAA_{16}$	$99C31397C91F_{16}$
14	$387CCDAA_{16}$	$BD2DD2AB_{16}$	$251B8BC717D0_{16}$
15	$BD2DD2AB_{16}$	$CF26B472_{16}$	$3330C5D9A36D_{16}$
16	$CF26B472_{16}$	$19BA9212_{16}$	$181C5D75C66D_{16}$

Вихідні дані після початкової перестановки мають наступний вид: $(14A7D67818CA18AD)_{16}$. Після розбиття на ліву і праву частини: $L_0 = (14A7D678)_{16}$ та $R_0 = (18CA18AD)_{16}$. Після 16-го раунду і об'єднання маємо: $L_{16} \| R_{16} = (CF26B47219BA9212)_{16}$. В результаті виконання кінцевої перестановки отримаємо зашифровані дані: $C = (C0B7A8D05F3A829C)_{16}$.

Слід зазначити деякі положення. Права частина кожного раунду співпадає з лівою частиною наступного раунду. Причина в тому, що права частина проходить через змішувач без зміни, а пристрій заміни переносить її в ліву частину. Наприклад, частина R_1 передається через змішувач другого раунду без зміни, але потім, пройшовши пристрій заміни, вона стає L_2.

Приклад 4.15. Розглянемо, як отримувач може розшифрувати зашифровані дані за допомогою однакового ключа. У табл. 4.17 показано результати для кожного раунду.

Перше правило: ключі раундів повинні використовуватися в оберненому порядку. Порівняйте табл. 4.16 і табл. 4.17. Ключ раунду 1 такий самий, як ключ для раунду 16. Значення L_1 і R_1 при розшифруванні ті ж самі, що і значення R_{16} і L_{16} відповідно при шифруванні. Аналогічні збіги будуть отримані і в інших раундах. Це доводить не тільки, що прямий і обернений шифр інверсні один одному, але також те, що кожний раунд при шифруванні має відповідний раунд при розшифруванні в оберненому шифрі. Результат свідчить, що початкові і кінцеві перестановки також є інверсіями одна одній.

Зашифровані дані: $C = (C0B7A8D05F3A829C)_{16}$. Після початкової перестановки мають наступний вид: $(19BA9212CF26B472)_{16}$. Після розбиття на ліву і праву секції: $L_0 = (19BA9212)_{16}$ та $R_0 = (CF26B472)_{16}$. Після 16-го раунду і об'єднання: $L_{16} \| R_{16} = (18CA18AD14A7D678)_{16}$. В результаті виконання кінцевої перестановки отримаємо розшифровані дані: $(123456ABCD132536)_{16}$.

Таблиця 4.17

Результати 16 раундів для прикладу 4.15

Раунди	Ліві частини L_i	Праві частини R_i	Раундові ключі k_i
1	$CF26B472_{16}$	$BD2DD2AB_{16}$	$181C5D75C66D_{16}$
2	$BD2DD2AB_{16}$	$387CCDAA_{16}$	$3330C5D9A36D_{16}$
3	$387CCDAA_{16}$	$22A5063B_{16}$	$251B8BC717D0_{16}$
4	$22A5063B_{16}$	$FF3C485F_{16}$	$99C31397C91F_{16}$
5	$FF3C485F_{16}$	$6CA6CB20_{16}$	$C2C1E96A4BF3_{16}$
6	$6CA6CB20_{16}$	$10AF9D37_{16}$	$6D5560AF7CA5_{16}$
7	$10AF9D37_{16}$	$308BEE97_{16}$	$02765708B5BF_{16}$
8	$308BEE97_{16}$	$A9FC20A3_{16}$	$84BB4473DCCC_{16}$
9	$A9FC20A3_{16}$	$2E8F9C65_{16}$	$34F822F0C66D_{16}$
10	$2E8F9C65_{16}$	$A15A4B87_{16}$	$708AD2DDB3C0_{16}$
11	$A15A4B87_{16}$	$236779C2_{16}$	$C1948E87475E_{16}$
12	$236779C2_{16}$	$B8089591_{16}$	$69A629FEC913_{16}$
13	$B8089591_{16}$	$4A1210F6_{16}$	$DA2D032B6EE3_{16}$
14	$4A1210F6_{16}$	$5A78E394_{16}$	$06EDA4ACF5B5_{16}$
15	$5A78E394_{16}$	$18CA18AD_{16}$	$4568581ABCCE_{16}$
16	$18CA18AD_{16}$	$14A7D678_{16}$	$194CD072DE8C_{16}$

4.8. Аналіз алгоритму DES

DES був підданий ретельному аналізу, щоб виміряти інтенсивність деяких бажаних властивостей у блоковому шифрі. Елементи DES пройшли дослідження на відповідність деяким критеріям.

4.8.1. Властивості алгоритму DES

Дві бажаних властивості блокового шифру – лавинний ефект і ефект повноти.

Лавинний ефект означає, що невеликі зміни у вихідних даних (або ключі шифру) можуть викликати значні зміни в зашифрованих даних. Було

доведено, що DES має всі ознаки цієї властивості [27, 36, 40].

Приклад 4.16. Щоб перевірити лавинний ефект у DES, спробуємо зашифрувати два блоки вихідних даних, які відрізняються тільки одним бітом, за допомогою того самого ключа шифру і визначимо різницю в числі біт у кожному раунді.

Перший випадок:

– відкриті дані: $(0000000000000000)_{16}$;

– ключ шифру: $(23234513987ABA23)_{16}$;

– зашифровані дані: $(4789FD476E82A5F1)_{16}$.

Другий випадок:

– відкриті дані: $(0000000000000001)_{16}$;

– ключ шифру: $(23234513987ABA23)_{16}$;

– зашифровані дані: $(0A4ED5C15A63FEA3)_{16}$.

Хоча два блоки вихідного тексту відрізняються тільки крайнім правим бітом, блоки зашифрованого тексту відрізняються на 29 біт. Це означає, що зміни приблизно в 1,5% вихідних даних створюють зміни близько 45% зашифрованих даних. Табл. 4.18 показує зміну в кожному раунді в порівнянні L_i і R_i у двох зазначених випадках. Можна побачити, що істотні зміни виникають вже в третьому раунді.

Таблиця 4.18

Число різних біт в прикладі 4.16

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Різниця в бітах	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

Ефект повноти полягає в тому, що кожний біт зашифрованих даних повинен залежати від багатьох бітів вихідних даних. Розсіювання і перемішування, зроблене *P*-блоками прямої перестановки, *P*-блоками розширення і *S*-блоками заміни в DES, вказують на дуже сильний ефект повноти.

4.8.2. Критерії розробки DES

Численні випробування DES показали, що він задовольняє деяким із заявлених критеріїв. Нижче коротко обговорюються декілька проблем розробки DES.

Проблеми розробки S-блоків

У розділі 3 були обговорені загальні критерії побудови *S*-блоків. Розглянемо критерії, вибрані для DES. Структура блоків забезпечує перемішування і розсіювання від кожного раунду до наступного. Згідно з цим положенням і деякого аналізу, можна згадати декілька властивостей *S*-блоків заміни:

– входи кожного рядка є перестановки значень між 0 і 15;

– *S*-блоки – нелінійні. Іншими словами, вихід – неафінне перетворення;

– якщо змінювати один єдиний біт на вході, на виході буде змінено два або більше бітів;

– якщо два входи *S*-блоку відрізняються тільки двома середніми бітами (біти 3 і 4), вихідна інформація повинна відрізнятися принаймні двома бітами;

– якщо два входи *S*-блоку відрізняються першими двома бітами (біти 1 і 2) і останніми двома бітами (5 і 6), два виходи повинні бути різні;

– у будь-якому *S*-блоці, якщо єдиний вхідний біт зберігається як константа (0 або 1), то інші біти змінюються випадково так, щоб різниці між числом нулів і одиниць були мінімізовані.

Проблеми розробки P-блоків

Між двома рядами *S*-блоків (у двох наступних раундах), є один *P*-блок перестановки і розширення (32 на 48) і один *P*-блок прямої перестановки (32 на 32). Ці два блоки разом забезпечують розсіювання бітів. Про загальний принцип побудови блоку заміни зазначено у розділі 3. Розглянемо тільки

прикладні блоки, використовувані в DES. У структурі блоків DES були реалізовані наступні критерії:

– кожний вхід *S*-блоку підключається до виходу іншого *S*-блоку попереднього раунду;

– жодний вхід до даного *S*-блоку не з'єднується з виходом від того самого блоку попереднього раунду;

– чотири біта від кожного *S*-блоку надходять у шість різних *S*-блоків наступного раунду;

– жодний з двох бітів виходу *S*-блоку не надходить у той самий *S*-блок наступного раунду;

– вихід S_{j-2}-блоку переходить в один з перших двох бітів S_j-блоку наступного раунду;

– біт виходу від S_{j-1}-блоку переходить в один з останніх двох бітів S_j-блоку наступного раунду;

– вихід S_{j+1}-блоку переходить в один з двох середніх бітів S_j-блоку наступного раунду;

– для кожного *S*-блоку два біта виходу надходять у перші або останні два біта *S*-блоку наступного раунду; інші два біта виходу надходять у середні біти *S*-блоку наступного раунду;

– якщо вихід від S_j-блоку переходить в один з середніх бітів S_k-блоку наступного раунду, то біт виходу від S_k-блоку не може йти в середній біт S_j-блоку. Якщо допустити, що $j = k$, то жодний середній біт *S*-блоку не може надходити в один з середніх бітів того самого *S*-блоку в наступному раунді.

Кількість раундів

DES використовує шістнадцять раундів шифру Фейстеля. Доведено [27, 36, 40], що після того як кожний блок вихідних даних пройшов вісім раундів шифрування, кожний біт зашифрованих даних – функція кожного біта вихідного блоку даних і кожного ключового біта. Зашифровані дані –

повністю випадкова функція вихідних даних і ключа. Звідси начебто витікає, що восьми раундів має бути достатньо для гарного шифрування. Однак експерименти показують, що деякі версії DES з менш ніж шістнадцятьма раундами більш уразливі до атак знання вихідних даних, ніж до атаки “грубої сили”, яка вимагає використання шістнадцяти раундів DES.

4.8.3. Слабкості DES

Протягом минулих багатьох років криптографічними аналітиками були знайдені деякі слабкі (уразливі) місця в DES. Вкажемо на деякі з них, які були виявлені в структурі шифру.

S-блоки заміни. У літературі вказуються принаймні три проблеми *S*-блоків:

– у S_4-блоці три біта виходу можуть бути отримані тим самим способом, що і перший біт виходу: доповненням деяких з вхідних бітів;

– два спеціально обрані входи до масиву *S*-блоку можуть створити той самий вихід;

– можна отримати той самий вихід в одному єдиному раунді, змінюючи біти тільки в трьох сусідніх *S*-блоках.

P-блоки перестановки. У структурі блоків початкової і кінцевої перестановок було знайдено одну загадку. Незрозуміло, чому проектувальники DES використали початкову і кінцеву перестановки. Ці перестановки не вносять ніяких нових властивостей з погляду безпеки. У *P*-блоці перестановки і розширення перші і четверті біти послідовностей на 4 біта повторюються.

Розмір ключа шифру DES

Критики стверджують, що найбільша слабкість DES – це розмір ключа (56 біт). Щоб провести атаку “грубої сили” блоку зашифрованих даних, зловмисник повинен перевірити 2^{56} ключів:

– використовуючи доступну сьогодні технологію, можна перевірити один мільйон ключів за секунду. Це означає, що знадобиться більш ніж дві тисячі років, щоб провести атаку “грубої сили” на DES, використовуючи комп’ютер тільки з одним процесором;

– якщо можна зробити комп’ютер з одним мільйоном чипів процесорів (паралельна обробка), то можна перевірити всю множину ключів приблизно за 20 годин. Коли був уведений DES, вартість такого комп’ютера була більшою, ніж декілька мільйонів доларів, але вона швидко знизилася. Спеціальний комп’ютер був створений у 1998 році, завдяки йому ключ був знайдений за 112 годин;

– комп’ютерні мережі можуть моделювати паралельну обробку. У 1977 році команда дослідників використала 3500 комп’ютерів, підключених до Internet, щоб знайти ключ DES за 120 днів. Безліч ключів було розділено серед усіма цими комп’ютерами, і кожний комп’ютер був відповідальний за перевірку частини домену DES. Якщо 3500 зв’язаних в мережу комп’ютерів змогли знайти ключ за 120 днів, то спільнота з 42000 членів може знайти ключ за 10 днів.

Наведене вище показує, що DES з розміром ключа шифру 56 біт не забезпечує достатньої безпеки.

Слабкі ключі шифру DES

Чотири ключі з 2^{56} можливих ключів називаються слабкими ключами. Слабкі ключі – це такі, які після операції видалення перевірочних біт складаються з усіх нулів або усіх одиниць або половини нулів і половини одиниць. Такі ключі показано в табл. 4.19.

Ключі раундів, створені від будь-якого з цих слабких ключів, – такі самі і мають такий самий тип, що і ключ шифру. Наприклад, ці чотири ключа створюють перший ключ, який складається з усіх нулів або усіх одиниць, або наполовину з нулів і одиниць. Це відбувається з тієї причини, що алгоритм

генерування ключів спочатку ділить ключ шифру на дві половини. Зсув або перестановка блоку не змінюють блок, якщо він складається з усіх нулів або усіх одиниць або наполовину з нулів і одиниць.

Таблиця 4.19

Слабкі ключі шифру DES

Ключі до видалення перевірочних біт і перестановки (64 біта)	Діючі ключі (56 біт)
$0101\ 0101\ 0101\ 0101_{16}$	$0000\ 000\ 0000\ 000_{16}$
$1F1F\ 1F1F\ 1F1F\ 1F1F_{16}$	$0000\ 000\ FFFF\ FFF_{16}$
$E0E0\ E0E0\ E0E0\ E0E0_{16}$	$FFFF\ FFF\ 0000\ 000_{16}$
$FEFE\ FEFE\ FEFE\ FEFE_{16}$	$FFFF\ FFF\ FFFF\ FFF_{16}$

У чому небезпека використання слабких ключів? Якщо зашифрувати блок даних слабким ключем і згодом ще раз зашифрувати результат тим самим слабким ключем, виходить початковий блок даних. Процес створює той самий початковий блок даних, якщо розшифровувати блок двічі. Іншими словами, кожний слабкий ключ є інверсія самого себе:

$$E_K(E_K(M)) = M,$$

як це показано на рис. 4.12.

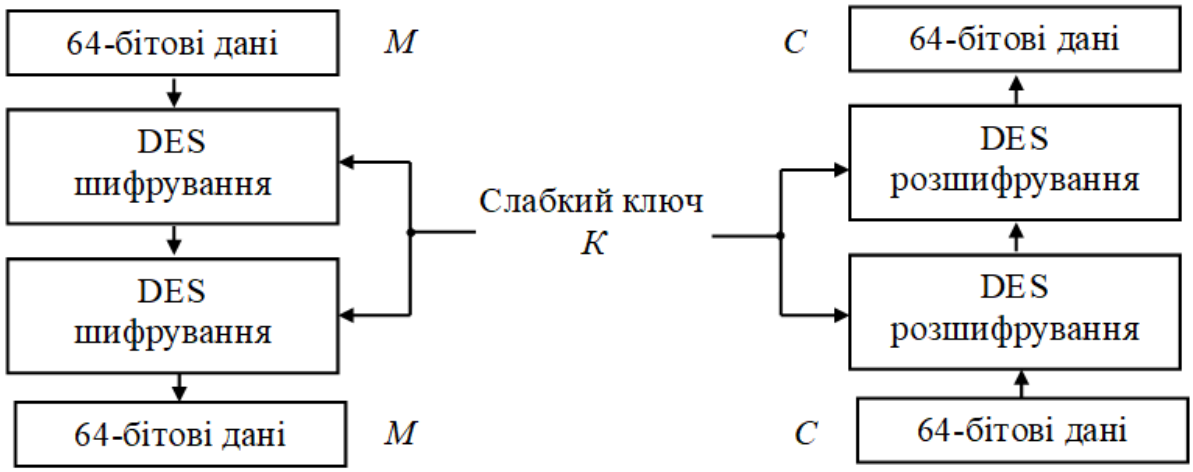


Рис. 4.12. Подвійне шифрування і розшифрування даних слабким ключем

Слабких ключів треба уникати, бо противник може легко розпізнати їх на перехопленому зашифрованому повідомленні. Якщо після двох етапів розшифрування результат такий самий, противник визначає, що він знайшов ключ.

Приклад 4.17. Спробуємо застосувати перший слабкий ключ з табл. 4.19, щоб двічі зашифрувати блок даних. Після того як проведено два шифрування з однаковим ключем, отримаємо початковий блок даних. Зверніть увагу, що при цьому жодного разу не використовувався алгоритм розшифрування, а тільки проведено двічі шифрування.

Ключ: $(0101010101010101)_{16}$.

Вихідний блок даних: $(1234567887654321)_{16}$.

Зашифрований блок даних: $(814FE938589154F7)_{16}$.

Ключ шифру: $(0101010101010101)_{16}$.

Вихідний блок даних: $(814FE938589154F7)_{16}$.

Зашифрований блок даних: $(1234567887654321)_{16}$.

Напівслабкі ключі шифру DES

Є шість ключових пар, які названі напівслабкими ключами. Ці шість пар показано в табл. 4.20 (64 біта перед видаленням перевірочних біт).

Таблиця 4.20

Напівслабкі ключі шифру DES

Перший ключ пари	Другий ключ пар
$01FE\ 01FE\ 01FE\ 01FE_{16}$	$FE01\ FE01\ FE01\ FE01_{16}$
$1FE0\ 1FE0\ 0EF1\ 0EF1_{16}$	$E01F\ E01F\ F10E\ F10E_{16}$
$01E0\ 01E1\ 01F1\ 01F1_{16}$	$E001\ E001\ F101\ F101_{16}$
$1FFE\ 1FFE\ 0EFE\ 0EFE_{16}$	$FE1F\ FE1F\ FE0E\ FE0E_{16}$
$011F\ 011F\ 010E\ 010E_{16}$	$1F01\ 1F01\ 0E01\ 0E01_{16}$
$E0FE\ E0FE\ F1FE\ F1FE_{16}$	$FEE0\ FEE0\ FEF1\ FEFL_{16}$

Напівслабкі ключі створюють тільки два різних ключа раунду і потім повторюють їх вісім разів. Крім того, ключі раундів, створені від кожної пари, ті самі в різному порядку. Щоб проілюструвати ідею, створимо ключі раундів від першої пари, як показано у табл. 4.21. Як показує аналіз цієї таблиці, є вісім однакових ключів раундів у кожному напівслабкому ключі. Крім того, ключ раунду 1 для першого ключа пари той самий, що і ключ раунду 16 для другого ключа пари; ключ раунду 2 для першого ключа пари той самий, що і ключ раунду 15 для другого ключа пари, і так далі. Це означає, що ключі інверсні один одному:

$$E_{K_2}\left(E_{K_1}\left(M\right)\right) = M,$$

як показано на рис. 4.13.

Таблиця 4.21

Раундові ключі для напівслабких ключів шифру DES

Раунди	Раундові ключі для першого ключа пари	Раундові ключі для другого ключа пари
1	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
2	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
3	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
4	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
5	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
6	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
7	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
8	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$
9	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
10	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
11	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
12	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
13	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
14	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
15	$9153E54319BD_{16}$	$6EAC1ABCE642_{16}$
16	$6EAC1ABCE642_{16}$	$9153E54319BD_{16}$

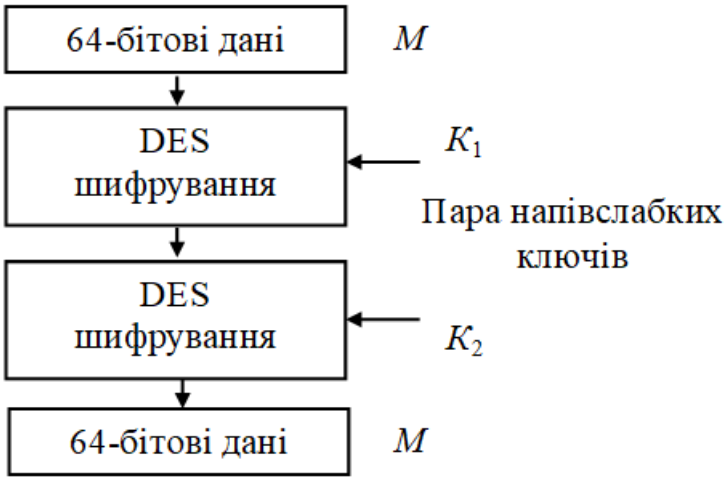


Рис. 4.13. Подвійне шифрування даних напівслабким ключем

Можливо слабкі ключі шифру DES

Також є 48 ключів, які називаються можливо слабкими ключами. Можливо слабкий ключ створює тільки чотири різні ключі раундів; іншими словами, шістнадцять ключів раундів розділені на чотири групи, і кожна група складається з чотирьох однакових ключів раундів.

Приклад 4.18. Яка ймовірність випадкового вибору слабкого, напівслабкого або можливо слабкого ключа?

Рішення. Множина ключів DES дорівнює 2^{56}. Загальна кількість вищезазначених ключів – 64 (4 + 12 + 48). Ймовірність вибору одного з цих ключів дорівнює $8{,}8\times10^{-16}$, тобто виключно мала.

4.9. Багаторазове застосування DES

Як вже було зазначено, основна критика DES спрямована на довжину ключа шифру. Можливі технології і можливості паралельних процесорів роблять реальною атаку “грубої сили”.

Одне з рішень для поліпшення безпеки – це застосування алгоритму шифрування Advanced Encryption Standard (AES), який буде розглянуто в наступному розділі. Друге рішення – багаторазове (каскадне) застосування алгоритму шифрування DES з різними ключами шифру. Це рішення, яке

використовувалося деякий час, не вимагало істотних інвестицій у нове програмне забезпечення і апаратні засоби. Розглянемо такий підхід.

Як відомо з розділу 3, підстановка, яка розміщує усі можливі входи в усі можливі виходи, є групою з відображеннями елементів множини і набором операцій. У цьому випадку використання двох послідовних відображень даремне, тому що можна завжди знайти третє відображення, яке еквівалентне композиції цих двох (властивість замкнутості). Це означає, що якщо DES – група, то одноразовий DES з ключем k_3 робить те саме (див. рис. 4.14).

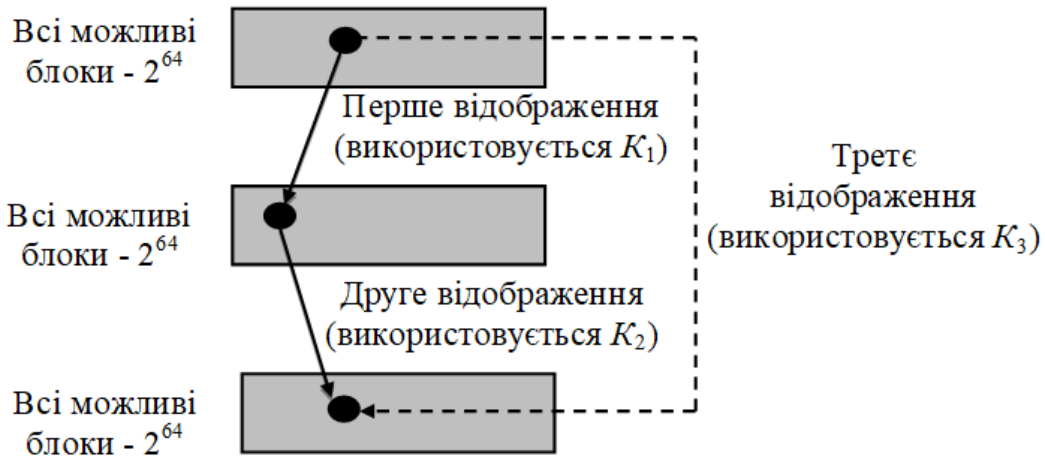


Рис. 4.14. Композиція відображень

На щастя, DES не група. Номер можливих входів або виходів в DES – $N = 2^{64}$. Це означає, що $N! = (2^{64}!) = 10347380000000000000000$ відображень. Щоб зробити DES групою, необхідно підтримати усі ці відображення з розміром ключа $\log_2 (2^{64}!) \approx 270$ біт. Оскільки довжина ключа шифру DES – 56 біт, то це тільки маленька частина необхідного величезного ключа.

Якщо DES не є групою, то малоймовірно, що можна знайти ключ k_3, такий, що:

$$E_{K_2} (E_{K_1}(M)) = E_{K_3}(M).$$

Це означає, що можна використовувати дворазові або триразові DES, щоб збільшити розмір ключа.

4.9.1. Дворазовий DES

Перший підхід полягає в тому, щоб використовувати дворазовий DES. За цього підходу застосовується два типи прямих шифрів DES для шифрування і два типи обернених шифрів для розшифрування. Кожний тип використовує різні ключі шифру, це означає, що розмір ключа подвоївся (112 біт). Однак дворазовий DES уразливий до атаки знання відкритих даних.

На перший погляд дворазові DES збільшують число випробувань при пошуку ключа від 2^{56} (в одноразовому DES) до 2^{112} (у дворазовому DES). Однак за використання атаки на відомий вхідний текст, званої атакою “зустрічі посередині”, можна довести, що дворазовий DES покращує цю стійкість (тільки до 2^{57} випробувань), але не надзвичайно (до 2^{112}). Рис. 4.15. показує структуру дворазового DES.

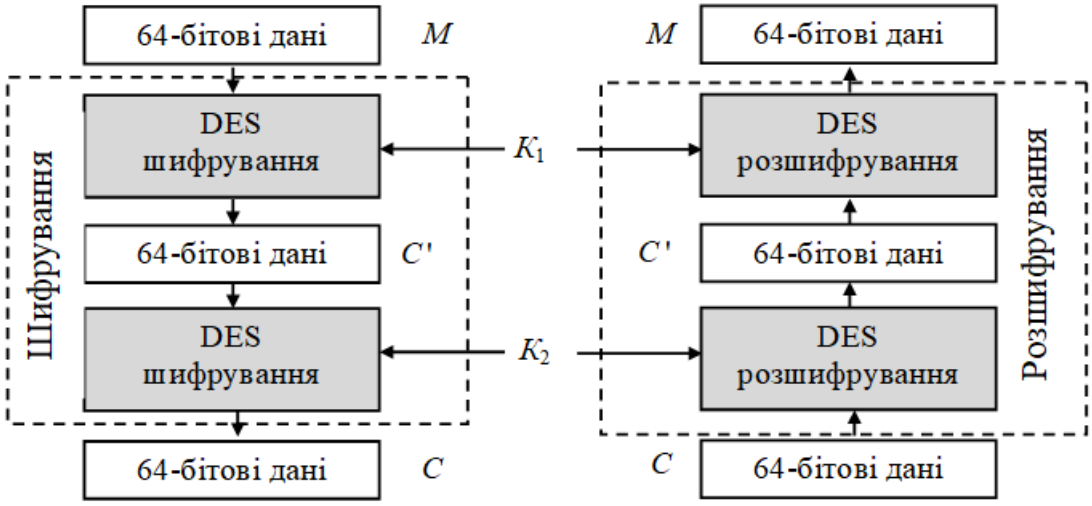


Рис. 4.15. Структура дворазового DES

Відправник використовує два ключа шифру, K_1 і K_2, щоб зашифровувати вихідний блок даних M у зашифрований блок даних C; отримувач використовує зашифрований блок даних C і два ключа шифру, K_1 і K_2, для відновлення вихідного блоку даних M.

У середній точці C' – блок даних, створений першим шифруванням або першим розшифруванням. Для забезпечення належного функціонування він

повинен бути однаковим для шифрування і розшифрування. Іншими словами, маємо два співвідношення:

$$C' = E_{K_1}(M) \quad \text{і} \quad C' = D_{K_2}(C).$$

Припустимо, що зловмисник перехопив попередню пару M і C (атака знання вихідних даних). Базуючись на першому співвідношенні із згаданих вище, зловмисник зашифровує M, використовуючи усі можливі значення (2^{56}) ключа K_1, і записує всі значення, отримані для C'. Базуючись на другому співвідношенню, зловмисник розшифровує C', використовуючи всі можливі значення (2^{56}) ключа K_2. Він також записує всі значення, отримані для C'. Далі зловмисник створює дві таблиці, відсортовані відповідно до значень C'. Він порівнює значення C', доки не знаходить ті пари K_1 і K_2, для яких значення C' є однаковим в обох таблицях (як показано на рис. 4.16).

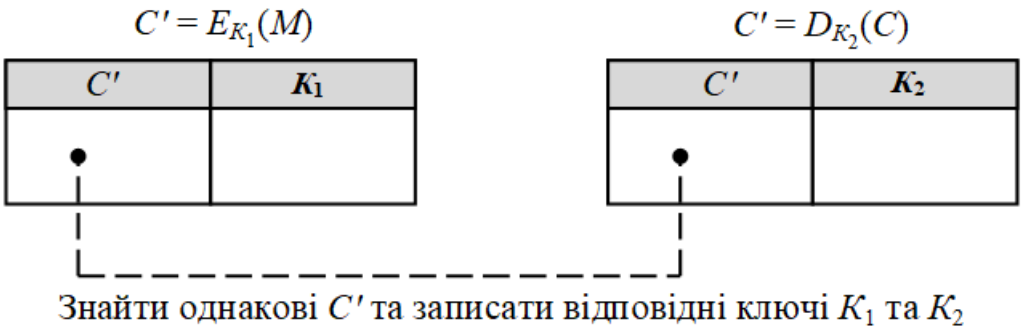


Рис. 4.16. Таблиці для атаки “зустрічі посередині”

Зверніть увагу, що повинна бути принаймні одна пара, тому що вона робить вичерпний пошук комбінації двох ключів:

1. Якщо є тільки одна відповідність, то зловмисник знайшов два ключа (K_1 і K_2). Якщо є більше ніж один кандидат, то зловмисник переходить до наступного кроку.

2. Зловмисник бере іншу перехоплену пару зашифрованого блоку даних і вихідного блоку даних і використовує кожного кандидата для отримання пари ключів, щоб встановити, чи може він отримати зашифрований блок даних з вихідного блоку даних. Якщо він знову

знаходить більше ніж одного кандидата у вигляді пари ключів, то він повторює крок 2 доти, поки нарешті не знайде унікальну пару.

Було доведено [27, 38], що після застосування другого кроку до декількох перехоплених пар "зашифровані дані – вихідні дані" ключі були знайдені. Це означає, що замість того, щоб використовувати пошук ключів шифру за допомогою 2^{112} випробувань, зловмисник використовує 2^{56} випробувань пошуку ключа і перевіряє двічі (дещо більше випробувань потрібно, якщо знайдено на першому кроці не єдиний кандидат). Іншими словами, рухаючись від одноразового DES до дворазового DES, збільшується обсяг випробувань від 2^{56} до 2^{57} (а не до 2^{112}, як це здається за поверхневого підходу).

4.9.2. Триразовий DES

Для того, щоб поліпшити безпеку DES, було запропоновано триразовий DES. Він використовує три каскади DES для шифрування і розшифрування. Сьогодні використовуються дві версії триразових DES: триразовий DES з двома ключами шифру і триразовий DES з трьома ключами шифру.

Триразовий DES з двома ключами шифру

У триразовому DES з двома ключами шифру є тільки два ключі шифру: K_1 і K_2. Перший і третій каскади використовують K_1; другий каскад використовує K_2. Щоб зробити триразовий DES сумісним з DES, середній каскад застосовує розшифрування (обернений шифр) на стороні шифрування і шифрування (прямий шифр) на стороні розшифрування. У такий спосіб повідомлення, зашифроване DES з ключем шифру K, може бути розшифровано триразовим DES, якщо тільки $K_1 = K_2 = K$. Хоча триразовий DES з двома ключами шифру також вразливий при атаці "на відомий вхідний текст", він набагато стійкіше, ніж дворазовий DES (прийнятий в свій час для банків). На рис. 4.17 показано триразовий DES з двома ключами шифру.

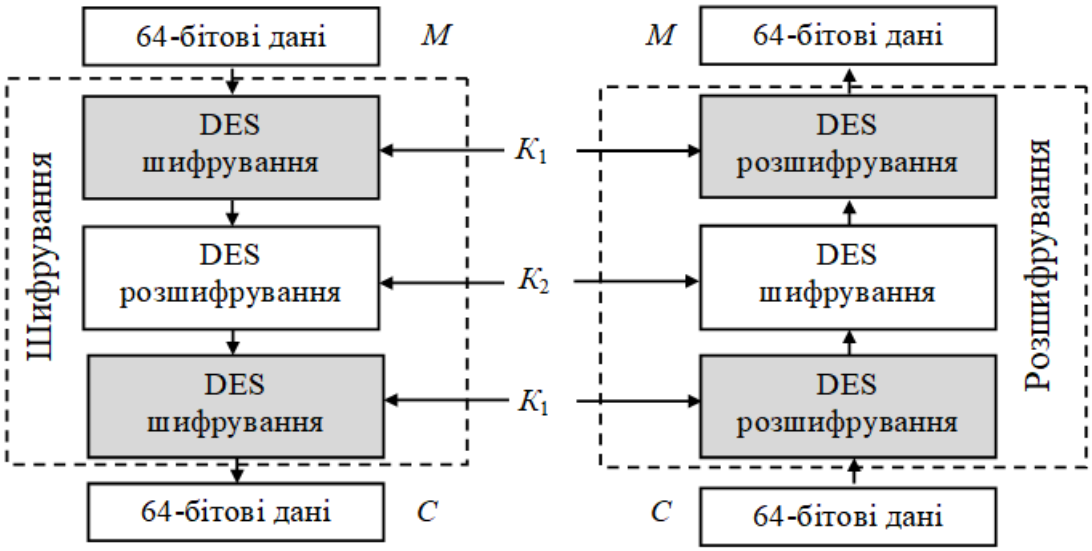


Рис. 4.17. Триразовий DES з двома ключами шифру

Триразовий DES з трьома ключами шифру

Можливість атак “на відомий вхідний текст” за триразового DES з двома ключами шифру змусила деякі додатки використовувати триразовий DES з трьома ключами шифру. Алгоритм може застосовувати три каскади прямого шифру DES на стороні шифрування і три каскади обернених шифрів на стороні розшифрування. На рис. 4.18 показано триразовий DES з трьома ключами шифру.

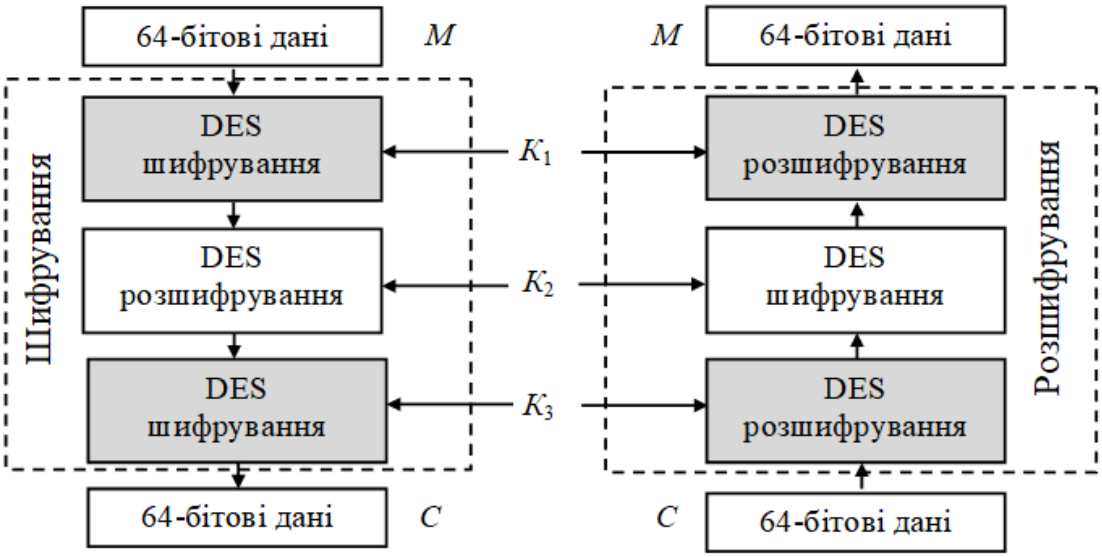


Рис. 4.18. Триразовий DES з трьома ключами шифру

Для сумісності з одноразовим DES сторона шифрування використовує EDE, а сторона розшифрування – DED. E (encryption) – каскад шифрування, D (decryption) – каскад розшифрування. Загальна довжина ключа шифру у цьому варіанті зростає (112 + 56 = 168 біт).

Сумісність з одноразовим DES забезпечується за $K_1 = K$ і установкою K_2 і K_3 до того самого довільного ключа. Триразовий DES з трьома ключами шифру використовується багатьма додатками, такими як PGP (Pretty Good Privacy – досить "хороша" конфіденційність), оскільки забезпечує високу захищеність властивості інформації – конфіденційність. Триразовий DES є достатньо популярною альтернативою DES і використовується при управлінні ключами в стандартах ANSI X9.17 і ISO 8732. Деякі криптографічні аналітики пропонують для ще більш надійного шифрування використовувати п'ятиразовий DES з трьома або п'ятьма ключами.

4.10. Безпека DES

DES, як перший блоковий шифр, що має важливе значення, пройшов через багато випробувань на безпеку. Серед вжитих атак лише три становлять інтерес: атака "грубої сили", диференційний і лінійний криптографічні аналізи.

Атака "грубої сили" DES

Вже обговорювалась слабкість DES з коротким ключем шифру. Слабкість ключа шифру спільно з іншими розглянутими недоліками, робить очевидним, що DES може бути зламаний з числом випробувань 2^{56}. Однак сьогодні більшість додатків використовує або триразовий DES з двома ключами шифру (розмір ключа 2^{112}), або триразовий DES з трьома ключами шифру (розмір ключа 2^{168}). Ці дві багаторазові версії DES дозволяють йому показувати істотну стійкість до атак "грубої сили".

Диференціальний криптографічний аналіз DES

У розділі 3 вже обговорювали методику диференціального криптографічного аналізу для сучасних блокових шифрів. DES не є стійким до такого виду атаки. Однак багато що вказує, що розробники DES вже знали про цей тип атаки і проектували *S*-блоки та спеціально вибирали число раундів таким, щоб зробити DES стійким до цього типу атаки. Сьогодні показано, що DES може бути зламаний, використовуючи диференціальний криптографічний аналіз, якщо маємо 2^{47} вибірок вихідних даних або 2^{55} відомих вихідних даних. Хоча це виглядає більш ефективно, ніж в атаці "грубої сили", припустити, що хтось знає таку кількість інформації практично неможливо. Тому можна сказати, що DES є стійким до диференціального криптографічного аналізу. Також показано, що збільшення числа раундів до 20 збільшує число необхідних вибірок вихідних даних для атаки більш ніж 2^{64}. Таке збільшення неможливо, тому що число блоків вихідних даних в DES тільки 2^{64}.

Лінійний криптографічний аналіз DES

Методика лінійного криптографічного аналізу для сучасних блокових шифрів обговорювалась в розділі 3. Лінійний криптографічний аналіз – більш нова методика, ніж диференціальний криптографічний аналіз. DES більш уразливий до застосування лінійного криптографічного аналізу, ніж до диференціального крипто аналізу: ймовірно тому, що цей тип атак не був відомий проектувальникам DES і *S*-блоки не є дуже стійкими до лінійного криптографічного аналізу. Показано, що DES може бути зламаний з використанням 2^{43} пар відомих вихідних даних. Однак з практично погляду перехоплення такої кількості пар малоймовірне.

Контрольні питання до розділу 4

1. Який розмір блоку даних, розмір ключа шифру і розмір ключів раунду в DES?

2. Яка кількість раундів в DES?

3. Яка кількість перестановок використовується в алгоритмі DES?

4. Скільки операцій *xor* використовується в DES?

5. Для чого у функції DES необхідна операція, яка здійснює перестановку з розширенням?

6. Яка різниця між слабким, напівслабким і можливо слабким ключем шифру DES?

7. Що таке дворазовий DES? Яка атака зробила його марним?

8. Що таке триразовий DES? Що таке триразовий DES з двома ключами шифру? Що таке триразовий DES з трьома ключами шифру?

9. Значення послідовності вхідних даних в DES становить $(3899567890ABCDEF)_{16}$. Визначити значення послідовності на виході блоку початкової перестановки.

10. Значення послідовності R_0 в DES становить $(90ACE8A5)_{16}$. Визначити значення послідовності на виході *P*-блоку розширення.

11. Значення послідовності на виході *P*-блоку розширення в DES становить $(9C2B57D029B4)_{16}$. Значення відповідного раундового ключа становить $(7968581CABFE)_{16}$. Визначити значення послідовності на виході операції *xor* функції Фейстеля змішувача.

12. Значення послідовності на виході операції *xor* змішувача в DES становить $(2344FEA79BD1)_{16}$. Визначити значення на виходах блоків заміни.

13. Значення послідовності на виходах *S*-блоків заміни змішувача в DES становить $(DCF375FE)_{16}$. Визначити значення послідовності на виході прямого *P*-блоку.

14. Значення послідовності на виході прямого *P*-блоку змішувача в DES становить $(F6D41A3B)_{16}$. Значення лівої частини вхідної послідовності –

$(56FE87CD)_{16}$. Визначити значення послідовності на виході функції *xor* змішувача.

15. Значення послідовності ключа шифру DES становить $(ABBA08192637CDDC)_{16}$. Визначити значення послідовності після проходження блоку видаленням біт перевірки.

16. Значення послідовності після проходження блоку видаленням біт перевірки становить $(43FA86B1ACEA94)_{16}$. Визначити значення:

а) четвертого раундового ключа;

б) дев’ятого раундового ключа;

в) тринадцятого раундового ключа;

г) шістнадцятого раундового ключа.

РОЗДІЛ 5
РЕЖИМИ РОБОТИ БЛОКОВИХ СИСТЕМ ШИФРУВАННЯ

Незабаром після DES в США був прийнятий ще один федеральний стандарт, що рекомендує чотири режима роботи алгоритму DES для шифрування даних. Відтоді ці режими стали загальноприйнятими і застосовуються з будь-якими блоковими шифрами. Пізніше був доданий ще один режим. Для будь-якого симетричного блокового алгоритму шифрування визначено п’ять режимів роботи [4, 24, 29, 36]:

1. *Режим електронної кодової книги* (*Electronic Code Book – ECB*). У цьому режимі роботи кожний блок незашифрованих даних шифрується незалежно від інших блоків, із застосуванням одного і того ж ключа шифрування. Типове застосування – безпечна передача коротких повідомлень допоміжного характеру (наприклад, сеансові ключі, паролі).

2. *Режим зчеплення блоків шифрованих даних* (*Cipher Block Chaining – CBC*). У цьому режимі роботи входом криптографічного алгоритму є результат застосування операції *xor* до поточного блоку незашифрованих і попереднього блоку зашифрованих даних. Типове застосування – поблочна передача даних загального призначення та аутентифікація.

3. *Режим зворотного зв’язку за зашифрованими даними* (*Cipher Feedback – CFB*). У цьому режимі роботи за кожного виклику алгоритму обробляється l біт вхідного значення. Попередній зашифрований блок використовується в якості входу алгоритму шифрування; до l біт виходу алгоритму і l біт наступного вхідного блоку застосовується операція *xor*, результатом якої є наступний зашифрований блок з l біт. Типове застосування – потокова передача даних загального призначення та аутентифікація.

4. *Режим зворотного зв’язку за виходом* (*Output Feedback – OFB*). Цей режим роботи аналогічний режиму *CFB*, за винятком того, що на вхід алгоритму шифрування наступного блоку подається результат шифрування

попереднього блоку (регістру зсуву); тільки після цього виконується операція *xor* з черговими *l* бітами незашифрованих даних. Типове застосування – потокова передача по каналах з перешкодами (наприклад, супутниковий зв'язок).

5. *Режим лічильника* (*Counter mode – CTR*). Цей режим роботи дуже схожий на режим *OFB*, але замість використання випадкових унікальних значень вектора ініціалізації для генерації значень ключового потоку, цей режим використовує лічильник, зашифроване значення якого додається до кожного блоку відкритого тексту, який потрібно зашифрувати. Унікальне значення лічильника гарантує, що кожний блок об'єднується з унікальним значенням ключового потоку.

5.1. Режим роботи "електронна кодова книга"

Режим електронної кодової книги ECB є найпростішим серед стандартних способів використання блокових шифрів (рис. 5.1).

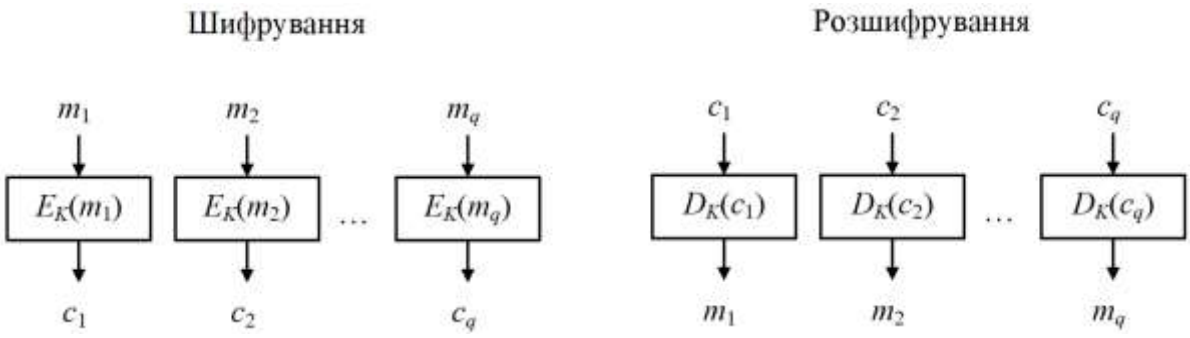


Рис. 5.1. Структура режиму роботи "електронна кодова книга"

Дані, які належить зашифрувати M, поділяються на блоки заданої довжини m_i. Останній з них, при необхідності, доповнюють. Кожний з цих блоків зашифровують незалежно з використанням того самого ключа шифрування

$$c_i = E_K(m_i),$$

де c_i – блоки зашифрованих даних; E – функція шифрування; K – ключ.

Процес розшифрування – просте обернення попередньої операції

$$m_i = D_K(c_i),$$

де D – функція розшифрування.

Основна перевага цього режиму – простота реалізації. Однак з режимом ECB пов'язаний ряд проблем.

Перша проблема виникає через те, що однакові вхідні блоки ($m_i = m_j$) будуть зашифровані у однакові вихідні блоки ($c_i = c_j$). Це, дійсно, проблема, оскільки шаблонні початок і кінець повідомлення збігаються, що дає криптографічному аналітику деяку інформацію про зміст повідомлення. Друга проблема пов'язана з тим, що видалення з повідомлення будь-якого блоку не залишає слідів, і атакуючий може таким чином спотворити інформацію, що передається. Третя проблема подібна до другої, але пов'язана зі вставкою блоків з інших повідомлень.

Щоб краще уявити собі ці проблеми, візьмемо найпростішу модель шифру, в якій блок відповідає слову, і припустимо, що відкрите повідомлення "*Плати Алісі сто гривень, не плати Бобу двісті гривень*" у зашифрованому вигляді виглядає так: "*У кішки чотири ноги, а у людини дві ноги*". Можна тепер змусити отримувача оплатити Алісі дві сотні гривень, замість однієї, відправивши йому повідомлення – "*У кішки дві ноги*", яке отримане з першого заміною одного з блоків на блок з другого повідомлення. Крім того, можна спонукати отримувача зашифрованого повідомлення виплатити Бобу двісті гривень, якщо видалити відповідний блок з зашифрованого повідомлення.

Таким атакам можна протистояти, додаючи контрольні суми декількох блоків відкритих даних або використовуючи режим, за якого до кожного блоку зашифрованих даних додається "контекстний ідентифікатор".

При шифруванні в режимі ECB помилка в одному біті зашифрованих даних, що з'являється на стадії передачі повідомлення, вплине на увесь блок,

в якому вона допущена, і дані цього блоку, звичайно, будуть розшифровані невірно, але це не впливає на інші блоки розшифрованих даних. Однак, якщо біт зашифрованих даних випадково втрачений або доданий, то всі наступні зашифровані дані будуть розшифровані неправильно, якщо для вирівнювання меж блоків не використовується яка-небудь кадрова структура.

Більшість повідомлень точно не ділиться на 64-бітові (або будь-якого іншого розміру, наприклад 128 біт) блоки для шифрування, у кінці зазвичай використовується укорочений блок. Режим ECB вимагає використовувати 64-бітові блоки. Способом вирішення цієї проблеми є доповнення останніх блоків даних до заданої довжини.

Останній блок доповнюється деяким регулярним шаблоном: нулями, одиницями, нулями і одиницями, що чергуються, для отримання повного блоку. За необхідності видалити додавання після розшифрування, в останній байт останнього блоку записується кількість байтів (бітів) додавання. Наприклад, нехай розмір блоку – 64 біта і останній блок складається з трьох байтів (24 біта). Для доповнення блоку до 64 біт потрібно п’ять байтів. Тому необхідно додати чотири байти доповнення (наприклад, нулів) і один байт довжини додавання, який повинен дорівнювати 5. Після розшифрування необхідно взяти останній байт останнього блоку і, визначивши довжину доповнення, видалити з останнього блоку розшифрованих даних останні п’ять байтів. Щоб цей метод працював правильно, кожне повідомлення має бути доповнене, навіть якщо відкриті дані містять ціле число блоків. У такому випадку доведеться додати один повний блок, який буде мати сім байт доповнення та один байт довжини доповнення.

5.2. Режим роботи “зчеплення блоків шифрованих даних”

Одним із шляхів уникнення проблем, що виникають при використанні режиму ECB, є “зчепленні” зашифрованих блоків даних, тобто в додаванні до кожного зашифрованого блоку даних контекстного ідентифікатора.

Найпростіший спосіб зробити це – застосувати режим “зчеплення блоків шифрованих даних” або CBC (рис. 5.2).

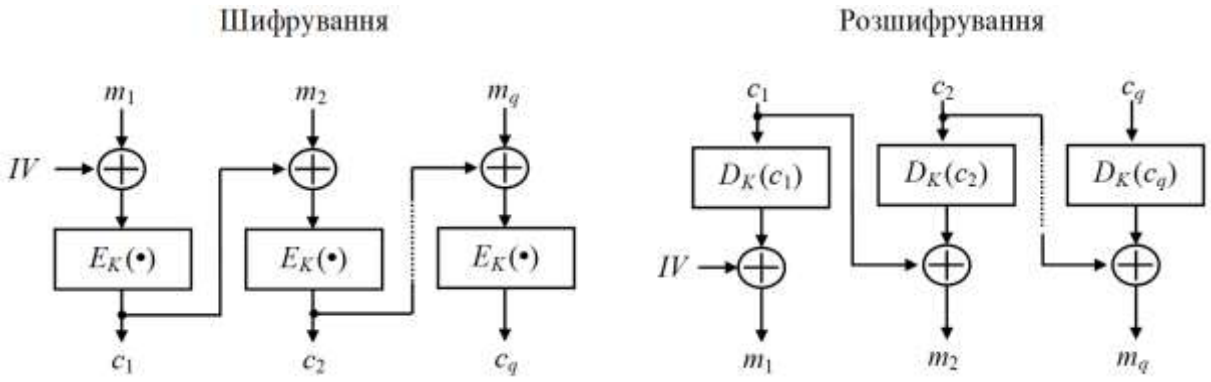


Рис. 5.2. Структура режиму роботи “зчеплення блоків шифрованих даних”

В цьому режимі вихідні дані для шифрування, як зазвичай, розбиваються на серію блоків

$$m_1, m_2, m_3, \ldots, m_q,$$

де q – кількість блоків вихідних даних (з урахуванням доповнення), які підлягають шифруванню.

Як і в попередньому режимі, останньому блоку потрібне доповнення, щоб довжина вихідних даних стала кратна довжині блоку.

Шифрування здійснюється відповідно до виразів:

$$c_1 = E_K(m_1 \oplus IV), \quad c_i = E_K(m_i \oplus c_{i\text{-}1}), \quad i = 2, 3, \ldots, q.$$

У ході шифрування першого блоку бере участь початкова величина IV (*initialization vector – вектор ініціалізації*), яку слід віднести до завдання функції шифрування. Величину IV застосовують для шифрування для того, щоб зашифровані версії однакових частин вихідних даних виглядали по-різному.

Природно величина IV бере участь і при розшифруванні. Цей процес виглядає наступним чином:

$$m_1 = D_K(c_1) \oplus IV, \quad m_i = D_K(c_i) \oplus c_{i\text{-}1}, \quad i = 2, 3, \ldots, q.$$

Очевидно, що останній 64-бітовий блок зашифрованих даних є функцією секретного ключа *K*, початкового значення *IV* і кожного біта вихідних даних незалежно від їх довжини. Цей блок зашифрованих даних називають кодом аутентифікації повідомлення (message authentication code – *MAC*). MAC може бути легко перевірений після розшифрування даних отримувачем, який володіє секретним ключем *K* і початковим значенням *IV*, шляхом повторення процедури, виконаної відправником. Проте, сторонній не може здійснити генерацію MAC, який би отримувач сприйняв як справжній, щоб додати його до хибного повідомлення, або відокремити MAC від істинного повідомлення для використання його зі зміненим або хибним повідомленням.

Перевага цього режиму полягає в тому, що він не дозволяє накопичуватися помилкам під час передачі. Як випливає з виразів для розшифрування, блок m_i, після розшифрування є тільки функцією c_{i-1} і c_i. Тому помилка під час передачі призведе до втрати тільки двох блоків вихідних даних m_{i-1} і m_i, причому, якщо перший блок вихідних даних буде втрачено повністю, то у другому блоці буде спотворений тільки один біт даних з номером позиції, яка відповідає номеру позиції спотвореного біта.

Як випливає з рис. 5.2, для шифрування даних використовується додатково один блок випадкових даних *IV*. Цей блок не має ніякого змістовного значення, він використовується тільки для того, щоб зробити кожне повідомлення унікальним. Хорошим *IV* служить мітка часу або випадкова послідовність бітів.

Із використанням *IV* повідомлення з однаковими вихідними даними в ході шифрування переходять у повідомлення з різними зашифрованими даними. Отже, зловмисник не зможе зробити вставку (видалення) або підміну блоків зашифрованих даних.

Вимога унікальності *IV* для кожного повідомлення, що передається, не є обов’язковою. Крім того, *IV* не потрібно зберігати в секреті, він може передаватися відкрито разом з зашифрованими даними. Нехай повідомлення,

що передається, складається з декількох блоків $m_1, m_2, m_3, \ldots, m_q$. Блок m_1 зашифровується з використанням *IV*; блок m_2 – з використанням зашифрованих даних c_1 в ролі *IV*; блок m_3 – з використанням зашифрованих даних c_2 в ролі *IV*, і так далі. Отже, якщо кількість блоків даних дорівнює q, то q-1 "векторів ініціалізації" відкриті, навіть якщо початковий *IV* зберігається в секреті. Тому причин зберігати в секреті *IV* немає, *IV* – це просто блок-заглушка, його можна вважати нульовим блоком зчеплення m_0.

Доповнення використовується так само, як і в режимі ECB, але в деяких додатках розмір зашифрованих даних повинен точно співпадати з розміром вихідних даних. Наприклад, зашифрований файл повинен зайняти точно той об'єм пам'яті, що і файл відкритих даних. У цьому випадку останній короткий блок доведеться шифрувати інакше. Нехай останній неповний блок складається з l бітів. Зашифрувавши останній повний блок (рис. 5.3) необхідно знову зробити шифрування q-1-го блоку зашифрованих даних. Потім, вибравши з отриманого блоку зашифрованих даних l старших (лівих) бітів, виконати для них і короткого блоку операцію *xor*, створюючи остаточно зашифровані дані останнього блоку.

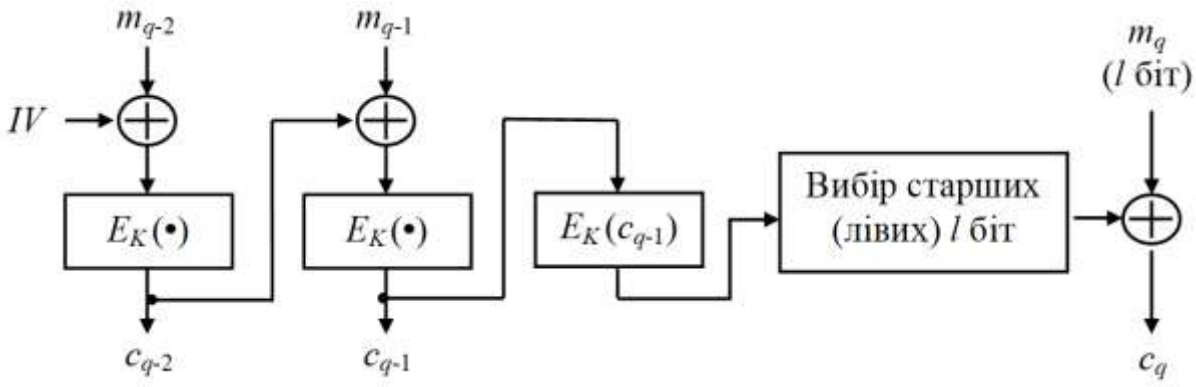


Рис. 5.3. Шифрування короткого останнього блоку в режимі CBC

5.3. Режими роботи зі "зворотним зв'язком"

Режими ECB і CBC призначені для шифрування і розшифрування блоків повідомлень. У режимах ECB і CBC шифрування даних не почнеться,

якщо не буде отримано цілий блок даних. Це створює проблеми для деяких мережевих додатків. Наприклад, у безпечному мережевому середовищі термінал (клієнт) повинен мати можливість передати головному комп'ютеру (серверу) символ відразу, як тільки він буде введений. Якщо дані потрібно обробляти байтами, режими ECB і CBC не будуть задовольняти вимогам.

5.3.1. Режим роботи "зворотній зв'язок за зашифрованими даними"

Рішення полягає в тому, щоб застосувати алгоритми симетричного блокового шифрування або інші в режимі зворотного зв'язку за зашифрованими даними (CFB). У цьому режимі загальний розмір блоку n, але розмір блоку вихідних або зашифрованих даних – l, де $l < n$.

Ідея полягає в тому, що алгоритми симетричного блокового шифрування використовуються не для того, щоб зашифрувати вихідні або розшифрувати зашифровані дані, а для того, щоб зашифрувати або розшифрувати зміст регістра зсуву, розміром n. Шифрування виконується із застосуванням операції *xor* до l-бітового блоку вихідних даних з старшими l-бітами зашифрованого значення регістру зсуву (рис. 5.4).

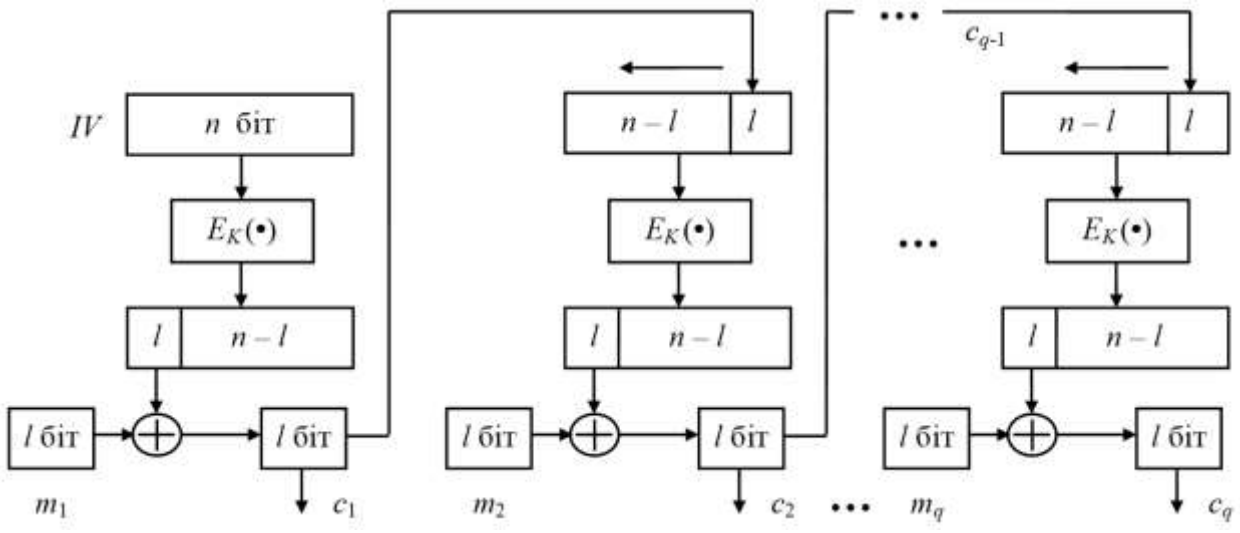


Рис. 5.4. Шифрування у режимі "зворотній зв'язок за зашифрованими даними"

Вхідний блок (регістр зсуву вліво) при шифруванні спочатку містить вектор ініціалізації *IV*, вирівняний по правому краю.

Припустимо, що в результаті розбиття вихідних даних на блоки, отримано q блоків завдовжки l біт кожен. Тоді кожний i-й блок зашифрованих даних буде визначатися наступним чином:

$$c_i = m_i \oplus g_l, \quad i = 1, 2, 3, \ldots, q,$$

де g_l – позначає l старших зашифрованих біт значення регістра зсуву.

Оновлення регістра зсуву (вхідного блоку) здійснюється шляхом відкидання старших l біт і дописування справа c_{i-1}.

Розшифрування виконується із застосуванням операції *xor* до l-бітового блоку зашифрованих даних з старшими l-бітами зашифрованого значення регістру зсуву. Вхідний блок (регістр зсуву вліво) при розшифруванні спочатку містить вектор ініціалізації *IV*, вирівняний по правому краю (рис. 5.5).

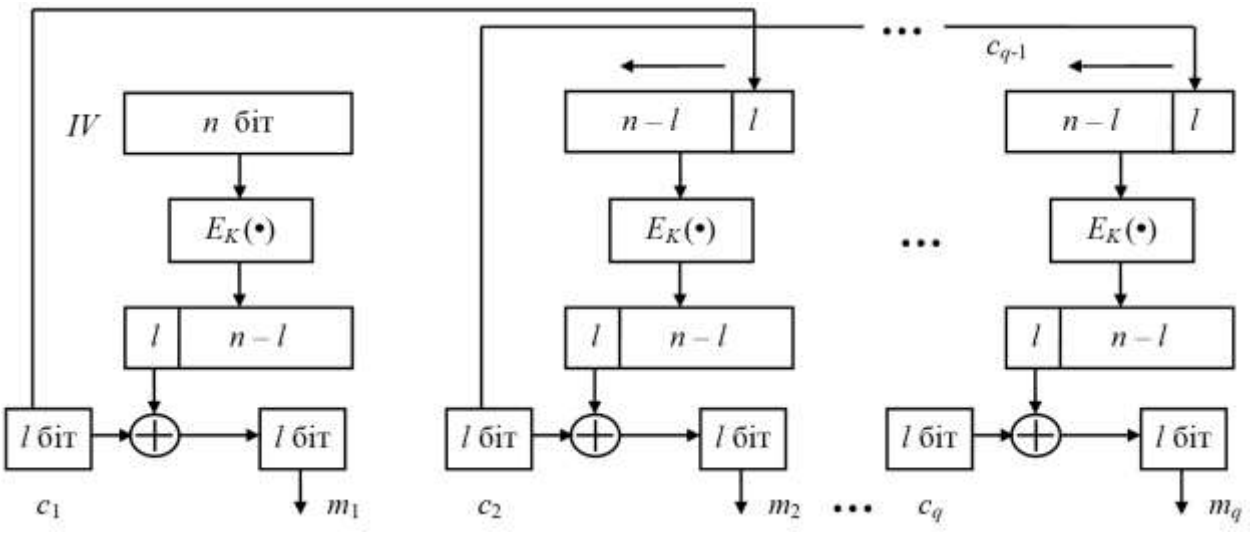


Рис. 5.5. Розшифрування у режимі “зворотній зв’язок за зашифрованими даними ”

Кожний i-й блок розшифрованих даних буде визначатися наступним чином:

$$m_i = c_i \oplus g_l, \quad i = 1, 2, 3, \ldots, q,$$

де g_l – позначає l старших зашифрованих біт значення регістру зсуву.

Для кожного блоку, крім першого, регістр зсуву виконує зсув змісту (попередній регістр зсуву) на *l* біт вліво, заповнюючи праві *l* біт значеннями з c_{i-1}. Вміст регістра зсуву зашифровується і тільки крайні ліві *l* біт обробляються за допомогою *xor* з зашифрованими даними блоку c_i отримуючи m_i.

Цікаво, що в цьому режимі не потрібне доповнення блоків, тому що розмір блоків, *l*, зазвичай вибирається так, щоб задовольнити розміру блоку даних, який потрібно зашифрувати (наприклад, символ). Цікаве також інше – система не повинна чекати отримання великого блока даних (64 біта або 128 біт) для того, щоб почати шифрування. Процес шифрування виконується для маленького блока даних (таких, як символ). Ці дві переваги призводять до двох недоліків. Режим CFB менш ефективний, ніж ECB або CBC, тому що він застосовує шифрування основним блоковим шифром маленького блока розміром *l*.

Як і режим CBC, режим CFB зв’язує разом символи відкритих даних так, що зашифровані дані залежать від усіх попередніх вихідних даних. Як і в режимі CBC *IV* має бути унікальним. Однак треба уникати повторення *IV* для різних повідомлень, які зашифровуються однаковим ключем.

У режимі CFB помилка одного біта у блоці m_i приведе до помилки одного біта у c_i та в середньому у половині біт усіх інших зашифрованих блоках, починаючи з c_{i+1}, але при розшифруванні отримуємо початковий текст з тією самою єдиною помилкою. Набагато цікавіша помилка в зашифрованих даних. Спотворення одного біта у блоці c_i приведе до спотворення одного біта у блоці m_i. Потім помилка потрапляє в регістр зсуву та спотворює в середньому половину біт у кожному з наступних розшифрованих блоків, доки зіпсований біт не вийде з регістра зсуву. В подальшому блоки розшифровуються коректно.

Режим CFB самостійно відновлюється після помилки синхронізації (значення регістру зсуву).

5.3.2. Режим роботи “зворотній зв’язок за виходом”

Режим роботи OFB теж використовує змінний розмір блока і регістр зсуву, що ініціалізується так само, як у режимі CFB, а саме – вхідний блок спочатку містить початкове значення *IV*, вирівняне по правому краю. Шифрування виконується із застосуванням операції *xor* до *l*-бітового блока вихідних даних з старшими *l*-бітами зашифрованого значення регістру зсуву (рис. 5.6).

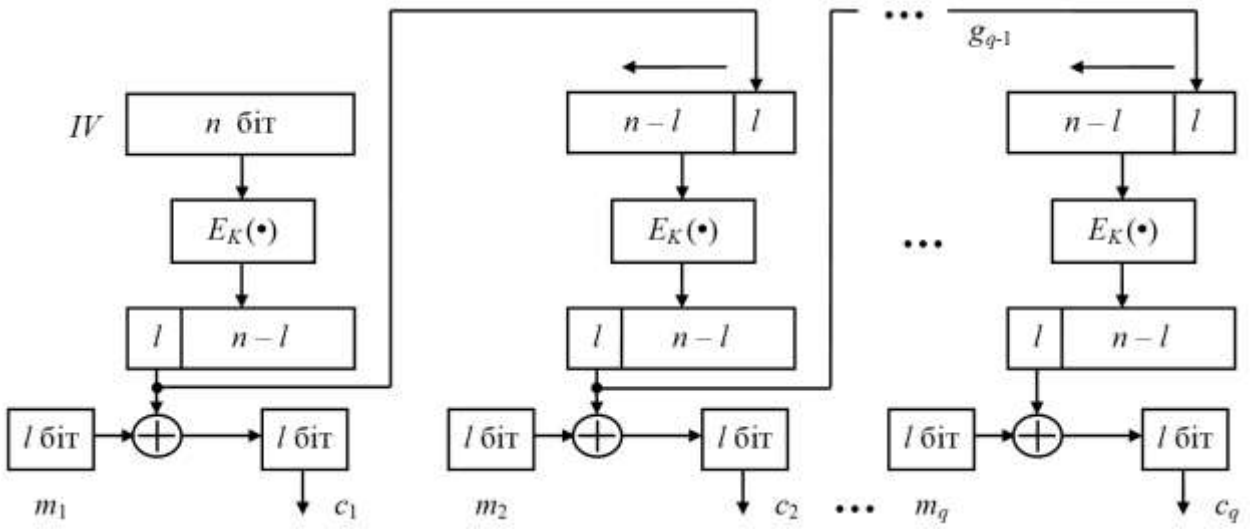


Рис. 5.6. Шифрування у режимі “зворотній зв’язок за виходом”

При цьому для кожного сеансу шифрування даних необхідно використовувати нове початкове значення стану регістра, яке повинне пересилатися по каналу відкритими даними.

Припустимо, що відкрите повідомлення представляється у вигляді блоків: $m_1, m_2, m_3, \ldots, m_q$. Для усіх $i = 1, 2, 3, ..., q$

$$c_i = m_i \oplus g_l, \quad i = 1, 2, 3, \ldots, q,$$

де g_l – позначає l старших зашифрованих біт значення регістру зсуву.

Відмінність від режиму зворотного зв’язку за зашифрованими даними полягає в методі оновлення регістра зсуву (вхідного блоку). Це здійснюється шляхом відкидання старших l біт і дописування справа g_l.

Розшифрування виконується із застосуванням операції *xor* до *l*-бітового блоку зашифрованих даних з старшими *l*-бітами зашифрованого значення регістру зсуву. Вхідний блок (регістр зсуву вліво) при розшифруванні спочатку містить вектор ініціалізації *IV*, вирівняний по правому краю (рис. 5.7).

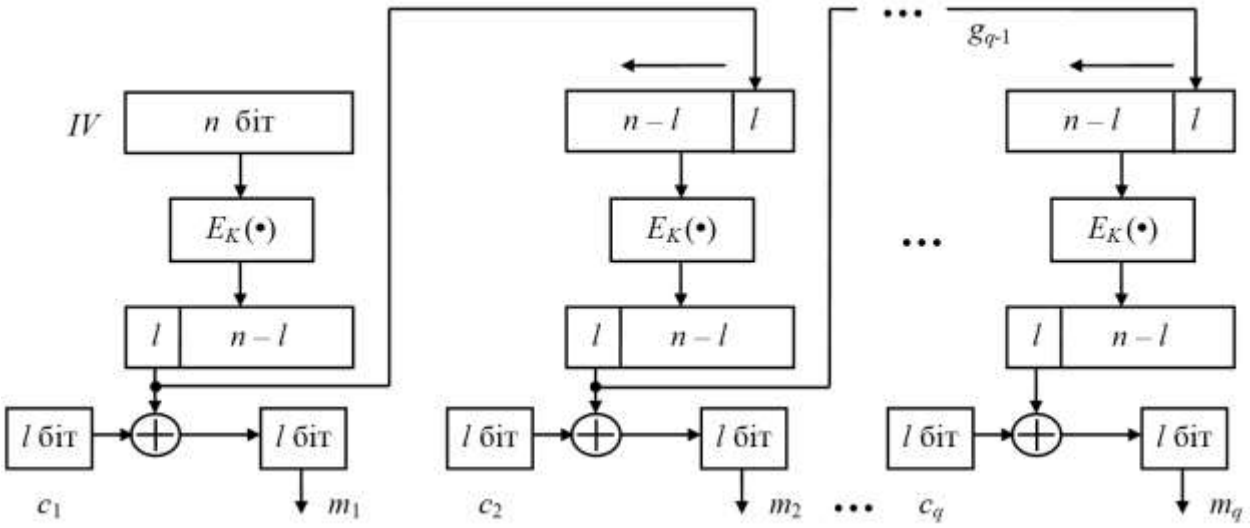


Рис. 5.7. Розшифрування у режимі “зворотній зв’язок за виходом”

Кожний *i*-й блок розшифрованих даних буде визначатися наступним чином:

$$m_i = c_i \oplus g_l, \quad i = 1, 2, 3, \ldots, q,$$

де g_l – позначає *l* старших зашифрованих біт значення регістру зсуву.

Основна перевага режиму OFB полягає в тому, що якщо під час передачі відбулася помилка, то вона не поширюється на наступні зашифровані блоки, і тим самим зберігається можливість розшифрування наступних блоків. Наприклад, якщо відбудеться спотворення одного біта у блоці c_i, то це призведе до спотворення тільки одного біта у блоці m_i. Подальша послідовність блоків буде розшифрована коректно.

При використанні режиму OFB надзвичайно важливо зберегти синхронізацію. Для цього необхідно передбачити засоби контролю над синхронізацією та засоби відновлення у випадку її втрати.

5.4. Режим роботи “лічильник”

У режимі лічильника (CTR) немає інформації зворотного зв’язку. Псевдовипадковий ключовий потік досягається за допомогою лічильника. Лічильник на n біт ініціалізується в заздалегідь визначене значення (IV) і збільшується за основним і заздалегідь визначеним правилом (mod 2^n). Щоб забезпечувати випадковість, величина приросту може залежати від номера блока. Вихідні і зашифровані дані мають той самий розмір блоку, як і основний шифр. Блоки розміру n вихідних даних зашифровані так, щоб створити зашифровані дані з блоком розміру n. На рис. 5.8 показано шифрування даних у режимі лічильника.

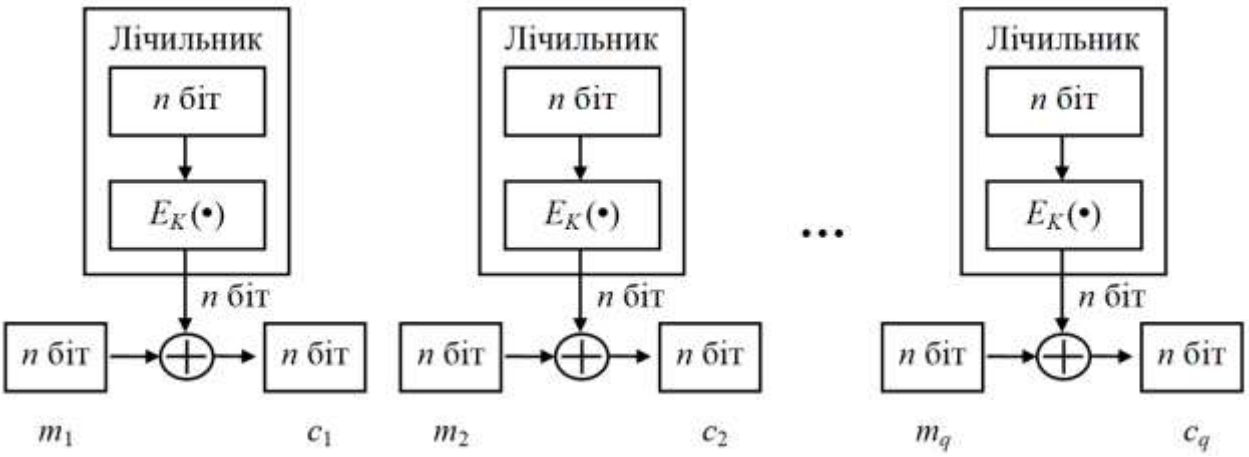


Рис. 5.8. Шифрування даних у режимі роботи “лічильник”

Відношення між вихідними даними і блоками зашифрованих даних показано нижче.

Шифрування:

$$c_i = m_i \oplus E_K(n_i),$$

де n_i – n-бітове значення лічильника.

Розшифрування (рис. 5.9):

$$m_i = c_i \oplus E_K(n_i).$$

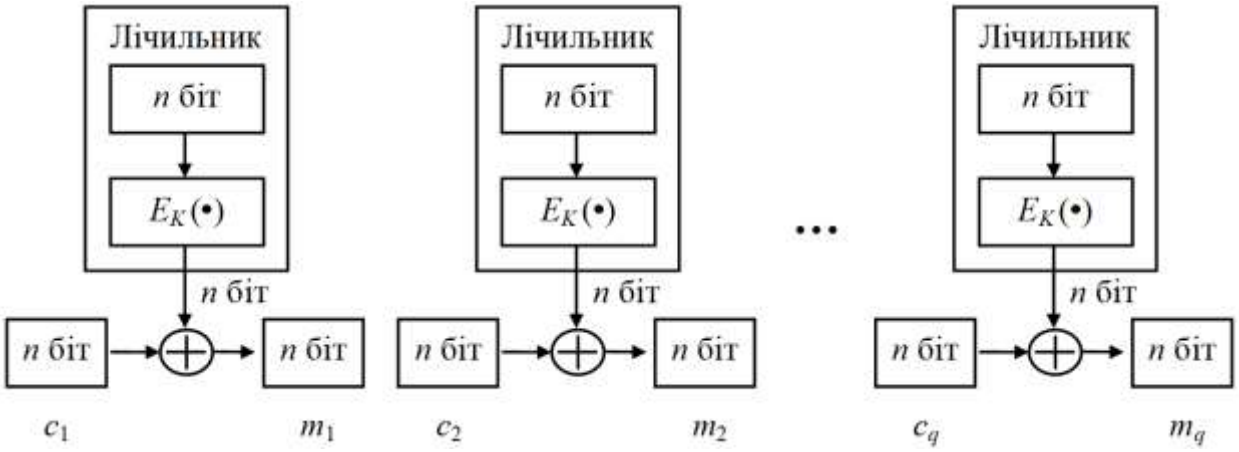


Рис. 5.8. Розшифрування даних у режимі роботи “лічильник”

Режим CTR використовує функцію шифрування основного блокового шифру (E_K) і для шифрування, і для розшифрування. Досить легко довести, що блок m_i вихідних даних може бути відновлений із зашифрованих даних c_i.

Можна порівняти режим CTR з режимами OFB і ECB. Подібно до OFB, CTR створює ключовий потік, який не залежить від попереднього блоку зашифрованих даних, але CTR не використовує інформацію зворотного зв’язку. Так само як ECB, CTR створює n-бітові зашифровані дані, блоки яких незалежні один від одного – вони залежать тільки від значень лічильника. Негативною стороною цієї властивості є те, що режим CTR, подібно до режиму ECB, не може використовуватися для обробки в реальному масштабі часу. Алгоритм шифрування чекає перед шифруванням закінчений n-розрядний блок даних. Позитивною стороною цієї властивості є те, що режим CTR, подібно до режиму ECB, може використовуватися, щоб зашифрувати і розшифрувати файли довільного доступу, і значення лічильника може бути пов’язано з номером запису у файлі.

Проблеми безпеки для режиму CTR ті самі, що і для режиму OFB. Єдина помилка в зашифрованих даних впливає тільки на відповідний біт в розшифрованих даних.

Контрольні питання до розділу 5

1. Поясніть, для чого необхідні режими роботи, якщо для шифрування використовуються сучасні блокові шифри.

2. Перерахуйте основні режими роботи симетричних блокових шифрів.

3. Визначте режим ECB і перерахуйте його переваги і недоліки.

4. Визначте режим CBC і перерахуйте його переваги і недоліки.

5. Визначте режим CFB і перерахуйте його переваги і недоліки.

6. Визначте режим OFB і перерахуйте його переваги і недоліки.

7. Визначте режим CTR і перерахуйте його переваги і недоліки.

8. Розділіть основні режими роботи на дві групи: ті, які використовують функції шифрування і розшифрування, – основні шифри (наприклад, DES), і ті, які використовують тільки функцію шифрування.

9. Розділіть основні режими роботи на дві групи: ті, які вимагають доповнення даних, і ті, які не вимагають цього.

10. Розділіть основні режими роботи на дві групи: ті, які використовують той самий ключ для шифрування всіх блоків, і ті, які використовують ключовий потік для шифрування блоків.

11. Перерахуйте режими роботи, які можуть використовуватися для шифрування файлів довільного доступу.

12. Покажіть, чому режим CFB створює несинхронний шифр потоку, а режим OFB – синхронний.

13. Скільки блоків зачіпає єдиний біт помилки при передачі в режимі CFB?

14. У режимі ECB ($n = 64$) у першому блоці зашифрованих даних спотворений один (9-ий) біт протягом передачі. Визначте можливі спотворені біти в розшифрованих даних.

15. У режимі CBC ($n = 64$) у першому блоці зашифрованих даних спотворений один (9-ий) біт протягом передачі. Визначте можливі спотворені біти в розшифрованих даних.

16. У режимі CFB ($l = 8$, $n = 64$) п’ятий біт першого блока відкритих даних спотворений. Що буде відбуватися при розшифруванні даних?

17. У режимі CFB ($l = 8$, $n = 64$) п’ятий біт першого блока зашифрованих даних спотворений. Визначте можливі спотворені біти в розшифрованих даних.

18. У режимі OFB ($l = 8$, $n = 64$) п’ятий біт першого блока зашифрованих даних спотворений. Визначте можливі спотворені біти в розшифрованих даних.

19. У режимі CTR ($n = 64$) п’ятий біт другого блока зашифрованих даних спотворений. Визначте можливі спотворені біти в розшифрованих даних.

20. Доведіть, що вихідні дані, які використовував відправник, можуть бути відновлені отримувачем у режимі CFB.

21. Доведіть, що вихідні дані, які використовував відправник, можуть бути відновлені отримувачем у режимі OFB.

22. Доведіть, що вихідні дані, які використовував відправник, можуть бути відновлені отримувачем у режимі CTR.

РОЗДІЛ 6
СТАНДАРТ КРИПТОГРАФІЧНОГО ПЕРЕТВОРЕННЯ ДАНИХ ДСТУ ГОСТ 28147:2009

6.1. Загальні положення ДСТУ ГОСТ 28147:2009

Важливим завданням у забезпеченні гарантованої безпеки інформації в інформаційних системах є розробка і використання стандартних алгоритмів шифрування даних. Першим серед подібних стандартів, як зазначалося в розділі 4, став американський стандарт DES, що представляє собою послідовне використання замін і перестановок. На сьогодні все частіше говорять про невиправдану складність і невисоку криптографічну стійкість DES. На практиці доводиться використовувати його модифікації.

Ефективнішим є стандарт криптографічного перетворення даних ДСТУ ГОСТ 28147:2009. Він рекомендований до використання для захисту будь-яких даних, представлених у вигляді двійкового коду, хоча не виключаються й інші методи шифрування. Даний стандарт формувався з урахуванням світового досвіду, і зокрема, були прийняті до уваги недоліки і нереалізовані можливості алгоритму DES.

Цей стандарт закріплений ГОСТ 28147-89, прийнятим, як виявляється з його позначення, ще у 1989 р. у СРСР і введений у дію з 1 липня 1990 р. Проте, без сумніву, історія цього шифру набагато давніша. Стандарт народився ймовірно в надрах Восьмого головного управління комітету Державної безпеки СРСР, перетвореного тепер у Федеральне агентство урядового зв’язку та інформації. Ще в 70-х рр. XX ст. він мав гриф “цілком таємно”, пізніше гриф був змінений на “таємно”, потім знятий зовсім. На жаль, на відміну від самого стандарту, історія його створення й критерії проектування шифру досі залишаються таємницею.

Опис стандарту шифрування Російської Федерації міститься в документі ГОСТ 28147-89 “Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования”, який з однойменною назвою прийнятий Державним комітетом України з питань технічного регулювання та споживчої політики як ДСТУ ГОСТ 28147:2009 [17, 33]. Те, що в його назві замість терміну “шифрування” фігурує загальніше поняття “криптографічне перетворення”, зовсім не випадково. Окрім декількох тісно пов’язаних між собою процедур шифрування, в документі описано побудований на загальних принципах з ними алгоритм вироблення імітовставки, що є криптографічною контрольною комбінацією, тобто кодом, що виробляється з вихідних даних із використанням секретного ключа з метою імітозахисту, або захисту даних від внесення в них несанкціонованих змін.

Стандарт криптографічного перетворення даних ДСТУ ГОСТ 28147:2009 оперує з блоками даних довжиною 64 біта. Шифрування і розшифрування даних здійснюється з використанням 32 або 16 раундів (циклів). Ключ шифру 256 біт, з якого формуються вісім підключів довжиною по 32 біта (з цих восьми підключів формуються 32 раундових (циклових) підключів).

Стандарт передбачає чотири режими роботи:

– шифрування (розшифрування) даних у режимі простої заміни;

– шифрування (розшифрування) даних у режимі гамування;

– шифрування (розшифрування) даних у режимі гамування зі зворотним зв’язком;

– шифрування з метою вироблення імітовставки.

6.2. Режим простої заміни

Структура ДСТУ ГОСТ 28147:2009 у режимі простої заміни

Для реалізації алгоритму шифрування даних у режимі простої заміни (аналог режиму – “Електронна кодова книга”) використовується тільки частина блоків загальної криптографічної системи (рис. 6.1).

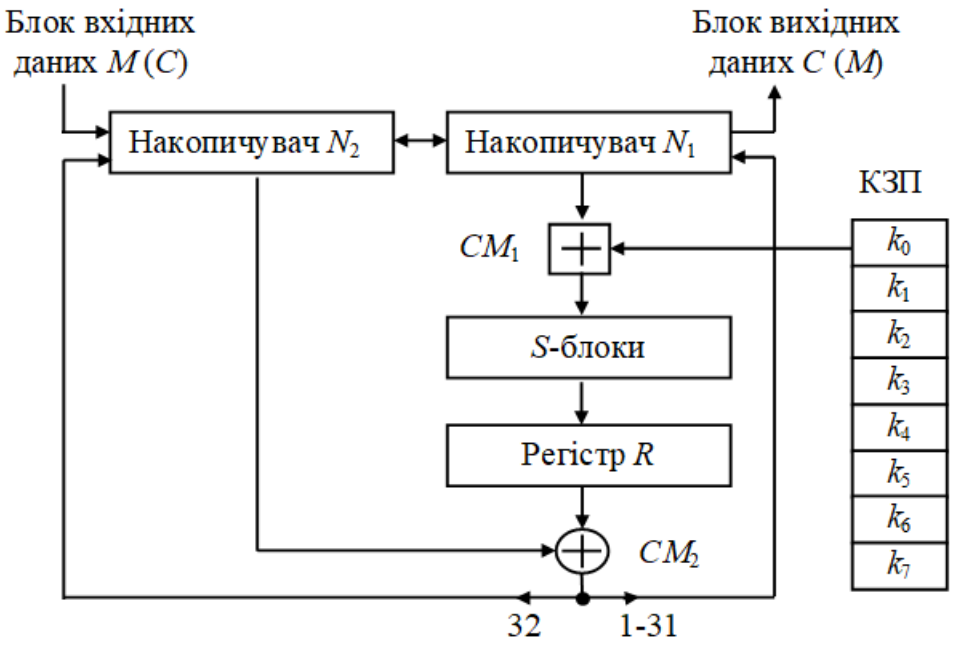


Рис. 6.1. Структура реалізації режиму простої заміни ДСТУ ГОСТ 28147:2009

Позначення на рисунку:

– N_1 і N_2 – 32-розрядні накопичувачі (регістри зсуву);

– CM_1 – 32-розрядний суматор за модулем 2^{32} ($\boxplus$);

– CM_2 – 32-розрядний суматор за модулем 2 ($\oplus$);

– R – 32-розрядний регістр циклічного зсуву даних на 11 розрядів вліво (у бік старших розрядів);

– КЗП – ключовий запам’ятовуючий пристрій на 256 біт, що складається з восьми 32-розрядних накопичувачів k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;

– S-блок заміни, що складається з восьми вузлів заміни: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

Шифрування даних ДСТУ ГОСТ 28147:2009 у режимі простої заміни

Відкриті дані, що підлягають шифруванню, розбивають на 64-розрядні блоки. Процедура шифрування 64-розрядного блоку M у режимі простої заміни включає 32 цикли (раунди). У ключовий запам'ятовуючий пристрій вводять 256 біт ключа шифру K у вигляді восьми 32-розрядних підключів k_i:

$$K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7).$$

Послідовність біт блоку відкритих даних

$$M = (a_1(0), a_2(0), ..., a_{32}(0), b_1(0), b_2(0), ..., b_{32}(0))$$

розбивають на дві послідовності по 32 біта (взяті у зворотному порядку):

$$b(0) = b_{32}(0), b_{31}(0), ..., b_1(0) \text{ і } a(0) = a_{32}(0), a_{31}(0), ..., a_1(0),$$

де b (0) – ліві або старші біти, а a (0) – праві або молодші біти.

При позначенні послідовностей $b(0)$ і $a(0)$ індекс у нижній частині визначає номер біта.

Ці послідовності вводять у накопичувачі N_1 і N_2 перед початком першого циклу шифрування. В результаті початкове заповнення накопичувача N_1 є вектор $a(0)$

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), ..., a_1(0)),$$

а початкове заповнення накопичувача N_2 є вектор b (0)

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), ..., b_1(0)).$$

Перший цикл процедури шифрування 64-розрядного блока відкритих даних можна описати рівняннями:

$$\begin{cases} a\,(1) = f\,(a\,(0 \boxplus k_0) \oplus b\,(0); \\ b\,(1) = a\,(0), \end{cases}$$

де $a\,(1)$ – заповнення N_1, після першого циклу шифрування;

$b\,(1)$ – заповнення N_2 після першого циклу шифрування;

$f\,(\bullet)$ – функція шифрування.

Аргументом функції $f\,(\bullet)$ є додавання за модулем 2^{32} числа $a\,(0)$ (початкового заповнення накопичувача N_1) і числа k_0 – підключа з накопичувача КЗП. Кожне з цих чисел дорівнює 32 бітам.

Наприклад, $a(0) = (1010\ 0100\ 0100\ 1101\ 1100\ 1011\ 0001\ 0101)_2$,

а $k_0 = (1001\ 0100\ 1000\ 1001\ 1000\ 0101\ 0010\ 1001)_2$.

Після виконання операції додавання за модулем 2^{32} результат на виході суматора CM_1 буде дорівнювати

$$a(0) \boxplus k_0 = (0011\ 1000\ 1101\ 0111\ 0101\ 0000\ 0011\ 1110)_2.$$

Функція $f(\bullet)$ включає дві операції над отриманою 32-розрядною сумою.

Перша операція називається підстановкою (заміною) і виконується *S*-блоком підстановки. *S*-блок підстановки, як було вище сказано, складається з восьми вузлів заміни з пам’яттю 64 біта кожний. 32-розрядний вектор з виходу CM_1 послідовно розбивають на вісім 4-розрядних векторів, кожний з яких перетворюється в 4-розрядний вектор відповідним вузлом заміни. Кожний вузол заміни можна представити у вигляді таблиці-перестановки шістнадцяти 4-розрядних двійкових чисел у діапазоні: $(0000, 0001, ..., 1111)_2$. Вхідний вектор вказує адресу стовпця в таблиці, а число в цьому стовпці є вихідним вектором. Потім 4-розрядні вихідні вектори послідовно об’єднують у 32-розрядний вектор. Вузли заміни представляють собою ключові елементи, які є загальними для мережі ЕОМ і рідко змінюються. Ці вузли заміни повинні зберігатися у секреті. У табл. 6.1 наведено приклад вузла заміни для *S*-блоків, який визначений для цілей тестування.

Таблиця 6.1

Таблиця заміни *S*-блоків

		Значення 4-розрядних двійкових векторів															
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Номери *S*-блоків	00	04	10	09	02	13	08	00	14	06	11	01	12	07	15	05	03
	01	14	11	04	12	06	13	15	10	02	03	08	01	00	07	05	09
	02	05	08	01	13	10	03	04	02	14	15	12	07	06	00	09	11
	03	07	13	10	01	00	08	09	15	14	04	06	12	11	02	05	03
	04	06	12	07	01	05	15	13	08	04	10	09	14	00	03	11	02
	05	04	11	10	00	07	02	01	13	03	06	08	05	09	12	15	14
	06	13	11	04	01	03	15	05	09	00	10	14	07	06	08	02	12
	07	01	15	13	00	05	07	10	04	09	02	03	14	06	11	08	12

Наприклад, якщо після виконання операції додавання за модулем 2^{32} результат на виході суматора CM_1 дорівнює

$$a(0) \boxplus k_0 = (0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111)_2,$$

то, з урахуванням табл. 6.1, буде сформовано значення

$$S\,(a\,(0) \boxplus k_0) = (0001\ 1011\ 1010\ 0001\ 0000\ 0011\ 1111\ 1110)_2.$$

Друга операція – циклічний зсув вліво на 11 розрядів 32-розрядного вектора, отриманого з виходу *S*-блока заміни. Циклічний зсув виконується регістром зсуву *R*.

Далі результат роботи функції шифрування $f(\bullet)$ додається порозрядно за модулем 2 в суматорі CM_2 з 32-розрядним початковим заповненням $b(0)$ накопичувача N_2. Потім значення N_1 переписують у накопичувач N_2 (значення $b(1) = a(0)$), а отриманий на виході CM_2 результат (значення $a(1)$) записують у накопичувач N_1. Після цього перший цикл завершений.

Наступні цикли здійснюються аналогічно, при цьому у другому циклі з КЗП зчитують підключ k_1, у третьому циклі – k_2, і т.д., у восьмому циклі – k_7. У циклах з 9-го по 16-й, а також у циклах з 17-го по 24-й підключи з КЗП зчитуються в тому самому порядку:

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7.$$

В останніх восьми циклах (з 25-го по 32-й) порядок зчитування підключів з КЗП зворотний: k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0. Таким чином, при шифруванні в 32 циклах здійснюється наступний порядок вибірки з КЗП підключей:

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7,$$
$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0.$$

Після завершення 32-го циклу результат з суматора CM_2 записується в накопичувач N_2, а в накопичувачі N_1 зберігається попереднє заповнення. Отримані після 32-го циклу шифрування заповнення накопичувачів N_2 і N_1 є блоком зашифрованих даних C, що відповідає блоку відкритих даних M.

Блок зашифрованих даних C (64 розряди) виводиться з накопичувачів N_2 і N_1 у наступному порядку: з розрядів 1 ... 32 накопичувача N_1, потім з розрядів 1 ... 32 накопичувача N_2, тобто починаючи з молодших розрядів:

$$C = (a_1(32), a_2(32), ..., a_{32}(32), b_1(32), b_2(32), ..., b_{32}(32)).$$

Решта блоків відкритих даних зашифровуються в режимі простої заміни аналогічно.

Структурна схема основного кроку криптографічного перетворення стандарту приведена на рис. 6.2.

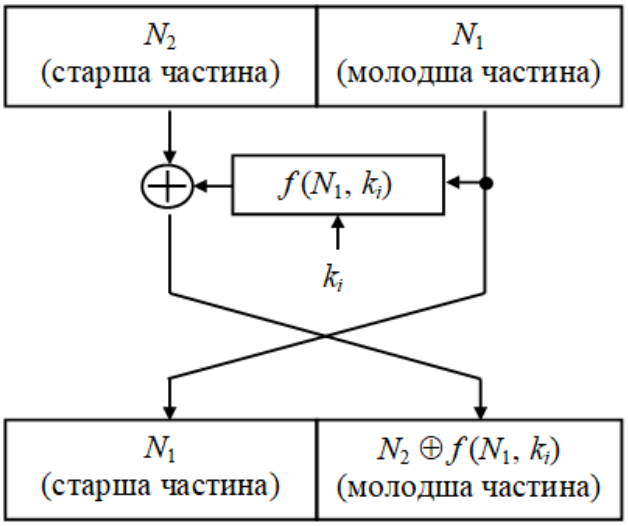


Рис. 6.2. Структурна схема основного кроку криптографічного перетворення стандарту

Розшифрування даних ДСТУ ГОСТ 28147:2009 у режимі простої заміни

Криптографічний схема, що реалізує алгоритм розшифрування даних у режимі простої заміни, має той самий вигляд, що і при шифруванні (див. рис. 6.1).

Розшифрування даних у режимі простої заміни здійснюється за тим самим алгоритмом, що і шифрування, з тією зміною, що заповнення накопичувачів зчитуються з КЗП у циклах розшифрування в наступному порядку (оберненим у відношенні до шифрування):

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0,$$
$$k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0.$$

Отримані після 32 циклів роботи заповнення накопичувачів N_1 і N_2 утворюють блок розшифрованих (відкритих) даних

$$M = (a_1(0), a_2(0), ..., a_{32}(0), b_1(0), b_2(0), ..., b_{32}(0)).$$

При цьому стан накопичувача N_1:

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), ..., a_1(0)),$$

а стан накопичувача N_2:

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), ..., b_1(0)).$$

Аналогічно розшифровуються інші блоки зашифрованих даних.

Слід мати на увазі, що режим простої заміни допустимо використовувати для шифрування даних тільки в обмежених випадках – при виробленні ключа шифру і шифруванні його із забезпеченням імітозахисту для передачі по каналам зв’язку або для зберігання в пам’яті ЕОМ.

6.3. Режим гамування

Структура ДСТУ ГОСТ 28147:2009 у режимі гамування

Для реалізації алгоритму шифрування даних у режимі гамування (аналог “Режиму зворотного зв’язку за виходом”) використовуються повністю всі блоки загальної криптографічної системи (рис. 6.3).

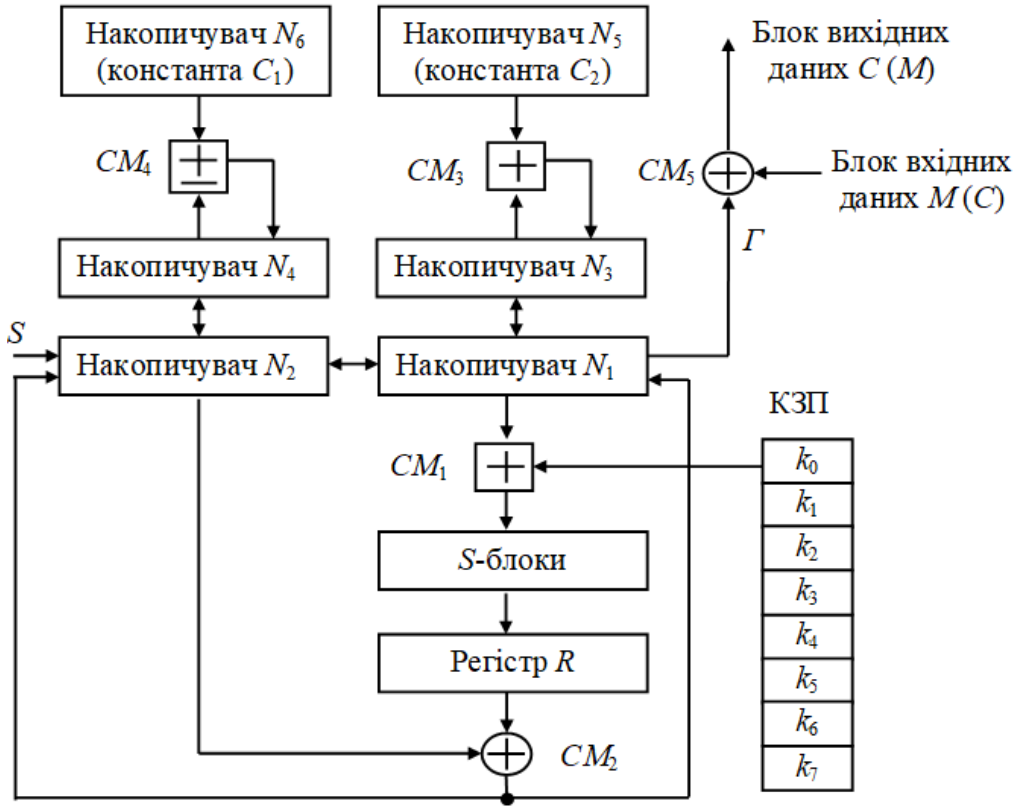


Рис. 6.3. Структура реалізації режиму гамування ДСТУ ГОСТ 28147:2009

Позначення на рисунку:

– N_1 – N_6 – 32-розрядні накопичувачі;

– CM_1 і CM_3 – 32-розрядні суматори за модулем 2^{32} ([+]);

– CM_2 і CM_5 – 32-розрядні суматори за модулем 2 ($\oplus$);

– CM_4 – 32-розрядний суматор за модулем 2^{32}-1 ([±]);

– R – 32-розрядний регістр циклічного зсуву даних на 11 розрядів вліво (у бік старших розрядів);

– КЗП – ключовий запам’ятовуючий пристрій на 256 біт, що

складається з восьми 32-розрядних накопичувачів k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;

– *S*-блок заміни, що складається з восьми вузлів заміни: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

Шифрування даних ДСТУ ГОСТ 28147:2009 у режимі гамування

Відкриті дані розбивають на 64-розрядні блоки

$$M = (M^{(1)}, M^{(2)}, \ldots, M^{(q)}),$$

де $M^{(i)}$ – *i*-й 64-розрядний блок відкритих даних ($i = 1, 2, ..., q$, а q визначається обсягом даних, що шифруються).

Ці блоки по черзі зашифровуються у режимі гамування шляхом порозрядного додавання за модулем 2 в суматорі CM_5 з гамою шифру Γ, яка виробляється блоками по 64 біта, тобто

$$\Gamma = (\Gamma^{(1)}, \Gamma^{(2)}, \ldots, \Gamma^{(q)}),$$

де $\Gamma^{(i)}$ – *i*-й 64-розрядний блок гами ($i = 1, 2, ..., q$, а q визначається обсягом даних, що шифруються).

Число двійкових розрядів у блоці $M^{(q)}$ може бути менше 64, при цьому невикористана для шифрування частина гами шифру з блоку $\Gamma^{(q)}$ відкидається.

Рівняння шифрування даних у режимі гамування має вигляд

$$C^{(i)} = M^{(i)} \oplus \Gamma^{(i)}$$

де $\Gamma^{(i)} = A\,(Y_{i-1} \boxplus C_2,\ Z_{i-1} \boxplus C_1)$, $i = 1, 2, ..., q$;

$C^{(i)}$ – *i*-й 64-розрядний блок зашифрованих даних;

$A(\bullet)$ – функція шифрування у режимі простої заміни;

C_1 і C_2 – 32-розрядні двійкові константи;

Y_i і Z_i – 32-розрядні двійкові послідовності, які визначаються ітераційно в міру формування гами $\Gamma^{(i)}$.

Спочатку формуються вихідні значення Y_i та Z_i наступним чином:

$$(Y_0, Z_0) = A\,(S),$$

де S – 64-розрядна двійкова послідовність (синхропосилка).

Потім ітераційно

$$(Y_i, Z_i) = A\,(Y_{i-1} \boxplus C_2,\ Z_{i-1} \boxplus C_1),\ i = 1, 2, ..., q.$$

Розглянемо реалізацію процедури шифрування у режимі гамування. У накопичувачі N_6 і N_5 заздалегідь записані 32-розрядні двійкові константи C_1 і C_2, що мають наступні значення:

$$C_1 = (01010104)_{16} = (0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0100)_2;$$
$$C_2 = (01010101)_{16} = (0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0001)_2.$$

У КЗП вводиться 256 бітів ключа; у накопичувачі N_1 і N_2 – 64-розрядна двійкова послідовність (синхропосилка)

$$S = (s_1, s_2, \ldots, s_{64}),$$

де s_i – i-й двійковий розряд синхропосилки.

Синхропосилка S є початковим заповненням накопичувачів N_1 і N_2 для послідовного вироблення q блоків гами шифру. Початкове заповнення накопичувача N_1: $N_1 = (s_{32}, s_{31}, ..., s_1)$, а початкове заповнення накопичувача N_2: $N_2 = (s_{64}, s_{63}, ..., s_{33})$.

Заповнення накопичувача N_4 додають за модулем 2^{32}-1 у суматорі CM_4 з 32-розрядної константою C_1 з накопичувача N_6. Додавання за модулем 2^{32}-1, по суті, є адитивною інверсією результату додавання. Результат записується в N_4. Заповнення накопичувача N_3 додають за модулем 2^{32} у суматорі CM_3 з 32-х розрядної константою C_2 з накопичувача N_5. Результат записується в N_3. Заповнення N_3 переписують в N_1, а заповнення N_4 – в N_2, при цьому заповнення N_3 і N_4 зберігаються. Заповнення накопичувачів N_1 і N_2 шифруються в режимі простої заміни.

Отримане в результаті шифрування заповнення накопичувачів N_1 і N_2 утворюють перший 64-розрядний блок гами шифру

$$\Gamma^{(1)} = (\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}, \ldots, \gamma_{63}^{(1)}, \gamma_{64}^{(1)}),$$

де $\gamma_i^{(1)}$ ($i = 1, 2, \ldots, 64$) – i-й двійковий розряд послідовності гами.

Цей блок гами ($\Gamma^{(1)}$) додається порозрядно за модулем 2 у суматорі CM_5 з першим 64-розрядним блоком відкритих даних

$$M^{(1)} = (m_1^{(1)}, m_2^{(1)}, m_3^{(1)}, \ldots, m_{63}^{(1)}, m_{64}^{(1)}),$$

де $m_i^{(1)}$ ($i = 1, 2, \ldots, 64$) – i-й двійковий розряд першого блоку відкритих даних.

В результаті додавання за модулем 2 значень $M^{(1)}$ та $\Gamma^{(1)}$ отримують перший 64-розрядний блок зашифрованих даних:

$$C^{(1)} = M^{(1)} \oplus \Gamma^{(1)} = (c_1^{(1)}, c_2^{(1)}, c_3^{(1)}, \ldots, c_{63}^{(1)}, c_{64}^{(1)}),$$

де $c_i^{(1)} = m_i^{(1)} \oplus \gamma_i^{(1)}$ ($i = 1, 2, \ldots, 64$) – i-й двійковий розряд першого блоку зашифрованих даних.

Для отримання наступного 64-розрядного блоку гами шифру $\Gamma^{(2)}$ заповнення N_4 додають за модулем 2^{32}-1 у суматорі CM_4 з 32-розрядної константою C_1 з накопичувача N_6. Результат записується в N_4. Заповнення накопичувача N_3 додають за модулем 2^{32} у суматорі CM_3 з 32-розрядної константою C_2 з накопичувача N_5. Результат записується в N_3. Нове заповнення N_3 переписують у N_1, а нове заповнення N_4 – у N_2, при цьому заповнення N_3 і N_4 зберігаються. Заповнення накопичувачів N_1 і N_2 шифруються в режимі простої заміни.

Отримане в результаті шифрування заповнення накопичувачів N_1 і N_2 утворюють другий 64-розрядний блок гами шифру $\Gamma^{(2)}$, який додається порозрядно за модулем 2 у суматорі $CM5$ з другим блоком відкритих даних $M^{(2)}$:

$$C^{(2)} = M^{(2)} \oplus \Gamma^{(2)} = (c_1^{(2)}, c_2^{(2)}, c_3^{(2)}, \ldots, c_{63}^{(2)}, c_{64}^{(2)}).$$

Аналогічно виробляються наступні блоки гами шифру $\Gamma^{(3)}, \Gamma^{(4)}, \ldots, \Gamma^{(q)}$ і зашифровуються наступні блоки відкритих даних $M^{(3)}, M^{(4)}, \ldots, M^{(q)}$.

У канал зв'язку або пам'ять ЕОМ передаються синхропосилка S і блоки зашифрованих даних $C^{(1)}, C^{(2)}, \ldots, C^{(q)}$.

Розшифрування даних ДСТУ ГОСТ 28147:2009 у режимі гамування

При розшифруванні даних криптографічна схема має той самий вигляд, що і при шифруванні (див. рис. 6.3).

Рівняння розшифрування:

$$M^{(i)} = C^{(i)} \oplus \Gamma^{(i)} = C^{(i)} \oplus A\,(Y_{i-1} \boxplus C_2,\ Z_{i-1} \boxplus C_1),\ \ i = 1, 2, ..., q.$$

Слід зазначити, що розшифрування даних можливе тільки за наявності синхропосилки, яка не є секретним елементом шифру і може зберігатися в пам'яті ЕОМ або передаватися по каналах зв'язку разом із зашифрованими даними.

Розглянемо реалізацію процедури розшифрування. У КЗП вводять 256 бітів ключа, за допомогою якого здійснювалось шифрування даних $M^{(1)}, M^{(2)}, \ldots, M^{(q)}$. У накопичувачі N_1 і N_2 вводиться синхропосилка, і здійснюється процес вироблення q блоків гами шифру

$$\Gamma^{(1)}, \Gamma^{(2)}, \ldots, \Gamma^{(q)}.$$

Блоки зашифрованих даних $C^{(1)}, C^{(2)}, \ldots, C^{(q)}$ додаються порозрядно за модулем 2 у суматорі CM_5 із блоками гами шифру $\Gamma^{(1)}, \Gamma^{(2)}, \ldots, \Gamma^{(q)}$. У результаті отримуємо блоки відкритих даних $M = (M^{(1)}, M^{(2)}, \ldots, M^{(q)})$, при цьому, $M^{(q)}$ може містити менше 64 розрядів.

6.4. Режим гамування зі зворотним зв’язком

Структура ДСТУ ГОСТ 28147:2009 у режимі гамування зі зворотним зв’язком

Для реалізації алгоритму шифрування даних у режимі гамування зі зворотним зв’язком (аналог “Режиму зворотного зв’язку за зашифрованими даними”) використовуються тільки частина блоків загальної криптографічної системи (рис. 6.4).

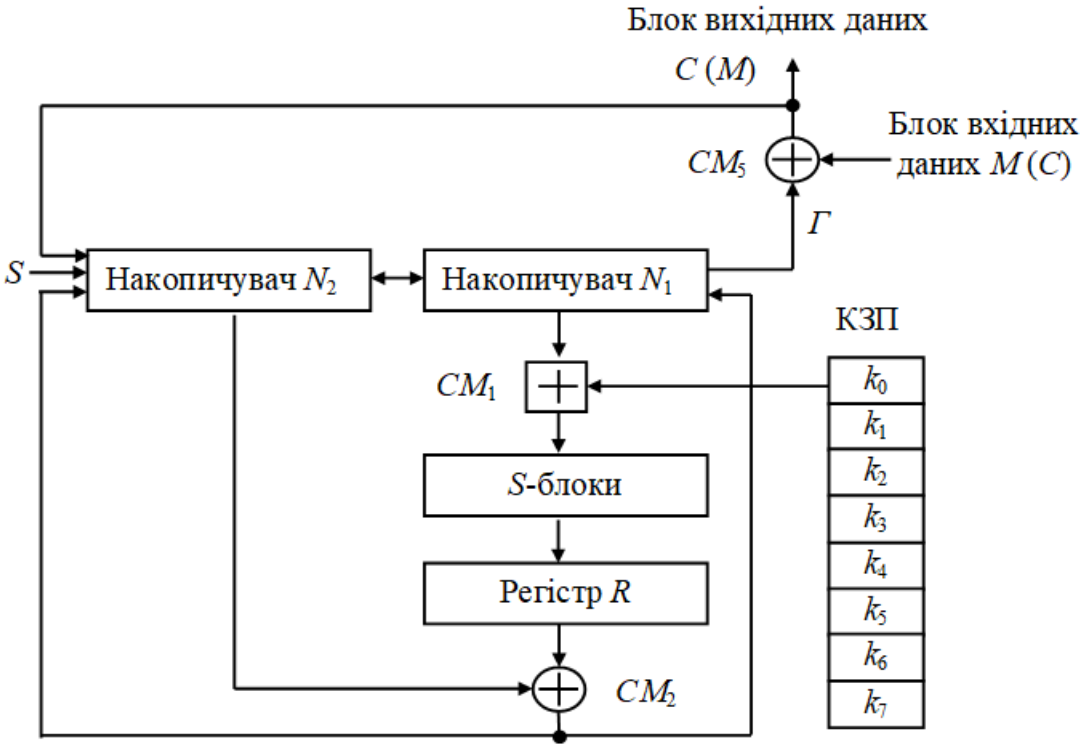


Рис. 6.4. Структура реалізації режиму гамування зі зворотним зв’язком ДСТУ ГОСТ 28147:2009

Позначення на рисунку:

– N_1, N_2 – 32-розрядні накопичувачі;

– CM_1 – 32-розрядний суматор за модулем 2^{32} ($\boxplus$);

– CM_2 і CM_5 – 32-розрядні суматори за модулем 2 ($\oplus$);

– R – 32-розрядний регістр циклічного зсуву даних на 11 розрядів вліво (у бік старших розрядів);

– КЗП – ключовий запам’ятовуючий пристрій на 256 біт, що складається з восьми 32-розрядних накопичувачів k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;

– S-блок заміни, що складається з восьми вузлів заміни: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

Шифрування даних ДСТУ ГОСТ 28147:2009 у режимі гамування зі зворотним зв’язком

Відкриті дані, розбиті на 64-розрядні блоки $M = (M^{(1)}, M^{(2)}, \ldots, M^{(q)})$, зашифровуються у режимі гамування зі зворотним зв’язком шляхом порозрядного додавання за модулем 2 в суматорі CM_5 з гамою шифру $Г$, яка виробляється блоками по 64 біта, тобто

$$Г = (Г^{(1)}, Г^{(2)}, \ldots, Г^{(q)}).$$

Число двійкових розрядів у блоці $M^{(q)}$ може бути меншою за 64, при цьому невикористана для шифрування частина гами шифру з блоку $Г^{(q)}$ відкидається.

Рівняння шифрування у режимі гамування зі зворотним зв’язком мають вигляд:

$$C^{(1)} = Г^{(1)} \oplus M^{(1)} = A\,(S) \oplus M^{(1)};$$
$$C^{(i)} = A\,(C^{(i-1)}) \oplus M^{(i)},\ \ i = 2, 3, \ldots, q.$$

де $C^{(i)}$ – i-й 64-розрядний блок зашифрованих даних;

$A(\bullet)$ – функція шифрування у режимі простої заміни.

Аргументом функції $A(\bullet)$ на першому кроці ітеративного алгоритму є 64-розрядна синхропосилка S, а на усіх наступних кроках – попередній блок зашифрованих даних $C^{(i-1)}$.

Процедура шифрування даних у режимі гамування зі зворотним зв’язком реалізується такий чином. У КЗП вводяться 256 бітів ключа. У накопичувачі N_1 і N_2 вводиться синхропосилка $S = (s_1, s_2, \ldots, s_{64})$, що складається з 64 біт.

Початкове заповнення накопичувачів N_1 і N_2 (синхропосилка) зашифровується у режимі простої заміни. Отримане в результаті шифрування заповнення накопичувачів N_1 і N_2 утворює перший 64-розрядний блок гами шифру

$$\Gamma^{(1)} = A\,(S),$$

який додається порозрядно за модулем 2 в суматорі CM_5 з першим 64-розрядним блоком відкритих даних

$$M^{(1)} = (m_1^{(1)}, m_2^{(1)}, m_3^{(1)}, \ldots, m_{63}^{(1)}, m_{64}^{(1)}).$$

В результаті отримують перший 64-розрядний блок зашифрованих даних:

$$C^{(1)} = M^{(1)} \oplus \Gamma^{(1)} = (c_1^{(1)}, c_2^{(1)}, c_3^{(1)}, \ldots, c_{63}^{(1)}, c_{64}^{(1)}).$$

Блок зашифрованих даних $C^{(1)}$ одночасно є також початковим станом накопичувачів N_1 і N_2 для вироблення другого блоку гами шифру $\Gamma^{(2)}$, і тому за зворотним зв'язком $C^{(1)}$ записується у вказані накопичувачі N_1 і N_2.

Заповнення накопичувача N_1:

$$N_1 = (c_{32}^{(1)}, c_{31}^{(1)}, c_{30}^{(1)}, \ldots, c_2^{(1)}, c_1^{(1)}),$$

а заповнення накопичувача N_2:

$$N_2 = (c_{64}^{(1)}, c_{63}^{(1)}, c_{62}^{(1)}, \ldots, c_{34}^{(1)}, c_{33}^{(1)}).$$

Заповнення накопичувачів N_1 і N_2 зашифровується у режимі простої заміни. Отримане в результаті шифрування заповнення накопичувачів N_1 і N_2 утворює другий 64-х розрядний блок гами шифру $\Gamma^{(2)}$, який додається порозрядно за модулем 2 в суматорі CM_5 з другим блоком відкритих даних $M^{(2)}$. В результаті отримуємо другий 64-розрядний блок зашифрованих даних:

$$C^{(2)} = \Gamma^{(2)} \oplus M^{(2)} = A\,(C^{(1)}) \oplus M^{(2)}.$$

Формування наступних блоків гами шифру $Г^{(i)}$ та шифрування відповідних блоків відкритих даних $M^{(i)}$ проводиться аналогічно.

Якщо довжина останнього q-го блоку відкритих даних $M^{(q)}$ менше 64-розрядів, то з $Г^{(q)}$ використовується тільки відповідне число розрядів гами шифру, інші розряди відкидаються.

У канал зв'язку або пам'ять ЕОМ передаються синхропосилка S і блоки зашифрованих даних $C^{(1)}$, $C^{(2)}$, …, $C^{(q)}$.

Розшифрування даних ДСТУ ГОСТ 28147:2009 у режимі гамування зі зворотним зв'язком

При розшифруванні даних криптографічна схема має той же вигляд, що і при шифруванні (див. рис. 6.4).

Рівняння розшифрування:

$$M^{(1)} = Г^{(1)} \oplus C^{(1)} = A\,(S) \oplus C^{(1)};$$
$$M^{(i)} = A\,(M^{(i\text{-}1)}) \oplus C^{(i)},\ \ i = 2, 3, \ldots, q.$$

З наведених співвідношень випливає, що розшифрування даних можливе тільки за наявності синхропосилки S, яка не є секретним елементом шифру і може зберігатися в пам'яті ЕОМ або передаватися по каналах зв'язку разом з зашифрованими даними.

Розглянемо реалізацію процедури розшифрування. У КЗП вводять 256 бітів ключа, за допомогою якого здійснювалось шифрування даних $M^{(1)}$, $M^{(2)}$, …, $M^{(q)}$. У накопичувачі N_1 і N_2 вводиться синхропосилка S. Початкове заповнення накопичувачів N_1 і N_2 (синхропосилка) зашифровується у режимі простої заміни. Отримане в результаті шифрування заповнення N_1 і N_2 утворює перший блок гами шифру $Г^{(1)} = A\,(S)$, який додається порозрядно за модулем 2 в суматорі $СМ_5$ з блоком зашифрованих даних $C^{(1)}$. В результаті виходить перший блок відкритих даних $M^{(1)} = Г^{(1)} \oplus C^{(1)}$.

Отриманий блок відкритих даних $M^{(1)}$ є початковим заповненням накопичувачів N_1 і N_2 для вироблення другого блоку гами шифру $\Gamma^{(2)} = A(M^{(1)})$. Отримане заповнення накопичувачів N_1 і N_2 зашифровується у режимі простої заміни. Отриманий в результаті шифрування блок $\Gamma^{(2)}$ додається порозрядно за модулем 2 в суматорі CM_5 з другим блоком зашифрованих даних $C^{(2)}$. В результаті отримуємо другий блок відкритих даних $M^{(2)}$.

Аналогічно в N_1 і N_2 послідовно записують блоки відкритих даних $M^{(2)}$, $M^{(3)}$, …, $M^{(q)}$, з яких у режимі простої заміни виробляються відповідні блоки гами шифру $\Gamma^{(3)}$, $\Gamma^{(4)}$, …, $\Gamma^{(q)}$. Блоки гами шифру додаються порозрядно за модулем 2 в суматорі CM_5 з блоками зашифрованих даних $C^{(3)}$, $C^{(4)}$, …, $C^{(q)}$. В результаті отримують блоки відкритих даних $M^{(3)}$, $M^{(4)}$, …, $M^{(q)}$, при цьому останній блок відкритих даних $M^{(q)}$ може містити менше 64 розрядів.

6.5. Режим вироблення імітовставки

Імітовставка – це блок з L бітів, який виробляється за певним правилом з відкритих даних із використанням ключа і потім додається до зашифрованих даних для забезпечення їх імітозахисту.

Імітозахист – це захист системи шифрованого зв’язку від нав’язування помилкових даних.

У стандарті ДСТУ ГОСТ 28147:2009 визначається процес вироблення імітовставки, який однаковий для будь-якого з режимів шифрування даних. Імітовставка I_L виробляється з блоків відкритих даних або перед шифруванням усього повідомлення, або паралельно з шифруванням по блокам. Перші блоки відкритих даних, які беруть участь у виробленні імітовставки, можуть містити службову інформацію (наприклад, адресну частину, час, синхропосилку) і не зашифровуються. Значення параметра L (число двійкових розрядів у імітовставки) визначається криптографічними

вимогами з урахуванням того, що ймовірність нав'язування помилкових перешкод дорівнює $P_{ПП} = 2^{-L}$.

Цей режим не є у загальноприйнятому сенсі режимом шифрування. При роботі у режимі вироблення імітовставки створюється певний додатковий блок, що залежить від усіх відкритих і ключових даних. Даний блок використовується для перевірки того, що у зашифровані дані випадково або навмисно не були внесені зміни. Це особливо важливо для шифрування у режимі гамування, де зловмисник може змінити конкретні біти, навіть не знаючи ключа; однак і при роботі у інших режимах ймовірні зміни не можна виявити, якщо у переданих даних немає надлишкової інформації. Імітовставка виробляється тільки у тому випадку, якщо блоків відкритих даних по 64 біта не менш двох.

Для реалізації алгоритму криптографічного перетворення у режимі вироблення імітовставки (аналог "Режиму зчеплення блоків шифрованих даних") використовуються теж тільки частина блоків загальною криптосистеми (рис. 6.5).

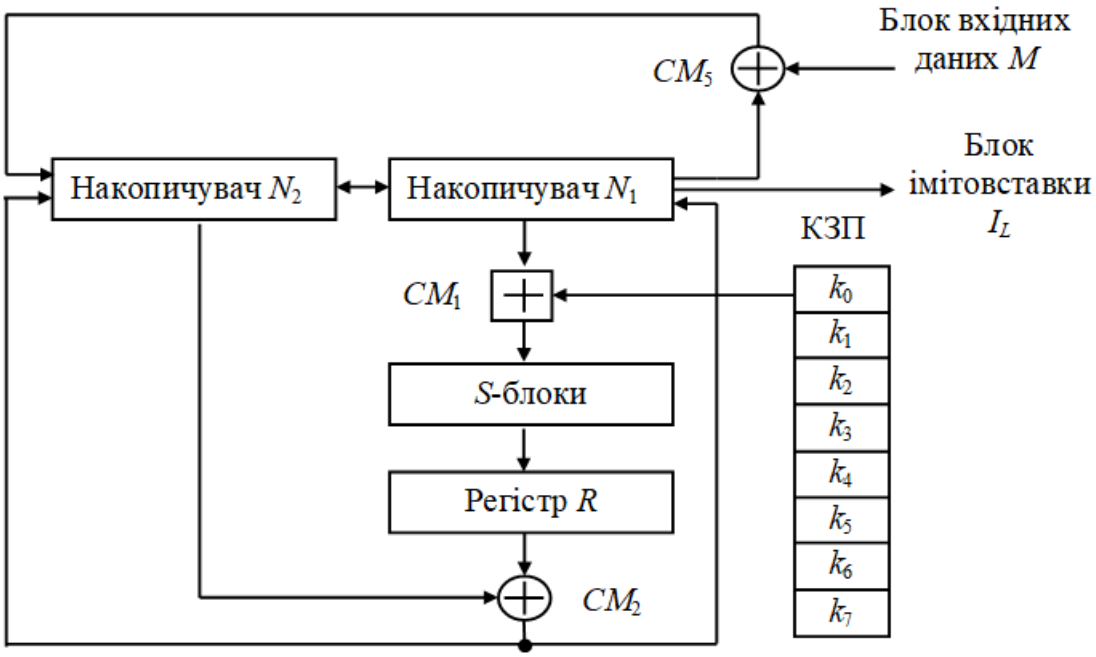


Рис. 6.5. Структура реалізації режиму вироблення імітовставки ДСТУ ГОСТ 28147:2009

Для вироблення імітовставки відкриті дані представляють у вигляді послідовності 64-розрядних блоків $M^{(i)}$ (i = 1, 2, …, q). У КЗП вводять 256 біт ключа.

Перший блок відкритих даних $M^{(1)}$ записується у регістри N_1 і N_2, після чого зазнає перетворення $A'(\bullet)$, що відповідає першим 16 циклам алгоритму шифрування у режимі простої заміни.

Отримане після 16 циклів 64-розрядне число $A'(M^{(1)})$ додається за модулем 2 з другим блоком відкритих даних $M^{(2)}$. Результат додавання $(A'(M^{(1)}) \oplus M^{(2)})$ знову зазнає перетворення $A'(\bullet)$.

Отримане 64-розрядне число $A'(A'(M^{(1)}) \oplus M^{(2)})$ додається за модулем 2 з третім блоком відкритих даних $M^{(3)}$ і знов зазнає перетворення $A'(\bullet)$, отримуючи 64-розрядне число

$$A'(A'(A'(M^{(1)}) \oplus M^{(2)}) \oplus M^{(3)}),$$

і так далі.

Останній блок $M^{(q)}$ (за необхідності доповнений нулями до повного 64-розрядної блоку) додається за модулем 2 з результатом обчислень на кроці q-1, після чого зашифровується у режимі простої заміни, використовуючи перетворення $A'(\bullet)$.

З отриманого 64-розрядного числа вибирають безперервний відрізок I_L (імітовставка) завдовжки L старших (лівих) біт (не менше одного і не більше 64 біт). Імітовставка I_L додається у кінець зашифрованих даних та разом передається по каналам зв'язку або зберігається в пам'яті ЕОМ, тобто

$$C^{(1)}, C^{(2)}, C^{(3)}, \ldots, C^{(q\text{-}1)}, C^{(q)}, I_L.$$

Зашифровані дані, що надійшли до отримувача

$$C^{(1)}, C^{(2)}, C^{(3)}, \ldots, C^{(q\text{-}1)}, C^{(q)}$$

розшифровуються, і з отриманих блоків відкритих даних $M^{(1)}$, $M^{(2)}$, $M^{(3)}$, …, $M^{(q\text{-}1)}$, $M^{(q)}$, аналогічно виробляється імітовставка I_L^*. Ця імітовставка I_L^*

порівнюється з імітовставкою I_L, отриманою разом з зашифрованими даними з каналу зв'язку або з пам'яті ЕОМ. Якщо $I_L^* \neq I_L$, то отримані при розшифруванні блоки відкритих даних $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(q-1)}, M^{(q)}$ вважаються помилковими.

Слід зазначити, що вироблення імітовставки може проводитися паралельно шифруванню з використанням одного з описаних вище режимів роботи.

6.6. Безпека ДСТУ ГОСТ 28147:2009

Шифр вважається добре спроектованим, якщо немає способу розкрити його більш ефективним способом, ніж повним перебором по всьому ключовому простору, тобто по усіх можливих значеннях ключа. ДСТУ ГОСТ 28147:2009 відповідає цьому принципу – за роки інтенсивних досліджень не було запропоновано жодного результативного істотного способу його криптографічного аналізу. У плані стійкості він на багато порядків перевершує розглянутий стандарт шифрування DES. У ДСТУ ГОСТ 28147:2009 використовується 256-бітовий ключ і обсяг ключового простору становить 2^{256}. На жодному з існуючих тепер чи передбачуваних для реалізації в недалекому майбутньому електронному пристрої неможливо підібрати такий ключ за коротший час, ніж кілька сотень років.

Вичерпну відповідь на питання про критерії якості ключів і таблиць ДСТУ ГОСТ 28147:2009 якщо і можна взагалі де-небудь отримати, то тільки у розробників алгоритму. Відповідні дані не були опубліковані у відкритій пресі. Однак згідно з установленим порядком, для шифрування інформації, що має гриф, потрібно використовувати ключові дані, отримані від уповноваженої організації. Непрямим чином це може свідчити про наявність методик перевірки ключових даних на достовірність. Якщо наявність слабких ключів у ДСТУ ГОСТ 28147:2009 – дискусійне питання, то наявність

слабких вузлів заміни не викликає сумніву. Очевидно, що “тривіальна” таблиця замін, за якою будь-яке значення замінюється ним же, є настільки слабкою, що за її використання шифр зламується елементарно, яким би не був ключ.

Набагато складніше оцінити стійкість ослаблених модифікацій стандарту до інтенсивних способів криптографічного аналізу. Однак загальну закономірність можна простежити і тут. Справа в тому, що “профільні” характеристики багатьох найбільш сильних на сьогоднішній момент видів криптографічного аналізу експоненціально залежать від числа раундів шифрування. Так, для лінійного криптографічного аналізу це буде характеристика лінійності L: $L = C \cdot e^{-\alpha \cdot r}$, де C і α – константи; r – кількість раундів. Аналогічна залежність існує і для диференціального криптографічного аналізу. За своїм фізичним змістом усі характеристики такого роду – імовірностні. Зазвичай обсяг необхідних для криптографічного аналізу вихідних даних і його трудомісткість обернено пропорційні подібним характеристикам. Звідси випливає, що ці показники трудомісткості зростають експоненціально зі зростанням числа основних кроків шифрування. Тому при зниженні кількості раундів у кілька разів трудомісткість найбільш відомих видів аналізу зміниться (дуже приблизно і грубо) як корінь цього ступеня з початкової кількості. Це дуже велике падіння стійкості.

З іншого боку, ДСТУ ГОСТ 28147:2009 проектувався з великим запасом міцності і на сьогодні стійкий до всіх відомих видів криптографічного аналізу, включаючи диференціальний і лінійний. Стосовно лінійного криптографічного аналізу це означає, що для його успішного проведення потрібно більше пар “відкритий блок – зашифрований блок”, ніж існує, тобто більш 2^{64}. З урахуванням сказаного вище це означає, що для успішного лінійного криптографічного аналізу 16-раундового ДСТУ ГОСТ 28147:2009 буде потрібно не менше $\sqrt{2^{64}} = 2^{32}$ блоків або 2^{35} байтів або 32 Гбайти даних.

Контрольні питання до розділу 6

1. Для яких цілей може використовуватися алгоритм криптографічного перетворення даних ДСТУ ГОСТ 28147:2009?

2. Перерахуйте основні параметри алгоритму симетричного шифрування ДСТУ ГОСТ 28147:2009.

3. Які операції використовуються в блоковому алгоритмі шифрування ДСТУ ГОСТ 28147:2009?

4. Чим відрізняється останній цикл шифрування і розшифрування даних від усіх попередніх?

5. Які основні відмінності алгоритму шифрування ДСТУ ГОСТ 28147:2009 від алгоритму DES?

6. У яких режимах може проводитися шифрування даних за допомогою алгоритму криптографічного перетворення ДСТУ ГОСТ 28147:2009?

7. Чим відрізняються режими роботи гамування та гамування зі зворотним зв'язком в алгоритмі ДСТУ ГОСТ 28147:2009?

8. Що таке імітовставка? З якою метою може бути використана імітовставка?

9. У регістрі N_1 ДСТУ ГОСТ 28147:2009 знаходяться дані, які мають наступний вигляд: $(267AF2DB)_{16}$, а у N_2 – $(4C4665B2)_{16}$. Ключ шифрування має вигляд: $k_0 = (EB8A7159)_{16}$, $k_1 = (7CE5D63D)_{16}$, $k_2 = (4AC1D6E0)_{16}$, $k_3 = (BAFE4731)_{16}$, $k_4 = (A3DEB025)_{16}$, $k_5 = (8BB389AC)_{16}$, $k_6 = (10D3B61A)_{16}$, $k_7 = (E9AC340F)_{16}$. Цикл шифрування – 20. Що буде знаходиться в N_1 і N_2 після завершення циклу. Як блоки заміни використовувати табл. 6.1.

10. У регістрі N_1 ДСТУ ГОСТ 28147:2009 знаходяться дані, які мають наступний вигляд: $(267AF2DB)_{16}$, а у N_2 – $(4C4665B2)_{16}$. Ключ шифрування має вигляд: $k_0 = (EB8A7159)_{16}$, $k_1 = (7CE5D63D)_{16}$, $k_2 = (4AC1D6E0)_{16}$, $k_3 = (BAFE4731)_{16}$, $k_4 = (A3DEB025)_{16}$, $k_5 = (8BB389AC)_{16}$, $k_6 = (10D3B61A)_{16}$, $k_7 = (E9AC340F)_{16}$. Цикл шифрування – 32. Що буде знаходиться в N_1 і N_2 після завершення циклу. Як блоки заміни використовувати табл. 6.1.

РОЗДІЛ 7
СТАНДАРТ БЛОКОВОГО СИМЕТРИЧНОГО ШИФРУВАННЯ ДАНИХ AES

7.1. Історія розробки стандарту AES

У 1997 році Національний інститут стандартів і технологій США (NIST) оголосив про початок програми з прийняття нового стандарту криптографічного захисту – стандарту XXI століття для закриття важливої інформації урядового рівня на заміну, існуючому з 1974 року, алгоритму DES, найпоширенішого криптографічного алгоритму в світі [4, 32, 40]. DES вважається застарілим за багатьма параметрами: довжині ключа, зручності реалізації на сучасних процесорах, швидкодії тощо, за винятком найголовнішого – стійкості. За 23 роки інтенсивного криптографічного аналізу не було знайдено методів розкриття цього шифру, що істотно відрізняються за ефективністю від повного перебору по ключовому простору.

Вимоги до кандидатів були наступні:

– криптографічний алгоритм повинен бути відкрито опублікований;

– криптографічний алгоритм повинен бути симетричним блоковим шифром, що допускає розміри ключів у 128, 192 і 256 біт;

– криптографічний алгоритм повинен бути призначений як для апаратної, так і для програмної реалізації;

– криптографічний алгоритм повинен бути доступний для відкритого використання в будь-яких продуктах, а значить, не може бути запатентований, в іншому випадку патентні права повинні бути анульовані;

– криптографічний алгоритм повинен піддаватися вивченню за такими параметрами: стійкості, вартості, гнучкості, можливості бути реалізованим в smart-картах.

Стійкість. Це найважливіший критерій в оцінці алгоритму. Оцінювалися: здатність шифру протистояти різним методам

криптографічного аналізу; статистична безпека і відносна захищеність у порівнянні з іншими кандидатами.

Вартість. Не менш важливий критерій, з огляду на одну з основних цілей NIST, – широка область використання і доступність алгоритму. Вартість залежить від обчислювальної ефективності (в першу чергу швидкодії) на різних платформах, зручності програмної і апаратної реалізації, низьких вимог до пам'яті, простоти (прості алгоритми легше реалізовувати, вони більш прозорі для аналізу).

Гнучкість. Гнучкість включає здатність алгоритму обробляти ключі більше обумовленого мінімуму (128 біт), надійність і ефективність виконання в різних середовищах, можливість реалізації інших криптографічних функцій: комбінованого шифрування, хешування тощо. Іншими словами, алгоритм повинен бути істотно більш ефективним з точки зору практичної реалізації (в першу чергу швидкості шифрування і формування ключів), мати більший запас міцності, ніж триразовий DES, при цьому не поступаючись йому в стійкості.

Реалізація в smart-картах. Важлива область використання алгоритму в майбутньому – smart-карти, при цьому головною проблемою є невеликий обсяг доступної пам'яті.

На відкритий конкурс було прийнято 15 алгоритмів, розроблених криптографами 12 країн – Австралії, Бельгії, Великобританії, Німеччини, Ізраїлю, Канади, Коста-Ріки, Норвегії, США, Франції, Південної Кореї та Японії. У фінал конкурсу вийшли наступні алгоритми: Mars, Twofish і RC6 (США), Rijndael (Бельгія), Serpent (Великобританія, Ізраїль, Норвегія). За своєю структурою Twofish є класичним шифром Фейстеля; MARS і RC6 можна віднести до модифікованих шифрів Фейстеля, у них використовується нова маловивчена операція циклічного "прокручування" бітів слова на число позицій, що змінюються залежно від шифрованих даних і секретного ключа; Rijndael і Serpent є класичними SP-мережами. Mars і Twofish мають найскладнішу конструкцію, Rijndael і RC6 – найпростішу.

Фіналісти будуть описані за єдиною схемою, поданою у документі NIST. Спочатку описуються виявлені “слабкості” алгоритму (якщо такі є), потім переваги і, нарешті, недоліки.

Mars виставлений на конкурс фірмою IBM, одним з авторів шифру є Д. Копперсміт, учасник розробки DES. В алгоритмі не виявлено слабкостей в захисті.

Переваги:

– високий рівень захищеності;

– висока ефективність на 32-розрядних платформах, особливо які підтримують операції множення і циклічного зсуву;

– потенційно підтримує розмір ключа більше 256 біт.

Недоліки:

– складність алгоритму ускладнює аналіз його надійності;

– зниження ефективності на платформах без необхідних операцій;

– складність захисту від часового аналізу і аналізу потужності.

RC6 запропонований фірмою RSA Lab, одним з його авторів є Р. Ривест. В алгоритмі не виявлено слабкостей в захисті.

Переваги:

– висока ефективність на 32-бітових платформах, особливо які підтримують операції множення і циклічних зрушень;

– проста структура алгоритму спрощує аналіз його надійності;

– наявність добре вивченого попередника – RC5;

– швидка процедура формування ключа;

– потенційно підтримує розмір ключа більше 256 біт;

– довжина ключа і число раундів можуть бути змінними.

Недоліки:

– відносно низький рівень захищеності;

– зниження ефективності на платформах, які не мають необхідних операцій;

– складність захисту від часового аналізу і аналізу потужності;

– неможливість генерації раундових ключів “на льоту”.

Rijndael більшістю учасників конкурсу названий як найкращій вибір, якщо буде відкинутий їх власний шифр. Заснований на шифрі Square тих самих авторів. В алгоритмі не виявлено слабкостей в захисті.

Переваги:

– висока ефективність на будь-яких платформах;

– високий рівень захищеності;

– добре підходить для реалізації в smart-картах через низькі вимоги до пам’яті;

– швидка процедура формування ключа;

– хороша підтримка паралелізму на рівні інструкцій;

– підтримка різних довжин ключа з кроком 32 біта.

Недолік: вразливий до аналізу потужності.

Serpent – розробка професійних криптографічних аналітиків Р. Андерсона, Е. Біхам і Л. Кнудсена. Створивши шифр, який успішно протистоїть всім відомим на сьогодні атакам, розробники потім подвоїли кількість його раундів. В алгоритмі не виявлено слабкостей в захисті.

Переваги:

– високий рівень захищеності;

– добре підходить для реалізації в smart-картах через низькі вимоги до пам’яті.

Недоліки:

– найповільніший алгоритм серед фіналістів;

– вразливий до аналізу потужності.

Twofish заснований на широко використовуємому шифрі Blowfish; один з авторів розробки – Б. Шнайер. Головна особливість шифру – зміна в залежності від секретного ключа таблиць замін. В алгоритмі не виявлено слабкостей в захисті.

Переваги:

– високий рівень захищеності;

– добре підходить для реалізації в smart-картах через низькі вимоги до пам’яті;

– висока ефективність на будь-яких платформах, у тому числі на очікуваних у майбутньому 64-розрядних архітектур фірм Intel і Motorola;

– підтримує обчислення раундовий ключів “на льоту”;

– підтримує розпаралелювання на рівні інструкцій;

– допускає довільну довжину ключа до 256 біт.

Недоліки:

– особливості алгоритму ускладнюють його аналіз;

– висока складність алгоритму;

– застосування операції додавання робить алгоритм вразливим до аналізу потужності і часового аналізу.

У жовтні 2000 року конкурс завершився. Переможцем був визнаний бельгійський шифр Rijndael, він має найкраще поєднання стійкості, продуктивності, ефективності реалізації та гнучкості. Його низькі вимоги до об’єму пам’яті роблять його таким, що ідеально підходить для вбудованих систем. Авторами шифру є Йон Демен (Joan Daemen) і Вінсент Раймен (Vincent Rijmen), початкові літери прізвищ яких і утворюють назву алгоритму – Rijndael.

Після цього NIST почав підготовку попередньої версії Федерального Стандарту Обробки Інформації (FIPS) і в лютому 2001 року опублікував його на сайті http://csrc.nist.gov/encryption/aes/. Протягом 90-денного періоду відкритого обговорення попередню версію FIPS переглядали з урахуванням коментарів, після чого почався процес виправлень і затвердження. Нарешті, 26 листопада 2001 року було опубліковано остаточну версію стандарту FIPS-197, що описує новий американський стандарт шифрування AES (Advanced Encryption Standard). Згідно з цим документом стандарт набрав чинності з 26 травня 2002 року [32].

7.2. Формат даних AES

Rijndael – це ітераційний блоковий шифр, що має архітектуру “Квадрат”. Довжина ключа в шифрі може дорівнювати 128, 192 або 256 бітам. У стандарті AES визначена довжина блоку даних, які обробляються, що дорівнює 128 бітам.

AES використовує п’ять одиниць для представлення даних: біти, байти, слова, блоки та масиви станів. Біт – найменша й елементарна одиниця; інші одиниці можуть бути виражені в термінах менших одиницях. В AES біт – двійкова цифра зі значенням 0 або 1. Для позначення бітів будемо використовувати малі літери. Рис. 7.1 показує одиницю (формат) даних – байт.

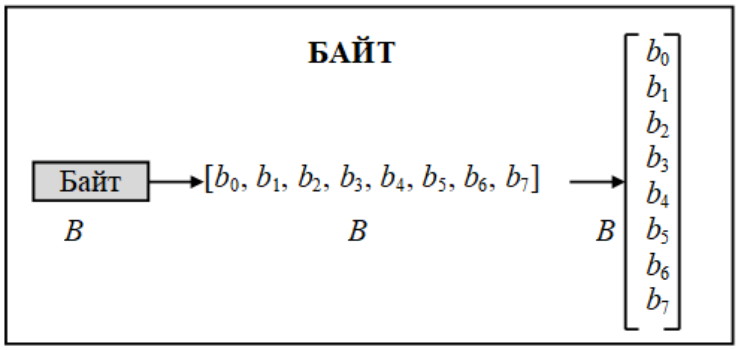


Рис. 7.1. Одиниця (формат) даних – байт

Байт – група з восьми бітів, яка може бути оброблена як єдиний об’єкт: матриця з одного рядка (1×8) з восьми бітів або матриця-стовпець (8×1) з восьми бітів. Коли інформація байта обробляється як матриця-рядок, то біти вставляються в матрицю зліва направо. Коли байт обробляється як матриця-стовпець, то біти вставляються в матрицю зверху вниз. Для позначення байта будемо використовувати велику літеру (*B*).

Рис. 7.2 показує одиницю (формат) даних – слово. Слово – група з 32 бітів, яка може бути оброблена як єдиний об’єкт. Це матриця з рядка або стовпця із чотирьох байтів. Коли слово обробляється як матриця-рядок, байти вставляються зліва направо. Коли слово обробляється як матриця-

стовпець, байти вставляються зверху вниз. Для позначення слова будемо використовувати велику літеру *W*.

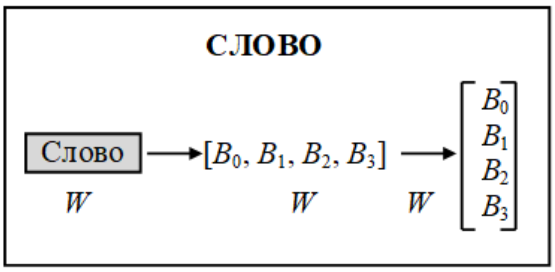


Рис. 7.2. Одиниця (формат) даних – слово

Рис. 7.3 показує одиницю (формат) даних – блок. AES зашифровує і розшифровує блоки даних. Блок в AES – група з 128 бітів. Однак блок може бути представлений як матриця-рядок з 16 байтів.

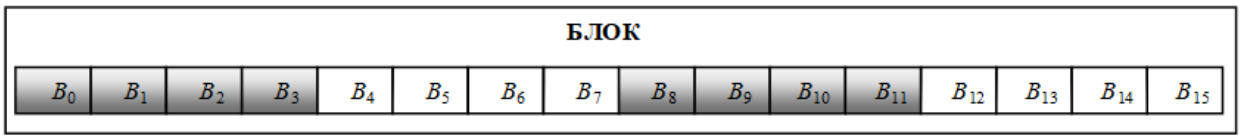


Рис. 7.3. Одиниця (формат) даних – блок

AES використовує декілька раундів, кожний раунд складається з кількох каскадів. Блок даних перетворюється з одного каскаду в інший. На початку та в кінці шифру AES застосовується термін блок даних; до й після кожного каскаду блок даних називається матрицею стану. Для позначення цієї матриці будемо використовувати велику літеру *S*. Рис. 7.4 показує одиницю (формат) даних – матрицю стану.

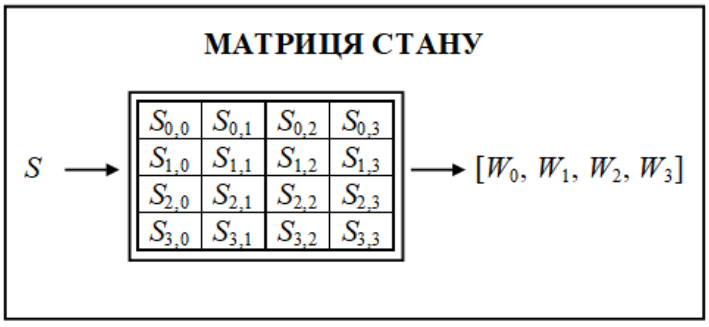


Рис. 7.4. Одиниця (формат) даних – матриця стану

Матриця стану, подібно до блоку, складається з 16 байтів, але зазвичай обробляється як матриця 4×4 байтів. У цьому випадку кожний елемент матриці станів позначається як $S_{r,c}$, де r (від 0 до 3) визначає рядок і c (від 0 до 3) визначає стовпець. Іноді матриця стану обробляється як матриця-рядок слів (1×4). Це має сенс, якщо представляти слово як матрицю-стовпець. На початку шифру байти в блоці даних вставляються в матрицю стану стовпець за стовпцем, у кожному стовпці – зверху вниз. У кінці шифру байти з матриці стану вилучаються, як це показано на рисунку 7.5.

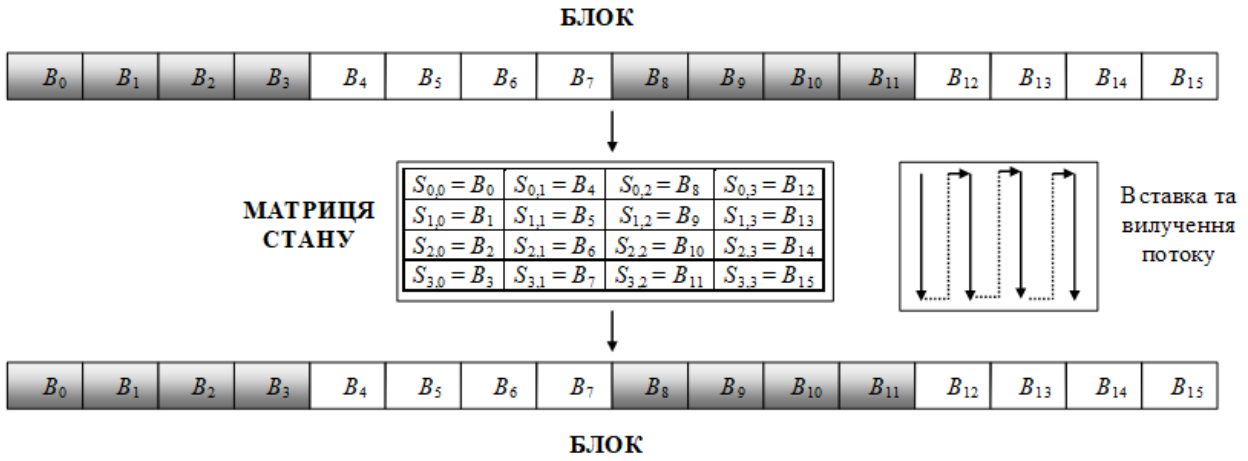


Рис. 7.5. Перетворення блоку в матрицю стану і матриці стану в блок

Приклад 7.1. Розглянемо, як можна представити блок із 16 символів у вигляді матриці розміром 4×4. Припустимо, що текстовий блок – "*AES uses a matrix*". Додамо в кінець блока два фіктивних символи та отримаємо блок "*AESUSESAMATRIXZZ*". Тепер замінимо кожний символ його десятковим числом між 00 і 25 (див. рис. 2.1).

Представимо кожний байт як ціле число з двома шістнадцятковими цифрами. Наприклад, символ *S* спочатку поміняємо на десяткове "18", а потім запишемо в шістнадцятковому зображенні як "12". Матриця стану тоді заповнюється стовпець за стовпцем, як це показано на рис. 7.6.

Текст	*A*	*E*	*S*	*U*	*S*	*E*	*S*	*A*	*M*	*A*	*T*	*R*	*I*	*X*	*Z*	*Z*
Десяткове значення	00	04	18	20	18	04	18	00	12	00	19	17	08	23	25	25
Шістнадцяткове значення	00	04	12	14	12	04	12	00	0C	00	13	11	08	17	19	19

МАТРИЦЯ СТАНУ

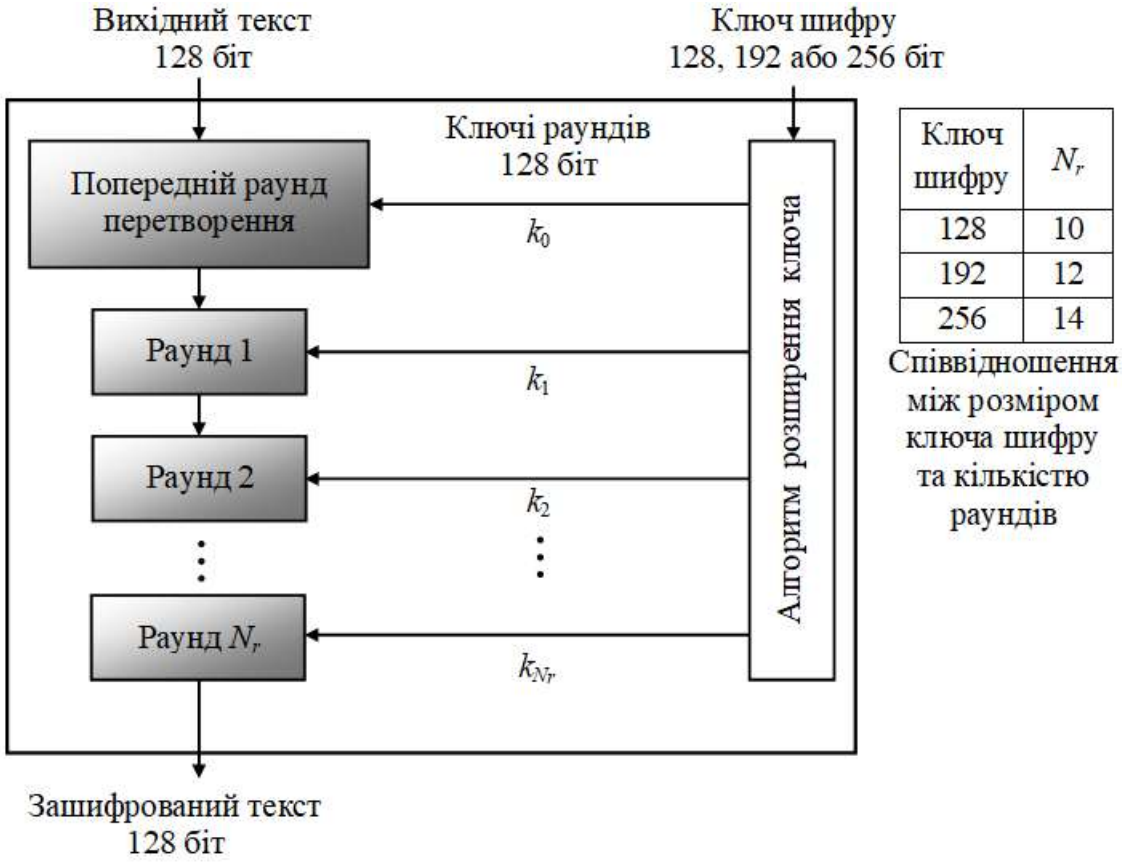
00	12	0C	08
04	04	00	17
12	12	13	19
14	00	11	19

Рис. 7.6. Перехід від початкового тексту (блоку) до матриці стану

7.3. Структура алгоритму і раундів AES

Структура алгоритму AES

AES – шифр не-Фейстеля, який зашифровує й розшифровує блок даних 128 бітів із використанням розміру ключа 128, 192 або 256 бітів. Кількість раундів – 10, 12 або 14 – залежить від довжини ключа. На рис. 7.7 показано загальну структуру алгоритму шифрування AES.



Ключ шифру	N_r
128	10
192	12
256	14

Співвідношення між розміром ключа шифру та кількістю раундів

Рис. 7.7. Загальна структура алгоритму шифрування AES

Значення N_r визначає кількість раундів. Таблиця показує співвідношення між розміром ключа і кількістю раундів. Це означає, що існує три різних версії AES; вони позначаються як AES-128, AES-192 і AES-256. Однак ключі раундів, які створені алгоритмом розширення ключа завжди 128 біт, мають той самий розмір, що й блоки вихідних та зашифрованих даних. Кількість ключів раундів завжди на один більше, ніж кількість раундів. Іншими словами, маємо кількість раундовий ключів $N_r + 1$. Позначимо раундові ключі як: k_0, k_1, k_2, ..., k_{N_r}.

Структура раундів алгоритму AES

На рис. 7.8 показано структуру кожного раунду на стороні шифрування.

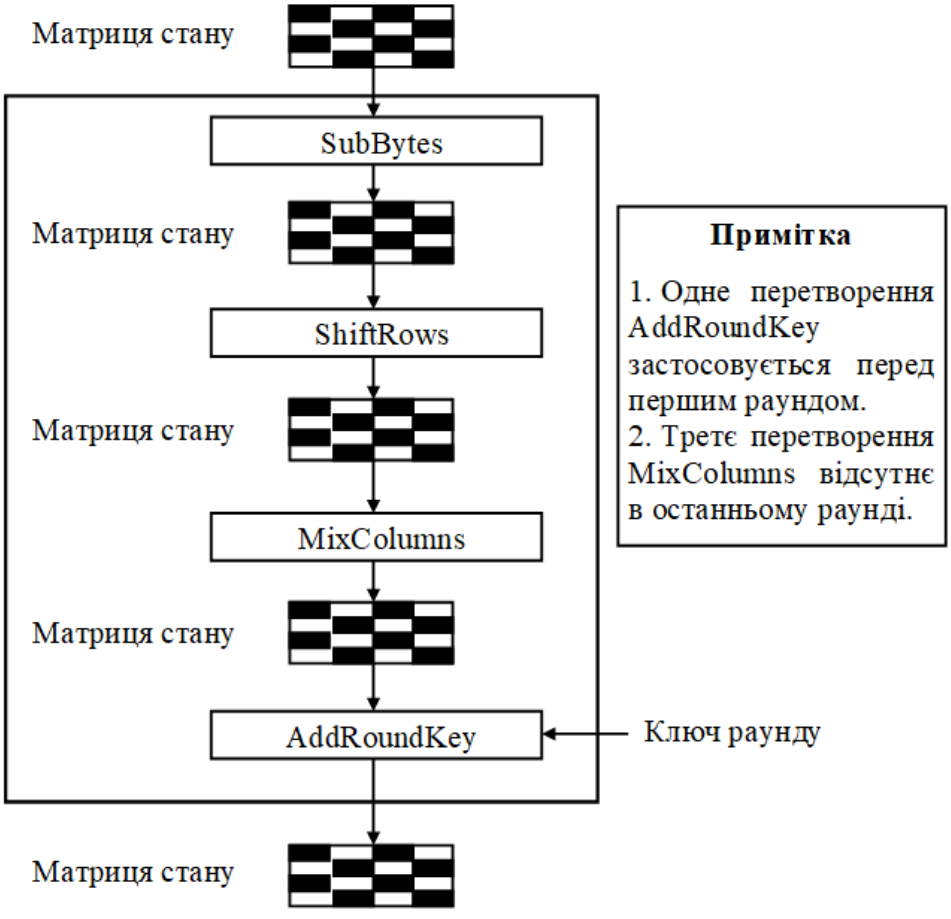


Рис. 7.8. Структура кожного раунду AES на стороні шифрування

Кожний раунд, крім останнього, використовує чотири перетворення, які є оберненими. Останній раунд має тільки три перетворення (перетворення MixColumns відсутнє). Попередній раунд використовує тільки одне перетворення (AddRoundKey).

На відміну від попередніх розглянутих симетричних блокових шифрів, AES не використовує будь-який аналог структури Фейстеля. Кожний раунд складається з трьох різних обернених перетворень, що називаються шарами:

– лінійний шар, що змішує, гарантує високу ступінь взаємопроникнення символів блоку для маскування статистичних зв'язків;

– нелінійний шар реалізований за допомогою *S*-блоків, що мають оптимальну нелінійність, і запобігає можливості використання диференціального, лінійного та інших сучасних методів криптографічного аналізу;

– шар додавання з ключем безпосередньо виконує шифрування.

Шифр починається і закінчується додаванням із ключем. Це дозволяє закрити вхід першого раунду під час атаки за відомим текстом і зробити криптографічно значущим результат останнього раунду.

7.4. Раундові перетворення алгоритму AES

Для шифрування даних алгоритм AES використовує так звану колову функцію, яка складається із чотирьох різних байт-орієнтованих (раундових) перетворень:

– SubBytes – заміни байтів матриці стану з використанням таблиці підстановки;

– ShiftRows – зсуву рядків матриці стану на різну кількість позицій;

– MixColumns – перемішування стовпців матриці стану;

– AddRoundKey – додавання раундового ключа до матриці стану.

Для розшифрування даних алгоритм AES використовує також колову функцію, яка складається із чотирьох різних байт-орієнтованих (раундових)

перетворень, які є оберненими (інверсними) до перетворень при шифруванні відповідно: InvSubBytes; InvShiftRows і InvMixColumns. Інверсне перетворення InvAddRoundKey аналогічне перетворенню AddRoundKey, тому що використовує операцію порозрядного додавання *xor*.

Заміна байтів матриці стану – SubBytes і InvSubBytes

Процедура ***SubBytes*** використовується на стороні шифрування й реалізує шар нелінійного перетворення (є нелінійна заміна байтів), що виконується незалежно з кожним байтом стану. Щоб застосувати заміну до байта, необхідно інтерпретувати байт як дві шістнадцяткові цифри. Ліва цифра визначає рядок, а права – стовпець в табл. 7.1. На перетині рядка і стовпця, позначених цими шістнадцятковими цифрами, знаходиться новий байт. Рисунок 7.9 ілюструє цю ідею.

Наприклад, результат перетворення байта $(53)_{16}$ знаходиться на перетині 5-го рядка і 3-го стовпця і дорівнює $(ED)_{16}$.

Таблиця 7.1

Таблиця перетворення SubBytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	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	F7	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	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

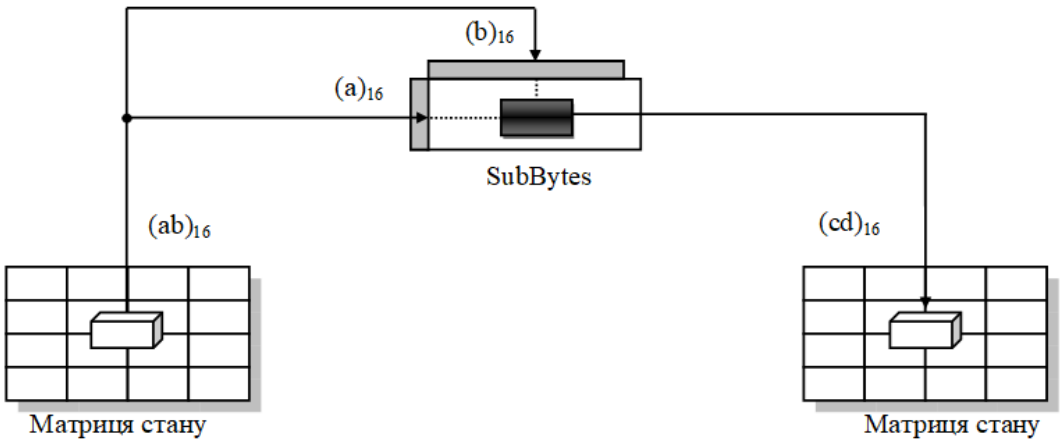


Рис. 7.9. Пояснення ідеї табличної заміни в перетворенні SubBytes

У перетворенні SubBytes матриця стану обробляється як матриця байтів 4×4. В один момент проводиться перетворення одного байта. Зміст кожного байта змінюється, але розташування байтів у матриці залишається таким самим. У процесі перетворення кожний байт перетворюється незалежно від інших – це шістнадцять незалежних перетворень байт у байт.

Приклад 7.2. Нехай на вхід функції SubBytes алгоритму AES надходить наступна матриця стану:

$$S = (193DE3BEA0F4E22B9AC68D2AE9F84808)_{16}$$

Якщо провести табличну заміну кожного байта (див. табл. 7.1), то вийде (рис. 7.10):

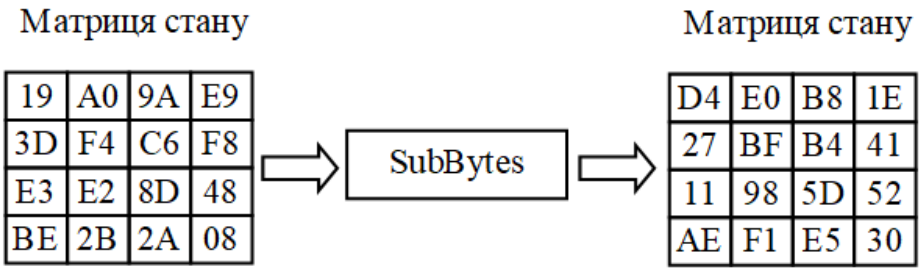


Рис. 7.10. Результат виконання табличної заміни для прикладу 7.2

Перетворення SubBytes забезпечує ефект перемішування. Наприклад, два байти, $(5A)_{16}$ і $(5B)_{16}$, які відрізняються тільки одним бітом (крайній правий біт), перетворюються в $(BE)_{16}$ і $(39)_{16}$ відповідно, які відрізняються чотирма бітами.

Процедура ***InvSubBytes*** використовується на стороні розшифрування і є інверсією перетворення SubBytes. Табл. 7.2. представляє таблицю замін для перетворення InvSubBytes. Табл. 7.1 і 7.2 взаємообернені. Наприклад, якщо виконати перетворення над байтом $(2D)_{16}$ за допомогою SubBytes (див. табл. 7.1), то отримаємо $(D8)_{16}$. Якщо тепер застосувати до байта $(D8)_{16}$ перетворення InvSubBytes (табл. 7.2), то отримаємо $(2D)_{16}$.

Таблиця 7.2

Таблиця перетворення InvSubBytes

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

Приклад 7.3. Рис. 7.11 показує, як матриця стану перетворюється з використанням SubBytes і InvSubBytes.

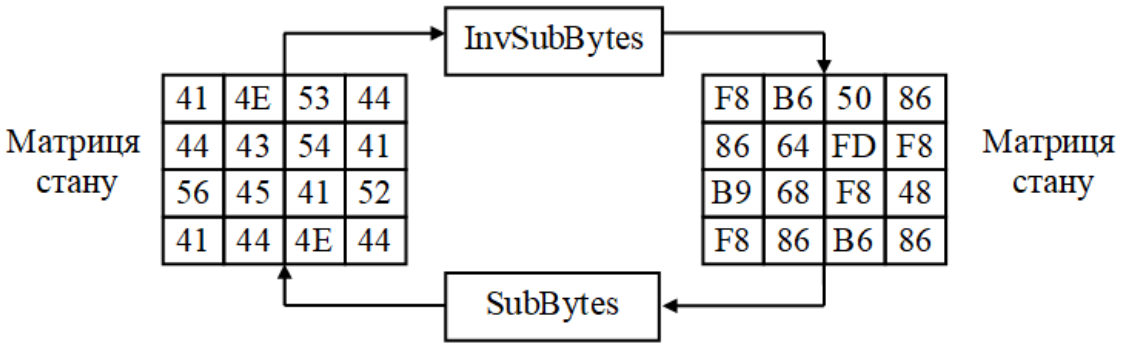


Рис. 7.11. Перетворення SubBytes і InvSubBytes для прикладу 7.3

Бачимо, що InvSubBytes однозначно відтворює оригінал. Зауважимо, що якщо два байти мають однакове значення, то вони перетворяться також однаково. Наприклад, два однакових байта $(4E)_{16}$ у лівій матриці стану перетворюються в однакові байти $(B6)_{16}$ у правій матриці стану і навпаки. Причина в тому, що кожний байт використовує ту саму таблицю перетворень.

Хоча можна використовувати табл. 7.1 і 7.2 для перестановки кожного байта, AES дає визначення алгебраїчним перетворенням на основі поля $GF(2^8)$ за допомогою поліномів, як це показано на рис. 7.12.

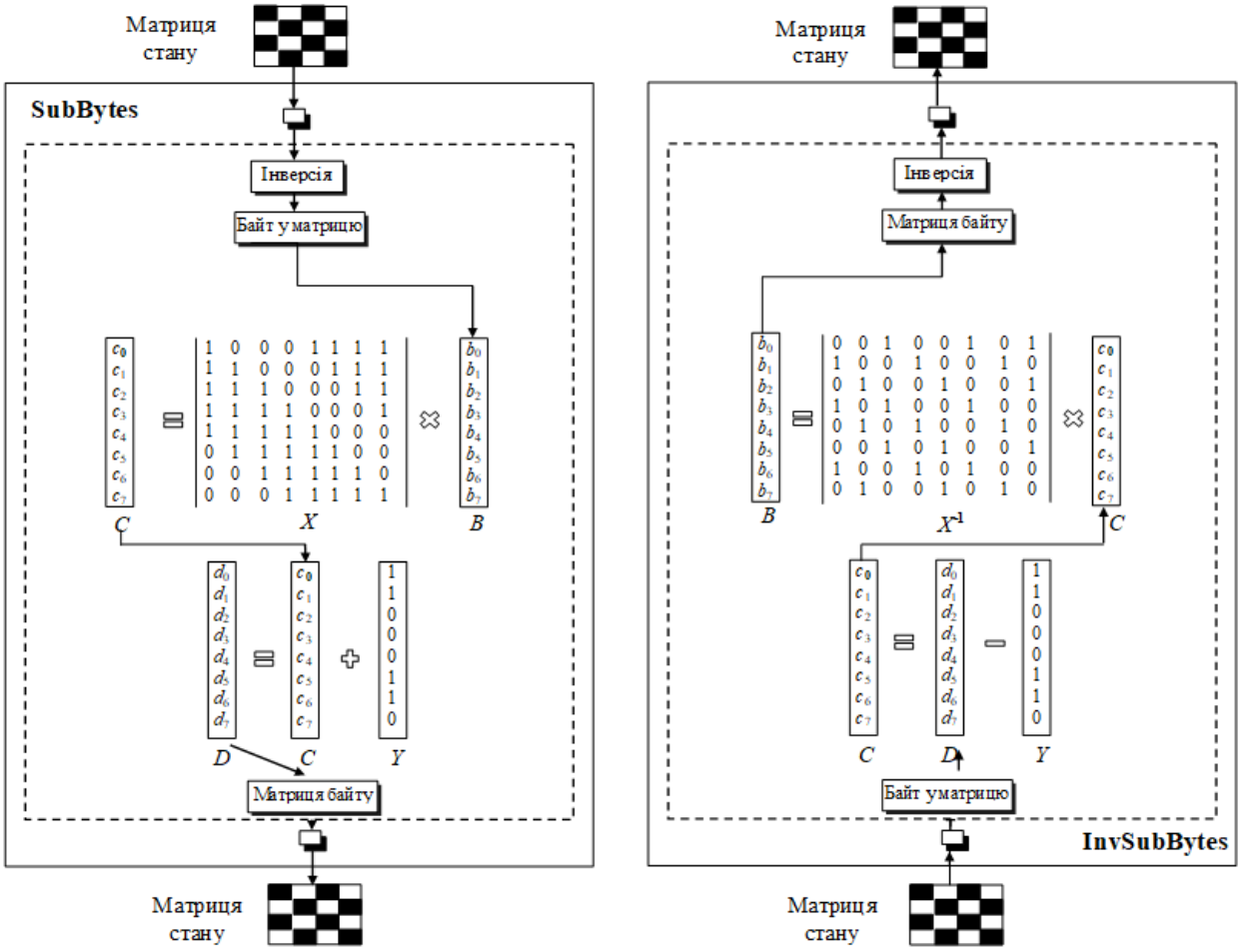


Рис. 7.12. Перетворення SubBytes і InvSubBytes

У процедурі SubBytes байт (двійковий рядок на 8 біт) знаходиться в полі $GF(2^8)$ з незвідним поліномом $p(x) = x^8+x^4+x^3+x+1$. Зауважимо, що байт $(00)_{16}$ сам є власним оберненням. Обернений байт інтерпретується як

матриця-стовпець *B* із наймолодшим бітом угорі та найстаршим бітом унизу. Ця матриця-стовпець множиться на постійну квадратну матрицю *X* і результат, який є матрицею-стовпцем *C*, додається з постійною матрицею-стовпцем *Y*, що дає новий байт. Зауважимо, що множення або додавання бітів відбувається в $GF(2^8)$. InvSubBytes робить ті самі дії, але в зворотному порядку.

У процесі шифрування множення є першою операцією, додавання – другою. У ході розшифрування віднімання (додавання з інверсією) є першою операцією, а ділення (множення з інверсією) – другою.

Хоча множення й додавання матриць у процедурі SubBytes і InvSubBytes – перетворення афінного типу й лінійні, заміна байта його мультиплікативно оберненим значенням в $GF(2^8)$ – нелінійне. Цей крок робить все перетворення нелінійним.

Зсув рядків матриці стану – ShiftRows і InvShiftRows

Процедура ***ShiftRows*** використовується на стороні шифрування, реалізує шар лінійного перетворення і є циклічним зсувом вліво рядків матриці стану. Число зсувів залежить від номеру рядка (0, 1, 2 або 3) матриці стану. Це означає, що нульовий рядок взагалі не зсувається, а останній рядок зсувається на три байта. Рис. 7.13 показує перетворення ShiftRows. Зауважимо, що це перетворення працює одночасно тільки з одним рядком.

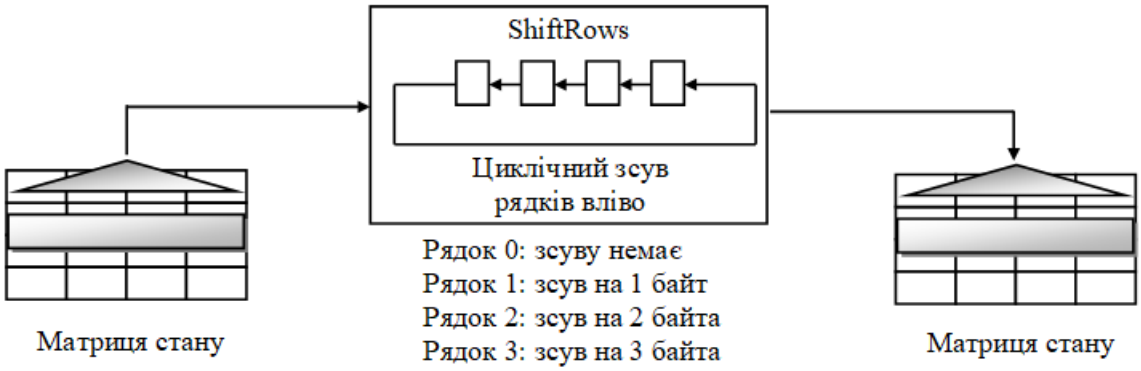


Рис. 7.13. Перетворення ShiftRows

Процедура ***InvShiftRows*** використовується на стороні розшифрування і є інверсією перетворення ShiftRows. Рядки матриці стану циклічно зсуваються вправо на відповідну кількість байтів (0, 1, 2 або 3).

Приклад 7.4. Нехай на вхід функції ShiftRows алгоритму AES надходить наступна матриця стану:

$$S = (\text{F886B9F8B664688650FDF8B686F84886})_{16}$$

На рис. 7.14 показано результат виконання зсувів рядків матриці стану вліво за допомогою функції ShiftRows та вправо за допомогою функції InvShiftRows.

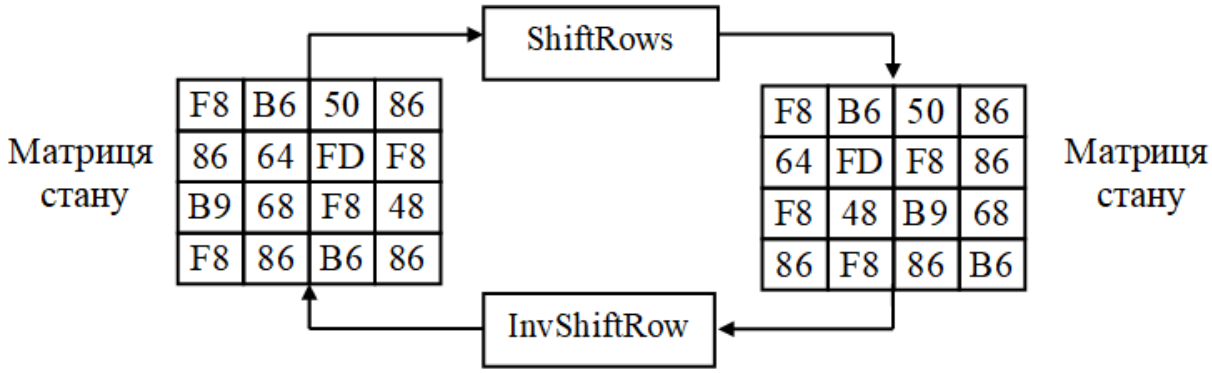


Рис. 7.14. Перетворення ShiftRows та InvShiftRows для прикладу 7.4

Перемішування стовпців матриці стану – MixColumns і InvMixColumns

Процедура ***MixColumns*** використовується на стороні шифрування, також реалізує шар лінійного перетворення і являє собою матричне множення кожного стовпця матриці стану на квадратну матрицю констант. Байти в стовпці матриці стану і в матриці констант інтерпретуються як слова по 8 бітів (або поліном) з коефіцієнтами в $GF(2)$. Множення байтів виконується в $GF(2^8)$ з незвідним поліномом $p(x) = x^8+x^4+x^3+x+1$. Додавання – це застосування операції *xor* до слів по 8 біт.

Рис. 7.15 демонструє застосування перетворення MixColumns до одного стовпця матриці стану. Видно, що кожний елемент вихідної матриці стану залежить від усіх чотирьох елементів початкової матриці стану. Дане перетворення забезпечує розсіювання на розрядному рівні, так як кожний біт нового байту залежить від усіх біт сусідніх байтів (стовпця).

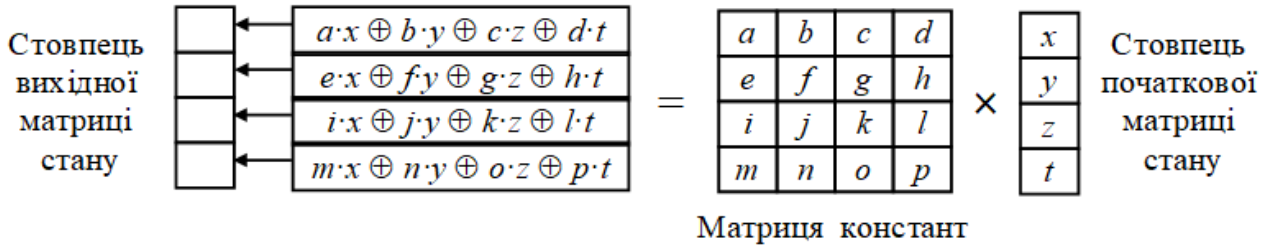


Рис. 7.15. Дія MixColumns – перетворення стовпців матриці

Процедура ***InvMixColumns*** використовується на стороні розшифрування. Для того, щоб це перетворення було інверсією, перетворення MixColumns застосовується інверсна матриця констант. Рис. 7.16 показує матриці констант, що використовуються в AES для перетворень MixColumns та InvMixColumns.

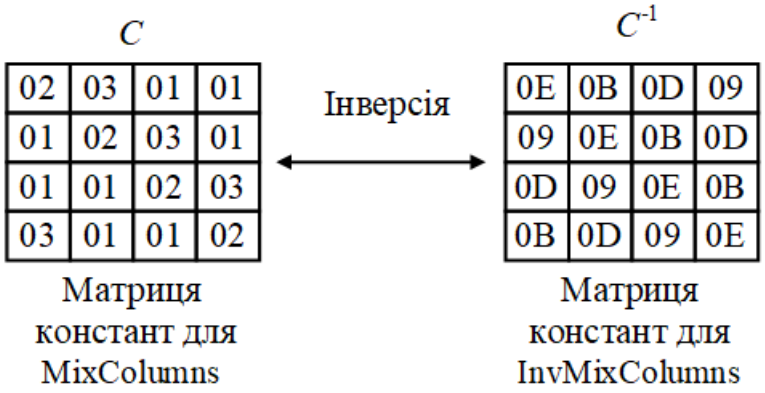


Рис. 7.16. Матриці констант, що використовують перетворення MixColumns та InvMixColumns

Перетворення InvMixColumns схоже на MixColumns. Рис. 7.17 показує принцип перетворення MixColumns або InvMixColumns.

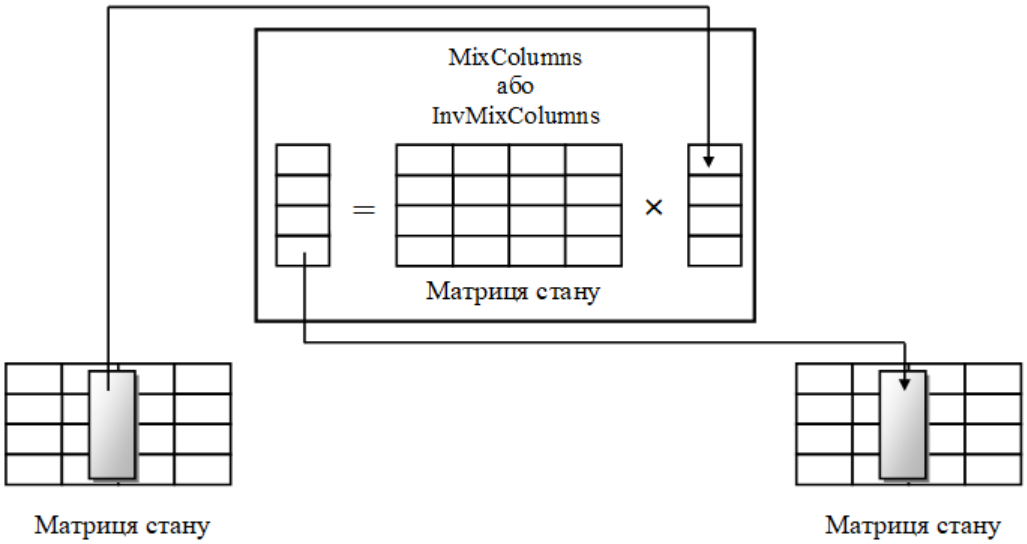


Рис. 7.17. Принцип перетворення MixColumns або InvMixColumns

Приклад 7.5. Нехай на вхід перетворення MixColumns алгоритму AES надходить наступна матриця стану:

$$S = (63F27DD4C963D4FAFE26C96330F2C982)_{16}$$

Рис. 7.18 показує, як матриця стану перетвориться, якщо використовувати перетворення MixColumns. Рисунок також показує, що перетворення InvMixColumns створює початкове значення матриці стану.

Відмітимо, що байти, які рівні між собою у вихідній матриці станів, більше не рівні в новій матриці станів. Наприклад, два однакових байти $(F2)_{16}$ у другому рядку вихідної матриці стану перетворюються на різні байти $(CF)_{16}$ та $(0D)_{16}$.

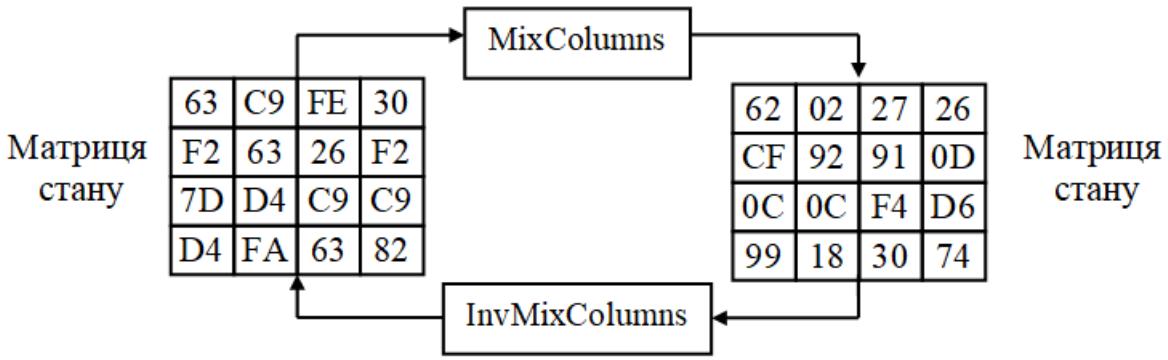


Рис. 7.18. Перетворення MixColumns та InvMixColumns для прикладу 7.5

Додавання раундового ключа з матрицею стану – ADDRoundKey

Ймовірно, найважливіше перетворення – це перетворення, яке включає ключ шифру. Усі попередні перетворення використовують відомі алгоритми, які є оберненими. Якщо не додавати ключовий шифр у кожному раунді, противник дуже просто визначить вихідне повідомлення за даним йому зашифрованим. У цьому випадку охоронцем таємниці буде тільки ключ шифру.

AddRoundKey обробляє в один момент часу один стовпець. Перетворення подібне до MixColumns. MixColumns помножує квадратну матрицю констант на кожний стовпець матриці станів. AddRoundKey додає ключове слово раунду з кожним стовпцем матриці стану. У MixColumns застосовується матричне множення, в AddRoundKey – операції додавання і віднімання. Оскільки додавання й віднімання в кінцевому полі ті самі, то перетворення AddRoundKey обернене само до себе. Рис. 7.19 показує принцип перетворення AddRoundKey. Перетворення AddRoundKey реалізує шар додавання оброблюваних даних (стану) із раундовим ключем. У даній операції раундовий ключ додається до матриці стану за допомогою простого порозрядного *xor*.

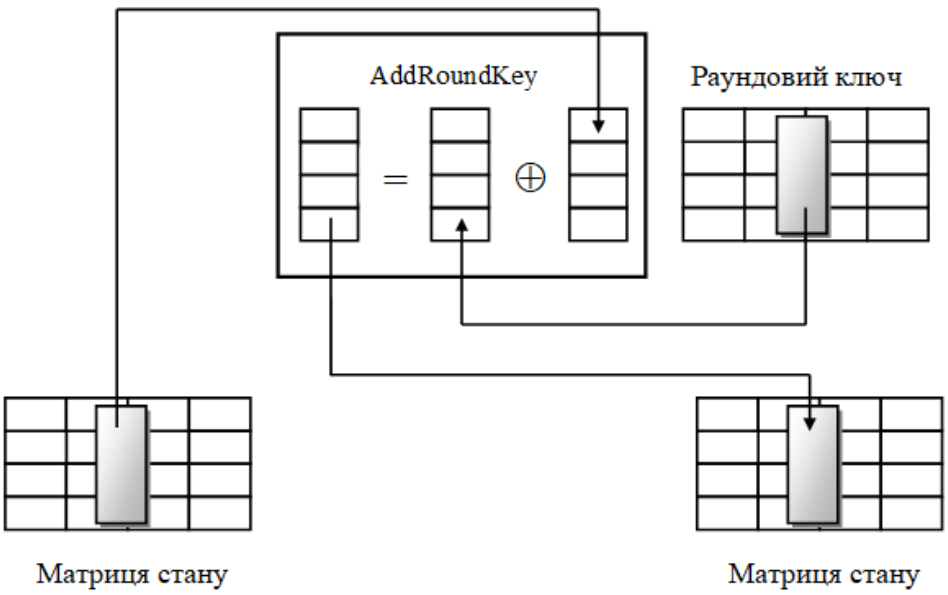


Рис. 7.19. Принцип перетворення AddRoundKey

Приклад 7.6. Нехай на вхід функції AddRoundKey алгоритму AES надходять наступні матриці:

– стану $S = (046681E5C0CB199A48F8D37A2806264C)_{16}$

– раундового ключа $S = (A0FAFE1788542CB123A339392A6C7605)_{16}$.

Виконуючи порозрядне додавання за модулем 2 (операція *xor*) стовпців матриці стану та матриці раундових ключів, отримаємо матрицю стану на виході функції AddRoundKey (рисунок 7.20).

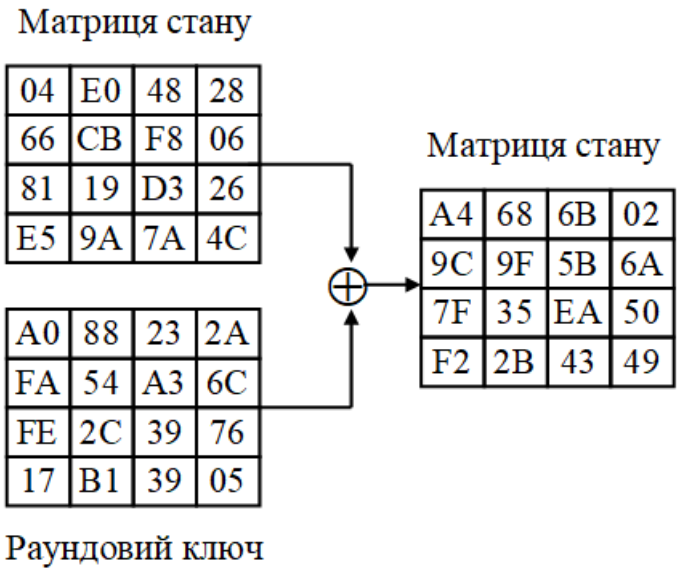


Рис. 7.20 Перетворення AddRoundKey для прикладу 7.6

Підстановка, яка робиться перетворенням SubBytes, що змінює значення байту, засноване тільки на початковому значенні та вході в таблицю; процес не включає сусідні байти. Можна сказати, що SubBytes – внутрішньобайтове перетворення. Перестановка, яка робиться перетворенням ShiftRows, міняє місцями байти, не переставляючи біти в байтах. Можна сказати, що ShiftRows – перетворення обміну байтами. Тепер нам потрібно внутрішньобайтове перетворення, що змінює біти в байтах і засноване на бітах у сусідніх байтах. Необхідно змішати байти, щоб забезпечити розсіювання на розрядному рівні.

Перетворення змішування MixColumns змінює зміст кожного байта, перетворюючи чотири байти одночасно й об'єднуючи їх, щоб отримати чотири нових байта. Щоб гарантувати, що кожний новий байт буде відрізнятися від іншого (навіть якщо всі чотири байти ті самі), процес спочатку помножує кожний байт на різний набір констант і потім змішує їх. Змішування може бути забезпечене матричним множенням. Коли множимо квадратну матрицю на матрицю-стовпець, результат – нова матриця-стовпець. Після того, як матриця помножена на значення рядка в матриці констант, кожний елемент у новій матриці буде залежить від усіх чотирьох елементів старої матриці.

7.5. Алгоритм розширення ключа в AES

Для того, щоб створити ключ для кожного раунду, AES використовує алгоритм ключового розширення. Якщо кількість раундів N_r, то процедура розширення ключів створює N_r+1 раундових ключів по 128 біт кожний від єдиного ключа шифру. Перший раундовий ключ використовується для попереднього перетворення перед початком раундів із використанням перетворення AddRoundKey; інші раундові ключі використовуються наприкінці кожного раунду також з використанням перетвореня AddRoundKey.

Процедура розширення ключа створює слово за словом, з яких надалі будуть формуватися раундові ключі. Слово – це масив із чотирьох байтів. Процедура розширення ключа створює $4\times(N_r+1)$ слів, які позначаються

$$W = w_0, w_1, w_2, ..., w_{4(N_r+1)-1}.$$

Іншими словами, у версії AES-128 ($N_r = 10$) буде $W = 44$ слова; у версії AES-192 ($N_r = 12$) – $W = 52$ слова; та у версії AES-256 ($N_r = 14$) – $W = 60$ слів. Кожний раундовий ключ складається із чотирьох слів. Табл. 7.3 показує відношення між раундами та словами.

Таблиця 7.3

Слова для кожного раунду алгоритму AES

Раунд	Слова			
0	w_0	w_1	w_2	w_3
1	w_4	w_5	w_6	w_7
2	w_8	w_9	w_{10}	w_{11}
…	…	…	…	…
N_r	$w_{4 \cdot N_r}$	$w_{4 \cdot N_r+1}$	$w_{4 \cdot N_r+2}$	$w_{4 \cdot N_r+3}$

Розглянемо алгоритм розширення ключа для версії AES-128; алгоритми інших версій за винятком невеликих змін – такі самі. Рис. 7.21 показує, як з ключа шифру завдовжки 128 біт отримати 44 слова.

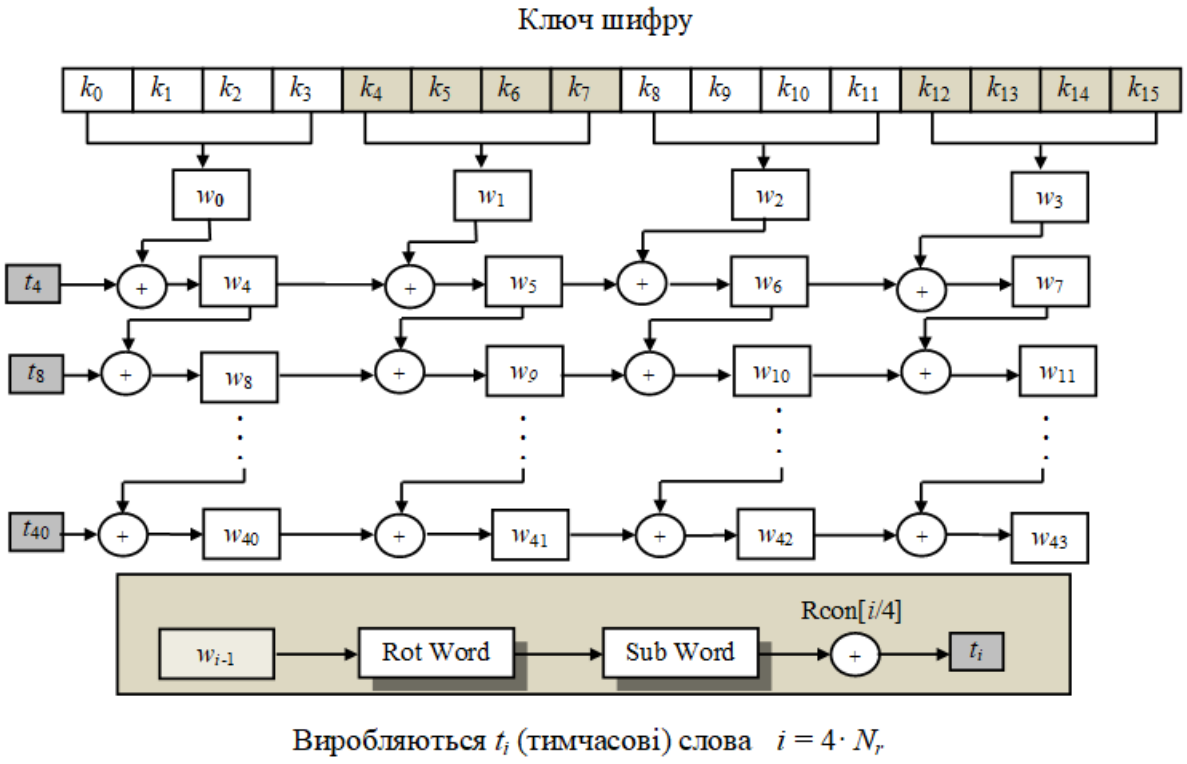


Рис. 7.21. Процес розширення ключа в AES-128

Алгоритм розширення ключа AES-128

1. Перші чотири слова (w_0, w_1, w_2, w_3) виходять із ключа шифру. Ключ шифру представлений як масив із 16 байтів (від k_0 до k_{15}). Перші чотири

байти (від k_0 до k_3) стають w_0; наступні чотири байти (від k_4 до k_7) стають w_1 і так далі. Іншими словами, послідовне з'єднання (конкатенація) слів у цій групі копіює ключ шифру.

2. Інші частини слів w_i від $i = 4$ до $i = 43$ отримують так:

а) якщо $(i \bmod 4) \neq 0$, то $w_i = w_{i-1} \oplus w_{i-4}$ (відповідно до рис. 7.21 це означає, що кожне слово отримане з одного лівого й одного верхнього);

б) якщо $(i \bmod 4) = 0$, то $w_i = t_i \oplus w_{i-4}$ (у даному випадку t_i – тимчасове слово, результат застосування двох процесів, SubWord та RotWord, зі словом w_{i-1} та застосування операції *xor* з константою раунду Rcon [i/4]).

Іншими словами, маємо

$$t_i = \text{SubWord (Rotword } (w_{i-1})) \oplus \text{Rcon } [i/4].$$

RotWord (Rotate Word) – процедура, подібна до перетворення ShiftRows, але використовується тільки до одного стовпця. Процедура приймає слово як масив із чотирьох байтів і циклічно зсуває на один байт угору.

SubWord (Substitute Word) – процедура, подібна до перетворення SubBytes, але застосовується тільки до одного стовпця. Процедура приймає кожний байт у слові й замінює його іншим.

Кожна константа раунду Rcon (RoundConstants) – це 4-байтове значення, в якому крайні праві (старші) три байти завжди є нульовими. Значення констант раундів Rcon наведено в табл. 7.4.

Таблиця 7.4

Значення констант раундів Rcon для AES-128

Раунд	Константа (RCon)	Раунд	Константа (RCon)
1	$(01\ 00\ 00\ 00)_{16}$	6	$(20\ 00\ 00\ 00)_{16}$
2	$(02\ 00\ 00\ 00)_{16}$	7	$(40\ 00\ 00\ 00)_{16}$
3	$(04\ 00\ 00\ 00)_{16}$	8	$(80\ 00\ 00\ 00)_{16}$
4	$(08\ 00\ 00\ 00)_{16}$	9	$(1B\ 00\ 00\ 00)_{16}$
5	$(10\ 00\ 00\ 00)_{16}$	10	$(36\ 00\ 00\ 00)_{16}$

Приклад 7.7. Нехай на вхід алгоритму AES-128 надходить наступний ключ шифрування K:

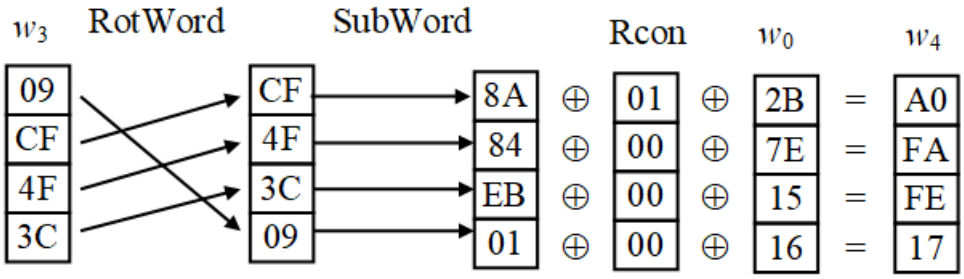
w_0	w_1	w_2	w_3
2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

Сформувати чотирьохбайтові слова розширеного ключа: w_4, w_5, w_6, та w_7.

Рішення. Сформуємо слово w_4. У зв'язку з тим, що $4 \bmod 4 = 0$, то слово w_4 буде сформовано за виразом:

$$w_4 = \text{SubWord}\,(\text{Rotword}\,(w_3)) \oplus \text{Rcon}\,[1] \oplus w_0$$

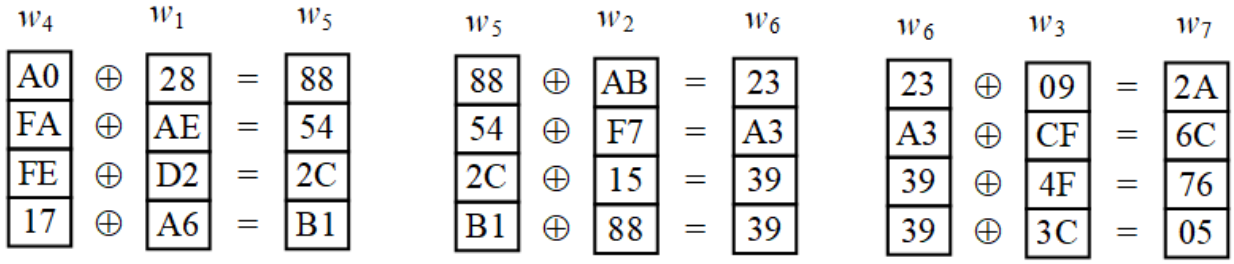
або



Сформуємо слова w_5, w_6, та w_7. Оскільки значення 5 mod 4, 6 mod 4 та 7 mod 4 не дорівнюють нулю, то слова будуть сформовані за виразами:

$$w_5 = w_4 \oplus w_1; \quad w_6 = w_5 \oplus w_2; \quad w_7 = w_6 \oplus w_3$$

або



Отже, з урахуванням чотирьохбайтових слів ключа шифру K та сформованих чотирьохбайтових слів: w_4, w_5, w_6, та w_7, слова розширеного ключа виглядатимуть наступним чином

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
2B	28	AB	09	A0	88	23	2A
7E	AE	F7	CF	FA	54	A3	6C
15	D2	15	4F	FE	2C	39	76
16	A6	88	3C	17	B1	39	05

Алгоритм розширення ключа повинен забезпечувати стійкість проти наступних типів атак:

– атак, в яких частина початкового ключа шифрування криптографічному аналітику відома;

– атак, в яких ключ шифрування відомий заздалегідь або може бути вибраний, наприклад, якщо шифр застосовується для стиснення даних при хешуванні;

– атак “еквівалентних ключів” [4, 36, 40]; необхідною умовою стійкості до таких атак є відсутність занадто великих однакових наборів раундових ключів, отриманих із двох різних початкових ключів шифрування.

Процедура розширення ключа також відіграє важливу роль у виключенні:

– симетрії однораундового перетворення, яке обробляє усі вхідні байти однаково; для її усунення в процедурі розширення ключа при отриманні кожного першого 32-розрядного слова раундового ключа використовуються функція SubWord (RotWord) і раундова константа;

– міжраундової симетрії (раундове перетворення однакове для всіх ітерацій циклу перетворення); для її порушення в алгоритм роширення ключа введені константи, значення яких різні для кожного раунду.

Таким чином, алгоритм розширення ключа вибраний у відповідності з наступними критеріями:

– оборотність використовуваних перетворень, тобто з будь-яких

чотирьох (для варіанту AES-128) послідовних слів розширеного ключа можна однозначно відновити увесь розширений ключ;

– висока швидкість на різних типах процесорів;

– наявність раундових констант для зменшення симетричності;

– ефективне розсіювання змін у початковому ключі шифрування на раундовий ключ, що формується;

– відсутність можливості на основі знання частини біт раундового або початкового ключа обчислити значну частину інших біт;

– достатня нелінійність для попередження повного визначення міжбітових залежностей розширеного раундового ключа лише на підставі знання таких залежностей у початковому ключі шифрування;

– простота опису.

Було обрано байт-орієнтовану схему алгоритму, яка може ефективно реалізовуватися на 8-розрядних процесорах. Застосування перетворення SubBytes забезпечує нелінійність і вимагає невеликого додаткового простору пам’яті на 8-розрядних процесорах.

Раундові ключі для розшифрування використовуються у зворотному порядку відносно раундових ключів для шифрування.

7.6. Шифрування та розшифрування даних в AES

Розглянемо, як AES використовує чотири перетворення для шифрування та розшифрування даних. У стандарті алгоритм шифрування згадується як *шифр*, а алгоритм розшифрування – як *обернений шифр*. AES відноситься до шифрів не-Фейстеля, а це означає, що кожне перетворення або група перетворень повинно бути оберненим. Крім того, шифр та обернений шифр повинні використовувати перетворення таким чином, щоб вони відміняли одне одного. Раундові ключі повинні використовуватися в оберненому порядку. Розглянемо два різних проекти для AES-128, які можуть бути використані для різних реалізацій.

Початковий проект

У початковому проекті порядок перетворень у кожному раунді при шифруванні та розшифруванні не співпадає. Рис. 7.22 демонструє цю версію.

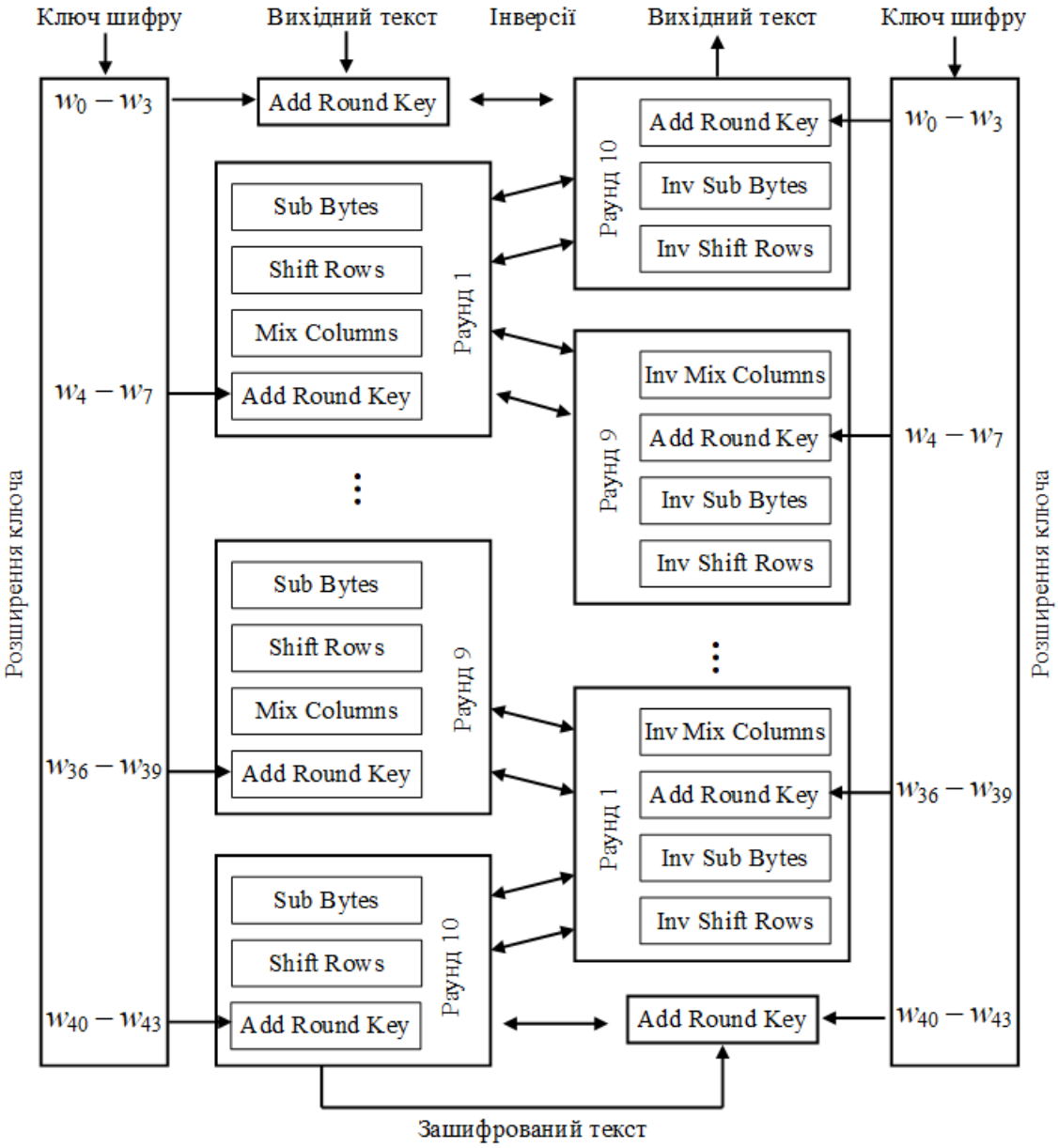


Рис. 7.22. Шифр та обернений шифр початкового проекту

Як видно з рис. 7.22, по-перше, у оберненому шифрі змінюється порядок виконання перетворень SubBytes (InvSubBytes) і ShiftRows (InvShiftRows); по-друге, у оберненому шифрі також змінено порядок виконання перетворень MixColumns (InvMixColumns) та AddRoundKey. Ці

зміни необхідні, щоб у оберненому шифрі зробити порядок виконання перетворень інверсним у відношенні до порядку прямого шифру. Отже, алгоритм розшифрування в цілому – інверсія алгоритму шифрування. На рисунку показано тільки три раунди, але інші мають той самий вигляд. Необхідно звернути увагу, що ключі раундів при розшифруванні використовуються в зворотному порядку, а також на те, що алгоритми шифрування й розшифрування у початковому проекті не збігаються.

Альтернативний проект

Для тих додатків, яким потрібні однакові алгоритми шифрування й розшифрування, було розроблено альтернативний проект розшифрування. У цьому проекті перетворення оберненого шифру перебудовані так, щоб зробити їх порядок таким самим, як і при шифруванні. У цієї версії забезпечується оберненість для пари перетворень, а не для кожного перетворення окремо.

Пари SubBytes / ShiftRows

Перетворення SubBytes змінює зміст кожного байта, не змінюючи порядок байтів матриці стану; ShiftRows змінює порядок байтів у матриці стану, не змінюючи зміст байтів. Тому, можна змінити порядок використання цих двох перетворень при розшифруванні, не зачіпаючи оберненість усього алгоритму. Рис. 7.23 ілюструє цю ідею. Зверніть увагу, що комбінація двох перетворень – у шифрі та оберненому шифрі – інверсні одна одній.

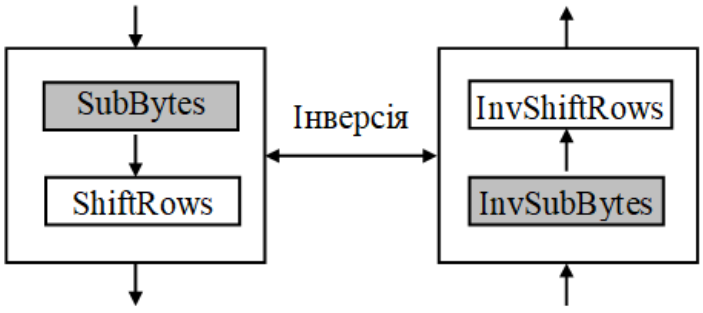


Рис. 7.23. Оберненість пари перетворень SubBytes і ShiftRows

Пари MixColumns / AddRoundKey

Перетворення MixColumns і AddRoundKey мають різні властивості, притаманні тільки їм. Однак ці перетворення можуть стати інверсіями один одного, якщо помножити матрицю раундових ключів на інверсію матриці констант, що використовується в перетворенні MixColumns. Назвемо нове перетворення InvAddRoundKey. Рис. 7.24 показує нову конфігурацію.

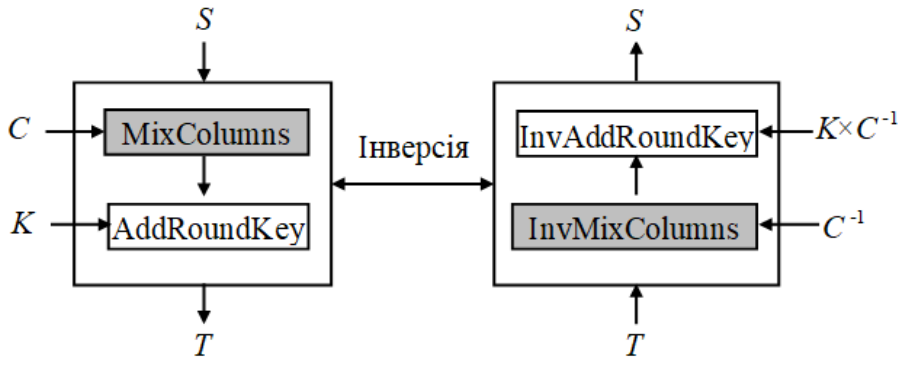


Рис. 7.24. Оберненість пари перетворень MixColumns і AddRoundKey

Можна доказати, що ці комбінації інверсні одна одній. При шифруванні вхідну матрицю стану позначимо як *S*, а вихідну – *T*. При розшифруванні вхідну матрицю стану позначимо як *T*. Покажемо, що в такому випадку вихідна матриця стану – *S*. Необхідно звернути увагу на те, що перетворення MixColumns – фактично результат множення матриці констант *C* на матрицю стану *S*. Отже, результат шифрування – матриця стану *T*:

$$T = C \times S \oplus K.$$

При розшифруванні:

$$C^{-1} \times \mathrm{T} \oplus C^{-1} \times K = C^{-1} \times (C \times S \oplus K) \oplus C^{-1} \times K =$$
$$= C^{-1} \times C \times S \oplus C^{-1} \times K \oplus C^{-1} \times K = S.$$

Таким чином, отримали альтернативний проект, який показано на рис. 7.25. Слід звернути увагу, що при розшифруванні необхідно двічі використовувати перетворення AddRoundKey та дев’ять разів InvAddRoundKey.

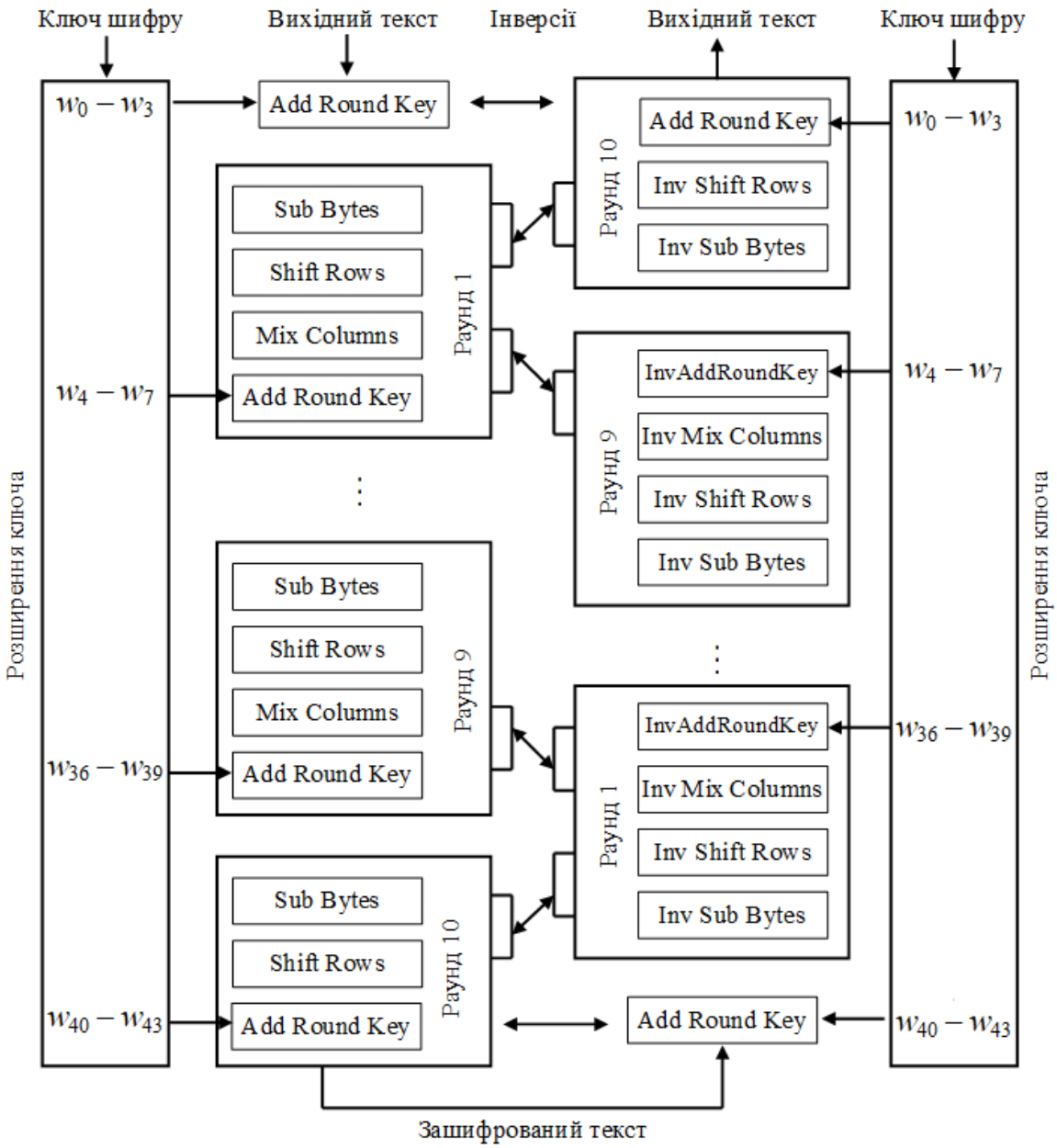


Рис. 7.25. Шифр та обернений шифр альтернативного проекту

7.7. Безпека AES

AES був розроблений після DES. Більшість відомих атак на DES було перевірено на AES; жодна з них до теперішнього часу не порушила безпеку AES.

Атака "грубої сили"

AES явно безпечніший ніж DES, через великий розмір ключа (128, 192 та 256 біт). Порівняємо DES з ключем шифру на 56 біт та AES з ключем

шифру на 128 біт. Для DES, щоб знайти ключ, необхідно виконати 2^{56} випробувань; для AES – 2^{128} випробувань. Це означає, що якщо можливо порушити DES за t секунд, то необхідно $(2^{72} \times t)$ секунд, щоб порушити AES. Це майже неможливо. Крім того, AES забезпечує дві інші версії з більш довшими ключами шифру.

Статистичні атаки

Сильне розсіювання та перемішування, яке забезпечується комбінацією перетворень ShiftRows і MixColumns, видаляють будь-яку частотну закономірність у вихідному тексті. Численні спроби виконати статистичний аналіз зашифрованого тексту не мали успіху. Рис. 7.26 демонструє розсіювальні й перемішувальні властивості шифру. Видно, що навіть два раунди забезпечують повне розсіювання і перемішування.

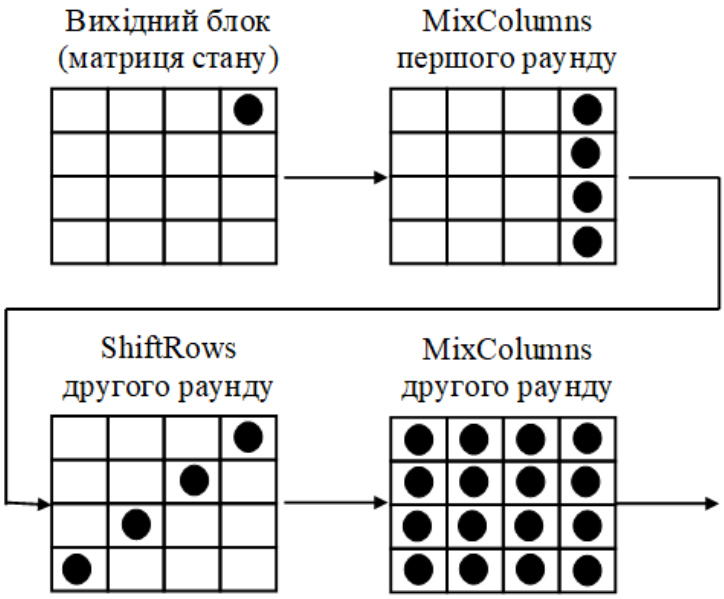


Рис. 7.26. Демонстрація властивостей розсіювання і перемішування AES

– змінений байт

Диференціальні та лінійні атаки

AES був розроблений після DES. Диференціальні та лінійні атаки криптоаналізу були, без сумніву, враховані. Подібні атаки на AES поки не виявлені.

Контрольні питання до розділу 7

1. Перерахуйте параметри (розмір блока, довжину ключа та кількість раундів) для трьох версій AES.

2. Скільки перетворень використовується в кожній версії AES? Скільки раундових ключів необхідно для кожної версії?

3. Порівняйте DES і AES. Який з них орієнтований на роботу з бітом, а який – на роботу з байтом?

4. Визначте матрицю стану в AES.

5. Які із чотирьох перетворень, визначених для AES, змінюють зміст байтів, а які не змінюють?

6. Порівняйте підстановку в DES і AES. Чому в AES є тільки одна таблиця підстановки і декілька в DES?

7. Порівняйте перестановки в DES і AES. Чому треба мати розширення й стиснення перестановки в DES і не треба в AES?

8. Порівняйте ключі раунду в DES і AES. У якому шифрі розмір ключа раунду дорівнює розміру блоку?

9. Чому перетворення MixColumns, яке змішує дані, необхідне в AES, але не потрібне в DES?

10. Які особливості розшифрування за алгоритмом AES?

11. Значення байтів матриці стану даних на вході функції SubBytes криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: F09C7FF2689F352B6B5BEA43026A5049. Визначити значення байтів матриці стану даних на виході функції SubBytes.

12. Значення байтів матриці стану даних на вході функції InvSubBytes криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: C396DB453B53027789D2DE491A87397F. Визначити значення байтів матриці стану даних на виході функції InvSubBytes.

13. Значення байтів матриці стану даних на вході функції ShiftRows криптографічної системи AES-128 в шістнадцятковій системі дорівнюють:

31DED28945DB96F17F39871A7702533B. Визначити значення байтів матриці стану даних на виході функції ShiftRows.

14. Значення байтів матриці стану даних на вході функції InvShiftRows криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: AB96DB453B53027789D2DE491A8F3971. Визначити значення байтів матриці стану даних на виході функції InvShiftRows.

15. Значення байтів матриці стану даних на вході функції MixColumns криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: FFDB873B4539538A7F02D2F177DE96A1. Визначити значення байтів матриці стану даних на виході функції MixColumns.

16. Значення байтів матриці стану даних на вході функції InvMixColumns криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: CDAFB22485C967CAEDD5E4A2FF9E5E4A. Визначити значення байтів матриці стану даних на виході функції InvMixColumns.

17. Значення байтів матриці стану даних на вході функції AddRoundKey криптографічної системи AES-128 в шістнадцятковій системі дорівнюють: 254DCAF1EB4B5AACDBE7CAA81B6DB9E1. Значення байтів матриці стану раундового ключа на вході функції AddRoundKey – FEC295F27A9BB9745935807A7359F6ED. Визначити значення байтів матриці стану на виході функції AddRoundKey.

18. Значення байтів матриці стану раундового ключа криптографічної системи AES-128 для дев’ятого раунду шифрування в шістнадцятковій системі дорівнюють: 23AC76F319FADC23E8D12941575C071E. Визначити значення байтів матриці стану раундового ключа для десятого раунду.

РОЗДІЛ 8
СТАНДАРТ СИМЕТРИЧНОГО БЛОКОВОГО ПЕРЕТВОРЕННЯ ДАНИХ ДСТУ 7624:2014

8.1. Загальні положення ДСТУ 7624:2014

У якості основного стандарту блокового шифрування в Україні з 1990 року використовувався ДСТУ ГОСТ 28147:2009 (ГОСТ 28147-89) [17]. Зараз це перетворення ще забезпечує практичну стійкість у режимах забезпечення конфіденційності, водночас, з точки зору швидкодії реалізації на програмних платформах загального призначення, суттєво поступається більш сучасним рішенням, таким як AES, що призводить до ускладнення та подорожчання засобів криптографічного захисту інформації при однакових інших характеристиках. Крім того, для ГОСТ 28147-89 відомі теоретичні атаки, більш ефективні, ніж повний перебір ключів [29, 33].

Враховуючи сучасні міжнародні тенденції та позитивний досвід у галузі розробки перспективних криптографічних перетворень, Державна служба спеціального зв'язку та захисту інформації України (ДССЗЗІ) успішно провела національний відкритий конкурс симетричних блокових криптографічних алгоритмів. По результатам конкурсу був відзначений алгоритм "Калина", на базі якого був розроблений національний стандарт ДСТУ 7624:2014. Цей стандарт був прийнятий наказом Мінекономрозвитку від 29 грудня 2014 року № 1484 "Про прийняття європейських стандартів як національних стандартів України та скасування національних стандартів України" та введений в дію з 01 липня 2015 року [7].

Алгоритм шифрування "Калина" виконує криптографічне перетворення блоків інформації розміром 128, 256 і 512 біт, використовуючи ключ шифрування довжиною 128, 256 і 512 бітів. Довжина ключа співпадає з розміром блока або в два рази перебільшує його. Допустимі комбінації розміру блока та довжини ключа шифрування наведено у табл. 8.1.

Таблиця 8.1

Комбінації розміру блока та довжини ключа шифрування

Розмір блоку, біт	Довжина ключа, біт
128 ($N_b = 2$)	128 ($N_k = 2$), 256 ($N_k = 4$)
256 ($N_b = 4$)	256 ($N_k = 4$), 512 ($N_k = 8$)
512 ($N_b = 8$)	512 ($N_k = 8$)

У табл. 8.1: N_b – розмір блоку відкритого тексту, виражений 64-бітовими елементами (може набувати значень 2, 4 або 8 відповідно для 128-бітного, 256-бітного або 512-бітного блоку); N_k – довжина ключа шифрування, виражена 64-бітовими елементами (може набувати значень 2, 4 або 8 відповідно для 128-бітного, 256-бітного або 512-бітного ключа)

Алгоритм шифрування є процедурою, що складається з попереднього і прикінцевого забілювання та ітераційного раундового перетворення. На вхід кожного раунду подається поточний стан, а також необхідна кількість ключової інформації (ключ раунду). Відкритий текст копіюється в поточний стан перед початком шифрування, а після його завершення в поточному стані знаходиться зашифрований текст. Кількість раундів шифрування (N_r) залежить від довжини ключа та розміру блоку (дивись табл. 8.2).

Таблиця 8.2

Кількість раундів шифрування алгоритму “Калина” (N_r)

Довжина ключа (біт) / Розмір блока (біт)	128 ($N_k = 2$)	256 ($N_k = 4$)	512 ($N_k = 8$)
128 ($N_b = 2$)	10	14	–
256 ($N_b = 4$)	–	14	18
512 ($N_b = 8$)	–	–	18

Кожний раунд, крім останнього, використовує чотири перетворення (SubBytes, ShiftRows, MixColumns та AddRoundKey), які є оберненими.

Останній раунд має тільки три перетворення (перетворення AddRoundKey відсутнє). Попереднім та кінцевим перетворенням є операція додавання за модулем 2^{64}. На рис. 8.1 показано загальну структуру ДСТУ 7624:2014.

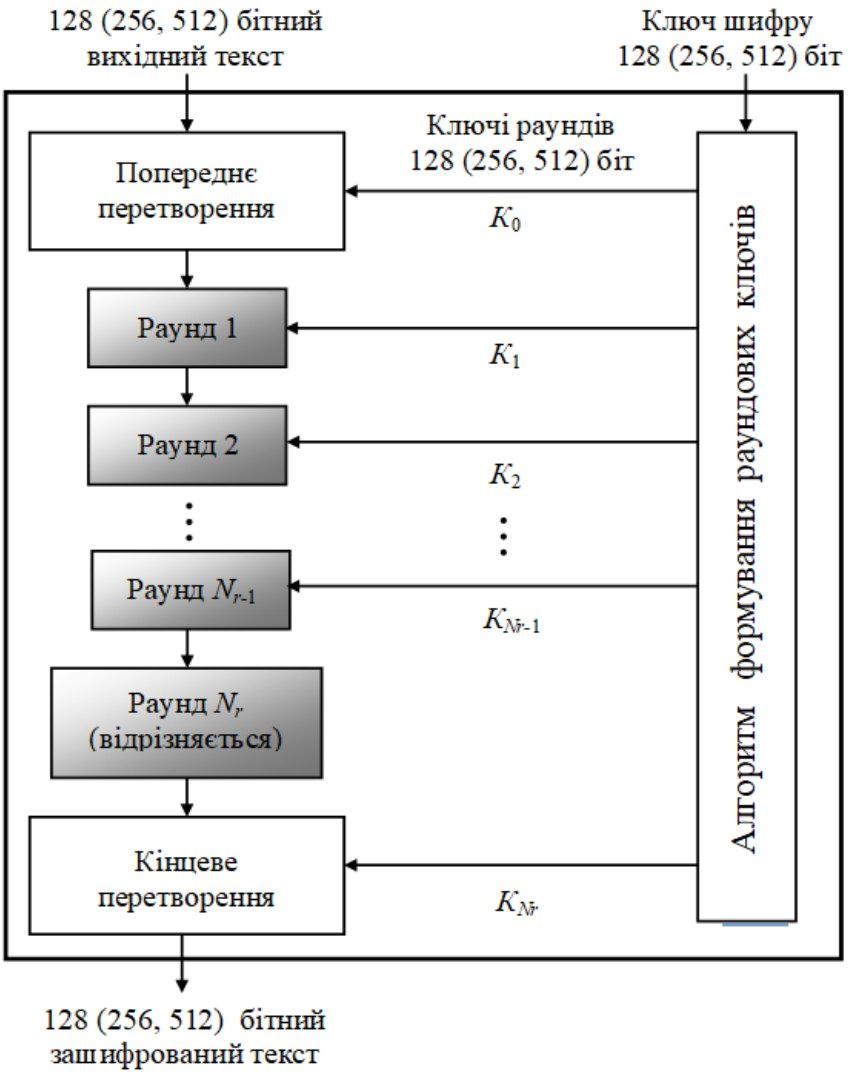


Рис. 8.1. Загальна структура ДСТУ 7624:2014

8.2. Формат даних ДСТУ 7624:2014

До вхідних та вихідних даних алгоритму “Калина” належать відкритий текст та зашифрований тексти відповідно. Ці параметри представляються у вигляді рядків заданої довжини $8\times N_b$ байт ($64 \times N_b$ біт). Додатковим вхідним параметром є ключ, розмір якого дорівнює $8\times N_k$ байт ($64 \times N_k$ біт).

При виконанні шифрування або розшифрування операції виконуються

над двомірним масивом байт (поточним станом шифру). Поточний стан можна представити у вигляді матриці розмірністю $8 \times N_b$ байт (вісім рядків довжиною N_b байт). Кожний байт в поточному стані має два індекси: номер рядка (r, $0 \le r < 8$) і номер стовпця (c, $0 \le c < N_b$). Байт адресується як $S_{r,c}$ або $S[r, c]$.

Крім того, у ряді операцій поточний стан шифру $S = (s_{i,j})$ розглядається як послідовність 64-бітових слів $S = (S_i)$. У цьому випадку кожний стовпець з номером i, $1 \le i < N_b$, розглядається як окреме слово, що має значення $S_i = s_{0,i} \cdot 2^{0 \cdot 8} + s_{1,i} \cdot 2^{1 \cdot 8} + \ldots + s_{7,i} \cdot 2^{7 \cdot 8}$. Відповідно, молодшим бітом 64-бітового блока є молодший біт байта рядка з найменшим номером, старшим бітом 64-бітового блока – старший біт байта із рядка з найбільшим номером.

Заповнення поточного стану

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

Довжина блока 128 біт

Відкритий текст представлено байтовою послідовністю in_0, in_1, …, in_{15}. Отриманий зашифрований текст представлено як послідовність байт out_0, out_1, …, out_{15}. Заповнення поточного стану шифру перед початком шифрування та після його закінчення представлено на рис. 8.2.

Вхідна послідовність	
in_0	in_8
in_1	in_9
in_2	in_{10}
in_3	in_{11}
in_4	in_{12}
in_5	in_{13}
in_6	in_{14}
in_7	in_{15}

→

Поточний стан	
$s_{0,0}$	$s_{0,1}$
$s_{1,0}$	$s_{1,1}$
$s_{2,0}$	$s_{2,1}$
$s_{3,0}$	$s_{3,1}$
$s_{4,0}$	$s_{4,1}$
$s_{5,0}$	$s_{5,1}$
$s_{6,0}$	$s_{6,1}$
$s_{7,0}$	$s_{7,1}$

→

Вихідна послідовність	
out_0	out_8
out_1	out_9
out_2	out_{10}
out_3	out_{11}
out_4	out_{12}
out_5	out_{13}
out_6	out_{14}
out_7	out_{15}

Рис. 8.2. Заповнення поточного стану для довжини блока 128 біт

Довжина блока 256 біт

Відкритий текст представлено байтовою послідовністю in_0, in_1, …, in_{31}. Отриманий зашифрований текст представлено як послідовність байт out_0, out_1, …, out_{31}. Заповнення поточного стану шифру перед початком шифрування та після його закінчення представлено на рис. 8.3.

Вхідна послідовність			
in_0	in_8	in_{16}	in_{24}
in_1	in_9	in_{17}	in_{25}
in_2	in_{10}	in_{18}	in_{26}
in_3	in_{11}	in_{19}	in_{27}
in_4	in_{12}	in_{20}	in_{28}
in_5	in_{13}	in_{21}	in_{29}
in_6	in_{14}	in_{22}	in_{30}
in_7	in_{15}	in_{23}	in_{31}

↓

Поточний стан			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$
$s_{4,0}$	$s_{4,1}$	$s_{4,2}$	$s_{4,3}$
$s_{5,0}$	$s_{5,1}$	$s_{5,2}$	$s_{5,3}$
$s_{6,0}$	$s_{6,1}$	$s_{6,2}$	$s_{6,3}$
$s_{7,0}$	$s_{7,1}$	$s_{7,2}$	$s_{7,3}$

↓

Вихідна послідовність			
out_0	out_8	out_{16}	out_{24}
out_1	out_9	out_{17}	out_{25}
out_2	out_{10}	out_{18}	out_{26}
out_3	out_{11}	out_{19}	out_{27}
out_4	out_{12}	out_{20}	out_{28}
out_5	out_{13}	out_{21}	out_{29}
out_6	out_{14}	out_{22}	out_{30}
out_7	out_{15}	out_{23}	out_{31}

Рис. 8.3. Заповнення поточного стану для довжини блока 256 біт

Довжина блока 512 біт

Відкритий текст представлено байтовою послідовністю in_0, in_1, …, in_{63}. Отриманий зашифрований текст представлено як послідовність байт out_0, out_1, …, out_{63}. Заповнення поточного стану шифру перед початком шифрування та після його закінчення представлено на рис. 8.4.

Вхідна послідовність							
in_0	in_8	in_{16}	in_{24}	in_{32}	in_{40}	in_{48}	in_{56}
in_1	in_9	in_{17}	in_{25}	in_{33}	in_{41}	in_{49}	in_{57}
in_2	in_{10}	in_{18}	in_{26}	in_{34}	in_{42}	in_{50}	in_{58}
in_3	in_{11}	in_{19}	in_{27}	in_{35}	in_{43}	in_{51}	in_{59}
in_4	in_{12}	in_{20}	in_{28}	in_{36}	in_{44}	in_{52}	in_{60}
in_5	in_{13}	in_{21}	in_{29}	in_{37}	in_{45}	in_{53}	in_{61}
in_6	in_{14}	in_{22}	in_{30}	in_{38}	in_{46}	in_{54}	in_{62}
in_7	in_{15}	in_{23}	in_{31}	in_{39}	in_{47}	in_{55}	in_{63}

↓

Поточний стан							
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	$s_{0,4}$	$s_{0,5}$	$s_{0,6}$	$s_{0,7}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,4}$	$s_{1,5}$	$s_{1,6}$	$s_{1,7}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	$s_{2,4}$	$s_{2,5}$	$s_{2,6}$	$s_{2,7}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	$s_{3,5}$	$s_{3,6}$	$s_{3,7}$
$s_{4,0}$	$s_{4,1}$	$s_{4,2}$	$s_{4,3}$	$s_{4,4}$	$s_{4,5}$	$s_{4,6}$	$s_{4,7}$
$s_{5,0}$	$s_{5,1}$	$s_{5,2}$	$s_{5,3}$	$s_{5,4}$	$s_{5,5}$	$s_{5,6}$	$s_{5,7}$
$s_{6,0}$	$s_{6,1}$	$s_{6,2}$	$s_{6,3}$	$s_{6,4}$	$s_{6,5}$	$s_{6,6}$	$s_{6,7}$
$s_{7,0}$	$s_{7,1}$	$s_{7,2}$	$s_{7,3}$	$s_{7,4}$	$s_{7,5}$	$s_{7,6}$	$s_{7,7}$

↓

Вихідна послідовність							
out_0	out_8	out_{16}	out_{24}	out_{32}	out_{40}	out_{48}	out_{56}
out_1	out_9	out_{17}	out_{25}	out_{33}	out_{41}	out_{49}	out_{57}
out_2	out_{10}	out_{18}	out_{26}	out_{34}	out_{42}	out_{50}	out_{58}
out_3	out_{11}	out_{19}	out_{27}	out_{35}	out_{43}	out_{51}	out_{59}
out_4	out_{12}	out_{20}	out_{28}	out_{36}	out_{44}	out_{52}	out_{60}
out_5	out_{13}	out_{21}	out_{29}	out_{37}	out_{45}	out_{53}	out_{61}
out_6	out_{14}	out_{22}	out_{30}	out_{38}	out_{46}	out_{54}	out_{62}
out_7	out_{15}	out_{23}	out_{31}	out_{39}	out_{47}	out_{55}	out_{63}

Рис. 8.4. Заповнення поточного стану для довжини блока 512 біт

8.3. Шифрування даних в ДСТУ 7624:2014

Процедура шифрування одного блоку даних алгоритму “Калина” представлено на рис. 8.5.

Базове перетворення шифрування $T_{l,k}^{(K)}$ визначається наступним чином:

$$T_{l,k}^{(K)} = \eta_l^{(K_{Nr})} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \left(\prod_{\nu=1}^{N_r-1} \left(K_l^{(K_\upsilon)} \circ \psi_l \circ \tau_l \circ \pi_l' \right) \right) \circ \eta_l^{(K_0)},$$

де l – розмір блоку в бітах (128, 256 або 512);

k – довжина ключа шифрування в бітах (128, 256 або 512);

K – ключ шифрування;

$\eta_l^{(K_v)}$ – функція додавання раундового ключа K_v ($v \in \{0, N_r\}$) за модулем 2^{64};

$\pi_l^{'}$ – байтова підстановка (перетворення SubBytes);

τ_l – перестановка елементів поточного стану (перетворення ShiftRows);

ψ_l – множення матриці лінійного перетворення на матрицю поточного стану над скінченним полем (перетворення MixColumns);

$K_l^{(K_v)}$ – функція додавання раундового ключа K_v ($v \in \{1, 2, \ldots, N_r\text{-}1\}$) за модулем 2 (перетворення AddRoundKey).

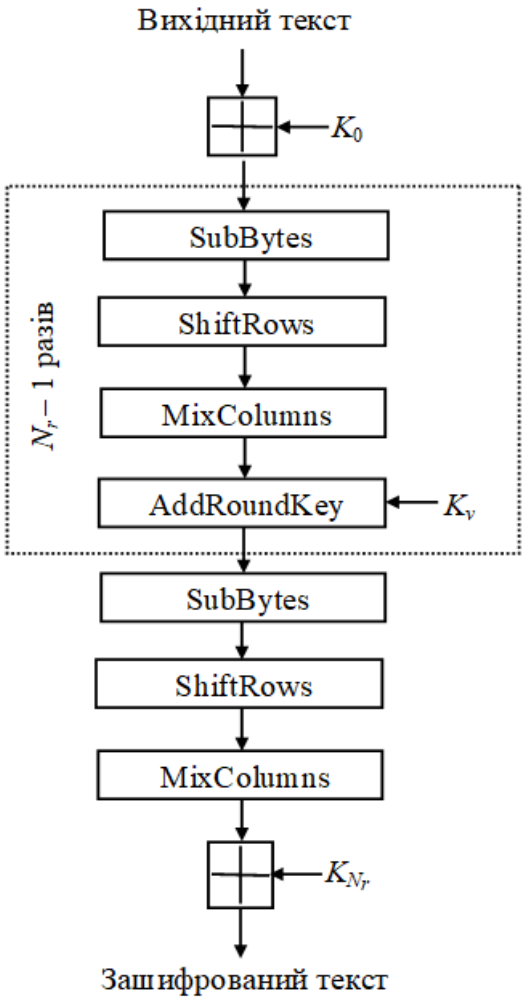


Рис. 8.5. Процедура шифрування алгоритму “Калина”

Розглянемо кожне перетворення окремо.

Функція додавання раундового ключа K_v за модулем 2^{64}

Перетворення додавання раундового ключа за модулем 2^{64} виконує додавання за модулем 2^{64} стовпців матриці поточного стану (64-бітових слів) та стовпців ключа раунду. Після виконання операції результат записується на місце першого аргументу.

Представлення ключа раунду як матриці відповідного розміру є аналогічним представленню відкритого тексту у вигляді поточного стану шифру (дивись п. 8.2). При розбитті поточного стану та ключа раунду на 64-бітові 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. Аналогічним чином проходить розбиття ключа раунду на 64-бітові блоки, після чого виконується додавання відповідних блоків за модулем 2^{64}.

Для поточного стану $A = (a_i)$ ($a_i = a_{0,i} \cdot 2^{0 \cdot 8} + a_{1,i} \cdot 2^{1 \cdot 8} + ... + a_{7,i} \cdot 2^{7 \cdot 8}$; $0 \le i < N_b$) і ключа раунду $K = (k_i)$, результат перетворення $B = (b_i)$, обчислюється за формулою $b_i = a_i + k_i \pmod{2^{64}}$.

Схема розбиття на 64-бітові блоки 128-бітного поточного стану та ключа раунду такого ж розміру з наступним перетворенням додавання за модулем 2^{64} наведено на рис. 8.6.

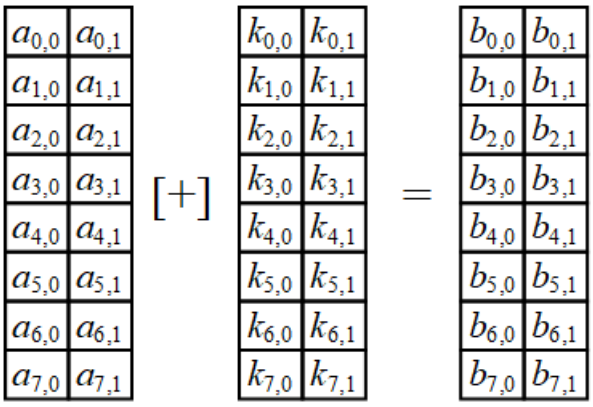


Рис. 8.6. Перетворення додаванням за модулем 2^{64} для розміру блока 128 біт

Перетворення додавання за модулем 2^{64} для поточних станів розміром 256 або 512 біт буде відрізнятися лише кількістю стовпців, на які розбивається стан та ключ раунду (приклад для 256-бітного стану наведено на рис. 8.7).

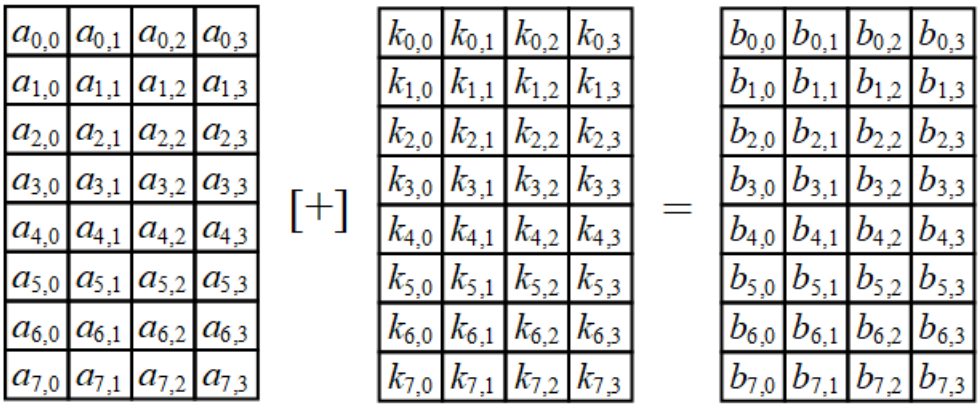


Рис. 8.7. Перетворення додаванням за модулем 2^{64} для розміру блока 256 біт

Перетворення SubBytes

Перетворення SubBytes виконує заміну кожного байта поточного стану у відповідності з заданою таблицею підстановки. Використовуються чотири 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 S_0;

для байт 1-го рядка (елементи $s_{1,i}$) – підстановка S_1;

для байт 2-го рядка (елементи $s_{2,i}$) – підстановка S_2;

для байт 3-го рядка (елементи $s_{3,i}$) – підстановка S_3;

для байт 4-го рядка (елементи $s_{4,i}$) – підстановка S_0;

для байт 5-го рядка (елементи $s_{5,i}$) – підстановка S_1;

для байт 6-го рядка (елементи $s_{6,i}$) – підстановка S_2;

для байт 7-го рядка (елементи $s_{7,i}$) – підстановка S_3.

Таблиці підстановок алгоритму “Калина” наведено в табл. 8.3 – 8.6.

Таблиця 8.3

Підстановка S_0

	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	1A	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	2A	85	25	E6	CA	7C	8B	56	80

Таблиця 8.4

Підстановка S_1

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

Таблиця 8.5

Підстановка S_2

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

Таблиця 8.6

Підстановка S_3

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

На рис. 8.8 наведено порядок виконання перетворення SubBytes для розміру блока 256 біт.

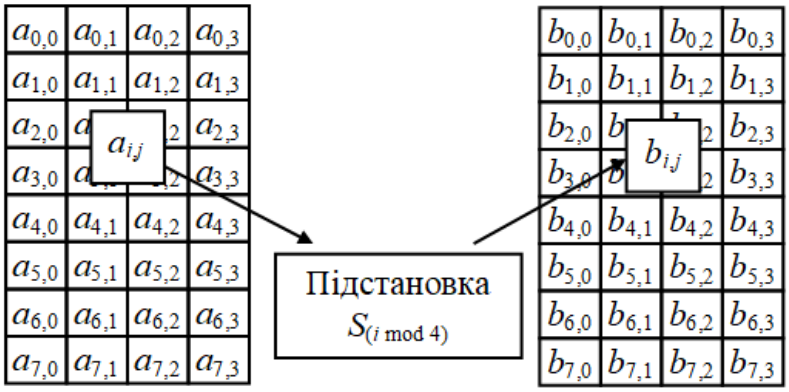


Рис. 8.8. Порядок перетворення SubBytes для розміру блока 256 біт

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

Приклад 8.1. Виконати перетворення SubBytes для таблиці S_0 байта з шістнадцятковим значенням 3D.

Рішення. Старші 4 біти визначають рядок, молодші 4 біти – стовпець. Результат підстановки для значення 3D – це число в шістнадцятковому представленні A4, що знаходиться в таблиці на перетині 4-го рядка (з індексом 3) та 14-го стовпця (з індексом D) (дивись табл. 8.3).

Для здійснення перетворення SubBytes може використовуватися інший набір підстановок (від 1 до 8 таблиць), відмінний від наведеного. У цьому випадку набір підстановок має постачатися в установленому порядку і може бути додатковим секретним параметром шифру, як і ключ шифрування.

Перетворення ShiftRows

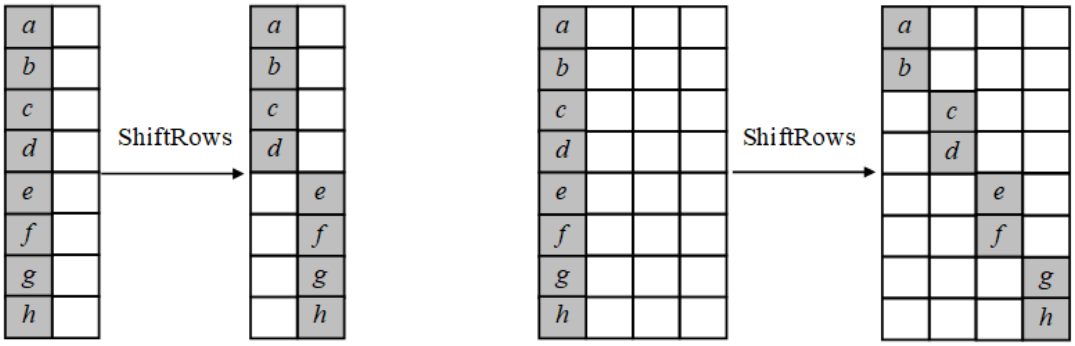
Перетворення ShiftRows здійснює рівномірне розподілення байт кожного 64-бітного стовпця серед інших стовпців. Це досягається шляхом циклічного зсуву вправо рядків поточного стану на різну кількість байт.

Значення зсувів залежать від номеру рядку та розміру блока шифрування і представлені у табл. 8.7. На рис. 8.9 наведено порядок розподілення байт першого стовпця при здійсненні перетворення ShiftRows для різних розмірів блоку.

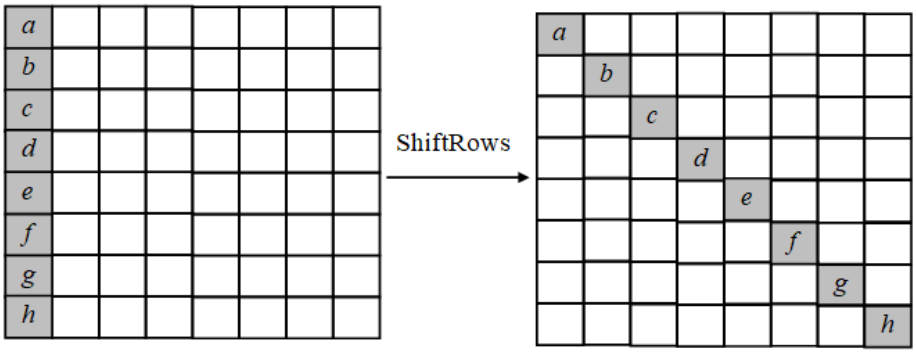
Таблиця 8.7

Значення циклічних зсувів рядків для різних розмірів блока

Номер рядка	Значення зсуву, байти		
	Довжина блоку128 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



в) 512-бітний блок

Рис. 8.9. Порядок розподілення байт першого стовпця при виконанні перетворення ShiftRows

Перетворення MixColumns

Під час здійснення перетворення MixColumns виконується послідовна обробка всіх стовпців поточного стану. Ця операція еквівалентна матричному множенню у полі $GF(2^8)$ початкового 8-байтового стовпця (вектор a) на фіксовану матрицю C, результат зберігається у новий 8-байтний стовпець (вектор b) (рис. 8.10).

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

Рис. 8.10. Матричне представлення перемішування у стовпці

Порядок обчислення елементів результуючого вектора b пояснюється на рис. 8.11, при цьому всі операції множення на байт виконуються у полі $GF(2^8)$ з незвідним поліномом $x^8+x^4+x^3+x+1$.

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

Рис. 8.11. Порядок обчислення байт при виконанні перемішування в стовпці

На рис. 8.12 наведено порядок виконання перетворення MixColumns для поточного стану шифру з розміром блоку 256 біт.

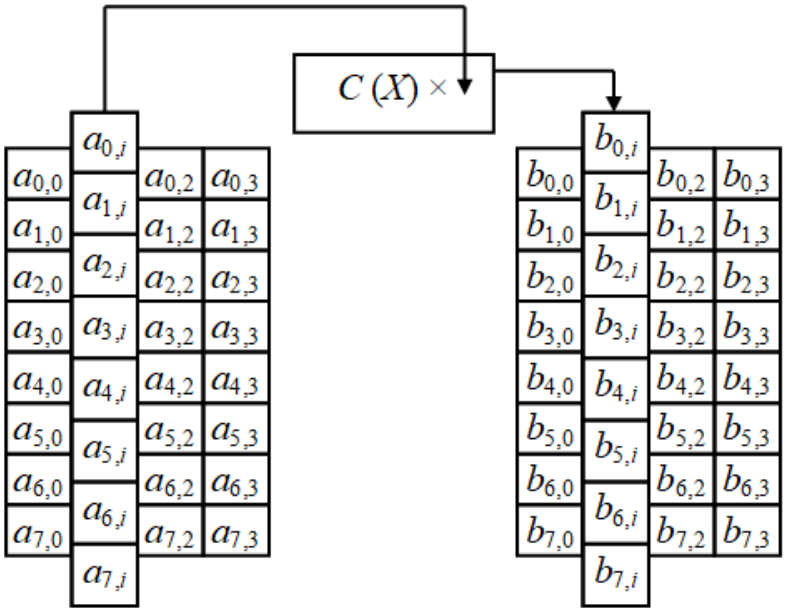


Рис. 8.12. Порядок виконання перетворення MixColumns для поточного стану шифру з розміром блоку 256 біт

Перетворення AddRoundKey

Перетворення AddRoundKey виконує побітове додавання за модулем два поточного стану та ключа раунду. Після виконання операції результат записується на місце першого аргументу.

Представлення ключа раунду як матриці відповідного розміру є аналогічним представленню відкритого тексту у вигляді поточного стану шифру (дивись п. 8.2).

Для поточного стану $A = (a_{i,j})$ та раундового 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 AddRoundKey для розміру блока 256 біт представлено на рис. 8.13.

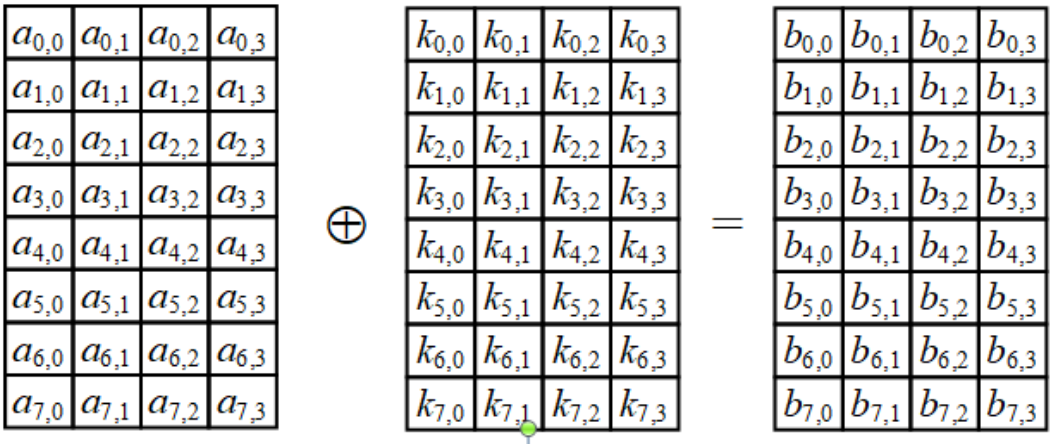


Рис. 8.13. Перетворення AddRoundKey для розміру блока 256 біт

8.4. Розшифрування даних в ДСТУ 7624:2014

Процедура розшифрування є оберненою шифруванню. На вхід подається зашифрований текст та ключі раундів. Після закінчення розшифрування отриманий відкритий текст формується у вигляді байтового рядка. Процедура розшифрування одного блоку даних алгоритму “Калина” представлено на рис. 8.14.

Базове перетворення розшифрування $C_{l,k}^{(K)}$ визначається наступним чином:

$$C_{l,k}^{(K)} = {}_{-1}\eta_l^{(K_0)} \circ \left(\prod_{v=N_r-1}^{1} \left({}_{-1}\pi_l' \circ {}_{-1}\tau_l \circ {}_{-1}\psi_l \circ K_l^{(K_v)} \right) \right) \circ {}_{-1}\pi_l' \circ {}_{-1}\tau_l \circ {}_{-1}\psi_l \circ {}_{-1}\eta_l^{(K_{Nr})},$$

де l – розмір блоку в бітах (128, 256 або 512);

k – довжина ключа шифрування в бітах (128, 256 або 512);

K – ключ шифрування;

${}_{-1}\eta_l^{(K_v)}$ – функція віднімання раундового ключа $K_v (v \in \{0, N_r\})$ за модулем 2^{64} (обернена до $\eta_l^{(K_v)}$);

$_{-1}\psi_l$ – множення матриці оберненого лінійного перетворення на матрицю поточного стану (перетворення InvMixColumns);

$_{-1}\tau_l$ – обернена перестановка елементів поточного стану (перетворення InvShiftRows);

$_{-1}\pi_l^{'}$ – обернена байтова підстановка (перетворення InvSubBytes);

$K_l^{(K_v)}$ – функція додавання раундового ключа K_v ($v \in \{1, 2, \ldots, N_r\text{-}1\}$) за модулем 2 (перетворення AddRoundKey).

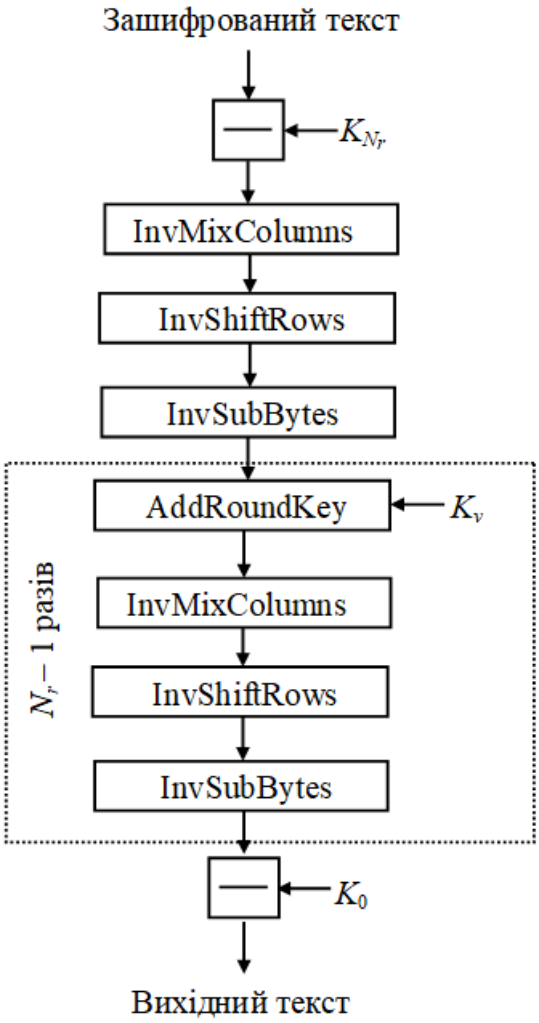


Рис. 8.14. Процедура розшифрування алгоритму “Калина”

Розглянемо кожне перетворення окремо.

Функція віднімання раундового ключа K_v за модулем 2^{64}

Виконується аналогічне розбиття поточного стану та ключа раунду на 64-бітні блоки та віднімання за модулем 2^{64} від блоків стану $B = (b_i)$ відповідних блоків раундового ключа $K = (k_i)$ для отримання нового стану $A = (a_i)$: $a_i = b_i - k_i \pmod{2^{64}}$, $0 \le i < N_b$).

Після виконання операції результат записується на місце першого параметра. Правила переходу від байт поточного стану до 64-бітових блоків та навпаки повністю ідентичні з перетвореннями, що виконуються під час додавання раундового ключа за модулем 2^{64}.

Перетворення InvMixColumns

Перетворення InvMixColumns виконує послідовну обробку всіх стовпців поточного стану. Ця операція еквівалентна матричному множенню у полі $GF(2^8)$ початкового 8-байтового стовпця (вектор a) на фіксовану матрицю (обернена до матриці, яка використовувалась при шифруванні), результат зберігається у новий 8-байтний стовпець (вектор b) (рис. 8.15).

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} \text{AD} & 95 & 76 & \text{A8} & 2\text{F} & 49 & \text{D7} & \text{CA} \\ \text{CA} & \text{AD} & 95 & 76 & \text{A8} & 2\text{F} & 49 & \text{D7} \\ \text{D7} & \text{CA} & \text{AD} & 95 & 76 & \text{A8} & 2\text{F} & 49 \\ 49 & \text{D7} & \text{CA} & \text{AD} & 95 & 76 & \text{A8} & 2\text{F} \\ 2\text{F} & 49 & \text{D7} & \text{CA} & \text{AD} & 95 & 76 & \text{A8} \\ \text{A8} & 2\text{F} & 49 & \text{D7} & \text{CA} & \text{AD} & 95 & 76 \\ 76 & \text{A8} & 2\text{F} & 49 & \text{D7} & \text{CA} & \text{AD} & 95 \\ 95 & 76 & \text{A8} & 2\text{F} & 49 & \text{D7} & \text{CA} & \text{AD} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}$$

Рис. 8.15. Матричне представлення оберненого перемішування у стовпці

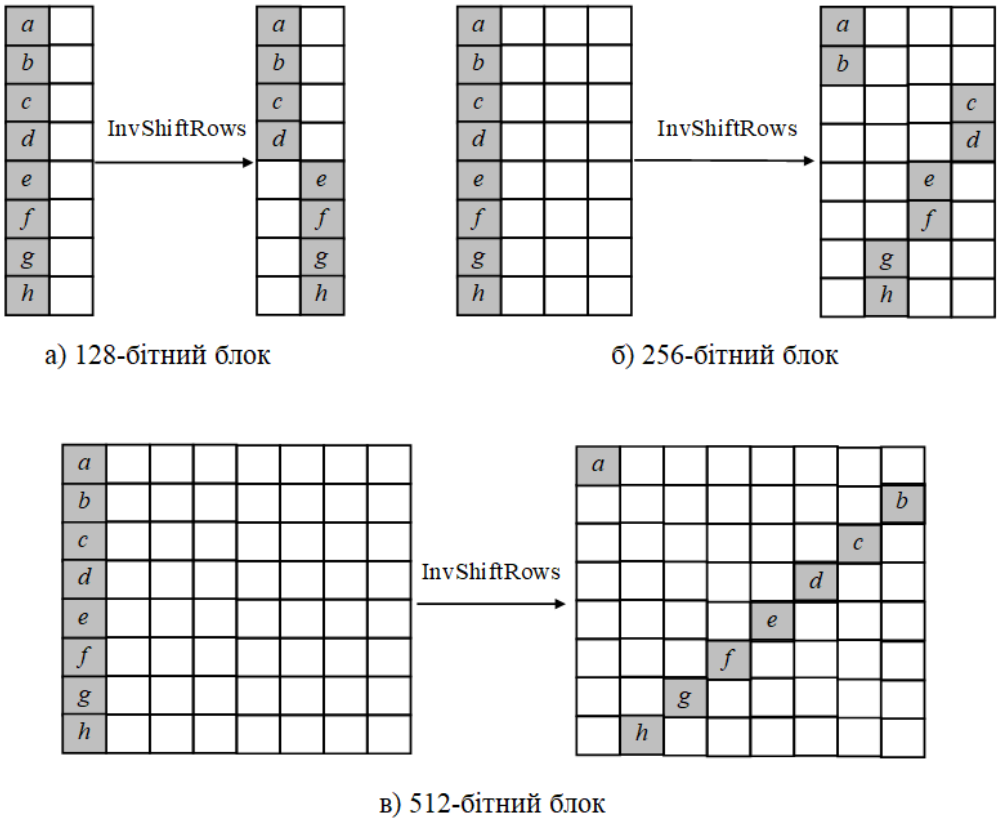
Порядок обчислення елементів результуючого вектора b пояснюється на рис. 8.16, при цьому всі операції множення на байт виконуються у полі $GF(2^8)$ з незвідним поліномом $x^8+x^4+x^3+x+1$.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} \text{AD} \cdot a_0 \oplus 95 \cdot a_1 \oplus 76 \cdot a_2 \oplus \text{A8} \cdot a_3 \oplus 2\text{F} \cdot a_4 \oplus 49 \cdot a_5 \oplus \text{D7} \cdot a_6 \oplus \text{CA} \cdot a_7 \\ \text{CA} \cdot a_0 \oplus \text{AD} \cdot a_1 \oplus 95 \cdot a_2 \oplus 76 \cdot a_3 \oplus \text{A8} \cdot a_4 \oplus 2\text{F} \cdot a_5 \oplus 49 \cdot a_6 \oplus \text{D7} \cdot a_7 \\ \text{D7} \cdot a_0 \oplus \text{CA} \cdot a_1 \oplus \text{AD} \cdot a_2 \oplus 95 \cdot a_3 \oplus 76 \cdot a_4 \oplus \text{A8} \cdot a_5 \oplus 2\text{F} \cdot a_6 \oplus 49 \cdot a_7 \\ 49 \cdot a_0 \oplus \text{D7} \cdot a_1 \oplus \text{CA} \cdot a_2 \oplus \text{AD} \cdot a_3 \oplus 95 \cdot a_4 \oplus 76 \cdot a_5 \oplus \text{A8} \cdot a_6 \oplus 2\text{F} \cdot a_7 \\ 2\text{F} \cdot a_0 \oplus 49 \cdot a_1 \oplus \text{D7} \cdot a_2 \oplus \text{CA} \cdot a_3 \oplus \text{AD} \cdot a_4 \oplus 95 \cdot a_5 \oplus 76 \cdot a_6 \oplus \text{A8} \cdot a_7 \\ \text{A8} \cdot a_0 \oplus 2\text{F} \cdot a_1 \oplus 49 \cdot a_2 \oplus \text{D7} \cdot a_3 \oplus \text{CA} \cdot a_4 \oplus \text{AD} \cdot a_5 \oplus 95 \cdot a_6 \oplus 76 \cdot a_7 \\ 76 \cdot a_0 \oplus \text{A8} \cdot a_1 \oplus 2\text{F} \cdot a_2 \oplus 49 \cdot a_3 \oplus \text{D7} \cdot a_4 \oplus \text{CA} \cdot a_5 \oplus \text{AD} \cdot a_6 \oplus 95 \cdot a_7 \\ 95 \cdot a_0 \oplus 76 \cdot a_1 \oplus \text{A8} \cdot a_2 \oplus 2\text{F} \cdot a_3 \oplus 49 \cdot a_4 \oplus \text{D7} \cdot a_5 \oplus \text{CA} \cdot a_6 \oplus \text{AD} \cdot a_7 \end{bmatrix}$$

Рис. 8.16. Порядок обчислення байт при оберненому перемішуванні в стовпці

Перетворення InvShiftRows

Обернене для ShiftRows перетворення InvShiftRows полягає у виконанні циклічних зсувів рядків стану на ту ж саму кількість байт (табл. 8.7), але вліво. На рис. 8.17 наведено порядок розподілення байт першого стовпця при виконанні перетворення InvShiftRows.



а) 128-бітний блок

б) 256-бітний блок

в) 512-бітний блок

Рис. 8.17. Порядок розподілення байт першого стовпця при виконанні перетворення InvShiftRows

Перетворення InvSubBytes

Перетворення InvSubBytes полягає у заміні кожного байта поточного стану шифру відповідно за таблицями обернених підстановок (табл. 8.8 – 8.11). Отже, обернена процедура відрізняється від прямої лише таблицями підстановок. Порядок виконання оберненого перетворення для кожного окремо взятого байта є повністю ідентичним порядку виконання прямого перетворення.

Приклад 8.2. Виконати перетворення InvSubBytes для таблиці $_{-1}S_0$ байта з шістнадцятковим значенням A4.

Рішення. Результат підстановки для значення A4 – це число в шістнадцятковому представленні 3D, що знаходиться в таблиці на перетині 10-го рядка (з індексом A) та 5-го стовпця (з індексом 4) (дивись табл. 8.8).

Якщо при шифруванні використовувався набір підстановок, що відрізняється від наведеного, то при розшифруванні застосовується відповідний набір таблиць обернених підстановок.

Таблиця 8.8

Обернена підстановка $_{-1}S_0$

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

Таблиця 8.9

Обернена підстановка $_{-1}S_1$

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

Таблиця 8.10

Обернена підстановка $_{-1}S_2$

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

Таблиця 8.11

Обернена підстановка $_{-1}S_3$

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

Перетворення AddRoundKey

Операція додавання за модулем два (перетворення AddRoundKey) є оберненою до себе (подвійне застосування дає початкове значення), відповідно, якщо до стану $B = A \oplus K$ додати ключ раунду K, то буде отримано початковий стан A.

8.5. Формування раундових ключів в ДСТУ 7624:2014

Для отримання раундових ключів застосовується процедура формування, що базується на раундовому перетворенні (для поточного розміру блоку). При шифруванні використовується N_r+1 раундових ключів K_i (i = 0, 1, …, N_r), кожний довжиною 64×N_b біт (розмір раундового ключа співпадає з розміром блоку відкритого або зашифрованого тексту і поточного стану шифру).

Алгоритм формування раундових ключів містить три етапи

1) З ключа шифрування K формується проміжний ключ K_t довжиною, що дорівнює розміру блока ($64 \times N_b$ біт) з використанням трьох раундів шифрування (рис. 8.18а).

2) На основі ключа шифрування K та проміжного ключа K_t формуються раундові ключі K_{2i} (з парними індексами) довжиною, що дорівнює розміру блока ($64 \times N_b$ біт), з використанням двох раундів шифрування для кожного раундового ключа (рис. 8.18б).

3) З раундових ключів K_{2i} з парними індексами формуються раундові ключі K_{2i+1} (з непарними індексами) шляхом циклічного зсуву попереднього ключа з парним індексом вліво на $2 \cdot N_b + 3$ байт (рис. 8.18в).

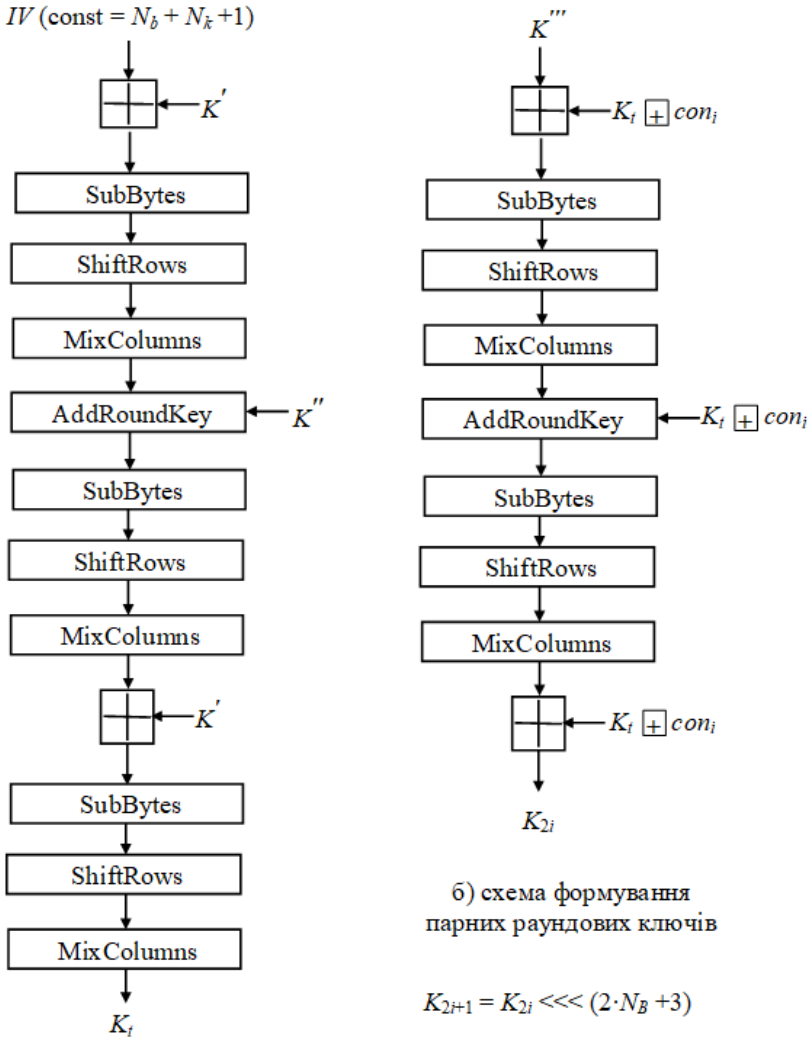


Рис. 8.18. Схема формування раундових ключів

Раундові ключі можуть формуватися незалежно один від одного з однаковою обчислювальною складністю як для шифрування, так і для розшифрування.

Проміжний ключ K_t формується на основі ключа шифрування K довжиною $64 \times N_k$ біт та константи (вектора ініціалізації IV), яка залежить від довжини ключа шифрування та розміру блоку (const = $N_b + N_k + 1$). Перетворення формування проміжного ключа $Z_{l,k}^{(K)}$ визначається наступним чином:

$$Z_{l,k}^{(K)} = \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(K')} \circ \psi_l \circ \tau_l \circ \pi_l' \circ K_l^{(K'')} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(K')},$$

де l – розмір блоку в бітах (128, 256 або 512);

k – довжина ключа шифрування в бітах (128, 256 або 512);

K – ключ шифрування;

$\eta_l^{(K')}$, π_l', τ_l, ψ_l, $K_l^{(K'')}$ – перетворення, описані в п. 8.3.

Якщо довжина ключа і розмір блока збігаються ($k = \cdot l$), то $K' = K'' = K$.

Якщо довжина ключа і розмір блока не збігаються ($k = 2 \cdot l$), то K' дорівнює лівій половині ключа шифрування K, а K'' – правій половині ключа шифрування K (рис. 8.18а).

Проміжний ключ K_t необхідний для того, щоб забезпечити односпрямованість (необоротність) схеми формування раундових ключів. У цьому випадку може бути забезпечена експоненційна обчислювальна складність відновлення ключа шифрування на основі раундових ключів. При цьому руйнування симетрії шифрувального перетворення (шифрування відкритих текстів з однаковими 64-бітовими або менше блоками) блоків на ключах зі співпадаючими значеннями у рамках блоків не призводить до симетрії значень зашифрованих текстів, оскільки вихідна послідовність не має будь-якої регулярності і має всі властивості псевдовипадковості [4, 5, 31].

Основним призначенням вектора ініціалізації *IV* є забезпечення руйнування симетрії. Крім того, він необхідний для формування унікальної послідовності раундових ключів для кожної комбінації розміру блока і довжини ключа (наприклад, для режиму 128/128 та 128/256 раундові ключі будуть формувати унікальні псевдовипадкові послідовності, навіть якщо 256-бітовий ключ складається зі співпадаючих 128-бітових блоків, що дорівнюють ключу режиму 128/128).

Кожний раундовий ключ K_i ($i = 0, 1, \ldots, N_r$) формується на основі ключа шифрування K довжиною $64 \times N_k$ біт, проміжного ключа K_t та власного індексу i.

Перетворення формування раундових ключів з парними індексами ($i = 0, 2, 4, \ldots, N_r$) $\Pi_{l,k}^{(K, K_t)}$ визначається наступним чином (рис. 8.18б):

$$\Pi_{l,k}^{(K, K_t, i)} = \eta_l^{(\lambda_i(K_t))} \circ \psi_l \circ \tau_l \circ \pi_l' \circ K_l^{(\lambda_i(K_t))} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(\lambda_i(K_t))},$$

де l – розмір блоку в бітах (128, 256 або 512);

k – довжина ключа шифрування в бітах (128, 256 або 512);

K – ключ шифрування;

K_t – проміжний ключ;

$\eta_l^{(*)}, \pi_l', \tau_l, \psi_l, K_l^{(*)}$ – перетворення, описані в п. 8.3;

$\lambda_i(K_t)$ – функція додавання проміжного ключа K_t з константами, зсунутими відповідно до індексу раундового ключа, за модулем 2^{64}.

Початкова константа (для $i = 0$) формується повторенням байт 00 та 01 (в шістнадцятковому представленні) до повного заповнення стану. Наступні константи для ключів раундів з парними індексами ($i = 2, 4, \ldots, N_r$) формується шляхом зсуву вліво на один біт попередніх констант:

$$con_0 = 0 \text{ x } 0001\ 0001 \ldots 0001; \qquad con_{i+2} = con_i <<1.$$

Якщо довжина ключа й розмір блока збігаються ($k = \cdot l$), то

$$K''' = K >>> 32 \cdot i,$$

де K – ключ шифрування;

$>>> 32 \cdot i$ – операція циклічного зсуву вправо на «$32 \cdot i$» позицій ($i = 0, 2, \ldots, N_r$).

Якщо довжина ключа і розмір блока не збігаються ($k = 2 \cdot l$), то:

– для формування раундових ключів з індексами, кратними 4 ($i = 0, 4, 8, \ldots$)

$$K''' = K_L >>> 16 \cdot i,$$

де K_L – ліва частина ключа шифрування K;

– для формування раундових ключів з індексами, не кратними 4 ($i = 2, 6, \ldots$)

$$K''' = K_R >>> 64 \cdot \lfloor i/4 \rfloor,$$

де K_R – права частина ключа шифрування K;

$\lfloor x \rfloor$ – ціла частина x, тобто для дійсного x найбільше ціле y таке, що $y \le x$.

Кожний раундовий ключ K_i з непарними індексами (i = 1, 3, 5, …, N_r-1) обчислюється із попереднього ключа з парним індексом шляхом циклічного зсуву вліво на $2 \cdot N_b + 3$ байта (рис. 8.18в):

$$K_{2i+1} = K_{2i} <<< (2 \cdot N_b + 3), \quad (i = 0, 1, 2, \ldots)$$

де N_b – розмір блоку відкритого тексту (див. табл. 8.1).

Залежність констант, що визначають значення циклічного зсуву вліво раундового ключа з парним індексом, від розміру блока наведена у табл. 8.12.

Таблиця 8.12

Константи, що визначають значення циклічного зсуву вліво раундового ключа з парним індексом, від розміру блока

Розмір блоку біт (байт)	Зсув вліво (байт)
128 (16)	7
256 (32)	11
512 (64)	19

8.6. Режими роботи ДСТУ 7624:2014

Режими роботи криптографічного алгоритму, визначеного в ДСТУ 7624:2014, їх позначення та послуги безпеки, які забезпечує відповідний режим, визначено в табл. 8.13.

Таблиця 8.13

Режими роботи ДСТУ 7624:2014

№	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

Режими роботу стандарту позначаються наступним чином (для деяких режимів параметрів немає):

"Калина-*l*/*k*-позначення режиму-параметри режиму"

де *l* – розмір блоку базового перетворення;

k – довжина ключа.

Наприклад. “Калина-256/512-CCM-32, 128” визначає використання базового перетворення з розміром блоку 256 біт, довжиною ключа 512 біт, застосування режиму вироблення імітовставки і гамування, довжина конфіденційної (та відкритої) частини повідомлення завжди менше, ніж 2^{32} байтів, довжина імітовставки дорівнює 128 біт.

Режим простої заміни збігається з базовим перетворенням, тому, крім позначення “Калина-*l*/*k*-ECB”, використовується позначення “Калина-*l*/*k*”.

Режим простої заміни є компонентом усіх інших режимів. Базове перетворення реалізує пряме перетворення (шифрування) (описано в п. 8.3) та обернене перетворення (розшифрування) (описано в п. 8.4).

Контрольні питання до розділу 8

1. Перерахуйте параметри (розмір блоку, довжина ключа і кількість раундів) для різних версій ДСТУ 7624:2014.
2. Які операції використовуються в блоковому алгоритмі шифрування ДСТУ 7624:2014?
3. Які основні відмінності алгоритму шифрування ДСТУ 7624:2014 від алгоритму AES?
4. Визначте поточний стан в ДСТУ 7624:2014 для різних версій.
5. Які основні відмінності схеми формування раундових ключів ДСТУ 7624:2014 від алгоритму AES?
6. Для чого потрібен проміжний ключ K_t в схемі формування раундових ключів ДСТУ 7624:2014?
7. Для чого використовується вектор ініціалізації *IV* (константа) в схемі формування раундових ключів ДСТУ 7624:2014?
8. Які режими роботи ДСТУ 7624:2014 забезпечують послугу конфіденційності?
9. Які режими роботи ДСТУ 7624:2014 забезпечують послугу цілісності?

10. Що таке імітовставка? З якою метою може бути використана імітовставка?

11. Значення байтів поточного стану даних на вході функції SubBytes ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: A49C7FF2689F352B6B5BEA43026A5049. Визначити значення байтів поточного стану даних на виході функції SubBytes.

12. Значення байтів поточного стану даних на вході функції InvSubBytes ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: F196DB453B53027789D2DE491A87397F. Визначити значення байтів поточного стану даних на виході функції InvSubBytes.

13. Значення байтів поточного стану даних на вході функції ShiftRows ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: 49DED28945DB96F17F39871A7702533B. Визначити значення байтів поточного стану даних на виході функції ShiftRows.

14. Значення байтів поточного стану даних на вході функції InvShiftRows ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: F196DB453B53027789D2DE491A87397F. Визначити значення байтів поточного стану даних на виході функції InvShiftRows.

15. Значення байтів поточного стану даних на вході функції MixColumns ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: 49DB873B453953897F02D2F177DE961A. Визначити значення байтів поточного стану даних на виході функції MixColumns.

16. Значення байтів поточного стану даних на вході функції InvMixColumns ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: 31ADB22485C967CAEDD5E4A2FF9E4AC3. Визначити значення байтів поточного стану даних на виході функції InvMixColumns.

17. Значення байтів поточного стану даних на вході функції AddRoundKey ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: 584DCAF11B4B5AACDBE7CAA81B6DB0E5. Значення байтів поточного стану раундового ключа на вході функції AddRoundKey –

F2C295F27A9BB9435935807A7359F67F. Визначити значення байтів поточного стану на виході функції AddRoundKey.

18. Значення байтів поточного стану даних на вході функції додавання за модулем 2^{64} ДСТУ 7624:2014 в шістнадцятковій системі дорівнюють: 49DB873B453953897F02D2F177DE961A. Значення байтів поточного стану раундового ключа на вході функції додавання за модулем 2^{64} – F2C295F27A9BB9435935807A7359F67F. Визначити значення байтів поточного стану на виході функції додавання за модулем 2^{64}.

РОЗДІЛ 9
ПОТОКОВІ СИСТЕМИ ШИФРУВАННЯ

9.1. Загальні відомості про потокові шифри та їх класифікація

Блоковий алгоритм призначений для шифрування блоків певної довжини. Однак може виникнути необхідність шифрування даних не блоками, а, наприклад, символами. Такі вимоги задовольняє потоковий шифр (stream cipher), який виконує перетворення вхідного повідомлення по одному біту (або байту) за операцію. Потоковий алгоритм шифрування усуває необхідність розбивати повідомлення на ціле число блоків досить великої довжини, і тому, він може працювати в реальному часі. Отже, якщо передається потік символів, кожний символ може шифруватися й передаватися відразу. Принцип роботи типового потокового шифру представлено на рис. 9.1.

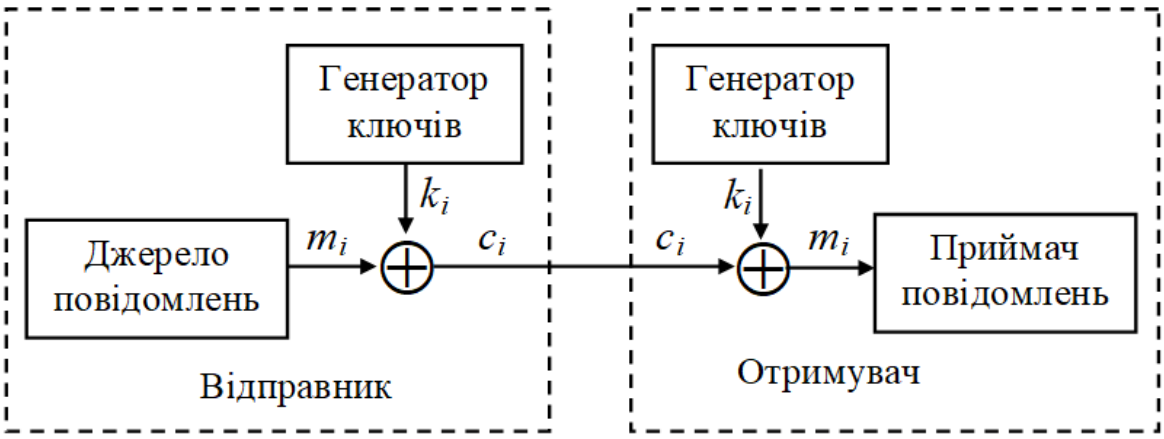


Рис. 9.1. Принцип роботи потокового шифру

Генератор ключів видає потік бітів k_i, які будуть використовуватися в якості гами. Джерело повідомлень генерує біти відкритого тексту m_i, які додаються за модулем два з гамою, в результаті чого виходять біти зашифрованого повідомлення c_i:

$$c_i = m_i \oplus k_i, \quad i = 1, 2, \ldots, n.$$

Щоб із зашифрованого повідомлення $c_1, c_2, \ldots, c_n$ відновити початкове повідомлення $m_1, m_2, \ldots, m_n$, необхідно згенерувати таку саму ключову послідовність $k_1, k_2, \ldots, k_n$, що і при шифруванні, і використовувати для розшифрування формулу:

$$m_i = c_i \oplus k_i, \quad i = 1, 2, \ldots, n,$$

так як операція порозрядного додавання за модулем два має властивість оберненості.

Зазвичай вихідним повідомленням і ключовою послідовністю є незалежні потоки бітів. Отже, оскільки перетворення шифрування та розшифрування для всіх потокових шифрів таке саме, то вони повинні відрізнятися тільки способом побудови генераторів ключів. Виходить, що безпека системи повністю залежить від властивостей генератора потоку ключів. Якщо генератор потоку ключів видає послідовність, що складається тільки з одних нулів (або з одних одиниць), то зашифроване повідомлення буде точно таким самим, як і вихідний потік бітів (у разі одиничних ключів зашифроване повідомлення буде інверсією вихідного).

Класифікація потокових шифрів

Припустимо, наприклад, що в режимі гамування для потокових шифрів під час передачі по каналу зв’язку відбулося спотворення одного знака зашифрованого повідомлення. Очевидно, що в цьому випадку всі знаки, прийняті без спотворення, будуть розшифровані правильно. Відбудеться втрата лише одного знака повідомлення. А тепер уявімо, що один зі знаків зашифрованого повідомлення під час передачі по каналу зв’язку був втрачений. Це призведе до неправильного розшифрування всього повідомлення, наступного за втраченим знаком. Практично у всіх каналах передачі даних для потокових систем шифрування присутні перешкоди. Тому для запобігання втрати інформації вирішують проблему синхронізації

шифрування і розшифрування повідомлення. За способом вирішення цієї проблеми криптографічні системи поділяються на синхронні системи й системи з самосинхронізацією.

Синхронні потокові шифри

Синхронні потокові шифри – це шифри, в яких потік ключів (гама) генерується незалежно від відкритого й зашифрованого повідомлення.

Під час шифруванні генератор потоку ключів видає біти потоку ключів, які ідентичні бітам потоку ключів при розшифруванні. Втрата знака зашифрованого повідомлення призведе до порушення синхронізації між цими двома генераторами й неможливості розшифрування частини повідомлення. Очевидно, що в цій ситуації відправник і отримувач повинні повторно синхронізуватись для продовження роботи. Зазвичай синхронізація проводиться вставкою в передане повідомлення спеціальних маркерів. Внаслідок цього пропущений під час передачі знак призводить до невірного розшифрування лише до того часу, поки не буде прийнятий один із маркерів. Зауважимо, що виконуватися синхронізація повинна так, щоб жодна частина потоку ключів не повторювалась. Тому, переводити генератор у більш ранній стан не має сенсу.

Основними перевагами синхронних потокових шифрів є:

– відсутність ефекту поширення помилок (тільки спотворений біт буде розшифрований невірно);

– оберігають від будь-яких вставок і вилучень зашифрованого повідомлення, оскільки вони призведуть до втрати синхронізації й будуть виявлені.

Недоліком синхронних потокових шифрів є уразливість до зміни окремих бітів зашифрованого повідомлення. Якщо зловмисникові відоме відкрите повідомлення, він може змінити ці біти так, щоб вони розшифровувались, як йому потрібно.

Потокові шифри з самосинхронізацією

Потокові шифри з самосинхронізацією – це шифри, в яких потік ключів створюється функцією ключа й фіксованою кількістю знаків зашифрованого повідомлення.

Отже, внутрішній стан генератора потоку ключів є функцією попередніх n бітів зашифрованого повідомлення. Тому генератор потоку ключів, на боці розшифрування, прийнявши n бітів, автоматично синхронізується із генератором потоку ключів на боці шифрування.

Реалізація цього режиму відбувається наступним чином: кожне повідомлення починається випадковим заголовком довжиною n бітів; заголовок зашифровується, передається й розшифровується; розшифрування буде неправильним, проте після цих n бітів обидва генератори будуть синхронізовані.

Основними перевагами потокових шифрів з самосинхронізацією є розмішування статистики відкритого повідомлення. Оскільки кожний символ відкритого повідомлення впливає на наступне зашифроване повідомлення, статистичні властивості відкритого повідомлення поширюються на всі зашифровані повідомлення. Отже, потоковий шифр з самосинхронізацією може бути більш стійким до атак на основі надмірності відкритого повідомлення, ніж синхронний потоковий шифр.

Недоліками потокових шифрів з самосинхронізацією є:

– поширення помилки (кожному неправильному біту зашифрованого повідомлення відповідають n помилок у відкритому повідомленні);

– чутливі до розкриття повторною передачею.

9.2. Уніфікація функцій потокових шифрів

Для уніфікації представлення нові потокові шифри подаються як генератори гами. Такий підхід суттєво спростив їх реалізацію та впровадження [4, 10].

Синхронні генератори гами

Синхронний генератор гами є скінченним автоматом. Він визначається через такі складові:

1. Функція ініціалізації *Init* – яка в якості вхідних даних приймає ключ *K* і вектор ініціалізації *IV* і виводить початковий стан S_0 для генератора гами. Причому вектор ініціалізації має вибиратися таким чином, щоб ніякі два повідомлення ніколи не були зашифровані з використанням одного й того ж ключа та одного й того ж *IV*.

2. Функція наступного стану *Next* – яка в якості вхідних даних приймає поточний стан генератора гами S_i і виводить наступний стан генератора гами S_{i+1}.

3. Функція гами *Strm* – яка в якості вхідних даних приймає стан генератора гами S_i і виводить блок гами Z_i.

Коли синхронний генератор гами ініціюється вперше, то він вводить початковий стан S_0, визначений як

$$S_0 = Init\ (IV, K).$$

За запитом синхронний генератор гами для $i = 0, 1, \ldots$ виконує:

– виведення блока гами

$$Z_i = Strm\ (S_i, K);$$

– оновлення стану скінченного автомата

$$S_{i+1} = Next\ (S_i, K).$$

Тому для визначення синхронного генератора гами необхідно тільки описати функції *Init, Next* і *Strm*, разом з довжинами й алфавітами ключа, вектором ініціалізації, станом і вихідним блоком.

Генератори гами з самосинхронізацією

Генерація гами для потокового шифру з самосинхронізацією відрізняється від синхронного потокового шифру залежністю гами тільки від попередніх зашифрованих текстів, ключа і вектора ініціалізації, тобто генератор гами працює без внутрішнього стану. У результаті, алгоритм розшифрування для такого шифру може відновлюватися зі стану втрати синхронізації після отримання достатньої кількості блоків зашифрованих текстів. Метод генерації гами залежить від вибраної вихідної функції *Out*, яка зазвичай являє собою бінарний адитивний режим.

Основними функціями генератора гами з самосинхронізацією є такі:

1. Функція ініціалізації *Init* – яка в якості вхідних даних приймає ключ *K* і вектор ініціалізації *IV* і виводить внутрішні вхідні дані для генератора гами.

2. Функція гами *Strm* – яка в якості вхідних даних приймає стан генератора гами *S* та *r* блоків шифротексту $c_{i-1}, c_{i-2}, \ldots, c_{i-r}$ і виводить блок гами Z_i.

Для визначення генератора гами з самосинхронізацією необхідно тільки визначити число блоків зворотного зв'язку *r* і функції *Init* і *Strm.*

Вихідні функції потокового шифру

Розглянемо дві вихідні функції потокового шифру, що використовуються в потоковому шифрі для комбінування гами з відкритим текстом для отримання зашифрованого тексту. Вихідна функція виводу для синхронного потокового шифру або потокового шифру з самосинхронізацією є оборотною функцією *Out*, у якій для отримання блоку зашифрованого тексту C_i $(i > 0)$ комбінується блок відкритого тексту M_i, блок гами Z_i і, необов'язково, деякі інші вхідні дані *R*. Шифрування блоку відкритого тексту M_i за допомогою блоку гами Z_i подається таким чином:

$$C_i = Out\ (M_i, Z_i, R),$$

а розшифрування блоку зашифрованого тексту C_i за допомогою блока гами Z_i задається як:

$$M_i = Out^{-1}(C_i, Z_i, R).$$

Вихідна функція має бути такою, щоб для будь-якого блоку гами Z_i, блоку відкритого тексту M_i та інших вхідних даних R, виконувалась умова:

$$M_i = Out^{-1}(Out(M_i, Z_i, R), Z_i, R).$$

9.3. Принципи використання генераторів псевдовипадкових чисел при потоковому шифруванні

Сучасна інформатика широко використовує псевдовипадкові числа в самих різних додатках – від методів математичної статистики й імітаційного моделювання до криптографії. При цьому від якості використовуваних генераторів псевдовипадкових чисел (ГПВЧ) безпосередньо залежить якість одержуваних результатів [27].

ГПВЧ можуть використовуватися в якості генераторів ключів у потокових шифрах. Метою використання ГПВЧ є отримання “нескінченного” ключового слова, за використання відносно малої довжини самого ключа. ГПВЧ створює послідовність бітів, схожу на випадкову. Насправді, такі послідовності обчислюються за певними правилами та не є випадковими, тому вони можуть бути абсолютно точно відтворені як на передаючій, так і на приймаючій сторонах. Послідовність ключових символів, що використовується під час шифрування, повинна бути не тільки надто довгою. Якщо генератор ключів за кожного включення створює ту саму послідовність бітів, то зламати таку систему також буде можливо. Отже, вихід генератора потоку ключів повинен бути функцією ключа. У цьому випадку розшифрувати та прочитати повідомлення можна буде тільки з використанням того самого ключа, який використовувався під час шифрування.

Для використання з криптографічною метою ГПВЧ повинен мати наступні властивості:

– період повторення послідовності повинен бути не менше допустимого (у реальних потокових шифрах він може змінюватися у межах 2^{128}-2^{256});

– породжувана послідовність не повинна відрізнятися від дійсно випадкової;

– ймовірності появи (породження) різних значень повинні бути рівними;

– можливість відновлення послідовності у просторі і часі;

– для того, щоб тільки законний отримувач міг розшифрувати повідомлення, слід під час отримання потоку ключових біт k_i використовувати певний секретний ключ, причому обчислення числа k_{i+1} за відомим попереднім елементом послідовності k_i без знання ключа повинно бути важким завданням;

– ймовірність перекриття шифру не повинна перевищувати допустиму величину. Перекриття шифру – це факт генерування та зашифровування різних повідомлень однаковими відрізками гами шифрування, тобто

$$\begin{cases} C_i = M_i \oplus \Gamma_i, & i = \overline{1,k} \\ C_i' = M_i' \oplus \Gamma_i, & i = \overline{1,k}. \end{cases}$$

Додаємо за модулем два обидва зашифрованих тексти:

$$C_i \oplus C_i' = \left(M_i \oplus M_i'\right) \oplus \left(\Gamma_i \oplus \Gamma_i\right) = M_i \oplus M' .$$

Дешифрування суми двох повідомлень є нескладним, що дозволяє знайти повідомлення.

За наявності зазначених властивостей послідовності псевдовипадкових чисел можуть бути використані в потокових шифрах.

9.3.1. Лінійний конгруентний генератор псевдовипадкових чисел

Генератори псевдовипадкових чисел можуть працювати за різними алгоритмами. Одним із найпростіших генераторів є так званий лінійний конгруентний генератор, який для обчислення чергового числа k_i використовує формулу:

$$k_i = (a \cdot k_{i-1} + b) \bmod c,$$

де a, b, c – деякі константи, а k_{i-1} – попереднє псевдовипадкове число.

Для отримання k_1 задається початкове значення k_0. Візьмемо як приклад $a = 5$, $b = 3$, $c = 11$ і нехай $k_0 = 1$. У цьому випадку можна за наведеною вище формулою можливо отримати значення від 0 до 10 (оскільки $c = 11$). Обчислимо кілька елементів послідовності:

$$k_1 = (5 \cdot 1 + 3) \bmod 11 = 8;$$
$$k_2 = (5 \cdot 8 + 3) \bmod 11 = 10;$$
$$k_3 = (5 \cdot 10 + 3) \bmod 11 = 9;$$
$$k_4 = (5 \cdot 9 + 3) \bmod 11 = 4;$$
$$k_5 = (5 \cdot 4 + 3) \bmod 11 = 1.$$

Отримані значення (8, 10, 9, 4, 1) виглядають схожими на випадкові числа. Однак наступне значення k_6 буде знову дорівнювати 8:

$$k_6 = (5 \cdot 1 + 3) \bmod 11 = 8,$$

а значення k_7 і k_8 дорівнюватимуть 10 і 9 відповідно:

$$k_7 = (5 \cdot 8 + 3) \bmod 11 = 10;$$
$$k_8 = (5 \cdot 10 + 3) \bmod 11 = 9.$$

Виходить, генератор псевдовипадкових чисел формує послідовність, яка повторюється, породжуючи періодично числа 8, 10, 9, 4, 1. На жаль, ця

властивість характерна для всіх лінійних конгруентних генераторів. Змінюючи значення основних параметрів *a*, *b* і *c*, можна впливати на довжину періоду й на самі значення k_i, які породжуються. Так, наприклад, збільшення числа *c* у загальному випадку приводить до збільшення періоду. Якщо параметри *a*, *b* і *c* обрані правильно, то генератор буде породжувати випадкові числа з максимальним періодом, що дорівнює *c*. За програмної реалізації значення *c* зазвичай встановлюється таким, що дорівнює 2^{b-1} або 2^b, де *b* – довжина слова електронної обчислювальної машини або автоматизованої системи у бітах.

Перевагою лінійних конгруентних генераторів псевдовипадкових чисел є їх простота й висока швидкість отримання псевдовипадкових значень. Лінійні конгруентні генератори застосовуються під час розв'язання задач моделювання та математичної статистики, проте з криптографічною метою їх не можна рекомендувати для використання, так як фахівці в області криптографічного аналізу навчилися відновлювати всю послідовність псевдовипадкових чисел із кількох значень. Наприклад, припустимо, що зловмисник може визначити значення k_0, k_1, k_2, k_3. Тоді:

$$k_1 = (a \cdot k_0 + b) \bmod c;$$
$$k_2 = (a \cdot k_1 + b) \bmod c;$$
$$k_3 = (a \cdot k_2 + b) \bmod c.$$

Розв'язавши систему цих трьох рівнянь, можна знайти *a*, *b* і *c*. Для отримання псевдовипадкових чисел пропонувалося використовувати також квадратичні й кубічні генератори:

$$k_i = (a_1^2 \cdot k_{i-1} + a_2 \cdot k_{i-1} + b) \bmod c;$$
$$k_i = (a_1^3 \cdot k_{i-1} + a_2^2 \cdot k_{i-1} + a_3 \cdot k_{i-1} + b) \bmod c.$$

Однак такі генератори теж виявилися непридатними для мети криптографії з тієї самої причини – "передбачуваності".

9.3.2. Генератор псевдовипадкових чисел на основі методу Фібоначчі із запізненням

Метод Фібоначчі із запізнюваннями (Lagged Fibonacci Generator) – один із методів генерації псевдовипадкових чисел, що дозволяє отримати більш високу “якість” псевдовипадкових чисел. Найбільшу популярність датчики Фібоначчі отримали через те, що швидкість виконання арифметичних операцій із дійсними числами зрівнялася зі швидкістю цілочисельної арифметики, а датчики Фібоначчі природно реалізуються в дійсній арифметиці [1, 27].

Відомі різні схеми використання методу Фібоначчі із запізненням. Один із широко поширених датчиків Фібоначчі заснований на наступній рекурентній формулі:

$$k_i = \begin{cases} k_{i-a} - k_{i-b}, & \text{якщо} \quad k_{i-a} \geq k_{i-b}; \\ k_{i-a} - k_{i-b} + 1, & \text{якщо} \quad k_{i-a} < k_{i-b}, \end{cases}$$

де k_i – дійсні числа з діапазону [0, 1];

a, b – цілі позитивні числа, параметри генератора.

Для роботи датчика Фібоначчі потрібно знати значення $\max\{a, b\}$ попередніх згенерованих випадкових чисел. За програмної реалізації для зберігання згенерованих випадкових чисел необхідний певний обсяг пам’яті, що залежить від параметрів a і b.

Приклад 9.1. Обчислимо послідовність із перших десяти чисел, що генерується методом Фібоначчі із запізненням починаючи з k_5 за таких вихідних даних: $a = 4$; $b = 1$; $k_0 = 0{,}1$; $k_1 = 0{,}7$; $k_2 = 0{,}3$; $k_3 = 0{,}9$; $k_4 = 0{,}5$:

$$k_5 = k_1 - k_4 = 0{,}7 - 0{,}5 = 0{,}2;$$
$$k_6 = k_2 - k_5 = 0{,}3 - 0{,}2 = 0{,}1;$$
$$k_7 = k_3 - k_6 = 0{,}9 - 0{,}1 = 0{,}8;$$
$$k_8 = k_4 - k_7 + 1 = 0{,}5 - 0{,}8 + 1 = 0{,}7;$$

$$k_9 = k_5 - k_8 + 1 = 0{,}2 - 0{,}7 + 1 = 0{,}5;$$
$$k_{10} = k_6 - k_9 + 1 = 0{,}1 - 0{,}5 + 1 = 0{,}6;$$
$$k_{11} = k_7 - k_{10} = 0{,}8 - 0{,}6 = 0{,}2;$$
$$k_{12} = k_8 - k_{11} = 0{,}7 - 0{,}2 = 0{,}5;$$
$$k_{13} = k_9 - k_{12} = 0{,}5 - 0{,}5 = 0{,}0;$$
$$k_{14} = k_{10} - k_{13} = 0{,}6 - 0{,}0 = 0{,}6.$$

Бачимо, що генерується послідовність чисел, зовні схожа на випадкову. І дійсно, дослідження підтверджують, що отримані випадкові числа мають гарні статистичні властивості.

Для генераторів, побудованих за методом Фібоначчі із запізненням, існують рекомендовані параметри a і b, так би мовити, протестовані на якість. Наприклад, дослідники пропонують такі значення: $(a, b) = (55, 24)$, $(17, 5)$, або $(97, 33)$. Якість одержуваних випадкових чисел залежить від значення константи a: чим воно більше, тим вище розмірність простору, в якому зберігається рівномірність випадкових векторів, утворених з отриманих випадкових чисел. У той самий час із збільшенням значення константи a збільшується обсяг використовуваної алгоритмом пам'яті.

Отже, значення $(a, b) = (17, 5)$ рекомендуються для простих додатків. Значення $(a, b) = (55, 24)$ дозволяють отримати числа, які задовольняють більшість криптографічних алгоритмів, вимогливих до якості випадкових чисел. Значення $(a, b) = (97, 33)$ дозволяють отримати якісні випадкові числа й використовуються в алгоритмах, що працюють із випадковими векторами високої розмірності [24].

Генератори псевдовипадкових чисел, засновані на методі Фібоначчі із запізненням, використовувалися для цілей криптографії. Крім того, вони застосовуються в математичних і статистичних розрахунках, а також за моделювання випадкових процесів. Генератор псевдовипадкових чисел, побудований на основі методу Фібоначчі із запізненням, використовувався в широко відомої системі Matlab.

9.3.3. Генератор псевдовипадкових чисел на основі алгоритму BBS

Широке поширення набув алгоритм генерації псевдовипадкових чисел, названий алгоритмом BBS (від прізвищ авторів – L. Blum, M. Blum, M. Shub) або генератор із квадратичним залишком. Для цілей криптографії цей метод запропонований в 1986 році [24, 40].

Сутність цього алгоритму полягає в наступному. Спочатку вибираються два великих простих числа p і q, які повинні бути порівнянні з 3 за модулем 4, тобто при діленні p і q на 4 повинен виходити однаковий залишок 3. Далі обчислюється число $M = p \cdot q$, яке називається цілим числом Блюма. Потім вибирається інше випадкове ціле число x, взаємно просте (тобто не має спільних дільників, крім одиниці) з M і обчислюється $x_0 = x^2 \bmod M$, де x_0 називається стартовим числом генератора.

На кожному n-му кроці роботи генератора обчислюється:

$$x_{n+1} = x_n^2 \bmod M.$$

Результатом n-го кроку є один (зазвичай молодший) біт числа x_{n+1}. Іноді як результат приймають біт парності, тобто кількість одиниць у двійковому поданні елемента. Якщо кількість одиниць у записі числа парне, то біт парності приймається таким, що дорівнює 0, інакше (непарне) – біт парності приймається таким, що дорівнює 1.

Приклад 9.2. Нехай $p = 11$, $q = 19$ (переконуємося, що $11 \bmod 4 = 3$, $19 \bmod 4 = 3$). Тоді $M = p \cdot q = 11 \cdot 19 = 209$. Виберемо x, взаємно просте з M: нехай $x = 3$. Обчислимо стартове число генератора x_0:

$$x_0 = x^2 \bmod M = 3^2 \bmod 209 = 9 \bmod 209 = 9.$$

Обчислимо перші десять чисел x_i за алгоритмом BBS. У якості випадкових біт будемо брати молодший біт у двійковому записі числа x_i:

$x_1 = 9^2 \bmod 209 = 81 \bmod 209 = 81$ – молодший біт 1;

$x_2 = 81^2 \bmod 209 = 6561 \bmod 209 = 82$ – молодший біт 0;

$x_3 = 82^2 \bmod 209 = 6724 \bmod 209 = 36$ – молодший біт 0;

$x_4 = 36^2 \bmod 209 = 1296 \bmod 209 = 42$ – молодший біт 0;

$x_5 = 42^2 \bmod 209 = 1764 \bmod 209 = 92$ – молодший біт 0;

$x_6 = 92^2 \bmod 209 = 8464 \bmod 209 = 104$ – молодший біт 0;

$x_7 = 104^2 \bmod 209 = 10816 \bmod 209 = 157$ – молодший біт 1;

$x_8 = 157^2 \bmod 209 = 24649 \bmod 209 = 196$ – молодший біт 0;

$x_9 = 196^2 \bmod 209 = 38416 \bmod 209 = 169$ – молодший біт 1;

$x_{10} = 169^2 \bmod 209 = 28561 \bmod 209 = 137$ – молодший біт 1.

Найцікавішою для практичних цілей властивістю цього методу є те, що для отримання n-го числа послідовності не потрібно обчислювати всі попередні n чисел x_i. Виявляється x_n можна відразу отримати за формулою:

$$x_n = x_0^{2^n \bmod ((p-1)(q-1))} \bmod M.$$

Наприклад, обчислимо x_{10} відразу з x_0:

$$x_{10} = x_0^{2^{10} \bmod ((11-1)(19-1))} \bmod 209 = 9^{1024 \bmod 180} \bmod 209 = 137.$$

Як результат дійсно отримали таке саме значення, як і за послідовного обчислення, – 137. Обчислення здаються досить складними, проте насправді їх не складно оформити у вигляді невеликої процедури або програми й використовувати за необхідності.

Можливість “прямого” отримання x_n дозволяє використовувати алгоритм BBS при потоковому шифруванні, наприклад, для файлів з довільним доступом або фрагментів файлів із записами бази даних.

Безпека алгоритму BBS основана на складності розкладання великого числа M на множники. Стверджується, що якщо M досить велике, його можна навіть не тримати в секреті; до того часу, поки M не розкладено на

множники, ніхто не зможе передбачити вихід генератора псевдовипадкових чисел. Це пов’язано з тим, що задача розкладання чисел виду $n = p \cdot q$ (p і q – прості числа) на множники є надто складною для обчислення, якщо відоме тільки n, а p і q – великі числа, що складаються з декількох десятків або сотень бітів (це так звана задача факторизації).

Крім того, можна довести, що зловмисник, знаючи деяку послідовність, яка згенерована генератором BBS, не зможе визначити ні попередні до неї біти, ні наступні. Генератор BBS непередбачуваний у лівому та у правому напрямках. Ця властивість дуже корисна для цілей криптографії, що також пов’язана з особливостями розкладання числа M на множники.

Найістотнішим недоліком алгоритму BBS є недостатня швидкодія, що не дозволяє використовувати його в багатьох областях, наприклад, за обчислення у реальному часі, а також, на жаль, і за потокового шифрування.

Проте, цей алгоритм видає дійсно хорошу псевдовипадкову послідовність чисел із великим періодом (за відповідного вибору вхідних параметрів), що дозволяє використовувати його для криптографічних цілей під час генерації ключів для шифрування.

9.3.4. Генератори псевдовипадкових чисел на основі регістрів зсуву зі зворотним зв’язком

У теорії кодування й криптографії широко застосовуються так звані регістри зсуву зі зворотним зв’язком, які використовувалися в апаратурі шифрування ще до початку масового використання електронно-обчислювальних машин і сучасних високошвидкісних програмних шифраторів.

Регістри зсуву зі зворотним зв’язком можуть застосовуватися для отримання потоку псевдовипадкових біт і складаються з двох частин: власне n-бітного регістра зсуву та пристрою генерування функції зворотного зв’язку (рис. 9.2).

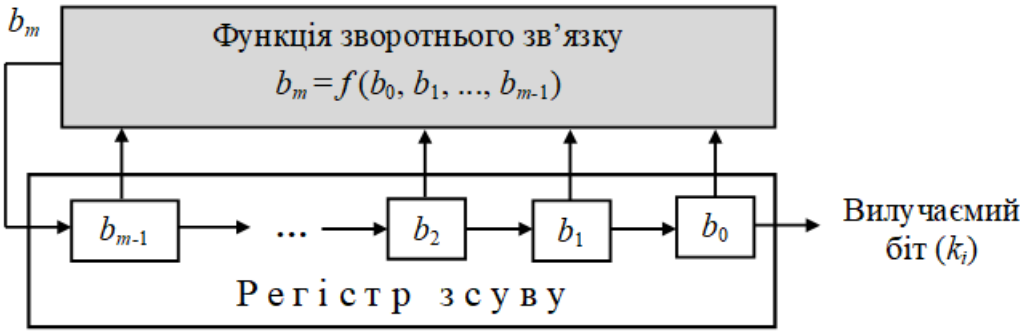


Рис. 9.2. Регістр зсуву зі зворотнім зв'язком

Вилучати біти з регістра зсуву можна тільки по одному (по черзі). Якщо необхідно вилучити наступний біт, усі біти регістра зсуваються вправо на один розряд. При цьому на вхід регістра зліва надходить новий біт, який формується пристроєм зворотного зв'язку й залежить від всіх інших біт регістра зсуву. Завдяки цьому біти регістра змінюються за певним законом, який і визначає схему отримання псевдовипадкової послідовності. Зрозуміло, що через деяку кількість тактів роботи регістра послідовність бітів почне повторюватися. Довжина одержуваної послідовності до початку її повторення називається періодом регістра зсуву.

Потокові шифри з використанням регістрів зсуву досить довго використовувалися на практиці. Це пов'язано з тим, що вони вдало реалізуються за допомогою цифрової апаратури.

Найпростішим видом регістра зсуву зі зворотним зв'язком є лінійний регістр зсуву зі зворотним зв'язком (ЛРЗЗЗ) (linear feedback shift register – LFSR). Зворотній зв'язок у цьому пристрої реалізується просто як додавання за модулем два всіх (або деяких) бітів регістра. Біти, які беруть участь у зворотному зв'язку, утворюють відвідну послідовність. Лінійні регістри зсуву зі зворотним зв'язком або їх модифікації часто застосовується в криптографії.

Для того, щоб стало зрозуміліше, як працює регістр зсуву зі зворотним зв'язком, розглянемо 4-бітовий ЛРЗЗЗ із відведенням від нульового й третього розрядів, який представлено на рис. 9.3.

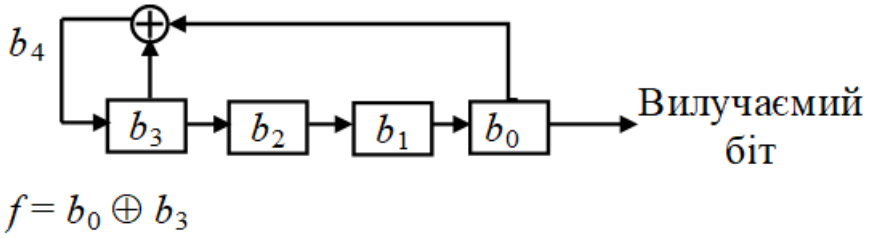


Рис. 9.3. Приклад 4-розрядного лінійного регістра зсуву зі зворотним зв'язком

Запишемо в зображений на рис. 9.3 регістр початкове значення: 1011. Обчислювати послідовність внутрішніх станів регістра зручно за допомогою представленої табл. 9.1. У таблиці відображені перші дев'ять станів регістра.

Таблиця 9.1

Послідовність роботи лінійного регістра зсуву зі зворотним зв'язком

Номер стану	Внутрішній стан регістра b_3, b_2, b_1, b_0	Результат обчислення функції зворотного зв'язку $f = b_0 \oplus b_3$	Біт, який вилучається (b_0)
0	1011	0	1
1	0101	1	1
2	1010	1	0
3	1101	0	1
4	0110	0	0
5	0011	1	1
6	1001	0	1
7	0100	0	0
8	0010	0	0

На кожному кроці весь вміст регістра зсувається вправо на один розряд. При цьому можна отримати в якості результату один біт. На вільне місце ліворуч надходить біт, що дорівнює результату обчислення функції зворотного зв'язку $f = b_0 \oplus b_3$. Вихідну послідовність генератора псевдовипадкових біт утворює останній стовпець таблиці (вилучаємий біт).

Лінійний регістр зсуву зі зворотним зв'язком розміром m бітів може перебувати в одному з 2^m–1 станів. Якщо початкова послідовність складається тільки з нулів, то результат марний – вихідна послідовність

також буде складатись тільки з нулів. Тому, теоретично такий регістр може генерувати псевдовипадкову послідовність із максимальним періодом 2^m–1. Таку послідовність називають *M*-послідовністю.

Лінійний регістр зсуву зі зворотним зв'язком буде генерувати циклічну послідовність бітів із максимальним періодом, якщо він має парну кількість комірок та задається примітивним поліномом. Примітивний поліном – незвідний поліном, який є дільником x^e+1, де e – найменше ціле число у формі $e = 2^k$–1 та $k \geq 2$. Примітивний поліном отримати нелегко. Поліном вибирається випадково, а потім перевіряється на примітивність. Однак існує таблиця перевірених примітивних поліномів. Табл. 9.2 показує незвідні та примітивні поліноми ступенів 1-8. В дужках показані незвідні, але не примітивні поліноми.

Таблиця 9.2

Незвідні та примітивні поліноми ступенів 1-8

Ступінь поліному	Поліноми (в шістнадцятирічному форматі)									
1	3	2								
2	7									
3	B	D								
4	13	19	(1F)							
5	25	29	2F	37	3B	3D				
6	43	(45)	49	57	5B	61	6D	73		
7	83	87	91	9D	A7	AB	B9	BF	C1	CB
	D3	D4	E5	EF	F1	F7	FD			
8	(11B)	11D	12B	12D	(139)	(13F)	14D	15F	163	165
	169	171	(177)	(17B)	187	(18B)	(19F)	(1A3)	1A9	(1B1)
	(1BD)	1CF	(1D7)	(1DB)	1E7	(1F3)	1F5	(1F9)		

Основним недоліком ГПВЧ на базі лінійних регістрів зсуву є складність програмної реалізації. Зсуви й бітові операції легко та швидко виконуються в електронній апаратурі, тому в різних країнах випускаються мікросхеми та пристрої для потокового шифрування на базі алгоритмів із використанням регістрів зсуву зі зворотним зв'язком.

Крім того, ЛРЗЗЗ легко піддаються криптографічному аналізу і не є достатньо надійними для використання у шифруванні. Існує декілька методів проектування генераторів псевдовипадкових послідовностей, які руйнують лінійні властивості і тим самим роблять такі системи криптографічно більш стійкими [4, 5, 34, 36]:

– використання нелінійної функції, яка об'єднує виходи декількох ЛРЗЗЗ;

– використання нелінійної функції, що фільтрує вміст кожної комірки єдиного ЛРЗЗЗ;

– використання виходу ЛРЗЗЗ для управління сінхросигналом одного або декількох ЛРЗЗЗ (алгоритм А5);

– динамічна зміна параметрів ЛРЗЗЗ (довжини регістра і коефіцієнтів зворотного зв'язку).

9.4. Потоковий шифр А5

9.4.1. Історія створення потокового шифру А5

А5 – це потоковий алгоритм шифрування, що використовується для забезпечення конфіденційності даних, які передаються між телефоном і базовою станцією в європейській системі мобільного цифрового зв'язку *GSM* (*Group Special Mobile*).

Шифр заснований на побітовому додаванні за модулем два (булева операція *xor*) псевдовипадкової послідовності, яка генерується, і інформації, що зашифровується. В А5 псевдовипадкова послідовність реалізується на основі трьох ЛРЗЗЗ. Регістри мають кількість розрядів (довжини) 19, 22 і 23 біти відповідно. Зсувами управляє спеціальна схема, яка організує на кожному кроці зсув, як мінімум, двох регістрів, що призводить до нерівномірного руху змісту їх послідовностей. Послідовність формується шляхом операції *xor* над вихідними бітами регістрів.

Спочатку французькі військові фахівці – криптографи розробили потоковий шифр для використання виключно у військових цілях. У кінці 80-х років XX століття для стандарту *GSM* знадобилося створити нову, сучасну систему безпеки. В її основу лягли три секретні алгоритми: аутентифікації – A3; шифрування потоку – A5, генерації ключа сеансу – A8. В якості алгоритму A5 було використано французьку розробку. Цей шифр забезпечував достатній рівень захищеності потоку, що забезпечувало конфіденційність розмови. Спочатку експорт стандарту з Європи не передбачався, але незабаром у цьому з'явилася необхідність. Саме тому, A5 перейменували в A5/1 і стали поширювати в Європі і США. Для інших країн алгоритм модифікували, значно знизивши криптографічну стійкість шифру. A5/2 був спеціально розроблений як експортний варіант для країн, що не входили до Європейського Союзу. В A5/0 шифрування відсутнє зовсім. У даний час розроблено також алгоритм A5/3, заснований на алгоритмі Касумі й затверджений для використання в мережах 3G. Ці модифікації позначають A5/x.

Офіційно ця криптографічна схема не публікувалася і її структура не розголошувалася. Це пов'язано з тим, що розробники покладалися на безпеку за рахунок невідомості, тобто алгоритми складніше зламати, якщо вони не доступні публічно. Дані надавалися операторам *GSM* тільки за потребою. Тим не менш, до 1994 року деталі алгоритму A5 стали відомі: британська телефонна компанія (British Telecom) передала всю документацію, що стосується стандарту, Бредфордському університету для аналізу, не уклавши угоду про нерозголошення інформації. Крім того, матеріали про стандарт з'явилися на одній конференції в Китаї. Як наслідок, його схема поступово "просочилася" в широкі кола. Цього самого року кембриджські вчені Росс Андерсон і Майкл Рое опублікували відновлену за цими даними криптографічну схему й дали оцінку її криптографічній стійкості. Остаточно алгоритм був представлений у роботі Йована Голіча на конференції Eurocrypt'97 [4, 5].

9.4.2. Потокове шифрування даних за допомогою A5/1

A5/1 використовується в Глобальній системі мобільного зв'язку (*GSM*). Телефонний зв'язок у *GSM* здійснюється як послідовність кадрів на 228 бітів, при цьому кожний кадр – 4,6 мілісекунди. A5/1 створює потік бітів, виходячи з ключа довжиною 64 біти. Розрядні потоки зібрані в буфери по 228 бітів, щоб додавати їх за модулем два з кадром на 228 бітів, як показано на рис 9.4.

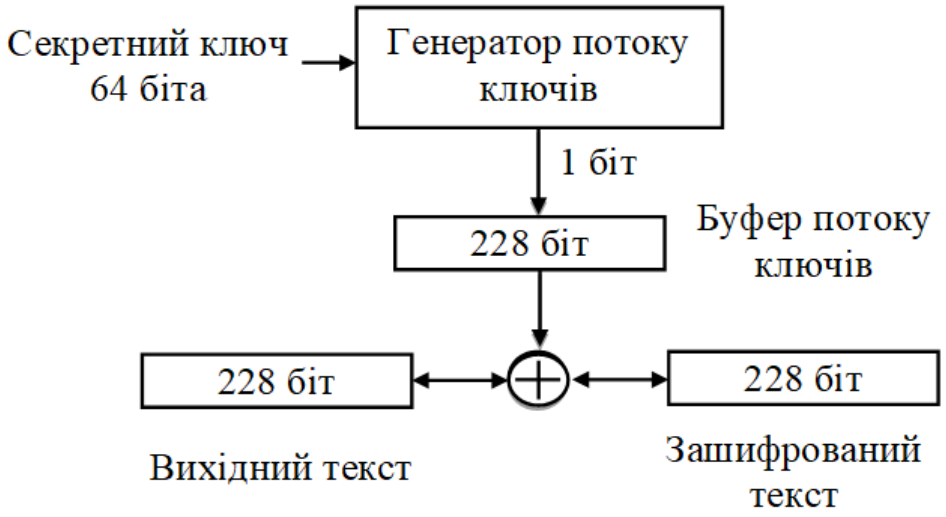


Рис. 9.4. Принцип побудови алгоритму потокового шифрування A5/1

На рис. 9.4 стрілки в обидві сторони показують, що під час шифрування беруть відкриті дані, а після додавання з потоком ключів за модулем два виходять зашифровані дані; під час розшифрування беруть зашифровані дані, а після додавання з потоком ключів за модулем два виходять відкриті дані.

У цьому алгоритмі кожному символу відкритого повідомлення відповідає символ зашифрованого повідомлення. Повідомлення не ділиться на блоки (як у блоковому шифруванні) і не змінюється в розмірі. Для спрощення апаратної реалізації і, отже, збільшення швидкодії, використовуються тільки найпростіші операції: додавання за модулем два *(xor)* і зсув бітів регістра. Формування вихідної послідовності відбувається шляхом додавання потоку вихідного повідомлення з послідовністю, яка

генерується (гамою). Особливість операції *xor* полягає в тому, що застосована парне число разів вона приводить до початкового значення. Звідси, розшифрування повідомлення відбувається шляхом додавання зашифрованого повідомлення з відомою послідовністю (гамою).

Генератор потоку ключів алгоритму A5/1 [4, 5, 36, 40] складається з трьох ЛРЗЗЗ над полем $GF(2)$ загальною довжиною 64 біти (рис. 9.5).

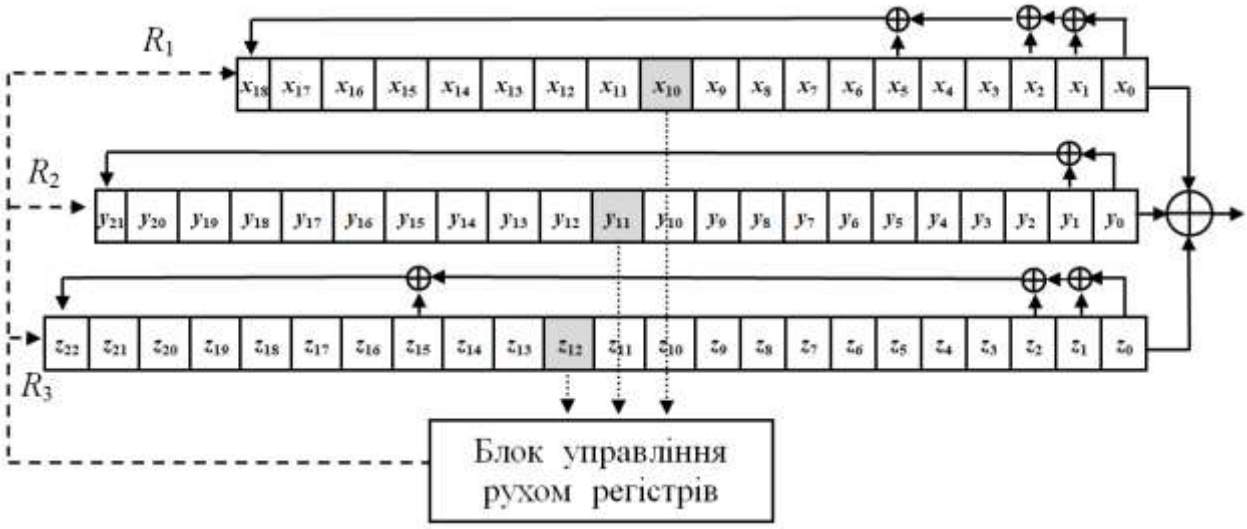


Рис. 9.5. Генератор потоку ключів алгоритму A5/1

Довжина регістра R_1 становить 19 біт, R_2 – 22 біта і R_3 – 23 біта. Вихідна послідовність формується шляхом додавання за модулем два виходів усіх регістрів. Нелінійність послідовності забезпечується за рахунок застосування схеми нерівномірного руху кожного з регістрів. Управління рухом виконується за рахунок вмісту самих регістрів, на основі значень одного з осередків у кожному з регістрів. Поліноми, що визначають зворотні зв’язки для цих регістрів, і номери бітів для управління рухом наведені у табл. 9.3.

Як ключ шифрування в A5/1 використовується 64-бітовий вектор, значення бітів кожного з регістрів R_1, R_2 і R_3 визначається як деяка лінійна комбінація значень координат. Крім ключа шифрування, як вектор ініціалізації використовується значення, отримане з 22-бітового номера кадру даних, переданих між базовою станцією та мобільним терміналом.

Таблиця 9.3

Параметри генератора потоку ключів алгоритму A5/1

Регістр	Довжина регістру	Поліном	Номер біта управління рухом регістру
R_1	19	$f_1(x) = x^{19} \oplus x^5 \oplus x^2 \oplus x \oplus 1$	10
R_2	22	$f_2(x) = x^{22} \oplus x \oplus 1$	11
R_3	23	$f_3(x) = x^{23} \oplus x^{15} \oplus x^2 \oplus x \oplus 1$	12

Блок управління рухом регістрів A5/1 працює таким чином. На основі значень бітів управління обчислюється мажоритарна функція, і для зсуву вибираються регістри, у яких значення бітів управління збігаються зі значенням мажоритарної функції на поточному кроці.

Після початкової ініціалізації регістрів R_1, R_2 і R_3 на основі значень ключа шифрування та номерів кадру відбувається робота генератора із застосуванням блоку управління рухом регістрів протягом 100 тактів, при яких отримані на виході значення не використовуються. Потім формуються 228 бітів гами, причому перші 114 застосовуються для шифрування переданого фрейму, наступні 114 – для розшифрування прийнятого.

9.4.3. Криптографічна стійкість потокового шифру A5

Розробка стандарту *GSM* мала на увазі потужний апарат шифрування, що не піддається зламу (особливо в реальному часі). Використовувані розробки за належної реалізації забезпечували якісне шифрування даних, які передаються. Саме таку інформацію можна отримати від компаній, які поширюють цей стандарт. Але варто зазначити важливий нюанс: прослуховування розмов – невід'ємний атрибут, який використовується спецслужбами. Вони були зацікавлені в можливості прослуховування телефонних розмов для своїх цілей. Отже, в алгоритм були внесені зміни, що дають можливість зламу за прийнятний час. Крім цього, для експорту A5

модифікували в A5/2. У MoU (Memorandum of Understand Group Special Mobile standard) визнають, що метою розробки A5/2 було зниження криптографічної стійкості шифрування, однак в офіційних результатах тестування кажуть, що невідомо деякі недоліки алгоритму [36].

З появою даних про стандарт A5 почалися спроби зламу алгоритму, а також пошуку вразливостей. Величезного значення надали особливостям стандарту, які різко послаблюють захист, а саме:

– 10 бітів ключа примусово занулені;

– відсутність перехресних зв'язків між регістрами (крім управління зсувами);

– зайва надмірність службової інформації, що шифрується та яка відома криптографічному аналітику;

– понад 40% ключів призводить до мінімальної довжини періоду генерованої послідовності, а саме $(2^{23}-1)\cdot 3/4$ [24];

– напочатку сеансу здійснюється обмін нульовими повідомленнями (по одному кадру);

– в A5/2 управління зсувами здійснюється окремим регістром довжиною 17 бітів.

На основі цих "дірок" в алгоритмі побудовані схеми зламу.

Ключем є сесійний ключ довжиною 64 біта, номер кадру вважається відомим. Таким чином, складність атаки, заснованої на прямому переборі, дорівнює 2^{64}.

Перші огляди шифру (робота Росса Андерсона) відразу виявили вразливість алгоритму – через зменшення ефективної довжини ключа (занулення 10 бітів) складність стала 2^{45} (відразу на 6 порядків). Атака Андерсона заснована на припущенні щодо початкового заповнення коротких регістрів і за вихідними даними отримання заповнення третього.

У 1997 році Йован Голіч опублікував результати аналізу A5. Він запропонував спосіб визначення початкового заповнення регістрів за

відомим відрізком гами довжиною лише 64 біта, який отримують із нульових повідомлень. Атака має середню складність 2^{40} [24].

У 1999 році Вагнеру і Голдбергу без зусиль вдалося продемонструвати, що для розкриття алгоритму A5/2, достатньо перебором визначити початкове заповнення додаткового регістра управління зсувами. Перевірка здійснюється завдяки нульовим кадрам. Складність цієї атаки дорівнює 2^{17}, отже, на сучасному комп'ютері злам шифру займає кілька секунд.

У грудні 1999 року група ізраїльських вчених Аді Шамір, Алекс Бірюков, а пізніше й американець Девід Вагнер показали, що атака на основі відомих малих вихідних текстів дозволяє в реальному часі знаходити ключ за декілька хвилин, але для цього потрібно провести етап попередньої обробки із 2^{48} кроками.

9.5. Потоковий шифр RC4

9.5.1. Історія створення шифру RC4

Алгоритм RC4 розробив Рональд Ривест у 1987 році спеціально як генератор потоку ключової інформації з ключем змінної довжини. Хоча офіційне скорочення – Rivest Cipher 4, його часто вважають скороченням від Ron's Code [27, 36, 40].

Шифр був комерційною таємницею, але у вересні 1994 року його опис було анонімно відправлено в розсилку Cypherpunks [27, 36, 40]. Незабаром опис RC4 було опубліковано в ньюс-групі sci.crypt. Саме звідти вихідний код потрапив до багатьох сайтів у мережі Інтернет. Опублікований шифр давав ті самі зашифровані дані на виході, які давав справжній RC4. Напевно, ці дані були отримані в результаті аналізу виконуваного коду. Опублікований шифр сумісний із наявними продуктами, які використовують RC4, а деякі учасники телеконференції, які мали, за їхніми словами, доступ до вихідного коду RC4,

підтвердили ідентичність алгоритмів за розбіжностей у позначеннях і структурі програми.

Оскільки даний алгоритм став відомим, він більше не становив комерційної таємниці. Однак назва RC4 є торговою маркою компанії RSA. Тому іноді цей шифр називають ARCFOUR або ARC4 (маючи на увазі, Alleged RC4 – передбачуваний RC4, оскільки RSA офіційно не опублікувала алгоритм), щоб уникнути можливих претензій з боку власника торгової марки.

Головними факторами, що сприяли широкому застосуванню RC4, були простота його апаратної і програмної реалізації, а також висока швидкість роботи алгоритму в обох випадках. RC4 широко застосовується у багатьох системах передачі даних та протоколах організації мереж, наприклад, протоколах SSL і TLS та алгоритмах забезпечення безпеки бездротових мереж WEP і WPA.

У США довжина ключа для використання усередині країни рекомендується такою, що дорівнює 128 бітів, але угода, укладена між асоціацією видавців програмного забезпечення (Software Publishers Association – SPA) і урядом США, дає RC4 спеціальний статус, який дозволяє експортувати шифри з довжиною ключа до 40 бітів. 56-бітові ключі дозволено використовувати закордонним відділенням американських компаній.

9.5.2. Опис алгоритму RC4

RC4 – байт-орієнтований шифр потоку, у якому байт вихідного тексту додається за модулем два з байтом ключа, щоб отримати байт зашифрованого тексту. Ключ засекречування, з якого генеруються однобайтові ключі в потоці ключів, може містити від 1 до 256 байтів.

RC4 базується на понятті матриці станів. У кожний момент матриця станів (256 байтів) активізується, з неї випадково вибирається один байт,

який буде ключем для шифрування. Ідея може бути показана у вигляді масиву байтів:

$$S[0], S[1], S[2], \ldots, S[254], S[255].$$

Зауважимо, що індекси діапазону елементів – між 0 та 255. Зміст кожного елемента – байт (8 бітів), який може інтерпретуватися як ціле число від 0 до 255.

На рис. 9.6 показано принцип побудови потокового шифру RC4. Перші два блоки виконуються тільки один раз (ініціалізація); перестановки для того, щоб створювати ключ потоку, повторюються, поки є байти вхідних даних, призначені для шифрування.

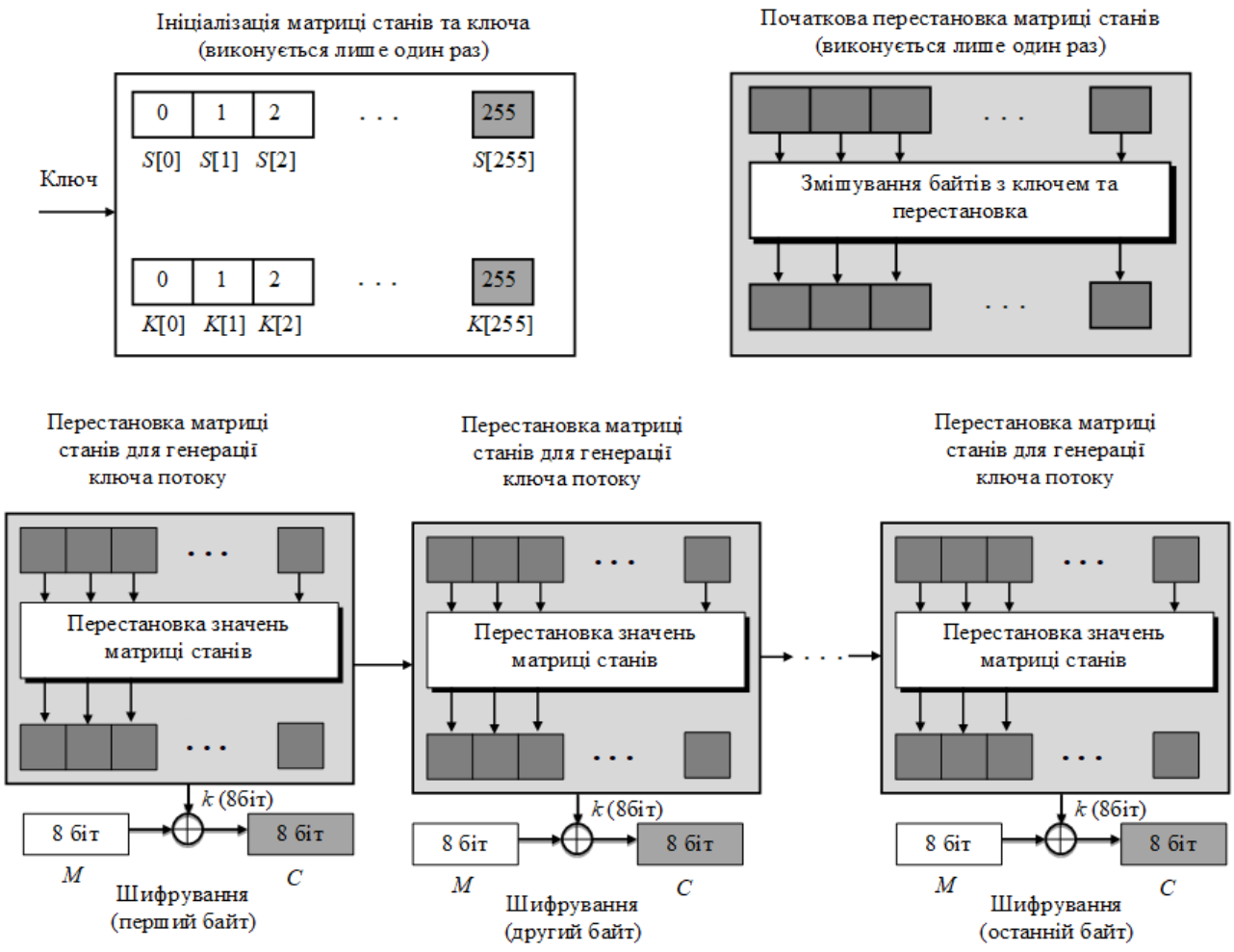


Рис. 9.6. Принцип побудови потокового шифру RC4

Алгоритм RC4 включає в себе два етапи. На першому, підготовчому етапі проводиться ініціалізація таблиці замін *S* (матриці станів), масиву ключів *K*, а також початкова перестановка матриці стану, яка заснована на значенні байтів у *K* [*i*]. На другому, основному етапі обчислюються безпосередньо псевдовипадкові числа *k*.

Ініціалізація матриці станів та масиву ключів

Ключ у RC4 являє собою послідовність байтів довільної довжини, по якій будується початковий стан шифру S – перестановка всіх 256 байтів.

Алгоритм ініціалізації RC4, який також називають алгоритмом ключового розкладу (англ. Key-Scheduling Algorithm or KSA), наведений на рис. 9.7.

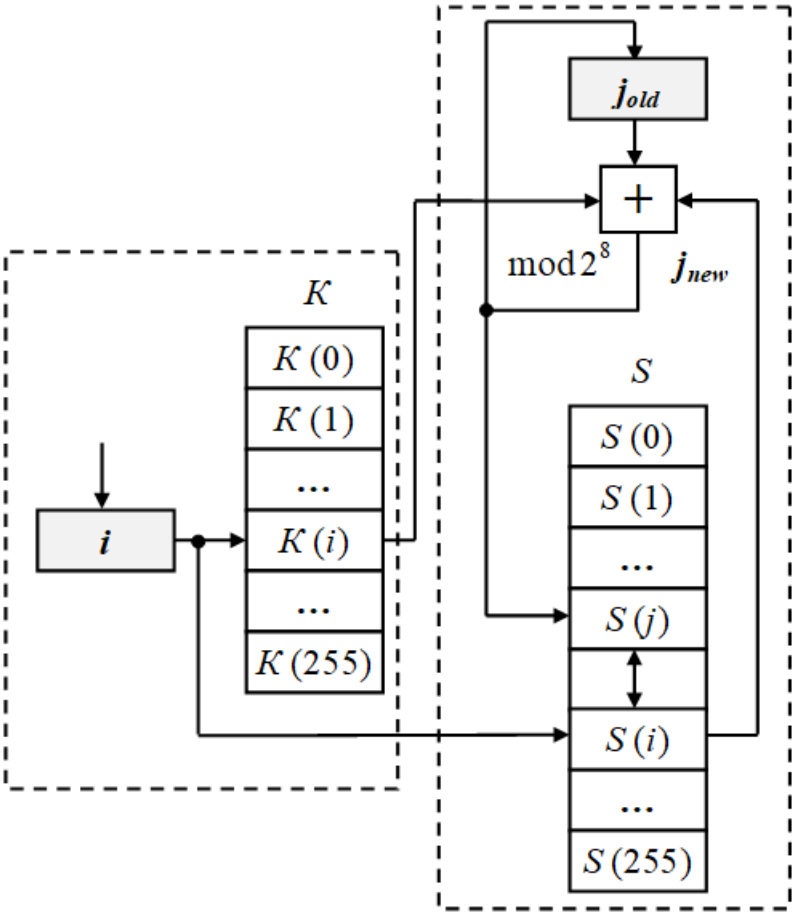


Рис. 9.7. Схема ініціалізації матриці станів *S* і масиву ключа *K*

Матриця станів ініціалізується для значень 0, 1, 2, ..., 254, 255. Створюється також маси в ключів $K[0]$, $K[1]$, $K[2]$, ..., $K[254]$, $K[255]$. Якщо ключ шифрування має точно 256 байтів, то байти копіюються у масив K; інакше – байти повторюються, поки не заповниться масив K.

Спочатку матриця станів S заповнюється послідовними значеннями від 0 до 255. Потім кожний наступний елемент S обмінюється місцями з елементом, номер якого визначається елементом ключа K, самим елементом і сумою номерів елементів, з якими відбувався обмін на попередніх ітераціях. Значення лічильників i та j спочатку дорівнюють 0.

Нижче показано програму на псевдокоді, яка ініціалізує матрицю станів S і масив ключів K:

```
for (i = 0 to 255)
{
        S [i] ← i
        K [i] ← Key [i mod LengthKey]
}
```

Далі у матриці станів відбувається перестановка (скремблювання елементів), заснована на значенні байтів у $K[i]$. Ключовий байт використовується тільки на цьому кроці, щоб визначити, які елементи потрібно замінити. Після цього кроку байти матриці повністю перетасовані.

Нижче показано програму на псевдокоді, яка здійснює перестановку байтів матриці станів:

```
j ← 0
for (i = 0 to 255)
{
        j ← (j + S [i] + K [i]) mod 256
        swap (S [i], S [j])
}
```

Перестановка матриці станів та генерація ключового потоку

Ключі *k* у ключовому потоці генеруються один за одним. Спочатку елементи матриці станів переставляються на основі значень своїх елементів і значень двох індивідуальних змінних *i* та *j*. Потім значення двох елементів матриці станів у позиціях *i* та *j* використовуються, щоб визначити індекс елемента матриці станів, який служить в якості ключа *k*. Наступний код повторюється для кожного байта початкових даних тексту, щоб створити новий ключовий елемент у ключовому потоці. Ядро алгоритму RC4 складається з функції генерації ключового потоку (рис. 9.8).

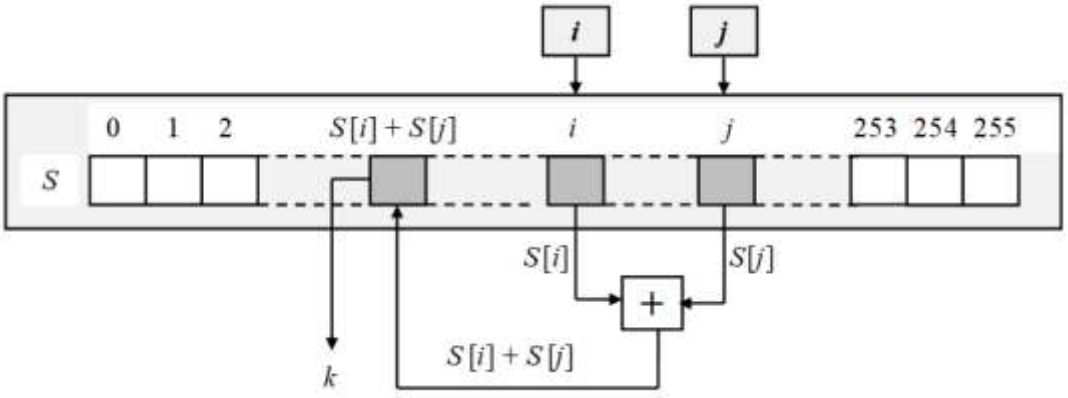


Рис. 9.8. Ядро алгоритму RC4

Таблиця замін (матриця станів *S*) повільно змінюється під час використання, при цьому лічильник *i* (змінна *i*) забезпечує зміну кожного елемента таблиці, а лічильник *j* (змінна *j*) гарантує, що елементи таблиці змінюються випадково.

Нижче показано програму на псевдокоді, яка здійснює перестановку байтів матриці станів *S* [*i*] і генерує ключовий потік *k*:

$i \leftarrow 0; \quad j \leftarrow 0$

$i \leftarrow (i + 1) \bmod 256$

$j \leftarrow (j + S[i]) \bmod 256$

swap ($S[i]$, $S[j]$)

$k \leftarrow S[(S[i] + S[j]) \bmod 256]$

Зашифрування (розшифрування) даних

Після того, як ключовий потік *k* був створений, байт відкритих даних *M* додається за модулем два з *k*, щоб створити байт зашифрованих даних *C*. Розшифрування даних полягає в регенерації ключового потоку *k* та додаванні його за модулем два із байтом зашифрованих даних *C*. Завдяки властивостям операції додавання за модулем два на виході отримаємо вихідні дані *M*.

Приклад 9.3. Щоб показати випадковість ключа потоку, використовуємо ключ засекречування з усіма нульовими байтами. Ключовий потік для 20 значень у такому випадку буде дорівнювати: 222, 24, 137, 65, 163, 55, 93, 58, 138, 6, 30, 103, 87, 110, 146, 109, 199, 26, 127, 163.

Приклад 9.4. Повторимо приклад 9.3, але нехай ключем засекречування буде п’ять байтів (15, 202, 33, 6, 8). Ключовий потік: 248, 184, 102, 54, 212, 237, 186, 133, 51, 238, 108, 106, 103, 214, 39, 242, 30, 34, 144, 49. Знову випадковість у ключовому потоці очевидна.

Аналіз процесів перетворення даних в алгоритмі RC4

RC4 – фактично є класом алгоритмів, що визначаються розміром його блоку або слова – параметром *n*. Зазвичай $n = 8$, але можна використовувати й інші значення. Для спрощення аналізу алгоритму приймемо $n = 4$. Внутрішній стан RC4 складається з масиву розміром 2^n слів і двох лічильників, кожний розміром в одне слово. Обидва лічильника (за $n = 4$) 4-бітові. Позначимо їх як *i* та *j*. Усі обчислення проводяться за модулем 2^n.

Приклад 9.5. Нехай початкове значення ключа $K = (12, 2, 3, 8)$. Показати процес ініціалізації матриці станів і масиву ключів, а також вихідну перестановку матриці станів.

Рішення. На рис. 9.9 показано приклад ініціалізації 4-розрядних матриці станів і масиву ключів та вихідну перестановку матриці станів (9 тактів з 16).

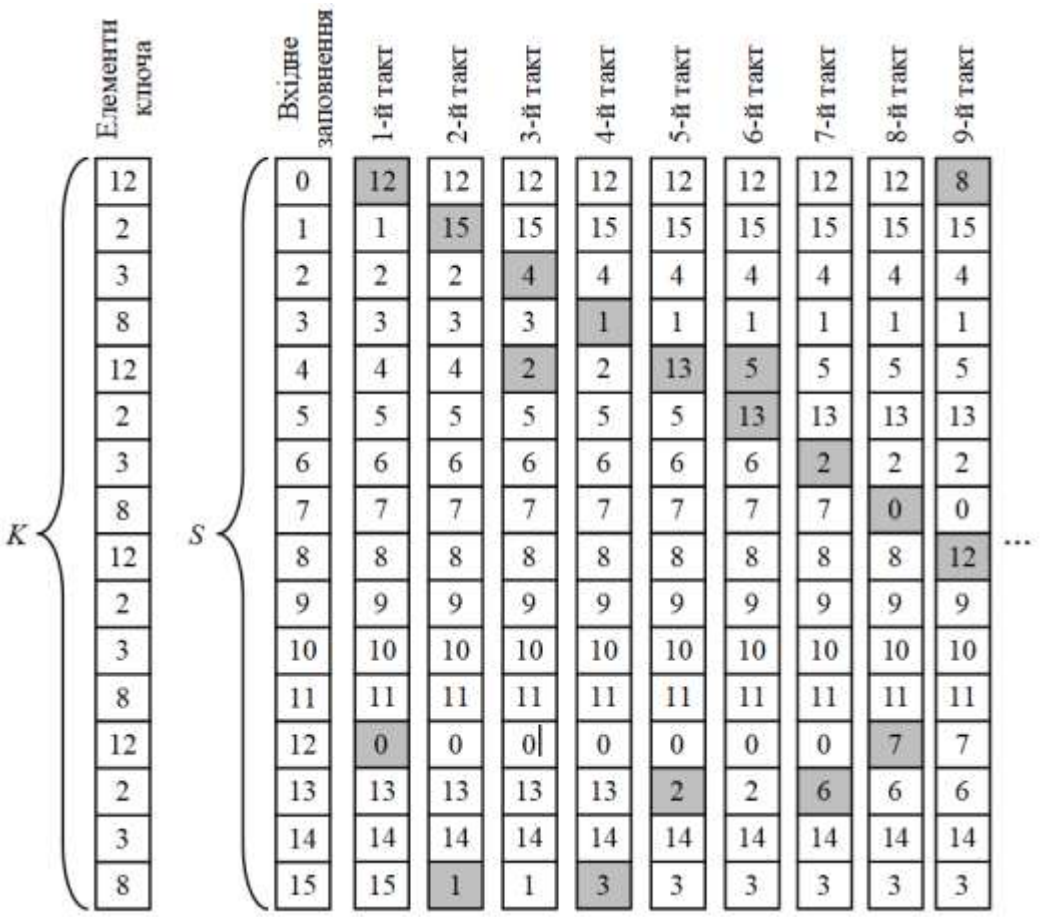


Рис. 9.9. Послідовність тактів перестановки матриці станів

Приклад 9.6. Показати приклад роботи 4-розрядного генератора псевдовипадкових чисел RC4 за заданих значеннях *i* та *j* і заповненні 4-розрядної таблиці замін *S*.

Рішення. Приклад роботи 4-розрядного генератора псевдовипадкових чисел RC4 зображено на рис. 9.10. На виході генератора формується 4-розрядна послідовність: 4, 5, 2, 0, 13, 5, 8, 4, 1, ...

У розглянутому прикладі розмір *n* слова або блока алгоритму дорівнював чотирьом. Це значення можна брати і іншим, наприклад, 8 або 16. У разі використання $n = 8$ таблиця замін *S* повинна складатися з $2^8 = 256$ значень, а елементами таблиці замін повинні бути числа від 0 до 255. Розмір лічильників *i* та *j* необхідно також змінити до восьми біт (максимальне значення – 255). Крім того, всі обчислення у разі $n = 8$ необхідно виконувати за модулем 256. Аналогічні зміни в алгоритмі необхідно виконувати і при інших значеннях параметра *n*.

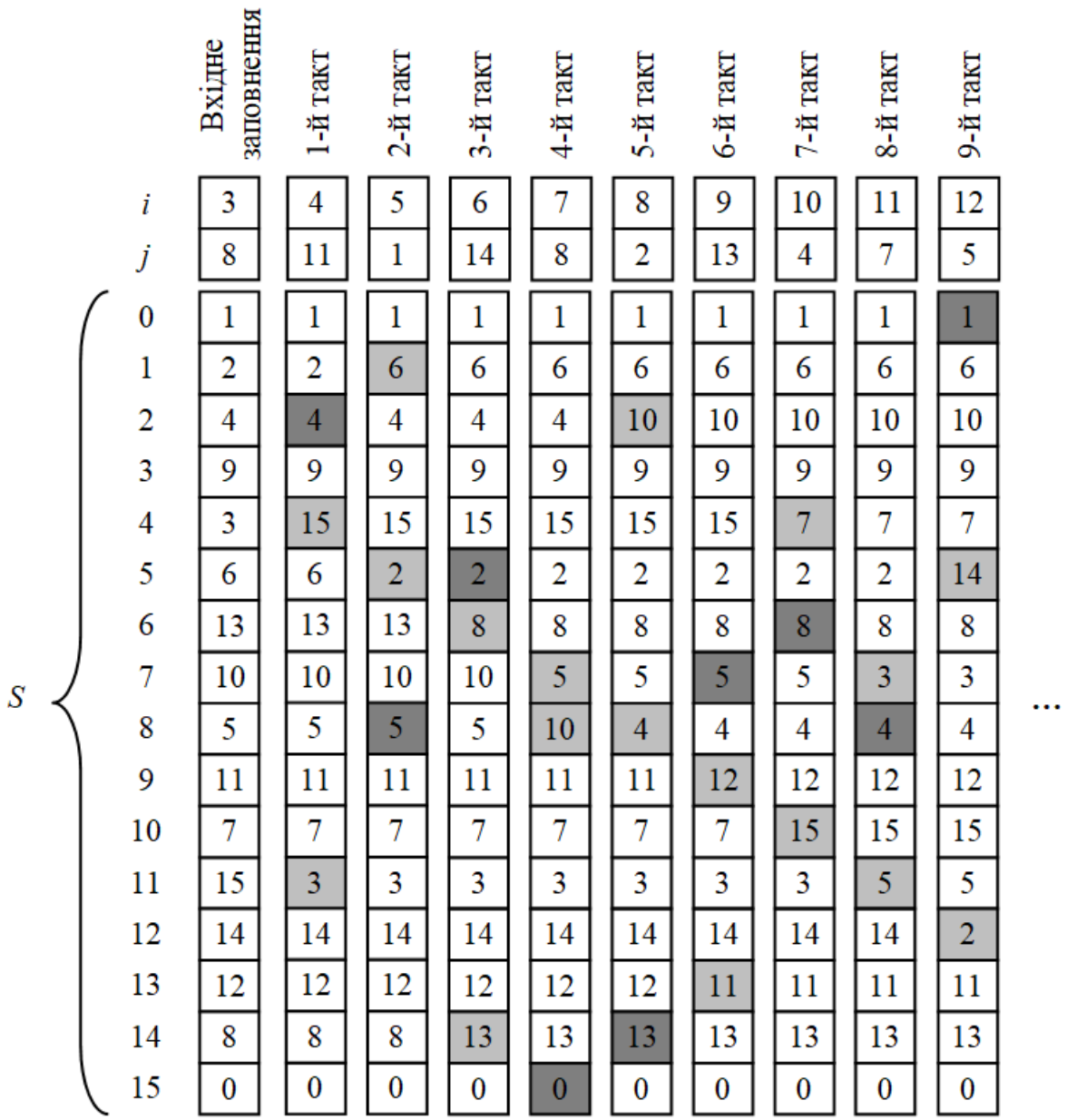


Рис. 9.10. Приклад роботи 4-розрядного ГПВЧ RC4

9.5.3. Криптографічна стійкість потокового шифру RC4

Алгоритм RC4 ретельно вивчався криптографічними аналітиками. У ньому не виявлено будь-яких слабких місць. Крім високої стійкості до криптографічного аналізу, цей алгоритм дуже швидкий і може використовуватися для генерації ключової послідовності за потокового шифрування.

Усі методи криптографічного аналізу потокових шифрів зазвичай поділяють на три класи:

1. *Силові* (атака “грубою силою”). Атаки шляхом повного перебору (перебір усіх можливих варіантів). Складність повного перебору залежить від кількості всіх можливих розв’язків задачі (розміру простору ключів або простору відкритих даних). Цей вид атаки застосовується до всіх видів систем потокового шифрування. Розробляючи системи шифрування, розробники прагнуть зробити так, щоб цей вид атак був найбільш ефективним у порівнянні з іншими існуючими методами зламу.

2. *Статистичні.* Діляться на два підкласи:

– метод криптографічного аналізу статистичних властивостей шифрувальної гами, спрямований на вивчення вихідної послідовності криптографічної системи (криптографічний аналітик намагається встановити значення наступного біта послідовності з ймовірністю вище ймовірності випадкового вибору за допомогою різних статистичних тестів);

– метод криптографічного аналізу складності послідовності: криптографічний аналітик намагається знайти спосіб генерувати послідовність, аналогічну гамі, але простіше реалізованим способом.

Обидва методи використовують принцип лінійної складності.

3. *Аналітичні методи.* Цей вид атак розглядається в припущенні, що криптографічному аналітику відомі опис генератора, відкриті й відповідні закриті дані. Завдання криптографічного аналітика – визначити використаний ключ (початкове заповнення регістрів).

Відомо, що шифр є безпечним, якщо розмір ключа принаймні 128 бітів (16 байтів). Це підтверджується повідомленнями про деякі атаки для малих розмірів ключів (менше ніж 5 байтів). Протоколи, які сьогодні використовують RC4, встановлюють такий розмір ключів, що роблять їх безпечними. Однак, як і для багатьох інших шифрів, рекомендується, щоб різні сеанси використовували різні ключі. Це перешкоджає криптоаналітику використовувати диференціальний криптоаналіз шифру.

9.6. Алгоритм симетричного потокового перетворення ДСТУ 8845:2019

9.6.1. Загальні положення ДСТУ 8845:2019

Національний стандарт симетричного потокового перетворення ДСТУ 8845:2019 був прийнятий наказом Державного підприємства "Український науково-дослідний і навчальний центр проблем стандартизації, сертифікації та якості" від 2 квітня 2019 року № 85 та введений в дію з 1 жовтня 2019 року.

У цьому стандарті генератор псевдовипадкових чисел (ключового потоку), позначений як "СТРУМОК". В основі алгоритму "СТРУМОК" лежить класична схема підсумовуючого генератора [10, 34, 39], подібна генератору SNOW-2.0, який визначено в ДСТУ ISO/IEC 18033-4:2015 [10]. Алгоритм "СТРУМОК" використовує 256-бітний вектор ініціалізації (*IV*) та 256-бітовий або 512-бітовий 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 таким, що дорівнює 64 бітам.

Залежно від довжини секретного ключа застосовують два режими генерації ключових потоків:

– режим із 256-бітовим ключем, що позначається як "СТРУМОК-256";

– режим із 512-бітовим ключем, що позначається як "СТРУМОК-512".

9.6.2. Генератор ключових потоків "СТРУМОК"

Основними структурними компонентами генератору є лінійний регістр зсуву зі зворотним зв'язком та скінченний автомат (finite-state machine, FSM), у якому виконується нелінійне перетворення. Вхідні дані використовуються

для ініціалізації змінної стану S_i ($i \geq 0$), що складається з вісімнадцяти 64-бітових блоків, до складу яких входять дві компоненти:

– 16 змінних $s^{(i)}$ – комірок ЛРЗЗЗ: $s^{(i)} = (s_{15}^{(i)}, s_{14}^{(i)}, ..., s_0^{(i)})$;

– два регістри скінченого автомату $r^{(i)}$: $r^{(i)} = (r_2^{(i)}, r_1^{(i)})$.

На виході отримуємо ключовий потік (гаму шифру), що формується з 64-бітових слів Z_i. Схематичне зображення генератора ключових потоків “СТРУМОК” у режимі генерації гами шифру наведено на рис. 9.11.

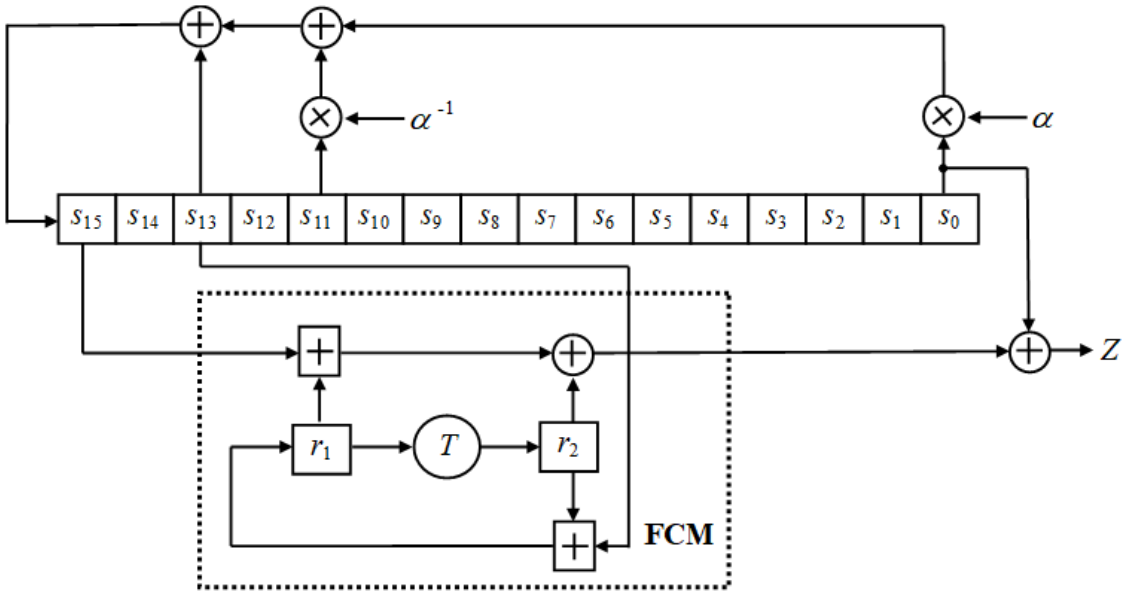


Рис. 9.11. Схема генератора ключових потоків “СТРУМОК” у режимі генерації гами шифру

Відводи зворотного зв’язку в ЛРЗЗЗ будуються 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 $GF(2^8)$ полінома:

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

У 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 елемента β поля $GF(2^8)$, тобто β – корінь полінома $p(y)$.

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 ЛРЗЗЗ як факторкільце $GF(2^{64})[x] / (f(x))$,

– 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 ЛРЗЗЗ є максимальним і дорівнює $2^{1024} - 1$.

Структурно в алгоритмі "СТРУМОК" можна виділити три основні функції:

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

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

– функція наступного стану *Next*, що приймає на вхід змінну стану

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

та виробляє в якості вихідних даних наступне значення змінної стану

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

Функцію *Next* можна виконувати у двох режимах, в залежності від способу виконання ітерації: звичайному (режим генерації вихідних даних) або у режимі ініціалізації;

– функція ключового потоку *Strm*, що приймає на вхід змінну стану

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

та виробляє в якості вихідних даних 64-бітовий ключовий потік Z_i.

Функція ініціалізації внутрішнього стану Init

Функція ініціалізації внутрішнього стану *Init* описується наступним чином.

Вхід: 256 або 512-бітовий ключ K, 256-бітовий вектор ініціалізації IV.

Вихід: початкове значення змінної стану $S_0 = (s^{(0)}, r^{(0)})$.

Ключ для режиму “СТРУМОК-256” можна представити у вигляді чотирьох 64-бітових слів:

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

а для режиму “СТРУМОК-512” – у вигляді восьми 64-бітових 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 та 512 біт, – найбільш значущі слова, а K_0 – найменш значущі.

Вектор ініціалізації можна представити у вигляді чотирьох 64-бітових слів:

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

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

1. До 16 комірок ЛРЗЗЗ заноситься значення ключа.

Для версії з 256-бітовим ключем K виконуються операції:

$$s_{15}^{(-33)} = \overline{K}_0,\ s_{14}^{(-33)} = K_1,\ s_{13}^{(-33)} = \overline{K}_2,\ s_{12}^{(-33)} = K_3,\ s_{11}^{(-33)} = K_0,\ s_{10}^{(-33)} = \overline{K}_1,$$
$$s_9^{(-33)} = K_2,\ s_8^{(-33)} = K_3,\ s_7^{(-33)} = \overline{K}_0,\ s_6^{(-33)} = \overline{K}_1,\ s_5^{(-33)} = K_2 \oplus IV_3,$$
$$s_4^{(-33)} = K_3, s_3^{(-33)} = K_0 \oplus IV_2,\ s_2^{(-33)} = K_1 \oplus IV_1,\ s_1^{(-33)} = K_2,\ s_0^{(-33)} = K_3 \oplus IV_0,$$

де $\overline{K}_i$ – оператор заперечення,

$\oplus$ – оператор побітового додавання за модулем два (*xor*).

Для версії з 512-бітовим ключем K виконуються операції:

$$s_{15}^{(-33)} = K_0,\ s_{14}^{(-33)} = \overline{K}_1,\ s_{13}^{(-33)} = K_2,\ s_{12}^{(-33)} = K_3,\ s_{11}^{(-33)} = \overline{K}_7,\ s_{10}^{(-33)} = K_5,$$
$$s_9^{(-33)} = \overline{K}_6,\ s_8^{(-33)} = K_4 \oplus IV_3,\ s_7^{(-33)} = \overline{K}_0,\ s_6^{(-33)} = K_1,\ s_5^{(-33)} = K_2 \oplus IV_2,$$
$$s_4^{(-33)} = K_3, s_3^{(-33)} = K_4 \oplus IV_1,\ s_2^{(-33)} = K_5,\ s_1^{(-33)} = K_6,\ s_0^{(-33)} = K_7 \oplus IV_0.$$

2. Виконуються 32 ініціюючих такти без генерації ключового потоку, тобто чотири повних 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*, $S_{-33} = (s^{(-33)}, r^{(-33)})$ – значення змінної стану: обчислені на попередньому кроці 16 комірок регістру зсуву $s^{(-33)}$ та початкові нульові значення двох регістрів $r^{(-33)} = (r_2^{(-33)}, r_1^{(-33)})$ скінченного автомату, що в шістнадцятковому поданні мають вигляд $r^{(-33)}) = (0000000000000000, 0000000000000000)$.

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 наступного стану *Next* описується таким чином.

Вхід: змінна стану $S_i = (s^{(i)}, r^{(i)})$, вибраний режим (звичайний або режим ініціалізації). За замовчуванням використовують звичайний режим.

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

$$r_1^{(i+1)} = r_2^{(i)} \boxplus s_{13}^{(i)},$$

де $\boxplus$ – оператор додавання цілих чисел за модулем 2^{64}.

3. Оновлюється значення 15 комірок ЛРЗЗЗ:

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

для всіх $j = 0, 1, \ldots, 14$.

4. Оновлюється значення 16-ої комірки ЛРЗЗЗ. 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 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

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

Схематичне зображення генератора ключових потоків “СТРУМОК” при виконанні функції *Next* у режимі ініціалізації представлено на рис. 9.12.

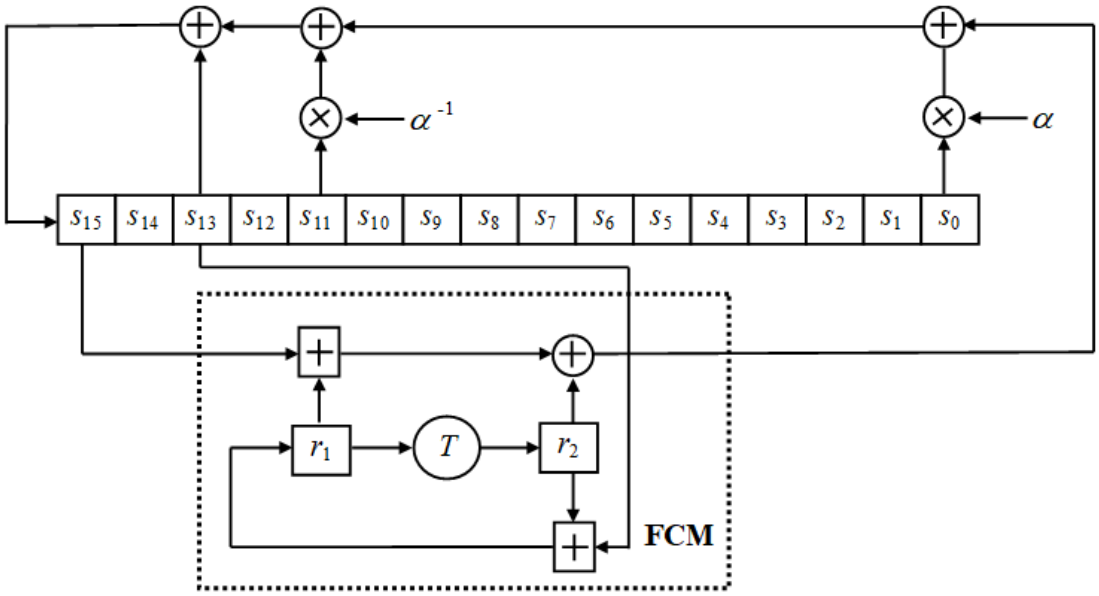


Рис. 9.12. Схема генератора ключових потоків “СТРУМОК” при виконанні функції *Next* у режимі ініціалізації

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 Z_i.

1. Обчислюється значення:

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

2. Виводиться вихідне $\boldsymbol{Z_i}$.

Функція скінченного автомату FSM

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

Вхід: три 64-бітові рядки x, y і z.

Вихід: 64-бітовий рядок m.

1. Обчислюється значення $m = (x \boxplus y) \oplus z$.

2. Виводиться вихідне значення m.

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 [7].

Вхід: 64-бітовий рядок w.

Вихід: 64-бітовий рядок q.

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 підблока w_j виконується підстановка з алгоритму ДСТУ 7624:2014 за допомогою чотирьох табличних перетворень S_0, S_1, S_2, S_3 (табл. 8.3 – 8.6). Приклад виконання підстановки за допомогою цих перетворень схематично (на прикладі підблоків w_j, що подано у шіснадцятковому вигляді) зображено на рис. 9.13.

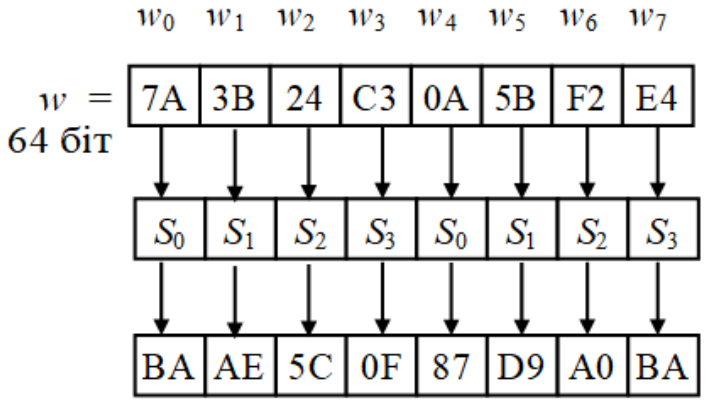


Рис. 9.13. Приклад виконання підстановки

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

$$r_j = S_{j \bmod 4}[w_j],$$

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 $GF(2^8)$, заданого як фактор-кільце $GF(2)[y] / (p(y))$.

4. Виводиться вихідне значення q, яке інтерпретується як 64-бітовий рядок.

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 у 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

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

де $w << 8$ є результатом зсуву ліворуч (в бік старших розрядів) 64-бітового рядка w на 8 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 [9];

Mul_α [c] – 64-бітове значення таблиці передобчислень у рядку з індексом c, де c представляє елемент поля $GF(2^8)$; Mul_α [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 передобчислень $Mul_{\alpha^{-1}}$ 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

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

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

$w \,\&\, \gamma$ 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 $w \,\&\, \gamma$

інтерпретуються як елемент поля $GF(2^8)$ для індексації таблиці передобчислень $Mul_{\alpha\text{-}1}$;

$Mul_{\alpha\text{-}1}$ – таблиця-константа з 256 рядків по 64 біта в кожному [9];

$Mul_{\alpha\text{-}1}[c]$ – 64-бітне значення таблиці передобчислень у рядку з індексом c, де c представляє елемент поля $GF(2^8)$; $Mul_{\alpha\text{-}1}[c]$ представляє елемент поля $GF(2^{64})$.

2. Виводиться вихідне значення w'.

9.6.3. Безпека ДСТУ 8845:2019

У табл. 9.3 представлено установлені рівні захисту генераторів ключових потоків алгоритму симетричного потокового перетворення ДСТУ 8845:2019 за умови, що довжина відрізку гами, яка виробляється за будь-яких фіксованих ключів та векторів ініціалізації, не перевищує 2^{80} [9].

Таблиця 9.3

Рівні захисту генераторів ключових потоків

Генератор ключових потоків (рівень захисту)	Довжина ключа	Обчислювальна складність для найкращої відомої атаки	Обчислювальна складність квантового криптоаналіза
"СТРУМОК-256" (високий)	256	2^{256}	2^{128}
"СТРУМОК-512" (надвисокий)	512	2^{512}	2^{256}

У табл. 9.4 представлено відомості щодо швидкості генерації ключових потоків [9].

Як видно із даних табл. 9.4, 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.

Таблиця 9.4

Швидкість генерації ключових потоків

Генератор ключових потоків	Intel Core i3-4005U 1,7 GHz Windows 10×64	Intel Core i5-7200U 2,5 GHz Windows 10×64	Intel Core i7-7700 3,6 GHz Windows 10×64
SNOW 2.0-128	4,2 Гбіт/с	8,0 Гбіт/с	10,6 Гбіт/с
SNOW 2.0-256	4,2 Гбіт/с	8,0 Гбіт/с	10,6 Гбіт/с
СТРУМОК-256	6,8 Гбіт/с	12,8 Гбіт/с	17,4 Гбіт/с
СТРУМОК-512	6,8 Гбіт/с	12,8 Гбіт/с	17,4 Гбіт/с

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. Також в алгоритмі “СТРУМОК” збільшені, в порівнянні з SNOW2.0, довжини секретного ключа та вектору ініціалізації, що дозволяє надійно застосовувати потоковий шифр навіть з із врахуванням квантових методів криптографічного аналізу.

Контрольні питання до розділу 9

1. Чим потоковий шифр відрізняється від блокового шифру?
2. Яким чином організовується шифрування потоку даних змінної довжини?
3. Які числа називають псевдовипадковими?
4. Назвіть існуючі ГПСЧ. Які властивості повинен мати ГПСЧ для використання з криптографічною метою?

5. Перерахуйте основні характеристики, переваги та недоліки кожного з розглянутих ГПСЧ.

6. Яким можуть використовуватися регістри зсуву зі зворотним зв'язком для отримання псевдовипадкових чисел? Поясніть їх принцип роботи.

7. У чому різниця між синхронними потоковими шифрами та потоковими шифрами з самосинхронізацією?

8. Дайте характеристику функції *Init* потокових шифрів.

9. Дайте характеристику функції *Next* потокових шифрів.

10. Дайте характеристику функції *Strm* потокових шифрів.

11. У чому різниця між генераторами випадкових і псевдовипадкових чисел?

12. Чи можна використовувати генератор справжніх випадкових чисел для отримання гами при потоковому шифруванні?

13. Визначте послідовність із перших десяти чисел і період лінійного конгруентного ГПСЧ для різних параметрів a , b і c (k_0 прийняти таким, що дорівнює 0):

а) $a = 7,\ b = 11,\ c = 23$;

б) $a = 8,\ b = 17,\ c = 31$.

14. Визначте послідовність із десяти чисел, що генерується методом Фібоначчі із запізненням, починаючи з k_a за таких вихідних даних:

а) $a = 4,\ b = 2,\ k_0 = 0{,}4;\ k_1 = 0{,}3;\ k_2 = 0{,}2$;

б) $a = 3,\ b = 5,\ k_0 = 0{,}8;\ k_1 = 0{,}3;\ k_2 = 0{,}7;\ k_3 = 0{,}5$.

15. Значення k_0, k_1, k_2, k_3, отримані за допомогою лінійного конгруентного генератора та дорівнюють: $k_0 = 1$; $k_1 = 8$; $k_2 = 10$; $k_3 = 9$. Визначити параметри a, b та c ГПВЧ.

16. Обчислити x_{13} за методом генерації ПВЧ BBS, якщо:

а) $p = 17,\ q = 23,\ x = 3$;

б) $p = 31,\ q = 23,\ x = 3$.

17. Обчислити псевдовипадкову двійкову послідовність довжиною 11 бітів за методом генерації ПВЧ *BBS*, якщо:

а) $p = 17,\ q = 23,\ x = 3$;

б) $p = 31,\ q = 23,\ x = 3$.

В якості випадкових бітів використовувати молодший біт двійкового запису числа, починаючи з x_0.

18. Дайте характеристику потоковому шифру A5/1.

19. Дайте характеристику потоковому шифру RC4.

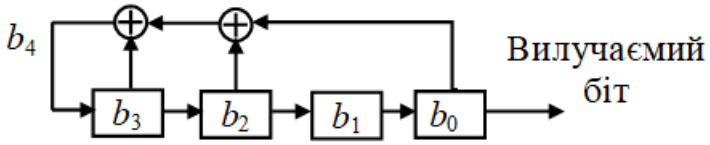
20. Дайте характеристику ДСТУ 8845:2019.

21. Порівняйте потокові шифри A5/1, RC4 та ДСТУ 8845:2019.

22. Для потокового алгоритму RC4 показати перші 20 елементів ключового потоку, якщо ключ сеансу складає 7 байтів зі значеннями 1, 2, 3, 4, 5, 6 та 7.

23. Покажіть ЛРЗЗЗ із характеристичним поліномом $x^5 + x^2 + 1$. Який максимальний період послідовності, що формується даним ЛРЗЗЗ?

24. Визначити характеристичний поліном представленого ЛРЗЗЗ:



Який максимальний період послідовності, що формується даним ЛРЗЗЗ?

25. Для потокового алгоритму A5/1 знайдіть максимальний період двійкової послідовності кожного лінійного регістру зсуву.

26. Для потокового алгоритму A5/1 знайдіть значення функцій:

а) $f(x, y, z) = f(1, 1, 1)$; в) $f(x, y, z) = f\ (0, 0, 0)$;

б) $f(x, y, z) = f\ (0, 1, 1)$; г) $f(x, y, z) = f\ (1, 0, 0)$.

У кожному випадку показати, скільки синхронізується лінійних регістрів зсуву.

РОЗДІЛ 10
АСИМЕТРИЧНІ КРИПТОГРАФІЧНІ СИСТЕМИ

10.1. Передісторія та основні ідеї

Розглянемо три задачі, вирішення яких допоможе краще зрозуміти ідеї і методи криптографії з відкритими ключами (асиметричні криптографічні системи). Усі ці задачі мають важливе практичне значення.

Перша задача – збереження паролів у комп'ютері [19, 24]. Відомо, що кожний користувач у мережі має свій секретний пароль. Входячи в мережу, користувач вказує своє ім'я (несекретне) і потім вводить пароль. Проблема полягає в наступному: якщо зберігати пароль на диску комп'ютера, то зловмисник може прочитати його, а потім використовувати для несанкціонованого доступу (особливо легко це зробити, якщо зловмисник працює системним адміністратором цієї мережі). Тому необхідно організувати зберігання паролів у комп'ютері так, щоб такий "злам" був неможливий.

Друга задача виникла з появою радіолокаторів і системи протиповітряної оборони. При перетині літаком кордону радіолокатор запитує пароль. Якщо пароль вірний, то літак "свій", інакше – "чужий". Тут виникає така проблема: оскільки пароль повинен передаватися по відкритому каналу (повітряному середовищі), то противник може прослуховувати всі переговори й дізнаватися правильний пароль. Потім "чужий" літак у разі запиту повторить перехоплений раніше "правильний" пароль в якості відповіді локатору й буде пропущений.

Третя задача схожа на попередню і виникає в комп'ютерних мережах із віддаленим доступом, наприклад, при взаємодії банку й клієнта. Зазвичай на початку сеансу банк запитує у клієнта ім'я, а потім секретний пароль, але зловмисник може дізнатися пароль, оскільки лінія зв'язку відкрита.

На сьогодні всі ці проблеми вирішуються з використанням криптографічних методів. Вирішення усіх цих задач ґрунтується на важливому понятті односторонньої функції (one-way function).

Нехай дана функція $y = f(x)$, що визначена на кінцевій множині X ($x \in X$), для якої існує обернена функція $x = f^{-1}(y)$. Функція називається *односторонньою*, якщо обчислення y за відомим x – проста задача, що вимагає небагато часу, а обчислення x за відомими y – задача складна, що вимагає залучення маси обчислювальних ресурсів, наприклад, 10^6-10^{10} років роботи потужного суперкомп'ютера. Дане визначення, безумовно, неформальне. Точне визначення односторонньої функції може бути знайдено в [1, 19, 20, 28, 36], але для наших цілей досить й вищенаведеного.

Як приклад односторонньої функції розглянемо наступну:

$$y = a^x \bmod p,$$

де p – деяке просте число (тобто таке, яке ділиться без залишку тільки на себе й на одиницю); x – ціле число з множини $\{1, 2, \ldots, p\text{-}1\}$.

Обернена функція позначається

$$x = \log_a y \bmod p$$

і називається дискретним логарифмом.

Для того, щоб забезпечити складність обчислення дискретного логарифма при використанні кращих сучасних комп'ютерів, у даний час використовуються числа розміром більше 512 бітів. На практиці часто застосовуються й інші односторонні функції, наприклад, так звані хеш-функції, які оперують з істотно коротшими числами, порядку 128 – 512 бітів. На сьогоднішній день ефективних алгоритмів обчислення дискретного логарифма невідомо. Один із методів обчислення, який називається "крок немовля, крок велетня", вимагає порядку $2 \cdot \sqrt{p}$ операцій множення.

Покажемо, що за великих p функція $y = a^x \bmod p$ дійсно одностороння, якщо для обчислення оберненої функції використовується метод “крок немовля, крок велетня”. Отримуємо наступний результат (табл. 10.1).

Таблиця 10.1

Кількість операцій множень для обчислення прямої й оберненої функції

Кількість десяткових знаків у записі p	Обчислення $y = a^x \bmod p$ ($2 \cdot \log_2 p$ множень)	Обчислення $x = \log_a y \bmod p$ ($2 \cdot \sqrt{p}$ множень)
12	$2 \cdot 40 = 80$	$2 \cdot 10^6$
60	$2 \cdot 200 = 400$	$2 \cdot 10^{30}$
90	$2 \cdot 300 = 600$	$2 \cdot 10^{45}$

З табл. 10.1 видно, що якщо використовувати модулі, що складаються з 50-100 десяткових цифр, то “пряма” функція обчислюється швидко, а обернена – практично не обчислюється. Розглянемо, наприклад, суперкомп’ютер, який множить два 90-значних числа за 10^{-14} сек. (для сучасних комп’ютерів це доки не доступно). Для обчислення y такому комп’ютері буде потрібно:

$$t_{\text{пряме}} = 600 \cdot 10^{-14} = 6 \cdot 10^{-12} \text{ сек.},$$

а для обчислення x

$$t_{\text{обернене}} = 10^{45} \cdot 10^{-14} = 10^{31} \text{ сек.},$$

тобто більше 10^{22} років.

З цього виходить, що обчислення обернених функцій практично неможливе за довжини чисел порядку 90 десяткових цифр, і використання паралельних обчислень і комп’ютерних мереж істотно не змінює ситуацію. У розглянутому прикладі припускалося, що обернена функція обчислюється за $2 \cdot \sqrt{p}$ операцій. У даний час відомі і більш швидкі методи обчислення дискретного логарифма, проте загальна картина та сама – кількість необхідних операцій більше $2 \cdot \log_2 p$. Таким чином, можна стверджувати, що

розглянута функція дійсно одностороння, але із застереженням. Ніким не доведено, що обернена функція не може бути обчислена так само швидко, як і пряма.

Використовуємо розглянуту односторонню функцію для вирішення всіх трьох задач, описаних на початку цього розділу, не забуваючи, однак, що так само може бути використана й будь-яка інша одностороння функція.

Почнемо зі зберігання паролів у пам'яті комп'ютера. Вирішення задачі основане на тому, що паролі взагалі не зберігаються! Точніше, реєструючись у мережі, користувач набирає своє ім'я і пароль; нехай, наприклад, його ім'я – "овоч", а пароль – "морква". Комп'ютер розглядає слово "морква" як двійковий запис числа x і обчислює y, де a і p – два несекретні числа, можливо навіть, усім відомі. Після цього в пам'яті комп'ютера утворюється пара (ім'я, y), де y обчислено за формулою: $y = a^x \bmod p$ при x = пароль. За всіх подальших входів цього користувача після введення пари ("овоч"-"морква"), комп'ютер обчислює нове значення $y_{\text{нове}}$ з x = "морква" і порівнює зі збереженим у пам'яті раніше обчисленим значенням y. Якщо $y_{\text{нове}}$ збігається із збереженим у пам'яті y, відповідним цьому імені, то це законний користувач. Інакше це зловмисник. Зловмисник міг би спробувати знайти x за y. Проте, вже при 90-значних числах для цього знадобиться більш, ніж 10^{22} років. Отже, представлена система зберігання пароля досить надійна й у даний час використовується в багатьох реальних операційних системах.

Розглянемо вирішення другої задачі (протиповітряна оборона і літак). Можна використовувати наступний метод: кожному "своєму" літаку присвоюється секретне ім'я, відоме системі протиповітряної оборони і льотчику, точніше, бортовому комп'ютеру. Нехай, наприклад, одному з літаків присвоєно секретне ім'я ВОРОН, і цей літак наближається до кордону 01 вересня 2023 року о 12 годині 45 хвилин. Тоді перед наближенням до кордону бортовий комп'ютер літака формує слово:

ВОРОН	2023	09	01	12	45
ім'я	рік	місяць	день	години	хвилини

Іншими словами, комп'ютер на літаку та станція протиповітряної оборони додають до секретного слова відомості про поточний час і, розглядаючи отримане слово як число x, обчислюють $y = a^x \bmod p$, де a і p не секретні. Потім літак повідомляє число y станції протиповітряної оборони. Станція порівнює обчислене нею число y з отриманим від літака. Якщо обчислене й отримане значення співпали, то літак визначається як "свій".

Противник не може зламати цю систему. Дійсно, з одного боку, він не знає секретного слова ВОРОН і не може знайти його за y, оскільки обчислення x за y займає, скажімо, 10^{22} років. З іншого боку, він не може просто скопіювати y і використовувати його як відповідь у майбутньому, оскільки час перетину кордону ніколи не повторюється й наступні значення y будуть відрізнятися від початкових. Розглянутий варіант вирішення задачі протиповітряної оборони вимагає точної синхронізації годин у літаку й локаторі. Ця проблема досить легко вирішується. Наприклад, служба навігації постійно передає мітки часу у відкритому виді (час не секретний), і всі літаки й локатори використовують ці мітки для синхронізації своїх годин.

Інший спосіб вирішення "задачі протиповітряної оборони" можливий, якщо використовувати додатковий відкритий канал передачі даних від локатора до літака. Як описано вище, кожний "свій" літак і локатор знають секретне слово (типу ВОРОН), яке не замінюється. Виявивши ціль, локатор посилає їй випадково згенероване число a ("виклик"). Літак обчислює $y = a^x \bmod p$, де x – секретне слово (ВОРОН), й повідомляє число y локатору. Локатор відтворює ті самі обчислення й порівнює обчислене y і прийняте. У цій схемі не потрібно синхронізувати годинник, але, як і раніше, противник не може повторити число y, оскільки локатор щоразу посилає різні виклики (a). Цікаво, що ця задача, напевно, була історично першою, вирішуючи яку, використовувалися односторонні функції.

Третя задача вирішується абсолютно аналогічно, і обидва розглянутих методи формування пароля застосовані й використовуються в реальних мережних протоколах.

10.2. Криптосистема Діффі-Хеллмана

Ця криптосистема була відкрита в середині 70-х років американськими вченими У. Діффі (Whitfield Diffie) і М. Хеллманом (Martin Hellman) та привела до справжньої революції в криптографії та її практичному застосуванні. Це перша система, яка дозволяла захищати інформацію без використання секретних ключів, переданих по захищених каналах. Для того, щоб продемонструвати одну зі схем застосування таких систем, розглянемо мережу зв'язку з N користувачами, де N – велике число. Нехай необхідно організувати секретний зв'язок для кожної пари з них. Якщо використовувати звичайну систему розподілу секретних ключів, то кожна пара абонентів має бути забезпечена своїм секретним ключем, тобто всього знадобиться

$$K = \frac{N(N-1)}{2} \approx \frac{N^2}{2}$$

ключів.

Якщо абонентів 100, то потрібно 5000 ключів, якщо ж абонентів 10^4, то ключів повинно бути $5 \cdot 10^7$. З цього випливає, що за великої кількості абонентів система постачання їх секретними ключами стає дуже громіздкою й дорогою.

Діффі і Хеллман вирішили цю проблему за рахунок відкритого поширення й обчислення ключів. Перейдемо до опису запропонованої ними системи.

Нехай необхідно побудувати систему зв'язку для абонентів *A, B, C, ...* . У кожного абонента є своя секретна й відкрита інформація. Для організації цієї системи вибирається велике просте число p і деяке число g ($1 < g < p$-1) таке, що всі числа з множини {1, 2, …, p-1} можуть бути представлені як різні степені $g \bmod p$ (тобто g є генератор групи Z_p^*). Числа p і g відомі всім абонентам.

Абоненти вибирають великі числа d_A, d_B, d_C, які зберігають у таємниці (зазвичай такий вибір рекомендується проводити випадково, використовуючи датчики випадкових чисел). Кожний абонент обчислює відповідне число e_A, e_B, e_C, яке відкрито передається іншим абонентам,

$$\begin{cases} e_A = \mathrm{g}^{d_A} \bmod p, \\ e_B = \mathrm{g}^{d_B} \bmod p, \\ e_C = \mathrm{g}^{d_C} \bmod p. \end{cases}$$

В результаті отримаємо табл. 10.2.

Таблиця 10.2

Ключі користувачів у системі Діффі-Хеллмана

Абонент	Секретний ключ	Відкритий ключ
A	d_A	e_A
B	d_B	e_B
C	d_C	e_C

Припустимо, абонент A вирішив організувати сеанс зв'язку з B, при цьому обом абонентам доступна відкрита інформація з табл. 10.2. Абонент A повідомляє B по відкритому каналу, що він хоче передати йому повідомлення. Потім абонент A обчислює величину:

$$K_{AB} = (e_B)^{d_A} \bmod p.$$

Ніхто інший окрім A цього зробити не зможе, тому що число d_A секретне.

У свою чергу, абонент B розраховує число:

$$K_{BA} = (e_A)^{d_B} \bmod p.$$

Схема криптографічної системи Діффі-Хеллмана представлена на рис. 10.1.

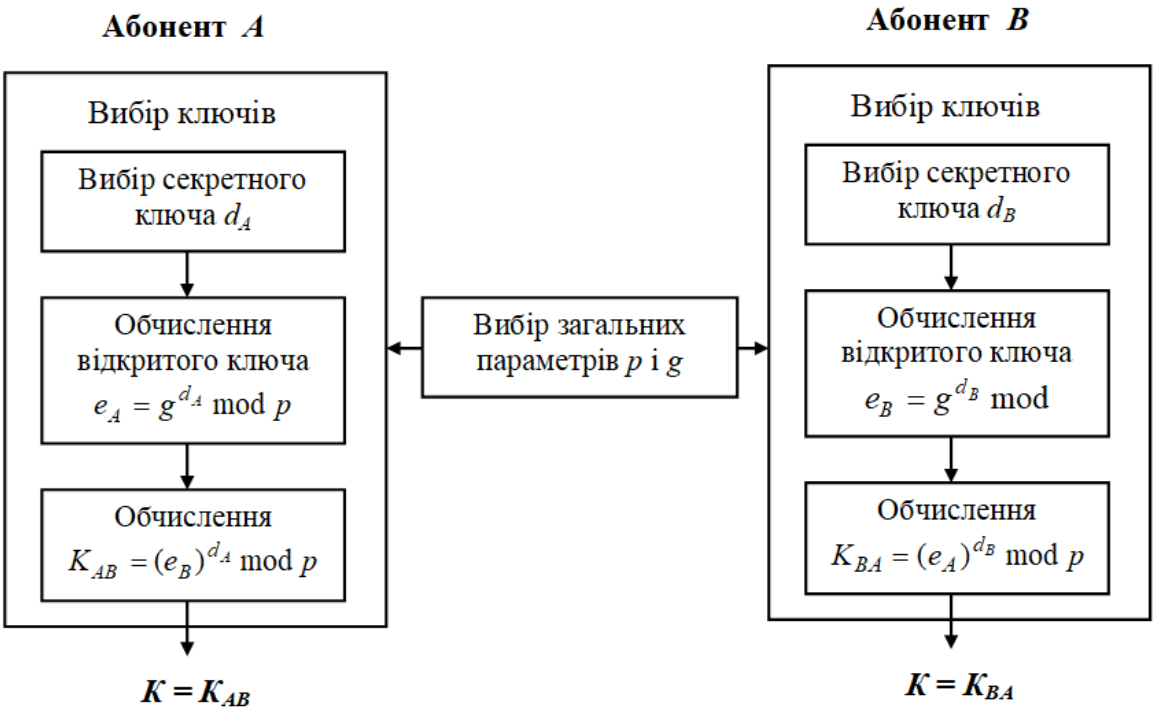


Рис. 10.1. Схема криптографічної системи Діффі-Хеллмана

Доведення, що $K_{AB} = K_{BA}$:

$$K_{AB} = (e_B)^{d_A} \bmod p = (\mathrm{g}^{d_B})^{d_A} \bmod p = \mathrm{g}^{d_A d_B} \bmod p = (e_A)^{d_B} \bmod p = K_{BA}.$$

Відмітимо головні властивості системи:

– абоненти A і B отримали те саме число $K = K_{AB} = K_{BA}$, яке не передавалося по відкритому каналу зв'язку;

– зловмисник не знає секретних чисел d_A і d_B, тому не може обчислити число K_{AB} або K_{BA} (взагалі, він міг би спробувати знайти секретне число d_A за e_A, проте за великих p це практично неможливо (потрібні мільйони років)).

Абоненти A і B можуть використовувати $K = K_{AB} = K_{BA}$ в якості секретного ключа для шифрування й розшифрування даних. Так само будь-яка пара абонентів може обчислити секретний ключ, відомий тільки їм.

Зупинимося тепер на задачі вибору числа g. За довільно заданого p це може виявитися важкою задачею, пов'язаною із розкладанням на прості множники числа p-1. Річ у тому, що для забезпечення високої стійкості розглянутої систем число p-1 повинне обов'язково містити великий простий

множник (інакше алгоритм Поліга-Хеллмана, описаний, наприклад, у [4, 20, 28], швидко обчислює дискретний логарифм). Тому часто рекомендують використовувати наступний підхід: просте число p вибирається таким, щоб виконувалася рівність

$$p = 2 \cdot q + 1,$$

де q – також просте число,
тоді в якості g можна взяти будь-яке число, для якого справедливі нерівності:

$$1 < g < p\text{-}1 \quad \text{і} \quad g^q \bmod p \neq 1.$$

Приклад 10.1. Нехай

$$p = 23 = 2 \cdot 11 + 1 \qquad (q = 11).$$

Обираємо параметр g. Спробуємо взяти $g = 3$. Перевірмо:

$$g^q \bmod p = 3^{11} \bmod 23 = 1,$$

а отже, таке g не підходить. Візьмемо $g = 5$. Перевірмо:

$$g^q \bmod p = 5^{11} \bmod 23 = 22 \neq 1.$$

Отже, можна вибрати параметри $p = 23$, $g = 5$.

Тепер кожний абонент вибирає секретне число та обчислює відповідне йому відкрите число. Нехай обрані $d_A = 7$, $d_B = 13$. Обчислюємо

$$e_A = \mathrm{g}^{d_A} \bmod p = 5^7 \bmod 23 = 17;$$
$$e_B = \mathrm{g}^{d_B} \bmod p = 5^{13} \bmod 23 = 21.$$

Нехай A та B вирішили сформувати спільний секретний ключ. Для цього A обчислює

$$K_{AB} = (e_B)^{d_A} \bmod p = 21^7 \bmod 23 = 10,$$

а B обчислює

$$K_{BA} = (e_A)^{d_B} \bmod p = 17^{13} \bmod 23 = 10.$$

Тепер вони мають спільний ключ $K = K_{AB} = K_{BA} = 10$, який не передавався по каналу зв'язку.

10.3. Криптографічна система RSA

Криптографічна система RSA названа на честь її розробників Рівеста, Шаміра й Адлемана. Ця криптографічна система досі є однією з найбільш широко використовуваних [5, 22, 34, 36, 37, 38]. Цікаве те, що вона базується на іншій односторонній функції, відмінній від дискретного логарифма. Крім того, в криптографічній системі RSA з'являється ще один винахід сучасної криптографії – одностороння функція з "лазівкою".

Криптографічна система RSA базується на тому, що задача розкладання чисел виду $n = p \cdot q$ (p і q – прості числа) на множники є надто складною, якщо відомо тільки n, а p і q – великі числа (це так звана задача факторизації).

Нехай у криптографічній системі RSA є абоненти A, B, C, ... Кожний абонент вибирає випадково два великі прості числа p і q. Потім він обчислює число

$$n = p \cdot q.$$

Число n є відкритою інформацією, доступною іншим абонентам.

Після цього абоненти обчислюють число $\varphi(n) = (p-1) \cdot (q-1)$ та обирають деякі числа $1 < e < \varphi(n)$, взаємно прості з $\varphi(n)$, і за узагальненим алгоритмом Евкліда знаходить числа d, такі, що

$$e \cdot d \bmod \varphi(n) = 1.$$

Усю інформацію, пов’язану з абонентами і є їх відкритими й секретними ключами, представлено в табл. 10.3.

Таблиця 10.3

Ключі користувачів у системі RSA

Абонент	Секретний ключ	Відкритий ключ
A	d_A	e_A, n_A
B	d_B	e_B, n_B
C	d_C	e_C, n_C

Опишемо протокол RSA. Нехай абонент A хоче передати повідомлення M абоненту B, причому повідомлення M розглядається як число, яке задовольняє нерівності $M < n_B$ (далі індекс B вказує на те, що відповідні параметри належать абонентові B).

Крок 1. Абонент A зашифровує повідомлення за формулою

$$C = (M^{e_B}) \bmod n_B,$$

використовуючи відкриті параметри абонента B, та пересилає C по відкритому каналу.

Крок 2. Абонент B, отримавши зашифроване повідомлення, обчислює

$$M' = (C^{d_B}) \bmod n_B.$$

Схема криптографічної системи RSA, яка показує процеси генерації ключів обома абонентами, шифрування даних при їх передачі від абонента A до абоненту B та розшифрування отриманого шифротексту абонентом B, представлена на рис. 10.2.

Схема криптографічної системи RSA, яка показує процес шифрування даних при їх передачі від абонента B до абонента A, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент B.

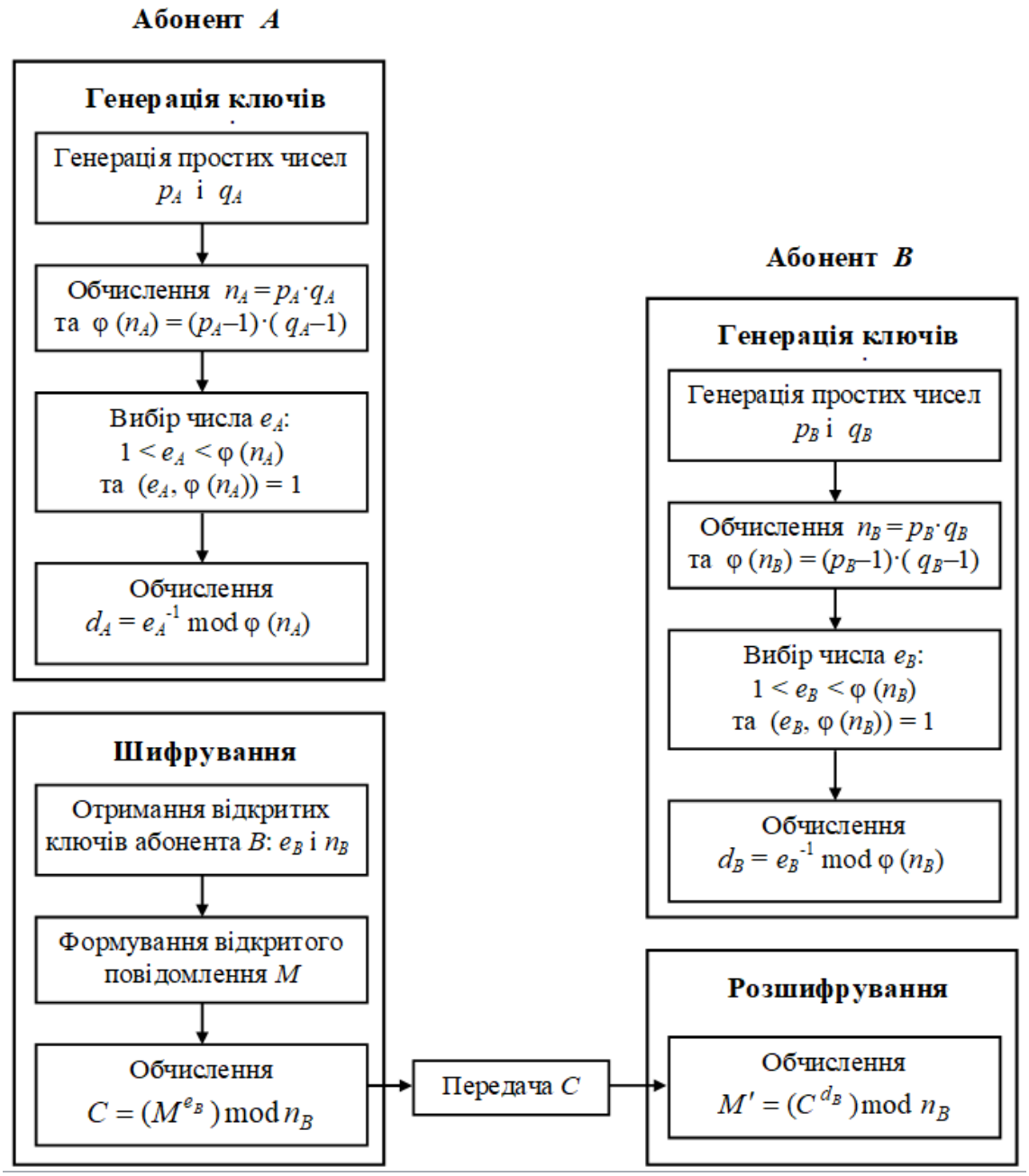


Рис. 10.2. Схема криптографічної системи RSA

Для описаного протоколу $M' = M$, тобто абонент B отримує від абонента A повідомлення.

Доведення, що $M' = M$.

$$M' = (C^{d_B}) \bmod n_B = (M^{e_B \cdot d_B}) \bmod n_B.$$

Рівність $e \cdot d \bmod \varphi(n) = 1$ означає, що для деякого k

$$e_B \cdot d_B = k \cdot \varphi(n_B) + 1.$$

В свою чергу

$$\varphi(n_B) = (p_B - 1)\cdot(q_B - 1)$$

де $\varphi(n_B)$ – функція Ейлера.

Використовуючи теорему Ферма маємо

$$M' = (M^{e_B \cdot d_B}) \bmod n_B = (M^{k\cdot\varphi(n_B)+1}) \bmod n_B = M.$$

Властивості протоколу RSA.

1) Протокол забезпечує шифрування й розшифрування інформації коректно.

2) Зловмисник, що перехоплює всі зашифровані повідомлення і знає всю відкриту інформацію, не зможе знайти початкове повідомлення M за великих p і q.

Доведення. Перша властивість протоколу випливає з твердження, що $M' = M$. Для доведення другої властивості зауважимо, що зловмисник знає тільки відкриті параметри n і e. Для того, щоб знайти d, він повинен знати значення $\varphi(n) = (p-1)\cdot(q-1)$, а для цього, в свою чергу, йому потрібно знати p і q. Взагалі, він може знайти p і q, розклавши n на множники, однак це складна задача (задача факторизації).

Одностороння функція $y = x^d \bmod n$, що використовується в системі RSA, має так звану "лазівку", що дозволяє легко обчислити обернену функцію $x = \sqrt[d]{y} \bmod n$, якщо відомо розкладання n на прості множники. Дійсно, легко обчислити $\varphi(n) = (p-1)\cdot(q-1)$, а потім $d = e^{-1} \bmod \varphi(n)$. Якщо p і q невідомі, то обчислення значення оберненої функції майже неможливе, а знайти p і q за n надто складно, тобто знання p і q – це "лазівка" або "потайний хід". Такі односторонні функції з "лазівкою" знаходять застосування і в інших розділах криптографії.

Приклад 10.2. Припустимо, абонент A хоче передати абоненту B повідомлення $M = 15$. Нехай абонент B вибрав наступні параметри:

$$p_B = 3, \quad q_B = 11, \quad n_B = 33, \quad e_B = 3,$$

e_B взаємно просте з $\varphi(n_B) = \varphi(33) = 20$.

За допомогою узагальненого алгоритму Евкліда знаходимо секретний ключ абонента B – d_B (він буде дорівнювати 7).

Перевіримо: $e_B \cdot d_B \bmod \varphi(n_B) = 3 \cdot 7 \bmod 20 = 1$.

Зашифруємо повідомлення M:

$$C = M^{e_B} \bmod n_B = 15^3 \bmod 33 = 15^2 \cdot 15 \bmod 33 = 27 \cdot 15 \bmod 33 = 9.$$

Число C абонент A передає абоненту B по відкритому каналу зв'язку. Тільки абонент B знає секретний ключ $d_B = 7$, тому він розшифровує прийняте повідомлення C:

$$M' = C^{d_B} \bmod n_B = 9^7 \bmod 33 = (9^2)^2 \cdot 9^2 \cdot 9 \bmod 33 = 15^2 \cdot 15 \cdot 9 \bmod 33 = 15.$$

Таким чином, абонент B розшифрував повідомлення абонента A.

Аналіз безпеки криптосистеми RSA

Складність знаходження секретного ключа системи RSA визначається складністю розкладання числа n на прості множники. У зв'язку з цим необхідно вибирати числа p і q таким чином, щоб задача розкладання числа n була надто складною в обчислювальному плані. Для цього рекомендуються наступні вимоги:

– числа p і q повинні бути досить великими (порядку 512 бітів або 154 десяткових цифр), не дуже сильно відрізнятися одне від одного та в той самий час бути не дуже близькими одне до одного;

– числа p і q повинні бути таким, щоб найбільший спільний дільник чисел $p-1$ і $q-1$ був невеликим; бажано, щоб він дорівнював двом;

– числа p і q повинні бути сильно простими числами (сильним простим числом називається таке просте число m, що $m+1$ має великий простий

дільник, $m-1$ має великий простий дільник s, такий, що число $s-1$ також має досить великий простий дільник).

Відмітимо, що для схеми RSA важливо, щоб кожен абонент вибирав власну пару простих чисел p і q, тобто усі модулі n_A, n_b, n_C, … мають бути різні (інакше один абонент міг би читати зашифровані повідомлення, призначені для іншого абонента).

Зазначимо, що асиметричні криптосистеми повільніші, ніж симетричні. Для підвищення швидкості шифрування RSA на практиці використовують малу експоненту шифрування. Якщо вибрати e невеликим або таким, що у його двійковому представленні було небагато одиниць, то процедуру шифрування можна значно прискорити. Наприклад, обравши $e = 3$ (при цьому ні $p-1$, ні $q-1$ не повинні ділитися на 3), можна реалізувати шифрування за допомогою одного зведення до квадрату по модулю n і одного множення. Однак є деякі потенційні атаки на вибір невеликої експоненти e (атака теореми Куперсміта, атака зв'язних між собою повідомлень, атака короткого списку, широкомовної передачі [36]). Ці атаки взагалі не закінчуються розкриттям системи, але необхідно запобігти їх виникненню. Для того, щоб зірвати ці атаки, рекомендується використовувати $e = 2^{16} + 1 = 65537$ – число у двійковому представленні має тільки дві одиниці, тоді можна реалізувати шифрування за допомогою 16-ті зведень до квадрату за модулем n і одного множення. Якщо експонента e вибирається випадково, то реалізація шифрування за алгоритмом RSA вимагає s зведень до квадрату по модулю n і в середньому $s/2$ множень за тим самим модулем, де s – довжина двійкового запису числа e.

Вибір малої експоненти розшифрування d також небажаний у зв'язку з можливістю визначення d простим перебором. Відомо також, що якщо $d = \sqrt[4]{n}$, то експоненту d легко знайти, використовуючи безперервні дробі (атака Вінера).

10.4. Криптографічна система Ель-Гамаля

Нехай є абоненти *A*, *B*, *C*, ..., які хочуть передавати один одному зашифровані повідомлення, не маючи ніяких захищених каналів зв'язку. У цьому підрозділі розглянемо криптографічну систему, запропоновану Ель-Гамалем. Фактично в ній використовується схема Діффі-Хеллмана, щоб сформувати загальний секретний ключ для двох абонентів, що передають один одному повідомлення, і потім повідомлення шифрується шляхом множення його на цей ключ. Для кожного наступного повідомлення секретний ключ обчислюється заново. Перейдемо до точного опису криптографічної системи.

Для всієї групи абонентів вибираються деяке велике просте число p і число g, такі, що різні степені g несуть різні числа за модулем p (тобто g є генератор групи Z_p^*). Числа p і g передаються абонентам у відкритому виді (вони можуть використовуватися всіма абонентами мережі).

Потім кожний абонент групи обирає своє секретне число d_i, яке задовольняє вимозі

$$1 < d_i < p-1$$

та обчислює відповідне йому відкрите число e_i

$$e_i = \mathrm{g}^{d_i} \bmod p.$$

В результаті отримаємо табл. 10.4.

Таблиця 10.4

Ключі користувачів у системі Ель-Гамаля

Абонент	Секретний ключ	Відкритий ключ
A	d_A	e_A
B	d_B	e_B
C	d_C	e_C

Покажемо тепер, як абонент *A* передає повідомлення *M* абоненту *B*. Будемо припускати, як і при описі криптографічної системи RSA, що повідомлення представлено у вигляді числа M < *p*.

Крок 1. Абонент *A* формує випадкове число *r*, 1 < *r* < *p*–1 і обчислює числа

$$C_1 = \mathrm{g}^r \bmod p;$$
$$C_2 = (M \cdot e_B^r) \bmod p$$

та передає пару чисел (C_1, C_2) абоненту *B*.

Крок 2. Абонент *B,* отримавши (C_1, C_2), обчислює

$$M' = \left(C_2 \cdot C_1^{-d_B}\right) \bmod p.$$

Схема криптографічної системи Ель-Гамаля, яка показує процеси генерації ключів обома абонентами, шифрування даних під час їх передачі від абонента *A* до абоненту *B* та розшифрування отриманого шифротексту абонентом *B*, представлена на рис. 10.3.

Схема криптографічної системи Ель-Гамаля, яка показує процес шифрування даних під час їх передачі від абонента *B* до абонента *A*, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент *B*.

Властивості криптографічної системи Ель-Гамаля.

1) Абонент *B* отримав повідомлення, тобто $M' = M$.

2) Зловмисник, знаючи p, g, e_B, C_1 і C_2 не може обчислити *M*.

Доведення.

$$M' = \left(C_2 \cdot C_1^{-d_B}\right) \bmod p = \left(M \cdot e_B^r \cdot C_1^{-d_B}\right) \bmod p = (M \cdot (\mathrm{g}^{d_B})^r \cdot (g^r)^{-d_B}) \bmod p =$$
$$= (M \cdot \mathrm{g}^{d_B \cdot r - d_B \cdot r}) \bmod p = M.$$

Для доведення другої частини зауважимо, що зловмисник не може обчислити *r* з виразу для C_1, оскільки це задача дискретного логарифмування.

Отже, він не може обчислити M з виразу для C_2, оскільки M було помножене на невідоме йому число. Зловмисник також не може відтворити дії законного отримувача повідомлення (абонента B), оскільки йому не відоме секретне число d_B (обчислення d_B з виразу для e_B – також є задачею дискретного логарифмування).

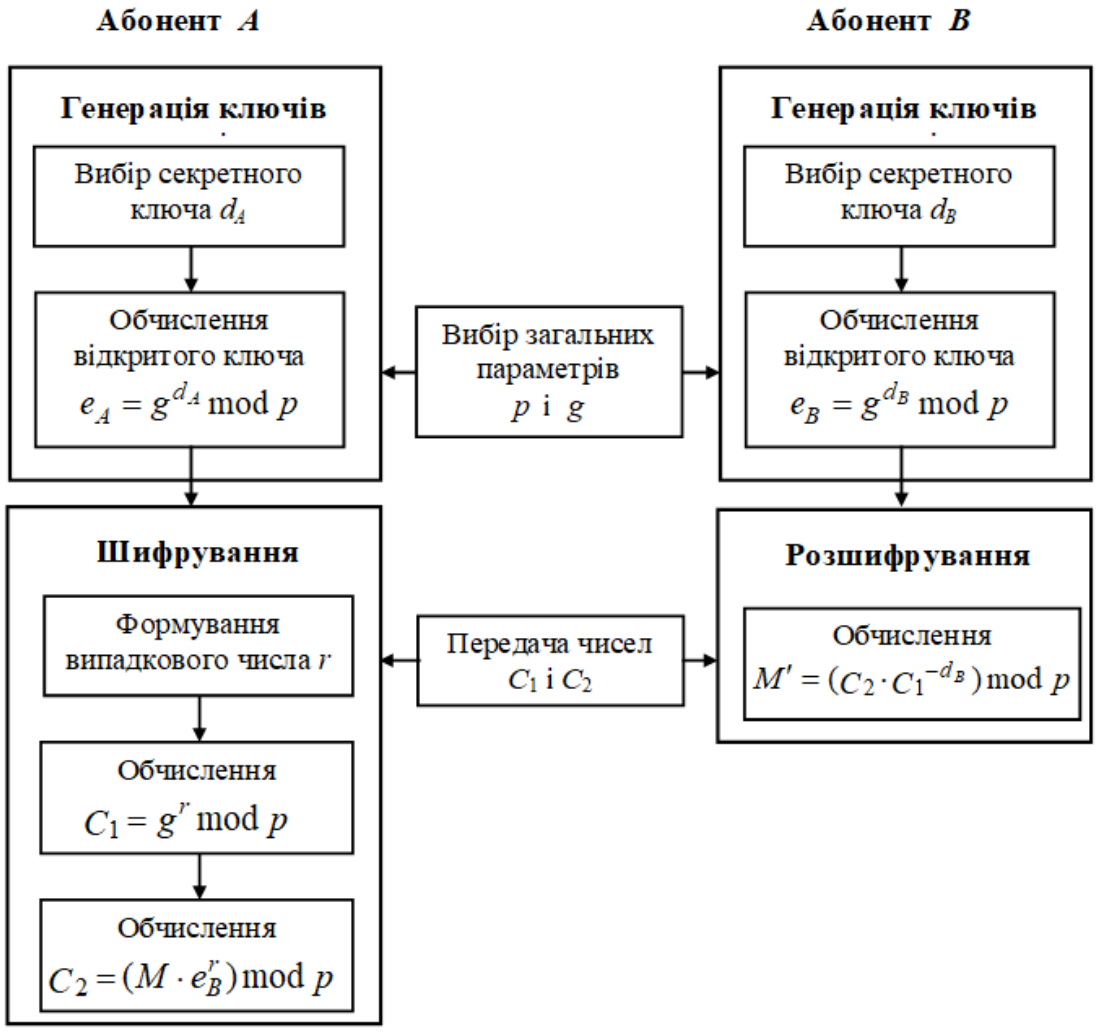


Рис. 10.3. Схема криптографічної системи Ель-Гамаля

Приклад 10.3. Передамо повідомлення $M = 15$ від абонента A до абонента B. Виберемо параметри аналогічно тому, як це було зроблено в прикладі 10.1. Візьмемо $p = 23$, $g = 5$. Нехай абонент B вибрав для себе секретне число $d_B = 13$ та обчислив e_B:

$$e_B = \mathrm{g}^{d_B} \bmod p = 5^{13} \bmod 23 = 21.$$

Абонент A вибирає випадкове число r, наприклад $r = 7$, та обчислює C_1 і C_2:

$$C_1 = \mathrm{g}^r \bmod p = 5^7 \bmod 23 = 17,$$
$$C_2 = (M{\cdot}e_B^r) \bmod p = (15{\cdot}21^7) \bmod 23 = (15{\cdot}10) \bmod 23 = 12.$$

Тепер абонент A посилає абоненту B зашифроване повідомлення у вигляді пари чисел $(C_1, C_2) = (17, 12)$.

Абонент B обчислює M'

$$M' = \left(C_2 \cdot C_1^{-d_B}\right) \bmod p = (12 \cdot 17^{-13}) \bmod 23 = (12 \cdot 19^{13}) \bmod 23 = 15.$$

Отже, абонент B розшифрував передане повідомлення абонента A.

За аналогічною схемою можуть передавати повідомлення всі абоненти в мережі. Зауважимо, що будь-який абонент, що знає відкритий ключ абонента B, може посилати йому повідомлення, зашифровані за допомогою відкритого ключа e_B. Але тільки абонент B, і ніхто інший, може розшифрувати ці повідомлення, використовуючи відомий тільки йому секретний ключ d_B. Зазначимо також, що обсяг зашифрованого повідомлення у два рази перевищує обсяг повідомлення, яке передається, але потрібно тільки одна передача даних (за умови, що таблиця з відкритими ключами заздалегідь відома всім абонентам).

Аналіз безпеки криптосистеми Ель-Гамаля

Як вже відмічалось, стійкість криптосистеми Ель-Гамаля визначається складністю рішення задачі дискретного логарифмування у групі Z_p^*. На теперішній час ця задача практично нереалізуєма для значень p, які мають не менше 150 десяткових цифр. Рекомендується також, щоб число $p-1$ мало великий простий дільник.

Введення в правило шифрування випадкового числа r (рандомізатор) робить криптосистему Ель-Гамаля схемою ймовірнісного шифрування. Для неї відкритий текст і ключ не визначають шифротекст однозначно.

Слід відмітити, що в наведеній системі необхідно використовувати різні значення r для шифрування різних відкритих текстів M та M'. В противному випадку відповідні шифротексти (C_1, C_2) і (C_1', C_2') виявляються пов'язаними співвідношенням $C_2 \cdot (C_2')^{-1} = M \cdot (M')^{-1}$ і тоді M' може бути легко обчислене за відомим M.

Система Ель-Гамаля може бути узагальнена для використання в будь-якій кінцевій циклічній групі G. Криптографічна стійкість такої загальної схеми визначається складністю задачі логарифмування в групі G. В якості G найчастіше всього вибираються наступні три групи:

– мультиплікативна група Z_p^* цілих чисел за модулем простого числа p;

– мультиплікативна група $GF(2^n)^*$ кінцевого поля $GF(2^n)$ характеристики два;

– група точок еліптичної кривої над кінцевим полем.

Ймовірнісний характер шифрування можна віднести до переваг криптосистеми Ель-Гамаля, так як схеми ймовірнісного шифрування мають, як правило, більшу стійкість у порівнянні із схемами з детермінованим процесом шифрування. Недоліком системи є подвоєння довжини відкритого тексту при шифруванні.

10.5. Криптографічна система Рабіна

Безпека криптографічної системи Рабіна заснована на складності пошуку квадратних коренів за модулем складеного числа. З позиції теорії чисел ця задача аналогічна задачі факторизації модуля. Розглянемо одну з реалізацій системи. Уявімо, що користувачі A і B бажають обмінюватися зашифрованими повідомленнями між собою. Тоді за схемою Рабіна:

1. Користувачі A і B генерують для себе відкриті і закриті ключі, для цього:

– кожний із користувачів вибирає по два великих простих числа p_A (p_B) та q_A (q_B) (далі алгоритм розшифрування буде особливо простим, якщо ці числа порівняні із числом 3 за модулем 4, тобто $p_A \bmod 4 = 3$, $q_A \bmod 4 = 3$, $p_B \bmod 4 = 3$, $q_B \bmod 4 = 3$). Числа p_A (p_B) та q_A (q_B) будуть закритими ключами системи (користувачів A і B);

– кожний із користувачів визначає свої відкриті ключі:

$$n_A = p_A \cdot q_A \quad \text{та} \quad n_B = p_B \cdot q_B;$$

– користувачі A і B обмінюються відкритими ключами.

2. Перед шифруванням повідомлень абоненти розбивають їх на блоки m_i, довжина яких менше відкритого ключа n_A (якщо відбувається передача повідомлення від користувача B до користувача A) або n_B (якщо відбувається передача повідомлення від користувача A до користувача B). Вважатимемо, що $M < n_A$ або $M < n_B$.

3. Для шифрування повідомлень абоненти A або B перетворюють дані згідно з рівнянням шифрування

$$C = E_n(M) = M^2 \bmod n$$

і після цього пересилають C іншому абонентові B або A.

4. Із виразу шифрування виходить, що M – квадратний корінь із числа C за модулем n. Якщо числа C і n взаємно прості, то кожний квадратичний залишок має в мультиплікативній групі залишків Z_n^* рівно чотири різні корені [4, 36]. Оскільки отримувач знає множники p і q числа n, то він вирішує два порівняння. Передусім, він обчислює

$$r_1 = C^{(p+1)/4} \bmod p; \quad r_2 = -C^{(p+1)/4} \bmod p;$$
$$r_3 = C^{(q+1)/4} \bmod q; \quad r_4 = -C^{(q+1)/4} \bmod q;$$
$$a = q \cdot (q^{-1} \bmod p); \quad b = p \cdot (p^{-1} \bmod q).$$

Чотири можливих корені із числа C за модулем n – це

$$M_1 = (a \cdot r_1 + b \cdot r_3) \bmod n; \quad M_2 = (a \cdot r_2 + b \cdot r_4) \bmod n;$$
$$M_3 = (a \cdot r_1 + b \cdot r_4) \bmod n; \quad M_4 = (a \cdot r_2 + b \cdot r_3) \bmod n.$$

Схема криптографічної системи Рабіна, яка показує процеси генерації ключів абонентом A, шифрування даних під час їх передачі від абонента B до абоненту A та розшифрування отриманого шифротексту абонентом A, представлена на рис. 10.4.

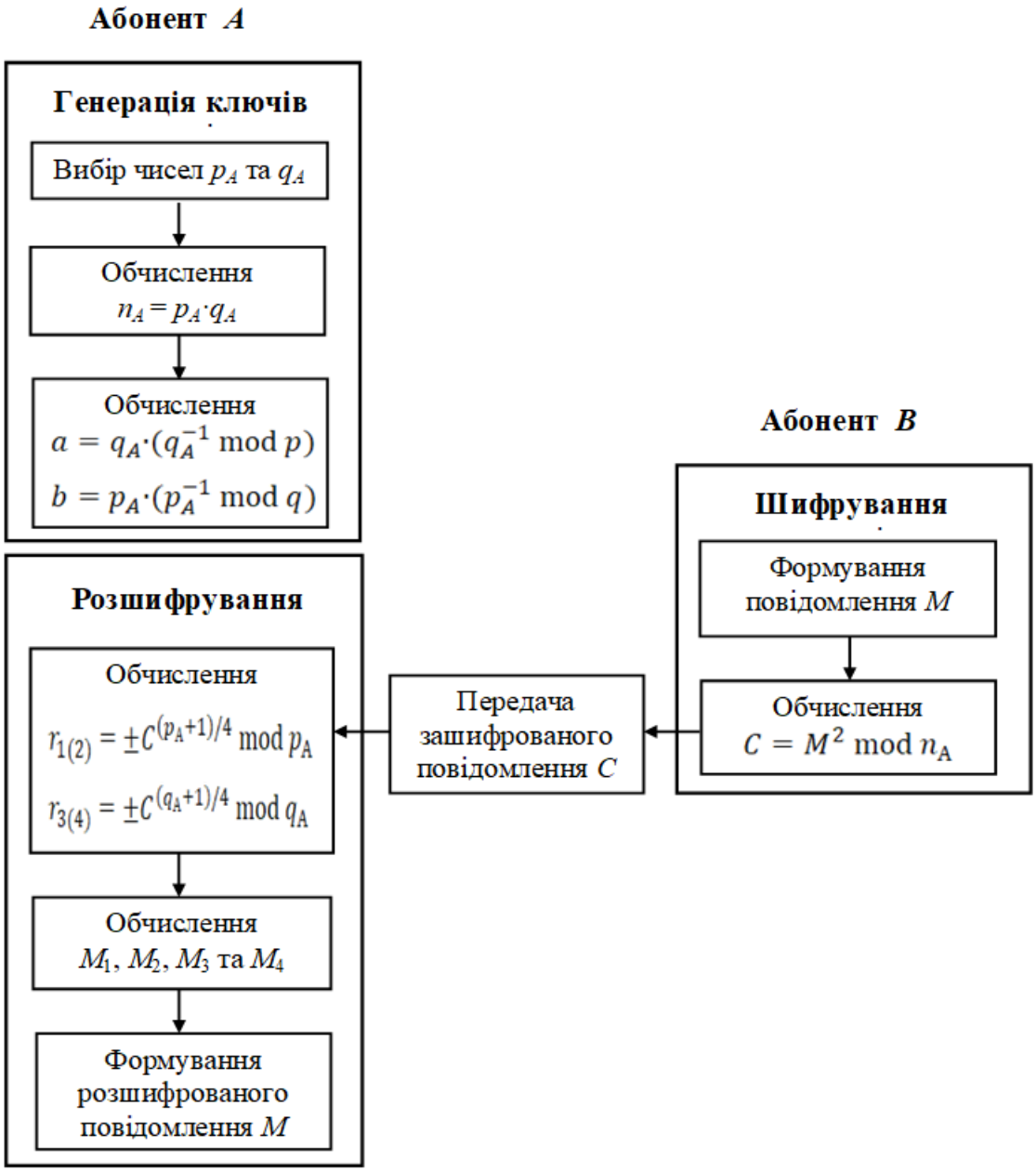


Рис. 10.4. Схема криптографічної системи Рабіна

Схема криптографічної системи Рабіна, яка показує процеси шифрування даних під час їх передачі від абонента *A* до абонента *B*, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент *A*.

Отже, в криптографічній системі Рабіна шифрувальне відображення неін'єктивне (невзаємно однозначне). Після обчислення всіх чотирьох коренів із них вибирають той, який є числовим еквівалентом змістовного повідомлення, яке було передане. Якщо повідомлення написане звичайною мовою, то вибрати правильне *M* неважко. Але якщо шифрувалася сукупність випадкових бітів, способу визначення правильного *M* не існує.

Приклад 10.4. Згенерувати ключі і зашифрувати за допомогою криптографічної системи Рабіна повідомлення "*перемога*" при передачі його від користувача *B* до користувача *A*.

Рішення. Нехай закритий ключ користувача *A* утворює пара простих чисел $p_A = 79$ та $q_A = 83$, тоді відкритий ключ – число $n_A = p_A \cdot q_A = 6557$. Кожну букву відкритого повідомлення замінимо її номером в українському алфавіті (див. рис. 2.2)

"*перемога*" $\Rightarrow$ 19 06 20 06 16 18 03 00.

Наведена числова послідовність – це відкрите повідомлення, яке записане в цифровій формі. Розіб'ємо послідовність на чотири блоки m_i, кожний з яких – деяке натуральне число, яке повинне бути меншим, ніж число $n_A = p_A \cdot q_A = 6557$:

$M \Rightarrow$ 1906 – 2006 – 1618 – 0300.

Зашифруємо відкрите повідомлення *M* за формулою $C = M^2 \bmod n$:

$$C_1 = E_{n_A}(m_1) \bmod n_A = 1906^2 \bmod 6557 = 0258;$$

$$C_2 = E_{n_A}(m_2) \bmod n_A = 2006^2 \bmod 6557 = 4595;$$

$$C_3 = E_{n_A}(m_3) \bmod n_A = 1618^2 \bmod 6557 = 1681;$$

$$C_4 = E_{n_A}(m_4) \bmod n_A = 300^2 \bmod 6557 = 4759.$$

В результаті отримаємо зашифроване повідомлення

$$C \Rightarrow 0258 - 4595 - 1681 - 4759.$$

Приклад 10.5. Користувач *A* згенерував власну пару ключів для криптографічної системи Рабіна: закритий ключ – прості числа $p_A = 59$ та $q_A = 67$, які він зберігає в таємниці, і відкритий ключ – загальнодоступне число $n_A = p_A \cdot q_A = 3953$. Розшифрувати отримане зашифроване повідомлення $C \Rightarrow 2409 - 0992$ від користувача *B*.

Рішення. Розшифрування – це добуття квадратного кореня із чисел $C_1 = 2409$ та $C_2 = 0992$ за модулем $n_A = 3953$ і зводиться до добування коренів за простими модулями $p_A = 59$ та $q_A = 67$.

Обчислюємо:

$$r_1 = {C_1}^{(p_A+1)/4} \bmod p_A = 2409^{(59+1)/4} \bmod 59 = 7;$$
$$r_2 = -{C_1}^{(p_A+1)/4} \bmod p_A = -2409^{(59+1)/4} \bmod 59 = -7 \bmod 59 = 52;$$
$$r_3 = {C_1}^{(q_A+1)/4} \bmod q_A = 2409^{(67+1)/4} \bmod 67 = 59;$$
$$r_4 = -{C_1}^{(q_A+1)/4} \bmod q_A = -2409^{(67+1)/4} \bmod 67 = -59 \bmod 67 = 8.$$

Отриманні розв’язки скомбінуємо між собою в системи та за китайською теоремою про залишки визначимо значення квадратних коренів. Найбільший спільний дільник чисел $p_A = 59$ і $q_A = 67$ дорівнює одиниці та за розширеним алгоритмом Евкліда маємо:

$$1 = (-22) \cdot 67 + 25 \cdot 59,$$

тобто $(q_A \cdot {q_A}^{-1}) \bmod p_A = 1$, звідки ${q_A}^{-1} \bmod p_A = -22 \bmod 59 = 37$; $(p_A \cdot {p_A}^{-1}) \bmod q_A = 1$, звідки ${p_A}^{-1} \bmod q_A = 25$.

Відповідно до значень $q_A, {q_A}^{-1}, p_A$ та ${p_A}^{-1}$:

$$a = q_A \cdot ({q_A}^{-1} \bmod p_A) = 67 \cdot 37 = 2479;$$
$$b = p_A \cdot ({p_A}^{-1} \bmod q_A) = 59 \cdot 25 = 1475.$$

Знайдемо перший корінь:

$$M_1 = (a \cdot r_1 + b \cdot r_3) \bmod n_A = (2479 \cdot 7 + 1475 \cdot 59) \bmod 3953 = 1600.$$

Аналогічно знаходимо другий корінь:

$$M_2 = (a \cdot r_2 + b \cdot r_4) \bmod n_A = (2479 \cdot 52 + 1475 \cdot 8) \bmod 3953 = 2353.$$

Можна впевнитися, що корені M_1 та M_2 протилежні, тобто

$$(M_1 + M_2) \bmod n_A = (1600 + 2353) \bmod 3953 = 0.$$

Аналогічно знаходимо третій та четвертий корені:

$$M_3 = (a \cdot r_1 + b \cdot r_4) \bmod n_A = (2479 \cdot 7 + 1475 \cdot 8) \bmod 3953 = 1482;$$
$$M_4 = (a \cdot r_2 + b \cdot r_3) \bmod n_A = (2479 \cdot 52 + 1475 \cdot 59) \bmod 3953 = 2471.$$

Корені M_3 та M_4 також протилежні $(M_3 + M_4) \bmod n_A = 0$.

Отже із числа $C_1 = 2409$ добуто чотири кореня за модулем 3953: 1600, 2353, 1482 та 2471. Аналогічно добудемо чотири кореня з другого числа $C_2 = 0992$ отриманого зашифрованого повідомлення: 1195, 2758, 2021, 1932. Лише корені 1600 та 2021 – числовий еквівалент повідомлення українського алфавіту, потужність якого 33 літери (див. рис. 2.2).

Оскільки числу 16 українського алфавіту відповідає буква “*м*”, 00 – “*а*”, 20 – “*р*”, 21 – “*с*”, 19 – “*п*”, 32 – “*я*”, то перші дві букви розшифрованого повідомлення будуть – “*ма*”, а другі дві букви – або “*рс*”, або “*пя*”. Отже, розшифроване повідомлення відповідає або “*марс*”, або “*мапя*”. Смислове навантаження в українській мові несе тільки слово “*марс*”.

Аналіз безпеки криптосистеми Рабіна

Криптосистеми Рабіна безпечна, поки p та q – великі числа. Складність криптографічної системи Рабіна така сама, як і алгоритму розкладання на множники великих чисел. Отже, криптографічної системи Рабіна так само безпечна, як і RSA.

Контрольні питання до розділу 10

1. Порівняйте симетричні та асиметричні криптосистеми.

2. Дайте визначення односторонньої функції. Наведіть приклади.

3. Назвіть призначення алгоритму Діффі-Хеллмана.

4. Опишіть послідовність дій при використанні алгоритму Діффі–Хеллмана.

5. Складність якої математичної задачі визначає стійкість системи RSA?

6. З якою метою може застосовуватися алгоритм RSA?

7. Опишіть процес генерації ключів у алгоритмі RSA.

8. Опишіть процес шифрування та розшифрування з використанням алгоритму RSA.

9. Назвіть рекомендації до вибору параметрів системи RSA.

10. Складність якої математичної задачі визначає стійкість системи Ель-Гамаля?

11. Опишіть процес генерації ключів у алгоритмі Ель-Гамаля.

12. Опишіть процес шифрування та розшифрування з використанням алгоритму Ель-Гамаля.

13. До якого типу належить схема шифрування, яка використовується в системі Ель-Гамаля? В чому її переваги?

14. Складність якої математичної задачі визначає стійкість системи Рабина?

15. Опишіть процес шифрування та розшифрування з використанням алгоритму Рабина.

16. Обчислити відкриті ключі e_A, e_B і загальний ключ K_{AB} для системи Діффі-Хеллмана з параметрами:

а) $p = 23,\ g = 5,\ d_A = 5,\ d_B = 7$;

б) $p = 19,\ g = 2,\ d_A = 5,\ d_B = 7$;

в) $p = 23,\ g = 7,\ d_A = 3,\ d_B = 4$;

г) $p = 17,\ g = 3,\ d_A = 10,\ d_B = 5$;

д) $p = 19,\ g = 10,\ d_A = 4,\ d_B = 8$.

17. У системі RSA із заданими параметрами p_A, q_A та e_A знайти неправильно обрані параметри й описати процес передачі повідомлення M користувачу A:

а) $p_A = 5,\ q_A = 11,\ e_A = 3,\ M = 12$;

б) $p_A = 5,\ q_A = 13,\ e_A = 5,\ M = 20$;

в) $p_A = 7,\ q_A = 11,\ e_A = 7,\ M = 17$;

г) $p_A = 7,\ q_A = 13,\ e_A = 5,\ M = 30$;

д) $p_A = 3,\ q_A = 11,\ e_A = 3,\ M = 15$.

18. Користувачу A системи RSA з параметрами $n_A = 187$ та $e_A = 3$ передано зашифроване повідомлення $C = 100$. Зловмисник перехопив це повідомлення. Пояснити процес зламу цієї системи RSA і визначити відкрите повідомлення, яке передавалося користувачу A.

19. Для криптографічної системи Ель-Гамаля із заданими параметрами p, g, d_B і k знайти неправильно обрані параметри й описати процес передачі повідомлення M користувачу B:

а) $p = 19,\ g = 2,\ d_B = 5,\ k = 7,\ M = 5$;

б) $p = 23,\ g = 5,\ d_B = 8,\ k = 10,\ M = 10$;

в) $p = 19,\ g = 2,\ d_B = 11,\ k = 4,\ M = 10$;

г) $p = 23,\ g = 7,\ d_B = 3,\ k = 15,\ M = 5$;

д) $p = 17,\ g = 3,\ d_B = 10,\ k = 5,\ M = 10$.

20. Дана криптографічна система Рабіна з параметрами $p_A = 53$, $q_A = 71$. Згенерувати відкритий ключ і зашифрувати відкритий текст: “криптографія”. При переході до цифрового запису тексту кожну букву замінити її двоцифровим десятковим номером з українського алфавіту (див. рис. 2.2) і розбити текст на блоки з чотирьох цифр.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AES – Advanced Encryption Standard (Удосконалений стандарт шифрування)

ASCII – American Standard Code for Information Interchange (Американський стандарт кодів для обміну інформацією)

BBS – Algorithm Blum-Blum-Shub (алгоритм Блюм-Блюма-Шуба)

CBC – Cipher Block Chaining (режим зчеплення блоків шифрованих даних)

CFB – Cipher Feedback (режим зворотного зв’язку за зашифрованими даними)

CTR – Counter mode (режим лічильника)

DEA – Data Encryption Algorithm (Алгоритм шифрування даних)

DES – Data Encryption Standard (Стандарт шифрування даних)

ECB – Electronic Code Book (режим електронної кодової книги)

FIPS – Federal Information Processing Standard (Федеральний стандарт обробки інформації)

FSM – Finite-State Machine (скінченний автомат)

GSM – Global System for Mobile Communications (Глобальний цифровий стандарт мобільного стільникового зв’язку)

IBM – International Business Machines (Корпорація міжнародних бізнес-машин)

Init – функція ініціалізації внутрішнього стану

IP – Initial Permutation (початкова перестановка)

IV – Initialization Vector (вектор ініціалізації)

KSA – Key-Scheduling Algorithm (алгоритм ключового розкладу)

LFSR – Linear Feedback Shift Register (лінійний регістр зсуву зі зворотним зв’язком)

MAC – Message Authentication Code (код автентифікації повідомлення)

Next – функція наступного стану

NIST – National Institute of Standards and Technology (Національний інститут стандартів і технологій)

OFB – Output Feedback (режим зворотного зв’язку за виходом)

RSA – абревіатура від прізвищ Rivest, Shamir і Adleman (криптографічний алгоритм з відкритим ключем)

SPA – Software Publishers Association (асоціація видавців програмного забезпечення)

Strm – функція ключового потоку

TCP – Transmission Control Protocol (протокол управління передачею)

ДССЗЗІ – Державна служба спеціального зв’язку та захисту інформації

ГПВЧ – генераторів псевдовипадкових чисел

ЕОМ – електронно-обчислювальна машина

ЕЦП – електронний цифровий підпис

ІКС – інформаційно-комунікаційна система

КЗП – ключовий запам’ятовуючий пристрій

ЛРЗЗЗ – лінійний регістр зсуву зі зворотним зв’язком

НСД – найбільший спільний дільник

СРСР – Союз Радянських Соціалістичних Республік

США – Сполучені Штати Америки

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Богуш В. М., Мухачов В. А. Криптографічні застосування елементарної теорії чисел: навч. посіб. Київ: Державний університет інформаційно-комунікаційних технологій, 2006. 126 с.

2. Вербицький О. В. Вступ до криптології: навч. посіб. Львів: ВНТЛ, 1998. 248 с.

3. Горбенко І. Д., Гріненко Т. О. Захист інформації в інформаційно-телекомунікаційних системах: навч. посіб. Ч.1. Криптографічний захист інформації. Харків: ХНУРЕ, 2004. 368 с.

4. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія. Теорія. Практика. Застосування: підручник для вищих навчальних закладів. Харків: Форт, 2013. 878 с.

5. Горбенко Ю.І. Побудування та аналіз систем, протоколів і засобів криптографічного захисту інформації: монографія. Ч.1. Методи побудування та аналізу, стандартизація та застосування криптографічних систем / за заг. ред. І. Д. Горбенко. Харків: Форт, 2015. 960 с.

6. Грайворонський М. В., Новіков О. М. Безпека інформаційно-комунікаційних систем: підручник. Київ: BHV, 2009. 608 с.

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

8. ДСТУ ISO/IES 13888-1:1997. Інформаційні технології. Методи захисту. Неспростовність. Частина 1: Загальні положення (переглянуто у 2004 році).

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

10. ДСТУ ISO/IEC 18033-4:2015. Інформаційні технології. Методи захисту. Алгоритми шифрування. Частина 4. Потокові шифри.

11. ДСТУ ISO/IEC 18033-3:2015. Інформаційні технології. Методи захисту. Алгоритми шифрування. Частина 3. Блокові шифри.

12. ДСТУ ISO/IEC 18033-2:2015. Інформаційні технології. Методи захисту. Алгоритми шифрування. Частина 2. Асиметричні шифри.

13. ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевіряння.

14. ДСТУ 7564:2014. Інформаційні технології. Криптографічний захист інформації. Функція хешування.

15. ДСТУ 8961:2019. Інформаційні технології. Криптографічний захист інформації. Алгоритми асиметричного шифрування та інкапсуляції ключів.

16. ДСТУ 9041:2020. Інформаційні технології. Криптографічний захист інформації. Алгоритм шифрування коротких повідомлень, що ґрунтується на скручених еліптичних кривих Едвардса.

17. ДСТУ ГОСТ 28147:2009. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. Державний комітет України з питань технічного регулювання та споживчої політики, Наказ № 495 від 22.12.2008 Київ, 2009.

18. Єсін В. І., Кузнецов О. О., Сорока Л. С. Безпека інформаційних систем і технологій: навч. посіб. Харків: ХНУ ім. В. Н. Каразіна, 2013. 632 с.

19. Задірака В. К., Олексюк О. С. Комп’ютерна криптологія: підручник. Київ: Тернопільська академія народного господарства, НАН України, Інститут кібернетики ім. В.М. Глушкова, 2002. 504 с.

20. Ковальчук Л. В., Конюшок С. М., Кучинська Н. В. Теоретична криптологія: основи алгебри та теорії чисел: навч. посіб. Київ: ІСЗЗІ НТУУ “КПІ ім. Ігоря Сікорського”, 2016. 106 с.

21. Козіна Г. Л. Криптографія від історії до сучасних стандартів: навч. посіб. Запоріжжя: НУ «Запорізька політехніка», 2020. 192 с.

22. Комп’ютерні технології криптографічного захисту інформації на спеціальних цифрових носіях: навч. посіб. / Задірака В. К., Кудін А. М., Людвиченко В. О., Олексюк О. С. Київ, Тернопіль: Підручники і посібники, 2007. 272 с.

23. Корченко О. Г., Дрейс Ю.О. Охорона конфіденційної інформації підприємства: навч. посіб. Житомир: ЖВІ НАУ, 2011. 172 с.

24. Корченко О. Г., Сіденко В. П., Дрейс Ю.О. Прикладна криптологія: системи шифрування: підручник. Київ: ДУТ, 2014. 448 с.

25. Кузнецов О. О., Євсеев С. П., Король О. Г. Стеганографія: навч. посіб. Харків: ХНЕУ, 2011. 232 с.

26. Лагун А. Е. Криптографічні системи та протоколи: навч. посіб. Львів: Львівська політехніка, 2013. 96 с.

27. Математичні основи криптоаналізу: навч. посіб. / Сушко С. О., Кузнецов Г. В., Фомичова Л. Я., Корабльов А. В. Дніпропетровськ: НГУ, 2010. 465 с.

28. Математичні основи криптографії: навч. посіб. / Кузнецов Г. В., Фомичов В. В., Сушко С. О., Фомичова Л. Я. Дніпропетровськ: НГУ, 2004. 391 с.

29. Методи та алгоритми симетричної криптографії: навч. посіб. / О.О. Кузнецов та ін. Кіровоград: КНТУ, 2012. 316 с.

30. Юдін О. К., Корченко О. Г., Конахович Г. Ф. Захист інформації в мережах передачі даних: підручник. Київ: ІНТЕРСЕРВІС, 2009. 716 с.

31. Щур Н. О., Покотило О. А. Основи криптології: навч. посіб. Житомир: Державний університет Житомирська політехніка, 2021. 120 с.

32. Advanced Encryption Standard (AES) [Electronic resource]: FIPS PUB 97. URL:http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

33. Courtois N. T. Security evaluation of GOST 28147-89 in view of international standardisation. Cryptologia 36.1 (2012): 2-13.

34. Ferguson N. and Schneier B. Practical Cryptography: John Wiley & Sons. 2003. 432 p.

35. FIPS 46. Data encryption standard. Federal Information Processing Standards.

36. Forouzan B. Introduction to cryptography and network security. New York: McGraw-Hill, 2008. 752 p.

37. Katz Jonathan, Lindell Yehuda. Introduction to Modern Cryptography. Boca Raton London New York: Chapman & Hall /CRC Taylor & Francis Group; 2008. 534 p.

38. Mao Wenbo. Modern Cryptography. Theory and Practice. Prentice Hall PTR, Upper Saddle River, New Jersey; 2004.

39. Menezes A. J., Oorschot P. C., Vanstone S. A., Handbook of Applied Cryptography. Publisher: CRC Press; 2001. 780 p.

40. Schneier B. Applied cryptography: protocols, algorithms and source code in C. New York: JohnWiley & Sons, Inc., 1995. 792 p.

41. Shannon Claude. Communication Theory of Secrecy Systems. Bell System Technical Journal, vol. 28(4), page 656–715, 1949.