МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ ВІЙСЬКОВИЙ ІНСТИТУТ імені С. П. КОРОЛЬОВА
ДЕРЖАВНОГО УНІВЕРСИТЕТУ ТЕЛЕКОМУНІКАЦІЙ

О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс

ПРИКЛАДНА КРИПТОЛОГІЯ:

системи шифрування

Підручник

*Затверджено Міністерством освіти і науки України
як підручник для студентів вищих навчальних закладів,
які навчаються за напрямом підготовки
«Безпека інформаційних і комунікаційних систем»*

Житомир
2014

УДК 003.26:004.056.55
ББК 32.973202я73
К34

Рекомендовано до друку Вченою радою Державного університету телекомунікацій (протокол № 12 від 23 квітня 2014 року)

Р е ц е н з е н т и:

І. Д. Горбенко – лауреат Державної премії України в галузі науки і техніки, професор, доктор технічних наук, професор кафедри безпеки інформаційних технологій Харківського національного університету радіоелектроніки;

М. Є. Шелест – лауреат Державної премії України в галузі науки і техніки, заслужений діяч науки і техніки, доктор технічних наук, професор, керівник підрозділу Служби зовнішньої розвідки України;

О. О. Кузнецов – професор, доктор технічних наук, професор кафедри безпеки інформаційних систем і технологій Харківського національного університету імені В.Н. Каразіна.

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів (лист № 1/11-10981 від 15.07.2014)

Корченко О. Г.

К34 Прикладна криптологія : системи шифрування : підручник / О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс. – К. : ДУТ, 2014. – 448 с.: іл.

ISBN 978-617-7092-30-7

Підручник містить загальні відомості: про основи криптології, криптографії та криптографічного аналізу; традиційно історичні шифри підстановки й перестановки; блокові та складені шифри й атаки на них; потокові шифри й генератори псевдовипадкових чисел; стандарти криптографічного шифрування та перетворення; блокові симетричні криптоалгоритми; принципи побудови симетричних та асиметричних криптографічних систем шифрування, які використовуються для забезпечення конфіденційності інформації в інформаційно-телекомунікаційних системах; а також приклади з розв'язками, контрольні завдання з відповідями на них та авторські комп'ютерні програми з візуалізацією процесів криптографічного шифрування та перетворення на компакт-диску.

Підручник призначений для студентів вищих навчальних закладів, які навчаються за напрямом підготовки «Безпека інформаційних і комунікаційних систем» в галузі знань «Інформаційна безпека».

Автори: доктор технічних наук, професор **О. Г. Корченко**, **В. П. Сіденко**, кандидат технічних наук **Ю. О. Дрейс**.

УДК 003.26:004.056.55

ISBN 978-617-7092-30-7 **ББК 32.973202я73**

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

AES – Advancer Encryption Standard (Удосконалений стандарт шифрування)
ASCII – American Standard Code for Information Interchange (Американський стандарт кодів для обміну інформацією)
ATM – Asynchronous Transfer Mode (Асинхронний метод перенесення)
BBS – Algorithm Blum– Blum–Shub (алгоритм Блюм–Блюма–Шаба)
CBC – Cipher Block Chaining (режим зчеплення блоків зашифрованих даних)
CFB – Cipher Feedback (режим зворотного зв'язку за зашифрованими даними)
CTR – Counter Mode (режим лічильника)
DES – Data Encryption Standard (Стандарт шифрування даних)
ECB – Electronic Code Book (режим "електронної кодової книги")
FIPS – Federal Information Processing Standards (Федеральний стандарт обробки інформації)
GSM – Global System for Mobile Communications (Глобальний цифровий стандарт мобільного стільникового зв'язку)
IBM – International Business Machines (Корпорація міжнародних бізнес-машин)
IDEA – International Data Encryption Algorithm (Міжнародний алгоритм шифрування даних)
IPES – Improved Proposed Encryption Standard (Удосконалений запропонований стандарт шифрування)
IST – Information Societies Technology (Інформаційні громадські технології)
IV – Initialization Vector (вектор ініціалізації)
KSA – Key-Scheduling Algorithm (алгоритм ключового розкладу)
LFSR – linear feedback shift register (регістр зсуву з лінійним зворотним зв'язком)
MA – Multiplication/Addition (множення/складання)
MAC – Message Authentication Code (код автентифікації повідомлення)

MDC – Modification/Manipulation Detection Code (код виявлення змін/маніпуляцій)
MIT – Massachusetts Institute of Technology (Массачусетський технологічний інститут)
NFS – Number Field Sieve (Загальний метод решета числового поля — метод факторизації цілих чисел)
NIST – National Institute of Standards and Technology (Національний інститут стандартів і технології)
OFB – Output Feedback (режим зворотного зв'язку за виходом)
PES – Proposed Encryption Standard (Запропонований стандарт шифрування)
RSA – абревіатура від прізвищ Rivest, Shamir і Adleman (криптографічний алгоритм з відкритим ключем)
SPA Software Publishers Association (Асоціація видавців програмного забезпечення)
TCP – Transmission Control Protocol (Протокол управління передачею)
АПШ – асинхронні потокові шифри
ГПВЧ – генератор псевдовипадкових чисел
ГППВЧ – генератор послідовності псевдовипадкових чисел
ДК – диференціальний криптографічний аналіз
ЕОМ – електронна обчислювальна машина
ЕЦП – електронний цифровий підпис
ІТС – інформаційно-телекомунікаційна система
КЗП – ключовий запам'ятовуючий пристрій
ЛК – лінійний криптографічний аналіз
ПВЧ – послідовність псевдовипадкових чисел
ППО – протиповітряна оборона
СПШ – синхронні потокові шифри
СРСР – Союз Радянських Соціалістичних Республік
ФАУЗІ – Федеральне агентство урядового зв'язку та інформації при Президентові РФ

ВСТУП

На сьогодні первинним чинником, що впливає на політичну й економічну складові національної безпеки, є ступінь захищеності інформації й інформаційного середовища. Ось чому важливого значення набувають питання забезпечення захисту інформації в автоматизованих (інформаційних) і телекомунікаційних системах під час обробки у різних сферах діяльності.

Під *обробкою інформації в системі* розуміють виконання однієї або кількох операцій, зокрема: збирання, введення, записування, перетворення, зчитування, зберігання, знищення, реєстрації, приймання, отримання, передавання, які здійснюються в системі за допомогою технічних і програмних засобів. Такі програмні засоби (або програмне забезпечення), як і сама інформація, що обробляється, є *об'єктами захисту*. У цілому захисту потребує інформація з обмеженим доступом (критична інформація), а саме: конфіденційна, службова або таємна інформація (наприклад, державна таємниця, банківська таємниця або інша таємниця). Під час обробки інформації з обмеженим доступом повинен забезпечуватися її захист від несанкціонованого та неконтрольованого ознайомлення, модифікації, знищення, копіювання, поширення, тобто забезпечення основних її властивостей захищеності — конфіденційності, цілісності, доступності та спостереженості. Зокрема, передача службової й таємної інформації з однієї системи до іншої здійснюється в зашифрованому вигляді або захищеними каналами зв'язку за допомогою технічного та *криптографічного захисту інформації.* Останній, своєю чергою, реалізується програмними, програмно-апаратними та апаратними засобами шляхом перетворення інформації з використанням

спеціальних (ключових) даних із метою приховування/відновлення змісту інформації, підтвердження її справжності, цілісності, авторства тощо. Сукупністю засобів криптографічного захисту інформації, необхідної ключової, нормативної, експлуатаційної, а також іншої документації (зокрема такої, що визначає заходи безпеки), використання яких забезпечує належний рівень захищеності інформації, що обробляється, зберігається та (або) передається називають *криптографічною системою* (криптосистемою). А сукупність органів, підрозділів, груп, діяльність яких спрямована на забезпечення криптографічного захисту інформації, та підприємств, установ і організацій, що розробляють, виробляють, експлуатують та (або) розповсюджують криптосистеми й засоби криптографічного захисту інформації – *системою криптографічного захисту інформації*.

Формування загальних принципів побудови систем криптографічного захисту інформації (систем шифрування) на основі використання сучасних алгоритмів симетричного й асиметричного шифрування для забезпечення конфіденційності даних в інформаційно-телекомунікаційній системі (ІТС) є *метою даного підручника*, який складається з таких розділів:

перший розділ містить загальні відомості про основи криптології, криптографії та криптографічного аналізу. У ньому розкрито основи захисту інформації в ІТС, роль криптографічних протоколів у загальній задачі забезпечення інформаційної безпеки, основи теорії засекреченого зв'язку *К. Шеннона*, наведено класифікацію методів шифрування повідомлень;

другий розділ присвячений традиційно історичним шифрам підстановки (моноалфавітні, багатоалфавітні) та перестановки;

третій розділ визначає принципи побудови сучасних симетричних криптографічних систем, що використовують блокові та складені шифри з криптографічним аналізом атаки на блокові шифри;

у *четвертому* розділі розкрито потокові шифри (*A5, RC4*) та генератори псевдовипадкових чисел, заснованих на алгоритмі *BBS* та регістрів зсуву зі зворотним зв'язком;

п'ятий розділ розкриває суть стандарту блокового симетричного шифрування *Data Encription Standard* (*DES*): історія створення, принципи побудови, структура, приклади шифрування та криптографічний аналіз можливих атак;

шостий розділ містить режими виконання криптографічних алгоритмів блокового симетричного шифрування даних;

сьомий розділ розкриває суть міжнародного стандарту шифрування даних *Internalional Data Encryption Algorithm* (*IDEA*): історію створення, принципи побудови, структуру, приклади шифрування та криптографічний аналіз можливих атак;

у *восьмому* розділі детально розглядається стандарт криптографічного перетворення *ДСТУ ГОСТ 28147:2009*: у режимі простої заміни, режимі гамування, режимі гамування зі зворотним зв'язком та режимі утворення імітовставки;

дев'ятий розділ присвячений блоковим симетричним криптоалгоритмам *Rijndael* та *Advanced Encryption Standard* (*AES*): структура стандарту, раундові перетворення тощо;

у *десятому* розділі розглянуто асиметричні криптографічні системи: *Діффі–Хеллмана*, *Шаміра*, *Ель-Гамаля*, *RSA*, *Рабіна*; методи зламу криптосистем, заснованих на дискретному логарифмуванні.

До кожного розділу додаються *контрольні питання та завдання* для самоконтролю засвоєних знань, а також відповіді на контрольні завдання для самоперевірки правильності їх виконання.

Слід зазначити, що до підручника додається *компакт-диск* (CD) на якому міститься достатня кількість комп'ютерних програм шифрування даних розглянутих криптоалгоритмів із можливостями покрокової демонстрації візуалізації процесів зашифрування / розшифрування, на які отримано авторські свідоцтва. На CD є електронна версія даного підручника й додаткові програми функціонального призначення.

Структуру та зміст підручника визначено, виходячи зі змісту робочої програми навчальної дисципліни «Прикладна криптологія», її змістовних модулів, тем навчальних занять, теоретичного та практичного досвіду авторів.

Підручник призначений для студентів вищих навчальних закладів, які навчаться в галузі знань «Інформаційна безпека» за напрямом підготовки 6.170101 «Безпека інформаційних і комунікаційних систем» та спеціалістів з експлуатації й застосування криптографічних систем, комплексів і засобів захисту інформації в ІТС, а також може бути використаний аспірантами й науковцями при проведенні наукових досліджень.

Розділ 1

ОСНОВИ КРИПТОЛОГІЇ, КРИПТОГРАФІЇ ТА КРИПТОГРАФІЧНОГО АНАЛІЗУ

1.1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ЗАХИСТ ІНФОРМАЦІЇ В ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМАХ

Стрімкий розвиток засобів обчислювальної техніки й відкритих мереж передачі даних зумовив їх широке розповсюдження в повсякденному житті й підприємницькій діяльності. Потужні обчислювальні можливості й оперативність передачі інформації не лише вплинули на принципи ведення бізнесу, що склалися в більшості традиційних галузей, але й відкрили нові напрями розвитку підприємницької діяльності [15].

Проте 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 ІТС можуть бути реалізовані безпосередньою дією на інформацію, що становить інтерес для кінцевих користувачів подібних систем, так і на інформаційні ресурси й телекомунікаційні служби, що забезпечуються в межах цієї ІТС [4, 8–11]. 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 обробляються, зберігаються й передаються в ІТС від незаконного ознайомлення, перетворення та знищення (як крайній

випадок модифікації), а також стан захищеності інформаційних ресурсів від дій, спрямованих на порушення їх працездатності [8–11].

Основними завданнями захисту інформації, яка призначена для користувача, є забезпечення [4, 8–11]:

- конфіденційності інформації;
- цілісності інформації;
- достовірності інформації;
- оперативності доступу до інформації;
- юридичної значущості інформації, наданої у вигляді електронного документа;
- невідстежуваності дій клієнта.

Конфіденційність — властивість інформації бути доступною тільки обмеженому колу користувачів ІТС, в якій циркулює ця інформація.

Під *цілісністю* розуміють властивість інформації або програмного забезпечення зберігати свою структуру та зміст у процесі передачі та зберігання. Розглядаючи питання передачі інформації у вигляді повідомлення через мережу, можна дійти висновку, що кожне повідомлення за своїм змістом утворює деякий клас. Іншими словами, сенс кінцевого повідомлення залишиться таким самим, як і початкового, навіть якщо форма представлення інформації в електронному вигляді істотно зміниться. Отже, кожне повідомлення будь-якою мовою буде мати свій клас еквівалентності, і для даного випадку властивість збереження цілісності можна сформулювати таким чином: передане повідомлення *M* вважається таким, що зберегло цілісність, якщо отримане внаслідок передачі повідомлення *M´* належить до класу еквівалентності повідомлення *M*.

Достовірність — властивість інформації, що виражається в строгій приналежності до об'єкта, який є її джерелом, або того об'єкта, від якого цю інформацію прийнято.

Оперативність — здатність інформації або деякого інформаційного ресурсу бути доступною для кінцевого користувача відповідно до його тимчасових потреб.

Юридична значущість означає, що документ має юридичну силу. З цією метою суб'єкти, які потребують підтвердження юридичної значущості повідомлення, що передається, домовляються про повсюдне 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 виражає її властивість бути юридично значущою. Інформацію необхідно сформувати так, щоб із формального погляду було достовірно зрозуміло, що тільки відправник, якому належить цей платіжний документ, міг його створити.

Невідстежуваність — здатність здійснювати деякі дії в ІТС непомітно для інших об'єктів. Актуальність цієї вимоги стала очевидною завдяки появі таких понять, як електронні гроші й *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 контролювати, тим самим виникнуть умови для тотального стеження за користувачами ІТС.

Існує декілька шляхів вирішення проблеми невідстежуваності:

- заборона за допомогою законодавчих актів будь-якого тотального стеження за користувачами ІТС;
- 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 ІТС виконувати задані функції.

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

- від порушення функціонування телекомунікаційної системи, що виражається в дії на інформаційні канали, канали сигналізації, управління й віддаленого завантаження баз даних комутаційного устаткування, системне та прикладне програмне забезпечення;
- від несанкціонованого доступу до інформаційних ресурсів і від спроб використання ресурсів мережі, що призводять до витоку даних, порушення цілісності мережі й інформації, зміни функціонування підсистеми розподілу інформації, доступності баз даних;
- від руйнування вбудованих і зовнішніх засобів захисту;
- від неправомірних дій користувачів і обслуговуючого персоналу мережі.

Пріоритети серед перерахованих завдань інформаційної безпеки визначаються індивідуально для кожної конкретної ІТС і залежать від вимог, що ставляться безпосередньо до інформаційних систем.

Слід врахувати, що з погляду державних структур заходи захисту, першою чергою, 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, імовірно, найважливішими є цілісність і доступність даних та послуг щодо їх обробки. У порівнянні з державними комерційні організації є більш відкритими й динамічними, тому ймовірні до них загрози відрізняються не лише кількістю, але і якістю.

Для вирішення завдання інформаційної безпеки в ІТС необхідно забезпечити:

- захист інформації під час її зберігання, обробки та передачі по каналах зв'язку;
- підтвердження достовірності об'єктів даних і користувачів (автентифікації сторін, що встановлюють зв'язок);
- виявлення й 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 виражатися в організаційних або технічних заходах захисту інформації.

Слід зазначити, що обсяг заходів із захисту даних, які обробляються й передаються, залежить передусім від величини передбачуваного збитку. Ця величина може виражатися в прямій (наприклад,

витрати на купівлю нового програмного забезпечення у разі порушення його цілісності) або в опосередкованій (наприклад, витрати від простою інформаційної системи банку) формах. Правда, у деяких ситуаціях розрахувати величину збитку складно (наприклад випадку витоку державної таємниці).

1.2. РОЛЬ КРИПТОГРАФІЧНИХ ПРОТОКОЛІВ У ЗАГАЛЬНІЙ ЗАДАЧІ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

Основу забезпечення інформаційної безпеки в ІТС 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 і криптографічних заходів.

В основі криптографічних методів лежить поняття криптографічного перетворення інформації встановленого за певними математичними законами від несанкціонованого доступу та неможливості безконтрольної її зміни сторонніми користувачами.

Застосування криптографічних методів захисту забезпечує вирішення основних завдань інформаційної безпеки. Цього можна досягнути шляхом реалізації таких криптографічних методів захисту: як конфіденційної та службової інформації, так і інформаційних ресурсів у цілому можна досягнути шляхом:

- шифрування всього інформаційного потоку, що передається через відкриті мережі передачі даних, й окремих повідомлень;
- криптографічної автентифікації різнорівневих об'єктів, що встановлюють зв'язок (йдеться про 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;
- шифрування даних, що надані у вигляді файлів або зберігаються в базі даних;
- контролю цілісності програмного забезпечення за допомогою застосування криптографічних стійких контрольних сум;
- застосування ЕЦП для забезпечення юридичної значущості платіжних документів;

- застосування затемнюючого 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 ІТС супроводжується обміном ідентифікуючої та автентифікуючої інформації.

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

Зокрема, *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 наступним 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 систему розподілу ключової інформації в розподілених мережах передачі даних);

- автентифікація сторін, що встановлюють зв'язок;

- авторизація користувачів при доступі до телекомунікаційних і інформаційних служб.

На сьогодні, завдяки широкому застосуванню відкритих мереж передачі даних, таких як *Internet*, та 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 окремих 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 КЛАСИЧНУ КРИПТОЛОГІЮ, КРИПТОГРАФІЮ ТА КРИПТОГРАФІЧНИЙ АНАЛІЗ

1.3.1. Загальні відомості про криптологію

Криптологія — це галузь знань, що вивчає тайнопис (криптографію), і методи її розкриття (криптографічний аналіз), яка за влучним висловлюванням Рональда Рівеста (професора Массачусетського технологічного інституту — *MIT*) і одного з авторів знаменитої криптографічної системи *RSA*, є "повитухою усієї *computer science* (комп'ютерної науки) взагалі" [3, 7, 10]. Назва криптології пішла від грецької мови (*криптос* — таємний, *логос* — наука або 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. Проте, незважаючи на органічно властиву їй складність, багато теоретичних досягнень криптології наразі широко використовуються в нашому насиченому інформаційними технологіями житті, наприклад: у пластикових *smart*-картах, електронній пошті, 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 застосованості для теоретичної науки, напевно, унікальне.

Криптографія — це розділ прикладної математики, що вивчає моделі, методи, алгоритми, програмні й апаратні засоби перетворення інформації (шифрування) у цілях приховання її змісту, запобігання зміні або несанкціонованого використання. Іншими словами, криптограф намагається знайти методи забезпечення секретності і/чи автентичності (достовірності) повідомлення.

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* [37]. Забезпечення цілісності інформації в умовах випадкової дії знаходиться у віданні *теорії завадостійкого кодування* [11]. Нарешті, суміжною областю відносно криптології є математичні методи стиснення інформації.

У даному підручнику 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 [6]: наївний, формальний, науковий, комп'ютерний.

Для *наївної криптології* (до початку XVI ст.) характерне використання будь-яких, зазвичай примітивних, способів заплутування супротивника відносно змісту шифрованих текстів. На початковому етапі для захисту інформації використовувалися методи кодування та 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 літери початкового тексту 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 складає 5×5). Кожна літера початкового тексту замінюється на літеру, 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 сталося в епоху Відродження, коли розвиток науки й 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 XVI ст. *Блеза Віженера,* полягав у послідовному "складанні" літер початкового тексту з ключем (процедуру можна полегшити за допомогою спеціальної таблиці). Його робота "Трактат про шифр" (1466 р.) вважається першою науковою роботою 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 літер (біграм).

Простим, але стійким способом багатоалфавітної заміни (підстановки біграм) є *шифр Плейфера*, який був відкритий на початку XIX ст. *Чарльзом Уітстоном*. Уітстону належить і важливе удосконалення — шифрування "подвійним квадратом". Шифри Плейфера й 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 криптографії, 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 1790 р. *Томасом Джефферсоном*, майбутнім президентом США, механічна машина. Багатоалфавітна підстановка за допомогою роторної машини реалізується варіацією взаємного положення 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 1917 р. *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 надстійкі 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* (1930–1960-ті рр.) — поява криптографічних систем зі строгим математичним обґрунтуванням криптографічної стійкості. На початок 30-х рр. XX ст. остаточно сформувалися розділи математики, що є науковою основою криптології: теорія вірогідності й 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.), який підвів наукову базу під криптографію та криптографічний аналіз. Із того часу почали говорити про криптологію — як науку про перетворення інформації для забезпечення її секретності. Етап розвитку криптографії та криптографічного аналізу до 1949 р. почали називати донауковою криптологією. Шеннон ввів поняття "*розсіювання*" й "*перемішування*", обґрунтував можливість створення скільки завгодно стійких криптографічних систем.

У 1960-х рр. провідні криптографічні школи підійшли до створення блокових шифрів, ще стійкіших у порівнянні з роторними криптографічними системами, які не зважаючи на це допускають практичну реалізацію тільки у вигляді цифрових електронних пристроїв.

Комп'ютерна криптографія (з 1970-х рр.) з'явилася завдяки 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 можливим із появою потужних і компактних обчислювальних засобів, стали блокові шифри. У 70-ті рр. XX ст. розробили американський стандарт шифрування *DES — Data Encryption Standard* (*Стандарт шифрування даних*), прийнятий 1978 р. Один із його авторів, *Хорст Фейстель* (співробітник *IBM*), описав модель блокових шифрів, на основі якої було побудовано інші, стійкіші симетричні криптосистеми, зокрема стандарт шифрування *ГОСТ 28147-89*, який був розроблений ще в СРСР 1989 р.

Із появою *DES* збагатився й 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 рр. XX ст. стався справжній прорив у сучасній криптографії — поява асиметричних криптосистем, які не вимагали передачі секретного ключа між сторонами. Тут, відправною точкою заведено вважати працю, опубліковану *Уітфілдом Діффі* та *Мартіном Хеллманом* 1976 р. під назвою "*Нові напрями в сучасній криптографії*", де вперше сформульовано принципи обміну шифрованою інформацією без обміну секретного ключа. Незалежно від 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 ЕЦП і електронних грошей.

У 1980-90-х рр. з'явилися абсолютно нові напрями криптографії: імовірнісне шифрування, квантова криптографія та інші. Усвідомлення їх практичної цінності ще попереду. Актуальним залишається також завдання вдосконалення симетричних криптографічних систем. У цей самий період було розроблено *нефейстелейвські шифри* (*SAFER*, *RC6* та ін.), а 2000 р. після відкритого міжнародного конкурсу прийнято новий Національний стандарт шифрування США —

AES (*Advanced Encryption Standard*) — Удосконалений стандарт шифрування.

Криптографія є одним із найбільш потужних засобів забезпечення конфіденційності й контролю цілісності інформації. Вона посідає центральне місце серед програмно-технічних регуляторів безпеки. Наприклад, для портативних комп'ютерів, які вкрай важко фізично захистити, тільки криптографія дозволяє гарантувати конфіденційність інформації навіть у разі її крадіжки.

1.3.3. Основні визначення, використовувані в криптології

Способи й методи перетворення (шифрування) інформації з метою її захисту від незаконних користувачів (від несанкціонованого доступу) називаються *шифрами* [13, 22–24, 29, 30, 32].

Шифрування (зашифрування) — 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 літерною або числовою послідовністю [23, 33]. Ця послідовність наче "настроює" алгоритм шифрування.

У деяких джерелах окремо виділяють термін "*дешифрування*", розуміючи під цим відновлення початкового повідомлення на основі шифрованого без знання ключа, тобто методами криптографічного аналізу. Надалі будемо вважати розшифрування й дешифрування синонімами. Також термін "шифрування" (у вузькому значенні) використовується як синонім зашифрування. Проте неправильно як синонім шифрування використовувати термін "*кодування*" (а замість "*шифру*" — "*код*"), оскільки під кодуванням зазвичай розуміють представлення інформації у вигляді знаків (літер алфавіту).

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.

Криптографічним протоколом називається такий, в основі якого лежить криптографічний алгоритм. Проте метою криптографічного протоколу часто є не лише збереження інформації в таємниці від сторонніх. Учасники криптографічного протоколу можуть бути близькими друзями, які не мають один від одного таємниць, але й можуть бути настільки непримиренними ворогами, що кожен із них відмовляється повідомити іншому, наприклад, яке сьогодні число. Проте, їм може знадобитися поставити велику кількість підписів 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 вимагається, щоб криптографічний протокол забезпечував таке: його учасники не можуть зробити або дізнатися більше того, що визначено протоколом [1, 42, 45].

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 здійснюється для зашифрування повідомлення. Відповідно можуть розрізнятися ключі зашифрування й розшифрування. Пару алгоритмів зашифрування й розшифрування зазвичай називають *криптографічною системою*, а пристрої, що реалізовують їх, — *шифрувальними пристроями*.

1.3.4. Узагальнена схема криптографічної системи

Криптографічна система — це система, реалізована програмно, апаратно або програмно-апаратно і здійснює криптографічне перетворення інформації з метою її захисту [1–45].

Припустимо, що відправник бажає надіслати повідомлення одержувачу. Більше того, цей відправник хоче послати своє повідомлення безпечно: він бажає бути впевненим, що зловмисник, який перехопив носій, не зможе впізнати зміст повідомлення. Саме повідомлення, що передається, називається *відкритим текстом* (пові-

домленням, даними). Зміна виду повідомлення з метою приховання його суті називається *зашифруванням*. Зашифроване повідомлення називається *зашифрованим текстом* (повідомленням, даними). Процес перетворення зашифрованого тексту у відкритий текст називається *розшифруванням* (дешифруванням за криптографічним аналізом).

Узагальнену схему криптографічного перетворення, що забезпечує шифрування передаваної інформації (повідомлень, текстів, даних) і її розшифрування, показано на рис. 1.1.

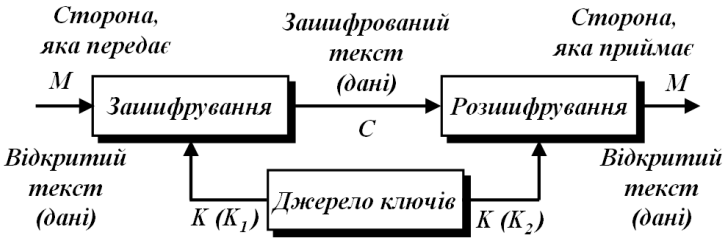


Рис. 1.1. Узагальнена схема криптографічного перетворення

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. Далі розглядаються тільки двійкові дані й комп'ютерна криптографія.

Позначимо шифротекст як C (від *ciphertext* — *зашифрований текст*). Це також двійкові дані, іноді того самого розміру, що й M, іноді й більшого. Якщо шифрування супроводжується стисненням, C може бути менше M. Проте 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 метою зашифрування й наступного 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 функція E і D у виразах (1.1) і (1.2)), яка використовується для зашифрування 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 виходить з групи, її члени повинні переходити на інший алгоритм. Алгоритм потрібно замінити, якщо хто-небудь іззовні 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 зашифрування (1.1) і розшифрування (1.2) будуть мати такий вигляд:

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

і

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

при цьому повинно виконуватися:

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

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

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

а рівність (1.5):

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

Як інформацію, що підлягає зашифруванню й розшифруванню, а також електронному підпису, будуть розглядати тексти (повідомлення), побудовані на деякому алфавіті. Під цими термінами розуміють таке: *алфавіт* — кінцева безліч знаків, що використовується для кодування інформації; *текст* (повідомлення, дані) — упорядкований набір з елементів алфавіту.

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

- алфавіт Z_{26} — 26 літер англійського алфавіту (виключаючи пропуск);
- алфавіт Z_{32} — 32 літери російського алфавіту (виключаючи "ё") і пропуск, а також букви українського алфавіту (виключаючи "ґ") і пропуск;
- алфавіт Z_{256} — 256 символів, що входять у стандартний код *ASCII*;
- алфавіт $Z_{16} = \{0,1,\ldots,f\}$ (шістнадцять символів шістнадцяткового алфавіту);
- алфавіт $Z_2 = \{0,1\}$ (два символи 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 можна опублікувати та проаналізувати. Продукти, що використовують цей алгоритм, можуть широко тиражуватися. Як було визначено раніше фундаментальне правило криптографічного аналізу, уперше сформульоване голландцем О. Керкгоффсом ще в XIX ст., 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 зашифрування й розшифрування, крім секретного ключа, відомий криптографічному аналітику супротивника. Це обумовлено тим, що криптографічна система, яка реалізовує сімейство криптографічних перетворень, зазвичай розглядається як відкрита система. Такий підхід відбиває дуже важливий принцип технології захисту інформації: захищеність системи не повинна залежати від секретності чого-небудь такого, що неможливо швидко змінити в разі витоку таємної (конфіденційної) інформації. Зазвичай

криптографічна система є сукупністю апаратних і програмних засобів, яку можна змінити тільки за значних витрат часу та засобів, 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 К. ШЕННОНА

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, ховається за допомогою криптографічних перетворень [43]. При цьому сам факт передачі інформації не приховується. В основі кожної системи засекреченого зв'язку — використання алгоритмів шифрування як основного засобу збереження конфіденційності.

К. Шеннон розглянув модель (рис. 1.2), в якій джерело повідомлень породжує відкритий текст *M*.

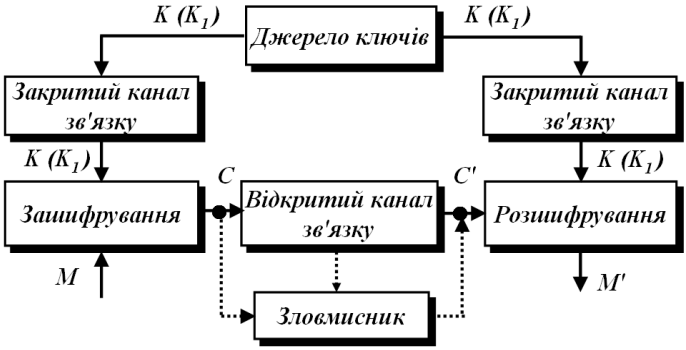


Рис. 1.2. Модель системи передачі шифрованих повідомлень

Джерело ключів генерує ключ *K*. Шифрувальний пристрій 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 супротивника є отримання відкритого тексту та ключа на основі аналізу зашифрованого тексту.

Шеннон розглянув питання теоретичної й практичної секретності. Для визначення теоретичної секретності Шеннон сформулював такі питання:

1. Наскільки стійкою є система, якщо криптографічний аналітик супротивника не обмежений у часі та має всі необхідні засоби для аналізу криптограм?

2. Чи має шифрований текст (криптограма) єдиний розв'язок?

3. Який об'єм зашифрованого тексту необхідно перехопити криптографічному аналітику, щоб розв'язок став єдиним?

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

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

де $p(M)$ — апріорна ймовірність повідомлення M; $p(C/M)$ — умовна ймовірність криптограми C за умови, що обрано повідомлення M, тобто сума ймовірностей усіх тих ключів, які приводять повідомлення M у криптограму C; $p(C)$ — ймовірність отримання криптограми C; $p(M/C)$ — апостеріорна ймовірність повідомлення M за умови, що перехоплено криптограму 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 криптографічних системи значення $p(M/C)$ і $p(M)$ повинні бути рівні для усіх 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.

Якщо $p(C/M) = p(C)$, то з (1.8) випливає, що $p(M/C) = p(M)$ і система цілком таємна.

Отже, можна сформулювати: 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 повинне залежати від M.

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

Далі, повинно 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 цілком таємних систем для кожного із цих C і будь-якого M

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

Отже, знайдеться принаймні один ключ, що відображує це 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 цілком секретною. Один приклад такої системи показано на рис. 1.3, де $s = (i + j - 1) \bmod 5$.

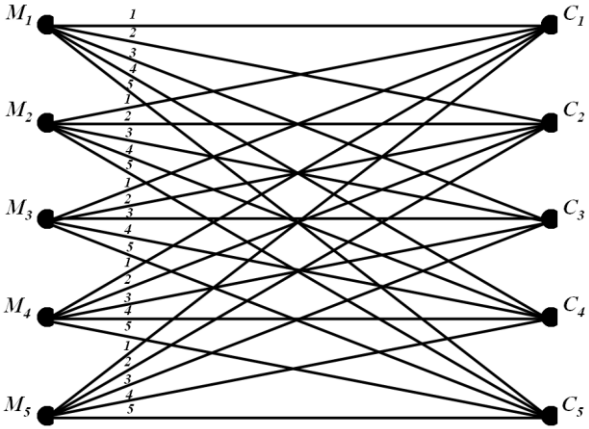


Рис. 1.3. Досконала криптографічна система

1.5. КЛАСИФІКАЦІЯ МЕТОДІВ ШИФРУВАННЯ ПОВІДОМЛЕНЬ

Класична або *одноключова криптографія* спирається на використання симетричних алгоритмів шифрування, у яких зашифрування й розшифрування відрізняються тільки порядком виконання та напрямком деяких кроків. Ці алгоритми використовують той самий секретний елемент (ключ), і друга дія (розшифрування) є простим оберненням першої (зашифрування). 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 і джерело ключів. Джерело ключів вибирає конкретний ключ K серед усіх можливих ключів даної системи. Цей ключ K передається деяким способом приймаючій стороні, причому передбачається, що його не можна перехопити, наприклад, ключ передається спеціальним кур'єром (тому симетричне шифрування називається також шифруванням із закритим ключем). Джерело повідомлень формує деяке повідомлення M, яке потім зашифровується з використанням вибраного ключа. Внаслідок процедури зашифрування 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 перехоплене противником. На приймаючій стороні криптограму C за допомогою ключа розшифровують і отримують вхідне повідомлення M.

Через велику надмірність природних мов безпосередньо в зашифроване повідомлення надзвичайно складно внести осмислену зміну, тому класична криптографія забезпечує також захист від нав'язування помилкових 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, званої *імітовставкою*.

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

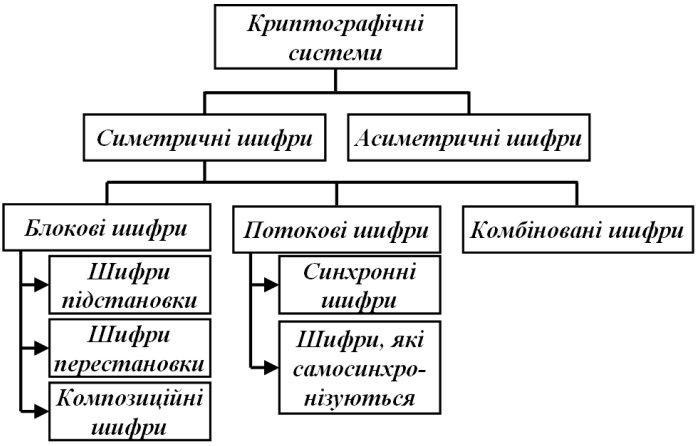


Рис. 1.4. Класифікація методів шифрування повідомлень

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

У *симетричних* криптографічних системах зашифрування й розшифрування проводяться за допомогою того самого 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 зашифровування, який ще називають *відкритим*, інший — для розшифрування, який називають *закритим*. 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, використаної симетричною криптографічною системою, зашифровується з використанням асиметричної криптографічної системи.

За типом обробки вхідної інформаційної послідовності криптографічні системи поділяються на *потокові*, у яких перетворюються всі повідомлення одразу, 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 зашифрування 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 зашифрування повідомлення.

Основна відмінність сучасної від докомп'ютерної криптографії полягає у тому, що раніше криптографічні алгоритми оперували символами природних мов, наприклад, літерами англійської чи російської (української) мови. Ці літери 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 криптостійкої системи шляхом багаторазового застосування відносно простих криптографічних перетворень, які К. Шеннон запропонував використовувати як перетворення *підстановки* (*substitution*) і *транспозиції* (*permutation*). Багаторазове використання цих перетворень дозволяє забезпечити дві властивості, які повинні бути притаманні стійким шифрам: *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 вхідного 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 і т.д.

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 [6].

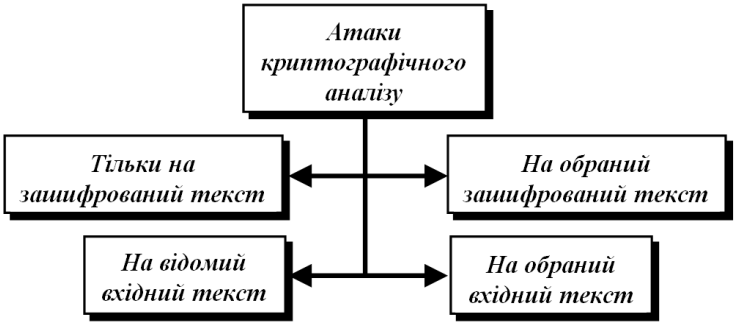


Рис. 1.5. Класифікація атак криптографічного аналізу

1.6.1. Атака на зашифрований текст

В *атаці 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 на зашифрований текст — найімовірніша, тому що криптографічному аналітику для неї потрібен тільки сам текст. Шифр повинен серйозно перешкоджати цьому типу атаки й не дозволити дешифрування повідомлення противником. Рис. 1.6 ілюструє процес атаки.

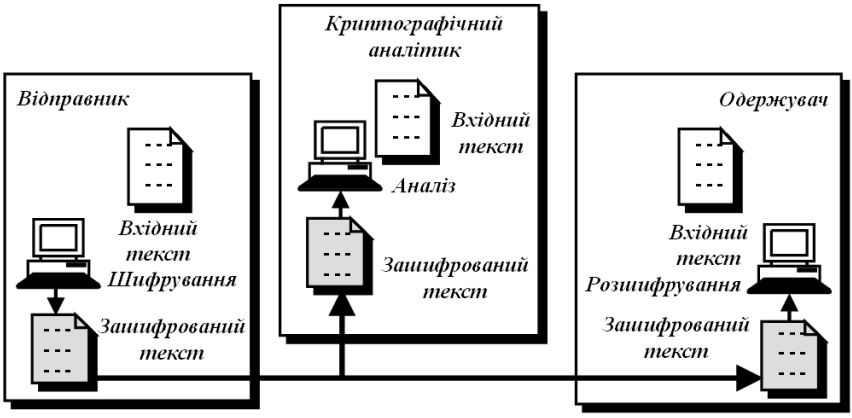


Рис. 1.6. Пояснення процесу атаки тільки на зашифрований текст

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 пробує 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. Створення атаки "грубої сили" було в минулому важким завданням; сьогодні за допомогою комп'ютера це стало простіше. Щоб запобігти цьому типу атаки, 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 літера *E* — найбільш часто використовувана літера в англійському тексті. Криптографічний аналітик знаходить найбільш часто використовуваний символ у зашифрованому тексті та приймає, що це відповідний символ вхідного тексту — *E*. Після визначення кількох пар аналітик може знайти ключ і розшифрувати повідомлення. Щоб запобігти цьому типу атаки, 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.

1.6.2. Атака на відомий вхідний текст

При *атаці знання вхідного тексту* криптографічний аналітик має доступ до деяких пар "*вхідний/зашифрований текст*" на додаток до перехопленого зашифрованого тексту, який він хоче зламати, як показано на рис. 1.7.

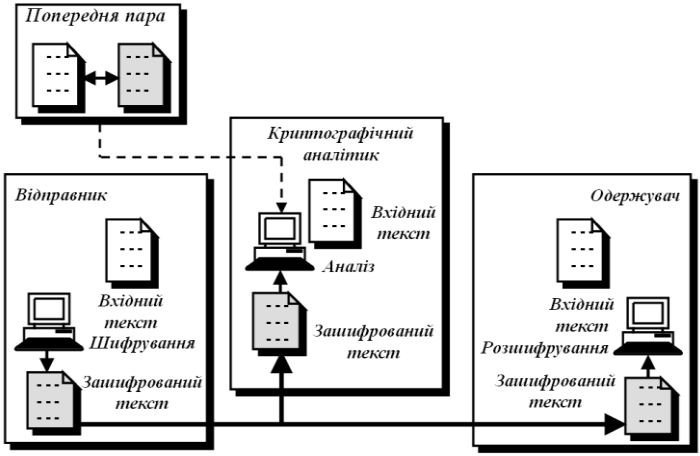


Рис. 1.7. Пояснення процесу атаки на відомий вхідний текст

Пари вхідного/зашифрованого тексту були зібрані раніше. Наприклад, відправник передав секретне повідомлення одержувачу, але він пізніше відкрив зміст повідомлення стороннім. Криптографічний аналітик зберігав і зашифрований текст, і вхідний текст, щоб використовувати їх, коли знадобиться зламати наступне секретне повідомлення від відправника до одержувача, 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 для зашифрованого тексту, можуть бути застосовані й 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.

1.6.3. Атака на обраний вхідний текст

Атака з вибіркою вхідного тексту подібна до атаки знання вхідного тексту, але пари "вхідний/зашифрований текст" були вибрані і виготовлені самим нападником. Цей процес ілюструє рис. 1.8.

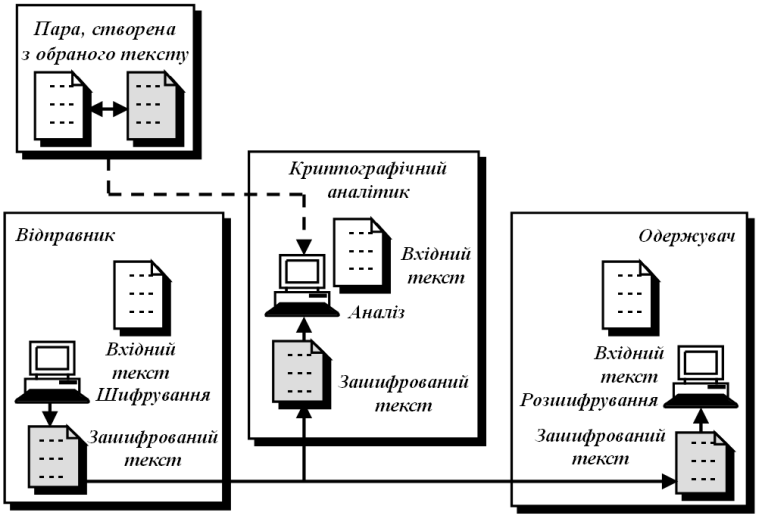


Рис. 1.8. Пояснення процесу атаки на обраний вхідний текст

Це може статися, наприклад, якщо криптографічний аналітик має доступ до комп'ютера відправника. Він вибирає певний вхідний 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 ймовірний, оскільки має на увазі занадто багато "*якщо*".

1.6.4. Атака на обраний зашифрований текст

Атака на обраний зашифрований текст подібна до атаки на обраний вхідний текст, за винятком того, що криптографічний аналітик вибирає певний зашифрований текст і розшифровує його, щоб сформувати пару "зашифрований/вхідний текст" (це трапляється, коли аналітик має доступ до комп'ютера відправника). Рис. 1.9 показує цей процес.

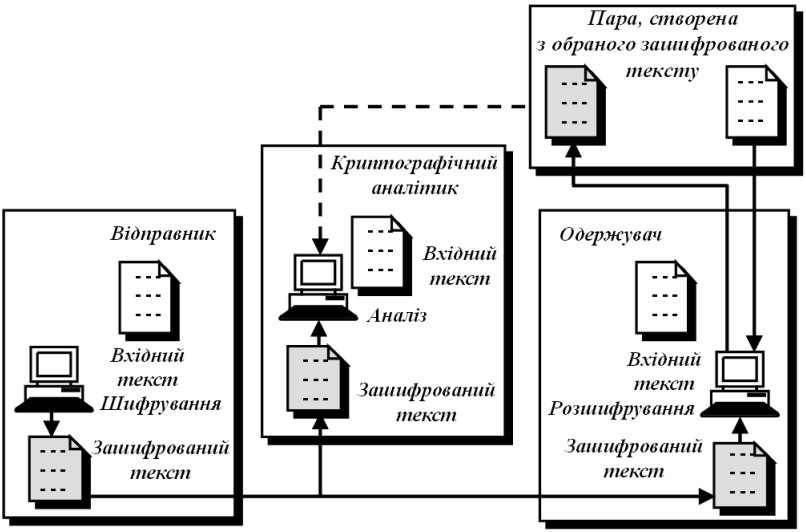


Рис. 1.9. Пояснення процесу атаки на обраний зашифрований текст

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

Контрольні питання та завдання

1. Назвати проблеми, під час вирішення яких можуть використовуватися криптографічні методи.
2. Перерахувати основні завдання захисту інформації, яка призначена для користувача.
3. Дати визначення властивостей інформації: конфіденційність; цілісність; достовірність.
4. Дати визначення: шифру; ключа; шифрування (розшифрування); криптографічного алгоритму; криптографічної системи.
5. Дати визначення криптографічного протоколу.
6. Дати пояснення необхідності виконання умови (1.5) при шифруванні даних.
7. Сформулювати необхідну й достатню умови для досконалої секретності криптографічної системи.
8. Дати пояснення сутності розсіювання даних у процесі їх шифрування.
9. Що є метою перемішування даних у процесі їх шифрування?
10. Що таке криптографічна атака?
11. Які типи криптографічних атак існують?
12. Дати характеристику атаки на зашифрований текст. Пояснити сутність атаки “грубої сили”.
13. Дати характеристику атаки на зашифрований текст. Пояснити сутність статистичної атаки.
14. Дати характеристику атаки на зашифрований текст. Пояснити сутність атаки за зразком.
15. Дати характеристику атаки на відомий вхідний текст.
16. Дати характеристику атаки на обраний вхідний текст.
17. Дати характеристику атаки на обраний зашифрований текст.

Розділ 2
ТРАДИЦІЙНІ ІСТОРИЧНІ ШИФРИ

Традиційні шифри з симетричним ключем можна розділити на дві великі категорії: шифри підстановки та шифри перестановки. У шифрі підстановки замінюється один символ у зашифрованому тексті на інший символ; у шифрі перестановки — міняються місцями позиції символів у початковому тексті [26].

2.1. ШИФРИ ПІДСТАНОВКИ

Шифр підстановки замінює один символ іншим. Якщо символи в початковому тексті — символи алфавіту, то одна літера замінюється іншою. Наприклад, можна замінити літеру *A* літерою *D,* використовуючи англійський алфавіт, а літеру *T* — літерою *Z*. Якщо символи — цифри (від *0* до *9*), то можна, наприклад, замінити *3* на *7*, а *2* на *6*. Шифри підстановки можуть бути розбиті на дві категорії: *моноалфавітної* (*одноалфавітної*) й *багатоалфавітної* підстановки.

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 літери алфавіту зручно ототожнювати з їх порядковими номерами.

2.1.1. Моноалфавітні шифри підстановки

У *моноалфавітних шифрах підстановці* літера (або символ) у початковому тексті завжди змінюється на ту саму літеру (або символ) у зашифрованому тексті незалежно від її позиції в тексті. Наприклад, якщо алгоритм визначає, що літера *A* в початковому тексті замінюється на літеру *D*, то при цьому кожна літера *A* замінюється

на літеру *D*. Іншими словами, літери в початковому тексті й зашифрованому тексті перебувають у відношенні один до одного.

Приклад 2.1. Вхідний текст "*hello*" внаслідок зашифрування перетворився на "*KHOOR*". Визначити, яким шифром забезпечувалося зашифрування.

Розв'язання

Використовуємо рядкові символи, щоб показати вхідний текст, і заголовні літери (символи верхнього регістру), щоб отримати зашифрований текст. Оскільки обидві літери *l* (ель) зашифровані як *O*, то шифр моноалфавітний.

Приклад 2.2. Вхідний текст "*hello*" внаслідок зашифрування перетворився на "*ABNZF*". Визначити, яким шифром забезпечувалося зашифрування.

Розв'язання

Шифр не є моноалфавітний, тому що кожна літера *l* зашифрована різними символами. Перша літера *l* зашифрована як *N*, друга — як *Z*.

Адитивні моноалфавітні шифри підстановки

Найпростіший моноалфавітний шифр підстановки — *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 літер (від *a* до *z*), а зашифрований текст складається з великих літер (від *A* до *Z*). Щоб забезпечити застосування математичних операцій до вхідного та зашифрованого текстів, надамо кожній літері числове значення (для нижнього й верхнього регістра), як це показано на рис. 2.1 для англійської (26 літер), на рис. 2.2 для російської (32 літери), а на рис. 2.3 — для української мови (32 літери).

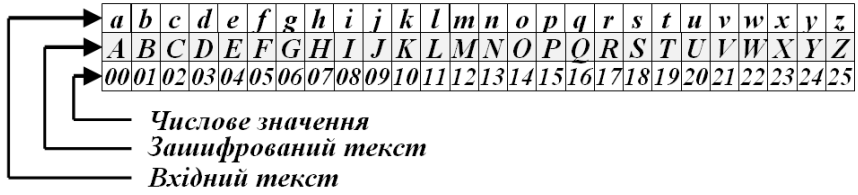


Рис. 2.1. Подання літер вхідного й зашифрованого тексту для англійської мови Z_{26}

На рис. 2.1 до кожного символу (нижній регістр або верхній регістр) поставлено у відповідність ціле число з Z_{26}. Ключ засекречування між відправником та одержувачем — також ціле число в Zn. Алгоритм зашифрування додає ключ до символу вхідного тексту; алгоритм розшифрування віднімає ключ із символу зашифрованого тексту. Усі операції проводяться в Zn. Рис. 2.4 показує процес шифрування й розшифрування повідомлень.

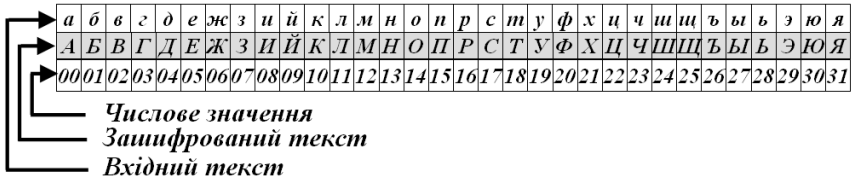


Рис. 2.2. Подання літер вхідного й зашифрованого тексту для російської мови Z_{32}

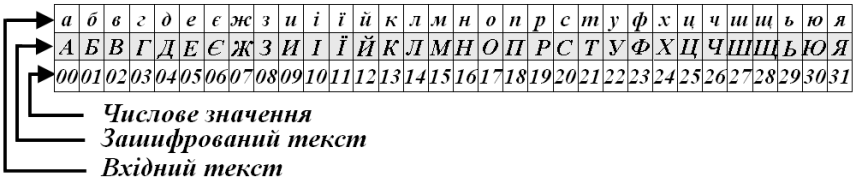


Рис. 2.3. Подання літер вхідного й зашифрованого тексту для української мови Z_{32}

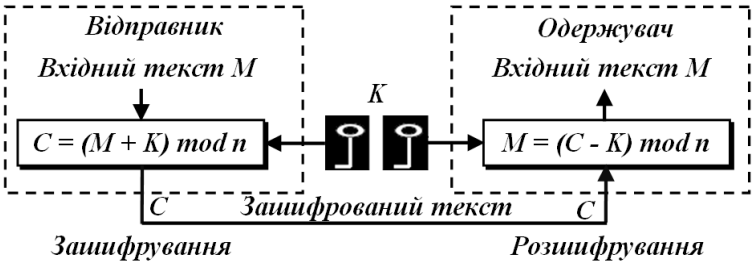


Рис. 2.4. Зашифрування й розшифрування повідомлень за допомогою адитивного шифру підстановки

Зашифрування текстових повідомлень за допомогою адитивного шифру підстановки здійснюється за допомогою виразу:

$$C = (M + K) \bmod n, \qquad (2.1)$$

де n — довжина алфавіту (для англійського алфавіту $n = 26$); M і C — вхідний і зашифрований текст відповідно; K — ключ зашифрування (розшифрування), а розшифрування

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

Можна легко показати, що зашифрування й розшифрування є інверсними один одному, тому що створений одержувачем (M') вхідний текст, той самий, що й передано відправником (M):

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

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

Розв'язання

Застосовуємо алгоритм зашифрування (2.1) до вхідного тексту, літера за літерою:

$m_1 = h \rightarrow 07;$ $\quad c_1 = (07 + 15) \bmod 26 = 22 \rightarrow W;$
$m_2 = e \rightarrow 04;$ $\quad c_2 = (04 + 15) \bmod 26 = 19 \rightarrow T;$
$m_3 = l \rightarrow 11;$ $\quad c_3 = (11 + 15) \bmod 26 = 0 \rightarrow A;$
$m_4 = l \rightarrow 11;$ $\quad c_4 = (11 + 15) \bmod 26 = 0 \rightarrow A;$
$m_5 = o \rightarrow 14;$ $\quad c_4 = (14 + 15) \bmod 26 = 3 \rightarrow D,$

як наслідок отримаємо зашифроване повідомлення "*WTAAD*".

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

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

Розв'язання

Застосовуємо алгоритм розшифрування (2.2) до зашифрованого тексту літера за літерою:

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

як наслідок отримаємо вихідне повідомлення "*hello*".

Зауважимо, 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 зашифрування може інтерпретуватися як "*клавіша зсуву літери вниз*", а алгоритм розшифрування може інтерпретуватися як "*клавіша зсуву літери вгору*". Наприклад, якщо ключ $K = 15$, то алгоритм зашифрування зсуває літеру на *15* літер вниз (до кінця алфавіту), алгоритм розшифрування зсуває літеру на *15* літер вгору (до початку алфавіту). Звичайно, коли досягається кінець чи початок алфавіту, потрібно рухатися по кільцю до початку (оголошення 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 як ключ цифру *3* ($K = 3$).

Адитивні шифри вразливі до атак тільки зашифрованого тексту, коли використовується вичерпний перебір ключів (*атака* "*грубої сили*"). Ключів адитивного шифру дуже мало — їх тільки *26* (для англійського алфавіту). Одним із недоцільних ключів є нульовий (зашифрований текст буде просто відповідати вхідному тексту). Отже, залишається тільки *25* можливих ключів. Зловмисник може легко почати атаку "грубої сили" зашифрованого тексту.

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

Розв'язання

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*":

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

Адитивні шифри також можуть бути об'єктами статистичних атак. Це особливо реально, якщо зловмисник перехопив зашифрований текст великої довжини та зможе використати отримані відомості про частоту появи символів конкретною мовою. Табл. 2.1 показує частоту появи певних літер для англійського тексту довжиною в *100* символів [23, 33, 40].

Таблиця 2.1

Частота появи букв в англійському тексті

Літера	*Частота*	*Літера*	*Частота*	*Літера*	*Частота*
E	*12,7*	*D*	*4,3*	*B*	*1,0*
T	*9,1*	*L*	*4,0*	*V*	*0,09*
A	*8,2*	*C*	*2,8*	*K*	*0,08*
O	*7,5*	*U*	*2,8*	*J*	*0,02*
I	*7,0*	*W*	*2,3*	*Q*	*0,01*
N	*6,7*	*F*	*2,2*	*X*	*0,01*
S	*6,3*	*G*	*2,0*	*Z*	*0,01*
H	*6,1*	*Y*	*1,9*		
R	*6,0*	*P*	*1,5*		

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

Найбільш використовувані групи з двома символами (*диграма* (*digrams)*) і групи з трьома символами (*триграма* (*trigrams*)) для англійського тексту показано в табл. 2.2.

Таблиця 2.2

Групи диграм і триграм, засновані на їх частоті появи в англійській мові

Диграма	*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 такий зашифрований текст:

"*XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRH*
SPPEVWMXMWASVX-LQSVILY-
VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW".

Використовуючи статистичну атаку, необхідно знайти вхідний текст.

Розв'язання

Коли зловмисник складе таблицю частоти появи літер у цьому зашифрованому тексті, він отримає: $I = 14$, $V = 13$, $S = 12$ і так далі. Найчастіший символ — *I* — має *14* поява. 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 шифрування $K = 4$, 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* алгоритм зашифрування застосовує множення вхідного тексту ключем, а алгоритм розшифрування застосовує ділення зашифрованого тексту ключем, як показано на рис. 2.5. Однак оскільки операції проводяться в Z_{26}, розшифрування тут означає множення на мультиплікативно обернене значення ключа. Зауважимо, що ключ повинен належати до набору Z_{n*}, це гарантує, що зашифрування й розшифрування будуть обернені один одному.

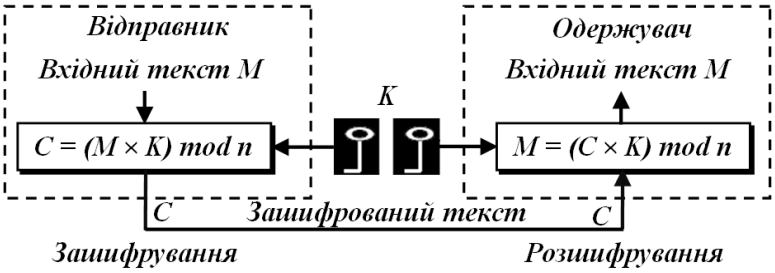


Рис. 2.5. Зашифрування й розшифрування повідомлень за допомогою мультиплікативного шифру підстановки

Зашифрування текстових повідомлень за допомогою мультиплікативного шифру підстановки здійснюється за допомогою виразу:

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

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 зашифрування (розшифрування) для мультиплікативного шифру за використання англійського алфавіту.

Розв'язання

Ключ повинен бути в 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 $K = 7$.

Розв'язання

Застосовуємо алгоритм зашифрування (2.3) до вхідного тексту символ за символом:

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

отримаємо зашифроване повідомлення "*XCZZU*".

Можна комбінувати адитивні та мультиплікативні шифри, щоб отримати те, що названо *афінним шифром* — комбінацією обох шифрів із парою ключів. Перший ключ використовується мультиплікативним шифром, другий — адитивним шифром. Рис. 2.6 показує, що афінний шифр — фактично два шифри, що застосовуються один за одним.

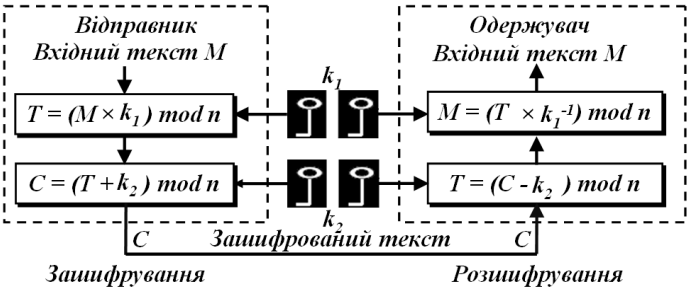


Рис. 2.6. Зашифрування й розшифрування повідомлень за допомогою афінного шифру підстановки

При використанні афінного шифру відношення між вхідним M і зашифрованим C текстом, а також ключами k_1 і k_2 визначається як

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

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.

Можна було б показати тільки одну комплексну операцію для зашифрування або розшифрування на рис. 2.6, таку, як (2.5). Однак на рис. 2.6 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 лінії і що вони використовуються у зворотному порядку під час зашифрування та розшифрування. Якщо додавання — остання дія під час зашифрування, то під час розшифрування першим повинно бути віднімання.

Приклад 2.9. Афінний шифр використовує пару ключів, з яких перший ключ з Z_{26}^*, а другий — з Z_{26}. Визначити множину ключів для афінного шифру, використовуючи англійський алфавіт.

Розв'язання

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, зашифрувати повідомлення "*hello*" з ключовою парою $k_1 = 7$ і $k_2 = 2$.

Розв'язання

Використовуємо $k_1 = 7$ для мультиплікативного шифру і $k_2 = 2$ для адитивного шифру, а також застосовуючи алгоритм зашифрування (2.5) до повідомлення, символ за символом:

$$
\begin{array}{ll}
m_1 = h \rightarrow 7; & c_1 = (7 \times 7 + 2) \bmod 26 = 25 \rightarrow Z; \\
m_2 = e \rightarrow 4; & c_2 = (4 \times 7 + 2) \bmod 26 = 4 \rightarrow E; \\
m_3 = l \rightarrow 11; & c_3 = (11 \times 7 + 2) \bmod 26 = 1 \rightarrow B; \\
m_4 = l \rightarrow 11; & c_4 = (11 \times 7 + 2) \bmod 26 = 1 \rightarrow B; \\
m_5 = o \rightarrow 14; & c_5 = (14 \times 7 + 2) \bmod 26 = 22 \rightarrow W,
\end{array}
$$

отримаємо зашифроване повідомлення "*ZEBBW*".

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

Розв'язання

Щоб знайти символи вхідного 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}, а k_1 має мультиплікативну інверсію в Z_{26}^*, вхідний текст — точно такий, що використовувався в прикладі 2.10:

$$
\begin{array}{ll}
c_1 = Z \rightarrow 25; & m_1 = ((25 + 24) \times 15) \bmod 26 = 7 \rightarrow h; \\
c_2 = E \rightarrow 4; & m_2 = ((4 + 24) \times 15) \bmod 26 = 4 \rightarrow e; \\
c_3 = B \rightarrow 1; & m_3 = ((1 + 24) \times 15) \bmod 26 = 11 \rightarrow l; \\
c_4 = B \rightarrow 1; & m_4 = ((1 + 24) \times 15) \bmod 26 = 11 \rightarrow l; \\
c_5 = W \rightarrow 22; & m_5 = ((22 + 24) \times 15) \bmod 26 = 14 \rightarrow o.
\end{array}
$$

Отже, отримаємо вихідне повідомлення "*hello*".

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

Хоча методи "грубої сили" та статистичної атаки можуть використовуватися тільки для зашифрованого тексту, використаємо

атаку на відомий вхідний текст. Припустимо, що зловмисник перехоплює такий зашифрований текст англійською мовою:

“PWUFFOGWCHFDWIWEJOUUN
JORSMDWRHVCMWJUPVCCG”.

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

Для того щоб знайти ключ, зловмисник використовує таку стратегію:

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

$m_1 = e \rightarrow 4;$ $\quad c_1 = (4 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W;$
$m_2 = t \rightarrow 19;$ $\quad c_2 = (19 \times k_1 + k_2) \bmod 26 = 2 \rightarrow C.$

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

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} \bmod 26 = \begin{pmatrix} 16 \\ 10 \end{pmatrix}.$$

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:

$m_1 = e \rightarrow 4;$ $\quad c_1 = (4 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W;$
$m_2 = t \rightarrow 19;$ $\quad c_2 = (19 \times k_1 + k_2) \bmod 26 = 5 \rightarrow F.$

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

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 5 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 5 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}.$$

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*):

$c_1 = W \rightarrow 22;$ $\quad m_1 = ((22 + 22) \times 19) \bmod 26 = 4 \rightarrow e,$
$c_2 = F \rightarrow 5;$ $\quad m_2 = ((5 + 22) \times 19) \bmod 26 = 19 \rightarrow t$

і розшифровує повідомлення

“PWUFFOGWCHFDWIWEJOUUN
JORSMDWRHVCMWJUPVCCG”.

Вхідний текст при цьому буде

“Best time of the year is spring when flower bloom”
(*Найкращий час року — весна, коли цвітуть квіти*).

Одноалфавітний шифр підстановки

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

Вдале рішення полягає в тому, щоб створити відображення кожного символу вхідного тексту у відповідний символ зашифрованого тексту. Відправник і одержувач можуть домовитися про відображення для кожної літери й записати його у вигляді таблиці. Рис. 2.7 показує приклад такого відображення для англійського алфавіту.

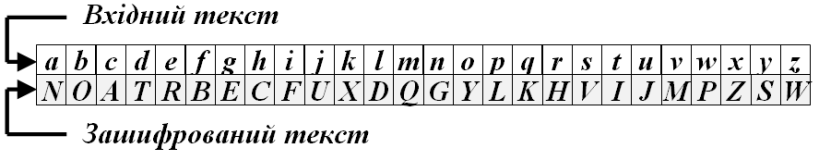


Рис. 2.7. Приклад ключа для одноалфавітного шифру підстановки для англійського алфавіту

Приклад 2.13. Використовуючи ключ, показаний на рис. 2.7, 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*).

Розв'язання

Зашифроване повідомлення буде мати вигляд:

“ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS”.

Розмір ключового простору для моноалфавітних шифрів підстановки для англійського алфавіту — кількість перестановок із n, тобто $n!$ (майже $4 \cdot 10^{26}$ за $n = 26$). Це робить атаку “грубої сили” надзвичайно складною для зловмисника, навіть при використанні потужного комп'ютера. Однак можливо 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 в зашифрованому тексті називається “*один до багатьох”*. Наприклад, “*a”* для англійської мови може бути зашифровано як “*D”* на початку тексту, але як “*N”* — у середині. Багатоалфавітні шифри мають перевагу: 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 вхідного тексту та його позиції в повідомленні. Це передбачає, що ключ повинен бути потоком підключів, в яких кожний підключ так чи інакше залежить від позиції символу вхідного тексту, який використовується для вибору підключа шифрування. Іншими словами, потрібно мати ключовий потік $k = (k_1, k_2, k_3, ...)$, в якому k_i застосовується для того, щоб зашифрувати i-й символ у вхідному тексті та створити i-й символ у зашифрованому тексті.

Автоключовий шифр підстановки

Для визначення залежності ключа від позиції використовується простий багатоалфавітний шифр, який називається *автоключовим*. У цьому шифрі ключ — потік підключів, в якому кожний підключ використовується для того, щоб зашифрувати відповідний символ у початковому тексті. Перший підключ — визначене заздалегідь значення, яке таємно погоджене відправником та одержувачем. Другий підключ — значення першого символу вхідного тексту (між *0* і *25*). Третій — *i*-те значення другого вхідного тексту й так далі. Нехай вхідний текст: $M = m_1, m_2, m_3, \ldots$. Зашифрований текст: $C = c_1, c_2, c_3, \ldots$. Ключ: $K = (k_1, k_2 = m_1, k_3 = m_2, k_4 = m_3, \ldots)$.

Зашифрування текстових повідомлень за допомогою автоключового шифру підстановки здійснюється за виразом

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

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 вхідного тексту у процесі шифрування.

Приклад 2.14. Припустимо, що відправник та одержувач погодилися використовувати автоключовий шифр із початковим ключовим значенням $k_1 = 12$. Тепер відправник хоче передати одержувачу повідомлення "*Attack is today*" ("*Атака сьогодні*").

Розв'язання

Зашифрування проводиться символ за символом. Кожний символ у початковому тексті спочатку замінюється на значення його цілого числа, як показано на рис. 2.1, перший підключ додається, щоб створити перший символ зашифрованого тексту. Інша частина ключа створюється в міру читання символів вхідного тексту. Результат зашифрування наведено в табл. 2.3.

Таблиця 2.3

Результат зашифрування для прикладу 2.14

Вхідний 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 ці три появи "*a*" у початковому тексті зашифровані по-різному. Три появи "*t*" також зашифровані по-різному.

Автоключовий шифр дійсно приховує статистику частоти окремого символу, однак залишається вразливим до атаки "грубої сили", як і адитивний шифр. Перший підключ може бути тільки одним із *25* значень (*1–25*). Існує потреба в багатоалфавітних шифрах, 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* літер 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 й чотири прості правила. Щоб скласти ключову матрицю, у першу чергу потрібно заповнити порожні комірки матриці літерами 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 алфавітом, складає матрицю *5 × 5* і є ключем шифру. За допомогою різних домовленостей про розміщення літер 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 вхідного 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 "*x*" використовується "*q*" або "_".

2. Якщо символи біграми вхідного тексту зустрічаються в одному рядку, то ці символи замінюються на символи, розташовані в найближчих стовпцях справа від відповідних символів. Якщо символ є останнім у рядку, то він замінюється на перший символ цього самого рядка.

3. Якщо символи біграми вхідного тексту зустрічаються в одному стовпці, то вони перетворюються 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 вхідного 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 вхідному 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 вхідного 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 вхідного тексту залежить від наступного або попереднього. Отже, розглядаючи

шифр Плейфера, можна сказати, що зашифрований текст — це фактично потік ключів.

Нехай вхідний 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 здійснюється за виразом

$$c_i = k_i, \tag{2.9}$$

а розшифрування

$$m_i = k_i. \tag{2.10}$$

Приклад 2.15. Зашифрувати вхідний текст "*Attack is today*" ("*Атака сьогодні*") за допомогою шифру *Плейфера* з ключовим словом "*playfair example*". У такому випадку матриця засекречування (або секретний ключ) набуде вигляду, наведено на рис. 2.8.

Секретний ключ =

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

Рис. 2.8. Приклад секретного ключа шифру Плейфера

Розв'язання

Коли згрупуємо літери по парах, то отримаємо "*at ta ck is to da y*". Для забезпечення останнього символу парою потрібно додати символ "*x*" за символом "*y*", після чого отримаємо "*at ta ck is to da yx*". Перетворюючи вхідний текст за допомогою таблиці (рис. 2.8), отримаємо:

1. Біграма "*at*" формує прямокутник, замінюємо її на "*PV*".
2. Біграма "*ta*" формує прямокутник, замінюємо її на "*VP*".
3. Біграма "*ck*" розташована в одному стовпці, замінюємо її на "*KY*".
4. Біграма "*is*" розташована в одному стовпці, замінюємо її на "*KI*".
5. Біграма "*to*" формує прямокутник, замінюємо її на "*WT*".
6. Біграма "*da*" розташована в одному стовпці, замінюємо її на "*EN*".
7. Біграма "*yx*" розташована в одному стовпці, замінюємо її на "*XG*".

Отже, зашифрований текст буде представлений у вигляді "*PVVPKYKIWIENXG*".

Цей приклад показує, що шифр Плейфера — фактично багатоалфавітний шифр: однакові символи зашифровані різними символами.

Для використання кириличного алфавіту необхідно збільшити розмір матриці до *6×6*. Використовуються *32* літери алфавіту та додатково чотири символи: “.” (крапка); “,” (кома); “–” (тире); “_” (знак підкреслення). Приклад матриць засекречування для російського та українського алфавітів (без використання ключового слова) наведено на рис. 2.9.

А	*Х*	*Б*	*М*	*Ц*	*В*
Ч	*Г*	*Н*	*Ш*	*Д*	*О*
Е	*Щ*	*,*	*Ж*	*У*	*П*
.	*З*	*Ъ*	*Р*	*И*	*Й*
С	*Ь*	*К*	*Э*	*Т*	*Л*
Ю	*Я*	*_*	*Ы*	*Ф*	*–*

а)

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

б)

Рис. 2.9. Приклад матриць засекречування для російського (а) та українського (б) алфавітів

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 символів. Однак частота появи двосимвольних комбінацій (біаграм) зберігається до деякого ступеня через вставки наповнювача так, що криптографічний аналітик для знаходження ключа може використовувати тільки атаку на зашифрований текст, засновану на випробуванні частоти появи біграм.

Шифр підстановки Віженера

Ще одним цікавим видом багатоалфавітного шифру підстановки є *шифр Віженера,* створений *Блезом де Віженером*, французьким математиком шістнадцятого століття [6, 35]. Шифр Віженера використовує різну стратегію створення потоку ключів. Потік ключів — повторення початкового потоку ключа засекречування довжини *n*.

Шифр може бути описаний таким чином: (k_1, k_2, ..., k_n) — секретний ключ, узгоджений відправником та одержувачем.

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

Зашифрування текстових повідомлень за допомогою шифру Віженера здійснюється 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 з важливих відмінностей між шифром Віженера та іншими розглянутими багатоалфавітними шифрами, є те що потік ключів шифру Віженера не залежить від символів вхідного тексту: він залежить тільки від позиції символу в початковому тексті. Іншими словами, потік ключів може бути створений без знання суті вхідного тексту.

Приклад 2.16. Зашифрувати повідомлення "*She is listening*" ("*Вона слухає*"), використовуючи ключове слово із шести символів "*PASCAL*".

Розв'язання

Початковий потік ключів — це (*15*, *0*, *18*, *2*, *0*, *11*). Потоком ключів є повторення цього початкового потоку ключів (стільки разів, скільки потрібно). Процес зашифрування надано у табл. 2.4.

Таблиця 2.4

Процес зашифрування для прикладу 2.16

Вхідний текст *M*	*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	*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*	*6*	*13*	*19*	*02*	*06*
Зашифрований текст *C*	*H*	*H*	*W*	*K*	*S*	*W*	*X*	*S*	*L*	*G*	*N*	*T*	*C*	*G*

Отже, зашифрований текст буде мати такий вигляд: "*HHWKSWXSLGNTCG"*.

Шифр Віженера може розглядатися як комбінації адитивних шифрів. Рис. 2.10 показує, що вхідний текст попереднього прикладу можна розглядати як такий, що складається з декількох частин по

шість елементів у кожному (хоча в одному не вистачило літер вхідного тексту), де кожний з елементів зашифрований окремо.

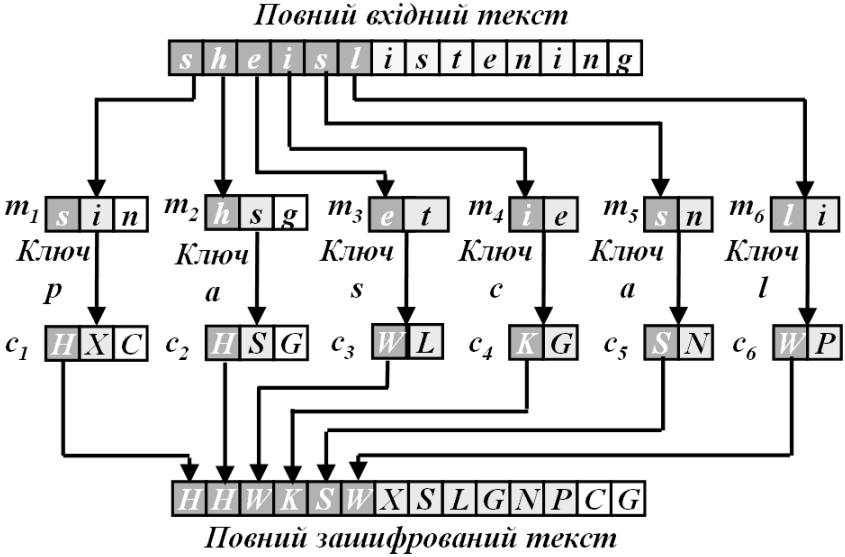


Рис. 2.10. Шифр Віженера як комбінація адитивних шифрів

Рисунок допоможе пізніше зрозуміти криптографічний аналіз шифрів Віженера. Є *n* частин вхідного тексту, кожна з яких зашифрована різним ключем, щоб розділити зашифрований текст на *m* частин.

Адитивний шифр — окремий випадок шифру Віженера, в якому *m = 1*.

Інший спосіб розгляду шифрів Віженера проводиться за допомогою відомого *списку або таблиці Віженера* (*Vigenere tableau*), що показано у табл. 2.5.

Перший рядок містить символи вхідного тексту, який буде зашифрований. Перша колонка містить стовпець символів, які використовуються ключем. Інша частина таблиці показує символи зашифрованого тексту. Щоб знайти зашифрований текст для вхідного "*she listening*", використовуючи слово "*PASCAL*" як ключ, необхідно знайти символ "*s*" з вхідного тексту в першому рядку, символ ключового слова "*P*" у першому стовпці; а на їх перетині знаходиться символ "*H*" зашифрованого тексту. 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 тексту не будуть знайдені.

Таблиця 2.5

Список *Віженера*

	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*

Щоб знайти вхідний текст для зашифрованого “*HHWKSWXSLG NTCG*”, використовуючи слово “*PASCAL*” як ключ, можна 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 вхідного тексту "*s*". Знаходимо символ "*A*" ключового слова в першому стовпці; далі, рухаючись вправо по рядку, знаходимо символ "*H*" зашифрованого тексту; на перетині стовпця, в якому знаходиться символ "*H*" і першого рядка — символ вхідного тексту "*h*". Повторюємо ті самі дії, поки всі символи вхідного тексту не будуть знайдені.

Шифр Віженера, подібно до всіх багатоалфавітних шифрів, не зберігають частоту 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 метод розглянемо нижче. У так званому тесті *Касіскі* (*Kasiski*) криптографічний аналітик у зашифрованому тексті шукає повторні сегменти принаймні з трьох символів [3, 7, 10, 19–21]. Припустимо, що знайдено два сегменти, і відстань між ними — *d*. Криптографічний аналітик припускає, що *d/m*, де *m* — довжина ключа. Якщо можна знайти більше повторних сегментів із відстанню d_1, d_2,..., d_n, тоді *gcd* (d_1, d_2,...,d_n)/*m* (позначення *gcd* для найбільшого загального дільника походить від англійських слів *greatest common divisor – найбільший спільний дільник* — і прийнято в сучасній літературі). Це припущення логічне, тому що якщо два символи однакові й $k \times m$ ($k = 1, 2,...$) — 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. Приклад 2.17 допоможе зрозуміти ці міркування.

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

Приклад 2.17. Припустимо, що зловмисник перехопив такий зашифрований текст, викладений англійською мовою:

"*LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETK – KMEVLWPCZVG'TH VTSGXQOVGCSVETQLTJSUMVWVEU – VLXEWSLGFZMVVWLGYHCUSWXQHKVGSHEEVFLCFDGV – SUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUEEHHVUC – FVGOWICQLIJSUXGLW*".

Зловмисникові необхідно дешифрувати цей зашифрований текст.

Розв'язання

Тест Касіскі на повторення сегментів у три символи приводить до результату, показаного в табл. 2.6.

Таблиця 2.6

Тест Касіскі для прикладу 2.17

Комбінація	Перша відстань	Друга відстань	Різниця
JSU	*68*	*168*	*100*
SUM	*69*	*117*	*48*
VWV	*72*	*132*	*60*
MPH	*119*	*127*	*8*

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

Таблиця 2.7

Процес дешифрування зашифрованого тексту за допомогою тесту Касіскі для прикладу 2.17

c_1	*LWGW*	*CRAO*	*KTEP*	*GTQC*	*TJVU*	*EGVG*
m_1	*jueu*	*apym*	*ircn*	*eroa*	*rhts*	*thin*
c_1	*UQGE*	*CVPR*	*PVJG*	*TJEU*	*GCJG*	

Закінчення табл. 2.7

m_1	*ytra*	*hcie*	*ixst*	*hcar*	*rehe*	
c_2	*IGGG*	*QHGW*	*GKVC*	*TSOS*	*QSWV*	*WFVY*
m_2	*usss*	*ctsl*	*swho*	*feae*	*ceih*	*cete*
c_2	*SHSV*	*FSHZ*	*HWWF*	*SOHC*	*OQSL*	
m_2	*soec*	*atnp*	*nkhe*	*rhck*	*esex*	
c_3	*OFDN*	*URWQ*	*ZKLZ*	*HGVV*	*LUVL*	*SZWH*
m_3	*lcae*	*rotn*	*whiw*	*edss*	*irsi*	*irh*
c_3	*WKHF*	*DUKD*	*HVIW*	*HUHF*	*WLUW*	
m_3	*eteh*	*retl*	*tiid*	*eatr*	*airt*	
c_4	*MEVN*	*CWIL*	*EMWV*	*VXGE*	*TMEX*	*LMLC*
m_4	*iard*	*yseh*	*aisr*	*rtca*	*piaf*	*pwte*
c_4	*XVEL*	*GMIM*	*BWHL*	*GEVV*	*ITX*	
m_4	*thec*	*arha*	*esft*	*erec*	*tpt*	

Якщо вхідний текст не є важливим, зловмисник пробує з іншим *m*. У даному прикладі вхідний текст є важливим (читаємо по стовпцях):

"*Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar chipper. It is an additive chipper with the key set to three. Each character in the plaintext is shift three characters to create ciphertext*".

Переклад цього тексту:

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

Для використання зашифрування та розшифрування повідомлень із використанням кириличного алфавіту можна скористатися табл. 2.8 (російський алфавіт) та табл. 2.9 (український алфавіт).

Таблиця 2.8

Список Віженера для російського алфавіту

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

Закінчення табл. 2.8

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

Таблиця 2.9

Список Віженера для українського алфавіту

	а	б	в	г	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	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	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	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	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
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
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	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	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	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	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	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	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	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	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	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	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	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л

Закінчення табл. 2.9

	а	б	в	г	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
Н	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	А	Б	В	Г	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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* [3, 7, 10, 19–21]. Відмінність від інших багатоалфавітних шифрів – вхідний 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 шифрами блока й шифрами потоку буде розглянуто в розд. 3 і 4.

$$\begin{pmatrix} k_{11} & k_{12} & \ldots & k_{1g} \\ k_{21} & k_{22} & \ldots & k_{2g} \\ \ldots & \ldots & \ldots & \ldots \\ k_{g1} & k_{g2} & \ldots & k_{gg} \end{pmatrix}$$

Рис. 2.11. Ключ у шифрі Хілла

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

Якщо позначити блоком g символи вхідного тексту $m_1, m_2, \ldots, m_g$, відповідні символи у блоці зашифрованого тексту будуть $c_1, c_2, \ldots, c_g$, тоді будемо мати один блок зашифрованого тексту:

$$c_1 = m_1 \cdot k_{11} + m_2 \cdot k_{21} + \ldots + m_g \cdot k_{g1};$$

$$c_2 = m_1 \cdot k_{12} + m_2 \cdot k_{22} + \ldots + m_g \cdot k_{g2};$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$c_g = m_1 \cdot k_{1g} + m_2 \cdot k_{2g} + \ldots + m_g \cdot k_{gg}.$$

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

Використання матриць дозволяє відправникові зашифрувати весь вхідний текст. У цьому випадку вхідний текст — $l \times g$ — матриця,

в якій l є номером блоків. Зашифрування текстових повідомлень за допомогою шифру Хілла здійснюється за правилом:

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

а розшифрування:

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

Приклад 2.18. Нехай вхідний текст "*cod is ready*" ("*код готовий*") необхідно зашифрувати, а потім розшифрувати зашифрований текст за допомогою шифру Хілла, використовуючи ключову матрицю K ("*GYBNQKURP*" у буквеному вигляді):

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

Розв'язання

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

Зашифрування 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. Зашифрування повідомлень за допомогою шифру *Хілла* для прикладу 2.18

Як бачимо, внаслідок зашифрування отримано зашифроване повідомлення: *20*, *11*, *05*, *20*, *10*, *16*, *24*, *04*, *05*, *24*, *23*, *20*. Це зашифроване

повідомлення відповідає "*ULFUKQYEFYXU*" з використанням англійського алфавіту.

Одержувач може розшифрувати повідомлення, використовуючи мультиплікативно 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 повідомлень 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 повідомлень за допомогою шифру Хілла для прикладу 2.18

З рис. 2.13 витікає, що внаслідок розшифрування отримано 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 вхідному тексту.

Криптографічний аналіз є складним для зашифрованого тексту шифром Хілла. По-перше, атака "грубої сили" на шифр Хілла є дуже складною через матрицю-ключ — $g \times g$. Кожний вхід може мати одне з n значень. Це означає, що розмір ключа $n^{g \times g}$. Однак не всі матриці мають мультиплікативно обернену матрицю-ключ. Тому область існування ключів все ж не така величезна. По-друге, шифри Хілла не зберігають статистику звичайного тексту. Зловмисник не може провести аналіз частоти окремих блоків з двох або трьох символів. Аналіз частоти слів розміру g міг би спрацювати, але дуже рідко вхідний текст має багато однакових рядків розміру g. Однак, зловмисник може провести атаку на шифр, використовуючи метод

знання вхідного тексту за відомих значень g і пари "*вхідний текст/зашифрований текст*", принаймні g блоків. Вони (блоки) можуть належати до того самого повідомлення або різних повідомлень, але повинні бути різними. Зловмисник може створити дві $g \times g$ матриці, M (звичайний текст) і C (зашифрований текст), в яких відповідні рядки представляють відомі пари "*вхідний 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 g пар "*вхідний текст/зашифрований текст*".

Якщо не відоме значення g, то зловмисник може спробувати різні значення за умови, що m не є дуже великим.

Приклад 2.19. Припустимо, що зловмисник знає, що $g = 3$. Він перехопив три пари блока "*вхідний текст/зашифрований текст*" (не обов'язково з того самого повідомлення), як показано на рис. 2.14.

$$\begin{array}{ccc} (05\ \ 07\ \ 10) & \longleftrightarrow & (03\ \ 06\ \ 00) \\ (13\ \ 17\ \ 07) & \longleftrightarrow & (14\ \ 16\ \ 19) \\ (00\ \ 05\ \ 04) & \longleftrightarrow & (03\ \ 17\ \ 11) \\ M & & C \end{array}$$

Рис. 2.14. Формування зашифрованого тексту для прикладу 2.19

Розв'язання

Зловмисник формує матриці M і C із пар, поданих на рис. 2.14. Оскільки в даному випадку матриця M є оберненою, то вона обертається і 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 для прикладу 2.19

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 не можна зруйнувати навіть за використання необмежених обчислювальних можливостей.

Існує єдиний такий шифр, що винайдений 1917 р. американцями *Г. Вернамом* і *Д. Моборном* [18, 40] 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 й вхідний текст, і обирається абсолютно випадково.

1949 р. було опубліковано роботу К. Шеннона [43], 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 таких правил [43]:

- ключ для шифрування вибирається випадково;
- довжина ключа повинна дорівнювати довжині тексту, що шифрують;
- 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 на зашифрований текст стає неможливою. У разі якщо відправник змінює ключ, використовуючи кожного разу іншу випадкову послідовність цілих чисел, то й інші типи атак також будуть неможливі.

Для реалізації такої системи підстановки 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 — одержувачем. 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 насправді надіслані.

Здавалося б, що завдяки такій перевазі одноразові системи слід застосовувати в усіх 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 одержувачу повідомлення заздалегідь або окремо по деякому секретному каналу. Ця вимога не є занадто обтяжливою для передачі дійсно важливих одноразових повідомлень. Однак така вимога майже нездійсненна 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 й одержувачу повідомлення. Ключова послідовність починається з вказаного місця цієї книги й використовується так само, як у системі Віженера. Іноді такий шифр називають *шифром із 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 1926 р. Г. Вернамом, співробітником фірми AT&T (США), використовує двійкове подання символів вхідних даних.

Кожний символ вхідного 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 п'ятибітових ключів *{* k_0, k_1, k_2, *... }* заздалегідь записувалася на паперовій стрічці.

Схему передачі повідомлення з використанням шифрування методом Вернама показано на рис. 2.16.

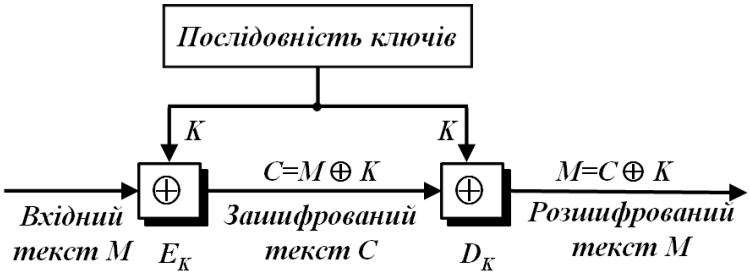


Рис. 2.16. Схема зашифрування й розшифрування повідомлень методом Вернама

Зашифрування вхідних даних, попередньо перетворених у послідовність двійкових символів *n*, здійснюється шляхом додавання за модулем *2* символів *n* із послідовністю ключів *k*.

Символи зашифрованих даних перетворюється таким чином:

$$c_i = m_i \oplus k_i,\ i = 1, 2, 3, \ldots, n. \qquad (2.15)$$

Розшифрування полягає в додаванні за модулем *2* символів зашифрованих даних c_i з тією самою послідовністю ключів k_i:

$$m_i = c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i,\ i = 1, 2, 3, \ldots, n. \qquad (2.16)$$

При цьому послідовності ключів, використані під час зашифрування й розшифрування, компенсують один одного (при додаванні за модулем *2*), і внаслідок цього відновлюються символи вхідних даних m_i.

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

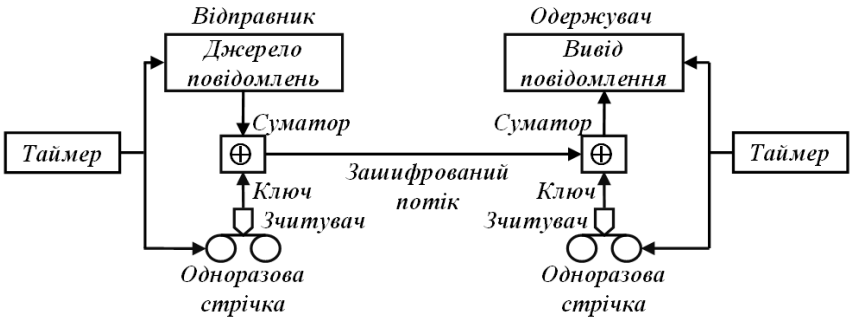


Рис. 2.17. Система Вернама, яка використовує закільцьовані стрічки

У реальних системах спочатку готують дві однакові стрічки з випадковими цифрами ключа. Одна залишається у відправника, а інша — передається "*неперехоплюючим*" способом, наприклад, кур'єром з охороною, законному одержувачу. Коли відправник хоче передати повідомлення, він спочатку перетворює його у двійкову форму й поміщає в пристрій, який до кожної цифри повідомлення додає за модулем *2* цифри, визначені в ключовій стрічці. На приймаючій стороні зашифроване повідомлення записується та пропускається через машину, схожу на пристрій, який використовується для зашифрування. Цей пристрій до кожної двійкової цифри повідомлення додає (віднімає, оскільки додавання й віднімання за модулем *2* еквівалентні) за модулем *2* цифри, визначені в ключовій стрічці, отримуючи у такий спосіб відкритий текст. При цьому ключова стрічка повинна просуватися абсолютно синхронно із своїм дублікатом, що використовується для зашифрування.

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. Зашифрування 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 цифр із декількох коротких потоків відповідно до деякого алгоритму. Одержувача зашифрованого повідомлення при цьому необхідно забезпечити точно таким самим генератором, як і у відправника. Але такі алгоритми додають регулярності в шифротекст, виявлення яких 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 складаються для отримання "*змішаного*" потоку (рис. 2.18).

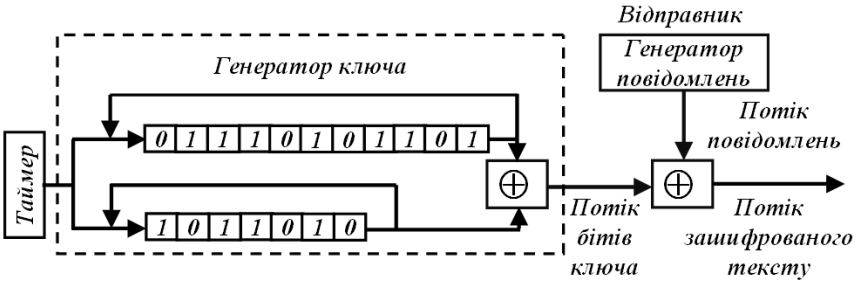


Рис. 2.18. Система Вернама, яка використовує "змішаний" потік

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 одержувачу тієї самої послідовності ключів як у відправника, або з необхідністю безпечного зберігання ідентичних послідовностей ключів у відправника й одержувача. Ці недоліки системи шифрування Вернама усуваються при зашифруванні методом гамування.

Шифрування методом гамування

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

Процедуру накладення гами на вхідні дані можна здійснити двома способами.

У першому способі символи вхідного тексту й гами замінюються цифровими еквівалентами, які потім складаються за модулем n, де n — кількість символів в алфавіті, тобто:

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

де c_i, m_i, g_i — символи відповідно зашифрованого тексту, вхідного тексту й гами.

Приклад 2.20. Нехай відкритий текст "*ГАМБИТ*", цифровий еквівалент якого відповідно до рис. 2.2 відповідає: *03 00 12 01 08 18*. Гама: "*МОДЕЛЬ*" — *12 14 04 05 11 28*. Для російського алфавіту $n = 32$.

Розв'язання

Для зашифрування відкритого тексту скористаємося співвідношенням (2.17) і отримаємо:

$$c_1 = (03 + 12) \bmod 32 = 15; \quad c_2 = (00 + 14) \bmod 32 = 14;$$

$$c_3 = (12 + 04) \bmod 32 = 16; \quad c_4 = (01 + 05) \bmod 32 = 06;$$

$$c_5 = (08 + 11) \bmod 32 = 19; \quad c_6 = (18 + 28) \bmod 32 = 14.$$

Отже, вийшов зашифрований текст: *15 14 16 06 19 14*, який з урахуванням рис. 2.2 представляється у вигляді: "*ПОРЖУО*".

За другого способу символи вхідного тексту й гами представляються у вигляді двійкового коду, а потім відповідні розряди складаються за модулем *2*:

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

де $\oplus$ — операція підсумовування за модулем *2*; c_{ij}, m_{ij} і g_{ij} — двійкові символи зашифрованого тексту, вхідного тексту й гами відповідно; n – кількість символів даних, які шифруються (кількість символів зашифрованих даних; кількість символів гами); l — кількість двійкових бітів вхідних символів, гами й зашифрованих символів.

Приклад 2.21. Нехай відкритий текст і гама як у вищенаведеному прикладі.

Розв'язання

Для зашифрування скористаємося співвідношенням (2.18):

	03	*00*	*12*	*01*	*08*	*18*
	00011	*00000*	*01100*	*00001*	*01000*	*10010*
⊕						
	01100	*01110*	*00100*	*00101*	*01011*	*11100*
	01111	*01110*	*01000*	*00100*	*00011*	*01110*
	15	*14*	*08*	*04*	*03*	*14*

Маємо зашифроване повідомлення: *15 14 08 04 03 14*, яке відповідає "*ПОИДГО*".

Замість складання за модулем *2* при гамуванні можна використовувати інші логічні операції, наприклад, перетворення за правилом логічної еквівалентності або логічної нееквівалентності. Така заміна рівнозначна введенню ще одного ключа, яким є вибір правила формування символів зашифрованого повідомлення із символів вхідного 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 шифротекста й гами оберненої операції:

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

або

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

Приклад 2.22. Розшифруємо шифротекст із попереднього прикладу, зашифрованого двома способами.

Розв'язання

Для першого способу:

$$m_1 = (15 - 12) \bmod 32 = 03; \quad m_2 = (14 - 14) \bmod 32 = 00;$$

$$m_3 = (16 - 04) \bmod 32 = 12; \quad m_4 = (06 - 05) \bmod 32 = 01;$$

$$m_5 = (19 - 11) \bmod 32 = 08; \quad m_6 = (14 - 28) \bmod 32 = 18.$$

Отже, вийшов відкритий текст: *03 00 12 01 08 18*, який відповідає повідомленню "*ГАМБИТ*".

Для другого способу:

	15	*14*	*08*	*04*	*03*	*14*
	01111	*01110*	*01000*	*00100*	*00011*	*01110*
⊕						
	01100	*01110*	*00100*	*00101*	*01011*	*11100*
	00011	*00000*	*01100*	*00001*	*01000*	*10010*
	03	*00*	*12*	*01*	*08*	*18*

Як бачимо, вийшов відкритий текст: *03 00 12 01 08 18*, який також відповідає "*ГАМБИТ*".

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

Розділяють два різновиди гамування — з кінцевою й нескінченною гамою. За кращих статистичних властивостей гами стійкість шифрування визначається лише довжиною її періоду. При цьому якщо довжина періоду гами перевищує довжину шифрованого 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.

Як нескінченна гама 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 розроблено алгоритми роботи таких датчиків, які забезпечують необхідні характеристики.

Роторний шифр підстановки

Роторний шифр повертається до ідеї моноалфавітної підстановки, але змінює принцип відображення вхідного тексту в символи зашифрованого тексту для кожного символу вхідного тексту. Рис. 2.19 показує спрощений приклад роторного шифру.

Ротор, показаний на рис. 2.19, застосовано лише для *6* літер, але реальні ротори використовують *26* літер англійського алфавіту. Ротор постійно пов'язує символи вхідного й зашифрованого текстів,

але підключення забезпечується щітками. Зауважимо, що з'єднання символів вхідного й зашифрованого текстів показано так, якби ротор був прозорий і можна було побачити внутрішню частину.

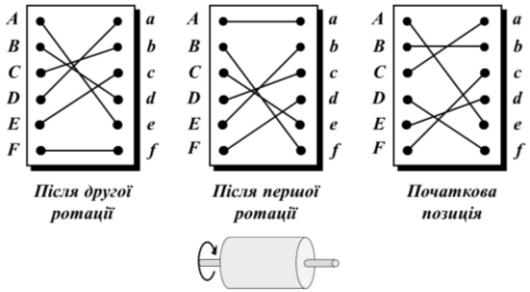


Рис. 2.19. Роторний шифр

Початкова установка (позиція) ротора — ключ засекречування між відправником і одержувачем — це зашифрований перший символ вхідного 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.

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

Роторний шифр є стійким до атаки "грубої сили", як моноалфавітний шифр підстановки, тому що зловмисник повинен знайти першу множину відображень серед можливих *n!*. Роторний шифр є набагато стійкішим до статистичної атаки, ніж моноалфавітний шифр підстановки, тому що в ньому не зберігається частота використання літер.

Роторну машину "*Енігма*" спочатку винайшли в Сербії, але фахівці німецької армії змінили її та інтенсивно використовували під час Другої світової війни [1, 24, 39, 44, 45].. Машина базувалася на принципі шифрів ротора. Рис. 2.20 показує спрощену схему будови машини.

Нижче перераховано головні компоненти машини:

1. Клавіатура з *26* ключами, використовуваними для того, щоб вводити вхідний текст під час зашифрування, і для того, щоб вводити зашифрований текст під час розшифрування.

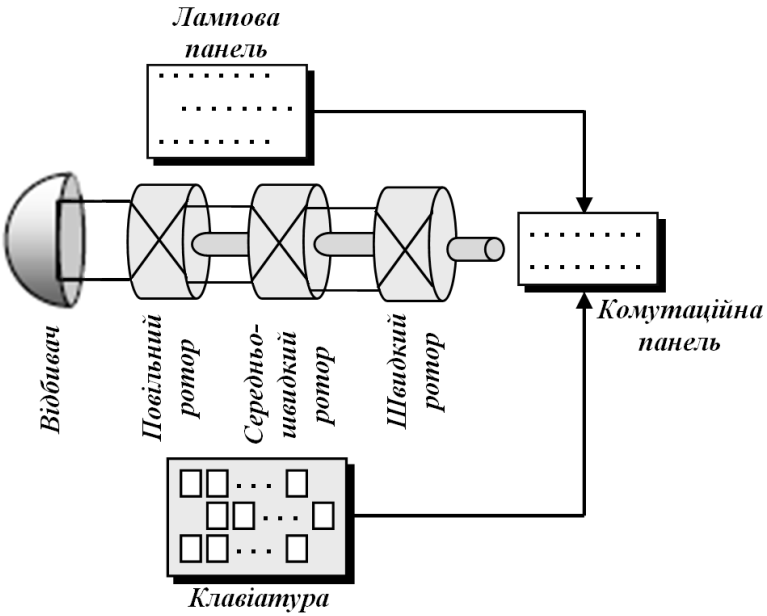


Рис. 2.20. Наближена будова машини "*Енігма*"

2. Лампова панель з *26* лампами, яка показує символи зашифрованого тексту під час зашифрування та символи вхідного тексту під час розшифрування.

3. Комутаційна панель з *26* штепселями, вручну підключеними тринадцятьма проводами. Конфігурація змінюється кожного дня, щоб забезпечити різне скремблювання.

4. Три змонтовані ротори, такі самі як розглянуті в попередній секції. Ці три ротори вибираються щодня з п'яти доступних роторів. Швидкий ротор обертається на *1/26* повороту для кожного символу, уведеного за допомогою клавіатури. Середній ротор робить *1/26* повороту для кожного повного повороту швидкого ротора. Повільний ротор робить *1/26* повороту для кожного закінченого повороту середнього ротора.

5. Відбивач, який є постійним і попередньо змонтованим.

Щоб використовувати машину "*Енігму*", було видано кодову книгу, яка протягом кожного дня дає кілька параметрів настройки, включаючи:

а) три ротори, які повинні бути вибрані з п'яти доступних;

б) порядок, в якому ці ротори повинні бути встановлені;

в) параметри установок для комутаційної панелі;

г) код з трьома літерами дня.

Щоб зашифрувати повідомлення, оператор повинен послідовно зробити кроки, перераховані нижче:

1. Установити стартову позицію роторів згідно з кодом дня. Наприклад, якщо код був "*HUA*", ротори мають бути ініційовані на "*H*", "*U*" і "*A*" відповідно.

2. Вибрати випадковий код із трьома літерами, наприклад "*ACF*". Зашифрувати текст "*ACFACF*" (повторний код), використовуючи початкову установку роторів кроку 1. Наприклад, припустимо, що зашифрований код — "*OPNABT*".

3. Установити стартові позиції роторів до "*OPN*" (половина зашифрованого коду).

4. Додати зашифрованих шість літер, отриманих на кроці 2 ("*OPNABT*"), у кінець до початкового повідомлення.

5. Зашифрувати повідомлення, включаючи код із шістьма літерами. Передати зашифроване повідомлення.

Щоб розшифрувати повідомлення, оператор повинен зробити такі кроки:

1. Отримати повідомлення й відокремити перші шість літер.

2. Встановити стартову позицію роторів згідно з кодом дня.

3. Розшифрувати перших шість літер, використовуючи початкову установку кроку 2.

4. Установити позиції роторів на першу половину розшифрованого коду.

5. Розшифрувати повідомлення (без перших шести літер).

Відомо, що "*Енігму*" під час війни було зламано, хоча німецька армія й інша частина світу не знала про цей факт ще кілька десятиліть після того [24, 45]. Хоча німецька мова є надто складною, союзники, так чи інакше, отримали деякі копії машин. Наступним кроком був пошук параметрів установки для кожного дня й коду, переданого

для ініціалізації роторів для кожного повідомлення. Винахід першого комп'ютера допоміг союзникам подолати й ці труднощі.

Шифрування методом розв'язання задачі про укладання рюкзака

Першим алгоритмом для узагальненого шифрування з відкритим ключем став *алгоритм рюкзака*, розроблений *Ральфом Мерклем* і *Мартіном Хеллманом* [14, 23, 33]. Безпека алгоритму рюкзака спирається на проблему рюкзака. Проблема рюкзака нескладна. Дано набір предметів різної маси. Чи можна покласти деякі із цих предметів до рюкзака так, щоб маса рюкзака стала дорівнювати певному значенню? Або більш формально, дано набір значень m_1, m_2, ..., m_n і сума C. Обчислити значення b_i, такі що:

$$C = b_1 \cdot m_1 + b_2 \cdot m_2 + \ldots + b_n \cdot m_n,$$

де b_i може бути нулем або одиницею. Одиниця показує, що предмет кладуть до рюкзака, а нуль — що не кладуть. У загальному випадку час, необхідний для вирішення цієї проблеми із зростанням кількості предметів у наборі, зростає експоненціально.

В основі алгоритму рюкзака Меркла–Хеллмана лежить ідея шифрувати повідомлення для вирішення набору проблем рюкзака. Предмети з набору вибираються за допомогою блока відкритого тексту, за довжиною однакової кількості предметів у наборі (біти відкритого тексту відповідають значенням b), а шифротекст є отриманою сумою.

Завдання про укладання рюкзака формулюються в такий спосіб: задано вектор:

$$E = \left| e_к, e_{к-1}, \ldots, e_2, e_1 \right|,$$

який надалі буде відігравати роль ключа. Як правило, елементами цього вектора є прості числа. Для однозначного зашифрування та розшифрування елементи вектора E обираються за правилом [5, 14]:

$$e_i > \sum_{j=1}^{i-1} e_j, \quad i = 2, 3, \ldots, k .$$

Кожний символ відкритого повідомлення m_i, що передається, представлено послідовністю з k бітів:

$$m_i = \left| m_{i,k},\ m_{i,k-1}, ...,\ p_{i,1} \right|^T,\ m_{i,j} \in \{0,1\},$$

$$j = 1, 2, ...,\ k, i = 1, 2, ..., n.$$

Як результат відкрите повідомлення являє собою матрицю

$$M = \begin{vmatrix} m_{1,k} & m_{2,k} & \dots & m_{n,k} \\ m_{1,k-1} & m_{2,k-1} & \dots & m_{n,k-1} \\ \dots & \dots & \dots & \dots \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{vmatrix},$$

де n — довжина відкритого повідомлення.

Символи зашифрованих даних C виходять як добуток матриці E та вектора-стовпця M:

$$C = E \times M. \tag{2.19}$$

Приклад 2.23. Зашифрувати відкрите повідомлення: "*НАКАЗ*" (*13, 00, 10, 00, 07*) із використанням ключа у вигляді

$$E = \left| 29 \quad 13 \quad 7 \quad 3 \quad 2 \right|.$$

Елементами вектора-рядка є прості числа.

Розв'язання

Запишемо символи відкритого повідомлення п'ятирозрядним двійковим кодом:

Н	*А*	*К*	*А*	*З*
13	*00*	*10*	*00*	*07*
01101	*00000*	*01010*	*00000*	*00111*

.

Сформуємо матрицю повідомлення:

$$M = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{vmatrix}.$$

Зробимо відповідні до (2.19) операції:

$$C = \left| 29 \quad 13 \quad 7 \quad 3 \quad 2 \right| \cdot \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{vmatrix} = \left| 22 \quad 00 \quad 16 \quad 00 \quad 12 \right|.$$

Отже, зашифроване повідомлення набуде вигляду: *22*, *00*, *16*, *00*, *12* (“*ЦАРАМ”*).

Приклад 2.24. Розшифруємо це зашифроване повідомлення.

Розв'язання

П'ятирозрядні двійкові числа відкритого повідомлення виходять із матричного рівняння (2.19).

Відкрите повідомлення можна одержати, розв'язавши шість рівнянь:

$$C = \left| 29 \quad 13 \quad 07 \quad 03 \quad 02 \right| \cdot \begin{vmatrix} m_{1,5} & m_{2,5} & m_{3,5} & m_{4,5} & m_{5,5} \\ m_{1,4} & m_{2,4} & m_{3,4} & m_{4,4} & m_{5,4} \\ m_{1,3} & m_{2,3} & m_{3,3} & m_{4,3} & m_{5,3} \\ m_{1,2} & m_{2,2} & m_{3,2} & m_{4,2} & m_{5,2} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} & m_{5,1} \end{vmatrix} =$$

$$= \begin{vmatrix} 29 \cdot m_{1,5} + 13 \cdot m_{1,4} + 7 \cdot m_{1,3} + 3 \cdot m_{1,2} + 2 \cdot m_{1,1} \\ 29 \cdot m_{2,5} + 13 \cdot m_{2,4} + 7 \cdot m_{2,3} + 3 \cdot m_{2,2} + 2 \cdot m_{2,1} \\ 29 \cdot m_{3,5} + 13 \cdot m_{3,4} + 7 \cdot m_{3,3} + 3 \cdot m_{3,2} + 2 \cdot m_{3,1} \\ 29 \cdot m_{4,5} + 13 \cdot m_{4,4} + 7 \cdot m_{4,3} + 3 \cdot m_{4,2} + 2 \cdot m_{4,1} \\ 29 \cdot m_{5,5} + 13 \cdot m_{5,4} + 7 \cdot m_{5,3} + 3 \cdot m_{5,2} + 2 \cdot m_{5,1} \end{vmatrix} = \begin{vmatrix} 22 \\ 00 \\ 16 \\ 00 \\ 12 \end{vmatrix}.$$

Перший елемент повідомлення (п'ятирозрядне двійкове число) отримаємо, розв'язавши перше скалярне рівняння.

Перше скалярне рівняння

$$29 \cdot m_{1,5} + 13 \cdot m_{1,4} + 7 \cdot m_{1,3} + 3 \cdot m_{1,2} + 2 \cdot m_{1,1} = 22.$$

Це скалярне рівняння буде справедливим тільки в тому випадку, якщо $m_{1,5} = 0$, $m_{1,4} = 1$, $m_{1,3} = 1$, $m_{1,2} = 0$ і $m_{1,1} = 1$.

Це двійковий код числа – $13_{10} = 01101_2$.

Друге скалярне рівняння:

$$29 \cdot m_{2,5} + 13 \cdot m_{2,4} + 7 \cdot m_{2,3} + 3 \cdot m_{2,2} + 2 \cdot m_{2,1} = 00.$$

Дане скалярне рівняння буде справедливим тільки в тому випадку, якщо $m_{2,5} = 0$, $m_{2,4} = 0$, $m_{2,3} = 0$, $m_{2,2} = 0$ і $m_{2,1} = 0$.

Це двійковий код числа – $00_{10} = 00000_2$.

Трете скалярне рівняння:

$$29 \cdot m_{3,5} + 13 \cdot m_{3,4} + 7 \cdot m_{3,3} + 3 \cdot m_{3,2} + 2 \cdot m_{3,1} = 16.$$

Це скалярне рівняння буде справедливим тільки в тому випадку, якщо $m_{3,5} = 0$, $m_{3,4} = 1$, $m_{3,3} = 0$, $m_{3,2} = 1$ і $m_{3,1} = 0$.

Це двійковий код числа – $10_{10} = 01010_2$.

Четверте скалярне рівняння:

$$29 \cdot m_{4,5} + 13 \cdot m_{4,4} + 7 \cdot m_{4,3} + 3 \cdot m_{4,2} + 2 \cdot m_{4,1} = 00.$$

Таке скалярне рівняння буде справедливим тільки в тому випадку, якщо $m_{4,5} = 0$, $m_{4,4} = 0$, $m_{4,3} = 0$, $m_{4,2} = 0$ і $m_{4,1} = 0$.

Це двійковий код числа – $00_{10} = 00000_2$.

П'яте скалярне рівняння:

$$29 \cdot m_{5,5} + 13 \cdot m_{5,4} + 7 \cdot m_{5,3} + 3 \cdot m_{5,2} + 2 \cdot m_{5,1} = 12.$$

Це скалярне рівняння буде справедливим тільки в тому випадку, якщо $m_{5,5} = 0$, $m_{5,4} = 0$, $m_{5,3} = 1$, $m_{5,2} = 1$ і $m_{5,1} = 1$.

Це двійковий код числа – $7_{10} = 00111_2$.

Отже, маємо відкрите повідомлення

01101	*00000*	*01010*	*00000*	*00111*
13	*00*	*10*	*00*	*07*

.

З отриманого виходить, що відкрите повідомлення: "*НАКАЗ*".

Було запропоновано велику кількість варіантів реалізації цього алгоритму, але майже всі було зламано. Та й ті, що залишилися, вимагають для зламу набагато менших зусиль, ніж деякі інші алгоритми асиметричної криптографії.

2.2. ШИФРИ ПЕРЕСТАНОВКИ

Шифр перестановки не замінює одним символом інший, замість цього він змінює місце розташування символів. Символ у першій позиції вхідного тексту може з'явитися в десятій позиції зашифрованого тексту. Символ, який знаходиться у восьмій позиції вхідного тексту, може з'явитися в першій позиції зашифрованого тексту. Іншими словами, шифр перестановки ставить (переміщає) в іншому порядку символи.

2.2.1. Шифри перестановки без використання ключа

Прості шифри перестановки, які застосовувалися в минулому, не використовували ключ. Є два методи 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 2.25. Вдалий приклад шифру без використання ключа — *шифр огорожі* (*rail fence cipher*). У цьому шифрі вхідний 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 біля парку*"), відправник пише одержувачу (рис. 2.21).

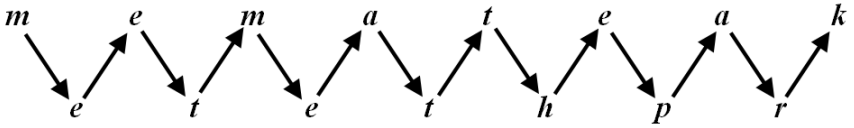


Рис. 2.21. Приклад шифрування повідомлення "*Meet me at the park*" за допомогою шифра огорожі

Відправник створює зашифрований текст "*MEMATEAKETET HPR*", посилаючи перший рядок, супроводжуваний другим рядком. Одержувач 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 зиґзаґу. Оскільки немає ніякого ключа і встановлено два рядки, криптографічний аналіз зашифрованого тексту був би дуже простий для зловмисника. Усе, що він повинен знати, що використовується шифр огорожі.

Приклад 2.26. Відправник і одержувач можуть домовитися про кількість стовпців і використовувати другий метод. Відправник пише той самий вхідний текст, рядок за рядком, у таблиці 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 зашифрування повідомлення "*Meet me at the park*" за допомогою таблиці

Відправник створює зашифрований текст "*MMTAEEHREAEK TTP*", передаючи символи стовпець за стовпцем. Одержувач отримує зашифрований текст і застосовує обернений процес, тобто пише отримане повідомлення стовпець за стовпцем і читає його рядок за рядком як вхідний текст. Зловмисник може легко розшифрувати повідомлення, якщо він знає кількість стовпців.

Шифр у вищевказаному прикладі — реальний шифр перестановки. Далі покажемо перестановку кожної літери вхідного тексту й зашифрований текст, базуючись на номерах їх позицій:

01	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
01	*05*	*09*	*13*	*02*	*06*	*10*	*14*	*03*	*07*	*11*	*15*	*04*	*08*	*12*

.

Другий символ у вхідному тексті пересунувся на п'яту позицію в зашифрованому тексті, третій символ — на дев'яту позицію і так далі. Хоча й символи переставлено, 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*.

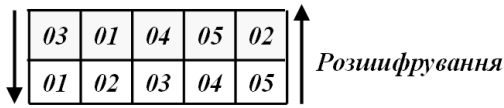
2.2.2. Шифри перестановки з використанням ключа

Безключові шифри перестановки переставляють символи, використовуючи запис вхідного 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 вхідний текст на групи заздалегідь визначеного розміру, називані блоками, а потім використовувати ключ, щоб переставити символи в кожному блоці окремо.

Приклад 2.27. Відправник повинен передати одержувачу повідомлення "*Enemy attacks tonight*" ("*Ворожі атаки сьогодні ввечері*"). Відправник і одержувач погодилися розділити текст на групи по п'ять символів і потім переставити символи в кожній групі. Нижче показано угруповання після додавання фіктивного символу в кінці, щоб зробити останню групу однаковою за розміром з іншими: "*Enemy attacks tonightz*".

Ключ, використовуваний для зашифрування й розшифрування, — ключ перестановки, який показує як переставляти символи. Для цього повідомлення приймемо, що відправник і одержувач використовували такий ключ:



Третій символ у блоці вхідного тексту стає першим символом у зашифрованому тексті в блоці, перший символ у блоці вхідного тексту стає другим символом у блоці зашифрованого тексту і так далі. Результати перестановки такі:

"*EEMYN TAACT TKONS HITZG*".

Відправник передає зашифрований текст "*EEMYN TAACT TKONS HITZG*" одержувачу. Одержувач розділяє зашифрований текст на групи по п'ять символів і, використовуючи ключ в оберненому порядку, знаходить вхідний текст.

Шифри перестановки стовпців за ключем

Сучасні шифри перестановки, щоб досягти кращого скремблювання, об'єднують два підходи. Зашифрування й розшифрування відбувається в три кроки.

Перший крок: текст пишеться в таблицю рядок за рядком.

Другий крок: робиться перестановка, змінюючи порядок проходження стовпців.

Третій крок: стовпець за стовпцем читається нова таблиця.

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

Приклад 2.28. Припустимо, що відправник знову зашифровує повідомлення "*Enemy attacks tonight*", цього разу використовуючи об'єднаний підхід. Зашифрування й розшифрування показано на рис. 2.23.

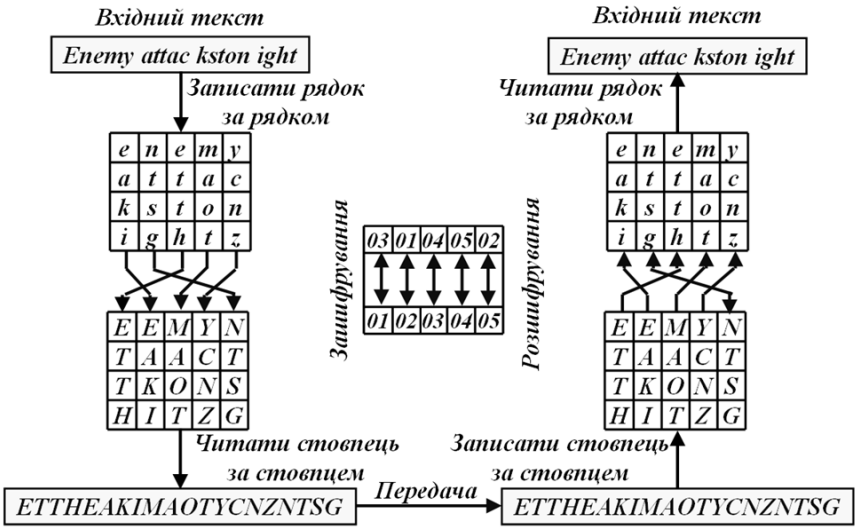


Рис. 2.23. Приклад зашифрування й розшифрування шляхом ключової перестановки

Перша таблиця, створена відправником, містить вхідний текст, записаний рядок за рядком. Стовпці переставлені з використанням того самого ключа, що й у попередньому прикладі. Зашифрований текст створений за допомогою читання другої таблиці стовпець за стовпцем. Одержувач робить ті самі три кроки, але в оберненому порядку, тобто зчитує таблицю зашифрованого тексту стовпець за стовпцем у першу таблицю, переставляє стовпці, а потім читає другу таблицю рядок за рядком.

У прикладі 2.28 єдиний ключ використовувався у двох напрямках для зміни порядку проходження стовпців — вниз для зашифрування, вверх для розшифрування. Зазвичай заведено створювати два

ключі для цього графічного представлення: один для зашифрування й один для розшифрування. Ключі накопичуються в таблицях, які мають одну адресу (вхід) для кожного стовпця. Вхід містить вхідний номер стовпця — номер стовпця пункту призначення, який вказує його положення від номера входу. Рис. 2.24 показує, як ці дві таблиці можуть бути створені за допомогою графічного представлення ключа.

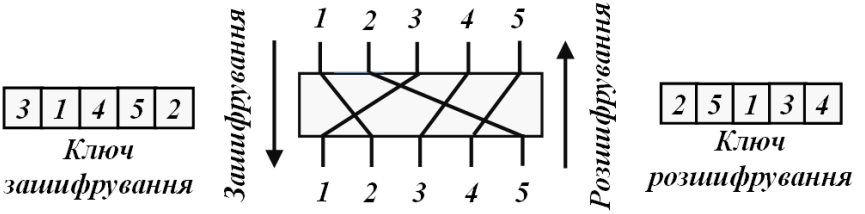


Рис. 2.24. Зашифрування/розшифрування в шифрі перестановки за допомогою двох ключів

Ключ зашифрування — (*3 1 4 5 2*). Перший вхід показує, що стовпець *3* (зміст) у джерелі стає стовпцем *1* (положення або індекс входу) в пункті призначення. Ключ розшифрування — (*2 5 1 3 4*). Перший вхід показує, що стовпець *2* в джерелі стає стовпцем *1* у пункті призначення.

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

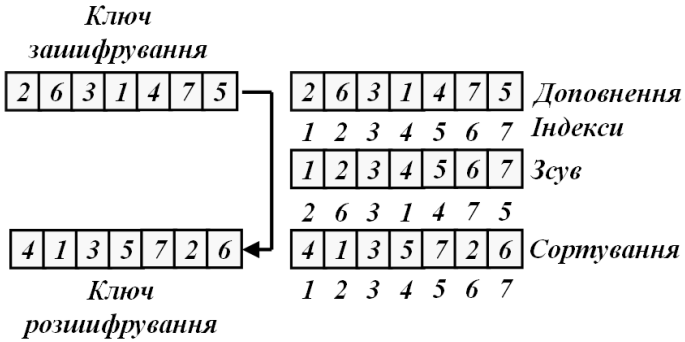


Рис. 2.25. Інверсія ключа в шифрі перестановки

Спочатку додамо індекси до таблиці ключів, потім зробимо зсув відповідно до отриманого ключа, і, нарешті, сортуємо пару згідно з індексом.

Можна використовувати матриці, щоб показати процес зашифрування/розшифрування для шифру перестановки. Оригінальний текст $M(l \times g)$ і зашифрований текст $C(l \times g)$ — матриці $l \times g$, що представляють числові значення символів; ключі $K(g \times g)$ — квадратні матриці розміру $g \times g$.

У такому випадку алгоритм зашифрування даних можна надати у вигляді

$$C(l \times g) = M(l \times g) \times K_{Ш}(g \times g),$$

а розшифрування

$$M(l \times g) = C(l \times g) \times K_{P}(g \times g),$$

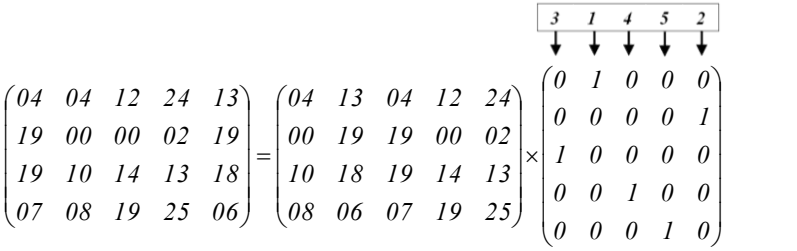
де $K_{Ш}(g \times g)$ і $K_{P}(g \times g)$ — ключові матриці, які використовуються для зашифрування й розшифрування відповідно. Ключові матриці є оберненими, тобто

$$K_{P}(g \times g) = K_{Ш}^{-1}(g \times g) \quad \text{і} \quad K_{Ш}(g \times g) = K_{P}^{-1}(g \times g).$$

У даному алгоритмі ключові матриці описують процес перестановки стовпців, тому їх частіше називають *матрицями перестановки*. У матриці перестановки кожний рядок або стовпець мають одну одиницю (*1*), а інша частина значень — нулі (*0*). Зашифрування виконується множенням матриці вхідного тексту на ключову матрицю, щоб отримати матрицю зашифрованого тексту; розшифрування виконується множенням зашифрованого тексту на обернену ключову матрицю, після чого отримуємо вхідний текст.

Цікаво, що матриця розшифрування у цьому випадку, як і завжди, — обернена матриця зашифрування. Однак немає необхідності шукати обернену матрицю зашифрування — ключова матриця зашифрування може просто бути переставлена зсувом рядків і стовпців, щоб отримати ключову матрицю розшифрування.

Приклад 2.29. Рис. 2.26 показує процес зашифрування повідомлення (див. приклад 2.28).



Зашифрований текст Початковий текст Матриця перестановки

Рис. 2.26. Пояснення процесу зашифрування даних
для прикладу 2.29 із використанням шифру перестановок

Повідомлення 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* із використанням перестановки стовпців матриці початкових даних *3*, *1*, *4*, *5*, *2*.

Множення матриці початкового тексту $M(4 \times 5)$ на ключову матрицю зашифрування $K_Ш(5 \times 5)$ дає матрицю зашифрованого тексту $C(4 \times 5)$. Матрична маніпуляція вимагає зміни символів у прикладі 2.29 до їх числових значень (від *00* до *25*). Зауважимо, що матричне множення забезпечує тільки перестановку стовпців; читання й запис у матрицю повинні бути забезпечені іншою частиною алгоритму.

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

Приклад 2.30. Рис. 2.27 показує процес розшифрування повідомлення (див. приклад 2.29) із використанням перестановки стовпців матриці початкових даних *2*, *5*, *4*, *1*, *3*, яка обернена матриці перестановки з прикладу 2.29.

Множення матриці зашифрованого тексту $C(4 \times 5)$ на ключову матрицю розшифрування $K_P(5 \times 5)$, дає матрицю початкового тексту $M(4 \times 5)$.

Отже, розшифрованний 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*".

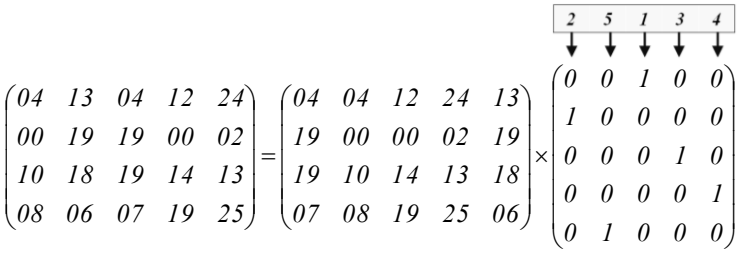


Рис. 2.27. Пояснення процесу розшифрування даних для прикладу 2.30 із використанням шифру перестановок

Криптографічний аналіз шифрів перестановки

Шифри перестановки уразливі до декількох видів атак тільки для зашифрованого тексту.

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

Зловмисник, щоб розшифрувати повідомлення, може спробувати всі можливі ключі. Однак кількість ключів може бути величезним $1! + 2! + 3! + ... + g!$, де g — довжина зашифрованого тексту. Кращий підхід полягає в тому, щоб спробувати відгадати кількість стовпців. Зловмисник знає, що кількість стовпців ділиться на g. Наприклад, якщо довжина шифру — 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 лише один стовпець і лише один рядок — малоймовірні варіанти [30, 32].

Приклад 2.31. Припустимо, що зловмисник перехопив повідомлення зашифрованого тексту "*EEMYNTAACTTKONSHITZG*". Дешифрувати це повідомлення.

Розв'язання

Довжина повідомлення $g = 20$, кількість 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 кількість стовпців — *2*, єдині дві перестановки — (*1 2*) і (*2 1*). Перше означає, що перестановки не було. Тоді зловмисник, використовуючи другу комбінацію, ділить зашифрований текст на модулі по два символи "*EE MY NT AA CT TK ON SH IT ZG*". Далі намагається переставляти кожний модуль, отримуючи текст "*ee ym nt aa tc kt no hs ti gz*", який не має сенсу.

2. Якщо кількість стовпців — *4*, тоді можливих перестановок може бути $4! = 24$. Перша перестановка (*1 2 3 4*) означає, що не було ніякої перестановки. Зловмисник повинен спробувати інші. Після випробування всіх *23* можливостей зловмисник розуміє, що будь-який початковий текст за таких перестановок не має сенсу.

3. Якщо кількість стовпців — *5*, тоді можливих перестановок може бути $5! = 120$ перестановок. Перша (*1 2 3 4 5*) означає відсутність перестановки. Зловмисник, спробувавши інші, нарешті отримує перестановку (*2 5 1 3 4*), яка дає результат — початковий текст "*enemyattack stonightz*", який має сенс після видалення фіктивної букви "*z*" і додавання пробілів ("*Enemy attacks tonight*").

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, щодо якого відомо, що кожний символ у зашифрованому тексті в прикладі 2.31 виходить із початкового тексту за таким правилом:

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 із *3*-го символу початкового тексту. *2*-й символ у зашифрованому тексті виходить з *8*-го символу початкового тексту. *20*-й символ у зашифрованому тексті виходить з *17*-го символу початкового тексту і так далі. Виходячи з вищезгаданого 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 (у цьому випадку воно дорівнює *5*), тоді можна перетворити зашифрований текст у групи по чотири символи. Перестановка груп може забезпечити ключ до знаходження початкового тексту.

Шифри з подвійною перестановкою

Шифри з подвійною перестановкою можуть ускладнити роботу криптографічного аналітика. Прикладом такого шифру було б повторення двічі алгоритму, використовуваного для зашифрування й розшифрування в прикладі 2.28. На кожному кроці може застосовуватися різний ключ, але зазвичай ключ використовується той самий.

Приклад 2.32. Повторимо приклад 2.28, за використання подвійної перестановки. Рис. 2.28 показує процес.

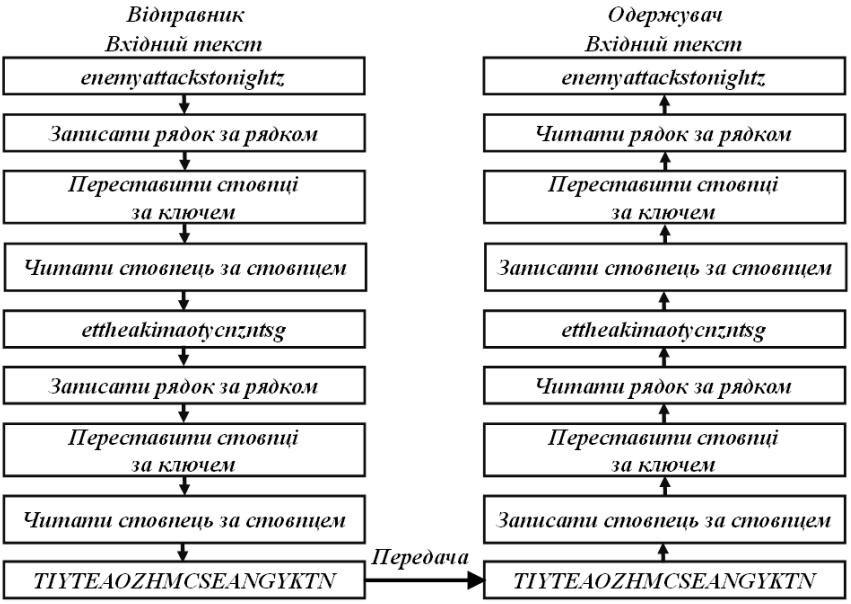


Рис. 2.28. Приклад зашифрування й розшифрування шляхом подвійної ключової перестановки

Хоча криптографічний аналітик може ще використовувати частоту появи окремого символу для статистичної атаки на зашифрований текст, атаку за зразком тепер ускладнено:

13 16 05 07 03 06 10 20 18 04

10 12 01 09 15 17 08 11 19 02.

Порівнявши наведений текст і результат прикладу 2.31, бачимо, що тепер немає повторюваних зразків. Подвійна перестановка видалила ту регулярність, яка була раніше.

Контрольні питання та завдання

1. Пояснити загальну схему симетричного шифрування. Що спільного мають усі методи шифрування із закритим ключем?
2. Дайте визначення шифрів підстановки. Сформулювати загальні принципи для методів шифрування підстановкою.
3. Пояснити сутність шифрів моноалфавітної (одноалфавітної) підстановки.
4. Пояснити сутність шифрів багатоалфавітної підстановки.
5. Пояснити сутність адитивного шифру підстановки.
6. Пояснити сутність мультиплікативного шифру підстановки.
7. Пояснити сутність афінного шифру підстановки.
8. Пояснити сутність автоключового шифру підстановки.
9. Пояснити сутність шифру підстановки Плейфера.
10. Пояснити сутність шифру підстановки Віженера.
11. До якої групи методів шифрування із закритим ключем належить метод із використанням таблиці Віженера? Які алгоритми зашифрування й розшифрування у цьому методі?
12. Пояснити сутність шифру Хілла.
13. Пояснити сутність одноразової системи шифрування.
14. Пояснити сутність шифрування методом гамування.
15. Пояснити сутність шифрування методом розв'язання задачі про укладання рюкзака.
16. Дати визначення шифру перестановки.
17. Пояснити сутність шифрів перестановки без використання та з використанням ключа.
18. Зашифрувати шифром Цезаря ($K = 3$) повідомлення українською мовою (див. рис. 2.3) "*КІМНАТА*".
19. Розшифрувати шифром Цезаря ($K = 5$) шифротекст "*ЕЄХЛПУЦ"*, при зашифруванні якого використовували українську мову (див. рис. 2.3).

20. Визначити ключі шифру Цезаря з використанням української мови (див. рис. 2.3), якщо відома пара *відкритий текст – шифротекст*: "*АПЕЛЬСИН*" – "*ЖЦЙТДШНФ*".

21. Використовуючи мультиплікативний шифр, зашифрувати повідомлення українською мовою (див. рис. 2.3) "*ВІТЧИЗНА*" з ключем $K = 3$.

22. Розшифрувати шифротекст "*АЄМТЯКИЇ*", при зашифруванні якого використовували мультиплікативний шифр ($K = 5$) та українську мову (див. рис. 2.3).

23. Використовуючи афінний шифр ($k_1 = 5$ і $k_2 = 7$) зашифрувати повідомлення українською мовою (див. рис. 2.3) "*АРГУМЕНТ*".

24. Розшифрувати шифротекст "*ЧЇГЩХЗЬЕ*", при зашифруванні якого використовували афінний шифр ($k_1 = 7$ і $k_2 = 5$) та українську мову (див. рис. 2.3).

25. Використовуючи автоключовий шифр ($k_1 = 13$), зашифрувати повідомлення українською мовою (див. рис. 2.3) "*ГОРДІСТЬ*".

26. Розшифрувати шифротекст "*КЇЦБСУРД*" при зашифруванні якого використовували автоключовий шифр ($k_1 = 11$) та українську мову (див. рис. 2.3).

27. Використовуючи шифр Плейфера (див. рис. 2.9, б), зашифрувати повідомлення українською мовою (див. рис. 2.3) "*ГАРМОНІЯ*".

28. Розшифрувати шифротекст "*ІФИ,ЧЬТБ,*" при зашифруванні якого використовували шифр *Плейфера* (див. рис. 2.9, б) та український алфавіт.

29. Використовуючи шифр Віженера та ключове слово "*КІЛОМЕТР*", зашифрувати повідомлення українською мовою (див. табл. 2.9) "*ГУСЕНИЦЯ*".

30. Розшифрувати шифротекст "*ЄВЕЬФЧШШГ*", при зашифруванні якого використовували шифр Плейфера (див. рис. 2.9), ключове слово "*ГАСТРОЛІ*" та український алфавіт (див. рис. 2.3).

31. Використовуючи шифр Хілла та ключову матрицю K з *прикладу 2.18*, зашифрувати повідомлення англійською мовою "*PERPENDICULAR*" (див. рис. 2.1).

32. Розшифрувати шифротекст "*CVJRKDXAHIBS*", при зашифруванні якого використовували шифр Хілла, ключову матрицю K^{-1} з прикладу 2.18 та англійський алфавіт (див. рис. 2.1).

Розділ 3

ПРИНЦИПИ ПОБУДОВИ СУЧАСНИХ СИМЕТРИЧНИХ КРИПТОГРАФІЧНИХ СИСТЕМ

Розглянуті вище традиційні шифри із симетричним ключем орієнтуються на символи. Із появою комп'ютерних систем необхідними стали шифри, орієнтовані на біти, тому що інформація, яку слід зашифрувати, не завжди може складатися тільки з тексту, але й також із чисел, графіків, аудіо- та відеоданих. Для зашифрування зручно перетворити ці типи даних у потік бітів і вже потім передавати зашифрований текст. Крім того, коли текст оброблено на розрядному рівні, кожний символ замінено на *8* (або *16*) бітів, це означає, що кількість символів стає у *8* (або *16*) разів більше. Саме змішування більшої кількості символів збільшує безпеку [3, 18].

3.1. СУЧАСНІ БЛОКОВІ ШИФРИ

Сучасні блокові шифри із симетричними ключами зашифровують *n*-бітовий блок вхідних даних або розшифровують *n*-бітовий блок зашифрованих даних. Алгоритми зашифрування або розшифрування використовують *k*-бітовий ключ.

Алгоритм розшифрування повинен бути інверсією алгоритму зашифрування, й обидва в роботі повинні використовувати той самий секретний ключ так, щоб одержувач мав змогу відновити повідомлення, яке передається відправником. Рис. 3.1 показує загальні процеси зашифрування й розшифрування сучасного блокового шифру.

Якщо повідомлення має розмір менше ніж *n* бітів, слід додати заповнення, щоб створити цей *n*-розрядний блок; якщо повідомлення має більше ніж *n* бітів, його слід розділити на *n*-розрядні блоки, і у разі потреби додати до останнього блока відповідне заповнення. Загальні значення для *n* зазвичай *64*, *128*, *256* або *512* бітів.

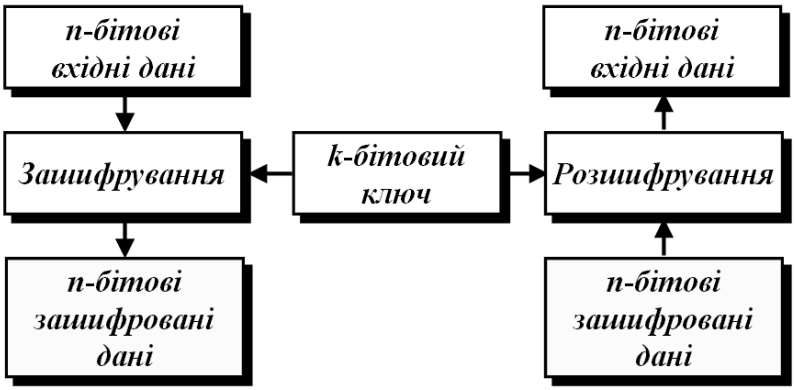


Рис. 3.1. Сучасний блоковий шифр

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

Розв'язання

Для кодування *100* символів, використовуючи *ASCII* коди по *8* бітів, довжина повідомлення буде містити *800* бітів. Початкові дані повинні ділитися без залишку на *64*. Якщо $|M|$ і $|Pad|$ — довжина повідомлення й довжина заповнення відповідно, то:

$$(|M| + |Pad|) \bmod 64 = 0.$$

Звідси виходить, що

$$|Pad| = -|M| \bmod 64 = -800 \bmod 64 = 32 \bmod 64 = 32.$$

Це означає, що до повідомлення слід додати *32* біти заповнення (наприклад нулів). Тоді початкові дані будуть складатися з *832* бітів або тринадцяти *64*-розрядних блоків. Зазначимо, що тільки останній блок містить заповнення. Шифратор використовує алгоритм зашифрування тринадцять разів, щоб створити тринадцять блоків зашифрованих даних.

3.1.1. Шифри підстановки та транспозиції

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

Якщо шифр спроектований як шифр підстановки, значення біта *1* або *0* у початкових даних можуть бути замінені або на *0*, або на *1*. Це означає, що початкові й зашифровані дані можуть мати різну кількість одиниць. Блок початкових даних на *64* біти, який містить *12* нулів і *52* одиниці, може бути представлений у зашифрованих даних, наприклад, *34* нулями і *30* одиницями. Якщо шифр спроектований як шифр перестановки (транспозиції), біти тільки змінюють порядок прямування (переміщуються), зберігаючи ту саму кількість символів у початкових і зашифрованих даних. У будь-якому випадку, кількість можливих *n*-бітових початкових або зашифрованих даних дорівнює 2^n, тому що кожний із *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 кількості 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 інші дані — нулі (*54* нулі). Скільки випробувань типу "*спроб і помилок*" повинен зробити зловмисник, щоб отримати початкові дані шляхом перехоплення зашифрованих даних у кожному з таких випадків:

а) шифр спроектований як шифр підстановки;

б) шифр спроектований як шифр транспозиції.

Розв'язання

У першому випадку (підстановка) зловмиснику невідомо скільки одиниць знаходиться в початкових даних. Тоді він повинен спробувати усі можливі 2^{64} блоки по *64* біти, щоб знайти один, який є змістовним. Перебір одного мільярда блоків за секунду міг би дати бажаний результат тільки через століття.

У другому випадку (перестановка) зловмисник знає, що у початкових даних точно є *10* одиниць, тому що транспозиція не змінює

кількості одиниць (чи нулів) у зашифрованих даних. Зловмисник може почати атаку вичерпного пошуку, використовуючи тільки ті *64*-бітові блоки, які мають точно *10* одиниць. Тоді маємо тільки

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

блоків з 2^{64} по *64* біти, які мають точно *10* одиниць. У виразі (3.1) m_1 — кількість 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 шифрі досить довгий, щоб створити відображення будь-яких можливих початкових даних у зашифруванні. 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 під час обміну між відправником і одержувачем, у шифрі використовуються також компоненти, які не залежать від ключа [22, 25].

Повнорозмірні ключові шифри

Хоча й повнорозмірні ключові шифри вже майже не використовуються, необхідно спочатку розглянути їх, щоб зробити зрозумілішим пояснення шифрів із 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 одержувачем. Для цього треба мати *n!* можливих ключів, і такий ключ повинен мати довжину $[\log_2 n!]$ бітів.

Приклад 3.3. Побудувати модель і множину таблиць перестановки для блокового шифру транспозиції на три біти, де розмір блока — $n = 3$ біти.

Розв'язання

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

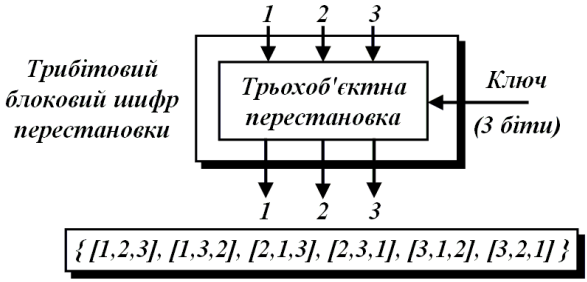


Рис. 3.2. Блоковий шифр транспозиції у вигляді перестановки

Ключ повинен бути завдовжки $[\log_2 n!] = 3$ біти. Зазначимо, що хоча ключ на три біти може вибрати $2^3 = 8$ різних відображень, ми використовуємо тільки 6 із них.

Повнорозмірні ключові блокові шифри підстановки, на перший погляд здається, не можуть бути змодельовані як перестановка. Проте можна застосувати модель перестановки також для шифру підстановки, якщо декодувати вхідну інформацію й кодувати вихідну. Таке декодування означає перетворення n-розрядного цілого числа в рядок із 2^n бітів з єдиною одиницею і $2^n–1$ нулями [2, 34]. Позиція єдиної одиниці вказує на значення цілого числа у впорядкованій послідовності позицій рядка від 0 до $2^n–1$. Оскільки нова вхідна інформація завжди має 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 блока на три біти.

Розв'язання

Три вхідні початкові блоки даних можуть бути позначені цілими числами від 0 до 7. Їх можна закодувати як рядок, що містить 8 бітів з одною 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.

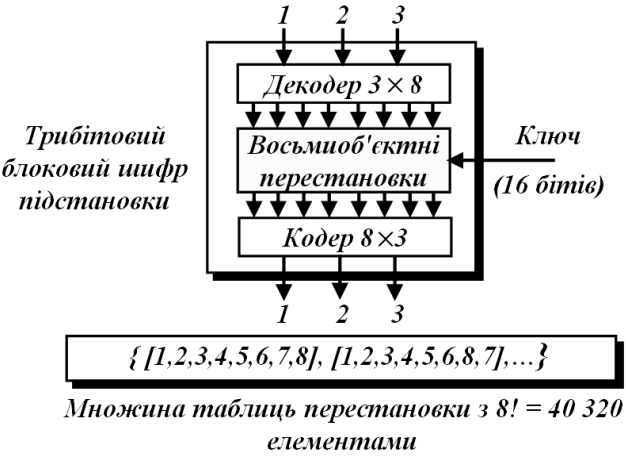


Рис. 3.3. Моделювання блокового шифру підстановки як шифру перестановки

Зазначимо, що кількість елементів у закодованій множині є набагато більшою, ніж кількість елементів у шифрі транспозиції (*8! = = 40 320*). Ключ також набагато довший *[log₂ 40 320] = 16* бітів. Хоча й ключ на *16* бітів може визначити *65 536* різних відображень, але використовуються тільки *40 320*.

Повнорозмірний ключ — це *n*-розрядний шифр транспозиції або блоковий шифр підстановки. Такі шифри можуть бути змодельовані як шифри перестановки, але розміри їх ключа різні: для шифру транспозиції — ключ завдовжки *[log₂ n!]*, для шифру підстановки — ключ завдовжки *[log₂ (2n)!]*.

Повнорозмірна ключова транспозиція або шифр підстановки/перестановки показує, що якщо зашифрування (чи розшифрування) використовує більше ніж одну будь-яку комбінацію із цих шифрів, результат еквівалентний операції групової перестановки. Відомо, що дві або більше каскадні перестановки можуть завжди бути замінені однією перестановкою. Це означає, що не потрібно 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 надто маленьким фрагментом повнорозмірного ключа. Це означає, що *DES* використовує тільки 2^{56} відображень із приблизно $2^{2^{70}}$ можливих.

Поставимо собі питання: "Чи можна встановити, що багатоступінчаста транспозиція із частковим ключем або підстановка — це група перестановки з композицією операцій?" Відповідь на це питання надзвичайно важлива, оскільки говорить про те, чи є багатоступінчаста версія із частковим шифром таким самим засобом шифрування, як і сам шифр. Цей факт дозволяє досягти більшого ступеня безпеки [19–21]..

Частковий ключовий шифр — це підгрупа відповідного розміру ключа шифру (повнорозмірного). Іншими словами, якщо повнорозмірний ключовий шифр — це група $G = <Q, O>$, де Q — множина відображень і O — композиція операцій, то шифр із ключем часткового розміру повинен представляти підгрупу $H = <U, O>$, де U — підмножина Q з тими самими операціями.

Наприклад, було доведено, що багатоступінчастий *DES* із *56*-бітовим ключем не є групою, тому що підгрупа з 2^{56} відображеннями не може бути створена з групи з $2^{64}!$ відображеннями.

Частковий ключовий шифр є група з набором операцій, якщо він є підгрупою відповідного повнорозмірного ключового шифру.

Шифри без ключа

Використання окремо шифру без ключа є фактично марним, але можливе його застосування як компонентів ключових шифрів.

Шифри транспозиції без ключа (чи із фіксованим ключем) можна розглядати як шифр транспозиції, реалізований в апаратних засобах. Фіксований ключ (єдине правило перестановки) може бути представлений таблицею в програмному забезпеченні при реалізації шифру. Далі будуть розглянуті шифри транспозиції без ключа,

названі *P-блоками перестановки*, які використовуються як стандартні блоки сучасних блокових шифрів.

Шифр підстановки без ключа (або із фіксованим ключем) розуміють як заздалегідь визначене відображення вхідної інформації у вихідну. Відображення може бути представлене таблицею, як математичною функцією, а також іншим способом. 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. Проте ці шифри зазвичай не проектуються як єдиний модуль. Щоб забезпечувати необхідні властивості сучасного блокового шифру, такі як розсіювання й перемішування інформації, цей шифр формується як комбінація модулів транспозиції (*P*-блоків перестановки), модулів підстановки (*S*-блоків заміни) та деяких інших модулів.

P-блок перестановки, подібний до традиційного шифру транспозиції символів, переміщає біти. Сучасні блокові шифри мають три типи *P*-блоків перестановки: прямі *P*-блоки; *P*-блоки розширення та *P*-блоки стискання, які показані на рис. 3.4.

На рис. 3.4 продемонстровано прямий *P*-блок *5×5* (рис. 3.4, а), *P*-блок стискання *5×3* (рис. 3.4, б) і *P*-блок розширення *3×5* (рис. 3.4, в). Розглянемо кожний із них детальніше.

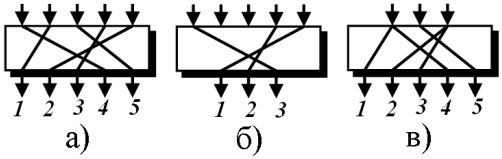


Рис. 3.4. Три типи *P*-блоків:
а) прямий *P*-блок; б) *P*-блок стискання; в) *P*-блок розширення

Прямі P-блоки

Прямий P-блок із *n* входами і *n* виходами — це перестановка з *n!* можливих відображень.

Приклад 3.5. Рис. 3.5 показує усі шість можливих відображень прямого *P*-блока *3 × 3.*

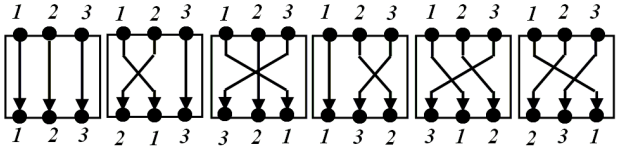


Рис. 3.5. Можливі відображення прямого *P*-блока *3 × 3*

Хоча прямий *P*-блок може використовувати ключ, щоб визначити одне з *n!* відображень, зазвичай прямі *P*-блоки — без застосування ключа, тобто відображення задане заздалегідь. Якщо прямий *P*-блок заданий заздалегідь і реалізований в апаратних засобах, або якщо має реалізацію в програмному забезпеченні, таблиці перестановок задають правило відображення. В іншому випадку входи таблиці вказують на позиції, в яких вказані позиції виходів. Табл. 3.1 показує приклад таблиці перестановок, коли *n* дорівнює *32*.

Таблиця 3.1

Приклад таблиці перестановки для прямого *P*-блока

	Номери 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 входи, які фіксують відповідність до *32* інформаційних виходів. Позиція (індекс) входу відповідає виходу. Наприклад, перший табличний вхід містить номер *26*. Це означає, що перший вихід буде відповідати *26*-му входу. Оскільки останній табличний вхід — *7*, це означає, що *32*-й вихід відповідає сьомому інформаційному входу і так далі.

Прямі *P*-блоки є оберненими. Це означає, що можна використовувати прямий *P*-бітовий блок для зашифрування, а потім для розшифрування. Проте таблиці перестановки повинні бути оберненими у відношенні один до одного.

Приклад 3.6. Необхідно скласти таблицю перестановки для прямого *P*-блока *8 × 8*, яка переміщує два середніх біти (біти *4* і *5*)

у вхідному слові до двох крайніх бітів (біти *1* і *8*) вихідного слова. Відносні позиції інших бітів не змінюються.

Розв'язання

Потрібно створити прямий *P*-блок із таблицею [*4*, *1*, *2*, *3*, *6*, *7*, *8*, *5*]. Відносні позиції бітів *1*, *2*, *3*, *6*, *7* і *8* не змінюються, але перший інформаційний вихід пов'язаний із четвертим інформаційним входом, восьмий інформаційний вихід — із п'ятим інформаційним входом. Перестановки для такого прямого *P*-блока наведено в табл. 3.2.

Таблиця 3.2

Таблиця перестановки для прямого *P*-блока для прикладу 3.6

	Номери бітів							
Вхід	*04*	*01*	*02*	*03*	*06*	*07*	*08*	*05*
Вихід	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*

P-блоки стискання

P-блок стискання — це *P*-блок із n входами і m виходами, де $m < n$. Деякі інформаційні входи блоковані й не пов'язані з виходом (див. рис. 3.4, б). *P*-блоки стискання, використовувані в сучасних блокових шифрах, зазвичай є безключовими з таблицею перестановки, яка визначає правила перестановки бітів. Необхідно враховувати, що таблиця перестановок для *P*-блока стискання має n табличних входів — від *1* до n, але в змісті кожного табличного виходу деякі з них можуть бути відсутні (ті інформаційні входи, які блоковано). Табл. 3.3 показує приклад таблиці перестановки для *P*-блока стискання 32×24, в якій входи *7*, *8*, *9*, *16*, *23*, *24* і *25* блоковані.

Таблиця 3.3

Приклад таблиці перестановки для *P*-блока стискання 32×24

	Номери 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 стискання використовуються, коли треба переставити біти та водночас зменшити кількість бітів для наступного ступеня.

Обернення *P*-блоків стискання є неможливим. У *P*-блоках стискання окремі входи в процесі зашифрування можуть бути відкинуті й при цьому алгоритм розшифрування не має ключа, щоб відновити відкинуті біти.

Рис. 3.6 демонструє випадок використання *P*-блока стискання для зашифрування й розшифрування.

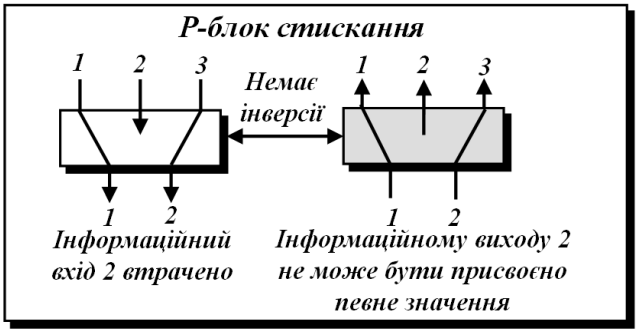


Рис. 3.6. *P*-блок стискання як безповоротний компонент

P-блоки розширення

P-блок розширення — *P*-блок із *n* входами і *m* виходами, де $m > n$. Деякі із входів пов'язані більше ніж з одним виходом (див. рис. 3.4, в). *P*-блоки розширення, які використовуються в сучасних блокових шифрах, зазвичай без ключа. Правила перестановки бітів вказуються в таблиці. Таблиця перестановки для *P*-блока розширення має *m* табличних виходів, але $m - n$ входів (входи, які пов'язані більше ніж з одним інформаційним виходом). Табл. 3.4 показує приклад таблиці перестановки для *P*-блока розширення 12×16. Зверніть увагу, що кожний з *1*, *3*, *9* і *12* входів сполучений із двома виходами.

Таблиця 3.4

Приклад таблиці перестановки для *P*-блока розширення 12×16

	Номери 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 та водночас збільшити кількість бітів для наступного каскаду шифрування.

Приклад 3.7. Рис. 3.7 показує, як треба змінити таблицю перестановки в разі одновимірної таблиці.

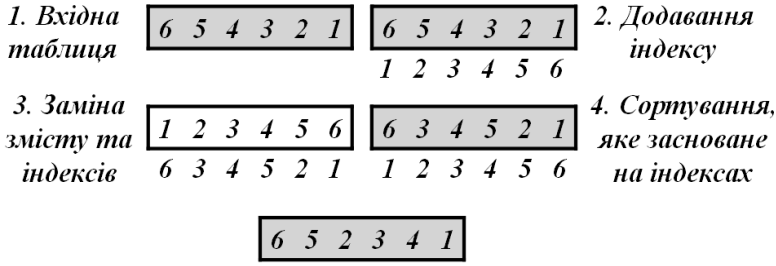


Рис. 3.7. Зміна таблиці перестановки

Обернення *P*-блоків розширення є також неможливим. У *P*-блоці розширення деякі входи в процесі зашифрування можуть бути відображені більше ніж в один вихід, але при цьому алгоритм розшифрування не має ключа і не може визначити, які з декількох входів відображені в цьому виході. Рис. 3.8 демонструє випадок використання *P*-блока розширення для зашифрування й розшифрування.

Рис. 3.6 і 3.8 також показують, що *P*-блок стискання не є оберненим шифром *P*-блока розширення і навпаки. Це означає, що якщо використовувати *P*-блок стискання для зашифрування, то не можна використовувати *P*-блок розширення для розшифрування. Проте, як показано нижче, є шифри, які застосовують *P*-блоки стискання або *P*-блоки розширення для зашифрування, але їх ефективність при цьому гірша, ніж у деяких інших способів.

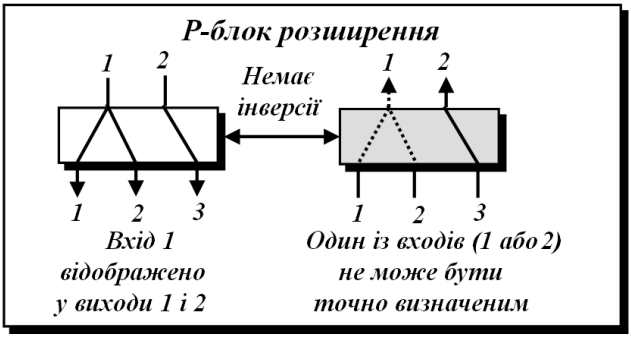


Рис. 3.8. *P*-блок розширення як безповоротний компонент

S-блоки підстановки

S-блок підстановки (заміни) являє собою мініатюрний шифр підстановки. Цей блок може мати різну кількість входів і виходів. Іншими словами, вхід до *S*-блока підстановки може бути *n*-бітовим словом, а вихід може бути *m* розрядним словом, де *m* і *n* необов'язково однакові числа. Хоча й *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.

S-блок підстановки — це $m \times n$ модуль підстановки.

У *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 й 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:

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

$$y_2 = a2,1 \cdot x_1 \oplus a_{2,2} \cdot x_2 \oplus \ldots \oplus a_{2,n} \cdot x_{n;}$$

$$\ldots$$

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

У нелінійному *S*-блоці також можна завжди задати для кожного виходу вказані вище співвідношення.

Приклад 3.8. Нехай у *S*-блоці з трьома інформаційними входами та двома інформаційними виходами маємо

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

S-блок лінійний тому, що

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

Ці співвідношення можуть бути представлені матрицями

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 111 \\ 100 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Приклад 3.9. У *S*-блоці з трьома входами та двома виходами маємо

$$y_1 = (x_1)^3 + x_2;$$

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

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

Приклад 3.10. Наступна таблиця (рис. 3.9) визначає залежність між входами/виходами для *S*-блока підстановки розміру 3×2. Крайній лівий біт входу визначає рядок; два крайні біти входу визначають стовпець. Два біти виходу — це значення на перетині секції вибраного рядка та стовпця.

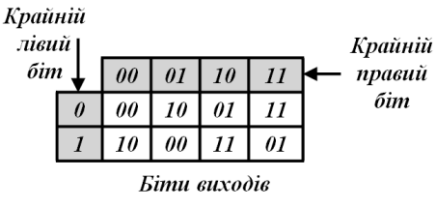


Рис. 3.9. Таблиця *S*-блока підстановки для прикладу 3.10

S-блоки — це шифри підстановки, в яких відношення між входом і виходом визначені таблицею або математичним співвідношенням. *S*-блок може або не може бути оберненим. В оберненому *S*-блоці підстановки кількість вхідних бітів повинна дорівнювати кількості бітів виходу.

Приклад 3.11. Рис. 3.10 показує приклад оберненого *S*-блока підстановки. Одна з таблиць використовується в алгоритмі зашифрування; інша таблиця — в алгоритмі розшифрування.

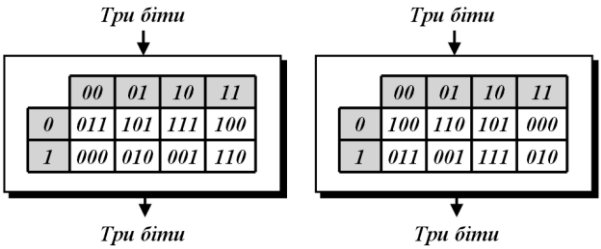


Рис. 3.10. Таблиці *S*-блока підстановки для прикладу 3.11

У кожній таблиці крайній лівий біт входу визначає рядок; наступні два біти визначають стовпець. Вихід — це значення на перетині рядка та стовпця таблиці.

Наприклад, якщо вхід до лівого блока — *001*, вихід — *101*. Вхід *101* у правій таблиці дає вихід *001*. Це показує, що ці дві таблиці дозволяють отримати обернений результат у відношенні один до одного.

Виключне або (xor)

Важливим компонентом у більшості блоків шифрування є о*перація виключне або* (*xor*). Операції складання й віднімання у полі Галуа $GF(2^n)$ виконуються за допомогою тієї самої операції, яку називають *виключним або* (*xor*).

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

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

2. *Асоціативність.* Дана властивість дозволяє використовувати більше ніж одне *виключне або* (*xor*), які можна обчислювати у будь-якому порядку:

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

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

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

4. *Існування нульового (тотожного) елемента.* Нульовий елемент для операції *виключного або* (*xor*) — це слово, яке складається з усіх нулів, або *00...0* — означає, що існує слово з нейтральними елементами, які під час операції не змінюють слово:

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

Ця властивість використовується в шифрі Фейстеля, який розглянемо далі.

5. *Існування інверсії.* У полі $GF(2^n)$ кожне слово є адитивна інверсія самого себе. Це означає, що проведення операції *виключного або* (*xor*) слова із самим собою приводить до нульового елемента:

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

Дана 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 зашифрування становлять інтерес операції з доповненням відносно операції *виключного або* (*xor*). Якщо $\bar{x}$ — доповнення "*x*", тоді два наступні співвідношення правильні:

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

Цією властивістю користуються тоді, коли говорять про безпеку деяких шифрів.

6. *Інверсія.* Інверсія компонента в шифрі має сенс, якщо компонент представляє одномісну операцію (один вхід і один вихід). Наприклад, *P*-блок без ключа або *S*-блок без ключа можуть бути оберненими, тому що вони мають один вхід і один вихід (рис. 3.11).

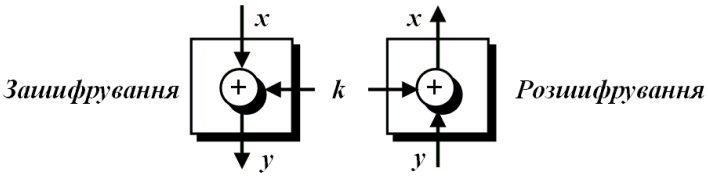


Рис. 3.11. Оборотність операції *виключного або* (*xor*)

Операція *виключне або* (*xor*) — бінарна операція. Інверсія операції *виключного або* (*xor*) може мати сенс тільки тоді, коли один із входів зафіксований (той самий за зашифрування й розшифрування). Наприклад, якщо один із входів — ключ, який зазвичай є таким самим під час зашифрування й розшифрування, тоді операція *виключного або* (*xor*) є оберненою, як показано на рис. 3.11:

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

Цією властивістю користуються тоді, коли розглядають структуру блокових шифрів.

Циклічний зсув

Наступний компонент, використовуваний у деяких сучасних блокових шифрах, — *операція циклічного зсуву*. Зсув може бути вліво або вправо. Кругова операція лівого зсуву зсуває кожний біт у n-бітовому слові на k позицій вліво; крайні ліві k-біти видаляються ліворуч і стають крайніми правими бітами. Кругова операція правого зсуву зсуває кожний біт у n-бітовом слові на k позицій вправо; крайні праві k-біти справа видаляються та стають крайніми лівими бітами. Рис. 3.12 показує як і ліві, так і праві операції у випадку, де $n = 8$ і $k = 3$.

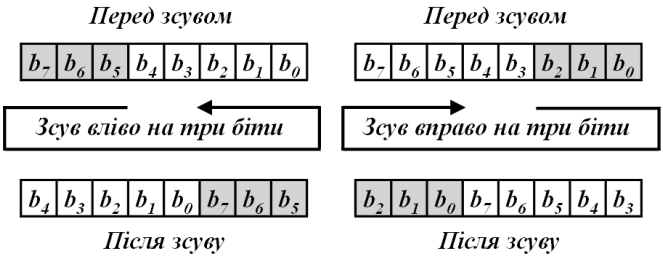


Рис. 3.12. Циклічний зсув 8-бітового слова вліво або вправо

Циклічна операція зсуву зміщує біти в слові й допомагає приховати зразки в первинному слові. Хоча кількість позицій, на які біти будуть зсунуті, може використовуватися як ключ, циклічна операція зсуву зазвичай без ключа; значення k встановлюється й задається заздалегідь.

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

Операція циклічного зсуву має дві властивості:

- *перша* — це зміщення за модулем n. Іншими словами, якщо $k = 0$ або $k = n$, зсуву не відбувається. Якщо k є більшим, ніж n, тоді вхідна інформація зсувається на $k \bmod n$ бітів;

- *друга* властивість операції 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 операція заміни можлива, тільки якщо n — парний номер. Оскільки зсув вліво $n/2$ — те саме, що і зсув $n/2$ вправо, то операція є оберненою. Операція заміни для зашифрування може бути повністю розкрита операцією заміни під час розшифрування. Рис. 3.13 ілюструє операцію заміни для слова з 8 бітів.

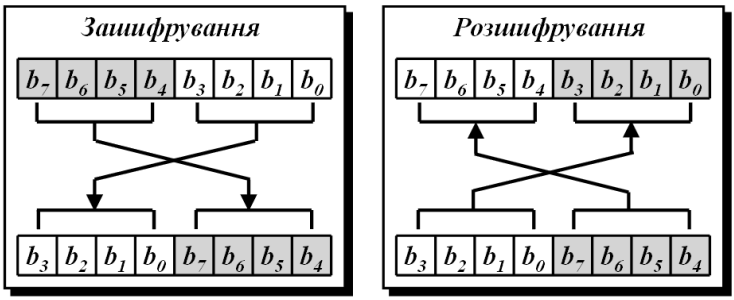


Рис. 3.13. Операція заміни у 8-бітовому слові

Розбиття й об'єднання

Наступні операції, вживані в деяких блокових шифрах, — *розбиття* й *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 урівноважити одна одну. Якщо одна використовується для зашифрування, то інша — для розшифрування. Рис. 3.14 показує ці операції для випадку $n = 8$.

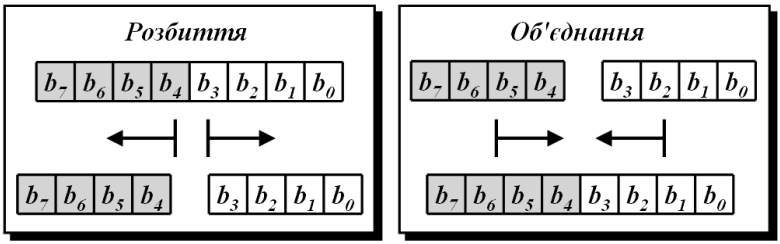


Рис. 3.14. Операції розбиття й об'єднання 8-бітового слова

3.2. СКЛАДЕНІ ШИФРИ

Шеннон увів поняття складених шифрів. *Складений шифр* — комплекс, який об'єднує підстановку, перестановку й інші компоненти, розглянуті вище.

3.2.1. Розсіювання й перемішування

Ідея Шеннона щодо складеного шифру повинна була дати можливість блоковим шифрам мати дві важливі властивості: *розсіювання* й *перемішування*.

Розсіювання повинне приховати відношення між зашифрованими й початковими даними. Це дезорієнтує зловмисника, який використовує статистику зашифрованих даних, щоб знайти початкові дані. Під розсіюванням розуміється, що кожний біт (символ) у зашифрованих даних залежить від одного або усіх бітів (символів) у початкових даних. Іншими словами, якщо один єдиний біт у початкових даних змінений, декілька або всі біти в зашифрованих даних будуть також змінені.

Ідея відносно *перемішування* — приховати залежність між зашифрованими даними та ключем. Це дезорієнтує зловмисника, який прагне використовувати зашифровані дані, щоб знайти ключ. Іншими словами, якщо один єдиний біт у ключі змінений, усі біти в зашифрованих даних будуть також змінені.

3.2.2. Раунди

Розсіювання й 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 від ключа шифру. У *N*-раундовому шифрі, щоб створити зашифровані дані, початкові дані зашифровуються *N* разів; відповідно, зашифровані дані розшифровуються *N* разів. Дані, створені на проміжних рівнях (між двома раундами), називаються *середніми даними*. Рис. 3.15. показує простий складений шифр із двома раундами.

На практиці складені шифри мають більше ніж два раунди. На рис. 3.15 у кожному раунді проводяться три перетворення:

1. *8*-бітові дані змішуються з ключем, щоб зробити біти даних рівноймовірними (приховати біти, використовуючи ключ) —

“*вибілити*” дані (*whiting*). Це зазвичай робиться за допомогою операції *виключного або* (*xor*) слова на *8* бітів із ключем на *8* бітів.

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

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

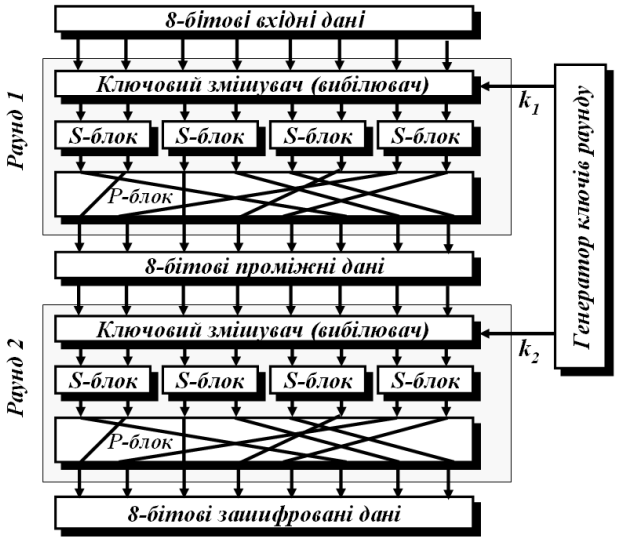


Рис. 3.15. Складений шифр, що складається з двох раундів

Спрощений складений шифр, схему якого показано на рис. 3.15, використовуючи комбінацію *S*-блоків заміни і *P*-блоків перестановки, може гарантувати розсіювання.

У першому раунді, після проведення операції *виключного або* (*xor*) з відповідними бітами ключа k_1, змінюються два біти (біти *7* і *8*) через *S*-блок *4*. Біт *7* переставлений і стає бітом *2*; біт *8* переставлений і стає бітом *4*. Після першого раунду біт *8* змінює біти *2* і *4*. У другому раунді біт *2* після проведення операції *виключного або* (*xor*) з відповідними бітами ключа k_2 змінює два біти (біти *1* і *2*) через *S*-блок *1*. Біт *1* переставлений і стає бітом *6*; біт *2* переставлений і стає бітом *1*. Біт *4* після проведення операції *виключного або* (*xor*) з відповідним бітом у k_2 змінює біти *3* і *4*. Біт *3* залишається,

біт *4* переставлений і стає бітом *7*. Після другого раунду з *8* бітів змінені біти *1*, *3*, *6* і *7*.

Проходження цих кроків в іншому напрямку (від зашифрованих до початкових даних) показує, що кожний біт у зашифрованих даних змінює початкові дані на декілька бітів.

На рис. 3.16 показано, як зміна одного єдиного біта в початкових даних викликає зміна багатьох бітів в зашифрованих даних.

Рис. 3.16 також доводить, що властивість перемішування може бути набутою за допомогою складеного шифру. Чотири біти зашифрованих даних (біти *1*, *3*, *6* і 7) перетворені за допомогою трьох бітів у ключах (біт *8* у k_1 і біти *2* і *4* у k_2). Проходження у зворотному напрямку показує, що кожний біт ключа у кожному раунді зачіпає декілька бітів в зашифрованих даних. Зв'язок між бітами зашифрованих даних і ключовими бітами показано в затемнених прямокутниках.

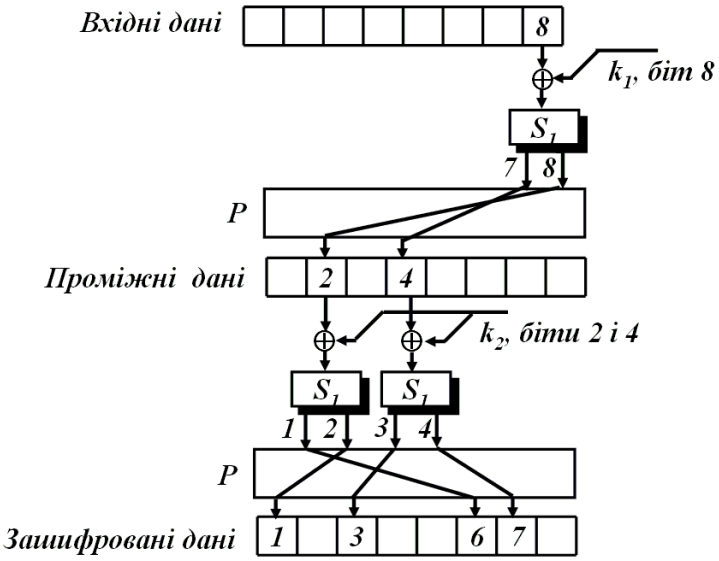


Рис. 3.16. Розсіювання й перемішування в блоковому шифрі

Щоб поліпшити розсіювання й перемішування, практичні шифри використовують великі блоки даних, більше *S*-блоків заміни і більше раундів. Очевидно, що деяке збільшення кількості раундів під час використання великої кількості *S*-блоків заміни може

створити кращий шифр, в якому зашифровані дані виглядають усе більше як випадкове *n*-бітове слово.

Отже, взаємозв'язки між зашифрованими й початковими даними будуть повністю приховані (розсіяні). Збільшення кількості раундів збільшує кількість ключів раундів, що краще приховує взаємозалежність між зашифрованими даними та ключем.

3.2.3. 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 інтелектуальний і цікавий шифр, який використовувався впродовж багатьох десятиліть. Шифр Фейстеля має три типи компонентів: самообернений, обернений і необернений.

Шифр Фейстеля містить у блоках усі необернені елементи та використовує один модуль в алгоритмах зашифрування й розшифрування. Питання в тому, як алгоритми зашифрування й розшифрування дозволяють інвертувати відкриті й закриті дані один до одного, якщо кожен містить необернений модуль. Фейстель показав, що вони можуть бути збалансовані [3, 22, 39].

Щоб краще зрозуміти шифр Фейстеля, подивимося, як можна використовувати той самий необернений компонент в алгоритмах зашифрування й розшифрування. Ефекти необерненого компонента в алгоритмі зашифрування можуть бути скасовані в алгоритмі розшифрування, якщо використовувати операцію *виключного або* (*xor*), як показано на рис. 3.17.

Під час зашифрування ключ надходить на вхід необерненої функції *f(K)*, яка є одним із доданків оператора *виключного або* (*xor*) з початковими даними. Внаслідок цього отримаємо зашифровані дані. Будемо називати комбінацію функції *f(K)* й операції *виключного або* (*xor*) – *змішувачем* (через відсутність іншої назви). Змішувач відіграє важливу роль у пізніших варіантах шифру Фейстеля.

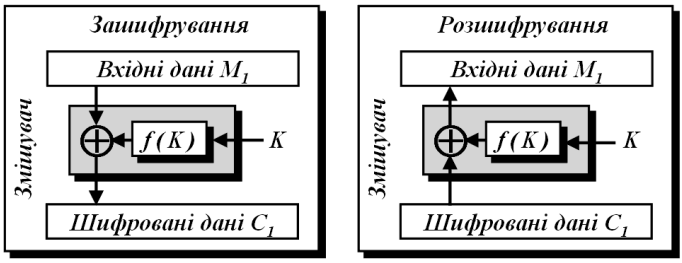


Рис. 3.17. Використання необерненого компонента
в алгоритмах зашифрування й розшифрування *Фейстеля*

Оскільки ключ під час зашифрування й розшифрування такий самий, можна довести, що два алгоритми інверсні один одному. Іншими словами, якщо $C_2 = C_1$ (будь-яка зміна в зашифрованих даних впродовж передачі), то $M_2 = M_1$.

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

Розшифрування:

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

$$= M_1 \oplus (00...0) = M_1.$$

Зауважимо, що для обчислення використовувалися дві властивості операції *виключного або* (*xor*) (існування інверсії й існування нульового коду).

Надані вище рівняння доводять, що хоча змішувач має неконвертований елемент, сам змішувач є самоконвертованим.

Приклад 3.12. Нехай є початкові й зашифровані дані, кожні *4* біти завдовжки, і ключ *3* біти. Припустимо, що функція *f(K)* використовує перший і третій біти ключа, інтерпретує біти як десятковий номер, підносить до другого степеня (квадрата) це число й інтерпретує результат як *4*-бітову двійкову послідовність. Необхідно показати результати зашифрування й розшифрування даних, якщо первинні початкові дані — *0111*, а ключ — *101*.

Розв'язання

Функція *f(K)* використовує перший і третій біт ключа: виходить *11* у двійковому вигляді або *3* у десятковому відображенні.

Результат піднесення до другого степеня (квадрат) — *9*, у двійковому відображенні *1001*.

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

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

Як видно, результат розшифрування співпаде з початковими даними *M*.

Функція $f(101) = 1001$ є неконвертованою, але операція *виключного або* (*xor*) дозволяє використовувати функцію $f(K)$ й в алгоритмах зашифрування й розшифрування. Іншими словами, функція $f(K)$ є неконвертованою, але змішувач буде самоконвертованим.

Щоб наблизитися до шифру Фейстеля необхідно удосконалити шифр, наданий на рис. 3.17.

Відомо, що треба застосувати вхід до неконвертованого елемента (функції), але при цьому необхідно використовувати не лише ключ. Задіюємо також вхід до функції $f(K)$, щоб застосувати її для зашифрування частини початкових даних і розшифрування частини зашифрованих даних. Ключ може використовуватися як другий вхід до функції $f(K)$. Завдяки цьому способу функція $f(K)$ стає складним елементом із деякими неключовими й ключовими елементами. Щоб досягти мети, розділимо початкові й зашифровані дані на два блоки рівної довжини — ліві (*L*) і праві (*R*). Правий блок вводиться у функцію $f(K)$ і набуває вигляду $f(R_1,K)$, а лівий блок складається за допомогою операції *виключного або* (*xor*) з виходом функції $f(R_1,K)$. Необхідно запам'ятати, що входи до функції $f(R_1,K)$ повинні точно співпадати під час зашифрування й розшифрування. Це означає, що права секція R_1 початкових даних до зашифрування й права секція R_1 зашифрованих даних після розшифрування будуть співпадати. Іншими словами, секція R_1 повинна увійти до зашифрування й вийти з розшифрування незміненою. Рис. 3.18 ілюструє цю ідею.

Алгоритми зашифрування й розшифрування інверсні один одному. Припустимо, що $L_3 = L_2$ і $R_3 = R_2$ (у зашифрованих даних впродовж передачі не сталося змін). Тоді

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

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

Початкові дані, що використовувались в алгоритмі зашифрування, — це дані, правильно відновлені алгоритмом розшифрування.

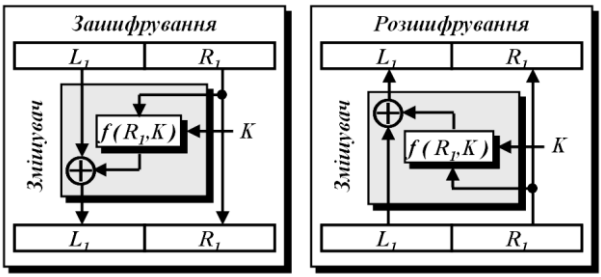


Рис. 3.18. Удосконалення попередньої схеми Фейстеля

Попереднє удосконалення має один недолік: права половина початкових даних ніколи не змінюється. Зловмисник може одразу знайти праву половину початкових даних, розбиваючи на частини зашифровані дані й розпаковуючи його праву половину. Проект потребує подальших кроків удосконалення: *по-перше,* необхідно збільшити кількість раундів; *по-друге,* необхідно додати новий елемент до кожного раунду — пристрій заміни. Ефект пристрою заміни в раунді зашифрування компенсується ефектом пристрою заміни в раунді розшифрування. Проте, це дозволяє змінювати ліві й праві половини в кожному раунді. Рис. 3.19 ілюструє новий варіант шифру Фейстеля з двома раундами.

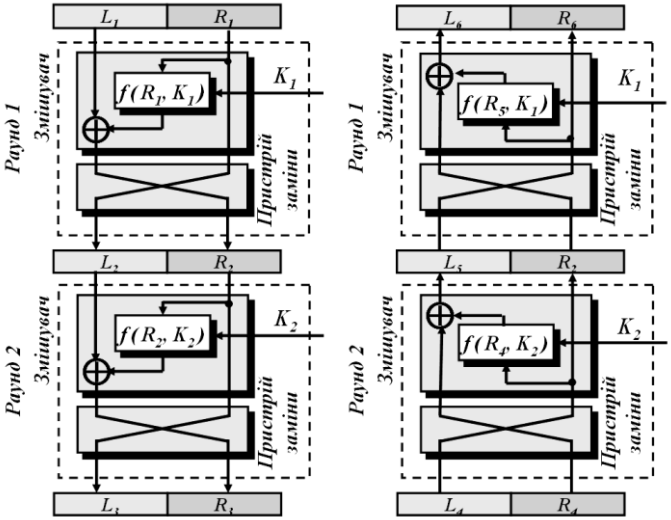


Рис. 3.19. Остаточний варіант шифру Фейстеля з двома раундами

Зауважимо, що є два ключі раундів: k_1 і k_2. Ключі використовуються у зворотному порядку під час зашифрування й розшифрування.

Оскільки два змішувачі й пристрої заміни інверсні один одному, очевидно, що зашифрування й розшифрування також інверсні один одному. Можна довести цей факт, використовуючи відношення між лівими й 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:

$$L_5 = R_4 \oplus f(L_4, K_2) = R_3 \oplus f(R_2, K_2) =$$

$$= L_2 \oplus f(R_2, K_2) \oplus f(R_2, K_2) = L_2,$$

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

Тоді просто довести рівність для двох блоків початкових даних:

$$L_6 = R_5 \oplus f(L_5,K_1) = R_2 \oplus f(L_2,K_1) =$$

$$= L_1 \oplus f(R_1,K_1) \oplus f(R_1,K_1) = L_1,$$

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

Шифри не-Фейстеля використовують тільки обернені компоненти. Компонент у початкових даних має відповідний компонент у шифрі. Наприклад, *S*-блоки заміни повинні мати рівну кількість входів і виходів, щоб бути сумісними (оберненими). Не дозволяється у шифрах ніяке стискання або розширення *P*-блоків, тому що вони стануть необернеными. У шифрах не-Фейстеля немає потреби ділити початковий текст на дві половини.

Рис. 3.19 можна розглядати як графічну ілюстрацію принципу шифру не-Фейстеля, тому що єдині компоненти в кожному раунді — самообернені операції *виключного або* (*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 кожний раунд є оберненим. Необхідно тільки застосовувати ключі раундів у зворотному порядку. Зашифрування 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, неможлива, тому що ключі зазвичай мають дуже велику довжину. Проте нещодавно було винайдено деякі нові види атак блокових шифрів, які засновані на структурі сучасних блокових шифрів. Ці атаки використовують також методи диференціального й лінійного аналізу.

3.3.1. Диференціальний криптографічний аналіз

Ідею відносно *диференціального криптографічного аналізу* запропонували *Елі Біхам* і *Аді Шамир*. Це — атака за вибіркою початкових даних [4, 8, 19, 22]. Зловмисник може яким-небудь чином отримати доступ до комп'ютера відправника й заволодіти вибірково частиною початкових даних і відповідних їм зашифрованих даних. Мета полягає в тому, щоб знайти ключ шифру відправника.

Алгоритм аналізу. Зловмисник, перш ніж зробить атаку за вибіркою початкових даних, повинен проаналізувати алгоритм зашифрування, щоб зібрати деяку інформацію про залежність зашифрованих і початкових даних. Очевидно, зловмисник не знає ключа шифру. Проте деякі шифри мають уразливі місця в структурах, які можуть дозволити зловмиснику знайти відмінності початкових даних і відмінності зашифрованих даних, не знаючи ключ.

Приклад 3.13. Припустимо, що шифр складається тільки з однієї операції *виключного або* (*xor*), як показано на рис. 3.20.

Не знаючи значення ключа, зловмисник може легко знайти співвідношення між різницями початкових й різницями зашифрованих даних.

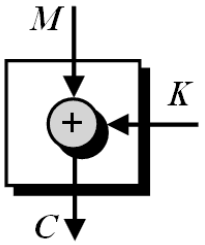


Рис. 3.20. Діаграма для прикладу 3.13

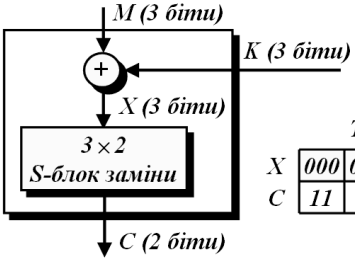
Якщо різницю початкових даних позначити $M_1 \oplus M_2$ а різницю зашифрованих даних позначити $C_1 \oplus C_2$, то наведені наступні перетворення доводять, що $C_1 \oplus C_2 = M_1 \oplus M_2$:

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

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

Проте цей приклад нереалістичний; модемні блокові шифри не настільки прості.

Приклад 3.14. У прикладі 3.13 необхідно додати один *S*-блок заміни, як показано на рис. 3.21.



Таблиця S-блока заміни

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

Рис. 3.21. Діаграма для прикладу 3.14

Хоча ефект зашифрування ключем не діє, коли використовуються різниці між двома X і двома P ($X_1 \oplus X_2 = M_1 \oplus M_2$), існування S-блока заміни заважає зловмиснику знайти певні співвідношення між різницями початкових даних і різницями зашифрованих даних. Проте можливо встановити ймовірнісні співвідношення. Зловмисник може скласти таблицю, яка показує скільки можна створити різниць зашифрованих даних — шифр для різниці початкових даних (табл. 3.5). Зауважимо, що таблицю складено за інформацією, яка створена з урахуванням таблиці входу-виходу S-блока заміни за рис. 3.21, тому що $M_1 \oplus M_2 = X_1 \oplus X_2$.

Таблиця 3.5

Диференціальна таблиця для входів і виходів для шифру в прикладі 3.14

	$C_1 \oplus C_2$				
		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 ключів — три біти, може бути вісім випадків для кожної різниці у введенні. Таблиця показує, що якщо вхідна різниця — $(000)_2$, 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 3.15. Евристичний результат прикладу 3.14 може створити ймовірнісну інформацію для зловмисника, як показано в табл. 3.6. Входи в таблиці відповідають імовірності появи. Різниці з нульовою ймовірністю ніколи не будуть виникати.

Як буде зазначено пізніше, зловмисник тепер має в розпорядженні достатню кількість інформації, щоб почати атаку. Табл. 3.6 показує, що ймовірність розподілена неоднорідно через уразливість у структурі *S*-блока заміни. Табл. 3.6 згадується іноді як диференціальна таблиця розподілу або профайл *виключного або* (*xor*).

Таблиця 3.6

Диференціальна таблиця для входів і виходів для шифру в прикладі 3.15

	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 вибрані початкові дані. Після того, як евристичний аналіз зроблено один раз, він може бути збережений для майбутнього використання, поки структура шифру не зміниться. Зловмисник може вибрати для атак початкові дані. Диференціальна таблиця розподілу ймовірностей (табл. 3.6) допоможе зловмиснику їх вибирати — перебрати, які мають найвищу ймовірність у таблиці.

Передбачуване значення ключа. Після запуску деяких атак за вибіркою початких даних зловмисник може знайти деяку пару "*початкові дані/зашифровані дані*", яка дозволяє йому припустити деяке значення ключа. Процес починається від *C* і просувається до *M*.

Приклад 3.16. Розглядаючи табл. 3.6, зловмисник знає, що якщо $M_1 \oplus M_2 = 001$, то $C_1 \oplus C_2 = 11$ з імовірністю 0,5. Він пробує взяти $C_1 = 00$ і отримує $M_1 = 010$ (атака за вибіркою зашифрованих даних). Він ще пробує взяти $C_2 = 11$ і отримує $M_2 = 011$ (інша атака за

вибіркою зашифрованих даних). Тепер зловмисник пробує повернутися до аналізу, заснованого на першій парі, M_1 і C_1 (див. рис. 3.21):

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

Якщо $X_1 = 001$, то $K = X_1 \oplus M_1 = 011$.
Якщо $X_1 = 111$, то $K = X_1 \oplus M_1 = 101$.
Використовуючи пару M_2 і C_2, отримаємо:

$$C_2 = 11 \rightarrow X_2 = 000 \text{ або } X_1 = 110.$$

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 знає, що крайній правий біт — *1* (загальний біт між двома значеннями). Продовжуючи атаку, можна враховувати, що крайній правий біт у ключі — *1*. Отже, можна визначити й інші біти у цьому ключі.

Загальна процедура. Сучасні блокові шифри мають більшу складність, ніж та, яка обговорювалася у цьому розділі. Крім того, вони можуть містити різну кількість раундів. Зловмисник може використовувати таку стратегію:

1. Оскільки кожний раунд містить ті самі операції, зловмисник може створити таблицю диференціальних розподілів (профайл *виключного або* (*xor*)) для кожного *S*-блока заміни й 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 початкових даних для атак, заснованих на таблиці розподілів, створеної на другому кроці. Зазначимо, що таблиця створена за кроком 2 допомагає зловмиснику вибирати тільки меншу кількість пар "*початкові дані/зашифровані дані*".

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

5. Зловмисник повторює дії кроку 4, 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.

Детальний диференціальний криптографічний аналіз наведено в розділі 5.

3.3.2. Лінійний криптографічний аналіз

Лінійний криптографічний аналіз запропонував *Міцуру Мацуі* 1993 року. Аналіз використовує атаки на відомі початкові дані (на відміну від атак за вибіркою початкових даних у диференціальному криптографічному аналізі). Повний розгляд 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. 3.22, де c_0, c_1 і c_2 являють собою три біти на виході й x_0, x_1 і x_2 — три біти на вході *S*-блока заміни.

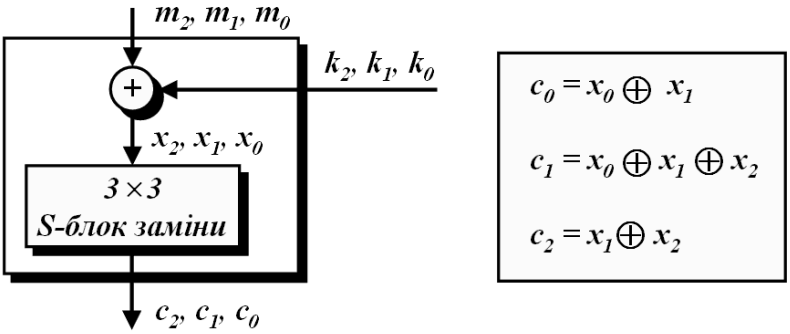


Рис. 3.22. Простий шифр із лінійним *S*-блоком заміни

S-блок заміни — лінійне перетворення, в якому кожний вхід є лінійною функцією виходу, яке було розглянуте вище. Із цим лінійним компонентом можна створити три лінійні рівняння між початковими даними й бітами зашифрованих даних, такі як:

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

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

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

Розв'язавши систему рівнянь для трьох невідомих, отримуємо:

$$k_1 = (m_1) \oplus (c_0 \oplus c_1 \oplus c_2);$$

$$k_2 = (m_2) \oplus (c_0 \oplus c_1);$$

$$k_0 = (m_0) \oplus (c_1 \oplus c_2).$$

Це означає, що три атаки на відомі початкові дані можуть знайти значення k_1 і k_2. Проте реальні блокові шифри не такі прості, як цей; вони мають більше компонентів, і *S*-блоки заміни не лінійні.

Лінійна апроксимація. В окремих сучасних блокових шифрах деякі *S*-блоки заміни не повністю нелінійні; тоді вони можуть бути за ймовірнісної суті апроксимовані деякими лінійними функціями. Взагалі, задаючи початкові й зашифровані дані в n бітах і ключ m бітів, шукають деякі рівняння, що мають вигляд:

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

Контрольні питання та завдання

1. Указати відмінності між сучасним і традиційним шифрами із симетричним ключем.
2. Пояснити, чому сучасні блокові шифри спроектовані як шифри підстановки замість того, щоб застосовувати шифри транспозиції.
3. Перерахувати компоненти сучасного блочного шифру.
4. Визначити *P*-блок і назвати три його варіанти. Який варіант є оберненим?
5. Визначити *S*-блок і покажіть необхідну умову оберненості *S*-блока.
6. Визначити складовий шифр і назвати два класи складових шифрів.
7. Пояснити відмінність між розсіюванням і перемішуванням.
8. Навести відмінність між блоковим шифром Фейстеля і не-Фейстеля.
9. Яка різниця між диференціальним і лінійним криптографічним аналізом? Який криптографічний аналіз використовує атаку на вибірки вхідного тексту, а який — атаку на знання вихідного тексту?
10. Повідомлення має *1500* символів, для подання яких використовуються *ASCII* коди. Це повідомлення буде зашифроване блоковим шифром довжиною *64* біти. Знайти розмір доповнення й кількість зашифрованих блоків.

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

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

13. Використовуючи слово $(10011011)_2$, показати результат циркулярного (циклічного) лівого зсуву на *3* біти.

14. Використовуючи слово $(10011011)_2$, показати результат циркулярного (циклічного) правого зсуву на *3* біти.

15. Знайти результат таких операцій:

а) $(01001101)_2 \oplus (01001101)_2$; б) $(01001101)_2 \oplus (10110010)_2$;

в) $(01001101)_2 \oplus (00000000)_2$; г) $(01001101)_2 \oplus (11111111)_2$.

16. Здійснити перестановку бітів для прямого *P*-блока, показаного на рис. 3.4, якщо на його вході діє послідовність: $(10110)_2$.

17. Здійснити перестановку бітів для *P*-блока стискання, показаного на рис. 3.4, якщо на його вході діє послідовність: $(10110)_2$.

18. Здійснити перестановку бітів для *P*-блока розширення, показаного на рис. 3.4, якщо на його вході діє послідовність: $(101)_2$.

19. Визначте, чи є *P*-блок з таблицею перестановки

8	*1*	*2*	*3*	*4*	*5*	*6*	*7*

прямим *P*-блоком, *P*-блоком стискання або *P*-блоком розширення.

20. Визначте, чи є *P*-блок із таблицею перестановки

1	*1*	*2*	*3*	*4*	*4*

прямим *P*-блоком, *P*-блоком стискання або *P*-блоком розширення.

21. Визначити, чи є *P*-блок із таблицею перестановки

1	*3*	*5*	*6*	*7*

прямим *P*-блоком, *P*-блоком стискання або *P*-блоком розширення.

22. Визначити, чи є *P*-блок із таблицею перестановки

7	*3*	*1*	*4*	*8*	*5*	*2*	*6*

прямим *P*-блоком, *P*-блоком стискання або *P*-блоком розширення.

23. Відношення вхід-вихід *2×2 S*-блока показано в такій таблиці

		Вхід: правий біт	
		0	*1*
Вхід: лівий біт	*0*	*01*	*11*
	1	*10*	*00*

Показати таблицю для інверсного блока.

24. Показати *LFSR* із характеристичним поліномом $x^5 + x^2 + 1$. Який період одержуваної послідовності?

25. Крайній лівий біт *S*-блока розміром 4×3 визначає зміщення інших трьох бітів. Якщо крайній лівий біт дорівнює *0*, то три інших біти переміщаються вправо на один біт. Якщо крайній лівий біт — *1*, три інші біти переміщаються вліво на один біт. Якщо вхід — *1011*, який результат буде на виході? Якщо вхід — *0110*, який результат буде на виході?

Розділ 4

ПОТОКОВІ ШИФРИ Й ГЕНЕРАТОРИ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ

4.1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПОТОКОВІ ШИФРИ

Блоковий алгоритм призначений для шифрування блоків певної довжини. Однак може виникнути необхідність шифрування даних не блоками, а, наприклад, по символах. 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, може працювати в реальному часі. Отже, якщо передається потік символів, кожний символ може шифруватися й передаватися відразу [30].

Принцип роботи типового потокового шифру надано на рис. 4.1.

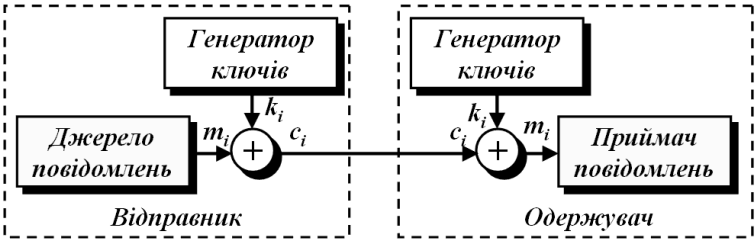


Рис. 4.1. Принцип роботи потокового шифру

Генератор ключів видає потік бітів k_i, які будуть використовуватися як гама. Джерело повідомлень генерує біти відкритих даних m_i, які складаються за модулем *2* з гамою, внаслідок чого виходять біти зашифрованих даних c_i:

$$c_i = m_i \oplus k_i,\ i = 1, 2, \ldots, n.$$

Щоб із зашифрованого повідомлення $c_1, c_2, ..., c_n$ відновити початкове повідомлення $m_1, m_2, ..., m_n$, необхідно згенерувати точно таку саму ключову послідовність $k_1, k_k, ..., k_n$, що й зашифрування, і використовувати для розшифрування формулу

$$m_i = c_i \oplus k_i,\ i = 1, 2, \ldots, n,$$

оскільки як операція порозрядного додавання за модулем *2* має властивість оберненості.

Зазвичай вхідним повідомленням й ключовою послідовністю є незалежні потоки бітів. Отже, оскільки зашифрувальне (й розшифрувальне) перетворення 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 вхідний потік бітів (у разі поодиноких ключів, зашифроване повідомлення буде інверсією вхідного). Якщо як гама використовується один символ, представлений, наприклад, вісьмома бітами, то хоча зашифроване повідомлення й буде зовні відрізнятися від вхідного, безпека системи буде дуже низькою. У цьому випадку — за багаторазового повторення коду ключа по всій довжині повідомлення існує небезпека його розкриття статистичним методом. Поясним це на простому прикладі двійкового повідомлення, закритого коротким двійковим кодом ключа методом гамування.

Приклад 4.1. Нехай відомо, що вхідне повідомлення являло собою двійково-десяткове число, тобто число, кожна тетрада (чотири біти) якого отримана при перекладі десяткової цифри *0...9* у двійковому вигляді. Перехоплено *24* біти зашифрованого повідомлення *M*, тобто шість тетрад m_1, m_2, m_3, m_4, m_5 і m_6, а саме значення $C = 1100\ 1101\ 1110\ 1111\ 0000\ 0001$. Відомо, що ключ зашифрування складався з чотирьох бітів, які теж являють собою однозначне десяткове число, тобто те саме значення $0 \le K \le 9$ використовувалося для зашифрування кожних чотирьох бітів вхідного повідомлення. Отже, зашифрування числа m_1, m_2, m_3, m_4, m_5 і m_6 ключем *K* можна представити у вигляді системи рівнянь:

$$m_1 \oplus K = 1100; \quad m_2 \oplus K = 1101; \quad m_3 \oplus K = 1110;$$

$$m_4 \oplus K = 1111; \quad m_5 \oplus K = 0000; \quad m_6 \oplus K = 0001.$$

Виходячи з умови, що m_i приймає десяткові значення від *0* до *9*, для пошуку невідомого K визначимо всі можливі значення m'_1 і K, сума яких за модулем *2* приводить до результату *1100*:

$$\begin{array}{ll} & K = \ 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001 \\ \oplus & c_1 = \ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100 \\ \hline & m'_1 = \ 1100\ 1101\ 1110\ 1111\ 1000\ 1001\ 1010\ 1011\ 0100\ 0101 \end{array}.$$

Оскільки початкове повідомлення складалося із цифр від *0* до *9*, то можна виключити з розгляду значення ключа *0000*, *0001*, *0010*, *0011*, *0110*, *0111*, тому що при складанні з ними виходять значення більше за *9* у десятковому еквіваленті. Такі значення не можуть бути у відкритому повідомленні. Отже, перший етап аналізу вже дозволив скоротити кількість можливих ключів із десяти до чотирьох.

Для подальшого пошуку невідомого K визначимо всі можливі значення m_2' і варіанти ключа, які залишилися, сума яких за модулем *2* приводить до результату $c_2 = 1101$:

$$\begin{array}{ll} & K = \ 0100\ 0101\ 1000\ 1001 \\ \oplus & c_2 = \ 1101\ 1101\ 1101\ 1101 \\ \hline & m'_2 = 1001\ 1000\ 0101\ 0100 \end{array}.$$

Видно, що цей етап не дозволив відкинути жодного з варіантів ключа, що залишилися. Спробуємо це зробити, використовуючи $c_3 = 1110$:

$$\begin{array}{ll} & K = \ 01000101\ 10001001 \\ \oplus & C_3 = 11101110\ 11101110 \\ \hline & m'_3 = 10101011\ 01100111 \end{array}.$$

Після проведення цього етапу стає зрозумілим, що ключем не могли бути значення *0100* і *0101*. Залишається два можливих значення ключа: $1000_{(2)} = 8_{(10)}$ і $1001_{(2)} = 9_{(10)}$.

Подальший аналіз за даною методикою в даному випадку, на жаль, не дозволить однозначно вказати, який із двох отриманих варіантів ключа використовувався під час зашифрування. Однак

можна вважати вдалим вже те, що простір можливих ключів знизився з десяти до двох. Залишається спробувати кожний із двох знайдених ключів для дешифрування повідомлень і проаналізувати зміст отриманих розкритих даних.

У реальних випадках, коли вхідне повідомлення складається не лише з одних цифр, але й з інших символів, використання статистичного аналізу дозволяє швидко й точно відновити ключ і вхідні повідомлення з короткою довжиною ключа, який приховує потік секретних даних.

4.2. ПРИНЦИПИ ВИКОРИСТАННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ ПІД ЧАС ПОТОКОВОГО ШИФРУВАННЯ

Сучасна інформатика широко використовує псевдовипадкові числа в різноманітних додатках — від методів математичної статистики й імітаційного моделювання до криптографії. При цьому від якості використовуваних *генераторів псевдовипадкових чисел* (ГПВЧ) безпосередньо залежить якість одержуваних результатів [19].

ГПВЧ можуть використовуватися як генератори 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 зашифрування, 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 зашифрування.

Для використання з криптографічною метою ГПВЧ повинен мати такі властивості:

- період послідовності повинен бути дуже великим;
- породжувана послідовність не повинна відрізнятися від дійсно випадкової;

- ймовірності появи (породження) різних значень повинні бути точно рівні;

- для того, щоб тільки законний одержувач міг розшифрувати повідомлення, слід під час отримання потоку ключових бітів k_i використовувати та враховувати певний секретний ключ, причому обчислення числа k_{i+1} за відомими попередніми елементами послідовності k_i без знання ключа повинно бути важким завданням.

За наявності зазначених властивостей послідовності псевдовипадкових чисел можуть бути використані в потокових шифрах.

4.2.1. 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 за наведеною вище формулою можливо отримати значення від *0* до *10* (оскільки $c = 11$). Обчислимо кілька елементів послідовності:

$$k_1 = (5 \cdot 1 + 3) \bmod 11 = 8; \quad k_2 = (5 \cdot 8 + 3) \bmod 11 = 10;$$

$$k_3 = (5 \cdot 10 + 3) \bmod 11 = 9; \quad k_4 = (5 \cdot 9 + 3) \bmod 11 = 4;$$

$$k_5 = (5 \cdot 4 + 3) \bmod 11 = 1.$$

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 будуть дорівнювати *10* і *9* відповідно:

$$k_7 = (5 \cdot 8 + 3) \bmod 11 = 10; \quad k_8 = (5 \cdot 10 + 3) \bmod 11 = 9.$$

Виходить, ГПВЧ 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* (ЕОМ) або *автоматизованої системи* (АС) у бітах.

Перевагою лінійних конгруентних ГПВЧ є їх простота й висока швидкість отримання псевдовипадкових значень. Лінійні конгруентні генератори застосовуються під час розв'язання задач моделювання та математичної статистики, однак з криптографічною метою їх не можна рекомендувати для використання, оскільки фахівці з криптографічного аналізу навчилися відновлювати всю послідовність *псевдовипадкових чисел* (ПВЧ) із кількох значень. Наприклад, припустимо, що зловмисник може визначити значення k_0, k_1, k_2, k_3. Тоді:

$$k_1 = (a \cdot k_0 + b) \bmod c; \quad k_2 = (a \cdot k_1 + b) \bmod c; \quad k_3 = (a \cdot k_2 + b) \bmod c.$$

Розв'язавши систему цих трьох рівнянь, можна знайти *a*, *b* і *c*. Для отримання ПВЧ 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*".

4.2.2. Метод Фібоначчі із запізненням

Метод Фібоначчі із запізненням (*Lagged Fibonacci Genera – tor*) — один із методів генерації ПВЧ, що дозволяє отримати більш високу "*якість*" ПВЧ. Найбільшу популярність фібоначчієві датчики отримали через те, що швидкість виконання арифметичних

операцій із числами зрівнялася зі швидкістю цілочисельної арифметики, а фібоначчієві датчики природно реалізуються в дійсній арифметиці [5, 19-21]..

Відомі різні схеми використання методу Фібоначчі із запізненням. Один із широко поширених фібоначчієвих датчиків заснований на такій 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 фібоначчієвого датчика потрібні *max {a, b}* значення попередніх згенерованих випадкових чисел. За програмної реалізації для зберігання згенерованих випадкових чисел необхідний певний обсяг пам'яті, яка залежить від параметрів *a* і *b*.

Приклад 4.2. Обчислимо послідовність із перших десяти чисел, що генерується методом Фібоначчі із запізненням починаючи з k_5 за таких вхідних даних: $a = 4$, $b = 1$, $k_0 = 0.1$; $k_1 = 0.7$; $k_2 = 0.3$; $k_3 = 0.9$; $k_4 = 0.5$:

$$k_5 = k_1 - k_4 = 0,7 - 0,5 = 0,2;$$

$$k_6 = k_2 - k_5 = 0,3 - 0,2 = 0,1;$$

$$k_7 = k_3 - k_6 = 0,9 - 0,1 = 0,8;$$

$$k_8 = k_4 - k_7 + 1 = 0,5 - 0,8 + 1 = 0,7;$$

$$k_9 = k_5 - k_8 + 1 = 0,2 - 0,7 + 1 = 0,5;$$

$$k_{10} = k_6 - k_9 + 1 = 0,1 - 0,5 + 1 = 0,6;$$

$$k_{11} = k_7 - k_{10} = 0,8 - 0,6 = 0,2;$$

$$k_{12} = k_8 - k_{11} = 0,7 - 0,2 = 0,5;$$

$$k_{13} = k_9 - k_{12} + 1 = 0,5 - 0,5 + 1 = 1;$$

$$k_{14} = k_{10} - k_{13} + 1 = 0,6 - 1 + 1 = 0,6,$$

Бачимо: послідовність чисел, що генерується, зовні схожа на випадкову. І дійсно, дослідження підтверджують, що одержані 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 великої розмірності [28].

ГПВЧ, засновані на методі Фібоначчі із запізненням, використовувалися для цілей криптографії. Крім того, вони застосовуються в математичних та статистичних розрахунках, а також за моделювання випадкових процесів. ГПВЧ, побудований на основі методу Фібоначчі із запізненням, використовувався у широко відомій системі *Matlab*.

4.2.3. Генератор псевдовипадкових чисел на основі алгоритму *BBS*

Широке поширення набув алгоритм генерації ПВЧ, названий *алгоритмом BBS* (від прізвищ авторів — *L. Blum, M. Blum, M. Shub*) або *генератором із квадратичним залишком*. Для цілей криптографії цей метод запропонований 1986 р. [1, 38, 44, 45].

Сутність цього алгоритму: спочатку вибираються два великих простих числа p та q, які потрібно порівняти з *3* за модулем *4*, тобто при діленні p і q на *4* повинен виходити однаковий залишок *3*. Далі обчислюється число $M = p \cdot q$, так зване *ціле число Блюма*. Потім вибирається інше випадкове ціле число x, взаємно просте (тобто не має загальних дільників, крім одиниці) з M і обчислюється $x_0 = x^2 \bmod M$, де x_0 *– стартове число 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, біт парності приймається таким, що дорівнює *0*, непарне — біт парності приймається таким, що дорівнює *1*.

Приклад 4.3. 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*. Як випадковий 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 з послідовності n-го числа 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} одночасно з x_0:

$$x_{10} = x_0^{2^{10 \bmod ((11-1)(19-1))}} \; mod\ 209 =$$
$$= 9^{1024 \bmod 180} mod\ 209 = 137.$$

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 таємниці; до того часу, поки *M* не розкладено на множники, ніхто не зможе передбачити вихід ГПВЧ. Це пов'язано з тим, що завдання розкладення чисел виду $n = p \cdot q$ (p і q — прості числа) на множники є надто складним для обчислення, якщо відоме тільки *n*, а *p* і *q* — великі числа, що складаються з декількох десятків або сотень бітів (це так зване завдання 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 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 вдалу послідовність ПВЧ із великим періодом (за відповідного вибору вхідних параметрів), що дозволяє використовувати його для криптографічних цілей під час генерації ключів для шифрування.

4.2.4. 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 ЕОМ і сучасних високошвидкісних програмних шифраторів.

Регістри зсуву зі зворотним зв'язком можуть застосовуватися для отримання потоку псевдовипадкових бітів і складаються з двох частин: власне *n*-бітного регістра зсуву та пристрою зворотного зв'язку (рис. 4.2).

Регістри зсуву зі зворотним зв'язком можуть застосовуватися для отримання потоку псевдовипадкових бітів. Регістр зсуву зі зворотним зв'язком складається з двох частин: власне *n*-бітного регістра зсуву та пристрою зворотного зв'язку (рис. 4.2).

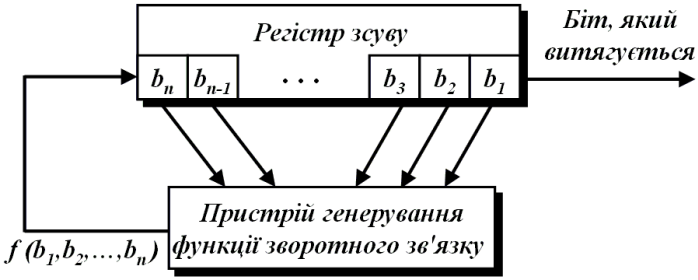


Рис. 4.2. Регістр зсуву із зворотним зв'язком

Витягувати біти з регістра зсуву можна тільки по одному (по черзі). Якщо необхідно отримати 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 бітів регістра зсуву. Завдяки цьому біти регістра змінюються за певним законом, який і визначає схему отримання ПВЧ. 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* (*linear feedback shift register — LFSR*). Зворотний зв'язок у цьому пристрої реалізується просто як сума за модулем *2* 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 застосовуються в криптографії.

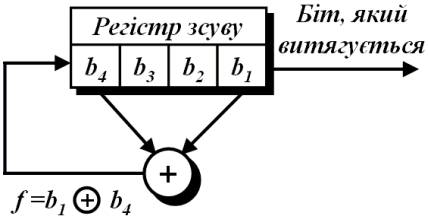


Рис. 4.3. Приклад *4*-розрядного лінійного регістра зсуву

Для того, щоб стало зрозуміліше, як працює регістр зсуву зі зворотним зв'язком, розглянемо *4*-бітовий *LFSR* із відведенням від першого й четвертого розрядів, який показано на рис. 4.3.

Для того, щоб стало зрозуміліше, як працює регістр зсуву зі зворотним зв'язком, розглянемо *4*-бітовий *LFSR* із відведенням від першого й четвертого розрядів, наданий на рис. 4.3.

Запишемо в зображений на рис. 4.3 регістр початкове значення *1011*. Обчислювати послідовність внутрішніх станів регістра зручно за допомогою таблиці (табл. 4.1). У таблиці наведено перші дев'ять станів регістра.

Таблиця 4.1

Послідовність роботи лінійного регістра зсуву

Номер стану	Внутрішній стан регістра (b_4, b_3, b_2, b_1)	Результат обчислення функції зворотного зв'язку $f = b_1 \oplus b_4$	Біт, який витягується (b_1)
0	*1 0 1 1*	*0*	*1*
1	*0 1 0 1*	*1*	*1*
2	*1 0 1 0*	*1*	*0*
3	*1 1 0 1*	*0*	*1*
4	*0 1 1 0*	*0*	*0*
5	*0 0 1 1*	*1*	*1*
6	*1 0 0 1*	*0*	*1*
7	*0 1 0 0*	*0*	*0*
8	*0 0 1 0*	*0*	*0*

На кожному кроці весь вміст регістра зсувається вправо на один розряд. При цьому можна отримати як результат один біт. На вільне місце ліворуч надходить біт, що дорівнює результату обчислення функції зворотного зв'язку $f = b_1 \oplus b_4$. Вихідну послідовність генератора псевдовипадкових біт утворює останній стовпець таблиці (біт, який витягується).

Лінійний регістр зсуву розміром *n* бітів може перебувати в одному з 2^n-1 станів. Тому, теоретично такий регістр може генерувати псевдовипадкову послідовність із максимальним періодом 2^n-1.

Лінійний регістр зсуву зі зворотним зв'язком буде генерувати циклічну послідовність бітів із максимальним періодом тільки за вибору відвідної послідовності окремих бітів. Розроблено математичну теорію, що дозволяє вибрати відповідні номери розрядів для бітів відвідної послідовності [3, 5, 14, 18-21, 40].

Лінійні регістри зсуву зі зворотним зв'язком часто використовувалися й використовуються до цього часу під час шифрування потоків даних.

Для підвищення криптографічної стійкості в таких пристроях шифрування застосовуються комбінації декількох регістрів зсуву зі зворотним зв'язком, а також вводяться додаткові перемішувальні операції. Такі електронні схеми пропонувалися й випускалися ще до початку Другої світової війни. Аналогічні принципи закладені також у деякі потокові шифри, створені наприкінці XX століття, наприклад, в алгоритм *A5*, який використовували в Європі для шифрування стільникових цифрових каналів зв'язку стандарту *GSM*. Незважаючи на те, що деякі криптографічні аналітики висловлюють сумніви щодо надійності алгоритмів потокового шифрування з використанням лінійних регістрів зсуву зі зворотним зв'язком, що покладені в основу функціонування різних військових і цивільних пристроїв зв'язку, вони використовуються до теперішнього часу [30].

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.

4.3. 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. Буде втрачено лише один знак 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. Майже у всіх каналах передачі даних для потокових систем шифрування є перешкоди. Тому для запобігання втрати інформації вирішують проблему синхронізації зашифрування й розшифрування повідомлення. За способом вирішення цієї проблеми криптографічні системи поділяються на синхронні системи й системи із самосинхронізацією.

4.3.1. Синхронні потокові шифри

Синхронні потокові шифри (СПШ) — це шифри, в яких потік ключів генерується незалежно від відкритого й зашифрованого повідомлення.

Під час зашифрування генератор потоку ключів видає біти потоку ключів, які ідентичні бітам потоку ключів під час розшифрування. Втрата знака зашифрованого повідомлення призведе до порушення синхронізації між цими двома генераторами й неможливості розшифрування частини повідомлення. Очевидно, що в цій ситуації відправник та одержувач повинні повторно синхронізуватися для продовження роботи.

Зазвичай синхронізація проводиться вставкою в передане повідомлення спеціальних маркерів. Внаслідок цього пропущений під час передачі знак призводить до неправильного розшифрування повідомлення лише до того часу, поки не буде прийнятий один із маркерів.

Зауважимо, що виконуватися синхронізація повинна так, щоб ні одна частина потоку ключів не повторювалась. Тому, переводити генератор у більш ранній стан не має сенсу.

Основними перевагами СПШ є:

- відсутність ефекту поширення помилок (тільки спотворений біт буде розшифрований неправильно);

- оберігають від будь-яких вставок і вилучень зашифрованого повідомлення, оскільки вони призведуть до втрати синхронізації й будуть виявлені.

Недоліками СПШ є: уразливість до зміни окремих бітів зашифрованого повідомлення. Якщо зловмисникові відоме відкрите повідомлення, він може змінити ці біти так, щоб вони розшифровувалися, як йому потрібно.

4.3.2. Самосинхронізуючі потокові шифри

Самосинхронізуючі потокові шифри (*асинхронні потокові шифри* (АПШ)) — 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, який розшифровує, прийнявши *n* бітів, автоматично синхронізується із шифрувальним генератором.

Реалізація цього режиму відбувається так: 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 АПШ є розмішування статистики відкритого повідомлення. Оскільки кожний знак відкритого повідомлення впливає на наступне зашифроване повідомлення, статистичні властивості відкритого повідомлення поширюються на всі зашифровані повідомлення. Отже, АПШ може бути більш стійким до атак на підставі надмірності відкритого повідомлення, ніж СПШ.

Недоліками АПШ є:

- поширення помилки (кожному неправильного біту зашифрованого повідомлення відповідають *n* помилок у відкритому повідомленні);

- чутливі до зламу повторною передачею.

4.4. ПОТОКОВИЙ ШИФР A5

4.4.1. Історія створення потокового шифру A5

A5 — 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 складанні за модулем *2* (булева операція *xor*) псевдовипадкової послідовності, яка генерується, і інформації, що зашифровується. В *A5* псевдовипадкова послідовність реалізується на основі трьох лінійних регістрів зсуву зі зворотним зв'язком. Регістри мають кількість розрядів (довжини) *19*, *22* і *23* біти відповідно. Зсувом управляє спеціальна схема, яка на кожному кроці зміщує як мінімум два регістри, що призводить до нерівномірного руху змісту їх послідовностей. Послідовність формується шляхом операції *xor* над вхідними бітами регістрів.

Спочатку французькі військові фахівці — криптографи розробили потоковий шифр для використання виключно у військових цілях. У кінці 80-х рр. XX ст. для стандарту *GSM* потрібно було створити нову, сучасну систему безпеки. В її основу лягли три секретні алгоритми: аутентифікації — *A3*; потокового шифрування — *A5*; генерації ключа сеансу — *A8*. Як алгоритм *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 та США. Для решти країн алгоритм модифікували, значно знизивши криптографічну стійкість шифру. *A5/2* був спеціально розроблений як експортний варіант для країн, що не входили до Євросоюзу. Криптографічну стійкість *A5/2* було знижено додаванням ще одного регістра (*17* бітів), який управляв зсувом інших. 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 розголошувалась. Це пов'язано з тим, що розробники покладалися на безпеку завдяки невідомості, тобто алгоритми складніше зламати, якщо вони не доступні публічно. Дані надавалися операто-

рам *GSM* тільки за потребою. Тим не менш, до 1994 р. деталі алгоритму *A5* були 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 криптографічної стійкості. Остаточно алгоритм був представлений у роботі *Йована Голіча* на конференції *Eurocrypt'97* [6, 30].

4.4.2. Опис алгоритму A5

Алгоритм *A5* у даний час — це ціле сімейство шифрів. Для опису візьмемо *A5/1* (рис. 4.4) як родоначальника цього сімейства.

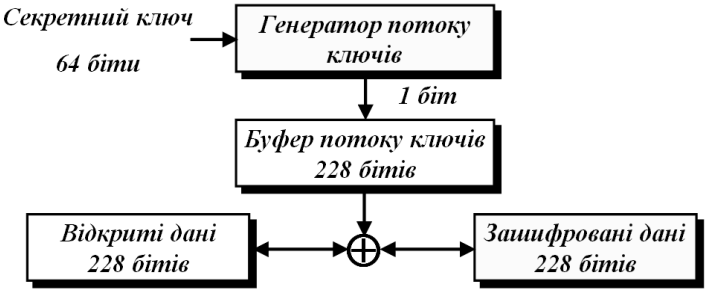


Рис. 4.4. Принцип побудови алгоритму потокового зашифрування (розшифрування) даних *A5/1*

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 складати їх за модулем *2* з кадром на *228* бітів, як показано на рис. 4.4. Зміни в похідних алгоритмах опишемо окремо.

На рис. 4.4 стрілки в обидва боки показують, що під час зашифрування беруть відкриті дані, а після підсумовування з потоком ключів за модулем *2* виходять зашифровані дані; під час розшифрування

беруть зашифровані дані, а після підсумовування з потоком ключів за модулем *2* 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 *2* і зсув бітів регістра.

Формування вихідної послідовності відбувається шляхом складання потоку вхідного повідомлення з послідовністю (гамою), яка генерується. Особливість операції *xor* полягає в тому, що застосована парне число разів вона приводить до початкового значення. Звідси, розшифрування повідомлення відбувається шляхом складання зашифрованого повідомлення з відомою послідовністю (гамою).

Отже, безпека шифру *A5* повністю залежить від властивостей послідовності. В ідеальному випадку кожний біт гами — це незалежна випадкова величина, і сама послідовність є випадковою. Таку схему, названу на його честь, винайшов Вернам 1917 р. Як довів Клод Шеннон 1949 р., це забезпечує абсолютну криптографічну стійкість. Але використання випадкової послідовності означає передачу по захищеному каналу цієї послідовності, рівної за обсягом відкритому повідомленню, що значно ускладнює завдання та майже ніде не використовується.

У реальних системах створюється ключ заданого розміру, який легко передається по закритому каналу. Послідовність генерується на його основі та є псевдовипадковою. Великий клас потокових шифрів (у тому числі *A5*) складають шифри, в яких генератор псевдовипадкової послідовності заснований на регістрах зсуву з лінійним зворотним зв'язком (*LFSR*).

LFSR складається з власне регістра (послідовності бітів заданої довжини) і зворотного зв'язку (рис. 4.5).

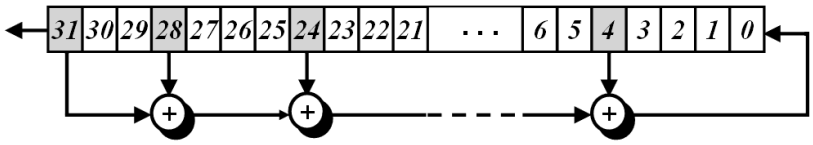


Рис. 4.5. *LFSR* із багаточленом зворотного зв'язку
$x^{32}+x^{29}+x^{25}+x^{5}+1$

На кожному такті відбуваються такі дії: крайній лівий біт (старший біт) вилучається, послідовність у регістрі зсувається вліво і у вільну праву клітинку (молодший біт) записується значення функції зворотного зв'язку. Ця функція є підсумовуванням за модулем *2* певних бітів регістра й записується у вигляді багаточлена, де ступінь вказує на номер біта. Вилучені біти формують вихідну послідовність.

Для *LFSR* основним показником є період псевдовипадкової послідовності, який буде максимальним (дорівнює 2^n-1), якщо багаточлен функції зворотного зв'язку примітивний за модулем *2*. Вихідна послідовність у такому випадку називається *M*-послідовністю (послідовність максимально можливої довжини, яка не повторюється).

Сам по собі *LFSR* легко піддається криптографічному аналізу і не є достатньо надійним для використання при шифруванні. Практичне застосування мають системи регістрів змінного тактування із різними довжинами й функціями зворотного зв'язку.

Схема потокового шифру *A5* включає в себе (рис. 4.6):

- три регістри (R_1, R_2, R_3), що мають довжину *19*, *22* і *23* розрядів відповідно, багаточлени зворотних зв'язків для яких:

$x^{19} + x^{18} + x^{17} + x^{14} + 1$ для R_1;
$x^{22} + x^{21} + 1$ для R_2;
$x^{23} + x^{22} + x^{21} + x^8 + 1$ для R_3;

- схему управління тактуванням.

Для управління тактуванням у кожному регістрі є біти синхронізації: *8* (R_1), *10* (R_2), *10* (R_3). Із використанням цих бітів обчислюється функція

$$f = x \,\&\, y \mid x \,\&\, z \mid y \,\&\, z,$$

де *&* — булеве *and*; **|** — булеве *or*; *x*, *y* і *z* — біти синхронізації R_1, R_2 і R_3 відповідно.

Розглянемо особливості функціонування алгоритму на підставі відомої схеми (рис. 4.6). Передача даних здійснюється в структурованому вигляді — із розбиттям на кадри (*114* бітів). Перед ініціалізацією регістри обнуляються, на вхід алгоритму надходять: ключ сеансу (*K — 64* біти), сформований алгоритмом *A8*, і номер кадру (*Fn — 22* біти). Далі послідовно виконуються такі дії: однобітовий вихід забезпечує тактовими імпульсами буфер на *228* бітів, який використовується для зашифрування (або розшифрування).

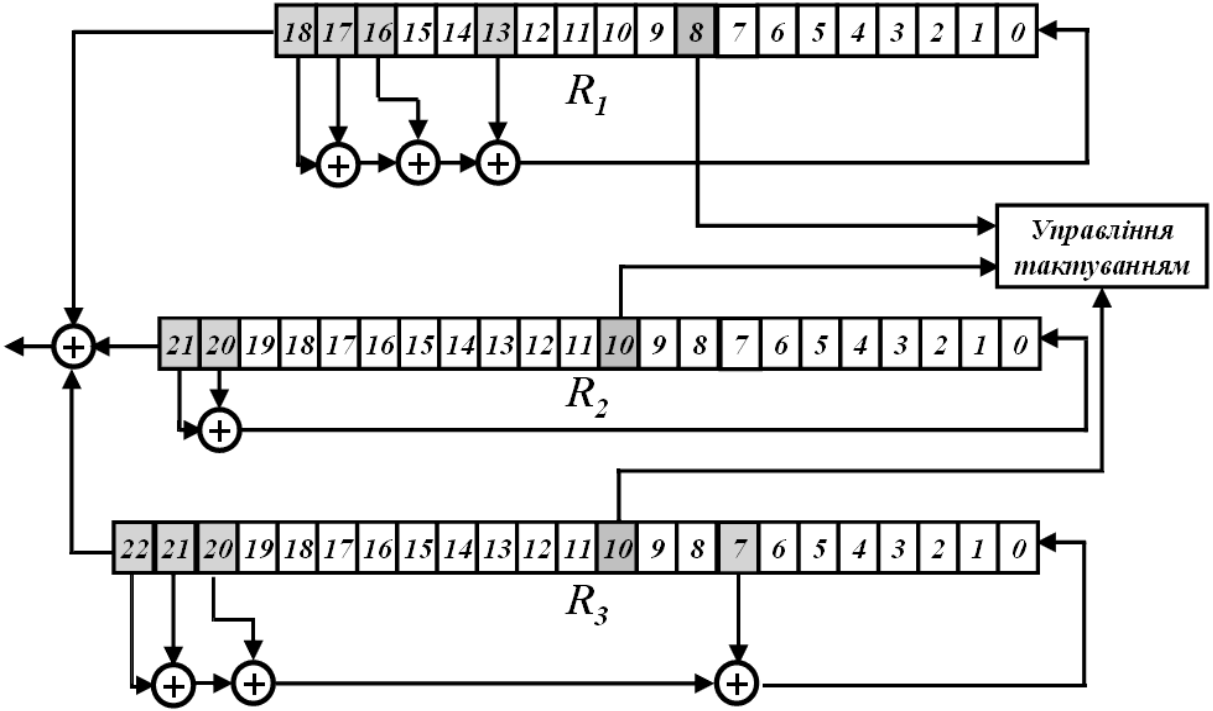


Рис. 4.6. Структура система *LFSR* в алгоритмі *A5/1*

Ініціалізація. Ініціалізація виконується для кожного кадру зашифрування (або розшифрування) і використовує ключ засекречування на *64* біти і *22* біти відповідного номера кадру. Під час ініціалізації виконуються такі кроки:

1. Спочатку всі біти в трьох лінійних регістрах зсуву встановлюються в *0* (обнуляються).

2. Ключ на *64* біти змішується із значенням регістра згідно з наступним псевдокодом (кожний лінійний регістр зсувається на один крок, тобто забезпечується синхронізація):

```
for (i = 0 to 63)
{
    Складання за модулем 2 ключа K[i] з крайніми лівими бітами
    всіх трьох регістрів.
    Синхронізація всіх трьох лінійних регістрів зсуву
}
```

3. Повторити попередній процес, але використовувати *22*-бітовий кадр відповідно до такого псевдокоду:

```
for (i = 0 to 21)
{
    Складання за модулем 2 номера кадру (i) з крайніми
    лівими бітами всіх трьох регістрів.
    Синхронізація всіх трьох лінійних регістрів зсуву
}
```

4. Протягом *100* циклів синхронізується весь генератор згідно з таким псевдокодом, при цьому використовується *мажоритарна функція* для того щоб визначити, який лінійний регістр зсуву повинен бути синхронізований:

```
for (i = 0 to 99)
{
    Синхронізація всього генератора на основі
    мажоритарної функції
}
```

Зауважимо, що іноді синхронізація тут означає, що два, а то й всі три лінійних регістри зсуву зсуваються.

Мажоритарна функція. Значення мажоритарної функції (*f*) з параметрами (*x*, *y*, *z*) дорівнює *1*, якщо значення більшості бітів — *1*;

якщо це — *0*, то її значення — *0*. Наприклад, *f(1,0,1) = 1*, але *f(0,0,1) = 0*. Значення мажоритарної функції визначається перед надходженням тактового імпульсу; три вхідні біти названі синхронізуючими бітами: якщо крайній правий біт дорівнює нулю, це — біти лінійних регістрів $R_1[10]$, $R_2[11]$ і $R_3[11]$. Необхідно звернути увагу на те, що в літературі ці біти $R_1[8]$, $R_2[10]$ і $R_3[10]$ відраховують зліва (як це показано на рис. 4.6). Будемо розглядати біти лінійних регістрів $R_1[10]$, $R_2[11]$ і $R_3[11]$, справа. Ця угода (умова, узгодженість) відповідає місцю біта в характеристичному поліномі.

Ключові біти потоку. Генератор ключів створює ключовий потік в один біт за кожного тактового імпульсу. Перш ніж ключ буде створений, обчислюється мажоритарна функція. Потім кожний лінійний регістр зсуву синхронізується, якщо його біт синхронізації відповідає результату мажоритарної функції; інакше — він не синхронізується.

Приклад 4.4. У деякий момент часу біти синхронізації $R_1[10]$, $R_2[11]$ і $R_3[11]$ дорівнюють *1*, *0* і *1* відповідно. Яким має бути зміст регістрів зсуву?

Розв'язання

Результат мажоритарної функції: *f(1,0,1) = 1*. Отже, зміст регістрів R_1 і R_3 зсувається, а R_2 — ні.

Зашифрування/розшифрування. Розрядні потоки, створені генератором ключів, записуються в буфер, щоб надалі сформувати ключ на *228* бітів, який потім складається за модулем *2* з кадром вихідних даних, щоб створити кадр зашифрованих даних. Водночас робиться зашифрування/розшифрування одного кадру.

До алгоритму *A5/2* додано ще один регістр на *17* бітів (R_4), який управляє рухом (тактуванням) інших (рис. 4.7).

Зміни структури такі:

- доданий регістр R_4 довжиною *17* бітів із багаточленом зворотного зв'язку для R_4: $x^{17} + x^{10} + 1$;

- управління тактуванням здійснює регістр R_4, біти якого — *3*, *7*, *10* є бітами синхронізації;

- мажоритарна функція, яка обчислюється:

$$f = x \,\&\, y \mid x \,\&\, z \mid y \,\&\, z,$$

де & — булеве *and*; | — булеве *or*; *x*, *y* і *z* — біти синхронізації R_4 (*3*-й біт), R_4 (*7*-й біт) і R_4 (*10*-й біт) відповідно.

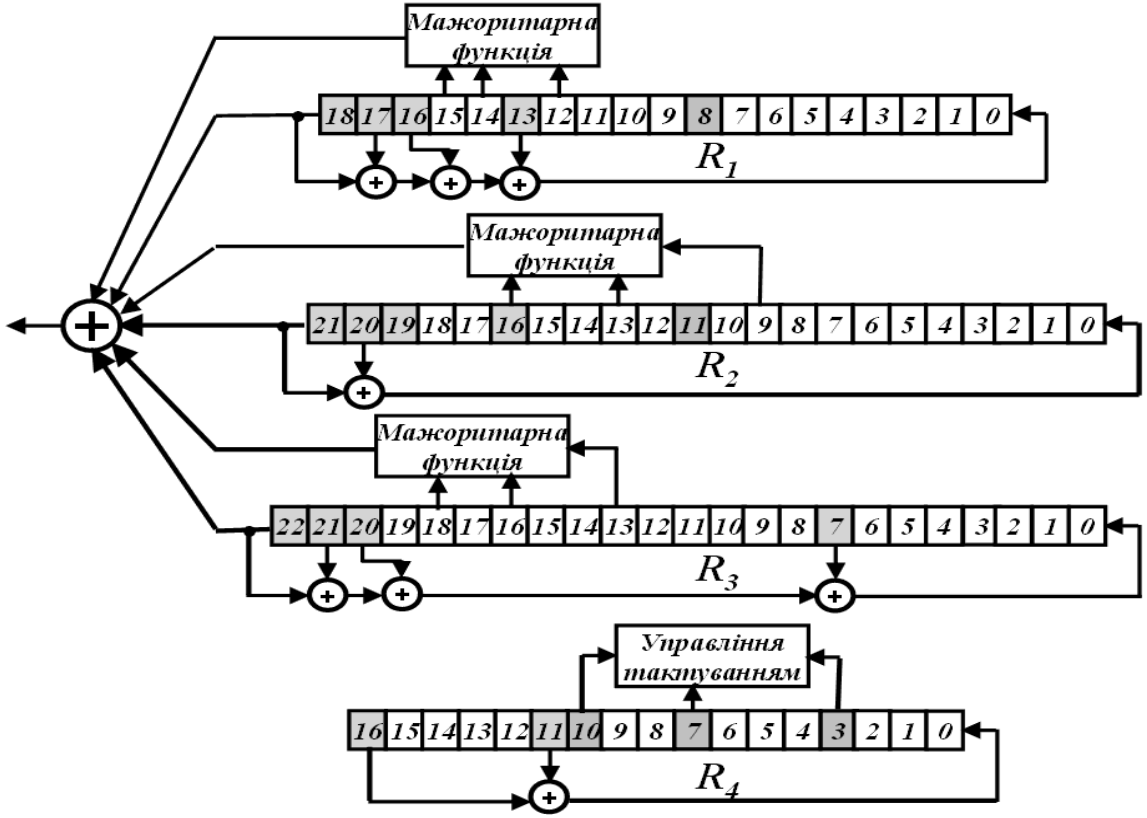


Рис. 4.7. Структура системи регістрів в алгоритмі *A5/2*

Зсув бітів у регістрах здійснюються, якщо:

- біти регістра R_1 зсуваються, якщо $R_4(10) = f$;
- біти регістра R_2 зсуваються, якщо $R_4(3) = f$;
- біти регістра R_3 зсуваються, якщо $R_4(7) = f$.

Фактично, зсуваються біти тих регістрів, сінхробіт яких належить більшості.

Вихідним бітом системи є результат операції *xor* над старшими бітами регістрів і мажоритарних функцій від певних бітів регістрів:

- регістр R_1 — біти: *12*, *14* і *15*;
- регістр R_2 — біти: *9*, *13* і *16*;
- регістр R_3 — біти: *13*, *16* і *18*.

Зміни у функціонуванні неістотні та стосуються тільки ініціалізації:

- *64 + 22* такти заповнюється сесійним ключем і номером кадру також R_4;
- один такт: $R_4(3)$, $R_4(7)$ і $R_4(10)$ заповнюються одиницями;
- *99* тактів з управлінням зсуву регістрів, але без генерації послідовності.

Видно, що ініціалізація займає такий самий час (*100* тактів без генерації розбиті на дві частини).

Розроблений 2001 р. алгоритм *A5/3* повинен змінити *A5/1* у третьому поколінні мобільних систем його називають алгоритмом Касумі. При його створенні за основу взято шифр *MISTY* корпорації *Mitsubishi*. У даний час вважається, що *A5/3* забезпечує необхідну стійкість.

Алгоритм *A5/0* не містить шифрування.

4.4.3. Криптографічна стійкість потокового шифру *A5*

Розробляючи стандарт *GSM,* розраховували на 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 спеціальними службами. Вони були зацікавлені в можливості прослуховування телефонних розмов для своїх цілей. Отже, в алгоритм було внесено 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 невідомі деякі недоліки алгоритму [3].

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 зсувом);
- зайва надмірність службової інформації, що шифрується та яка відома криптографічному аналітику;
- понад 40% ключів призводить до мінімальної довжині періоду послідовності, яка генерується, а саме $(2^{23} - 1)\cdot 3/4$ [3];
- спочатку сеансу здійснюється обмін нульовими повідомленнями (по одному кадру);
- в *A5/2* рух здійснюється окремим регістром довжиною *10* бітів.

На основі цих "дірок" в алгоритмі, побудовано схеми зламу.

Ключем є сесійний ключ довжиною *64* біти, номер кадру вважається відомим. Отже, 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 третього заповнення.

1997 р. Йован Голіч опублікував результати аналізу *A4*. Він запропонував спосіб визначення початкового заповнення регістрів за відомим відрізком гами довжиною лише *64* біти, який отримують із нульових повідомлень. Атака має середню складність 2^{40} [19].

1999 р. *Вагнеру* і *Голдбергу* без зусиль вдалося продемонструвати, що для розкриття системи достатньо перебором визначити початкове заповнення R_4. Перевірка здійснюється завдяки нульовим кадром. Складність цієї атаки дорівнює 2^{17}, отже, на сучасному комп'ютері злам шифру складає кілька секунд.

У грудні 1999 р. група ізраїльських вчених (*Аді Шамір*, *Алекс Бірюков*, а пізніше й американець *Девід Вагнер*) опублікували нетривіальний, але теоретично дуже ефективний метод зламу *A5/1*.

4.5. ПОТОКОВИЙ ШИФР RC4

4.5.1. Історія створення потокового шифру RC4

Алгоритм *RC4* розробив Рональд Лінн Рівест 1987 р. спеціально як генератор потоку ключової інформації з ключем змінної довжини. Хоча офіційне скорочення — *Rivest Cipher 4*, його часто вважають скороченням від *Ron's Code* [6, 30].

Шифр був комерційною таємницею, але у вересні 1994 р. його опис було анонімно надіслано в розсилку *Cypherpunks* [6, 30]. Незабаром опис *RC4* було опубліковано в ньюс-групі *sci.crypt*. Саме звідти вхідний 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 внаслідок аналізу виконуваного коду. Опублікований шифр сумісний із наявними продуктами, які використовують *RC4*, а деякі учасники телеконференції, що мали за їх словами, доступ до вхідного 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 із боку власника торговельної марки.

Головними факторами, що сприяли широкому застосуванню *RC4*, були простота його апаратної й програмної реалізації, а також висока швидкість роботи алгоритму в обох випадках.

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 довжиною ключа до *40* бітів. *56*-бітні ключі дозволено використовувати закордонним відділенням американських компаній.

4.5.2. Опис алгоритму *RC4*

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], ..., S[254], S[255].

Зауважимо, що індекси діапазону елементів — між *0* і *255*. Зміст кожного елемента — байт (*8* бітів), який може інтерпретуватися як ціле число від *0* до *255*.

Рис. 4.8 показує принцип побудови шифру потоку *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.

Алгоритм *RC4* включає в себе два етапи. На першому, підготовчому, етапі проводиться ініціалізація таблиці замін *S* (матриці стану), матриці ключа *K*, а також початкова перестановка матриці стану, заснована на значенні байтів у *K[i]*. На другому, основному, етапі обчислюються безпосередньо псевдовипадкові числа *k*.

Ініціалізація матриці станів і 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*), наведено на рис. 4.9. Цей алгоритм використовує ключ, збережений у *Key*, і має довжину *L* байтів. Ініціалізація починається із заповнення масиву *S*, далі цей масив перемішується шляхом перестановок, визначених ключем. Оскільки одна дія виконується над *S*, то повинно виконуватися твердження, що *S* завжди містить усі значення кодового слова.

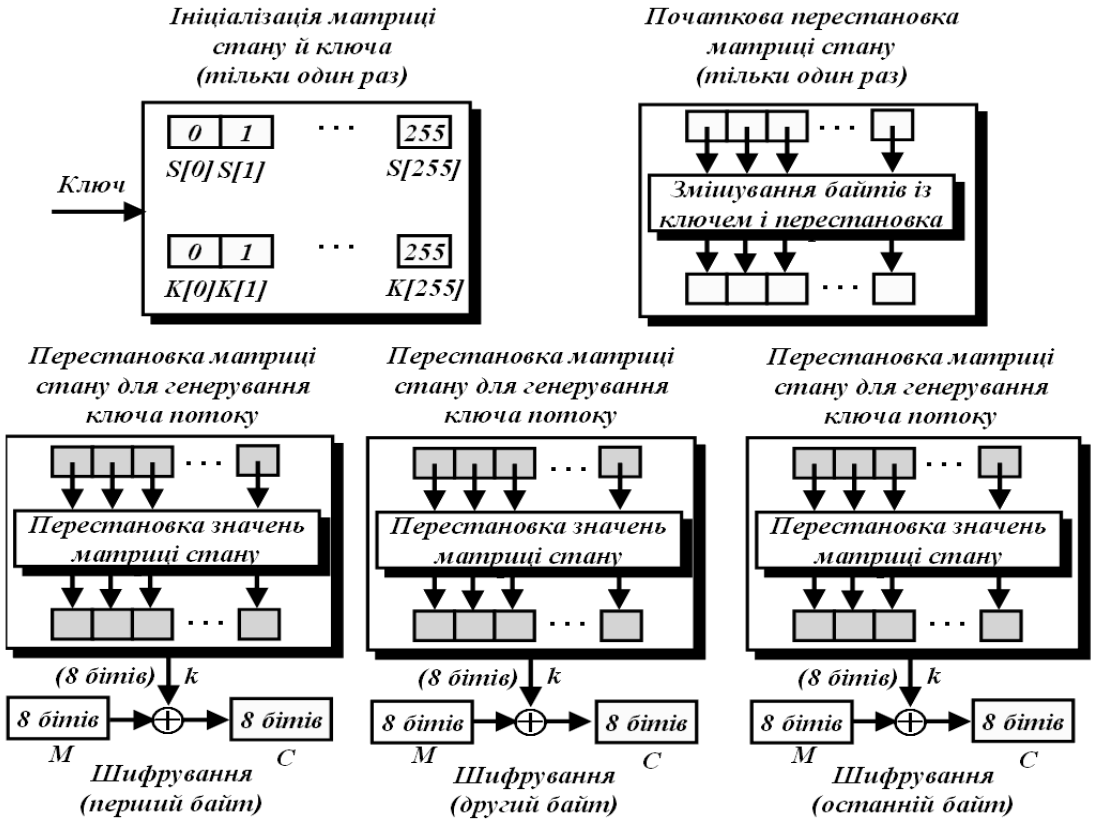


Рис. 4.8. Принцип побудови шифру потоку *RC4*

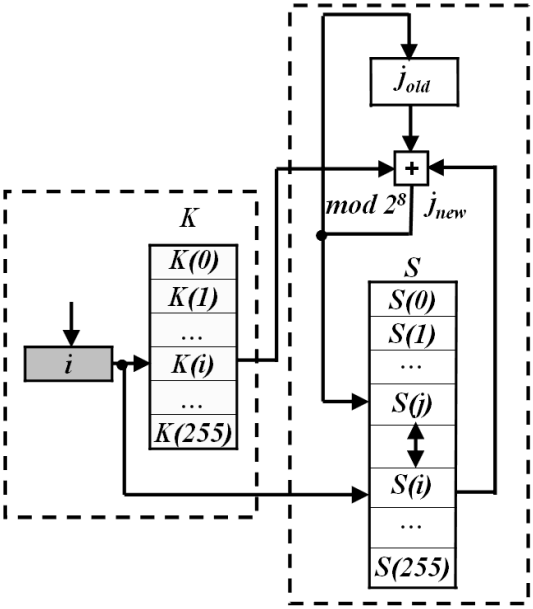


Рис. 4.9. Схема ініціалізації таблиці замін *S* і масиву ключа *K*

Матриця станів ініціалізується для значень *0*, *1*, *2*, ..., *254, 255*. Створюється також масив ключів *K[0], K[1], K[2], ..., K[254], K[255]*. Якщо ключ шифрування має точно *256* байтів, байти копіюються в масив *K*; інакше — байти повторюються, поки не заповниться масив *K*.

Спочатку *S* заповнюється послідовними значеннями від *0* до *255*. Потім кожний черговий 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 і масив ключів:

```
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 і 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 як ключ *k*. Наступний код повторюється для кожного байта початкових даних тексту, щоб створити новий ключовий елемент у ключовому потоці.

Ядро алгоритму складається з функції генерації ключового потоку (рис. 4.10).

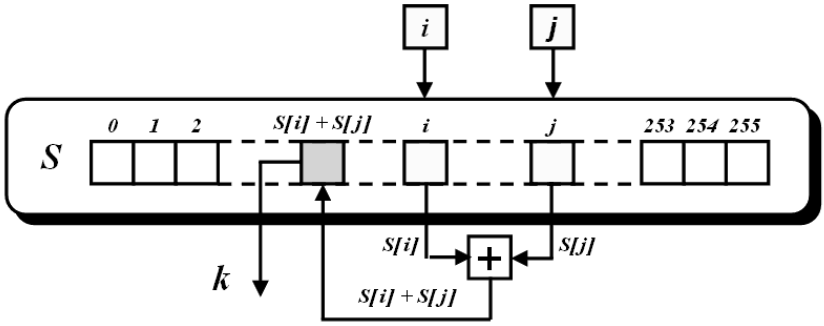


Рис. 4.10. Ядро алгоритму *RC4*

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.

Змінні *i* та *j* ініціалізувалися в *0* перш, ніж буде проведена перша ітерація, але значення копіюється від однієї ітерації до наступної.

Нижче показано програму на псевдокоді, яка здійснює перестановку байтів матриці станів *S[i]* і генерує ключовий потік *k*:

```
i ← 0
j ← 0
i ← ( i + 1 ) mod 256
j ← ( j + S[i] ) mod 256
swap ( S[i], S[j] )
k ← S[ (S[i] + S[j]) mod 256 ]
```

Зашифрування (розшифрування) даних

Після того як ключовий потік *k* був створений, байт відкритих даних зашифровується за допомогою *k*, щоб створити байт зашифрованих даних. Розшифрування являє собою обернений процес.

Нижче показано програму на псевдокоді, яка показує процес шифрування даних для *RC4*:

```
RC4_Encryption ( Key )
{
    // Створення початкової матриці станів
    // і ключових байтів
    for ( i = 0 to 255 )
    {
            S[i] ← i
            K[i] ← Key[ i mod LengthKey ]
    }
    // Перестановка байтів матриці станів на
    // основі значень байта ключа
    j ← 0
    for ( i = 0 to 255 )
    {
            j ← ( j + S[i] + K[i] ) mod 256
            swap ( S[i], S[j] )
    }
```

```
        // Безперервна перестановка байтів матриці
        // станів, генерація ключового потоку і
        // шифрування даних
        i ← 0;
        j ← 0
        while (поки є байти даних для шифрування)
        {
                i ← ( i + 1 ) mod 256
                j ← ( j + S[i] ) mod 256
                swap ( S[i], S[j] )
                k ← S[ (S[i] + S[j]) mod 256 ]
                // Ключ готовий, шифрування даних
                input M
                C ← M ⊕ k
                output C
        }
}
```

Розшифрування даних полягає в регенерації ключового потоку (*k*) та складанні його і зашифрованих даних (*C*) за модулем *2*. Завдяки властивостям підсумовування за модулем *2* на виході отримаємо вихідні дані (*M*).

Приклад 4.5. 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 4.6. Повторимо приклад 4.5, але нехай ключ 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 є, фактично, 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.

Два лічильники, обидва за $n = 4$, *4*-бітові. Позначимо їх як *i* та *j*. Усі обчислення проводяться за модулем 2^n.

Приклад 4.7. Нехай початкове значення ключа *Key = { 12, 2, 3, 8}*. Показати процес ініціалізації матриці станів і масиву ключів, а також початкову перестановку матриці станів.

Розв'язання

На рис. 4.11 показано приклад ініціалізації *4*-розрядних матриці станів і масиву ключів та початкову перестановку матриці станів (*9* тактів з *16*).

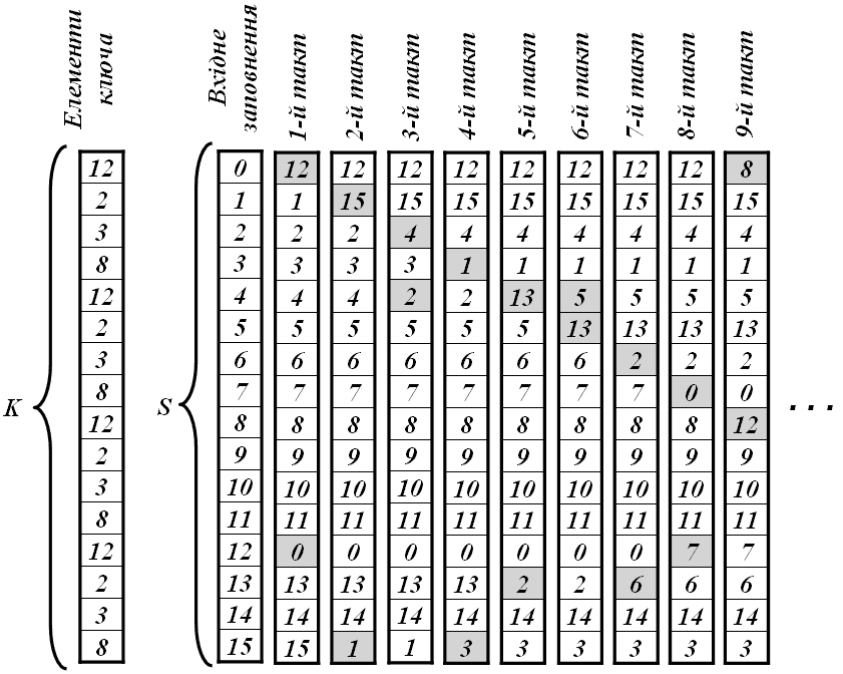


Рис. 4.11. Послідовність тактів перестановки матриці станів

Приклад 4.8. Показати приклад роботи *4*-розрядного генератора ПВЧ *RC4* за заданих значень *i* та *j* і заповненні *4*-розрядної таблиці замін *S*.

Розв'язання

Приклад роботи *4*-розрядного генератора ПВЧ *RC4* зображено на рис. 4.12.

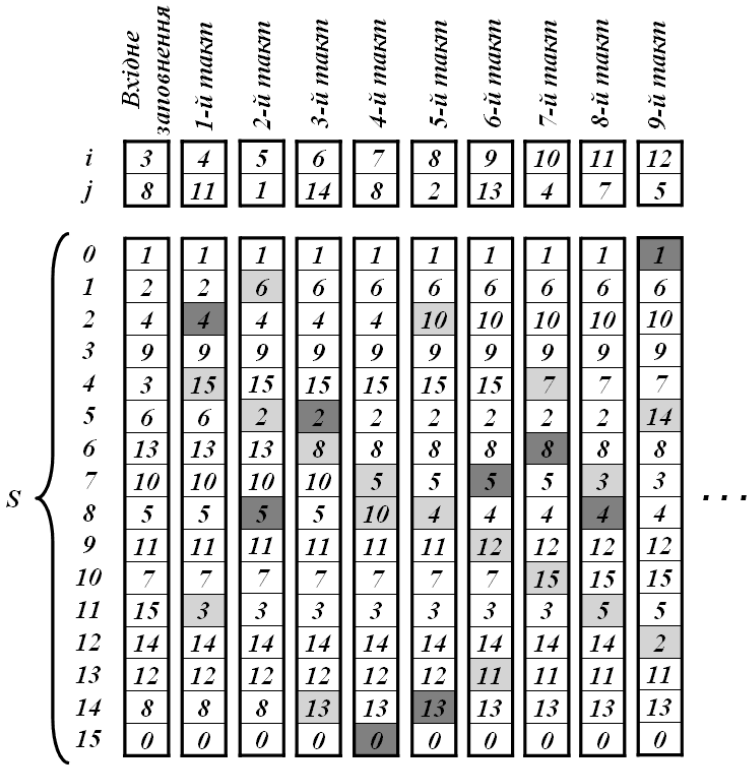


Рис. 4.12. Приклад роботи *4*-розрядного генератора ПВЧ *RC4*

На виході генератора ПВЧ *RC4* формується *4*-розрядна послідовність: *4*, *5*, *2*, *0*, *13*, *5*, *8*, *4*, *1*, … .

4.5.3. 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 "грубої сили").* 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.* Поділяються 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 імовірністю 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).

Потоковий шифр — це шифр, який виконує шифрування вхідного повідомлення по одному біту (або байту) за операцію. Потоковий алгоритм шифрування усуває необхідність розбивати повідомлення на ціле число блоків. Отже, якщо передається потік символів, кожний символ може шифруватися й передаватися відразу. Потокові шифри використовуються для шифрування даних у режимі реального часу.

Як генератори ключів у потокових шифрах можуть використовуватися генератори ПВЧ. Метою використання генераторів ПВЧ є отримання "*нескінченного*" ключового слова при використанні відносно малої довжини самого ключа. Для використання з криптографічною метою ГПВЧ повинен володіти деякими властивостями, наприклад, період послідовності, породжуваної генератором, повинен бути дуже великим.

Найпростішими ГПВЧ є: лінійний конгруентний генератор, генератор за методом Фібоначчі із запізнюваннями, генератор на основі алгоритму *BBS*.

Контрольні питання та завдання

1. Чим потоковий шифр відрізняється від блочного?
2. Яким чином організовується шифрування потоку даних змінної довжини?
3. Які числа називають псевдовипадковими?
4. Назвати існуючі ГПВЧ. Які властивості повинен мати ГПВЧ для використання з криптографічною метою?
5. Перерахувати основні характеристики, переваги й недоліки кожного ГПВЧ.
6. Як для отримання псевдовипадкових чисел можуть використовуватися регістри зсуву зі зворотним зв'язком? Пояснити їх принцип роботи.
7. У чому різниця між генераторами випадкових і псевдовипадкових чисел?
8. Чи можна використовувати генератор справжніх випадкових чисел для отримання гами за потокового шифрування?
9. З якою криптографічною метою можуть бути використані генератори справжніх випадкових чисел?
10. Визначити послідовність із перших десяти чисел і період лінійного конгруентного ГПВЧ для різних параметрів a, b і c (k_0 прийняти таким, що дорівнює — *0*):

а) $a = 5$, $b = 7$, $c = 17$;

б) $a = 6$, $b = 3$, $c = 23$.

11. Обчислити послідовність із десяти чисел, що генерується методом Фібоначчі із запізненням, починаючи з k_a, за таких вихідних даних:

а) $a = 3$, $b = 1$, $k_0 = 0{,}6$; $k_1 = 0{,}3$; $k_2 = 0{,}5$;

б) $a = 4$, $b = 2$, $k_0 = 0{,}9$; $k_1 = 0{,}3$; $k_2 = 0{,}5$; $k_3 = 0{,}9$.

12. Значення k_0, k_1, k_2, k_3, отримані з допомогою лінійного конгруентного генератора, дорівнюють: $k_0 = 1$, $k_1 = 8$, $k_2 = 10$, $k_3 = 9$. Знайти параметри a, b і c ГПВЧ.

13. Обчислити x_{11} за методом генерації ПВЧ *BBS*, якщо: а) $p = 19$, $q = 23$, $x = 3$; б) $p = 23$, $q = 31$, $x = 3$.

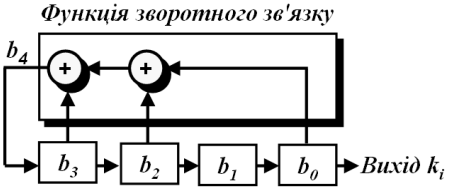
14. Обчислити псевдовипадкову двійкову послідовність довжиною *12* бітів за методом генерації ПВЧ *BBS*, якщо: а) $p = 19$, $q = 23$, $x = 3$; *б)* $p = 23$, $q = 31$, $x = 3$. Як випадкові біти використовувати молодший біт у двійковому записі числа починаючи з x_0.

15. Обчислити значення $x_1 - x_8$ за методом генерації ПВЧ *BBS*, якщо: а) $p = 19$, $q = 23$, $x = 3$; б) $p = 23$, $q = 31$, $x = 3$.

16. Для потокового алгоритму *RC4* показати перші *20* елементів ключового потоку, якщо ключ сеансу — *7* байтів зі значеннями *1*, *2*, *3*, *4*, *5*, *6* і *7*.

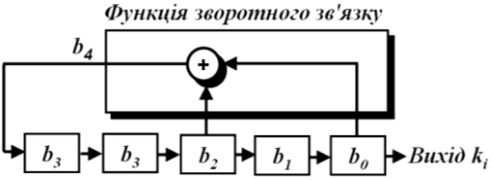
17. Показати *LFSR* із характеристичним поліномом $x^5 + x^2 + 1$. Який максимальний період послідовності, що формується даними *LFSR*?

18. Визначити характеристичний поліном представленого *LFSR*



Який максимальний період послідовності, що формується даними *LFSR*?

19. Визначити ключовий потік на *20* бітів, який буде згенерований представленим *LFSR*



якщо початкове значення послідовності – $(1110)_2$.

20. Максимальна довжина періоду *LFSR* — *32* розряди. Скільки розрядів має регістр зсуву *LFSR*?

21. Для потокового алгоритму *A5/1* знайти максимальний період кожного лінійного регістра зсуву.

22. Для потокового алгоритму *A5/1* знайти значення функцій: а) $f(x, y, z) = f(1, 0, 0)$; б) $f(x, y, z) = f(0, 1, 1)$; в) $f(x, y, z) = f(0, 0, 0)$; г) $f(x, y, z) = f(1, 1, 1)$. У кожному випадку показати, скільки синхронізується лінійних регістрів зсуву.

Розділ 5

СТАНДАРТ СИМЕТРИЧНОГО АЛГОРИТМУ БЛОКОВОГО ШИФРУВАННЯ ДАНИХ DATA ENCRYPTION STANDARD

5.1. ІСТОРІЯ СТВОРЕННЯ СТАНДАРТУ

Стандарт шифрування даних (*Data Encryption Standard — DES*) — блоковий шифр із симетричними ключами, розроблений Національним Інститутом Стандартів і Технології (*National Institute of Standards and Technology — NIST*) [6, 22, 31].

1973 р. *NIST* видав запит на розробку пропозиції національної криптографічної системи із симетричними ключами.

Запропонована *IBM* модифікація проекту, названа "*Люцифер*" (*Lucifer*), була прийнята як *DES*. *DES* опубліковано в ескізному вигляді у Федеральному Регістрі в березні 1975 р. як Федеральний Стандарт Обробки Інформації (*Federal Information Processing Standard — FIPS*) [6, 22, 31].

Після публікації ескіз строго критикувався з двох причин: перша — критикували сумнівно маленьку довжину ключа (тільки *56* бітів), що могло зробити шифр уразливим до атаки "*грубої сили*"; друга — критики були стурбовані деякою прихованою побудовою внутрішньої структури *DES*. Вони підозрювали, що деяка частина структури (*S*-блоки) може мати приховану "лазівку", яка дозволить розшифровувати повідомлення без ключа. Згодом проектувальники *IBM* повідомили, що внутрішню структуру було допрацьовано, щоб запобігти можливості криптографічного аналізу.

DES був нарешті виданий як *FIPS 46* у Федеральному Регістрі в січні 1977 р. Проте *FIPS* оголосив *DES* як стандарт для використання неофіційно.

Алгоритм, покладений в основу стандарту, поширювався досить швидко, і вже 1980 р. був схвалений *NIST США*. З цього часу *DES* перетворився на стандарт не лише за назвою (*Data Encryption*

Standard), але й фактично: з'явилося програмне забезпечення та спеціалізовані *мікроЕОМ*, призначені для зашифрування й 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 термінах *ANSI DEA* визначений як стандарт *X9.32*.

У документах *NIST* криптографічна система *DES* називається алгоритмом шифрування даних (*DEA*), а міжнародна організація за стандартизації, посилаючись на шифр *DES*, користується абревіатурою *DEA-1* [6, 22, 31].

Цей алгоритм був світовим стандартом упродовж більш ніж двадцяти років і затвердився як перший офіційний алгоритм, доступний усім бажаючим. 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* (триразово повторений алгоритм *DES*) для майбутніх застосувань.

Основні вимоги, поставлені розробникам *DES*, до алгоритму такі:

- повинен забезпечувати високий рівень безпеки;
- повинен бути повністю визначений і легко зрозумілий;
- безпека алгоритму повинна ґрунтуватися на ключі й не повинна залежати від збереження в таємниці самого алгоритму;
- повинен бути доступний усім користувачам;
- повинен дозволяти адаптацію до різних застосувань;
- повинен дозволяти економічну реалізацію у вигляді електронних приладів;
- повинен бути ефективним у використанні;
- повинен надавати можливості перевірки;
- повинен бути дозволений для експорту.

Основні переваги алгоритму *DES*:

- використовується тільки один ключ завдовжки *64* біти (*56* бітів довжина ключа й *8* контрольних розрядів);
- зашифрувавши повідомлення за допомогою одного пакета програм, для розшифрування можна використати будь-який інший пакет програм, що відповідає стандарту *DES*;

- відносна простота алгоритму забезпечує високу швидкість обробки;
- досить 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 й перестановки та складається з послідовності блоків перестановки й підстановки, що чергуються. У ній використовувався ключ завдовжки *128* бітів, що управляв станами блоків перестановки й підстановки. Система "Люцифер" виявилася надто складною для практичної реалізації через відносно малу швидкість шифрування (2190 *байт/с* — програмна реалізація й 96970 *байт/с* — апаратна реалізація).

5.2. ПРИНЦИПИ ПОБУДОВИ АЛГОРИТМУ

Як показано на рис. 5.1, *DES* – криптографічна система блокового симетричного зашифрування й розшифрування даних.

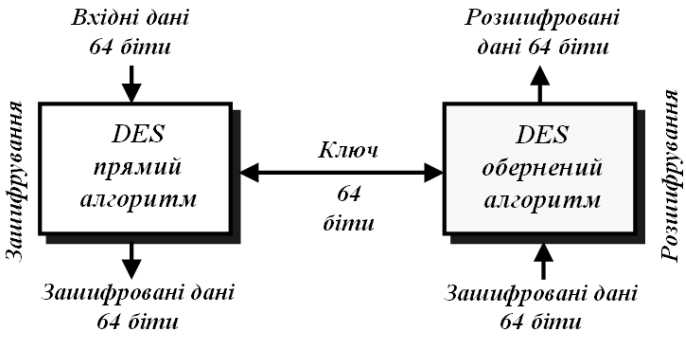


Рис. 5.1. Зашифрування та розшифрування даних
із використанням стандарту *DES*

На стороні зашифрування *DES* приймає *64*-бітовий початковий блок даних і породжує *64*-бітовий зашифрований блок даних; на стороні розшифрування *DES* приймає *64*-бітовий блок зашифрованих даних і породжує *64*-бітовий блок розшифрованих даних. На обох сторонах для зашифрування й розшифрування застосовується той самий *64*-бітовий ключ.

Алгоритм *DES* також використовує комбінацію підстановок і перестановок. *DES* здійснює зашифрування *64*-бітових блоків даних за допомогою *64*-бітового ключа, в якому значущими

є *56* бітів (кожний восьмий біт використовується для контролю парності інших бітів ключа) [22]. Розшифрування в *DES* є операцією, оберненою зашифруванню, і виконується шляхом повторення операцій зашифрування у зворотній послідовності.

5.3. СТРУКТУРА АЛГОРИТМУ ШИФРУВАННЯ ДАНИХ

Узагальнену схему процесу зашифрування в алгоритмі *DES* показано на рис. 5.2.

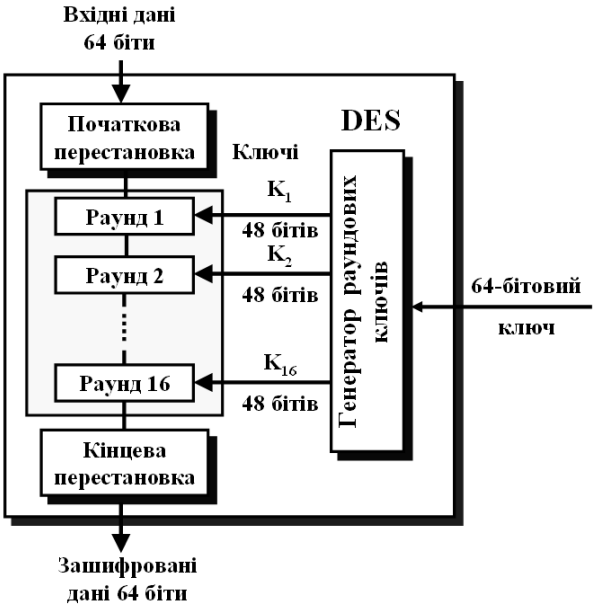


Рис. 5.2. Узагальнена схема процесу шифрування в алгоритмі *DES*

Процес зашифрування полягає в початковій перестановці бітів *64*-бітового блока, шістнадцяти раундах зашифрування і, нарешті, у кінцевій перестановці бітів.

Використання шістнадцяти раундів зашифрування обумовлене такими умовами:

- дванадцять раундів є мінімально необхідними для забезпечення належного рівня криптографічного захисту;

- за апаратної реалізації використання шістнадцяти раундів дозволяє повернути перетворений ключ у початковий стан для подальших перетворень;

- ця кількість раундів також потрібна, щоб унеможливити проведення атаки на блок зашифрованих даних із двох сторін.

Кожний раунд використовує різні (шістнадцять) згенеровані *48*-бітові ключі.

У деяких реалізаціях *DES* блоки відкритого повідомлення, перед тим, як вони будуть завантажені в регістри самого облаштування зашифрування, проходять процедуру початкової перестановки. Ця процедура застосовується для того, щоб здійснити початкове розсіювання статистичної структури повідомлення [22].

У разі використання початкової перестановки (*Initial Permutation — IP*), після завершення шістнадцяти раундів, до отриманого блоку застосовується обернена перестановка IP^{-1}.

IP перестановка бітів даних є провідною комутацією з 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 використати таке саме програмне або апаратурне забезпечення для двох сторін процесу: зашифрування й розшифрування. Рис. 5.3 показує початкові й кінцеві перестановки.

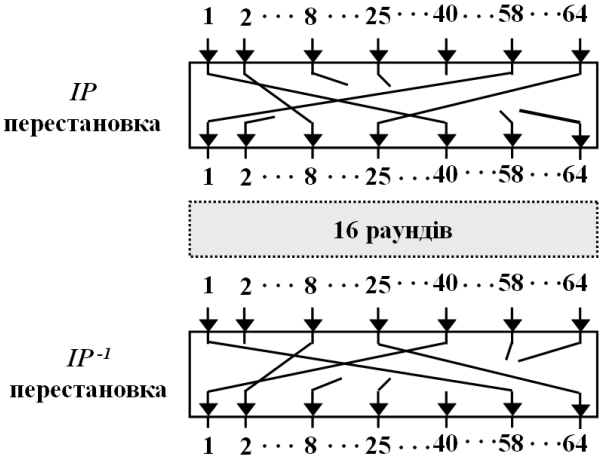


Рис. 5.3. Початкові та кінцеві кроки перестановки *DES*

Кожна з перестановок приймає на вхід *64*-бітові блоки даних і переставляє його елементи за заданим правилом. На рис. 5.3 показано тільки невелику кількість вхідних портів і відповідних вихідних портів. Ці перестановки — прямі перестановки без ключів, які інверсні одна одній. Наприклад, у початковій перестановці (*IP*) *58*-й біт на вході переходить у перший біт на виході. Аналогічно, у кінцевій перестановці (IP^{-1}) 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 кінцевою перестановкою.

Правила початкової *IP* та кінцевої IP^{-1} перестановок для *64*-бітового блока даних надано в табл. 5.1 і 5.2.

Таблиця 5.1

Таблиця початкової перестановки *IP*

	Номери бітів															
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 5.2

Таблиця кінцевої перестановки IP^{-1}

	Номери бітів															
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*

Ця заміна (IP і IP^{-1}) ніяк не впливає на стійкість шифру, і користувачі часто ставили питання, навіщо її взагалі робити. Один із членів творчого колективу розробників *DES* стверджував, що вона полегшує апаратурну реалізацію процедури шифрування [1, 44, 45].

Усі перестановки й коди в таблицях підібрані розробниками так, щоб максимально ускладнити процес розшифрування шляхом підбору ключа.

Слід відразу зазначити, що всі наведені таблиці є стандартними і їх слід включати в реалізацію алгоритму *DES* у незмінному вигляді.

Приклад 5.1. Необхідно визначити результат на виході початкового блока перестановки, коли на його вхід надходить шістнадцяткова послідовність, така як:

$$0002\ 0000\ 0000\ 0001_{16}.$$

Розв'язання

Вхід має тільки дві одиниці (біт *15* і біт *64*); вихід повинен також мати тільки дві одиниці, оскільки використовується пряма перестановка. Використовуючи табл. 5.1, можна знайти вихід, пов'язаний із цими двома бітами. Біт *15* на вході стає бітом *63* на виході. Біт *64* на вході стає бітом *25* на виході. На виході будемо мати тільки дві одиниці — біт *25* і біт *63*.

Отже, результат у шістнадцятковому численні:

$$0000\ 0080\ 0000\ 0002_{16}.$$

Приклад 5.2. Довести, що початкові та кінцеві перестановки, отримані в прикладі 5.1, інверсні одна одній.

Розв'язання

Перетворимо отриману вихідну послідовність у вхідну:

$$0000\ 0080\ 0000\ 0002_{16}.$$

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:

$$0002000000000001_{16}.$$

Як видно з результату початкові й кінцеві перестановки — це прямі блоки перестановки, які інверсні одна одній.

5.4. ГЕНЕРАЦІЯ РАУНДОВИХ КЛЮЧІВ

Генератор ключів створює шістнадцять ключів по *48* бітів із ключа шифру на *56* бітів. Проте ключ шифру зазвичай дається як ключ із *64* бітів, в якому *8* додаткових бітів є бітами перевірки. Вони відкидаються перед фактичним процесом генерації ключів, який показано на рис. 5.4.

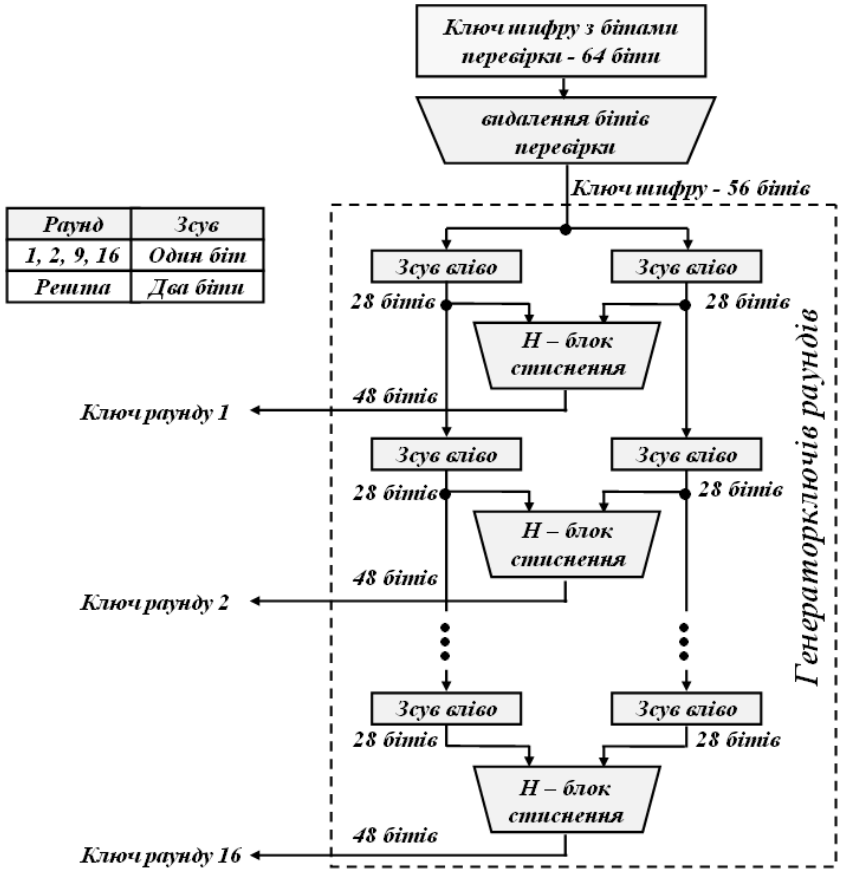


Рис. 5.4. Структурна схема генерації ключів

5.4.1. Видалення бітів перевірки ключової послідовності

Попередній процес перед розширенням ключів — перестановка стиснення, який називають *видаленням бітів перевірки*. Він видаляє біти парності (біти *08, 16, 24, 32, 40, 48, 56, 64*) із *64*-бітового ключа. Ці біти не впливають на шифрування. Перераховані біти ключа відповідають за те, щоб кожний байт ключа складався з непарної кількості одиничних бітів.

Отже, насправді ключ шифру є *56*-бітовим. Для видалення контрольних бітів і підготовки ключа до роботи використовується функція *G* первинної підготовки ключа (табл. 5.3). Результатом виконання функції *G* буде перестановка, яка в літературі [6, 26] називається *PC-1*.

Таблиця 5.3

Таблиця *G* первинної підготовки ключа

	Номери бітів													
Вхід K	*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 K	*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 K	*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 K	*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*

Табл. 5.3 поділена на дві частини. Результат перетворення *G(K)* поділяється на дві половини C_0 і D_0 по *28* бітів кожна. Перші два рядки табл. 5.3 визначають, як вибираються біти послідовності C_0 (першим бітом C_0 буде *57*-й біт ключа шифру, потім *49*-й біт і так далі, а останніми бітами — *44*-й і *36*-й біт ключа). Наступні два рядки табл. 5.3 визначають, як вибираються біти послідовності D_0 (тобто послідовність D_0 буде складати з бітів *63*, *55*, *47*, ., *12*, *04* ключа шифру).

Приклад 5.3. Необхідно визначити результат на виході функції первинної підготовки ключа *G*, якщо ключ шифру дорівнює

$$K = abab19283746cdcd_{16}.$$

Розв'язання

Замінюючи значення ключа шифру на двійкове

$$1010\ 1011\ 1010\ 1011\ 0001\ 1001\ 0010\ 1000_2$$
$$0011\ 0111\ 0100\ 0110\ 1100\ 1101\ 1100\ 1101_2$$

і використовуючи таблицю G первинної підготовки ключа (табл. 5.3), отримаємо послідовності:

$$C_0 = 1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010_2 = c3c033a_{16};$$

$$D_0 = 0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010_2 = 33f0cfa_{16}.$$

5.4.2. Циклічний зсув уліво послідовностей C_i і D_i

Після прямої перестановки ключ поділено на дві частини по *28* бітів. Після формування послідовностей C_0 і D_0 рекурсивно визначаються послідовності C_i і D_i для кожного раунду з *16* ($i = 1,2,...,16$). Для цього застосовуються операції циклічного зсуву вліво на один або два біти залежно від номера раунду, як показано у табл. 5.4.

Таблиця 5.4

Кількість бітів циклічного зсуву послідовностей C_i і D_i

Номери 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 бітів, що зсуваються															

Операції циклічного зсуву виконуються для послідовностей C_i і D_i незалежно. Наприклад, послідовність C_3 виходить за допомогою циклічного зсуву вліво на дві позиції послідовності C_2, а послідовність D_3 за допомогою циклічного зсуву вліво на дві позиції послідовності D_2. Послідовності C_{16} і D_{16} виходять із C_{16} і D_{16} відповідно за допомогою їх циклічного зсуву вліво на одну позицію.

Загальну кількість циклічних зсувів (загальний циклічний зсув послідовностей C_i і D_i складає *28* бітів) у схемі генерації раундових ключів вибрано так, щоб після формування останнього раундового ключа значення послідовностей C_{16} і D_{16} дорівнювали значенням послідовностей C_0 і D_0.

Приклад 5.4. Необхідно визначити результат після операції циклічного зсуву вліво послідовностей C_0 і D_0 для формування другого раундового ключа k_2, якщо значення C_0 і D_0 дорівнюють:

$$C_0 = 1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010_2 = c3c033a_{16};$$

$$D_0 = 0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010_2 = 33f0cfa_{16}.$$

Розв'язання

Для формування другого раундового ключа k_2, відповідно до табл. 5.4, послідовності C_0 і D_0 повинні бути циклічно зсунуті вліво на два розряди. Внаслідок циклічного зсуву вліво послідовностей C_0 і D_0 на два розряди отримаємо:

$$C_2 = 0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011_2 = 0f00ceb_{16};$$

$$D_2 = 1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000_2 = cfc33e8_{16}.$$

5.4.3. Об'єднання послідовностей C_i і D_i, їх перестановка та стиснення

Після циклічного зсуву послідовностей C_i і D_i, вони об'єднуються, щоб створити блок у *56* бітів, тобто виходить послідовність $C_i||D_i$, де || — знак математичної операції конкатенації (об'єднання).

Перестановка та стиснення (*H*-функція) змінює *56* бітів $C_i||D_i$ на *48* бітів k_i, які використовуються як раундові ключі. Перестановку та стиснення послідовності $C_i||D_i$ показано в табл. 5.5.

Таблиця 5.5

Таблиця кінцевої обробки раундових ключів *H*

Вхід	*14*	*17*	*11*	*24*	*01*	*05*	*03*	*28*	*15*	*06*	*21*	*10*	*23*	*19*	*12*	*04*
Вихід k_i	*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 k_i	*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 k_i	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*

Раундові ключі k_i, які визначаються на кожному кроці ітерації, є результатом вибору конкретних бітів із *56*-бітової послідовності $C_i||D_i$, їх перестановками та стисненням, тобто раундовий ключ:

$$k_i = H(C_i||D_i), \tag{5.1}$$

де H — функція, що завершує обробку ключа завдовжки *48* бітів, що визначається за табл. 5.5. Результатом виконання функції H буде перестановка, яка в літературі [6, 26] називається *PC-2*.

Як випливає з табл. 5.5, першим бітом ключа k_i буде *14*-й біт послідовності $C_i||D_i$, другим — *17*-й біт, *47*-м бітом ключа k_i буде *29*-й біт $C_i||D_i$, а *48*-м бітом — *32*-й біт $C_i||D_i$.

На вході функції H була послідовність $C_i||D_i$ завдовжки *56* бітів, а після її виконання довжина раундового ключа k_i стала такою, що дорівнює *48* бітів. Стиснення сталося через відмову від участі в перестановці за допомогою функції H восьми бітів із послідовності $C_i||D_i$ з номерами *9*, *18*, *22*, *25*, *35*, *38*, *43* і *54* (див. табл. 5.5).

Приклад 5.5. Необхідно визначити результат після операції об'єднання послідовностей C_2 і D_2 для подальшого формування другого раундового ключа k_2, якщо значення C_2 і D_2 дорівнюють:

$$C_2 = 0f00ceb_{16}; \quad D_2 = cfc33e8_{16}.$$

Розв'язання

Об'єднання послідовностей C_2 і D_2 можна здійснити у такий спосіб:

$$C_2||D_2 = 0f00ceb16||cfc33e8_{16} = 0f00cebcfc33e8_{16}.$$

Приклад 5.6. Необхідно згенерувати другий раундовий ключ k_2, тобто визначити результат на виході функції кінцевої обробки раундових ключів H, якщо на її вході об'єднані послідовності C_2 і D_2 дорівнюють:

$$C_2||D_2 = 0f00cebcfc33e8_{16}.$$

Розв'язання

Використовуючи двійкове значення $C_2||D_2$

$$C_2||D_2 = 0f00cebcfc33e8_{16} =$$

$$= 0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011$$

$$1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000_2$$

і зробивши заміну за допомогою табл. 5.5, отримаємо другий раундовий ключ k_2

$$k_2 = 0100\ 0101\ 0110\ 1000\ 0101\ 1000\ 0001\ 1010$$

$$1011\ 1100\ 1100\ 1110_2 = 4568581abcce_{16}.$$

5.5. ПРОЦЕС ШИФРУВАННЯ ДАНИХ

Розглянемо спочатку зашифрування, а потім розшифрування даних у *DES*.

Після здійснення початкової перестановки *IP* над вхідним блоком даних x отриману *64*-бітову послідовність $x_0 = IP(x)$ подано у вигляді:

$$x_0 = L_0 \,||\, R_0,$$

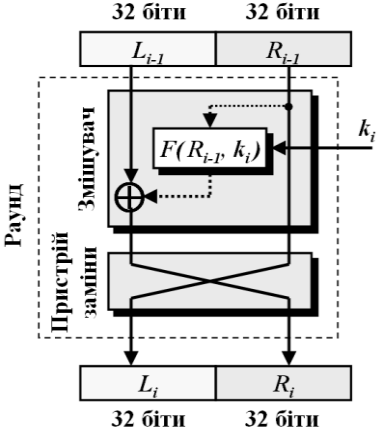


Рис. 5.5. Раунд зашифрування в *DES*

де L_0 — ліва (*left*) і R_0 — права (*right*) послідовності бітів, які мають однакову довжину, що дорівнює *32* бітам.

Безпосередньо процес зашифрування в *DES* складається із шістнадцяти раундів, тому блок даних $x_0 = L_0 \,||\, R_0$ перетвориться далі шістнадцять разів за так званою схемою Фейстеля. Кожний раунд *DES* використовує схему Фейстеля, як це показано на рис. 5.5.

Процес зашифрування даних із використанням шістнадцяти раундів можна представити математично:

$$\begin{aligned} L_i &= R_i; \\ R_i &= L_{i-1} \oplus F(R_{i-1}, k_i),\ i = 1,2,..,16, \end{aligned} \qquad (5.2)$$

де $F(\bullet)$ — функція зашифрування; k_i – i-й розклад ключів; $\oplus$ — операція побітового складання даних за модулем *2*.

Після виконання шістнадцяти раундів зашифрування блок даних $x_{16} = L_{16}||R_{16}$ зазнає оберненої IP^{-1} перестановки. Як результат виходить *64*-бітовий блок зашифрованих даних y

$$y = IP^{-1}(x_{16}) = IP^{-1}(L_{16}||R_{16}). \quad (5.3)$$

Кожний поточний раунд приймає L_{i-1} і R_{i-1} від попереднього раунду (чи початкового блока перестановки) і створює для наступного раунду L_i і R_i, які надходять на наступний раунд (чи кінцевий блок перестановки). Як було вище вказано, можна прийняти, що кожний раунд має два елементи шифру: змішувач і пристрій заміни. Кожний із цих елементів є оберненим. Пристрій заміни очевидно обернений, він міняє місцями ліву половину даних з правою половиною. Змішувач є оберненим, тому що операція *xor* може бути оберненою. Усі необернені елементи зосереджені у функції Фейстеля $F(R_{i-1}, k_i)$.

5.5.1. Функція *DES*

Основний блок *DES* — функція *DES* (рис. 5.6).

Функція *DES* за допомогою *48*-бітового ключа зашифровує *32* крайні праві біти R_{i-1}, щоб отримати на виході *32*-бітове слово. Ця функція містить, як показано на рис. 5.6, чотири секції:

- *E*-блок перестановки й розширення;
- порозрядного підсумовування за модулем *2* ($\oplus$);
- групу *S*-блоків заміни;
- *P*-блок прямої перестановки.

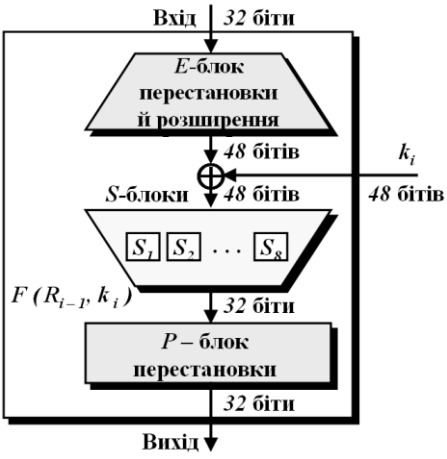


Рис. 5.6. Функція *DES*

Е-блок перестановки й розширення функції *DES*

Оскільки на вході R_{i-1} має довжину *32* біти, а ключ k_i — довжину *48* бітів, спочатку потрібно розширити R_{i-1} до *48* бітів. R_{i-1} розділяється

на *8* секцій по *4* біти. Кожна секція на *4* біти розширюється до *6* бітів. Ця перестановка з розширенням відбувається за заздалегідь визначеними правилами. Для секції значення вхідних бітів *1*, *2*, *3* і *4* привласнюються 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*.

Отже, після *E*-блока перестановки й розширення виходить послідовність R_{i-1}^{E}

$$R_{i-1}^{E} = E(R_{i-1}). \tag{5.4}$$

Рис. 5.7 показує входи й виходи в *E*-блоці перестановки й розширення.

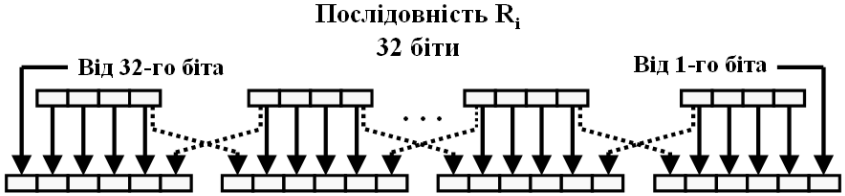


Рис. 5.7. Перестановка й розширення

Хоча відношення між входом і виходом можуть бути визначені математично, але щоб визначити цей *E*-блок, *DES* використовує табл. 5.6.

Таблиця 5.6

Таблиця *E*-блока перестановки й розширення

Вхід	*32*	*01*	*02*	*03*	*04*	*05*	*04*	*05*	*06*	*07*	*08*	*09*	*08*	*09*	*10*	*11*
Вихід R_{i-1}^{E}	*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 R_{i-1}^{E}	*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*	*31*	*31*	*32*	*01*
Вихід R_{i-1}^{E}	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*

Зауважимо, що кількість виходів — *48*, але діапазон значень — тільки від *1* до *32*. Деякі з входів надходять до більше ніж одного виходу. Наприклад, значення вхідного біта *5* стає значенням бітів виходу *6* і *8*.

Приклад 5.7. Необхідно визначити послідовність R_0^E на виході *E*-блока перестановки й розширення, якщо на його вхід надходить послідовність $R_0 = 18ca18ad_{16}$.

Розв'язання

Використовуючи двійкове значення R_0

$$R_0 = 18ca18ad_{16} = 0001\ 1000\ 1100\ 1010\ 0001\ 1000\ 1010\ 1101_2$$

і зробивши заміну за допомогою табл. 5.6, отримаємо:

$$R_0^E = 1000\ 1111\ 0001\ 0110\ 0101\ 0100\ 0000\ 1111$$
$$0001\ 0101\ 0101\ 1010_2 = 8f16540f155_{16}.$$

Порозрядне підсумовування функції DES

Після розширення послідовності R_{i-1} довжиною *32* біти до послідовності R_{i-1}^E завдовжки *48* бітів із використанням *E*-блока перестановки й розширення *DES* використовує операцію *xor* над розширеною частиною правої секції R_{i-1}^E і ключем раунду k_i. Зазначимо, що права секція та ключ мають довжину *48* бітів. Також зазначимо, що ключ раунду використовує тільки цю операцію.

Отже, після виконання операції *xor* отримаємо послідовність $R_{i-1}^{\oplus}$ довжиною *48* бітів:

$$R_{i-1}^{\oplus} = R_{i-1}^E \oplus k_i. \qquad (5.5)$$

Приклад 5.8. Необхідно визначити послідовність $R_0^{\oplus}$ на виході суматора за модулем *2*, якщо на його вхід надходить послідовність $R_0^E = 8f16540f155_{16}$ і раундовий ключ $k_1 = 194cd072de8c_{16}$.

Розв'язання

Представляючи значення R_0^E і k_1 у двійковому вигляді і використовуючи правило порозрядного підсумовування даних за модулем *2*, отримаємо

$$\begin{array}{cll} & R_0^E & 100011110001011001010100000011110001010101011010_2 \\ \oplus & k_1 & 000110010100110011010000011100101101111010001100_2 \\ \hline & R_0^{\oplus} & 100101100101101010000100011111011100101111010110_2 \end{array}$$

Перетворене значення $R_0^{\oplus}$ у шістнадцятирічне набуде вигляду $R_0^{\oplus} = 965a847dcbd6_{16}$.

S-блоки заміни функції DES

S-блоки заміни змішують інформацію (операція перемішування). *DES* використовує *S*-блоки, кожний із *6* вхідними бітами і *4* вихідними (рис. 5.8).

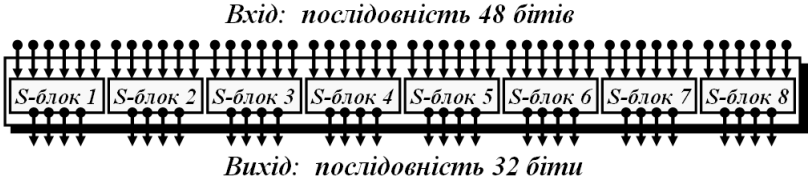


Рис. 5.8. *S*-блоки заміни

Послідовність із *48* бітів від другої операції *DES* $R_{i-1}^{\oplus}$ розділяється на вісім частин по *6* бітів, і кожна частина надходить у відповідний *S*-блок. Результат кожного *S*-блока — дані завдовжки *4* біти; коли вони об'єднуються, то результат буде представлений послідовністю R_{i-1}^S у *32* біти:

$$R_{i-1}^S = S(R_{i-1}^{\oplus}) . \tag{5.6}$$

Для визначення значення R_{i-1}^S обчислену суму $R_{i-1}^{\oplus}$ записується у вигляді конкатенації восьми *6*-бітових слів:

$$R_{i-1}^{\oplus} = B_1 \,||\, B_2 \,||\, B_3 \,||\, B_4 \,||\, B_5 \,||\, B_6 \,||\, B_7 \,||\, B_8 . \tag{5.7}$$

На цьому етапі кожне слово B_j надходить на відповідний S_i-блок підстановки та стискування. Основне завдання *S*-блоків полягає

в перетворенні *48*-бітового вектора в *32*-бітовий. Усього в *DES* використовується вісім *S*-блоків з *6*-бітовими входами і *4*-бітовими виходами. S_j-блок — це матриця розміром 4×16 із цілими елементами в діапазоні від *0* до *15*.

Кожне слово B_j представляється у вигляді:

$$B_j = \underbrace{b_1 || b_6}_{\substack{\text{Номер} \\ \text{рядка}}} || \underbrace{b_2 || b_3 || b_4 || b_5}_{\text{Номер стовпця}}. \tag{5.8}$$

Перший і шостий біти слова B_j, якщо їх розглядати як двійковий запис числа, визначають номер рядка матриці S_j-блока, а чотири інших біти визначають номер стовпця. На перетині певного номера рядка й стовпця заданого S_j-блока знаходиться елемент матриці. Його двійковий запис і є виходом.

Сукупність *6*-бітових блоків (5.7) забезпечує вибір *4*-бітового елемента в кожному з блоків:

$$S = S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8. \tag{5.9}$$

У результаті отримуємо:

$$R_{i-1}^S = S_1(B_1)||S_2(B_2)||S_3(B_3)||S_4(B_4)||S_5(B_5)||S_6(B_6)||S_7(B_7)||S_8(B_8)||.$$

Підстановка (заміна) в кожному *S*-блоці відбувається за заздалегідь визначеними правилами, що ґрунтуються на таблиці з *4* рядків і *16* стовпців, як показано на рис. 5.9.

Оскільки для кожного *S*-блока є власна таблиця, необхідно мати вісім таблиць, наприклад таких, як це показано в табл. 5.7–5.14. Значення входу (номер рядка й номер стовпця) і значення виходу даються

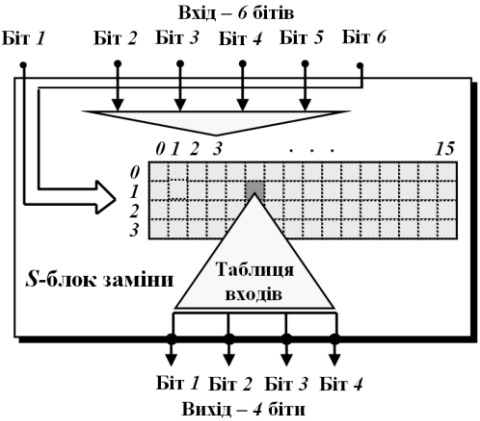


Рис. 5.9. Правила для *S*-блока заміни

як десяткові номери, щоб заощадити місце на сторінці. У реальності вони можуть бути замінені двійковими числами.

Таблиця 5.7

S_1-блок підстановки (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 5.8

S_2-блок підстановки (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 5.9

S_3-блок підстановки (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 5.10

S_4-блок підстановки (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 5.11

S_5-блок підстановки (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 5.12

S_6-блок підстановки (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 5.13

S_7-блок підстановки (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 5.14

S_8-блок підстановки (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 мають лінійний характер. Аналітичне обчислення лінійного перетворення не становить труднощів для криптографічного аналітика.

Нелінійність перетворень, здійснюваних *DES*, визначається тільки *S*-блоками заміни. Їх вибір не має досить детального обґрунтування. Висловлювалися думки про те, що *S*-блоки заміни мають деяку "лазівку", що дозволяє здійснювати контроль за шифрованим обміном повідомленнями.

Офіційна версія така: 1976 р. Національне бюро стандартів США заявило, що вибір *S*-блоків заміни визначений такими вимогами [17, 26]:

- кожний рядок табличного завдання кожного *S*-блока заміни повинен бути перестановкою великої кількості *{0,1, ..., 15}*;

- *S*-блоки заміни не повинні бути лінійними або афінними функціями своїх входів;

- зміна одного біта входу *S*-блока заміни повинна приводити до зміни принаймні двох бітів виходу;

- для кожного *S*-блока заміни і будь-якого входу *x* значення *S(x)* і $S(x \oplus (0,0,1,1,0,0))$ повинні розрізнятися мінімум двома бітами.

Пояснимо процеси підстановки (заміни) за допомогою *S*-блоків на прикладах.

Приклад 5.9. Вхідна послідовність S_1-блока підстановки (заміни) дорівнює *1 0 1 1 1 0*.

Визначити, яка послідовність буде на виході S_1-блока підстановки (заміни).

Розв'язання

Якщо записати перший і шостий біти разом, то отримаємо у двійковому численні 10_2, яке виражається як число *2* за десяткового числення. Частина бітів, що залишається, 0111_2 у двійковому численні є числом **7** у десятковому численні.

На перетині рядка *02* і стовпця *07* у табл. 5.7 (S_1-блок підстановки (заміни)) знаходимо значення результату — *11* у десятковому численні або 1011_2 у двійковому численні. Отже, вхід 101110_2 дає вихід 1011_2.

Приклад 5.10. Вхідна послідовність S_8-блока підстановки (заміни) дорівнює *0 0 0 0 0 0*.

Визначити, яка послідовність буде на виході S_8-блока підстановки (заміни).

Розв'язання

Якщо записати перший і шостий біти разом, то отримаємо у двійковому численні 00_2, яке виражається як число *0* за десяткового числення. Частина бітів, що залишається, 0000_2 у двійковому численні є *0* у десятковому численні.

На перетині рядка *00* і стовпця *00* у табл. 5.14 (S_8-блок підстановки (заміни)) знаходимо значення результату — *13* у десятковому численні або 1101_2 у двійковому численні. Отже, вхід 000000_2 дає вихід 1101_2.

P-блок прямої перестановки функції DES

P-блок перестановки — остання операція у функції *DES* — пряма перестановка з *32* бітами на вході та *32* бітами на виході:

$$R_{i-1}^{P} = P(R_{i-1}^{S}). \qquad (5.10)$$

Відношення "вхід-вихід" для цієї операції показано в табл. 5.15. Вони здійснюються за тими самими загальними правилами, як в попередніх таблицях перестановки. Наприклад, сьомий біт входу стає другим бітом виходу.

Таблиця 5.15

Таблиця *P*-блока прямої перестановки

Вхід R_{i-1}^S	*16*	*07*	*20*	*21*	*29*	*12*	*28*	*17*	*01*	*15*	*23*	*26*	*05*	*18*	*31*	*10*
Вихід R_{i-1}^P	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вхід R_{i-1}^S	*02*	*08*	*24*	*14*	*32*	*27*	*03*	*09*	*19*	*13*	*30*	*06*	*22*	*11*	*04*	*25*
Вихід R_{i-1}^P	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*

Приклад 5.11. Вхідна послідовність *P*-блока прямої перестановки дорівнює $R_0^S = ec5965b5_{16}$.

Визначити, яка послідовність буде на виході *P*-блока прямої перестановки.

Розв'язання

Використовуючи двійкове значення R_0^S:

$$R_0^S = ec5965b5_{16} = 1110\ 1100\ 0101\ 1001\ 0110\ 0101\ 1011\ 0101_2$$

і зробивши заміну за допомогою табл. 5.16, отримаємо:

$$R_0^P = 1000\ 0110\ 1000\ 1101\ 1010\ 1110\ 1111\ 1001_2 = 868daef9_{16}.$$

5.5.2. Порозрядне підсумовування в раунді

Результат перетворення за допомогою *P*-блока прямої перестановки підсумовується за модулем *2* (результат операції *xor*) із *32*-бітовою послідовністю L_{i-1} і виходить *32*-бітова послідовність R_{i-1}^L (див. рис. 5.6):

$$R_{i-1}^L = R_{i-1}^P \oplus L_{i-1}. \qquad (5.11)$$

Приклад 5.12. Необхідно визначити послідовність R_0^L на виході суматора за модулем *2* раунди *DES*, якщо на його вхід надходить послідовність $R_0^P = 868daef9_{16}$ і $L_0 = acf014a7_{16}$.

Розв'язання

Представляючи значення R_0^P і L_0 у двійковому вигляді та використовуючи правило порозрядного підсумовування даних за модулем *2*, отримаємо:

$$\begin{array}{cll} & R_0^P & 10000110100011011010111011111001_2 \\ \oplus & L_0 & 10101100111100000001010010100111_2 \\ \hline & R_0^L & 00101010011111011011101001011110_2 \end{array}.$$

Перетворене значення R_{i-1}^L у шістнадцяткове набуде вигляду

$$R_{i-1}^L = 2a7dba5e_{16}.$$

5.5.3. Пристрій заміни секцій раунду

Пристрій заміни секцій призначений для завершення операцій одного раунду *DES*, тобто він формує значення послідовностей L_i і R_i для наступного раунду. Відповідно до (5.2) і (5.11) послідовності L_i і R_i можна визначати таким чином:

$$\begin{aligned} L_i &= R_{i-1}; \\ R_i &= R_{i-1}^L; \quad i = 1,2,...,16. \end{aligned} \tag{5.12}$$

Приклад 5.13. Необхідно визначити послідовності L_1 і R_1 для другого раунду зашифрування даних, якщо на вхід пристрою заміни надходять послідовності $R_0 = 305a18ca_{16}$ і $R_0^L = 2a7dba5e_{16}$.

Розв'язання

Використовуючи вираз (5.12) і значення R_0 і R_0^L, отримаємо:

$$L_1 = R_0 = 305a18ca_{16}; \quad R_1 = R_0^L = 2a7dba5e_{16}.$$

5.5.4. Алгоритм шифрування даних *DES*

Використовуючи змішувач і пристрій заміни, можна створити прямий і обернений шифр для кожного із *16* раундів. Прямий шифр використовується на стороні зашифрування; обернений шифр — на стороні розшифрування.

Відповідно до виразів (5.2), (5.6) і (5.10), (5.12) алгоритм зашифрування можна представити у вигляді [14, 16]:

$$C = IP(L_0,R_0) \times F(L_0,R_0 \oplus k_1) \times$$
$$\times F(L_1,R_1 \oplus k_2) \times \ldots \times F(L_{15},R_{15} \oplus k_{16}) \times IP^{-1}(R_{16},L_{16}). \quad (5.13)$$

Як випливає з (5.13), у процесі зашифрування даних послідовно використовуються перетворення IP, $F(\bullet)$ і IP^{-1}.

Кожне з перетворень Фейстеля $F(\bullet)$ є композицією двох перетворень.

Перше з них – транспозиція (перестановка) L_i і R_i [14, 16]:

$$T(L_i,R_i) = (L_i,R_i),$$

де T — операція з перестановки місцями лівої L_i і правої R_i половини блока даних (операція транспозиції).

Друге — фактично є перетворенням Вернама *32*-бітових слів, тобто [14, 16]:

$$V(L_i,R_i) = (L_i,R_i \oplus C),$$

де V — перетворення Вернама; C — послідовність, яка залежить від раундового ключа k_i і правої секції L_i.

Отже, для будь-якого раунду

$$F_i(\bullet) = T_i(\bullet) \times V_i(\bullet).$$

Крім того, легко переконатися, що [14, 16]:

$$V^2_i(\bullet) \equiv T^2_i(\bullet)$$

тотожне перетворення. Такі перетворення (елементи групи) заведено називати *інволюціями*. Для них, зокрема, правильно [14, 16]:

$$V_i^{-1}(\bullet) = V_i(\bullet), \quad T_i^{-1}(\bullet) = T_i(\bullet).$$

За правилом обернення добутку елементів неабелевої групи маємо:

$$M = C^{-1} = IP^{-1}(L_0,R_0) \times V(L_0 \oplus F(R_0 \oplus k_1)) \times T \times$$
$$\times V(L_1,F(R_1 \oplus k_2)) \times T \times \ldots \times V(L_{15},F(R_{15}) \oplus k_{16})) \times T \times \quad (5.14)$$
$$\times T \times IP(R_{16},L_{16}).$$

Здійснюючи перетворення в (5.14), остаточно отримаємо алгоритм розшифрування даних [14, 16]:

$$M = IP(R_{16},L_{16}) \times F(R_{15} \oplus F(L_{15} \oplus k_{16})) \times$$
$$\times F(R_{14},F(L_{14} \oplus k_{15})) \times \ldots \times F(R_1,F(L_1) \oplus k_1)) \times IP^{-1}(L_0,R_0). \quad (5.15)$$

Це означає, що розшифрування здійснюється тим самим алгоритмом зашифрування (5.13) і раундовими ключами, але в розкладі раундових ключів потрібно внести деяку зміну: змінити на обернений порядок генерації раундових ключів, тобто вибірка раундових ключів при розшифруванні буде оберненою, тобто:

$$k_{16}, k_{15}, k_{14}, k_{13}, k_{12}, k_{11}, k_{10}, k_9, k_8, k_7, k_6, k_5, k_4, k_3, k_2, k_1,$$

якщо в процесі зашифрування вибірку раундових ключів позначати як

$$k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}.$$

Один із методів досягнення поставленої мети (зашифрування й розшифрування), полягає в тому, щоб зробити останній раунд таким, що відрізняється від інших; він буде містити тільки змішувач і не буде містити пристрою заміни, як це показано на рис. 5.10.

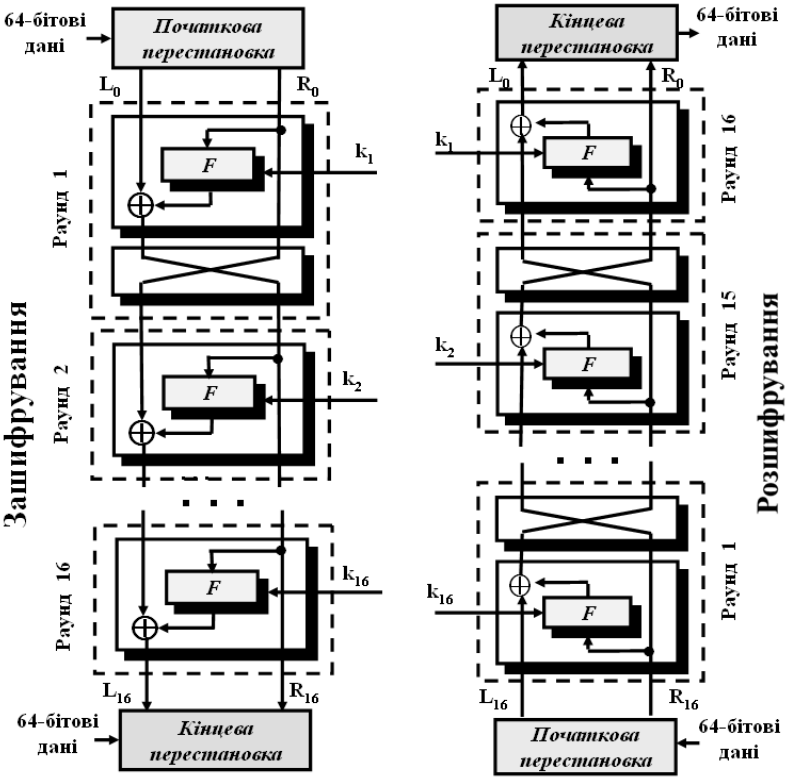


Рис. 5.10. *DES* прямий і обернений шифр для першого способу

У розділі 3 доведено, що змішувач і пристрій заміни самоінверсні. Кінцеві й початкові перестановки також інверсні одна одній. Ліва секція початкових даних на стороні зашифрування шифрується: L_0 як L_{16}, а L_{16} розшифровується на стороні розшифрування L_0. Аналогічна ситуація з R_0 і R_{16}.

У першому методі останній раунд не має пристрою заміни.

За першого способу раунд *16* відрізняється від інших раундів тим, що у ньому не застосовується пристрій заміни. Це необхідно, щоб зробити останній і перший змішувачі в шифрі однаковими.

За другого способу можна зробити усі *16* раундів однаковими, додаючи до *16*-го раунду додатковий пристрій заміни (два пристрої заміни дозволяють нейтралізувати один одного). Цю схему прямого та оберненого шифру показано на рис. 5.11.

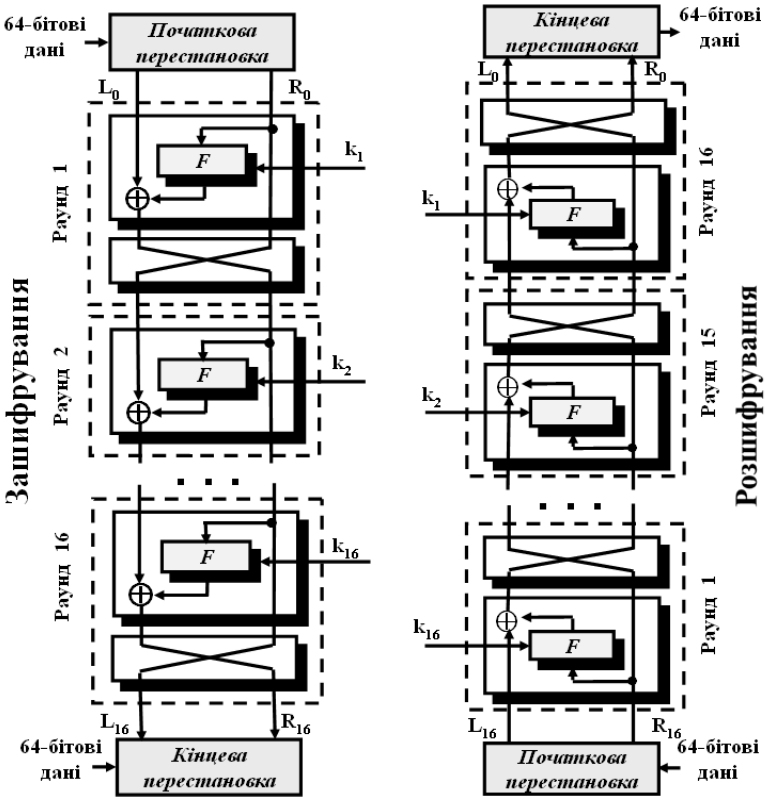


Рис. 5.11. *DES* прямий і обернений шифр для другого способу

5.6. ПРИКЛАДИ ШИФРУВАННЯ ДАНИХ

Перед аналізом *DES* розглянемо декілька прикладів, щоб зрозуміти, як зашифрування й розшифрування змінюють значення бітів у кожному раунді.

Приклад 5.15. Виберемо випадковий блок початкових даних і випадковий ключ і визначимо, яким має бути блок зашифрованих даних при використанні другого способу шифрування (усі цифри дані у шістнадцятковому численні):

- вхідні дані: $123456abcd132536_{16}$;
- ключ: $abba08192637cddc_{16}$.

Табл. 5.16 показує результат кожного раунду й дані, створені до та після раундів.

Таблиця показує результат *16* раундів, які включають змішування й заміну (виключаючи останній раунд).

Таблиця 5.16

Трасування даних у прикладі 5.15

Раунди	Ліві секції L_i	Праві секції 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	$cf26b472_{16}$	$19ba9212_{16}$	$181c5d75c66d_{16}$

Початкові (відкриті) дані – дані, що пройшли через початкову перестановку для отримання різних *64* бітів (*16* шістнадцяткових цифр): $14a7d67818ca18ad_{16}$.

Після розбиття на ліву та праву секції: $L_0 = 14a7d678_{16}$ і $R_0 = 18ca18ad_{16}$.

Після *16*-го раунду й об'єднання $L_{16}||R_{16} = cf26b47219ba9212_{16}$.

Внаслідок виконання кінцевої перестановки отримаємо зашифровані дані:

$$C = c0b7a8d05f3a829c_{16}.$$

Слід зазначити деякі положення. Права секція кожного раунду співпадає з лівою секцією наступного раунду. Причина в тому, що права секція проходить через змішувач без зміни, а пристрій заміни переносить її в ліву секцію. Наприклад, R_1 передається через змішувач другого раунду без зміни, але потім, пройшовши облаштування заміни, він стає L_2.

Приклад 5.16. Розглянемо, як абонент *B* у пункті призначення може розшифрувати зашифровані дані, отримані від абонента *A*, за допомогою співпадаючого ключа. Табл. 5.17 показує результат кожного раунду й даних, створених до та після раундів.

Таблиця 5.17

Трасування даних у прикладі 5.16

Раунди	Ліві секції L_i	Праві секції 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	$18ca18ad_{16}$	$14a7d678_{16}$	$194cd072de8c_{16}$

Перше правило: ключі раунду повинні використовуватися в оберненому порядку. Порівнюємо табл. 5.16 і табл. 5.17. Ключ раунду *1* такий самий, як ключ для раунду *16*. Значення L_0 і R_0 при розшифруванні ті самі, що і значення R_{16} і L_{16} відповідно при зашифруванні. Аналогічні збіги будуть отримані також в інших раундах. Це доводить не лише, що прямий і обернений шифр інверсні один одному, але також те, що кожний раунд при зашифруванні має відповідний раунд при розшифруванні в оберненому шифрі. Результат свідчить, що початкові й кінцеві перестановки також є інверсіями один одного.

Зашифровані дані: $C = c0b7a8d05f3a829c_{16}$.

Після первинної перестановки: $19ba9212cf26b472_{16}$.

Після розбиття на ліву та праву секції: $L_0 = 19ba9212_{16}$ і $R_0 = cf26b472_{16}$.

Після об'єднання: $L_{16}||R_{16} = 18ca18ad14a7d678_{16}$.

Внаслідок виконання кінцевої перестановки отримаємо розшифровані дані:

$$L_{16}||R_{16} = 18ca18ad14a7d678_{16}.$$

5.7. АНАЛІЗ АЛГОРИТМУ

DES ретельно проаналізували, провели випробування, щоб виміряти інтенсивність деяких бажаних властивостей у блоковому шифрі. Елементи *DES* пройшли дослідження на відповідність деяким критеріям.

5.7.1. Лавинний ефект і ефект повноти *DES*

Дві бажані властивості блокового шифру — лавинний ефект і закінченість (ефект повноти).

Лавинний ефект означає, що невеликі зміни в початкових даних (чи в ключі) можуть викликати значні зміни в зашифрованих даних. Було доведено, що *DES* має всі ознаки цієї властивості [19, 35, 37].

Приклад 5.17. Щоб перевірити лавинний ефект у *DES*, спробуємо зашифрувати два блоки початкових даних, які відрізняються тільки одним бітом, за допомогою того самого ключа визначимо різницю в кількості бітів у кожному раунді.

Перший випадок:
- відкриті дані: 0000000000000000_{16};
- ключ: $23234513987aba23_{16}$;
- зашифровані дані: $4789fd476e82a5f1_{16}$.

Другий випадок:
- відкриті дані: 0000000000000001_{16};
- ключ: $23234513987aba23_{16}$;
- зашифровані дані: $0a4ed5c15a63fea3_{16}$.

Хоча два блоки початкового тексту відрізняються тільки крайнім правим бітом, блоки зашифрованого тексту відрізняються на *29* бітів. Це означає, що зміна приблизно в *1,5*% початкових даних створюють зміну близько *45*% зашифрованих даних.

Табл. 5.18 показує зміну в кожному раунді в порівнянні L_i і R_i у двох вказаних випадках. Можна побачити, що істотні зміни виникають уже в третьому раунді.

Таблиця 5.18

Кількість різних бітів у прикладі *5.17*

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 початкових даних. Розсіювання й перемішування, зроблене *E*-блоками перестановки й розширення, *S*-блоками заміни й *P*-блоками прямої перестановки в *DES*, вказує на дуже сильний ефект повноти.

5.7.2. Критерії розробок

Численні випробування *DES* показали, що він задовольняє деякі із заявлених критеріїв. Нижче коротко викладено декілька проблем розробок *DES*.

Проблеми розробки S-блоків DES

У розділі 3 описано загальні критерії побудови *S*-блоків. Розглянемо критерії, вибрані для *DES*. Структура блоків забезпечує перемішування й розсіювання від кожного раунду до подальшого.

Згідно з цим положенням і деяким аналізом можна згадати декілька властивостей *S*-блоків.

1. Входи кожного рядка є перестановки значень між *0* і *15*.

2. *S*-блоки нелінійні. Іншими словами, вихід — неафінне перетворення.

3. Якщо змінювати один єдиний біт на вході, на виході буде змінено два або більше бітів.

4. Якщо два входи *S*-блока відрізняються тільки двома середніми бітами (бітами *3* і *4*), вихідна інформація повинна відрізнятися принаймні двома бітами. Іншими словами, *S(x)* і $S(x \oplus 001100)$ повинні відрізнятися принаймні двома бітами, де *x* — вхід і *S(x)* — вихід.

5. Якщо два входи в *S*-блок відрізняються першими двома бітами (біти *1* і *2*) і останніми двома бітами (*5* і *6*), два виходи повинні бути різними. Іншими словами, ми повинні мати таке відношення: $S(x) \neq S(x \oplus 0011bc00)$, в якому *b* і *c* — довільні біти.

6. Є тільки *32* шестибітові пари "вхід-вихід" (x_i і x_j), в яких $x_i \oplus x_j \neq 000000_2$. Ці *32* вхідні пари створюють *32* пари слова виходу по *4* біти. Якщо ми створюємо якісь відмінності між *32* виходами пар, $d = y_i \oplus y_j$, то з цих *d* мають бути однаковими не більше ніж *8* пар.

7. Такий самий критерій, як в пункті 6, застосовується до трьох *S*-блоків.

8. У будь-якому *S*-блоці, якщо єдиний вхідний біт зберігається як константа (*0* або *1*), то інші біти змінюються випадково так, щоб різниці між кількістю нулів і одиниць були мінімізовані.

Проблеми розробки E-блоків і P-блоків DES

Між двома рядами *S*-блоків заміни (у двох подальших раундах), є один *E*-блок перестановки й розширення (*32* на *48*) і один *P*-блок прямої перестановки (*32* на *32*). Ці два блоки разом забезпечують розсіювання бітів. Про загальний принцип побудови блока заміни сказано в розділі 3. Тут розглянемо тільки прикладні блоки, використовувані в *DES*. У структурі блоків *DES* було реалізовано такі критерії:

1. Кожний вхід *S*-блока підключається до виходу іншого *S*-блока (у попередньому раунді).

2. Жодний вхід до цього *S*-блока не з'єднується з виходом від того самого блока (у попередньому раунді).

3. Чотири біти від кожного *S*-блока надходять у шість різних *S*-блоків (у наступному раунді).

4. Жодний із двох бітів виходу від *S*-блока не надходить у той самий *S*-блок (у наступному раунді).

5. Вихід S_{j-2}-блока заміни переходить в один із перших двох бітів S_j-блока заміни (у наступному раунді).

6. Біт виходу від S_{j-1}-блока заміни переходить в один із останніх двох бітів S_j-блока заміни (у наступному раунді).

7. Вихід S_{j+1}-блока заміни переходить в один із двох середніх бітів S_j-блока заміни (у наступному раунді).

8. Для кожного *S*-блока заміни два біти виходу надходять у перші або останні два біти *S*-блока заміни в наступному раунді. Інші два біти виходу надходять у середні біти *S*-блока заміни в наступному раунді.

9. Якщо вихід від S_j-блока заміни переходить в один із середніх бітів S_k-блока заміни (у наступному раунді), то біт виходу від S_k-блока заміни не може йти в середній біт S_j-блока заміни. Якщо допустити, що $j = k$, то жодний середній біт *S*-блока заміни не може надходити в один із середніх бітів того самого *S*-блока заміни в наступному раунді.

Кількість раундів DES

DES використовує шістнадцять раундів шифру Фейстеля. Доведено [19, 35, 37], що після того, як кожний блок початкових даних зашифрований вісім раундів, кожний біт зашифрованих даних — функція кожного біта початкового блока даних і кожного ключового біта. Зашифровані дані — повністю випадкова функція початкових даних і ключа. Звідси начебто витікає, що восьми раундів повинно бути достатньо для вдалого шифрування. Проте експерименти показують, що деякі версії *DES* із менш ніж шістнадцятьма раундами більш уразливі до атак знання початкових даних, ніж до атаки "грубої сили", яка вимагає використання шістнадцяти раундів *DES*.

5.7.3. Уразливі місця

Впродовж минулих багатьох років криптографічними аналітиками було знайдено деякі 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:

1. У S_4-блоці заміни три біти виходу можуть бути отримані тим самим способом, що і перший біт виходу: доповненням деяких із вхідних бітів.

2. Два спеціально вибрані входи до масиву *S*-блока заміни можуть створити той самий вихід.

3. Можна отримати той самий вихід в одному єдиному раунді, змінюючи біти тільки в трьох сусідніх *S*-блоках заміни.

Блоки початкової IP і кінцевої IP^{-1} *перестановок.* У структурі цих блоків було знайдено одну загадку. Незрозуміло, чому проектувальники *DES* використали початкову й кінцеву перестановки. Ці перестановки не вносять ніяких нових властивостей з погляду безпеки.

E-блок перестановки й розширення. У блоці перестановки й розширення перші та четверті біти послідовностей на *4* біти повторюються.

5.7.4. Уразливість ключа шифру

Критики стверджують, що найбільша уразливість *DES* — це розмір ключа (*56* бітів); щоб провести атаку "грубої сили" блока зашифрованих даних, зловмисник повинен перевірити 2^{56} ключів:

1. Використовуючи доступну сьогодні технологію, можна перевірити один мільйон ключів за секунду. Це означає, що знадобиться більше, ніж дві тисячі років, щоб виконати атаку "грубої сили" на *DES*, використовуючи комп'ютер тільки з одним процесором.

2. 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 1998 р., завдяки йому ключ було знайдено за *112* год.

3. Комп'ютерні мережі можуть моделювати паралельну обробку. 1977 р. команда дослідників використала *3500* комп'ютерів, підключених до *Internet*, щоб знайти ключ *DES* за *120* днів. Безліч ключів поділили серед усіма цими комп'ютерами, і кожний комп'ютер був відповідальний за перевірку частини домену *DES*. Якщо *3500* зв'язаних із мережею комп'ютерів змогли знайти ключ за *120* днів, то суспільство із *42000* членів може знайти ключ за *10* днів.

Наведене вище показує, що *DES* із розміром ключа шифру *56* бітів не забезпечує достатньої безпеки.

Уразливі ключі в шифрі DES

Чотири ключі з 2^{56} можливих ключів називаються уразливими ключами. *Уразливі ключі* — це такі, які після операції видалення перевірочних бітів (використовуючи табл. 5.1) складаються з усіх нулів або усіх одиниць, або половини нулів і половини одиниць. Такі ключі показано в табл. 5.19.

Таблиця 5.19

Уразливі ключі в шифрі *DES*

Ключі до видалення перевірочних бітів і перестановки (*64* біти)	Діючі ключі (*56* бітів)
$0101\ 0101\ 0101\ 0101_{16}$	$0000\ 000\ 0000\ 0000_{16}$
$1f1f\ 1f1f\ 1f1f\ 1f1f_{16}$	$0000\ 000\ ffff\ ffff_{16}$
$e0e0\ e0e0\ e0e0\ e0e0_{16}$	$ffff\ ffff\ 0000\ 0000_{16}$
$fefe\ fefe\ fefe\ fefe_{16}$	$ffff\ ffff\ ffff\ ffff_{16}$

Ключі раунду, створені від будь-якого з цих уразливих ключів, — такі самі й мають такий самий тип, що й ключ шифру. Наприклад, ці шістнадцять ключів раунду 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. Зміщення або перестановка блока не змінюють блок, якщо він складається з усіх нулів, або усіх одиниць, або наполовину з нулів і одиниць.

У чому небезпека використання уразливих ключів? Якщо зашифрувати блок даних уразливим ключем і потім ще раз зашифрувати результат тим самим уразливим ключем, виходить первинний блок даних. Процес створює той самий первинний блок даних, якщо розшифровувати блок двічі. Іншими словами, кожний уразливий ключ є інверсією самого себе:

$$E_k\,(E_k\,(M)) = M,$$

як це показано на рис. 5.12.

Уразливих ключів потрібно уникати, тому що зловмисник може легко розпізнати їх на перехопленому зашифрованому повідомленні. Якщо після двох етапів розшифрування результат такий самий, супротивник визначає, що він знайшов ключ.

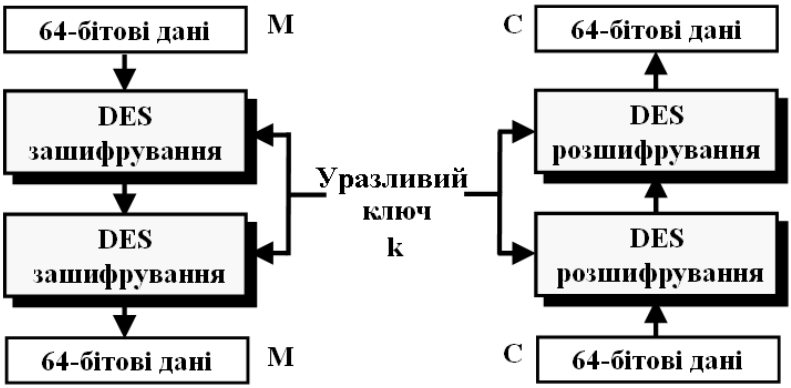


Рис. 5.12. Подвійне зашифрування й розшифрування даних
із уразливим ключем

Приклад 5.18. Спробуємо застосувати перший вразливий ключ із табл. 5.19, щоб двічі зашифрувати блок даних. Після того, як проведено два зашифрування з тим самим ключем, як результат отримаємо блок даних. Зауважимо, що при цьому жодного разу не використовувався алгоритм розшифрування, а тільки проведено двічі зашифрування.

Ключ: 0101010101010101_{16}.
Початковий блок даних: 1234567887654321_{16}.
Зашифрований блок даних: $814fe938589154f7_{16}$.
Ключ: 0101010101010101_{16}.
Початковий блок даних: $814fe938589154f7_{16}$.
Зашифрований блок даних: 1234567887654321_{16}.

Напівуразливі ключі в шифрі DES

Є шість ключових пар, які названі *напівуразливими ключами*. Ці шість пар показано в табл. 5.20 (формат на *64* біти перед видаленням перевірочних бітів).

Напівуразливі ключі створюють тільки два різні ключі раунду й потім повторюють їх вісім разів. Крім того, ключі раунду, створені від кожної пари, ті самі в різному порядку.

Таблиця 5.20

Напівуразливі ключі в шифрі *DES*

Перший ключ у парі	Другий ключ у парі
$01fe\ 01fe\ 01fe\ 01fe_{16}$	$fe01\ fe01\ fe01\ fe01_{16}$
$1fe0\ 1fe0\ 0ef1\ 0ef1_{16}$	$e01f\ e01f\ f10e\ f10e_{16}$
$01e0\ 01e1\ 01f1\ 01f1_{16}$	$e001\ e001\ f101\ f101_{16}$
$1ffe\ 1ffe\ 0efe\ oefe_{16}$	$fe1f\ fe1f\ feoe\ fe0e_{16}$
$011f\ 011f\ 010e\ 010e_{16}$	$1f01\ 1f01\ 0e01\ 0e01_{16}$
$e0fe\ e0fe\ f1fe\ f1fe_{16}$	$fee0\ fee0\ fef1\ fef1_{16}$

Щоб проілюструвати ідею, створимо ключі раунду від першої пари, як показано у табл. 5.21.

Таблиця 5.21

Раундові ключі для напівуразливих ключів у шифрі *DES*

Раунди	Раундові ключі для першого ключа в парі	Раундові ключі для другого ключа в парі
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	$6eac1abce642_{16}$	$9153e54319bd_{16}$

Як показує аналіз табл. 5.21, є вісім однакових ключів раунду в кожному напівуразливому ключі. Крім того, ключі раунду *1* у першій множині ті самі, що й ключі раунду *16* у другому; ключі раунду *2* в першому ті самі, що й ключі раунду *15* у другому, і так далі. Це означає, що ключі інверсні один одному:

$$E_{k_2}(E_{k_1}(M)) = M,$$

як показано на рис. 5.13.

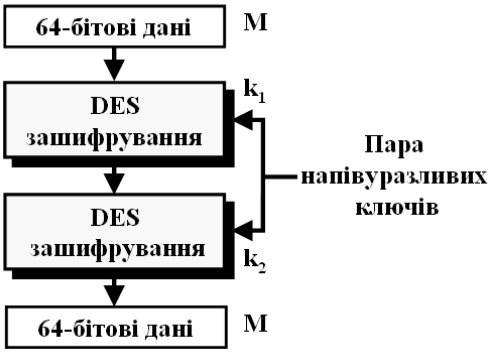


Рис. 5.13. Подвійне шифрування даних із напівуразливим ключем

Можливо уразливі ключі в шифрі DES

Також є *48* ключів, які називаються можливо уразливими ключами. *Можливо уразливий ключ* створює тільки чотири різні ключі раунду; іншими словами, шістнадцять ключів раундів поділені на чотири групи, і кожна група складається з чотирьох однакових ключів раунду.

Приклад 5.19. Яка ймовірність випадкового вибору уразливого, напівуразливого або можливо уразливого ключа?

Розв'язання

Множина ключів *DES* дорівнює 2^{56}. Загальна кількість вищезгаданих ключів — *64: 4 + 12 + 48*. Ймовірність вибору одного з цих ключів дорівнює $8{,}8 \cdot 10^{-16}$, тобто виключно мала.

Ключове доповнення в шифрі DES

Серед безлічі ключів (2^{56}) деякі ключі можуть бути отримані інверсією (зміна з *0* в *1* або *1* в *0*) кожного біта в ключі. Ключове доповнення спрощує процес криптографічного аналізу. Зловмисник може використати тільки половину можливих ключів (2^{56}), щоб виконати атаку "грубої сили", тому що:

$$M \oplus M_Д = ffffffffffffffff_{16}, \quad K \oplus K_Д = ffffffffffffffff_{16}$$

та

$$C \oplus C_Д = ffffffffffffffff_{16}.$$

Іншими словами, якщо зашифрувати доповнення початкового блока даних доповненням ключа, вийде доповнення зашифрованого блока даних. Зловмисник не повинен перевіряти усі 2^{56} можливих ключів, він може перевірити тільки половину з них і потім доповнити результат.

Приклад 5.20. Перевіримо ці відомості про ключове доповнення. Скористаємося довільним ключем і початковим блоком даних для того, щоб знайти відповідний зашифрований блок даних. Якщо є ключове доповнення й початковий блок даних, то вийде доповнення попереднього зашифрованого блока даних (табл. 5.22).

Таблиця 5.22

Результати прикладу 5.20

	Оригінал ключа	Ключове доповнення
Ключ	1234123412341234_{16}	$edcbedcbedcbedcb_{16}$
Початковий блок даних	$12345678abcdef12_{16}$	$edcba987543210ed_{16}$
Зашифрований блок даних	$e112be1defc7a367_{16}$	$1eed41e210385c98_{16}$

Кластерний ключ у шифрі DES

Кластерний ключ розглядає ситуації, в яких два або більше різних ключі створюють той самий зашифрований блок даних із того самого початкового блока даних. Очевидно, кожна пара напівуразливих ключів — ключовий кластер. Проте більше кластерів не було знайдено. Майбутні дослідження, можливо, можуть відкрити деякі інші.

5.8. БАГАТОРАЗОВЕ ЗАСТОСУВАННЯ АЛГОРИТМУ

Як вже було зазначено, основну критику *DES* спрямовано на довжину ключа. Можливі технології й можливості паралельних процесорів роблять реальною атаку “грубої сили”.

Одне з рішень для поліпшення безпеки — це застосування шифру *Advanced Encryption Standard* (*AES*).

Друге рішення – багатократне (каскадне) застосування множини ключів. Це рішення, яке використовувалося деякий час, не вимагало істотних інвестицій у нове програмне забезпечення й апаратні засоби. Розглянемо такий підхід.

Як відомо із розділу 3, підстановка, яка розміщує усі можливі входи в усі можливі виходи, є групою з відображеннями елементів великої кількості й набором операцій. У цьому випадку використання двох послідовних відображень даремне, тому що ми можемо завжди знайти третє відображення, яке еквівалентне композиції цих двох (властивість замкнутості). Це означає, що якщо *DES* — група, то одноразовий *DES* із ключем k_3 робить те саме (рис. 5.14).

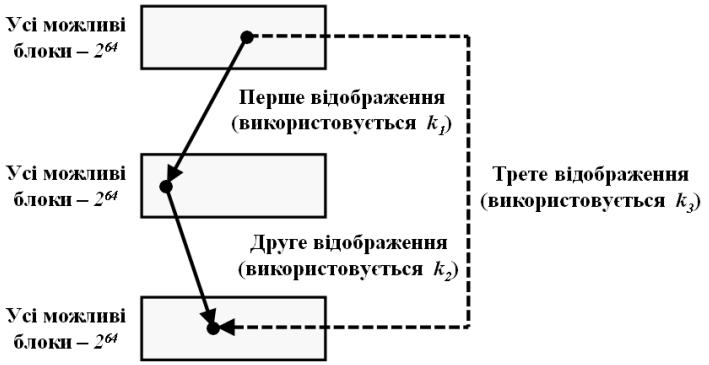


Рис. 5.14. Композиція відображень

Отже, *DES* не група. Вона базується на таких двох параметрах:

1. Номер можливих входів або виходів у *DES* – $N = 2^{64}$. Це означає, що $N! = (2^{64}!) = 10\ 347\ 380\ 000\ 000\ 000\ 000\ 000$ відображень. Один зі способів зробити *DES* групою — це підтримати усі ці відображення з розміром ключа $log_2\ (2^{64}!) \approx 270$ бітів. Оскільки довжина ключа в *DES* — *56* бітів, то це тільки маленька частина необхідного величезного ключа.

2. Інший спосіб зробити *DES* групою — необхідно, щоб множина відображень була підмножиною великої кількості відносно залежності від першого параметра. Було доведено [37], що групи, створені з групи за допомогою першого параметра, мають ключовий розмір *56* бітів.

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 використати подвійні або потрійні *DES*, щоб збільшити розмір ключа.

5.8.1. Подвійний *DES*

Перший підхід полягає в тому, щоб використати подвійний *DES* (*Double DES*). За цього підходу застосовуємо два типи прямих шифрів *DES* для зашифрування та два типи обернених шифрів для розшифрування. Кожний тип використовує різний ключ, що означає — розмір ключа тепер подвоївся (*112* бітів). Проте подвійний *DES* уразливий до атаки знання відкритих даних.

На перший погляд подвійні *DES* збільшують кількість випробувань при пошуку ключа від 2^{56} (в одноразовому *DES*) до 2^{112} (у подвійному *DES*). Проте за використання атаки знання відкритих даних, що називається *атакою* "*зустрічі посередині*", можна довести, що двократний *DES* покращує цю стійкість (тільки до 2^{57} за випробуваннями), але не надзвичайно (до 2^{112}).

Рис. 5.15. показує діаграму для подвійного *DES*. Відправник використовує два ключі, k_1 і k_2, щоб зашифровувати початковий блок даних *M* у зашифрований блок даних *C*; одержувач використовує зашифрований блок даних *C* і два ключі, k_1 і k_2, для відновлення початкового блока даних *M*.

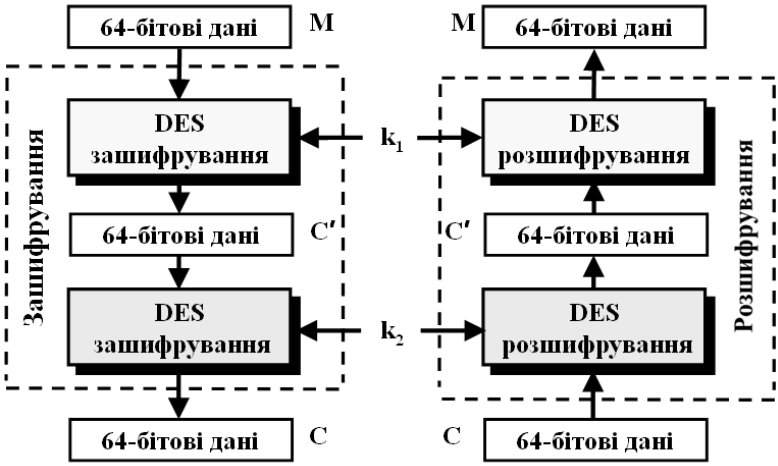


Рис.5.15 Атака "зустрічі посередині" в подвійному *DES*

У середній точці *C′* — блок даних, створений першим зашифруванням або першим розшифруванням. Для забезпечення правильної

роботи він повинен бути однаковим для зашифрування й розшифрування. Іншими словами, маємо два співвідношення:

$$C' = E_{K_1}(M) \quad \text{і} \quad C' = E_{K_2}(C).$$

Припустимо, що зловмисник перехопив попередню пару M і C (атака знання відкритих 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 інших співвідношеннях, згаданих вище, зловмисник розшифровує C', використовуючи всі можливі значення (2^{56}) ключа k_2. Він записує всі значення, отримані для C'. Далі зловмисник створює дві таблиці, відсортовані згідно зі значеннями C'. Він порівнює значення для C', доки не знаходить ті пари k_1 і k_2, для яких значення C' є такими самими в обох таблицях (як показано на рис. 5.16).

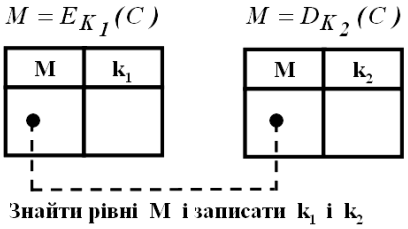


Рис. 5.16. Таблиці для атаки "зустріч посередині"

Зауважимо, що повинна бути принаймні одна пара, тому що вона робить вичерпним пошук комбінації двох ключів:

1. Якщо є тільки одна відповідність. Зловмисник знайшов два ключі (k_1 і k_2). Якщо є більше ніж один кандидат, зловмисник переходить до наступного кроку.

2. Зловмисник бере іншу перехоплену пару зашифрованого блока даних і початкового блока даних і використовує кожного кандидата для отримання пари ключів, щоб встановити, чи може він отримати зашифрований блок даних із початкового блока даних. Якщо він знаходить більше ніж одного кандидата у вигляді пари ключів, він повторює крок 2 доти, доки нарешті не знаходить унікальну пару.

Було доведено [19, 35, 37], що після застосування другого кроку до декількох перехоплених пар "зашифровані дані—відкриті дані" ключі було знайдено. Це означає, що замість того щоб використати пошук ключів за допомогою 2^{112} випробувань, зловмисник використовує 2^{56} випробувань пошуку ключа й перевіряє двічі (дещо більше

випробувань потрібно, якщо знайдений на першому кроці єдиний кандидат). Іншими словами, рухаючись від одноразового *DES* до подвійного *DES*, збільшується об'єм випробувань від 2^{56} до 2^{57} (а не до 2^{112}, як це здається за поверхневого підходу).

5.8.2. Потрійний *DES*

Для того щоб поліпшити безпеку *DES*, було запропоновано потрійний *DES* (*Triple DES*). Він використовує три каскади *DES* для зашифрування й розшифрування. Сьогодні використовуються дві версії потрійних *DES*: потрійний *DES* із двома ключами та потрійний *DES* із трьома ключами.

Потрійний DES із двома ключами

У потрійному *DES* із двома ключами є тільки два ключі: k_1 і k_2. Перший і третій каскади використовують k_1; другий каскад використовує k_2. Щоб зробити потрійний *DES* сумісним із *DES*, середній каскад застосовує розшифрування (обернений шифр) на стороні зашифрування й зашифрування (прямий шифр) на стороні розшифрування. У такий спосіб повідомлення, зашифроване *DES* є ключем k, може бути розшифроване потрійним *DES*, якщо тільки $k_1 = k_2 = k$. Хоча потрійний *DES* із двома ключами також уразливий при атаці "знання відкритих даних", він набагато стійкіший, ніж подвійний *DES* (прийнятий свого часу для банків).

Рис. 5.17 показує потрійний *DES* із двома ключами.

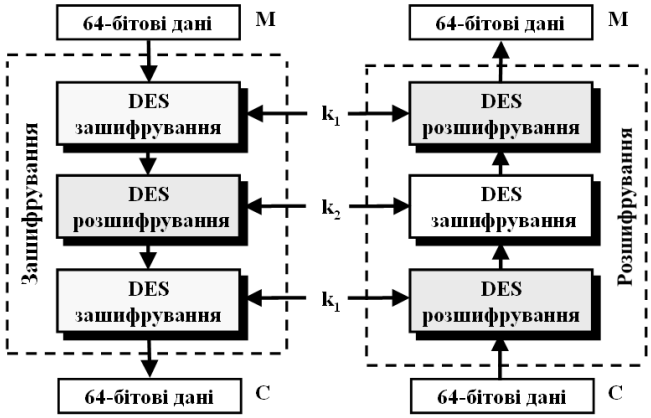


Рис.5.17. Потрійний *DES* із двома ключами

Потрійний DES із трьома ключами

Можливість атак "знання відкритих даних" за потрійного *DES* із двома ключами змусила деякі додатки використати потрійний *DES* із трьома ключами. Алгоритм може застосовувати три каскади прямого шифру *DES* на стороні зашифрування та три каскади зворотних шифрів на стороні розшифрування (рис. 5.18).

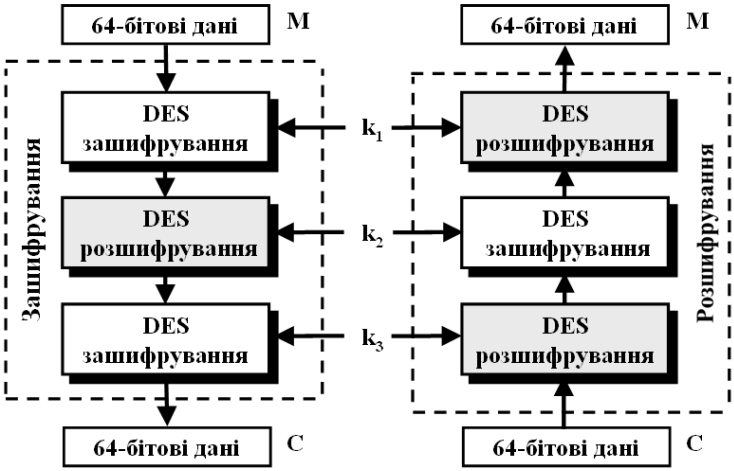


Рис. 5.18. Потрійний *DES* із трьома ключами

Для сумісності з одноразовим *DES* сторона зашифрування використовує *EDE*, а сторона розшифрування — *DED*. *E (encryption*) — каскад зашифрування, *D (decryption*) — каскад розшифрування.

Сумісність з одноразовим *DES* забезпечується за $k_1 = k$ і установкою k_2 і k_3 до того самого довільного ключа, вибраного одержувачем. Потрійний *DES* із трьома ключами використовується багатьма застосуваннями, такими як *PGP* (*Pretty Good Privacy – достатньо* "*гарна*" *конфіденційність*), оскільки забезпечує високу захищеність властивості інформації — конфіденційність.

5.9. БЕЗПЕКА ШИФРУ

DES як перший блоковий шифр, що має важливе значення, пройшов через багато випробувань на безпеку. Серед зроблених атак лише три становлять інтерес: атака "грубої сили", диференціальний і лінійний криптографічний аналіз.

5.9.1. Атака “грубої сили”

Уже описувалась уразливість шифру з коротким ключем. Уразливість ключа спільно з іншими розглянутими недоліками робить очевидним, що *DES* може бути зламаний із кількістю випробувань 2^{55}. Проте сьогодні більшість додатків використовують або *Double DES* (розмір ключа 2^{112}), або *Triple DES* (розмір ключа 2^{168}). Ці дві багаторазові версії *DES* дозволяють йому показувати істотну стійкість до атаки “грубої сили”.

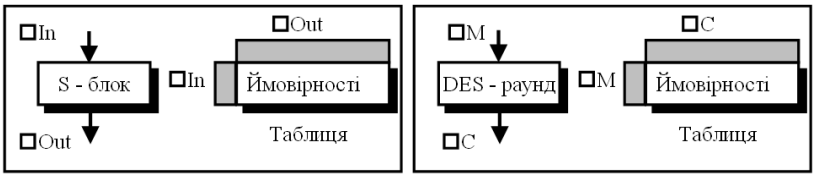
5.9.2. Диференціальний криптографічний аналіз

У розділі 3 вже описувалась методика диференціального криптографічного аналізу для сучасних блокових шифрів. *DES* не є стійким до такого виду атаки. Проте багато що вказує, що розробники *DES* уже знали про цей тип атаки та проектували *S*-блоки й спеціально вибрали кількість раундів, щоб зробити *DES* стійким до цього типу атаки. Сьогодні показано, що *DES* може бути зламаний, використовуючи диференціальний криптографічний аналіз, якщо ми маємо 2^{47} вибірок початкових (відкритих) даних або 2^{55} відомих початкових даних. Хоча це виглядає ефективніше, ніж в атаці “грубої сили”, припустити, що хтось знає 2^{47} вибірок початкових даних або 2^{55} вибірок відомих початкових даних, майже неможливо. Тому ми можемо сказати, що *DES* є стійким до диференціального криптографічного аналізу. Також показано, що збільшення кількості раундів до *20* збільшує кількість необхідних вибірок початкових даних для атаки більш ніж 2^{64}. Таке збільшення неможливе, тому що кількість блоків початкових даних тексту в *DES* тільки 2^{64}.

Диференціальний криптографічний аналіз для *DES* винайшли Біхам і Шамір. У цьому криптографічному аналізі зловмисник концентрується на атаках із вибіркою початкових даних. Аналіз використовує різницю в проходженні різних вхідних даних через пристрій або програму шифрування. Термін “різниця” тут застосовується, щоб розглянути за допомогою операції *xor* неспівпадання двох різних вхідних повідомлень (початкових даних). Іншими словами, зловмисник аналізує, як $M \oplus M'$ розрізняються при обробці в кожному раунді.

Ідея відносно диференціального криптографічного аналізу базується на ймовірнісних стосунках між різницями даних входу й різницями даних виходу. Два відношення становлять конкретний

інтерес в аналізі: диференціальний профайл і характеристика раунду, як це показано на рис. 5.19.

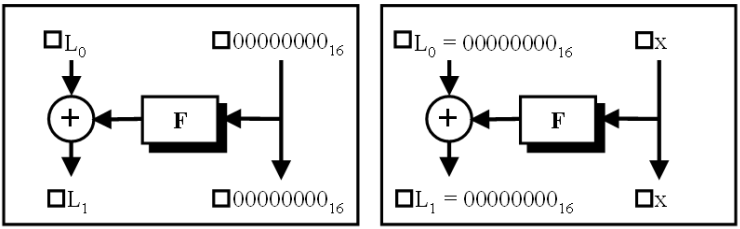


а) диференційний профайл б) характеристика раунду

Рис. 5.19. Диференціальний профайл і характеристика раунду в *DES*

Диференціальний профайл (він же профайл *xor*) показує ймовірнісне відношення між різницями даних входу й різницями даних виходу *S*-блока. Подібні профайли можуть бути створені для кожного з восьми *S*-блоків у *DES*.

Характеристика раунду подібна до диференціального профайлу, але обчислюється для цілого раунду. Вона показує ймовірність, з якою одна вхідна різниця створила б різницю певного виходу. Зауважимо, що характеристика та сама для кожного раунду, тому що будь-яке відношення, яке включає різниці, не залежить від ключів раунду. Рис. 5.20 показує чотири різні характеристики раунду.



а) p = 1 б) p = 1/234

б) p = 1/4 г) p = 14/64

Рис. 5.20. Деякі характеристики раунду
для диференціального криптографічного аналізу

Хоча існує багато характеристик для раунду, рис. 5.20 показує тільки чотири з них. У кожній характеристиці розділені різниці даних входу й різниці даних виходу в ліві та праві секції. Кожна ліва або права різниці складаються з *32* бітів або восьми шістнадцяткових цифр.

Усі ці характеристики можуть бути знайдені програмами, які можуть знайти відношення "входу-виходу" в раунді *DES*. Рис. 5.20, а показує, що різниця входу (x, 00000000_{16}) дає різницю на виході (x, 00000000_{16}) з імовірністю *1*.

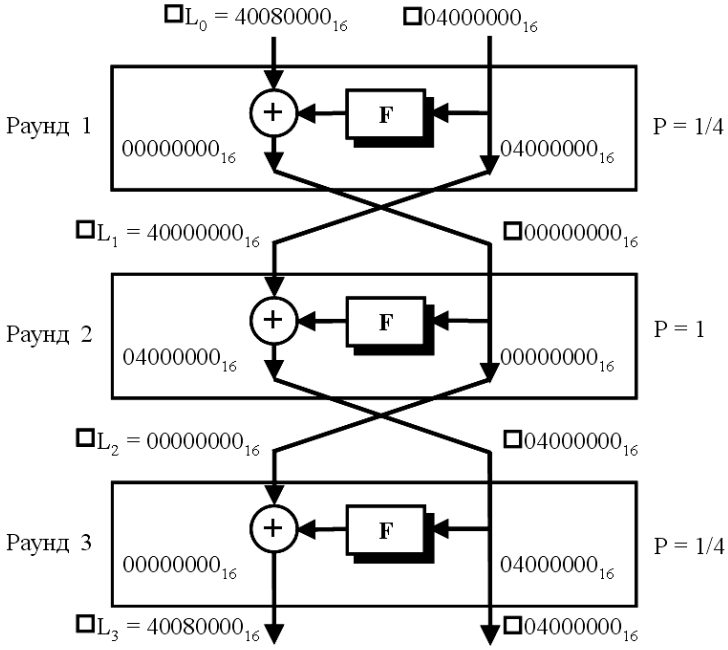


Рис. 5.21. Трираундова характеристика для диференціального криптографічного аналізу

Рис. 5.20, б показує ту саму характеристику, що й рис. 5.20, а, за винятком того, що лівий і правий вхід і вихід помінялися місцями; ймовірність зміниться надзвичайно. Рис. 5.20, в показує, що різниця входу (40080000_{16}, 04000000_{16}) дає різницю виходу (00000000_{16}, 04000000_{16}) з імовірністю *1/4*. Нарешті, рис. 5.20, г показує, що різниця входу (00000000_{16}, 60000000_{16}) дає різницю виходу (00808200_{16}, 60000000_{16}) із імовірністю *14/64*.

Після створення та зберігання однораундових характеристик аналітик може комбінувати різну кількість раундів, щоб створити множинну характеристику раунду. Рис. 5.21 показує випадок трираундового *DES*.

На рис. 5.21 використовуються три змішувачі й тільки два пристрої заміни, тому що останній раунд не потребує ніякого пристрою заміни. Характеристики, показані в змішувачах перших і третіх раундів, ті самі, що й на рис. 5.20, б. Характеристика змішувача в другому раунді така сама, що й на рис. 5.20, а. Цікаво зазначити, що точки, в цьому конкретному випадку, різниці "входу-виходу" — ті самі ($\Delta L_3 = \Delta L_0$ і $\Delta R_3 = \Delta R_0$).

Для шифру із шістнадцятьма раундами можна сформувати багато різних характеристик. (Приклад на рис. 5.22.)

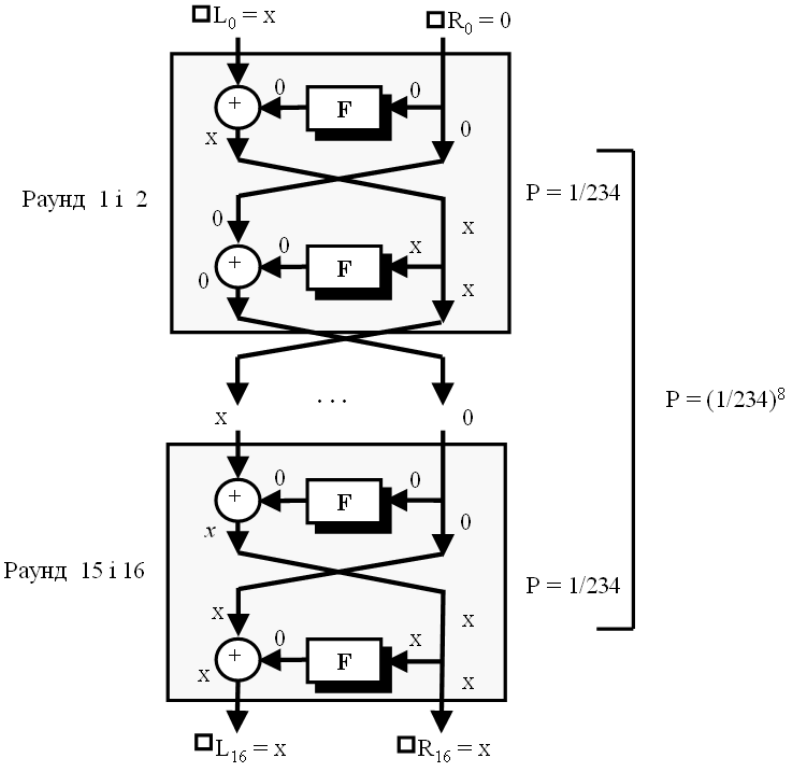


Рис. 5.22. Шістнадцятираундова характеристика для диференціального криптографічного аналізу

На цьому рисунку шифр *DES* складається з восьми секцій із двома раундами. Кожна секція використовує характеристики на рис. 5.20, а і 5.20, б. Зрозуміло, якщо останні раунди не мають пристрою заміни, вхід (*x, 0*) створює вихід (*0, x*) із імовірністю $(1/234)^8$.

Для прикладу припустимо, що зловмисник використовує характеристику за рис. 5.21, щоб створити атаку на *DES* із шістнадцятьма раундами. Зловмисник деяким чином провокує відправника повідомлень, щоб зашифрувати багато початкових даних у формі (*x, 0*), в якій ліва половина – *x* (різні значення) і права половина — *0*. Зловмисник потім зберігає усі зашифровані дані, отримані від відправника, у формі (*0, x*). Зауважимо, що *0* тут означає 00000000_{16}.

Остаточна мета зловмисника в диференціальному криптографічному аналізі полягає в тому, щоб знайти ключ шифру. Для цього треба знайти ключі кожного раунду від основи до вершини (k_{16} … k_1).

Якщо зловмисник має достатньо багато пар початкових даних/зашифрованих даних (кожна з різними значеннями *x*), він може використати відношення в останньому раунді, $0 = F(k_{16},x)$ і знайти деякі з бітів у k_{16}. Це можна зробити, вибираючи найімовірніші значення.

Ключі для інших раундів можна знайти, використовуючи інші характеристики або застосовуючи атаки “грубої сили”.

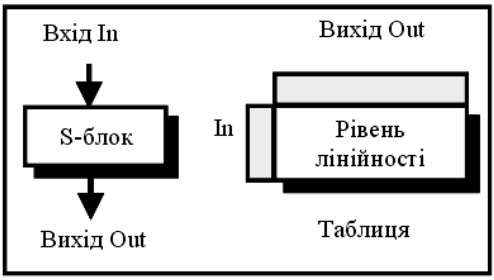
Відомо, що потрібно 2^{47} вибірок пар початкових даних/зашифрованих даних, щоб атакувати *DES* із *16* раундами. Знайти таку величезну кількість вибраних пар надзвичайно важко в ситуаціях реального життя. Це означає, що *DES* невразливий для цього типу атаки.

5.9.3. Лінійний криптографічний аналіз

Методику лінійного криптографічного аналізу для сучасних блокових шифрів було описано в розділі 3. Лінійний криптографічний аналіз — новіша методика, ніж диференціальний криптографічний аналіз. *DES* більш уразливий до застосування лінійного криптографічного аналізу, ніж до диференціального криптографічного 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 із використанням 2^{43} пар відомих початкових даних. Проте з практичного погляду перехоплення такої кількості пар малоймовірне.

Лінійний криптографічний аналіз для *DES* був розроблений Мацуї [6, 26]. Це — атака знання початкових даних. Аналіз використовує поширення конкретного набору бітів через пристрій шифрування.

Лінійний криптографічний аналіз ґрунтується на відношенні лінійності. У цьому типі криптографічного аналізу становлять інтерес два набори відношень: лінійні профайли й характеристики раунду, як показано на рис. 5.23.



а) диференційний профайл

б) характеристика раунду

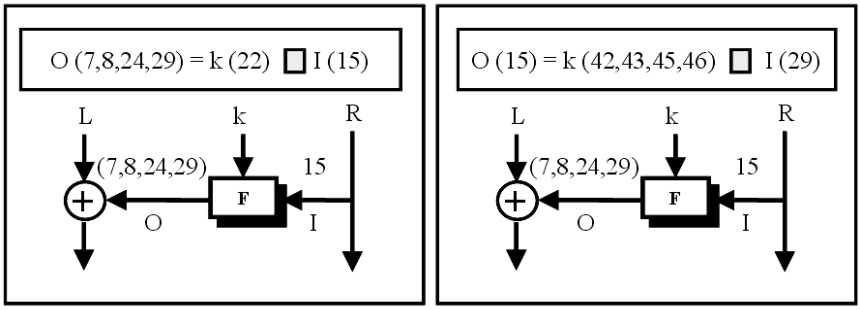
Рис. 5.23. Лінійний профайл і характеристика раунду в *DES*

Лінійний профайл показує рівень лінійності між входом і виходом *S*-блока. У *S*-блоці кожний біт виходу — функція усіх вхідних бітів. Бажану властивість у *S*-блоці досягнуто, якщо кожний біт виходу — нелінійна функція усіх вхідних бітів.

На жаль, ця ідеальна ситуація не існує в *DES*; деякі біти виходу — лінійна функція деяких комбінацій вхідних бітів. Іншими словами, деякі комбінації бітів "входу-виходу" можуть бути відображені між собою, використовуючи лінійну функцію. Лінійний профайл показує рівень лінійності (чи нелінійності) між входом і виходом.

Криптографічний аналіз може створити вісім різних таблиць, по одній для кожного *S*-блока, в яких перший стовпець показує можливі комбінації входів по шість бітів, від 00_{16} до $3F_{16}$. Перший рядок показує можливі комбінації виходів по чотири біти, від 00_{16} до F_{16}. Входи показують рівень лінійності (чи нелінійності) цього проекту. Не будемо заглиблюватися в деталі того, як вимірюється рівень лінійності, але входи з високого рівня лінійності цікаві для криптографічного аналізу.

Характеристика раунду в лінійному криптографічному аналізі показує комбінації вхідних бітів, бітів ключів раунду й бітів виходу для того, щоб визначити лінійне відношення. Рис. 5.24 зображує дві різні характеристики раунду.



а) P = 52/64 б) P = 42/64

Рис. 5.24. Деякі характеристики раунду
для лінійного криптографічного аналізу

Система позначень, використовувана для кожного випадку, визначає біти, які складаються за модулем *2*. Наприклад, *O (7, 8, 24, 29)* означає операцію *xor 7*, *8*, *24* і *29*-го бітів, що виходять із функції; *k(22)* означає *22*-й біт у ключі раунду; *I(15) — 15*-й біт, що входить у функцію.

Нижче показано відношення для частин рис. 5.24, а і рис. 5.24, б, що використовують індивідуальні біти:

- частина *а*: $O(7) \oplus O(8) \oplus O(24) \oplus O(29) = I(15) \oplus k(22);$
- частина *б*: $F(15) = I(29) \oplus k(42) \oplus k(43) \oplus k(45) \oplus k(46).$

Після створення й зберігання одноразундових характеристик аналітик може комбінувати різні раунди, щоб створити множинну характеристику раунду. Рис. 5.25 показує випадок трираундового *DES*, в якому раунди *1* і *3* використовують ту саму характеристику,

як це зображено на рис. 5.24, а, а в раунді *2* використано довільну характеристику.

Мета лінійного криптографічного аналізу полягає в тому, щоб знайти лінійне відношення між деякими бітами в парі "відкриті дані-зашифровані дані" та ключ. Чи можна встановити таке відношення для *DES* з трьома раундами, зображено на рис. 5.25.

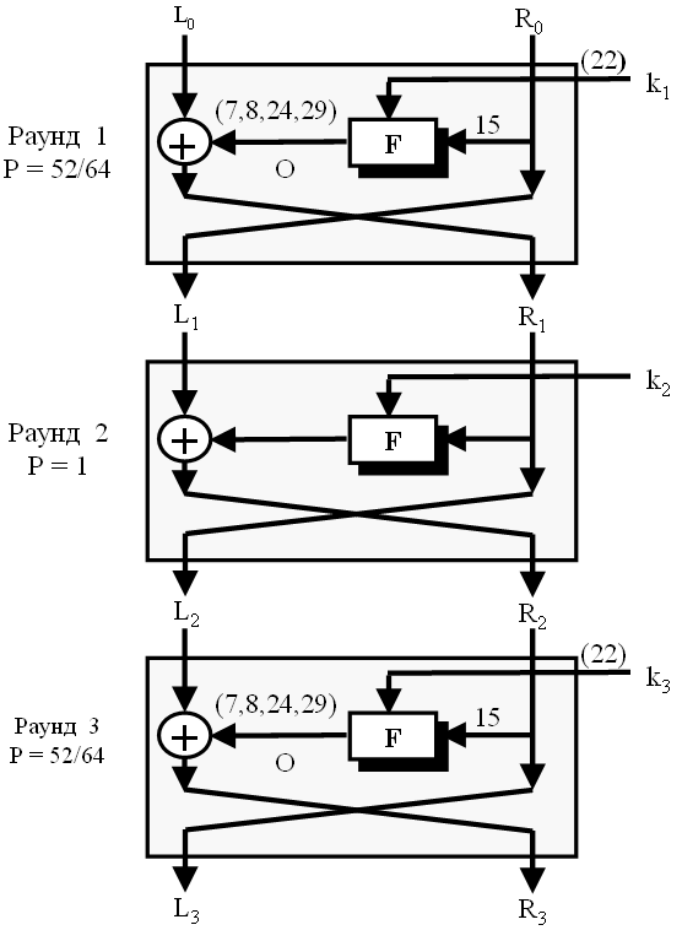


Рис. 5.25. Трираундові характеристики для лінійного криптографічного аналізу

Раунд *1*: $R_1(7, 8, 24, 29) = L_0(7, 8, 24, 29) \oplus R_0(15) \oplus k_1(22)$.
Раунд *3*: $L_3(7, 8, 24, 29) = L_2(7, 8,24, 29) \oplus R_2(15) \oplus k_3(22)$.

Але L_2 — той самий, що і R_1, і R_2 — той самий, що і R_3. Після заміни L_2 на R_1 і R_2 на R_3 у другому відношенні отримаємо:

$$L_3(7, 8, 24, 29) = R_1(7, 8, 24, 29) \oplus R_3(15) \oplus k_3(22).$$

Можна зазначити R_1 на його еквівалентне значення в раунді *1*, як результат отримаємо:

$$L_3(7, 8, 24, 29) = L_0(7, 8, 24, 29) \oplus R_0(15) \oplus$$
$$\oplus k_1(22) \oplus R_3(15) \oplus k_3(22).$$

Це відношення між бітами входу й виходу для всієї системи з трьох раундів після перетворень:

$$L_3(7, 8, 24, 29) \oplus R_3(15) = L_0(7, 8, 24, 29) \oplus$$
$$\oplus R_0(15) \oplus k_1(22) \oplus k_3(22).$$

Іншими словами, маємо:

$$C(7, 8, 15, 24, 29) = M(7, 8, 15, 24, 29) \oplus k_1(22) \oplus k_3(22).$$

Як знайти ймовірність трираундових (чи *n*-раундових) *DES*? Мацуі показав, імовірність в цьому випадку [6, 26]:

$$P = 0{,}5 + (2n-1)\prod_i (p_i - 0{,}5),$$

де n — кількість раундів; p_i — ймовірність кожної характеристики раунду; P — повна ймовірність.

Наприклад, повна ймовірність для трираундового аналізу на рис. 5.24.

$$P = 0{,}5 + (2\ 3 - 1)\ [(52/64 - 0{,}5) \times (1 - 0{,}5) \times (52/64 - 0{,}5)] = 0{,}695.$$

Шістнадцятираундова характеристика може також бути скомпільована, щоб забезпечити лінійні відношення між деякими бітами вхідних даних, деякими бітами зашифрованих даних і деякими бітами в ключах раунду:

$$C(\text{деякі біти}) = M(\text{деякі біти}) \oplus$$
$$\oplus k_1(\text{деякі біти}) \oplus \ldots \oplus k\ldots(\text{деякі біти}).$$

Після знаходження й збереження багатьох відношень між деякими бітами початкових даних, бітами зашифрованих даних і бітами ключів раунду зловмисник може звернутися до деяких пар початкових-зашифрованих даних (атака "знання відкритих даних") і використати відповідні біти зі збережених характеристик, щоб знайти біти в ключах раунду.

Відомо, що для того, щоб створити атаку на *16*-раундовий *DES*, потрібно 2^{43} відомих пар відкритих/зашифрованих даних. Лінійний криптографічний аналіз є ймовірнішим, ніж диференціальний криптографічний аналіз, з двох причин: перша — кількість кроків у ньому менша; друга — він простіший для атаки знання відкритих даних, ніж для атаки з вибіркою відкритих даних. Проте така атака все ще далека від того, щоб її серйозно побоюватися тим, хто працює з *DES*.

Контрольні питання та завдання

1. Який розмір блока в *DES*? Який розмір ключа шифру в *DES*? Який розмір ключів раунду в *DES*?
2. Яка кількість раундів у *DES*?
3. Скільки змішувачів і пристроїв заміни використовується за першого способу зашифрування та зворотного розшифрування? Скільки їх використовується за другого способу?
4. Скільки перестановок використовується в алгоритмі шифру *DES*?
5. Скільки операцій *xor* використовується в *DES*-шифрі?
6. Чому *DES*-функції необхідна операція, яка здійснює перестановку з розширенням?
7. Чому генератор ключів раунду потребує видалення перевірочних бітів?
8. Яка різниця між уразливим ключем, напівуразливим ключем і можливо вразливим ключем?
9. Що таке подвійний *DES*? Яка атака подвійного *DES* зробила його марним?
10. Що таке потрійний *DES*? Що таке потрійний *DES* із двома ключами? Що таке потрійний *DES* із трьома ключами?
11. Значення послідовності вхідних даних у шифрі *DES* дорівнює $1234567890abcdef_{16}$. Визначити значення послідовності на виході блока *IP*-перестановки.
12. Значення послідовності R_0 в шифрі *DES* дорівнює $R_0 = f0aae8a5_{16}$. Визначити значення послідовності на виході *E*-блока перестановки й розширення.

13. Значення послідовності R_1 в шифрі *DES* дорівнює $R_1 = 116ba133_{16}$. Визначити значення послідовності на виході *E*-блока перестановки й розширення.

14. Значення послідовності на виході *E*-блока перестановки і розширення в шифрі *DES* дорівнює $R_1^E = 8a2b57d029a6_{16}$. Значення раундового ключа $k_1 = 4568581abcce_{16}$. Визначити значення послідовності на виході операції *xor* функції Фейстеля змішувача.

15. Значення послідовності на виході *E*-блока перестановки й розширення в шифрі *DES* дорівнює $R_{12}^E = 9f3ca2bffea6_{16}$. Значення раундового ключа $k_{12} = c2c1e96a4bf3_{16}$. Визначити значення послідовності на виході операції *xor* функції Фейстеля змішувача.

16. Значення послідовності на виході функції *xor* змішувача в шифрі *DES* дорівнює $R_{12}^{\oplus} = 5dfd4bd5b555_{16}$. Визначити значення послідовності на виході *S*-блоків заміни змішувача: а) S_2; б) S_3; в) S_4; г) S_5.

17. Значення послідовності на виході функції *xor* змішувача в шифрі *DES* дорівнює $R_{13}^{\oplus} = e8848768_{16}$. Визначити значення послідовності на виході *S*-блоків заміни змішувача.

18. Значення послідовності на виході *S*-блоків заміни змішувача в шифрі *DES* дорівнює $R_2^S = d5b80915_{16}$. Визначити значення послідовності на виході *P*-блока заміни змішувача.

19. Значення послідовності на виході *P*-блока прямої перестановки змішувача в шифрі *DES* дорівнює $R_3^P = 07d0a03c_{16}$. Значення послідовності $L_3 = 4f73c3b3_{16}$. Визначити значення послідовності на виході функції *xor* змішувача.

20. Значення послідовностей на вході першого раунду зашифрування *DES* дорівнюють: $L_4 = 07eb4845_{16}$, $R_4 = 48a3638f_{16}$. Значення послідовності на виході *P*-блока заміни змішувача $R_4^P = 46932b6e_{16}$. Визначити значення послідовностей *L* та *R* на виході п'ятого раунду зашифрування *DES*.

21. Значення послідовностей на вході шостого раунду зашифрування *DES* дорівнюють: $L_5 = 48a3638f_{16}$, $R_5 = 4178632b_{16}$. Значення раундового ключа $k_6 = c1948e87475e_{16}$. Визначити значення послідовностей *L* і *R* на виході шостого раунду зашифрування *DES*.

22. Значення послідовності ключа шифру *DES* дорівнює $K = abba08192637cddc_{16}$. Визначити значення послідовностей C_0 і D_0 після виконання операції видалення перевірочних розрядів і перестановки.

23. Значення послідовностей C_0 і D_0 шифру *DES* дорівнюють: $C_0 = c3c033a_{16}$ і $D_0 = 33f0cfa_{16}$. Визначити значення послідовностей: а) C_2 і D_2; б) C_3 і D_3; в) C_7 і D_7; г) C_{12} і D_{12}.

24. Значення послідовностей C_0 і D_0 шифру *DES* дорівнюють: $C_0 = c3c033a_{16}$ і $D_0 = 33f0cfa_{16}$. Визначити значення раундового ключа k_2.

25. Значення послідовностей C_0 і D_0 шифру *DES* дорівнюють: $C_0 = c3c033a_{16}$ і $D_0 = 33f0cfa_{16}$. Визначити значення раундових ключів:

а) четвертого раундового ключа;

б) дев'ятого раундового ключа;

в) тринадцятого раундового ключа;

г) шістнадцятого раундового ключа.

26. Значення послідовностей на виході шістнадцятого раунду зашифрування *DES* дорівнюють: $L_{16} = c95320e2_{16}$, $R_{16} = 9b7b3602_{16}$. Визначити значення зашифрованих даних.

Розділ 6

РЕЖИМИ ВИКОНАННЯ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ БЛОКОВОГО СИМЕТРИЧНОГО ШИФРУВАННЯ ДАНИХ

Незабаром після *DES* у *США* був прийнятий ще один федеральний стандарт, що рекомендує чотири способи експлуатації (виконання) алгоритму *DES* для шифрування даних. Відтоді ці режими стали загальноприйнятими й застосовуються з будь-якими блоковими шифрами. Пізніше був доданий ще один режим.

Для будь-якого симетричного блокового алгоритму шифрування визначені п'ять режимів їх виконання:

1. *Режим електронної кодової книги* (*Electronic Code Book — ECB*). У цьому режимі виконання кожний блок незашифрованих даних шифрується незалежно від інших блоків, із застосуванням того самого ключа шифрування. Типові застосування — безпечна передача поодиноких значень (наприклад криптографічного ключа).

2. *Режим зчеплення блоків зашифрованих даних* (*Cipher Block Chaining — CBC*). У цьому режимі виконання вхід криптографічного алгоритму є результатом застосування операції *xor* до наступного блока незашифрованих і попереднього блока зашифрованих даних. Типові застосування — загальна блокоорієнтована передача, аутентифікація.

3. *Режим зворотного зв'язку за зашифрованими даними* (*Cipher Feedback — CFB*). У цьому режимі виконання за кожного виклику алгоритму обробляється l бітів вхідного значення. Попередній зашифрований блок використовується як вхід в алгоритм; до l бітів виходу алгоритму й наступного незашифрованого блока з l бітів застосовується операція *xor*, результатом якої є наступний зашифрований блок з l бітів. Типові застосування — потокоорієнтована передача, аутентифікація.

4. *Режим зворотного зв'язку за виходом* (*Output Feedback — OFB*). Цей режим виконання аналогічний режиму виконання *CFB*, за винятком того, що на вхід алгоритму при зашифруванні наступного блока подається результат зашифрування попереднього блока; тільки після цього виконується операція *xor* із черговими *l* бітами незашифрованих даних. Типові застосування — потокоорієнтована передача по зашумленому каналу (наприклад супутниковий зв'язок).

5. *Режим лічильника (Counter Mode — CTR*). Цей режим виконання дуже схожий на режим *OFB*, але замість використання випадкових унікальних значень вектора ініціалізації для генерації значень ключового потоку, цей режим використовує лічильник, значення якого додається до кожного блока відкритого тексту, який необхідно зашифрувати. Унікальне значення лічильника гарантує, що кожний блок об'єднується з унікальним значенням ключового потоку.

6.1. РЕЖИМ ВИКОНАННЯ "ЕЛЕКТРОННА КОДОВА КНИГА"

Режим електронної кодової книги ECB є простим серед стандартних способів використання блокових шифрів (рис. 6.1).

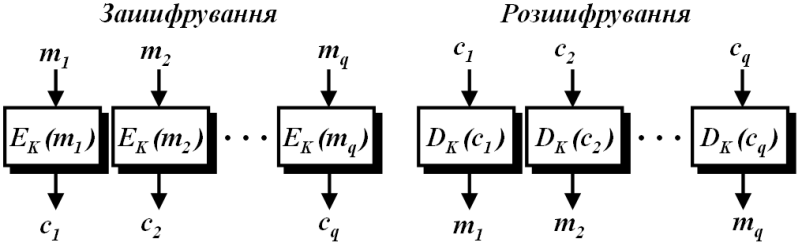


Рис. 6.1. Структура режиму виконання "*Електронна кодова книга*"

Дані, які належить шифрувати *M*, поділяються на блоки заданої довжини m_i. Останній із них, за необхідності, доповнюють. Кожний із цих блоків шифрують незалежно з використанням того самого ключа за шифрування

$$c_i = E_K(m_i), \tag{6.1}$$

де c_i — блоки зашифрованих даних; *E* — функція зашифрування; *K* — ключ шифру.

Процес розшифрування — просте обернення попередньої операції (6.1)

$$m_i = D_K(c_i), \tag{6.2}$$

де D — функція розшифрування.

Основна перевага цього режиму — простота реалізації. Проте з режимом *ECB* пов'язаний ряд проблем.

Перша проблема виникає через те, що за рівності $m_i = m_j$ виходять однакові блоки $c_i = c_j$, тобто однакові блоки на вході формують співпадаючі блоки на виході. Це, дійсно, проблема, оскільки шаблонні початок і кінець повідомлення співпадають, що дає криптографічному аналітику деяку інформацію про зміст повідомлення.

Друга проблема пов'язана з тим, що видалення з повідомлення якого-небудь блока не залишає слідів, і той, що атакує, може таким чином спотворити інформацію, що передається.

Третя — подібна до другої, але пов'язана зі вставкою блоків з інших повідомлень.

Щоб краще уявити собі ці проблеми, візьмемо просту модель шифру, в якій блок відповідає слову, і припустимо, що відкрите повідомлення:

"*плати Алісі сто фунтів, не плати Бобу двісті фунтів*"

у зашифрованому вигляді виглядають так:

"*у кішки чотири ноги, а у людини дві ноги*".

Можна тепер змусити одержувача сплатити Алісі дві сотні фунтів, замість однієї, відправивши йому повідомлення:

"*у кішки дві ноги*",

яке отримане з першого заміною одного з блоків на блок із другого повідомлення. Крім того, можна призупинити виплати Алісі, поставивши блок *A* зашифрованого повідомлення на початок першого повідомлення. *A* — зашифрований варіант частки "*не*" відкритого повідомлення. Можна спонукати одержувача криптограми виплатити *Бобу* двісті фунтів, якщо видалити блок *A* із зашифрованого повідомлення.

Таким атакам можна протистояти, додаючи контрольні суми декількох блоків відкритих даних або використовуючи режим, за

якого до кожного блока зашифрованих даних додається "*контекстний ідентифікатор*".

При зашифруванні 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 неправильно, але це не впливає на інші блоки розшифрованих даних. Проте, якщо біт зашифрованих даних випадково втрачений або доданий, то усі подальші зашифровані дані будуть розшифровані неправильно, якщо для вирівнювання меж блоків не використовується яка-небудь кадрова структура.

Більшість повідомлень точно не ділиться на *64*-бітові (чи будь-якого іншого розміру, наприклад *128* бітів) блоки для зашифрування, у кінці зазвичай використовується укорочений блок. Режим *ECB* вимагає використовувати *64*-бітові блоки. Способом вирішення цієї проблеми є набивання.

Останній блок доповнюється (набивається) деяким регулярним шаблоном: нулями; одиницями; нулями й одиницями, що чергуються, для отримання повного блока. За необхідності видалити набивання після розшифрування, в останній байт останнього блока записується кількість байтів набивання. Наприклад, нехай розмір блока — *64* біти й останній блок складається з трьох байтів (*24* біти). Для доповнення блока до *64* бітів потрібно п'ять байтів. Тому необхідно додати чотири байти набивання (наприклад нулів) і один байт довжини набивання, який повинен дорівнювати *6*. Після розшифрування необхідно взяти останній байт останнього блока і, визначивши довжину набивання, видалити з останнього блока останні п'ять байтів. Щоб цей метод працював правильно, кожне повідомлення необхідно доповнити (набити), навіть якщо відкриті дані містять ціле число блоків. У такому разі доведеться додати один повний блок, який буде мати сім байтів набивання й один байт довжини набивання.

6.2. РЕЖИМ ВИКОНАННЯ "ЗЧЕПЛЕННЯ БЛОКІВ ЗАШИФРОВАНИХ ДАНИХ"

Одним зі шляхів уникнення проблем, що виникають при використанні режиму *ECB*, є *зчеплення* зашифрованих блоків даних, тобто в додаванні до кожного зашифрованого блока даних контекстного ідентифікатора. Найпростіший спосіб зробити це — застосувати режим зчеплення блоків шифрованих даних або *CBC* (рис. 6.2).

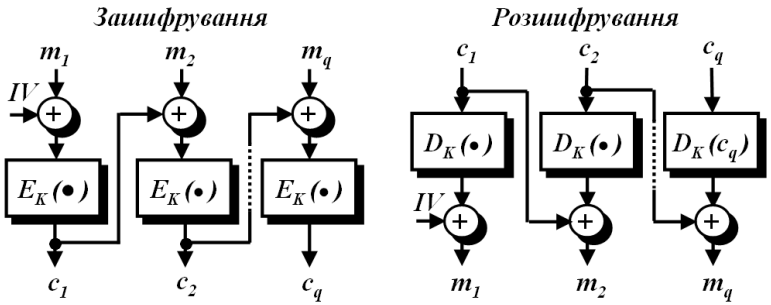


Рис. 6.2. Структура режиму виконання *"Зчеплення блоків шифрованих даних"*

У цьому режимі початкові дані, як завжди, розбиваються на серію блоків

$$m_1, m_2, m_3, \ldots, m_q ,$$

де q — кількість блоків початкових даних (з урахуванням набивання), яке підлягає зашифруванню.

Як і в попередньому режимі, останньому блоку потрібне доповнення, щоб довжина початкових даних стала кратна довжині блока.

Зашифрування здійснюється згідно з виразами:

$$c_1 = E_K(m_1 \oplus IV), \quad c_i = E_K(m_i \oplus c_{i-1}), \quad i=2,3,\ldots,q. \tag{6.3}$$

У ході зашифрування першого блока бере участь початкова величина *IV (initialization vector — вектор ініціалізації*), яку слід віднести до завдання функції, яка зашифровує. Величину *IV* застосовують для зашифрування, щоб зашифровані версії однакових частин початкових даних виглядали по-різному.

Величина *IV* бере участь також при розшифруванні. Цей процес виглядає таким чином:

$$m_1 = D_K(c_1) \oplus IV, \quad m_i = D_K(c_i) \oplus c_{i-1}, \quad i=2,3,\ldots,q. \tag{6.4}$$

Очевидно, що останній *64*-бітовий блок зашифрованих даних є функцією секретного ключа *K*, початкового значення *IV* і кожного біта початкових даних незалежно від його довжини. Цей блок зашифрованих даних називають *кодом аутентифікації повідомлення* (*message authentication code — MAC*).

MAC може легко перевірити одержувач, який володіє секретним ключем *K* і початковим значенням *IV*, шляхом повторення процедури, виконаної відправником. Проте сторонній, не може здійснити генерацію *MAC*, який би одержувач 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 (6.4), 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 початкових даних m_{i-1} і m_i.

Як випливає з рис. 6.2, для зашифрування 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. Вдалим *IV* служить мітка часу або випадкова послідовність бітів.

Із використанням *IV* повідомлення з ідентичними початковими даними в ході зашифрування 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 таємниці, він може передаватися відкрито разом із зашифрованими даними. Нехай повідомлення, що передається, складається з декількох блоків $m_1, m_2, \ldots, m_q$. m_1 зашифровується з *IV*. m_2 зашифровується з використанням зашифрованих даних c_1 у ролі *IV*. m_3 шифрується з використанням зашифрованих даних c_2 у ролі *IV* і так далі. Отже, якщо кількість блоків даних дорівнює q, то $q-1$ "векторів ініціалізації" відкриті, навіть якщо первинний *IV* зберігається в таємниці. Тому причин зберігати в таємниці *IV* немає, *IV* — це просто блок-заглушка, його можна вважати нульовим блоком зчеплення m_0.

Набивання використовується так само, як і в режимі *ECB*, але в деякому застосуванні розмір зашифрованих даних повинен точно співпадати з розміром початкових 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. 6.3), необхідно знову зробити зашифрування q-1-го блока зашифрованих даних. Потім, вибравши з отриманого блока зашифрованих даних l старших (лівих) бітів, виконати для них й короткого блока операцію *xor*, створюючи остаточно зашифровані дані.

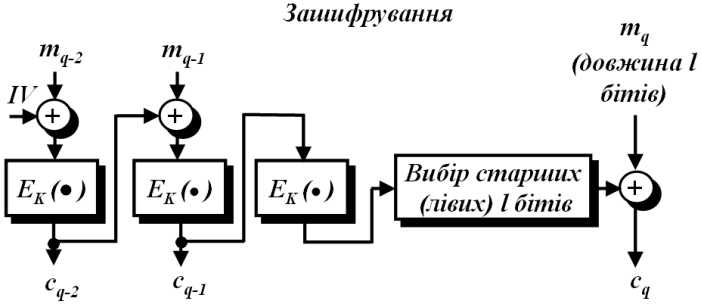


Рис. 6.3. Зашифрування короткого останнього блока в режимі *CBC*

6.3. РЕЖИМ ВИКОНАННЯ “ЗВОРОТНИЙ ЗВ'ЯЗОК”

Режими *ECB* і *CBC* призначені для зашифрування й розшифрування блоків повідомлень. У режимі *ECB* і *CBC* зашифрування даних не почнеться, якщо не буде отримано цілий блок даних. Це створює проблеми для деяких мережевих застосувань. 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 вимоги.

6.3.1. Режим “Зворотний зв'язок за зашифрованими даними”

Рішення полягає в тому, що необхідно застосувати алгоритми симетричного блокового шифрування або інші в режимі зворотного зв'язку за зашифрованими даними (*CFB*). У цьому режимі розмір блока n, але розмір блока початкових або зашифрованих даних — l, де $l < n$.

Ідея полягає в тому, що алгоритми симетричного блокового шифрування використовуються не для того, щоб зашифрувати початкові або розшифрувати зашифровані дані, а для того, щоб зашифрувати або

розшифрувати зміст регістра зсуву, розміром n. Зашифрування зроблене із застосуванням операції *xor* до l-бітового блока початкових даних з l-бітовим регістром зсуву (рис. 6.4).

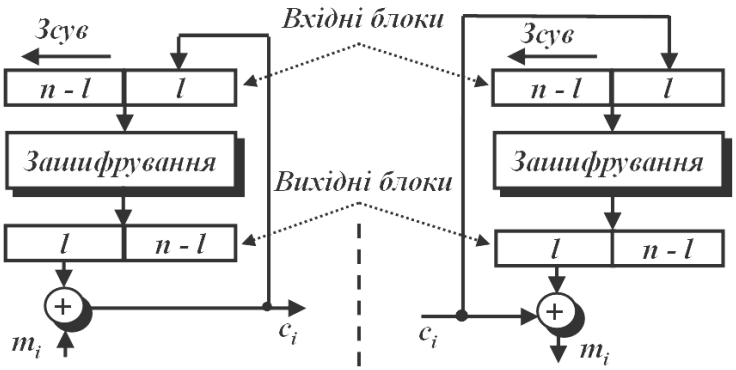


Рис. 6.4. Структура режиму виконання "*Зворотний зв'язок за зашифрованими даними*"

Вхідний блок (регістр зсуву вліво) при зашифруванні спочатку містить вектор ініціалізації, вирівняний по правому краю.

Припустимо, що внаслідок розбиття початкових даних на блоки, отримано q блоків завдовжки l бітів кожен. Тоді кожний i-й блок зашифрованих даних буде визначатися таким чином:

$$c_i = m_i \oplus g_{i-1}, \quad i=1,2,\ldots,q, \tag{6.5}$$

де g_{i-1} означає l старших бітів попереднього вихідного блока зашифрованих даних.

Оновлення регістра зсуву (вхідного блока) здійснюється шляхом зсуву його даних вліво на l бітів і запису c_i на місце l молодших (правих) розрядів.

Розшифрування виконано із застосуванням операції *xor* до l-бітового блока зашифрованих даних з l-бітовим регістром зсуву. Вхідний блок (регістр зсуву вліво) при розшифруванні спочатку містить вектор ініціалізації, вирівняний по правому краю. Зауважимо, що зміст регістра зсуву для першого блока — це IV — не зсувається.

Тоді кожний *i*-й блок розшифрованих (початкових) даних буде визначатися таким чином:

$$m_i = c_i \oplus g_{i-l}, \quad i=1,2,...,q, \tag{6.6}$$

де g_{i-l} означає l старших бітів попереднього вихідного блока розшифрованих даних.

Для кожного блока регістр зсуву виконує зсув змісту (попередній регістр зсуву) на l бітів вліво, заповнюючи крайні праві l бітів із c_{i-l}. Зміст регістра зсуву розшифровується, і тільки крайні ліві l біти обробляються за допомогою *xor* із зашифрованими даними, із блока c_i отримуючи m_i. Зауважимо, що зміст регістра зсуву для першого блока — це *IV* — не зсувається.

Цікаво, що в цьому режимі не потрібне доповнення блоків, тому що розмір блоків l зазвичай вибирається так, щоб задовольнити розмір блока даних, який потрібно зашифрувати (наприклад символ); а також система не повинна чекати отримання великого блока даних (*64* біти або *128* бітів) для того, щоб почати шифрування. Процес шифрування виконується для маленького блока даних (таких, як символ). Ці дві переваги призводять до двох недоліків: *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* поєднує разом символи відкритих даних так, що зашифровані дані залежать від усіх попередніх початкових даних.

Як і в режимі *CBC IV* повинен бути унікальним, крім того, *IV* повинен змінюватися для кожного повідомлення.

У режимі *CFB* помилка у відкритих даних впливає на всі подальші зашифровані дані, але самоусувається в ході розшифрування. Набагато цікавіша помилка в зашифрованих даних. Першим ефектом збою біта зашифрованих даних є збій одного біта розшифрованих даних. Потім помилка потрапляє в регістр зсуву, і доки збійний біт не вийде з регістра, будуть формуватися неправильні розшифровані дані.

6.3.2. Режим “Зворотний зв'язок за виходом”

Режим виконання *OFB* теж використовує змінний розмір блока й регістр зсуву, що ініціалізується так само, як у режимі *CFB*, а саме: вхідний блок спочатку містить початкове значення *IV*, вирівняне по правому краю (рис. 6.5).

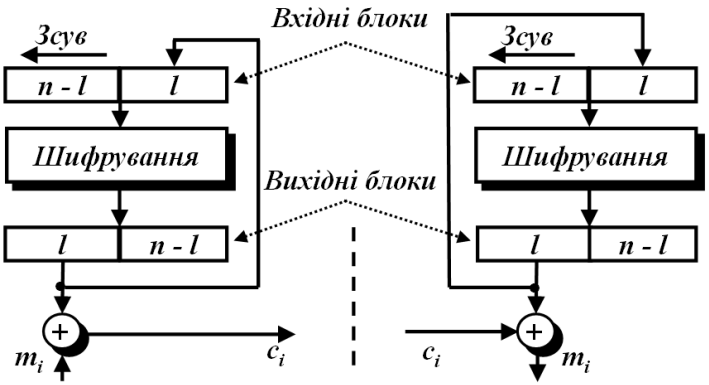


Рис. 6.5. Структура режиму виконання “*Зворотний зв'язок за виходом”*

При цьому для кожного сеансу шифрування даних необхідно використати нове початкове значення стану регістра, яке повинне пересилатися по каналу відкритими даними.

Припустимо, що відкрите повідомлення надано у вигляді блоків

$$m_1, m_2, \ldots, m_q.$$

Для усіх $i = 1, 2, 3,\ldots,q$

$$c_i = m_i \oplus g_{i-1}. \qquad (6.7)$$

Відмінність від режиму зворотного зв'язку за зашифрованими даними полягає в методі оновлення регістра зсуву. Це здійснюється шляхом відкидання старших l бітів і дописування справа g_i.

Основна перевага режиму *OFB* полягає в тому, що якщо під час передачі сталася помилка, то вона не поширюється на наступні зашифровані блоки, і тим самим зберігається можливість розшифрування подальших блоків. Наприклад, якщо з'являється помилковий біт у c_i, то це призведе тільки до неможливості розшифрування

цього блока й отримання m_i. Подальша послідовність блоків буде розшифрована коректно.

Недолік *OFB* у тому, що він більш уразливий до атак модифікації потоку повідомлень, ніж *CFB*.

6.4. РЕЖИМ ВИКОНАННЯ "ЛІЧИЛЬНИК"

У режимі лічильника (*CTR*) немає інформації зворотного зв'язку. Псевдовипадковий ключовий потік досягається за допомогою лічильника. Лічильник на n бітів 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. Початкові й зашифровані дані мають той самий розмір блока, як і основний шифр. Блоки розміру n початкових даних зашифровані так, щоб створити зашифровані дані з блоком розміру n. На рис. 6.6 показано зашифрування даних у режимі лічильника.

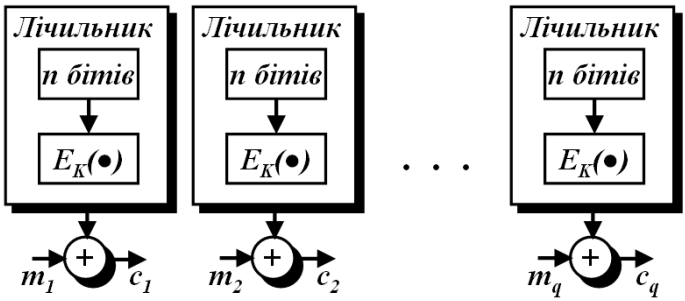


Рис. 6.6. Структура зашифрування даних у режимі виконання "Лічильник"

Відношення між початковими даними та блоками зашифрованих даних показане нижче.

Зашифрування:

$$c_i = m_i \oplus E_{K_i}(n_i),$$

де n_i — n-бітове значення лічильника.

Розшифрування (рис. 6.7):

$$m_i = c_i \oplus E_{K_i}(n_i).$$

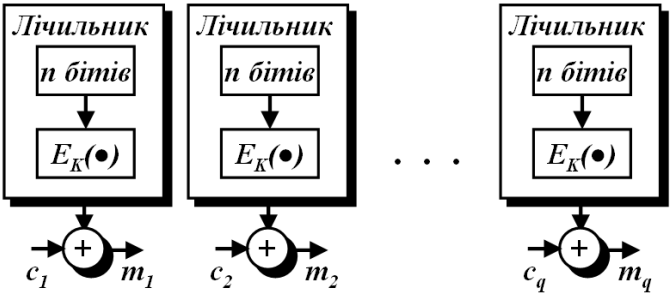


Рис. 6.7. Структура розшифрування даних у режимі виконання "Лічильник"

Режим *CTR* використовує функцію зашифрування основного блокового шифру (E_K) і для за шифрування (рис. 6.6), і для розшифрування (рис. 6.7). Досить легко довести, що блок m_i початкових даних може бути відновлений із зашифрованих даних c_i.

Можна порівняти режим *CTR* із режимами *OFB* і *ECB*. Подібно до *OFB*, *CTR* створює ключовий потік, незалежний від попереднього блока зашифрованих даних, але *CTR* не використовує інформацію зворотного зв'язку. Так само як *ECB*, *CTR* створює *n*-бітові зашифровані дані, блоки якого 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 зашифруванням закінчений *n*-розрядний блок даних. Позитивною стороною цієї властивості є те, що режим *CTR*, подібно до режиму *ECB*, може використовуватися, щоб зашифрувати й розшифрувати файли довільного доступу, і значення лічильника може бути пов'язане з номером запису у файлі.

Проблеми безпеки для режиму *CTR* ті самі, що й для режиму *OFB*.

Єдина помилка в зашифрованих даних стосується тільки відповідного біта в розшифрованих даних.

6.5. ІНШІ РЕЖИМИ ШИФРУВАННЯ

Існує також ряд додаткових режимів шифрування даних за допомогою блокових симетричних криптографічних алгоритмів [22]:

- *режим розповсюдженого зчеплення блоків зашифрованих даних* (*Propagating Cipher Block Chaining – PCBC*);

- *режим зчеплення блоків зашифрованих даних з контрольною сумою* (*Cipher Block Chaining with Checksum – CBCC*);

- *режим оберненого зв'язку за виходом із нелінійною функцією* (*Output Feedback With a Nonlinear Function – OFBNLF*);

- *режим оберненого зв'язку за відкритими даними (PFB);*

- *зчеплення блоків відкритих даних (PBC) тощо.*

Режим розповсюдженого зчеплення блоків зашифрованих даних. Однією з потенційних проблем режиму *CBC* є можливість внесення контрольованих змін у наступний розшифрований блок відкритих даних. Наприклад, якщо зловмисник змінить один біт у блоці, то весь блок буде розшифровано неправильно, але в наступному блоці з'явиться помилка у відповідній позиції. Є ситуації, коли це небажано. Для боротьби з цією загрозою відкриті дані повинні містити певну надмірність.

Режим *PCBC* подібний до режиму *CBC*, за виключенням того, що як попередній блок відкритих даних, так і попередній блок зашифрованих даних зазнають операції *xor* із поточним блоком відкритих даних перед зашифруванням (рис. 6.8, а) або після розшифрування (рис. 6.8, б).

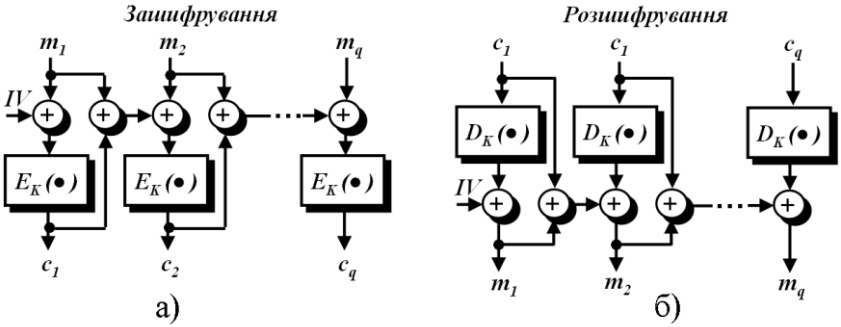


Рис. 6.8. Структура шифрування даних у режимі розповсюдженого зчеплення блоків:
а) зашифрування; б) розшифрування

У цьому випадку зашифрування й розшифрування даних здійснюється з використанням співвідношень:

$$c_i = E_k (m_i \oplus c_{i-1} \oplus m_{i-1}),\ i=1,2,...,q,\ c_0=IV;$$

$$m_i = c_{i-1} \oplus m_{i-1} \oplus D_k (c_i),\ i=1,2,...,q,\ c_0=IV.$$

Режим *PCBC* використовується в протоколі *Kerberos* для виконання за один прохід шифрування та для перевірки цілісності даних. У режимі *PCBC* помилка у зашифрованих даних спричиняє некоректне розшифрування всіх наступних блоків.

Це означає, що перевірка стандартного блока в кінці повідомлення використовується для перевірки цілісності повідомлень.

Недолік цього режиму також полягає в тому, що перестановка двох блоків зашифрованих даних призводить до некоректного розшифрування двох відповідних блоків відкритих даних. Однак завдяки природі операції *xor* над відкритими і зашифрованими даними, подальші помилки компенсуються. Тому якщо перевірка цілісності охоплює лише кілька останніх блоків розшифрованих (відкритих) даних, можна отримати частково спотворене повідомлення.

Це сумнівна властивість змусила розробників відмовитися від цього режиму на користь *CBC* в наступній версії протоколу *Kerberos*.

Режим зчеплення блоків зашифрованих даних з контрольною сумою. Режим *CBCC* відрізняється від режиму *CBC* тільки тим, що до останнього блока відкритих даних перед зашифруванням додається сума за модулем *2* для всіх попередніх блоків відкритих даних (рис. 6.9).

Це дає можливість проконтролювати цілісність переданих даних з невеликими додатковими накладними витратами.

У цьому випадку зашифрування даних здійснюється з використанням співвідношень:

$$s = (m_1 \oplus m_{i+1} \oplus ... \oplus m_{q-1});$$

$$c_i = E_k (m_i \oplus c_{i-1}),\ i=1,2,...,q-1,\ c_0=IV;$$

$$c_q = E_k (m_q \oplus s \oplus c_{q-1}),$$

а розшифрування — використанням співвідношень:

$$m_i = D_k(c_i) \oplus c_{i-1},\ i=1,2,\ldots,q\text{-}1,\ c_0=IV;$$

$$s = (m_1 \oplus m_{i+1} \oplus \ldots \oplus m_{q-1});$$

$$m_q = E_k(c_q) \oplus c_{q-1} \oplus s).$$

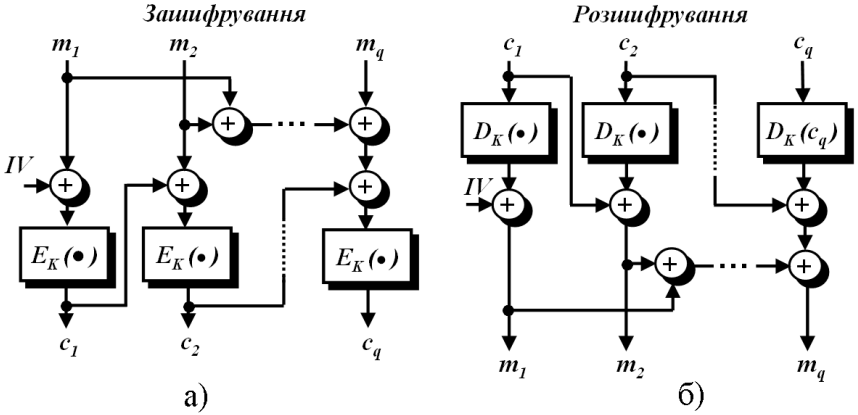


Рис. 6.9. Структура шифрування даних у режимі зчеплення блоків зашифрованого тексту з контрольною сумою:
а) зашифрування; б) розшифрування

Одинична помилка в зашифрованих даних поширюється тільки на один блок відкритих даних, однак потрібна підтримка синхронізації.

Швидкість обробки інформації визначається не тільки швидкістю шифрування базовим алгоритмом, але й швидкістю оновлення поточного значення ключа.

Режим зчеплення блоків відкритих даних (*Plaintext Block Chaining — PBC*) подібний до *CBC* за винятком того, що операція *xor* виконується над поточним і попереднім блоками відкритих даних, а не над блоком зашифрованих даних.

Режим зі зворотним зв'язком за відкритими даними (*Plaintext Feedback — PFB*) подібний до режиму *CFB*, однак для зворотного зв'язку використовуються не зашифровані дані, а відкриті дані. Для підвищення стійкості до зламування обидва режими допускають використання відкритих даних.

Контрольні питання та завдання

1. Пояснити, чому необхідні режими роботи, якщо для шифрування використовуються сучасні блокові шифри.

2. Перерахувати основні режими роботи блокових симетричних шифрів.

3. Визначити режим *ECB* і перерахувати його переваги та недоліки.

4. Визначити режим *CBC* і перерахувати його переваги та недоліки.

5. Визначити режим *CFB* і перерахувати його переваги та недоліки.

6. Визначити режим *OFB* і перерахувати його переваги та недоліки.

7. Визначити режим *CTR* і перерахувати його переваги та недоліки.

8. Розділити основні режими роботи на дві групи: ті, які використовують функції зашифрування й розшифрування, — основні шифри (наприклад, *DES* або *AES*), і ті, які використовують тільки функцію зашифрування.

9. Розділити основні режими роботи на дві групи: ті, які вимагають доповнення даних, і ті, які не вимагають цього.

10. Розділити 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. Перерахувати режими роботи, які можуть бути прискорені паралельною обробкою.

12. Перерахуйте режими роботи, які можуть використовуватися для зашифрування файлів довільного доступу.

13. Показати, чому режим *CFB* створює несинхронний шифр потоку, а режим *OFB* створює синхронний.

14. Скільки блоків охоплює єдиний біт помилки в передачі в режимі *CFB*?

15. У режимі *ECB* ($n = 64$) у першому блоку зашифрованих даних зруйновано один (*5*) біт протягом передачі. Знайти можливі зруйновані біти в розшифрованих даних.

16. У режимі *CBC* ($n = 64$) у першому блоці зашифрованих даних зруйновано один (*5*) біт протягом передачі. Знайти можливі зруйновані біти в розшифрованих даних.

17. У режимі *CFB* ($l = 8, n = 64$) другий біт першого блока відкритих даних зруйновано. Що буде відбуватися при розшифруванні даних?

18. У режимі *CFB* ($l = 8, n = 64$) другий біт першого блока зашифрованих даних зруйновано. Знайти можливі зруйновані біти в розшифрованих даних.

19. У режимі *OFB* ($l = 8$, $n = 64$) другий біт першого блока відкритих даних зруйновано. Що буде відбуватися при розшифруванні даних?

20. У режимі *CTR* ($n = 64$) другий біт другого блока зашифрованих даних зруйновано. Що буде відбуватися при розшифруванні даних?

21. Довести, що вихідні дані, використовувані відправником, можуть бути відновлені одержувачем у режимі *CFB*.

22. Довести, що вихідні дані, використовувані відправником, можуть бути відновлені одержувачем у режимі *OFB*.

23. Довести, що вихідні дані, використовувані відправником, можуть бути відновлені одержувачем у режимі *CTR*.

Розділ 7

СИМЕТРИЧНИЙ АЛГОРИТМ БЛОКОВОГО ШИФРУВАННЯ ДАНИХ INTERNATIONAL DATA ENCRYPTION ALGORITHM

7.1. ІСТОРІЯ СТВОРЕННЯ АЛГОРИТМУ

Алгоритм *IDEA* (англ. *International Data Encryption Algorithm* — Міжнародний алгоритм шифрування даних) є симетричним блоковим шифром, запатентований швейцарською фірмою *Ascom*. У листопаді 2000 р. *IDEA* був представлений як кандидат у проекті *NESSIE* в рамках програми Європейської комісії *IST* (англ. *Information Societies Technology* — інформаційні громадські технології).

Перша версія алгоритму *IDEA* була запропонована 1990 р., її автори — *Сюецзя Лай* і *Джеймс Мессі* зі Швейцарського інституту *ETH Zürich* (за контрактом із *Hasler Foundation*, яка пізніше влилася в *Ascom — Tech AG*) як заміна *DES* і назвали її *PES* (англ. *Proposed Encryption Standard* — запропонований стандарт шифрування). Потім, після публікації робіт Біхама й Шаміра за диференціального криптографічного аналізу *PES,* алгоритм був удосконалений з метою посилення криптографічної стійкості й названий *IPES* (англ. *Improved Proposed Encryption Standard* — удосконалений запропонований стандарт шифрування). Через рік його перейменували в *IDEA*.

Переглянуто версію алгоритму, посилену засобами захисту від диференціальних криптографічних атак, було представлено 1991 р. й детально описано 1992 р.

DEA є одним із декількох симетричних криптографічних алгоритмів, якими спочатку передбачалося замінити *DES*.

7.2. ПРИНЦИПИ ПОБУДОВИ АЛГОРИТМУ

Як і більшість інших блокових шифрів, алгоритм *IDEA* використовує при шифруванні процеси змішування й розсіювання, причому всі процеси легко реалізуються апаратними та програмними засобами.

IDEA оперує *64*-бітовими блоками початкових даних. Безперечною перевагою алгоритму *IDEA* є те, що його ключ має довжину *128* бітів. Той самий алгоритм використовується також для зашифрування й розшифрування.

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

Характеристики *IDEA*, що характеризують криптографічну стійкість алгоритму:

довжина блока: довжина блока повинна бути достатньою, щоб приховати всі статистичні характеристики початкового повідомлення. З іншого боку, складність реалізації криптографічної функції зростає експоненціально відповідно до розміру блока. Використання блока розміром у *64* біти в 90-ті рр. XX ст. означало достатню силу. Більше того, використання режиму шифрування CBC говорить про подальше посилення цього аспекту алгоритму;

довжина ключа: довжина ключа повинна бути досить великою для того, щоб запобігти можливості простого перебору ключа. За довжини ключа *128* бітів *IDEA* вважається досить безпечним;

конфузія: зашифровані дані повинні залежати від ключа складним і заплутаним способом;

дифузія: кожний біт початкових даних повинен впливати на кожний біт зашифрованих даних. Поширення одного незашифрованого біта на велику кількість зашифрованих бітів приховує статистичну структуру початкових даних. Визначити, як статистичні характеристики зашифрованих даних залежать від статистичних характеристик початкових даних, повинно бути непросто. *IDEA* з цього погляду є дуже ефективним алгоритмом.

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

Кожна операція виконується над двома *16*-бітовими входами та створює один *16*-бітовий вихід. Цими операціями є:

1. Побітове *xor*, що позначається як $\oplus$.

2. Сума цілих чисел за модулем 2^{16} (*65536*), при цьому входи й виходи трактуються як беззнакові *16*-бітові цілі. Цю операцію позначають як $\boxplus$.

3. Множення цілих чисел за модулем $2^{16} + 1$ (*65537*), при цьому входи й виходи трактуються як беззнакові *16*-бітові цілі, за винятком того, що блок із одних нулів трактується як 2^{17}. Цю операцію позначають як $\otimes$.

Ці три операції є несумісними в тому сенсі, що:

1. Не існує пари з трьох операцій, що задовольняють дистрибутивному закону. Наприклад,

$$a \otimes (b \boxplus c) \neq a \otimes b \boxplus a \otimes c.$$

2. Не існує пари з трьох операцій, що задовольняють асоціативному закону. Наприклад,

$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c.$$

Використання комбінації з цих трьох операцій забезпечує комплексну трансформацію входу, роблячи криптографічний аналіз складнішим, ніж у такому алгоритмі як *DES*, заснованому виключно на функції *xor*.

7.3. СТРУКТУРА АЛГОРИТМУ ШИФРУВАННЯ ДАНИХ

Розглянемо загальну схему шифрування *IDEA* (рис. 7.1). Як і в будь-якому алгоритмі шифрування, тут існує два входи: незашифрований блок і ключ. У даному випадку незашифрований блок має довжину *64* біти, ключ має довжину *128* бітів.

Алгоритм *IDEA* складається з восьми раундів, за якими йде кінцеве перетворення. Алгоритм поділяє блок даних на чотири *16*-бітові підблоки. Кожний раунд отримує на вході чотири *16*-бітові підблоки та створює чотири *16*-бітові вихідні підблоки. Кінцеве перетворення також отримує на вході чотири *16*-ти бітові підблоки та створює на виході чотири *16*-бітові підблоки. Кожний раунд використовує шість *16*-бітових раундових ключів, кінцеве перетворення використовує тільки чотири *16*-бітові раундові ключі, тобто всього в алгоритмі використовується *52* раундових ключі.

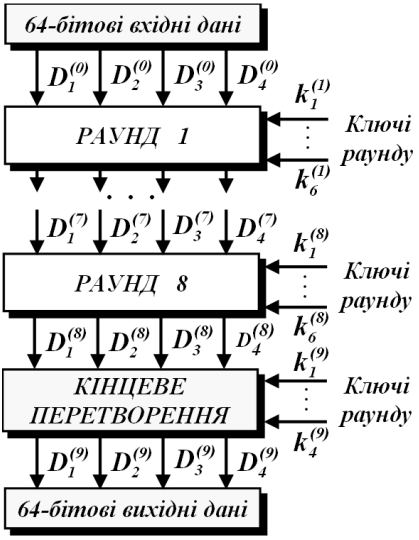


Рис. 7.1. Структура алгоритму *IDEA*

Одним з основних елементів алгоритму, що забезпечують дифузію, є структура, так звана *MA* (англ. *multiplication/addition* — множення/складання). Структуру пристрою *MA* показано на рис. 7.2.

На вхід цієї структури подаються два *16*-бітові значення ($f_1^{(i)}$ і $f_2^{(i)}$) і два *16*-бітові раундові ключі ($k_1^{(i)}$ і $k_2^{(i)}$), на виході створюються два *16*-бітові значення ($y_1^{(i)}$ і $y_2^{(i)}$).

Вичерпна комп'ютерна перевірка показує, що кожний біт виходу цієї структури залежить від кожного біта входів незашифрованого блока даних і від кожного біта раундових ключів.

Ця структура повторюється в алгоритмі вісім разів, забезпечуючи високоефективну дифузію.

Розглянемо послідовність перетворень окремого раунду. Раунд починається з перетворення (рис. 7.3), яке комбінує чотири вхідні підблоки даних: $D_1^{(i)}$, $D_2^{(i)}$, $D_3^{(i)}$ і $D_4^{(i)}$ кожний довжиною *16* бітів з чотирма раундовими ключами: $k_1^{(i)}$, $k_2^{(i)}$, $k_3^{(i)}$ і $k_4^{(i)}$ кожний довжиною також *16* бітів, використовуючи операції складання та множення.

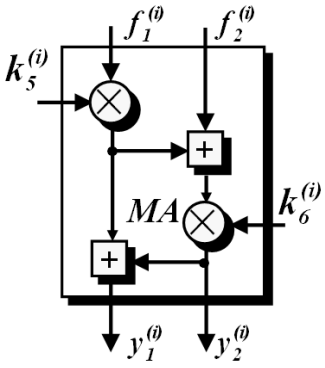


Рис. 7.2. Структура пристрою "множення/складання"

Чотири вихідних блоки цього перетворення комбінуються, використовуючи операцію *xor* для формування двох *16*-бітових підблоків, які є входами *MA* структури. Крім того, *MA* структура має на вході ще два раундових ключі і створює два *16*-бітові підблоки виходу.

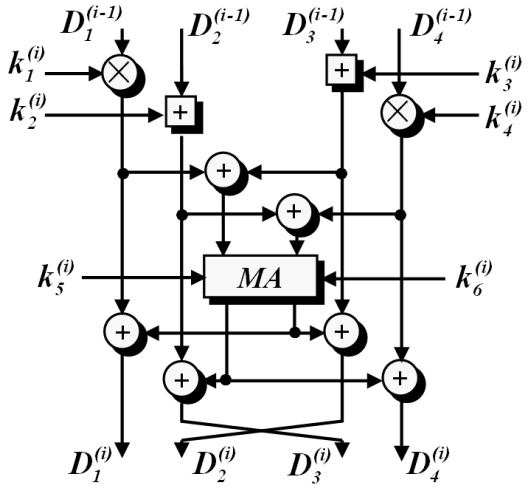


Рис. 7.3. Один раунд перетворення даних в *IDEA*

На закінчення чотири вихідних підблоки першого перетворення комбінуються з двома вихідними підблоками *MA* структури, використовуючи *xor* для створення чотирьох вихідних підблоків цієї ітерації. Зазначимо, що два виходи, які частково створюються другим і третім входами, міняються місцями для створення другого та третього виходів ($D_2^{(i)}$ і $D_3^{(i)}$). Це збільшує перемішування бітів і робить алгоритм стійкішим для диференціального криптографічного аналізу.

Розглянемо дев'ятий раунд алгоритму, позначений як кінцеве перетворення (рис. 7.4).

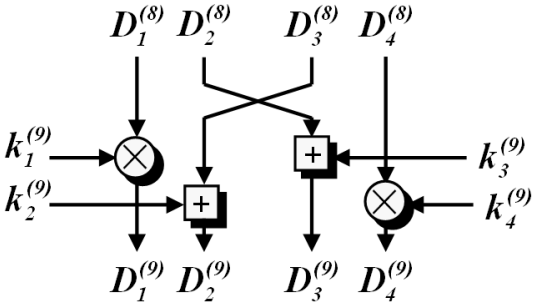


Рис. 7.4. Кінцеве перетворення (раунд *9*) в *IDEA*

Це така сама структура, що була описана вище. Єдина різниця полягає в тому, що другий і третій входи підблоків даних міняються місцями. Це зроблено для того, щоб розшифрування мало ту саму структуру, що й зашифрування. Зазначимо, що дев'ятий раунд (кінцеве перетворення) вимагає тільки чотири раундових ключі, тоді як для перших восьми раундів для кожної з них потрібно шість раундових ключів.

7.4. ГЕНЕРАЦІЯ РАУНДОВИХ КЛЮЧІВ ДЛЯ ШИФРУВАННЯ ДАНИХ

7.4.1. Генерація раундових ключів для за шифрування даних

Алгоритм розгортання ключа визначає порядок отримання раундових ключів із початкового ключа шифрування *K*. Раундові ключі виходять із ключа шифрування за допомогою алгоритму вироблення ключів. Він містить два компоненти:

- розширення ключа шифрування *K*;
- вибір раундових ключів.

Основні принципи алгоритму виглядають таким чином:

- ключ шифрування *K* розширюється в розширений ключ K_P;
- кількість бітів кожного раундового ключа дорівнює довжині підблока даних;
- кількість раундових ключів визначається з розрахунку шести ключів на кожний раунд зашифрування або розшифрування даних (*48* ключів, кожний завдовжки *16* бітів) і чотири ключі на кінцеве перетворення даних (кожний завдовжки також *16* бітів).

Разом повинно бути згенеровано п'ятдесят два *16*-бітових раундових ключі.

Розширення ключа шифрування *K* здійснюється таким чином: ключ шифрування *K* поділяється на вісім частин по *16* бітів кожна. Як результат виходять перші вісім ключів для зашифрування даних, які позначимо як k_1, k_2, ..., k_8, при цьому ключ k_1 дорівнює першим *16* бітам ключа шифрування *K*, k_2 дорівнює наступним *16* бітам ключа *K* і так далі. Потім відбувається циклічний зсув ключа шифрування *K* вліво на *25* бітів. Отримана *128*-розрядна послідовність також поділяється на вісім частин по *16* бітів кожна. Як результат

виходять другі вісім ключів для зашифрування даних, які позначимо як k_8, k_9, ..., k_{17}. Ця процедура повторюється до тих пір, поки не буде створено *56* ключів. Останні чотири ключі (k_{53}, k_{54}, k_{55} і k_{56}) у шифрі *IDEA* не використовуються і вони просто відкидаються.

Вибір раундових ключів зашифрування здійснюється з розширеного ключа K_P так: як ключі першого раунду $k_1^{(1)}$, $k_2^{(1)}$, $k_3^{(1)}$, $k_4^{(1)}$, $k_5^{(1)}$ і $k_6^{(1)}$ беруться перші шість ключів розширеного ключа K_P — k_1, k_2, k_3, k_4, k_5 і k_6 відповідно; як ключі другого раунду $k_1^{(2)}$, $k_2^{(2)}$, $k_3^{(2)}$, $k_4^{(2)}$, $k_5^{(2)}$ і $k_6^{(2)}$ беруться другі шість ключів розширеного ключа K_P — k_7, k_8, k_9, k_{10}, k_{11} і k_{12} відповідно і так далі. Як ключі кінцевого перетворення (дев'ятого раунду) $k_1^{(9)}$, $k_2^{(9)}$, $k_3^{(9)}$ і $k_4^{(9)}$ беруться останні чотири ключі розширеного ключа K_P — k_{49}, k_{50}, k_{51} і k_{52} відповідно.

Процес отримання розширення ключа зашифрування K_P із ключа шифрування K, а також вибору раундових ключів показано на рис. 7.5 і зведено в табл. 7.1.

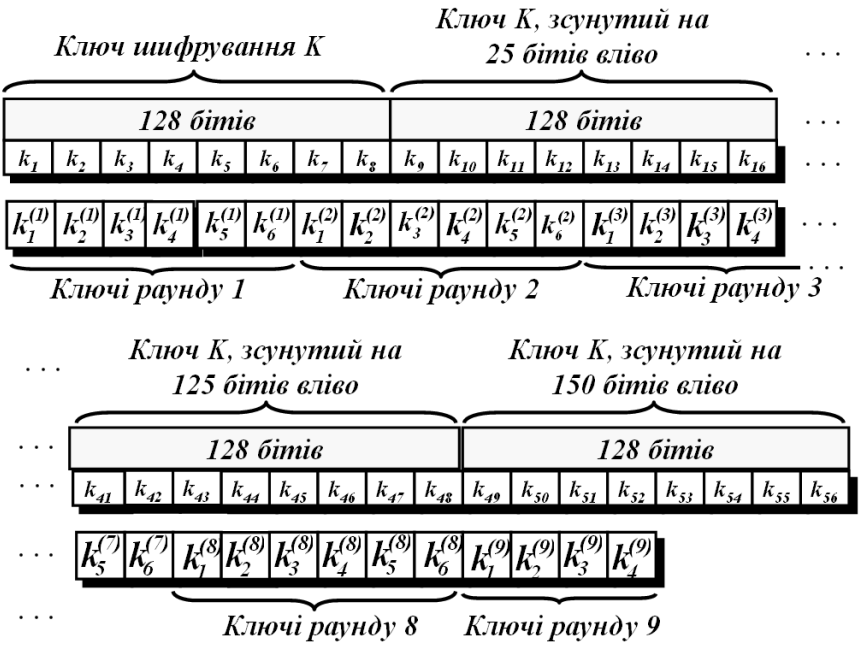


Рис. 7.5. Процес отримання раундових ключів зашифрування

Зазначимо, що кожний перший ключ раунду отриманий зі своєї підмножини бітів ключа *K*. Якщо весь ключ позначити як *K[1…128]*, то першими ключами у восьми раундах будуть ключі, що включають біти підмножини бітів ключа *K* (табл. 7.2).

Хоча в кожному раунді за винятком першого й восьмого використовуються тільки *96* бітів ключа шифрування *K*, множина бітів цього ключа на кожній ітерації не перетинаються, і не існує відношення простого зсуву між ключами різних раундів. Це відбувається, тому що в кожному раунді використовується тільки шість раундових ключів, тоді як при кожній ротації ключа виходить вісім ключів розширеного ключа K_P.

Таблиця 7.1

Ключі для кожного раунду зашифрування *IDEA*

Номер раунду	Номер ключа в раунді					
	1	*2*	*3*	*4*	*5*	*6*
1	$k_1^{(1)}$	$k_2^{(1)}$	$k_3^{(1)}$	$k_4^{(1)}$	$k_5^{(1)}$	$k_6^{(1)}$
2	$k_1^{(2)}$	$k_2^{(2)}$	$k_3^{(2)}$	$k_4^{(2)}$	$k_5^{(2)}$	$k_6^{(2)}$
3	$k_1^{(3)}$	$k_2^{(3)}$	$k_3^{(3)}$	$k_4^{(3)}$	$k_5^{(3)}$	$k_6^{(3)}$
4	$k_1^{(4)}$	$k_2^{(4)}$	$k_3^{(4)}$	$k_4^{(4)}$	$k_5^{(4)}$	$k_6^{(4)}$
5	$k_1^{(5)}$	$k_2^{(5)}$	$k_3^{(5)}$	$k_4^{(5)}$	$k_5^{(5)}$	$k_6^{(5)}$
6	$k_1^{(6)}$	$k_2^{(6)}$	$k_3^{(6)}$	$k_4^{(6)}$	$k_5^{(6)}$	$k_6^{(6)}$
7	$k_1^{(7)}$	$k_2^{(7)}$	$k_3^{(7)}$	$k_4^{(7)}$	$k_5^{(7)}$	$k_6^{(7)}$
8	$k_1^{(8)}$	$k_2^{(8)}$	$k_3^{(8)}$	$k_4^{(8)}$	$k_5^{(8)}$	$k_6^{(8)}$
Кінцеве перетворення (*9* раунд)	$k_1^{(9)}$	$k_2^{(9)}$	$k_3^{(9)}$	$k_4^{(9)}$		

Приклад 7.1. Нехай ключ шифрування даних *K* дорівнює:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Необхідно визначити значення раундових ключів зашифрування даних.

Розв'язання

Визначимо ключі розширеного ключа K_P шляхом розбиття ключа шифрування даних K на вісім частин і циклічного зсуву бітів ключа шифрування даних K. Дані зведемо в табл. 7.3.

Таблиця 7.2

Значення перших ключів для кожного раунду зашифрування *IDEA*

Номер раунду	Ключі розширеного ключа K_P	Ключ раунду	Біти ключа шифрування K
1	k_1	$k_1^{(1)}$	*K[1...16]*
2	k_7	$k_1^{(2)}$	*K[97...112]*
3	k_{13}	$k_1^{(3)}$	*K[90...105]*
4	k_{19}	$k_1^{(4)}$	*K[83...98]*
5	k_{25}	$k_1^{(5)}$	*K[77...91]*
6	k_{31}	$k_1^{(6)}$	*K[44...59]*
7	k_{37}	$k_1^{(7)}$	*K[37...52]*
8	k_{43}	$k_1^{(8)}$	*K[30...45]*
9	k_{49}	$k_1^{(9)}$	*K[23...38]*

Таблиця 7.3

Приклад ключів для кожного раунду шифрування *IDEA*

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
1	*0001*	*0002*	*0003*	*0004*	*0005*	*0006*
2	*0007*	*0008*	*0400*	*0600*	*0800*	*0a00*
3	*0c00*	*0e00*	*1000*	*0200*	*0010*	*0014*
4	*0018*	*001c*	*0020*	*0004*	*0008*	*000c*
5	*2800*	*3000*	*3800*	*4000*	*0800*	*1000*
6	*1800*	*2000*	*0070*	*0080*	*0010*	*0020*
7	*0030*	*0040*	*0050*	*0060*	*0000*	*2000*
8	*4000*	*6000*	*8000*	*a000*	*c000*	*e001*
9	*0080*	*00c0*	*0100*	*0140*	–	–

7.4.2. Генерація раундових ключів для розшифрування даних

Метод обчислення, що використовується для розшифрування даних по суті такий самий, як і при його зашифруванні. Єдина відмінність полягає в тому, що для розшифрування використовуються інші раундові ключі. У процесі розшифрування раундові ключі повинні використовуватися у зворотному порядку у відношенні до процесу зашифрования.

Для формування раундових ключів розшифрування спочатку формуються раундові ключі зашифрування.

Процес створення раундових ключів розшифрування визначається таким чином: перший і четвертий ключі *i*-го раунду розшифрування виходять із першого й четвертого ключа (*10*-*i*)-го раунду зашифрування мультиплікативної інверсії. Для *1*-го і *9*-го раундів другий і третій ключі розшифрування виходять із другого й третього ключів *9*-го і *1*-го раундів зашифрування адитивної інверсії. Для раундів з *2*-го по *8*-й другий і третій ключі розшифрування виходять із третього та другого ключів з *8*-го по *2*-й раунди зашифрування адитивної інверсії. Останні два ключі *i*-го раунду розшифрування дорівнюють останнім двом ключам (*9*-*i*)-го раунду зашифрування.

Мультиплікативну інверсію раундового ключа k позначимо як $k^{-1} = 1/k$ і тоді

$$\frac{1}{k} \cdot k \bmod (2^{16} + 1) = 1. \qquad (7.1)$$

Оскільки $2^{16}+1$ — просте число, то кожне ціле, що не дорівнює нулю, k має унікальну мультиплікативну інверсію за модулем $2^{16}+1$.

Адитивну інверсію раундового ключа k позначимо як $-k$ і тоді

$$(-k + k) \bmod (2^{16} + 1) = 0. \qquad (7.2)$$

Для реалізації алгоритму *IDEA* прийняли припущення, що нульовий субблок дорівнює $2^{16} = -1$, при цьому мультиплікативна обернена величина від 0 дорівнює 0 [22, 28]. Обчислення значень мультиплікативних обернених величин вимагає деяких витрат, але це доводиться робити тільки один раз для кожного ключа розшифрування.

У зв'язку з вказаним ключі розшифрування для різних раундів надано в табл. 7.4.

Таблиця 7.4

Ключі для кожного раунду розшифрування *IDEA*

Номер раунду	Номер ключа в раунді					
	1	*2*	*3*	*4*	*5*	*6*
1	$1/k_1^{(9)}$	$-k_2^{(9)}$	$-k_3^{(9)}$	$1/k_4^{(9)}$	$k_5^{(8)}$	$k_6^{(8)}$
2	$1/k_1^{(8)}$	$-k_3^{(8)}$	$-k_2^{(8)}$	$1/k_4^{(8)}$	$k_5^{(7)}$	$k_6^{(7)}$
3	$1/k_1^{(7)}$	$-k_3^{(7)}$	$-k_2^{(7)}$	$1/k_4^{(7)}$	$k_5^{(6)}$	$k_6^{(6)}$
4	$1/k_1^{(6)}$	$-k_3^{(6)}$	$-k_2^{(6)}$	$1/k_4^{(6)}$	$k_5^{(5)}$	$k_6^{(5)}$
5	$1/k_1^{(5)}$	$-k_3^{(5)}$	$-k_2^{(5)}$	$1/k_4^{(5)}$	$k_5^{(4)}$	$k_6^{(4)}$
6	$1/k_1^{(4)}$	$-k_3^{(4)}$	$-k_2^{(4)}$	$1/k_4^{(4)}$	$k_5^{(3)}$	$k_6^{(3)}$
7	$1/k_1^{(3)}$	$-k_3^{(3)}$	$-k_2^{(3)}$	$1/k_4^{(3)}$	$k_5^{(2)}$	$k_6^{(2)}$
8	$1/k_1^{(2)}$	$-k_3^{(2)}$	$-k_2^{(2)}$	$1/k_4^{(2)}$	$k_5^{(1)}$	$k_6^{(1)}$
9	$1/k_1^{(1)}$	$-k_2^{(1)}$	$-k_3^{(1)}$	$1/k_4^{(1)}$		

Приклад 7.2. Нехай ключ шифрування даних K дорівнює:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Необхідно визначити значення раундових ключів розшифрування даних.

Розв'язання

Скористаємося певними значеннями раундових ключів зашифрування даних із прикладу 7.1, наведених у табл. 7.3. Визначимо значення раундових ключів розшифрування даних за допомогою табл. 7.4 і виразів (7.1) і (7.2). Набуті значення раундових ключів розшифрування даних зведемо в табл. 7.5.

Таблиця 7.5

Приклад ключів для кожного раунду шифрування *IDEA*

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
1	*fe01*	*ff40*	*ff00*	*659a*	*c000*	*e001*
2	*fffd*	*8000*	*a000*	*cccc*	*0000*	*2000*

Закінчення табл. 7.5

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
3	*a556*	*ffb0*	*ffc0*	*52ab*	*0010*	*0020*
4	*554b*	*ff90*	*e000*	*fe01*	*0800*	*1000*
5	*332d*	*c800*	*d000*	*fffd*	*0008*	*000c*
6	*4aab*	*ffe0*	*ffe4*	*c001*	*0010*	*0014*
7	*aa96*	*f000*	*f200*	*ff81*	*0800*	*0a00*
8	*4925*	*fc00*	*fff8*	*552b*	*0005*	*0006*
9	*0001*	*fffe*	*fffd*	*c001*	–	–

7.5. ПРОЦЕС ШИФРУВАННЯ ДАНИХ

7.5.1. Зашифрування даних

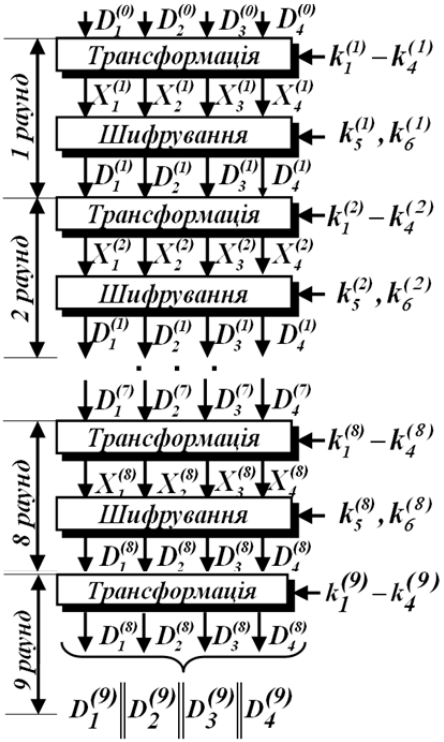


Рис. 7.6. Структурна схема алгоритму шифрування даних в *IDEA*

IDEA симетричний блоковий шифр і як вже було зазначено, процес зашифрування даних аналогічний процесу їх розшифрування. Структурну схему алгоритму зашифрування (розшифрування) даних в *IDEA* надано на рис. 7.1–7.4.

Процес зашифрування (розшифрування) даних являє собою вісім однакових раундів, а дев'ятий раунд — кінцеве перетворення.

Розглянемо загальну схему зашифрування (розшифрування) *IDEA* (рис. 7.6).

Як і в будь-якому алгоритмі шифрування, тут існує два входи: початковий блок даних і раундові ключі.

Трансформація є рядом операцій, показаних на рис. 7.7, а шифрування є рядом операцій, показаних на рис. 7.8.

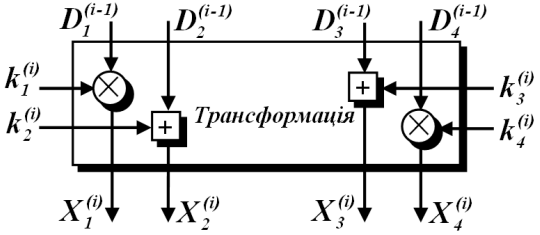


Рис. 7.7. Структура процесу трансформації даних
у раунді шифру *IDEA*

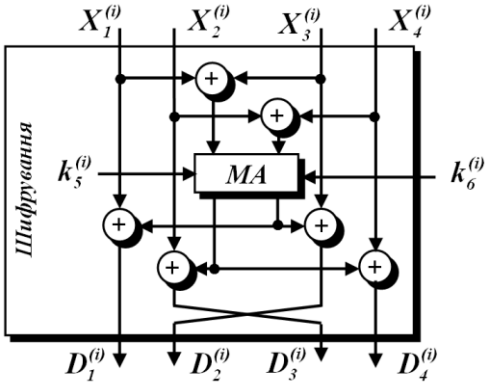


Рис. 7.8. Структура процесу шифрування даних
у раунді шифру *IDEA*

64-бітовий блок даних поділяється на чотири *16*-бітові підблоки (див. рис. 7.3):

$$D_1^{(0)},\ D_2^{(0)},\ D_3^{(0)},\ D_4^{(0)}.$$

Ці чотири підблоки стають входом до першого раунду алгоритму шифрування. Загалом виконується вісім раундів. Між раундами другий і третій підблоки міняються місцями (див. рис. 7.3).

У кожному раунді (крім дев'ятого) існує така послідовність операцій (рис. 7.8):

1. Множення підблока $D_1^{(i-1)}$ і першого раундового ключа $k_1^{(i)}$ за модулем $2^{16} + 1$ (*65537*):

$$X_1^{(i)} = D_1^{(i-1)} \otimes k_1^{(i)}.$$

2. Додавання підблока $D_2^{(i-1)}$ і другого раундового ключа $k_2^{(i)}$ за модулем 2^{16} (*65536*):

$$X_2^{(i)} = D_2^{(i-1)} \boxplus k_2^{(i)}.$$

3. Додавання підблока $D_3^{(i-1)}$ і третього раундового ключа $k_3^{(i)}$ за модулем 2^{16} (*65536*):

$$X_3^{(i)} = D_3^{(i-1)} \boxplus k_3^{(i)}.$$

4. Множення підблока $D_4^{(i-1)}$ і четвертого раундового ключа $k_4^{(i)}$ за модулем $2^{16} + 1$ (*65537*):

$$X_4^{(i)} = D_4^{(i-1)} \otimes k_4^{(i)}.$$

5. Додавання результатів кроків (1) і (3) за модулем *2*:

$$A^{(i)} = X_1^{(i)} \oplus X_3^{(i)}.$$

6. Додавання результатів кроків (2) і (4) за модулем *2*:

$$B^{(i)} = X_2^{(i)} \oplus X_4^{(i)}.$$

7. Множення результату кроку (5) і п'ятого раундового ключа $k_5^{(i)}$ за модулем $2^{16} + 1$ (*65537*):

$$C^{(i)} = A^{(i)} \otimes k_5^{(i)}.$$

8. Додавання результатів кроків (6) і (7):

$$E^{(i)} = B^{(i)} \boxplus C^{(i)}.$$

9. Множення результату кроку (8) з шостим раундовим ключем $k_6^{(i)}$ за модулем $2^{16} + 1$ (*65537*):

$$F^{(i)} = E^{(i)} \otimes k_6^{(i)}.$$

10. Додавання результатів кроків (7) і (9) за модулем 2^{16} (*65536*):

$$G^{(i)} = C^{(i)} \boxplus F^{(i)}.$$

11. Додавання результатів кроків (1) і (9) за модулем *2*:

$$D_1^{(i)} = X_1^{(i)} \oplus F^{(i)}.$$

12. Додавання результатів кроків (3) і (9) за модулем *2*:

$$D_2^{(i)} = X_3^{(i)} \oplus F^{(i)}.$$

13. Додавання результатів кроків (2) і (10) за модулем *2*:

$$D_3^{(i)} = X_2^{(i)} \oplus G^{(i)}.$$

14. Додавання результатів кроків (4) і (10) за модулем *2*:

$$D_4^{(i)} = X_4^{(i)} \oplus G^{(i)}.$$

Виходом раунду є чотири підблоки, які виходять як результати виконання кроків 11–14, і внаслідок формується вхід для наступного раунду.

Після восьмого раунду здійснюється кінцеве перетворення (див. рис. 7.7):

1. Множення підблока $D_1^{(8)}$ і першого раундового ключа $k_1^{(9)}$ за модулем $2^{16} + 1$ (*65537*):

$$D_1^{(9)} = D_1^{(8)} \otimes k_1^{(9)}. \qquad (7.3)$$

2. Додавання підблока $D_3^{(8)}$ і другого раундового ключа $k_2^{(9)}$ за модулем 2^{16} (*65536*):

$$D_2^{(9)} = D_3^{(8)} \boxplus k_2^{(9)}. \qquad (7.4)$$

3. Додавання підблока $D_2^{(8)}$ і третього раундового ключа $k_3^{(9)}$ за модулем 2^{16} (*65536*):

$$D_3^{(9)} = D_2^{(8)} \boxplus k_3^{(9)}. \qquad (7.5)$$

4. Множення підблока $D_4^{(8)}$ і четвертого раундового ключа $k_4^{(9)}$ за модулем $2^{16} + 1$ (*65537*):

$$D_4^{(9)} = D_4^{(8)} \otimes k_4^{(9)}. \qquad (7.6)$$

Нарешті, ці результуючі підблоки, отримані після кінцевого перетворення, знову об'єднують для отримання зашифрованого блока даних

$$C = D_1^{(9)} \| D_2^{(9)} \| D_3^{(9)} \| D_4^{(9)}.$$

Потім беруть наступний *64*-бітовий блок початкових даних і алгоритм шифрування повторюється. Так триває до тих пір, поки не будуть зашифровані усі *64*-бітові блоки початкових даних.

Приклад 7.3. Нехай ключ шифрування даних *K* дорівнює:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Початкові дані для зашифрування

$$M = 0000\ 0001\ 0002\ 0003_{16}.$$

Необхідно отримати результат зашифрування вказаних даних.

Розв'язання

Процес зашифрування даних алгоритмом *IDEA* пояснюється за допомогою табл. 7.6.

Таблиця 7.6

Раундові ключі й підблоки для кожного раунду зашифрування

Раунд	*Раундові ключі*						*Значення підблоків даних*			
	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$	$D_1^{(0)}$	$D_2^{(0)}$	$D_3^{(0)}$	$D_4^{(0)}$
							0000	*0001*	*0002*	*0003*
1	*0001*	*0002*	*0003*	*0004*	*0005*	*0006*	*00f0*	*00f5*	*010a*	*0105*
2	*0007*	*0008*	*0400*	*0600*	*0800*	*0a00*	*222f*	*21b5*	*f45e*	*e959*
3	*0c00*	*0e00*	*1000*	*0200*	*0010*	*0014*	*0f86*	*39be*	*8ee8*	*1173*
4	*0018*	*001c*	*0020*	*0004*	*0008*	*000c*	*57df*	*ac58*	*c65b*	*ba4d*
5	*2800*	*3000*	*3800*	*4000*	*0800*	*1000*	*8e81*	*ba9c*	*f77f*	*3a4a*
6	*1800*	*2000*	*0070*	*0080*	*0010*	*0020*	*6942*	*9409*	*e21b*	*1c64*
7	*0030*	*0040*	*0050*	*0060*	*0000*	*2000*	*99d0*	*c7f6*	*5331*	*620e*
8	*4000*	*6000*	*8000*	*a000*	*c000*	*e001*	*0a24*	*0098*	*ec6b*	*4925*
9	*0080*	*00c0*	*0100*	*0140*	–	–	*11fb*	*ed2b*	*0198*	*6de5*

Отже, результат шифрування початкових даних буде дорівнювати:

$$C = 11fb\ ed2b\ 0198\ 6de5_{16}.$$

7.5.2. Розшифрування даних

IDEA — симетричний блоковий шифр і, як вже було зазначено, процес розшифрування даних аналогічний процесу їх зашифрування, тому структурна схема алгоритму розшифрування даних в *IDEA* відповідає схемі алгоритму зашифрування (див. рис. 7.6).

Розшифрування полягає у використанні зашифрованих даних як входу в ту саму структуру *IDEA*, але з іншим набором раундових ключів ніж при зашифруванні.

Для доведення того, що алгоритм розшифрування з відповідними раундовими ключами має коректний результат, розглянемо одночасно процеси зашифрування й розшифрування. Кожний із восьми раундів розбитий на дві стадії перетворення, перша з яких називається трансформацією, а друга — шифруванням, як показано на рис. 7.6, 7.7 і 7.8.

Розглянемо перетворення, що виконуються в прямокутниках на рис. 7.7. Під час зашифрування підтримуються такі співвідношення на виході трансформації (7.3)...(7.6).

У ході розшифрування даних, необхідно на вхід схеми, наданої на рис. 7.6, подати такі дані:

$$D_1^{(0)} = D_1^{(9)},\ D_2^{(0)} = D_2^{(9)},\ D_3^{(0)} = D_3^{(9)},\ D_4^{(0)} = D_4^{(9)}.$$

Перша стадія першого раунду процесу розшифрування схемою, наданою на рис. 7.6, буде підтримувати такі співвідношення:

$$\begin{aligned} X_1^{(1)} &= D_1^{(9)} \otimes 1/k_1^{(9)}; \quad X_2^{(1)} = D_2^{(9)} \boxplus 1/k_2^{(9)}; \\ X_3^{(1)} &= D_3^{(9)} \boxplus 1/k_3^{(9)}; \quad X_4^{(1)} = D_4^{(9)} \otimes 1/k_4^{(9)}. \end{aligned} \tag{7.7}$$

Підставляючи відповідні значення (7.3), (7.4), (7.5) і (7.6) у (7.7), отримаємо:

$$\begin{aligned} X_1^{(1)} &= D_1^{(8)} \otimes 1/k_1^{(9)} \otimes k_1^{(9)} = D_1^{(8)}; \\ X_2^{(1)} &= D_3^{(8)} \boxplus (-k_2^{(9)}) \boxplus k_2^{(9)} = D_3^{(8)}; \\ X_3^{(1)} &= D_2^{(8)} \boxplus (-k_3^{(9)}) \boxplus k_3^{(9)} = D_2^{(8)}; \\ X_4^{(1)} &= D_4^{(8)} \otimes 1/k_4^{(9)} \otimes k_4^{(9)} = D_4^{(8)}. \end{aligned} \tag{7.8}$$

Отже, вихід першої стадії процесу розшифрування еквівалентний входу останньої стадії процесу зашифрування за винятком чергування другого й третього підблоків.

Значення підблоків на виході восьмого раунду згідно зі схемою на рис. 7.3 і 7.6 будуть визначатися такими співвідношеннями:

$$\begin{aligned} D_1^{(8)} &= X_1^{(8)} \oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_2^{(8)} &= X_3^{(8)} \oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_3^{(8)} &= X_2^{(8)} \oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_4^{(8)} &= X_4^{(8)} \oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}), \end{aligned} \tag{7.9}$$

де $MA_R(X,Y)$ — правий вихід MA структури з входами X і Y, і $MA_L(X,Y)$ — лівий вихід MA структури з входами X і Y.

Значення підблока $D_1^{(1)}$ на виході першого раунду згідно зі схемою на рис. 7.3 і 7.6 буде дорівнювати:

$$\begin{aligned} D_1^{(1)} &= X_1^{(1)} \oplus MA_R(X_1^{(1)} \oplus X_2^{(1)}, X_3^{(1)} \oplus X_4^{(1)}) = \\ &= D_1^{(8)} \oplus MA_R(D_1^{(8)} \oplus D_2^{(8)}, D_3^{(8)} \oplus D_4^{(8)}). \end{aligned} \tag{7.10}$$

Підставляючи значення (7.9) в (7.10) отримаємо

$$\begin{aligned} D_1^{(1)} = X_1^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus MA_R(X_1^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus X_3^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}), \\ X_2^{(8)} &\oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus X_4^{(8)} &\oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)})). \end{aligned} \tag{7.11}$$

Роблячи перетворення в (7.11), остаточно будемо мати

$$D_1^{(1)} = X_1^{(8)}.$$

Аналогічно можна отримати й інші значення підблоків на виході першого раунду:

$$D_2^{(1)} = X_3^{(8)}, \quad D_3^{(1)} = X_2^{(8)}, \quad D_4^{(1)} = X_4^{(8)}.$$

Отже, вихід другої стадії процесу розшифрування еквівалентний входу передостанньої стадії процесу зашифрування за винятком чергування другого й третього підблоків.

Аналогічно можна показати, що

$$D_1^{(8)} = X_1^{(1)}, \quad D_2^{(8)} = X_3^{(1)}, \quad D_3^{(8)} = X_2^{(1)}, \quad D_4^{(8)} = X_4^{(1)}.$$

Нарешті, оскільки вихід трансформації процесу розшифрування еквівалентний першій стадії процесу зашифрування за винятком чергування другого й третього підблоків, виходить, що вихід усього процесу розшифрування еквівалентний входу процесу зашифрування. Іншими словами, у двох суміжних (непарному й парному) раундах двічі відбувається перестановка внутрішніх підблоків даних. Отже, на виході восьмого раунду значення внутрішніх підблоків даних будуть мати природний порядок. У дев'ятому раунді порядок дотримання внутрішніх підблоків даних буде змінений двічі й тому на виході раунду порядок дотримання також буде природним.

Отже, доведено, що алгоритм розшифрування з відповідними раундовими ключами має коректний результат.

Приклад 7.4. Нехай ключ шифрування даних K дорівнює:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Зашифровані дані дорівнюють:

$$C = 11fb\ ed2b\ 0198\ 6de5_{16}.$$

Необхідно отримати результат розшифрування.

Розв'язання

Процес розшифрування даних алгоритмом *IDEA* пояснюється за допомогою табл. 7.7.

Отже, результат розшифрування зашифрованих даних буде дорівнювати:

$$M = 0000\ 0001\ 0002\ 0003_{16}.$$

З отриманого результату й результату прикладу 7.3 витікає, що розшифрування зроблено коректно.

Таблиця 7.7

Раундові ключі й підблоки для кожного раунду розшифрування

Раунд	Раундові ключі						Значення підблоків даних			
	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$	$D_1^{(0)}$	$D_2^{(0)}$	$D_3^{(0)}$	$D_4^{(0)}$
							11fb	*ed2b*	*0198*	*6de5*
1	*fe01*	*ff40*	*ff00*	*659a*	*c000*	*e001*	*d98d*	*d331*	*27f6*	*82b8*
2	*fffd*	*8000*	*a000*	*cccc*	*0000*	*2000*	*bc4d*	*e26b*	*9449*	*a576*
3	*a556*	*ffb0*	*ffc0*	*52ab*	*0010*	*0020*	*0aa4*	*f7ef*	*da9c*	*24e3*
4	*554b*	*ff90*	*e000*	*fe01*	*0800*	*1000*	*ca46*	*fe5b*	*dc58*	*116d*
5	*332d*	*c800*	*d000*	*fffd*	*0008*	*000c*	*748f*	*8f08*	*39da*	*45cc*
6	*4aab*	*ffe0*	*ffe4*	*c001*	*0010*	*0014*	*3266*	*045e*	*2fb5*	*b02e*
7	*aa96*	*f000*	*f200*	*ff81*	*0800*	*0a00*	*0690*	*050a*	*00fd*	*1dfa*
8	*4925*	*fc00*	*fff8*	*552b*	*0005*	*0006*	*0000*	*0005*	*0003*	*000c*
9	*0001*	*fffe*	*fffd*	*c001*	-	-	*0000*	*0001*	*0002*	*0003*

7.6. БЕЗПЕКА ШИФРУ

7.6.1. Аналіз шифру *IDEA*

Алгоритм *IDEA* з'явився внаслідок незначних модифікацій алгоритму *PES*. На рис. 7.9 наведено структуру восьми раундів зашифрування даних алгоритмом *PES*, а на рис. 7.10 — кінцевого перетворення.

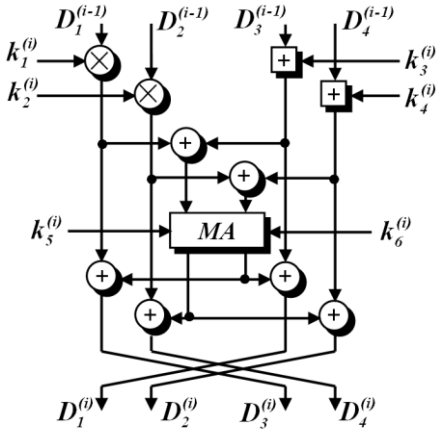


Рис. 7.9. Один раунд перетворення даних у *PES*

Із зіставлення структур на рис. 7.3 і 7.4, з рис. 7.9 і 7.10 видно, що змін не так вже й багато.

В алгоритмі *IDEA* в порівнянні з алгоритмом *PES* зроблено такі зміни (в усіх раундах, включаючи дев'ятий):

- множення підблока $D_2^{(i)}$ із другим раундовим ключем $k_2^{(i)}$ за модулем 2^{16} замінено їх додаванням за модулем $2^{16} + 1$;

- додавання підблока $D_4^{(i)}$ із четвертим раундовим ключем

$k_4^{(i)}$ за модулем $2^{16} + 1$ замінено їх множенням за модулем 2^{17}.:

Крім того, в алгоритмі *IDEA* в порівнянні з алгоритмом *PES* змінено зсув підблоків даних у кінці раунду (тільки в перших восьми раундах).

Також, в алгоритмі *IDEA* в порівнянні з алгоритмом *PES* змінено зсув підблоків даних на початку кінцевого перетворення.

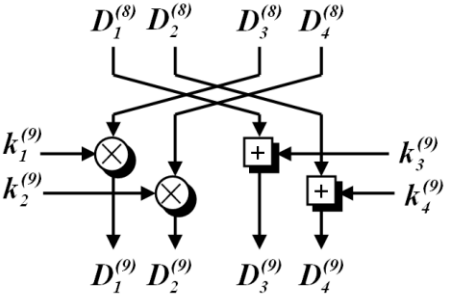


Рис. 7.10. Кінцеве перетворення (раунд 9) у *PES*

Один із найбільш відомих у світі криптологів *Б. Шнайєр* у своїй книзі "Прикладна криптографія" зазначив: "…дивно, як такі незначні зміни можуть привести до таких великих відмінностей" [45].

У тій самій книзі, що вийшла 1996 р., Б. Шнайєр відізвався про *IDEA* так: "Мені здається, це найкращий і надійніший блоковий алгоритм, опублікований до теперішнього часу".

В алгоритмі *IDEA* використовуються *64*-бітові блоки. Довжина блока повинна бути достатньою, щоб приховати статистичні характеристики початкового повідомлення. Але зі збільшенням розміру блока експоненціально зростає складність реалізації криптографічного алгоритму.

7.6.2. Уразливість ключа шифру

Розмір ключа в шифрі IDEA

В алгоритмі *IDEA* використовується *128*-бітовий ключ. Довжина ключа повинна бути досить великою, щоб запобігти можливості перебору ключа. Для розкриття *128*-бітового ключа повним перебором ключів за умови, що відомі відкриті й відповідні їм зашифровані дані, знадобиться 2^{128} (близько 10^{38}) шифрувань. За такої довжини ключа *IDEA* вважається досить безпечним. Висока криптографічна стійкість *IDEA* забезпечується також такими характеристиками:

- заплутування – шифрування, яке залежить від ключа складно й заплутано;

- розсіяння — кожний біт початкових даних впливає на кожний біт зашифрованих даних.

Сюецзя Лай і Джеймс Мессі ретельно проаналізували *IDEA* з метою з'ясування його криптографічної стійкості до диференціального криптографічного аналізу. Для цього вони ввели поняття марківського шифру й продемонстрували, що стійкість до диференціального криптографічного аналізу можна промоделювати й оцінити кількісно. Лінійної або алгебраїчної уразливості в *IDEA* виявлено не було. Спроба розкрити за допомогою криптографічного аналізу з пов'язаними ключами спробував Біхам, також не дала результатів [19, 25, 38, 42].

Існують вдалі атаки, застосовні до *IDEA* з меншою кількістю раундів (повний *IDEA* має *8* повних і один неповний раунд). Вдалою вважається атака, якщо розкриття шифру з її допомогою вимагає меншої кількості операцій, ніж за повного перебору ключів. Метод розкриття *Віллі Мейєра* виявився вдалим за розкриття повним перебором ключів тільки для *IDEA* з двома раундами. Методом "зустріч посередині" був розкритий *IDEA* із 4,5 раундами. Для цього потрібно знання усіх 2^{64} блоків із словника кодів і складність аналізу складає 2^{112} операції. Краща атака на 2007 р. застосована до всіх ключів і може зламати *IDEA 6* раундами [17, 28].

Уразливі ключі в шифрі IDEA

Існують великі класи уразливих ключів *IDEA*. Уразливі вони в тому сенсі, що існують процедури, які дозволяють визначити, чи належить ключ до цього класу, а потім і сам ключ. На сьогодні відомі такі:

1. $2^{23} + 2^{35} + 2^{51}$ уразливих до диференціального криптографічного аналізу ключів. Приналежність до класу 2^{51} можна вирахувати за 2^{12} операції за допомогою підібраних відкритих даних. Автори цієї атаки запропонували модифікацію алгоритму *IDEA*. Ця модифікація полягає в заміні раундових ключів $k_j^{(i)}$ на ті, що відповідають

$$k_j^{'(i)} = a \oplus k_j^{(i)},$$

де *i* — номер раунду шифрування; *j* — номер раундового ключа.

Точне значення *a* не критичне. Наприклад, за $a = 0dae_{16}$ (у шістнадцятковій системі числення) ці уразливі ключі виключаються.

2. 2^{63} уразливих до лінійного криптографічного аналізу ключів. Приналежність до цього класу з'ясовується за допомогою тесту на пов'язаних ключах.

3. $2^{53} + 2^{56} + 2^{64}$ уразливих ключів було знайдено з використанням *методу бумеранга* (англ. *boomerang attack*), запропонованого Девідом Вагнером. Тест на приналежність до цього класу виконується за 2^{16} операцій і потребує 2^{16} комірок пам'яті.

Існування таких великих класів слабких ключів не впливає на практичну криптографічну стійкість алгоритму *IDEA*, оскільки повне число усіх можливих ключів дорівнює 2^{128}.

7.7. ЗАСТОСУВАННЯ АЛГОРИТМУ

Алгоритм *IDEA* є торговельною маркою й запатентований в Австрії, Франції, Німеччині, Італії, Нідерландах, Іспанії, Швеції, Швейцарії, Англії, США і в Японії. Сьогодні ліцензія належить компанії *MediaCrypt* і дозволяє вільно використовувати алгоритм з некомерційною метою. У травні 2005 р. *MediaCrypt* офіційно представила новий шифр *IDEA NXT* (первинна назва *FOX*), що повинен замінити *IDEA*. Типові сфери застосування *IDEA*:

- шифрування аудіо- і відеоданих для кабельного телебачення, відеоконференцій, дистанційного навчання та ін.;
- захист комерційної та фінансової інформації, що відбиває кон'юнктурні коливання;
- лінії зв'язку через модем, роутер або *ATM* (*Asynchronous Transfer Mode* — асинхронний метод перенесення) лінію, *GSM (Global System for Mobile Communications* — Глобальний цифровий стандарт мобільного стільникового зв'язку) технологію;
- смарт-карти;
- загальнодоступний пакет конфіденційної версії електронної пошти *PGP v2.0* і в *OpenPGP*.

Контрольні питання та завдання

1. Який розмір блока в *IDEA*? Який розмір ключа шифру в *IDEA*? Який розмір ключів раунду в *IDEA*?
2. Яка кількість раундів в *IDEA*?
3. Які прості операції використовуються в раундах *IDEA*?
4. Які прості операції використовуються на етапі трансформації даних у раундах *IDEA*?

5. Які прості операції використовуються на етапі шифрування даних у раундах *IDEA*?

6. Пояснити алгоритм формування раундових ключів при зашифруванні даних в *IDEA*.

7. Пояснити алгоритм формування раундових ключів при розшифруванні даних в *IDEA*.

8. Початковий (сеансовий) ключ алгоритму *IDEA* — послідовність довжиною *128* бітів, яка дорівнює $K = 3f424cdc105ca00d7\ b3dbe8c96a2978e_{16}$. Сформувати всі раундові ключі для зашифрування.

9. Визначити раундові ключі розшифрування $k_2^{(i)}$ ($i = 1...9$) алгоритму *IDEA*, якщо раундові ключі зашифрування дорівнюють: $k_2^{(1)} = 4cdc_{16}$, $k_2^{(2)} = 97be_{16}$, $k_2^{(3)} = 452f_{16}$, $k_2^{(4)} = 5a8a_{16}$, $k_2^{(5)} = 64b5_{16}$, $k_2^{(6)} = 6bd9_{16}$, $k_2^{(7)} = 00d7_{16}$, $k_2^{(8)} = 9401_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(1)} = 105c_{16}$, $k_3^{(2)} = b820_{16}$, $k_3^{(3)} = 1c7e_{16}$, $k_3^{(4)} = 5e38_{16}$, $k_3^{(5)} = 14bc_{16}$, $k_3^{(6)} = 6a29_{16}$, $k_3^{(7)} = b3db_{16}$, $k_3^{(8)} = af67_{16}$, $k_3^{(9)} = 035e_{16}$.

10. Визначити раундові ключі розшифрування $k_3^{(i)}$ ($i = 1...8$) алгоритму *IDEA*, якщо раундові ключі зашифрування дорівнюють $k_2^{(1)} = 4cdc_{16}$, $k_2^{(2)} = 97be_{16}$, $k_2^{(3)} = 452f_{16}$, $k_2^{(4)} = 5a8a_{16}$, $k_2^{(5)} = 64b5_{16}$, $k_2^{(6)} = 6bd9_{16}$, $k_2^{(7)} = 00d7_{16}$, $k_2^{(8)} = 9401_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(1)} = 105c_{16}$, $k_3^{(2)} = b820_{16}$, $k_3^{(3)} = 1c7e_{16}$, $k_3^{(4)} = 5e38_{16}$, $k_3^{(5)} = 14bc_{16}$, $k_3^{(6)} = 6a29_{16}$, $k_3^{(7)} = b3db_{16}$, $k_3^{(8)} = af67_{16}$, $k_3^{(9)} = 035e_{16}$.

11. Визначити раундові ключі розшифрування $k_5^{(i)}$ ($i = 1...9$) алгоритму *IDEA*, якщо раундові ключі зашифрування дорівнюють: $k_5^{(1)} = 7b3d_{16}$, $k_5^{(2)} = 1af6_{16}$, $k_5^{(3)} = 8035_{16}$, $k_5^{(4)} = 3370_{16}$, $k_5^{(5)} = 1266_{16}$, $k_5^{(6)} = f424_{16}$, $k_5^{(7)} = c7e8_{16}$, $k_5^{(8)} = 92d4_{16}$.

12. Визначити раундові ключі розшифрування $k_1^{(i)}$ ($i = 1...9$) алгоритму *IDEA*, якщо раундові ключі зашифрування дорівнюють: $k_1^{(1)} = 3f42_{16}$, $k_1^{(2)} = 96a2_{16}$, $k_1^{(3)} = 192d_{16}$, $k_1^{(4)} = fa32_{16}$, $k_1^{(5)} = edf4_{16}$, $k_1^{(6)} = e500_{16}$, $k_1^{(7)} = 05ca_{16}$, $k_1^{(8)} = 820b_{16}$, $k_1^{(9)} = 3704_{16}$.

13. Визначити результат етапу перестановки (трансформації) першого раунду алгоритмом *IDEA*, якщо вхідні дані дорівнюють: $M = 3d550f51d71ee0aa_{16}$. Раундові ключі зашифрування дорівнюють: $k_1^{(1)} = 3f42_{16}$, $k_2^{(1)} = 4cdc_{16}$, $k_3^{(1)} = 105c_{16}$, $k_4^{(1)} = a00d_{16}$.

14. Визначити результат пристрою *MA* етапу шифрування першого раунду алгоритмом *IDEA*, якщо вхідні дані дорівнюють: $f_1^{(1)} = 4cb9_{16}$, $f_2^{(1)} = 4000_{16}$. Раундові ключі зашифрування дорівнюють: $k_5^{(1)} = 7b3d_{16}$, $k_6^{(1)} = be8c_{16}$.

15. Визначити результат зашифрування даних алгоритмом *IDEA*, якщо результат *8*-го раунду дорівнює: $D_1^{(8)} = e2fb_{16}$, $D_2^{(8)} = cd9d_{16}$, $D_3^{(8)} = cb10_{16}$, $D_4^{(8)} = 9936_{16}$. Раундові ключі зашифрування дорівнюють: $k_1^{(9)} = 3704_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(9)} = 035e_{16}$, $k_4^{(9)} = cf6f_{16}$.

16. Визначити вхідні дані пристрою *MA* результат етапу шифрування першого раунду алгоритмом *IDEA*, якщо результати перестановки даного раунду дорівнюють: $X_1^{(1)} = abc3_{16}$, $X_2^{(1)} = 5c2d_{16}$, $X_3^{(1)} = e77a_{16}$, $X_4^{(1)} = 1c2d_{16}$.

Розділ 8

СТАНДАРТ СИМЕТРИЧНОГО АЛГОРИТМУ БЛОКОВОГО ШИФРУВАННЯ ДАНИХ ДСТУ ГОСТ 28147:2009

8.1. ІСТОРІЯ СТВОРЕННЯ СТАНДАРТУ

Важливим завданням у забезпеченні гарантованої безпеки інформації в інформаційних системах є розробка й використання стандартних алгоритмів шифрування даних. Першим серед подібних стандартів, як зазначалося в розділі 5, став американський стандарт шифрування даних *DES*, що являє собою послідовне використанням замін і перестановок. На сьогодні все частіше говорять про невиправдану складність і невисоку криптографічну стійкість *DES*. На практиці доводиться використовувати його модифікації.

Ефективнішим є вітчизняний 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. Цей стандарт формувався з урахуванням світового досвіду, і зокрема, було взято до уваги недоліки й нереалізовані можливості алгоритму *DES*, тому використання стандарту ДСТУ ГОСТ 28147:2009 найвпевненіше. Алгоритм досить складний і нижче буде описано переважно його концепцію.

Цей стандарт закріплений ГОСТ 28147-89, прийнятим, як виявляється з його позначення, ще 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 ФАУЗІ (Федеральне агентство урядового зв'язку та інформації при Президентові РФ). 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 за сімома печатками" [2].

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 [34]. 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 початкових 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 даними довжиною *64* біти. Зашифрування й розшифрування даних здійснюється з використанням або *32* або *16* раундів (циклів). Ключ шифру *256* бітів, з якого формуються вісім підключів довжиною по *32* біти (із цих восьми підключів формуються *32* (*16*) раундових (циклових) підключів).

Стандарт передбачає чотири режими роботи:

- шифрування (розшифрування) даних у режимі простої заміни;
- шифрування (розшифрування) даних у режимі гамування;
- шифрування (розшифрування) даних в режимі гамування зі зворотним зв'язком;
- шифрування з метою вироблення імітовставки.

8.2. ПРИНЦИПИ ПОБУДОВИ АЛГОРИТМУ

На різних кроках стандарту ДСТУ ГОСТ 28147:2009 дані, якими вони оперують, інтерпретуються й використовуються по-різному. У деяких випадках елементи даних обробляються як масиви незалежних бітів, в інших випадках — як ціле число без знаку, у третіх — як складний елемент, який має структуру, що складається з декількох простіших елементів. Тому, щоб уникнути плутанини, слід домовитися про використовувані позначення.

Елементи даних позначаються великими латинськими літерами з похилим зображенням (наприклад, *X*). Через $|X|$ позначається розмір елемента даних *X* у бітах. Отже, якщо інтерпретувати елемент

даних X як ціле позитивне число, можна записати таку нерівність: $0 \le X < 2^{|X|}$.

Якщо елемент даних складається з декількох елементів меншого розміру, то цей факт позначається так:

$$X = (X_0, X_1, ..., X_{n-1}) = X_0 \| X_1 \| ... \| X_{n-1}.$$

Процедура об'єднання декількох елементів даних в один називається *конкатенацією* даних і позначається символом "$\|$". Природно, що для розмірів елементів даних повинне виконуватися таке співвідношення:

$$|X| = |X_0| + |X_1| + ... + |X_{n-1}|.$$

За завдання складних елементів даних і операції конкатенації складові елементи даних перераховуються в порядку зростання. Іншими словами, якщо інтерпретувати складовий елемент і всі елементи даних, що входять в нього, як цілі числа без знаку, то можна записати рівність:

$$X = (X_0, X_1, ..., X_{n-1}) = X_0 \| X_1 \| ... \| X_{n-1} =$$

$$= X_0 + 2^{|X_0|}(X_1 + 2^{|X_1|} \cdot (X_2 + 2^{|X_2|} \cdot (...2^{|X_{n-2}|} \cdot X_{n-1})...)).$$

В алгоритмі елемент даних може інтерпретуватися як масив окремих бітів. У цьому випадку біти позначаються тією самою літерою, що й масив, але в рядковому варіанті, як показано на такому прикладі:

$$X = (x_0, x_1, ..., x_{n-1}) = x_0 \cdot 2^0 + x_1 \cdot 2^1 + ... + x_{n-1} \cdot 2^{n-1}.$$

Отже, для госту прийнято так звану *"little-endian"* нумерацію розрядів, тобто усередині багаторозрядних слів даних окремі двійкові розряди та їх групи з меншими номерами є менш значущими. Про це прямо сказано: "При складанні й циклічному зсуві двійкових векторів старшими розрядами вважаються розряди накопичувачів із великими номерами" [34]. Стандарт ДСТУ ГОСТ 28147:2009 пропонує починати заповнення даними регістрів-накопичувачів віртуального шифруючого пристрою з молодших, тобто менш значущих розрядів. Такий самий порядок нумерації прийнятий у мікропроцесорній архітектурі *Intel x86*, саме тому при програмній реалізації

шифру на цій архітектурі ніяких додаткових перестановок розрядів усередині слів даних не потрібно.

Якщо над елементами даних виконується деяка операція, що має логічний сенс, то передбачається, що ця операція виконується над відповідними бітами елементів. Іншими словами,

$$A \bullet B = (a_0 \bullet b_0, a_1 \bullet b_1, a_{n-1} \bullet b_{n-1}),$$

де $n = |A| = |B|$, а символом "$\bullet$" позначається довільна бінарна логічна операція; як правило йдеться про операцію *xor*, яка є також операцією підсумовування за модулем *2*:

$$a \oplus b = (a + b) \cdot mod\ 2.$$

Крім того, в алгоритмі використовуються операції складання двох *32*-розрядних цілих позитивних чисел за модулем 2^{32} (позначається $\boxplus$) і за модулем $2^{32}-1$ ($\boxplus\!\!\!\!\pm$).

Два цілих числа *a* і *b*, де $0 \le a < 2^{32}$, $0 \le b < 2^{32}$:

$$a = (a_{32}, a_{31}, ..., a_2, a_1) \quad \text{і} \quad b = (b_{32}, b_{31}, ..., b_2, b_1),$$

надані у двійковому виді, тобто

$$a = a_{32} \cdot 2^{31} + a_{31} \cdot 2^{30} + ... + a_2 \cdot 2^1 + a_1 \cdot 2^0;$$

$$b = b_{32} \cdot 2^{31} + b_{31} \cdot 2^{30} + ... + b_2 \cdot 2^1 + b_1 \cdot 2^0,$$

підсумовуються за модулем 2^{32} (операція $\boxplus$) за таким правилом:

$$a \boxplus b = a + b, \text{ якщо } a + b < 2^{32};$$

$$a \boxplus b = a + b - 2^{32}, \text{ якщо } a + b \ge 2^{32}.$$

Два цілих числа *a* і *b*, де $0 \le a \le 2^{32}-1$, $0 \le b \le 2^{32}-1$, представлені аналогічно, підсумовуються за модулем 2^{32} (операція $\boxplus\!\!\!\!\pm$) за таким правилом:

$$a \boxplus\!\!\!\!\pm b = a + b, \text{ якщо } a + b < 2^{32}-1;$$

$$a \boxplus\!\!\!\!\pm b = a + b - (2^{32}-1), \text{ якщо } a + b \ge 2^{32}-1.$$

8.3. ШИФРУВАННЯ ДАНИХ У РЕЖИМІ ПРОСТОЇ ЗАМІНИ

8.3.1. Структура криптографічної системи шифрування даних у режимі простої заміни

Для реалізації алгоритму шифрування ДСТУ ГОСТ 28147:2009 у режимі простої заміни (аналогічний режим *ECB*) використовується тільки частина блоків загальної криптографічної системи (рис. 8.1).

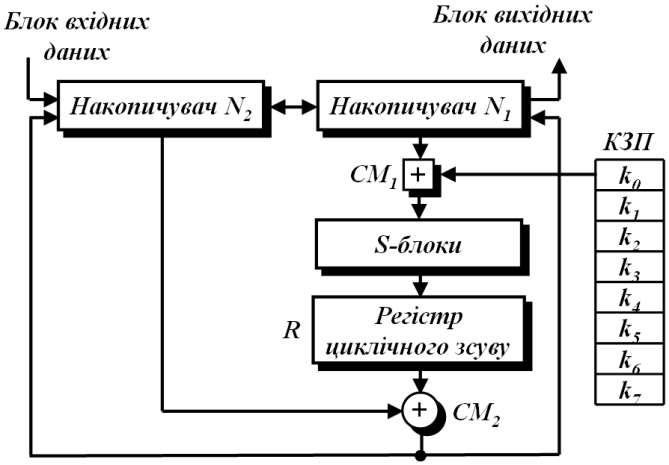


Рис. 8.1. Структура реалізації режиму простої заміни стандарту ДСТУ ГОСТ 28147:2009

Позначення на схемі:

- 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* бітів, 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 (бокс) підстановки (заміни), що складається з восьми вузлів заміни (*S*-блоків заміни) S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

8.3.2. Зашифрування даних у режимі простої заміни

Відкриті дані, що підлягають зашифруванню ДСТУ ГОСТ 28147:2009, розбивають на *64*-розрядні блоки. Процедура зашифрування *64*-розрядного блока T_O у режимі простої заміни включає *32* цикли (раунди). У ключовий запам'ятовуючий пристрій (КЗП) вводять *256* бітів ключа *K* у вигляді восьми *32*-розрядних підключів (чисел) k_i:

$$K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7).$$

Послідовність бітів блока

$$T_O = (a_1(0), a_2(0), \dots, a_{32}(0), b_1(0), b_2(0), \dots, b_{32}(0))$$

розбивають на дві послідовності по *32* біти (взяті у зворотному порядку):

$$b(0) = b_{32}(0), b_{31}(0), \dots, b_1(0) \text{ і } a(0) = a_{32}(0), a_{33}(0), \dots, a_1(0),$$

де $b(0)$ — ліві або старші, 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 зашифрування. Як результат початкове заповнення накопичувача N_1

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), \dots, a_1(0)),$$

початкове заповнення накопичувача N_2

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), \dots, b_1(0)).$$

Перший цикл процедури зашифрування *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 зашифрування; $b(1)$ — заповнення N_2 після першого циклу шифрування; $f(\bullet)$ — функція зашифрування.

Аргументом функції $f(\bullet)$ є сума за модулем 2^{32} числа $a(0)$ (початкового заповнення накопичувача N_1) і числа k_0 — підключа, що зчитується з накопичувача КЗП. Кожне із 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

$$a(0) \dot{\boxplus} k_0 = 0011\ 1000\ 1101\ 0111\ 0101\ 0000\ 0011\ 1110_2.$$

Функція $f(\bullet)$ включає дві операції над отриманою 32-розрядною сумою $a(0) \dot{\boxplus} k_0$.

Перша операція називається підстановкою (заміною) і виконується S-блоком підстановки. S-блок підстановки складається, як було вище сказано, складається з восьми вузлів заміни (S-блоків заміни) з пам'яттю 64 біти кожний. Надходячи з CM_1, на S-блок підстановки 32-розрядний вектор розбивають на вісім 4-розрядних векторів, що йдуть послідовно, кожний з яких перетвориться в чотирирозрядний вектор відповідним вузлом заміни. Кожний вузол заміни можна подати у вигляді таблиці-перестановки шістнадцяти чотирирозрядних двійкових чисел у діапазоні: $0000, 0001, \ldots, 1111_2$. Вхідний вектор вказує адресу рядка в таблиці, а число в цьому рядку є вихідним вектором. Потім чотирирозрядні вихідні вектори послідовно об'єднують у 32-розрядний вектор. Вузли заміни (таблиці-перестановки) є ключовими елементами, які є спільними для мережі ЕОМ і рідко змінюються. Ці вузли заміни повинні зберігатися у таємниці.

У табл. 8.1 наведено приклад замін за допомогою S-блоків.

Таблиця 8.1

Заміна даних за допомогою S-блоків

		Значення чотирирозрядних слів															
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Номери S-блоків заміни	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	12	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	04	10	09	02	13	08	00	14	06	11	01	12	07	15	05	03

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. 8.1, буде сформовано значення

$$S(a(0) \boxplus k_0) = 0100\ 1011\ 0001\ 0001\ 0101\ 0010\ 0101\ 1110_2.$$

Друга операція — циклічний зсув вліво (на *11* розрядів) *32*-розрядного вектора, отриманого з виходу *S*-блока підстановки. Циклічний зсув виконується регістром зсуву *R*.

Далі результат роботи функції зашифрування $f(\bullet)$ 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 ($a(1) = f(a(0) \boxplus k_0) \oplus b(0)$) записують у накопичувач N_1. Після цього перший цикл завершений.

Наступні цикли здійснюються аналогічно, при цьому в другому циклі з КЗП зчитують підключ k_1, у третьому циклі — k_2, і так далі, у восьмому циклі — k_7. У циклах із *9*-го по *16*-й, а також у циклах із *17*-го по *24*-й підключі 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 із КЗП зворотний: $k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0$.

Отже, за зашифрування в *32* циклах здійснюється такий порядок вибірки із КЗП підключів:

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7,$$
$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0.$$

Після завершення *32*-го циклу результат із суматора CM_2 вводиться в накопичувач N_2, а в накопичувачі N_1 зберігається попереднє заповнення. Отримані після *32*-го циклу зашифрування заповнення накопичувачів N_2 і N_1 є блоком зашифрованих даних $T_{Ш}$, що відповідає блоку відкритих даних T_O.

Рівняння зашифрування в режимі простої заміни мають вигляд:

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(j-1)}) \oplus b(33-j),\ j = 1, 2, \ldots, 8; \\ b(32-j) = a(33-j),\ j = 1, 2, \ldots, 8, \end{cases} \tag{8.1}$$

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(32-j) \bmod 8}) \oplus b(33-j),\ j = 9, 10, \ldots, 31; \\ b(32-j) = a(33-j),\ j = 9, 10, \ldots, 31, \end{cases} \tag{8.2}$$

$$\begin{cases} b(32-j) = f(a(33-j) \boxplus k_{32-j}) \oplus b(33-j),\ j = 32; \\ a(32-j) = a(33-j),\ j = 32. \end{cases} \tag{8.3}$$

де $a(j) = (a_{32}(j), a_{31}(j), \ldots, a_1(j))$ — заповнення N_1 після j-го циклу зашифрування; $b(j) = (b_{32}(j), b_{31}(j), \ldots, b_1(j))$ — заповнення N_2 після j-го циклу зашифрування, $j = 1, 2, \ldots, 32$.

Блок зашифрованих даних $T_{Ш}$ (*64* розряди) виводиться з накопичувачів N_2 і N_1 у такому порядку: із розрядів *1...32* накопичувача N_1, потім із розрядів *1...32* накопичувача N_2, тобто починаючи з молодших розрядів:

$$T_{Ш} = (a_1(32), a_2(32), \ldots, a_{32}(32), b_1(32), b_2(32), \ldots, b_{32}(32)).$$

Інші блоки відкритих даних зашифровуються в режимі простої заміни аналогічно.

Структурну схему алгоритму зашифрування даних у режимі простої заміни наведено на рис. 8.2, де операція циклічного зсуву на одинадцять розрядів вліво позначена $R_{11}^{\leftarrow}$.

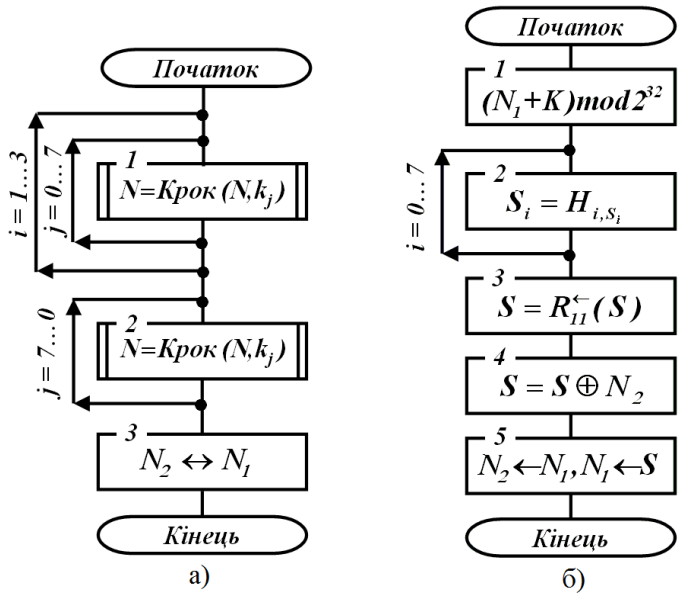


Рис. 8.2. Структурна схема алгоритму зашифрування даних у режимі простої заміни:
а) *32* циклів; б) одного циклу

Структурну схему основного кроку криптографічного перетворення стандарту ($N = Крок(N, K)$) наведено на рис. 8.3.

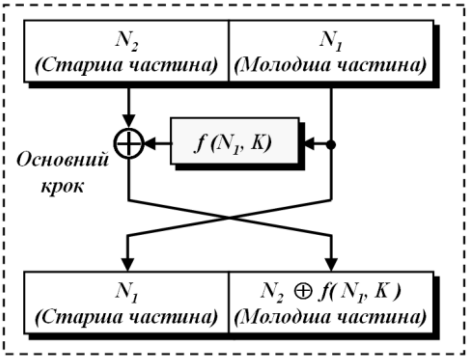


Рис. 8.3. Структурна схема основного кроку криптографічного перетворення стандарту ДСТУ ГОСТ 28147:2009

8.3.3. Розшифрування даних у режимі простої заміни

Криптографічна схема, що реалізовує алгоритм розшифрування даних у режимі простої заміни, має той самий вигляд, що й при зашифруванні (див. рис. 8.1).

Розшифрування даних у режимі простої заміни здійснюється за тим самим алгоритмом, що й зашифрування, з тією зміною, що заповнення накопичувачів зчитуються з КЗП у циклах розшифрування в такому порядку (оберненим у відношенні до зашифрування):

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0,$$

$$k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0 .$$

Відповідно до цього рівняння (8.1)…(8.3) для зашифрування даних у режимі простої заміни можуть бути представлені для розшифрування в такому вигляді:

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(j-1)}) \oplus b(33-j), \; j = 1, 2, \ldots, 8; \\ b(32-j) = a(33-j), \; j = 1, 2, \ldots, 8, \end{cases} \tag{8.4}$$

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(32-j) \bmod 8}) \oplus b(33-j), \; j = 9, 10, \ldots, 31; \\ b(32-j) = a(33-j), \; j = 9, 10, \ldots, 31, \end{cases} \tag{8.5}$$

$$\begin{cases} b(32–j) = f(a(33–j) \boxplus k_{32-j}) \oplus b(33–j),\ j = 32; \\ a(32–j) = a(33–j),\ j = 32. \end{cases} \tag{8.6}$$

Отримані після *32* циклів роботи заповнення накопичувачів N_1 і N_2 утворюють блок розшифрованих (відкритих) даних

$$T_O = (a_1(0), a_2(0), \dots , a_{32}(0), b_1(0), b_2(0), \dots , b_{32}(0)).$$

При цьому стан накопичувача N_1:

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), \dots , a_1(0)),$$

а стан накопичувача N_2

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), \dots , b_1(0)).$$

Аналогічно розшифровуються інші блоки шифрованих даних.

Якщо алгоритм шифрування даних у режимі простої заміни *64*-бітового блока T_O позначити через A, то

$$A(T_O) = A\ (a(0),b(0)) = (a(32), b(32)) = T_{Ш}, \tag{8.7}$$

а

$$A^{-1}(T_{Ш}) = A^{-1}(a(32), b(32)) = (a(0), b(0)) = T_O. \tag{8.8}$$

Слід мати на увазі, що режим простої заміни допустимо використовувати для шифрування даних тільки в обмежених випадках — при виробленні ключа шифру й шифруванні його із забезпеченням імітозахисту для передачі по каналах зв'язку або для зберігання в пам'яті ЕОМ.

У стандарті ДСТУ ГОСТ 28147:2009 зашифрування й розшифрування даних позначають "*Цикл 32–З*" ($Ц_{32–З}$) і "*Цикл 32-Р*" ($Ц_{32–Р}$) відповідно [34].

8.4. ШИФРУВАННЯ ДАНИХ У РЕЖИМІ ГАМУВАННЯ

8.4.1. Структура криптографічної системи шифрування даних у режимі гамування

Для реалізації алгоритму зашифрування даних ДСТУ ГОСТ 28147:2009 у режимі гамування використовується повністю всі блоки загальної криптографічної системи (рис. 8.4).

Позначення на схемі:

- N_1–N_6 — *32*-розрядні накопичувачі;
- CM_1 і CM_3 — *32*-розрядні суматори за модулем 2^{32} ($\boxplus$);
- CM_4 — *32*-розрядний суматор за модулем $2^{32}-1$ ($\boxplus$);
- 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* бітів, 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 (бокс) підстановки (заміни), що складається з восьми вузлів заміни (*S*-блоків заміни) $S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7$.

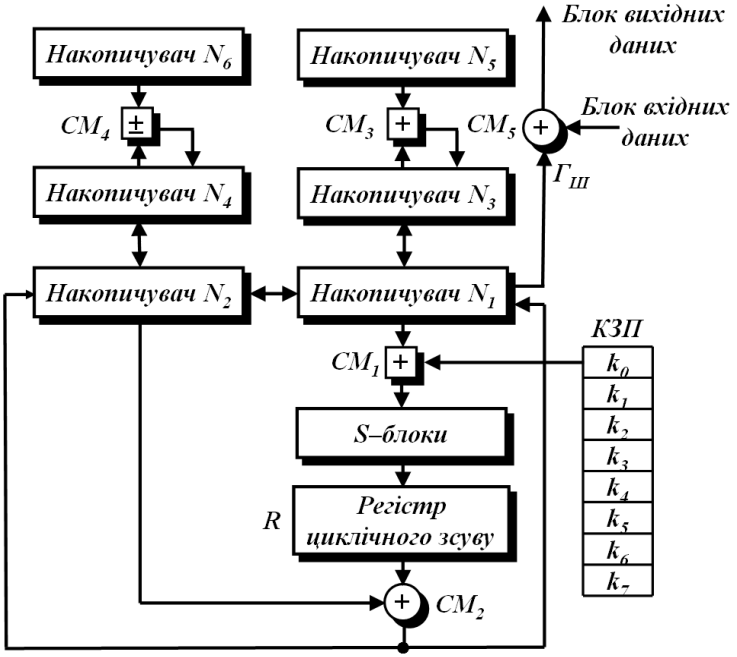


Рис. 8.4. Структура реалізації режиму гамування стандарту ДСТУ ГОСТ 28147:2009

8.4.2. Зашифрування даних у режимі гамування

Структурну схему алгоритму зашифрування даних у режимі гамування наведено на рис. 8.5.

Відкриті дані розбивають на *64*-розрядні блоки

$$\grave{O}_{\hat{I}} = (T_O^{(1)}, T_O^{(2)}, ..., T_O^{(q)}),$$

де $T_O^{(i)}$ — *i*-й *64*-розрядний блок відкритих даних (*i = 1, 2,…, q*, а *q* визначається об'ємом зашифрованих даних).

Ці блоки по черзі зашифровуються в режимі гамування шляхом порозрядного додавання за модулем *2* в суматорі CM_5 із гамою шифру, яка виробляється блоками по *64* біти, тобто

$$\tilde{A}_{\emptyset} = (\tilde{A}_{\emptyset}^{(1)}, \tilde{A}_{\emptyset}^{(2)}, ..., \tilde{A}_{\emptyset}^{(q)}),$$

де $\tilde{A}_{\emptyset}^{(i)}$ — *i*-й *64*-розрядний блок гами (*i = 1, 2,…, q*, а *q* визначається об'ємом зашифрованих даних).

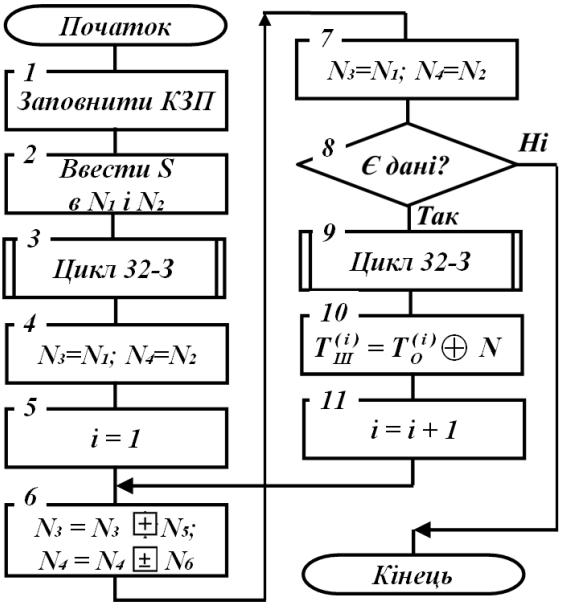


Рис. 8.5. Алгоритм зашифрування даних
ДСТУ ГОСТ 28147:2009 у режимі гамування

Число двійкових розрядів у блоці $\grave{O}_{\hat{I}}^{(q)}$ може бути менше за *64*, при цьому невикористана для зашифрування частина гами шифру з блока $\tilde{A}_{\emptyset}^{(q)}$ відкидається.

Рівняння зашифрування даних у режимі гамування має вигляд

$$\grave{O}_{\emptyset}^{(i)} = \grave{O}_{\hat{I}}^{(i)} \oplus \tilde{A}_{\emptyset}^{(i)},$$

де $\tilde{A}_{\emptyset}^{(i)} = A(Y_{i-1} \boxplus C_2,\ Z_{i-1} \boxplus C_1)$, $i = 1, 2, \ldots, q$; $\grave{O}_{\emptyset}^{(i)}$ — i-й *64*-розрядний блок зашифрованих даних; $A(\bullet)$ — функція зашифрування в 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 $\tilde{A}_{\emptyset}^{(i)}$.

На початку формуються початкові значення Y_i і Z_i таким чином:

$$(Y_0, Z_0) = A(\widetilde{S}),$$

де $\widetilde{S}$ — *64*-розрядна двійкова послідовність (синхропосилка).

Потім ітераційне

$$(Y_i, Z_i) = (Y_{i-1} \boxplus C_2,\ Z_{i-1} \boxplus C_1),\ i = 1, 2, \ldots, q.$$

Розглянемо реалізацію процедури зашифрування в режимі гамування. У накопичувачі N_6 і N_5 заздалегідь записані *32*-розрядні двійкові константи C_1 і C_2, що мають такі значення (у шістнадцятковій і двійковій формі):

$$C_1 = 01010101_{16} = 0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0001_2;$$

$$C_2 = 01010104_{16} = 0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0100_2.$$

У КЗП вводиться *256* бітів ключа; у накопичувачі N_1 і N_2 — *64*-розрядна двійкова послідовність (синхропосилка)

$$\widetilde{S} = (S_1, S_2, \ldots S_{64}),$$

де S_i — i-й двійковий розряд синхропосилки.

Синхропосилка $\widetilde{S}$ є початковим заповненням накопичувачів N_1 і N_2 для послідовного вироблення q блоків гами шифру. Початкове заповнення накопичувача N_1: $N_1 = (S_{32}, S_{31}, \ldots, S_1)$, а початкове заповнення накопичувача N_2: $N_2 = (S_{64}, S_{63}, \ldots, S_{33})$.

Початкове заповнення N_1 і N_2 (синхропосилка) зашифровується в режимі простої заміни. Результат зашифрування $A(\widetilde{S}) = (Y_0, Z_0)$ переписується в *32*-розрядні накопичувачі N_3 і N_4 так, що заповнення N_1, переписується в N_3, а заповнення N_2 — у N_4.

Заповнення накопичувача N_4 додається за модулем $2^{32}–1$ у суматорі CM_4 з *32*-розрядною константою C_1 з накопичувача N_6. Результат записується в N_4. Додавання за модулем $2^{32}–1$ є, по суті, адитивною інверсією результату.

Заповнення накопичувача N_3 додається за модулем 2^{32} в суматорі CM_3 з *32*-розрядною константою C_2 з накопичувача N_5. Результат записується в N_3. Заповнення N_3 переписують у N_1, а заповнення N_4 — у N_2, при цьому заповнення N_3 і N_4 зберігаються. Заповнення накопичувачів N_1 і N_2 зашифровуються в режимі простої заміни.

Отримане внаслідок зашифрування заповнення накопичувачів N_1 і N_2 утворюють перший *64*-розрядний блок гами шифру

$$\tilde{A}_{Ø}^{(1)} = (\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}, ..., \gamma_{63}^{(1)}, \gamma_{64}^{(1)}),$$

де γ_i^1, $i = 1, 2, ..., 64$-двійковий розряд послідовності гами.

Цей блок гами ($\Gamma_{Ш}^{(1)}$) додається порозрядно за модулем *2* в суматорі CM_5 із першим *64*-розрядним блоком відкритих даних

$$\dot{O}_I^{(1)} = (t_1^{(1)}, t_2^{(1)}, t_3^{(1)}, ..., t_{63}^{(1)}, t_{64}^{(1)}),$$

де $t_i^{(1)}$, $i = 1, 2, ..., 64$ — двійковий розряд першого блока зашифрованих даних.

Внаслідок додавання за модулем *2* значень $T_O^{(1)}$ і $\Gamma_{Ш}^{(1)}$ отримують перший *64*-розрядний блок зашифрованих даних:

$$\dot{O}_{Ø}^{(1)} = \dot{O}_I^{(1)} \oplus \tilde{A}_{Ø}^{(1)} = (\tau_1^{(1)}, \tau_2^{(1)}, \tau_3^{(1)}, ..., \tau_{63}^{(1)}, \tau_{64}^{(1)}),$$

де $\tau_i^{(1)} = t_i^{(1)} \oplus \gamma_i^{(1)}$, $i = 1, 2, ..., 64$ — перший двійковий блок зашифрованих даних.

Для отримання наступного *64*-розрядного блока гами шифру $\Gamma_{Ш}^{(2)}$ заповнення N_4 додають за модулем $2^{32}—1$ у суматорі CM_4 з *32*-розрядною константою C_1 із накопичувача N_6. Результат записується в N_4. Заповнення накопичувача N_3 додається за модулем 2^{32} в суматорі CM_3 з *32*-розрядною 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 зашифровуються в режимі простої заміни.

Отримане внаслідок зашифрування заповнення накопичувачів N_1 і N_2 утворюють другий *64*-розрядний блок гами шифру $\Gamma_{Ш}^{(2)}$, який додається порозрядно за модулем *2* в суматорі CM_5 із другим блоком відкритих даних $T_O^{(2)}$:

$$\grave{O}_{\emptyset}^{(2)} = \grave{O}_{\hat{I}}^{(2)} \oplus \tilde{A}_{\emptyset}^{(2)} = (\tau_1^{(2)}, \tau_2^{(2)}, \tau_3^{(2)}, ..., \tau_{63}^{(2)}, \tau_{64}^{(2)}).$$

Аналогічно утворюються блоки гами шифру

$$\tilde{A}_{\emptyset}^{(3)}, \tilde{A}_{\emptyset}^{(4)}, ..., \tilde{A}_{\emptyset}^{(q)}$$

і зашифровуються блоки відкритих даних $\grave{O}_{\hat{I}}^{(3)}$, $\grave{O}_{\hat{I}}^{(4)}$, ..., $\grave{O}_{\hat{I}}^{(q)}$.

У канал зв'язку або пам'ять ЕОМ передаються синхропосилка $\widetilde{S}$ і блоки зашифрованих даних $\grave{O}_{\emptyset}^{(1)}$, $\grave{O}_{\emptyset}^{(2)}$, ..., $\grave{O}_{\emptyset}^{(q)}$.

8.4.3. Розшифрування даних у режимі гамування

У процесі розшифрування даних криптографічна схема ДСТУ ГОСТ 28147:2009 має той самий вигляд, що й під час зашифрування (див. рис. 8.5), а структурну схему алгоритму розшифрування даних у режимі гамування наведено на рис. 8.6.

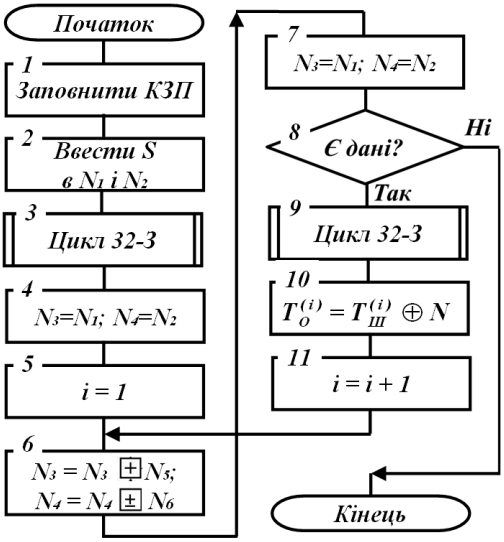


Рис. 8.6. Алгоритм розшифрування даних у режимі гамування стандарту ДСТУ ГОСТ 28147:2009

Рівняння розшифрування:

$$\grave{O}_{\hat{I}}^{(i)} = \grave{O}_{\emptyset}^{(i)} \oplus \tilde{A}_{\emptyset}^{(i)} = \grave{O}_{\emptyset}^{(i)} \oplus A(Y_{i-1} \boxplus C_2, Z_{i-1} \boxplus C_1),\ i = 1, 2, \ldots q$$

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. У накопичувачі N_1 і N_2 вводиться синхропосилка, і здійснюється процес утворення q блоків гами шифру

$$\tilde{A}_{\emptyset}^{(1)},\ \tilde{A}_{\emptyset}^{(2)}, \ldots, \tilde{A}_{\emptyset}^{(q)}.$$

Блоки зашифрованих даних $\grave{O}_{\emptyset}^{(1)}, \grave{O}_{\emptyset}^{(2)}, \ldots, \grave{O}_{\emptyset}^{(q)}$ додаються порозрядно за модулем *2* в суматорі CM_5 із блоками гами шифру $\tilde{A}_{\emptyset}^{(1)}, \tilde{A}_{\emptyset}^{(2)}, \ldots, \tilde{A}_{\emptyset}^{(q)}$. У результаті виходять блоки відкритих даних $\grave{O}_{\hat{I}} = (T_O^{(1)}, T_O^{(2)}, \ldots, T_O^{(q)})$ при цьому $T_O^{(q)}$ може містити менше *64* розрядів.

8.5. ШИФРУВАННЯ ДАНИХ У РЕЖИМІ ГАМУВАННЯ ЗІ ЗВОРОТНИМ ЗВ'ЯЗКОМ

8.5.1. Структура криптографічної системи шифрування даних у режимі гамування зі зворотним зв'язком

Для реалізації алгоритму шифрування даних у режимі гамування зі зворотним зв'язком використовується також тільки частина блоків загальної криптографічної системи стандарту ДСТУ ГОСТ 28147 : 2009 (рис. 8.7).

Позначення на схемі:

- N_1, N_2 — *32*-розрядні накопичувачі;
- CM_1 — *32*-розрядний суматор за модулем 2^{32} ($\dot{\boxplus}$);
- 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* бітів, 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 (бокс) підстановки (заміни), що складається з восьми вузлів заміни (*S*-блоків заміни) S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

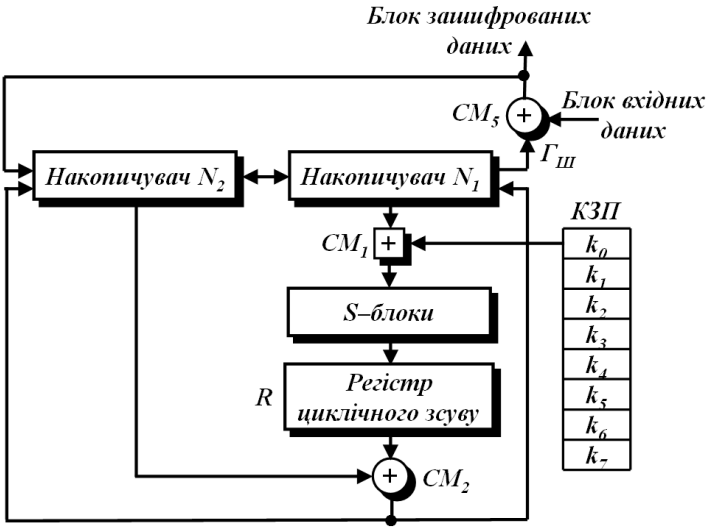


Рис. 8.7. Схема реалізації режиму гамування зі зворотним зв'язком стандарту ДСТУ ГОСТ 28147:2009

8.5.2. Зашифрування даних у режимі гамування зі зворотним зв'язком

Структурну схему алгоритму зашифрування даних у режимі гамування зі зворотним зв'язком наведено на рис. 8.8.

Відкриті дані, розбиті на *64*-розрядні блоки $T_O = (T_O^{(1)}, T_O^{(2)},..., T_O^{(q)})$, зашифровуються в режимі гамування із зворотним зв'язком шляхом порозрядного додавання за модулем *2* з гамою шифру $\Gamma_Ш$, яка виробляється блоками по *64* біти

$$\tilde{A}_Ø = (\tilde{A}_Ø^{(1)}, \tilde{A}_Ø^{(2)},..., \tilde{A}_Ø^{(q)}).$$

Кількість двійкових розрядів у блоці $T_O^{(q)}$ може бути меншою за *64*, при цьому невикористана для зашифрування частина гами шифру з блока $\Gamma_Ш^{(q)}$ відкидається.

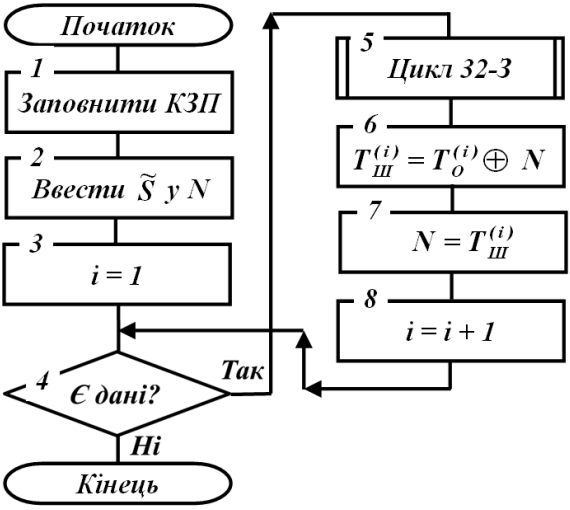


Рис. 8.8. Алгоритм шифрування даних у режимі гамування зі зворотним зв'язком стандарту ДСТУ ГОСТ 28147:2009

Рівняння зашифрування в режимі гамування зі зворотним зв'язком мають вигляд:

$$\grave{O}_{\emptyset}^{(1)} = \tilde{A}_{\emptyset}^{(1)} \oplus T_{\hat{I}}^{(1)} = A(\widetilde{S}) \oplus T_{\hat{I}}^{(1)};$$

$$T_{\emptyset}^{(i)} = A(T_{\emptyset}^{(i-1)}) \oplus T_{\hat{I}}^{(i)},\ i = 2, 3, 4, ..., q,$$

де $T_{Ш}^{(i)}$ — i-й *64*-розрядний блок зашифрованих даних; *A(•)* — функція зашифрування в режимі простої заміни.

Аргументом функції *A(•)* на першому кроці ітеративного алгоритму є *64*-розрядна синхропосилка $\widetilde{S}$, а на усіх наступних кроках — попередній блок зашифрованих даних $T_{Ш}^{(i-1)}$.

Процедура зашифрування даних у режимі гамування зі зворотним зв'язком реалізується таким чином: у КЗП вводяться *256* бітів ключа. У накопичувачі N_1 і N_2 вводиться синхропосилка $\widetilde{S} = (S_1, S_2, ..., S_{64})$ що складається з *64* бітів.

Початкове заповнення накопичувачів N_1 і N_2 (синхропосилка) зашифровується у режимі простої заміни. Отримане внаслідок

зашифрування заповнення накопичувачів N_1 і N_2 утворює перший *64*-розрядний блок гами шифру

$$\tilde{A}_{Ø}^{(1)} = A(\widetilde{S}),$$

який додається порозрядно за модулем *2* в суматорі CM_5 із першим *64*-розрядним блоком відкритих даних:

$$\grave{O}_{\hat{I}}^{(1)} = (t_1^{(1)}, t_2^{(1)}, \quad t_3^{(1)}, ..., t_{63}^{(1)}, t_{64}^{(1)}).$$

Як результат отримують перший *64*-розрядний блок зашифрованих даних

$$\grave{O}_{Ø}^{(1)} = \grave{O}_{\hat{I}}^{(1)} \oplus \tilde{A}_{Ø}^{(1)} = (\tau_1^{(1)}, \tau_2^{(1)}, \tau_3^{(1)}, ..., \tau_{63}^{(1)}, \tau_{64}^{(1)}).$$

Блок зашифрованих даних $T_{Ш}^{(1)}$ одночасно є також початковим станом накопичувачів N_1 і N_2 для утворення другого блока гами шифру $\Gamma_{Ш}^{(2)}$, і тому за зворотним зв'язком $T_{Ш}^{(1)}$ записується у вказані накопичувачі N_1 і N_2.

Заповнення накопичувача N_1:

$$N_1 = (\tau_{32}^{(1)}, \tau_{31}^{(1)}, \tau_{30}^{(1)}, ..., \tau_1^{(1)}, \tau_2^{(1)}),$$

а заповнення накопичувача N_2:

$$N_2 = (\tau_{64}^{(1)}, \tau_{63}^{(1)}, \tau_{62}^{(1)}, ..., \tau_{34}^{(1)}, \tau_{33}^{(1)}).$$

Заповнення накопичувачів N_1 і N_2 зашифрується в режимі простої заміни. Отримане внаслідок зашифрування заповнення накопичувачів N_1 і N_2 утворює другий *64*-розрядний блок гами шифру, який додається порозрядно за модулем *2* в суматорі CM_5 із другим блоком відкритих даних $T_{Ш}^{(2)}$

$$\grave{O}_{Ø}^{(2)} = \grave{O}_{\hat{I}}^{(2)} \oplus \tilde{A}_{Ø}^{(2)} = A(T_{Ø}^{(1)}) \oplus \grave{O}_{\hat{I}}^{(2)}).$$

Утворення наступних блоків гами шифру $\Gamma_{Ш}^{(i)}$ і зашифрування відповідних блоків відкритих даних $T_O^{(i)}$ робиться аналогічно.

Якщо довжина останнього *q*-го блока відкритих даних $T_O^{(q)}$ менше *64*-розрядів, то з $\Gamma_{Ш}^{(i)}$ використовується тільки відповідна кількість розрядів гами шифру, інші розряди відкидаються.

У канал зв'язку або пам'ять ЕОМ передаються синхропосилка $\widetilde{S}$ і блоки зашифрованих даних

$$\grave{O}_{\emptyset} = (\grave{O}_{\emptyset}^{(1)}, \grave{O}_{\emptyset}^{(2)}, \grave{O}_{\emptyset}^{(3)}, ..., \grave{O}_{\emptyset}^{(q-1)}, \grave{O}_{\emptyset}^{(q)}).$$

8.5.3. Розшифрування даних у режимі гамування зі зворотним зв'язком

У процесі розшифрування даних у режимі гамування зі зворотним зв'язком криптографічна схема ДСТУ ГОСТ 28147:2009 має той самий вигляд, що й під час зашифрування (див. рис. 8.7). Структурну схему алгоритму розшифрування даних наведено на рис. 8.9.

Рівняння розшифрування:

$$\grave{O}_{\hat{i}}^{(1)} = A(\widetilde{S}) \oplus T_{\emptyset}^{(1)} = \tilde{A}_{\emptyset}^{(1)} \oplus \grave{O}_{\emptyset}^{(1)};$$

$$T_{\hat{i}}^{(i)} = A(T_{\emptyset}^{(i-1)}) \oplus T_{\emptyset}^{(i)} = \tilde{A}_{\emptyset}^{(i)} \oplus \grave{O}_{\emptyset}^{(i)},\ i = 2, 3, ..., q.$$

Як випливає з наведених співвідношень, розшифрування даних можливе тільки за наявності синхропосилки $\widetilde{S}$, яка не є секретним елементом шифру і може зберігатися в пам'яті ЕОМ або передаватися по каналах зв'язку разом із зашифрованими даними.

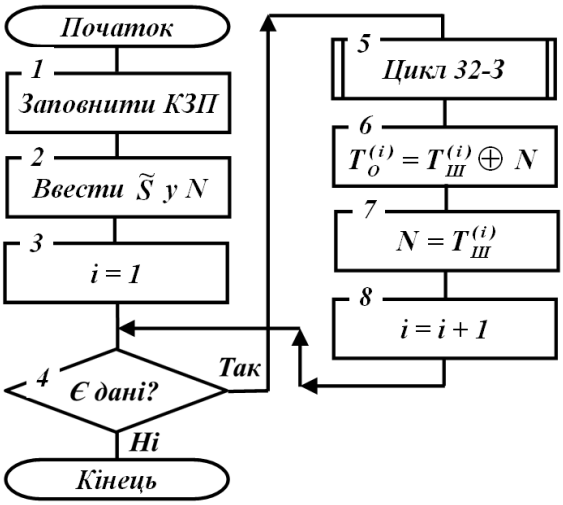


Рис. 8.9. Алгоритм розшифрування даних у режимі гамування зі зворотним зв'язком стандарту ДСТУ ГОСТ 28147:2009

Розглянемо реалізацію процедури розшифрування. У КЗП вводять *256* бітів ключа, за допомогою якого здійснюється зашифрування даних $T_{Ø}^{(1)}$, $T_{Ø}^{(2)}$,..., $T_{Ø}^{(q)}$. У накопичувачі N_1 і N_2 вводиться синхропосилка $\widetilde{S}$. Початкове заповнення накопичувачів N_1 і N_2 (синхропосилка $\widetilde{S}$) зашифровуються в режимі простої заміни. Отримане внаслідок зашифрування заповнення N_1 і N_2 утворює перший блок гами шифру

$$\tilde{A}_{Ø}^{(1)} = A(\widetilde{S}),$$

який додається порозрядно за модулем *2* в суматорі CM_5 із блоком зашифрованих даних $T_{Ш}^{(1)}$. Як результат виходить перший блок відкритих даних

$$\grave{O}_{\hat{I}}^{(1)} = \tilde{A}_{Ø}^{(1)} \oplus \grave{O}_{Ø}^{(1)}.$$

Блок зашифрованих даних $\grave{O}_{Ø}^{(1)}$ є початковим заповненням накопичувачів N_1 і N_2 для утворення другого блока гами шифру $\tilde{A}_{Ø}^{(2)}$

$$\tilde{A}_{Ø}^{(2)} = A(T_{Ø}^{(1)}).$$

Отримане заповнення накопичувачів N_1 і N_2 зашифровується в режимі простої заміни. Утворений внаслідок зашифрування блок $\tilde{A}_{Ø}^{(2)}$ додається порозрядно за модулем *2* в суматорі CM_5 із другим блоком зашифрованих даних $\grave{O}_{Ø}^{(2)}$. Як результат отримують другий блок відкритих даних. Аналогічно в N_1 і N_2 послідовно записують блоки зашифрованих даних

$$\grave{O}_{Ø}^{(2)}, \grave{O}_{Ø}^{(3)}, \grave{O}_{Ø}^{(4)}, ..., \grave{O}_{Ø}^{(q-1)}, \grave{O}_{Ø}^{(q)},$$

з яких у режимі простої заміни утворюються блоки гами шифру $\tilde{A}_{Ø}^{(3)}$, $\tilde{A}_{Ø}^{(4)}$,..., $\tilde{A}_{Ø}^{(q)}$.

Блоки гами шифру додаються порозрядно за модулем *2* в суматорі CM_5 із блоками зашифрованих даних $\grave{O}_{Ø}^{(3)}, \grave{O}_{Ø}^{(4)}, ... \grave{O}_{Ø}^{(q-1)}, \grave{O}_{Ø}^{(q)}$.

Як результат отримують блоки відкритих даних $\grave{O}_{O}^{(3)}, \grave{O}_{O}^{(4)}, ... \grave{O}_{O}^{(q)}$, при цьому останній блок відкритих даних $\grave{O}_{O}^{(q)}$ може містити менше *64* розрядів.

8.6. Криптографічні перетворення даних у режимі утворення імітовставки

Імітовставка — це блок з L бітів, який утворюється за певним правилом із відкритих даних із використанням ключа, що 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 утворення імітовставки, який однаковий для будь-якого з режимів шифрування даних. Імітовставка I_L утворюється з блоків відкритих даних або перед зашифруванням усього повідомлення, або паралельно із зашифруванням по блоках. Перші блоки відкритих даних, які беруть участь в утворенні імітовставки, можуть містити службову інформацію (наприклад адресну частину, час, синхропосилку) і не зашифровуються.

Значення параметра L (кількість двійкових розрядів в імітовставці) визначається криптографічними вимогами з урахуванням того, що ймовірність нав'язування помилкових перешкод дорівнює $P_{ПП} = 2^{-L}$.

Для реалізації алгоритму криптографічного перетворення в режимі утворення імітовставки використовується теж тільки частина блоків загальної криптосистеми (рис. 8.10). Структурну схему алгоритму криптографічних перетворень даних ДСТУ ГОСТ 28147:2009 у режимі утворення імітовставки наведено на рис. 8.11.

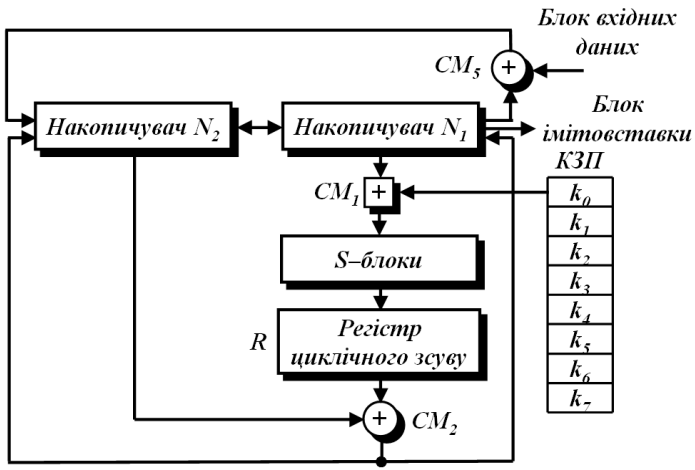


Рис. 8.10. Схема реалізації режиму утворення імітовставки стандарту ДСТУ ГОСТ 28147:2009

Отримане після *16* циклів *64*-розрядне число $\widetilde{A}(T_{\hat{I}}^{(1)})$ додають за модулем *2* із другим блоком відкритих даних $T_O^{(2)}$. Результат додавання $(\widetilde{A}(T_{\hat{I}}^{(1)}) \oplus \dot{O}_{\hat{I}}^{(2)})$ знову зазнає перетворення $\widetilde{A}(\bullet)$.

Отримане *64*-розрядне число $\widetilde{A}(\widetilde{A}(T_{\hat{I}}^{(1)}) \oplus \dot{O}_{\hat{I}}^{(2)}))$ додають за модулем *2* з третім блоком $T_O^{(3)}$ і воно знову зазнає перетворення $\widetilde{A}(\bullet)$, отримуючи *64*-розрядне число

$$\widetilde{A}(\widetilde{A}(\widetilde{A}(T_{\hat{I}}^{(1)}) \oplus \dot{O}_{\hat{I}}^{(2)}) \oplus T_{\hat{I}}^{(3)})$$

тощо.

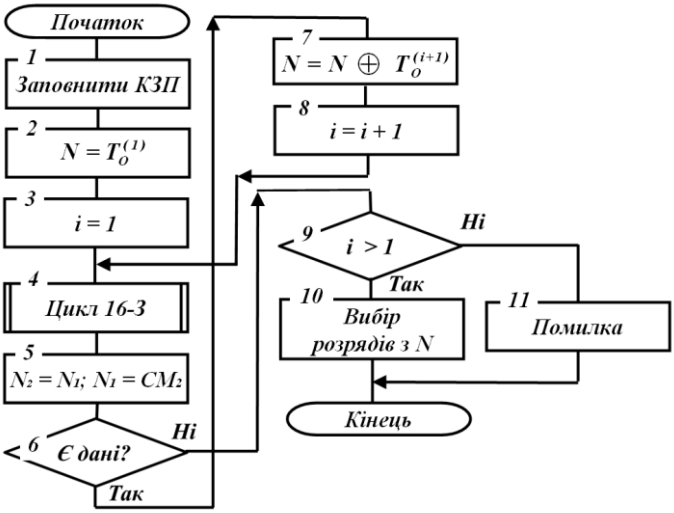


Рис. 8.11. Структурна схема алгоритму криптографічних перетворень даних ДСТУ ГОСТ 28147:2009 у режимі утворення імітовставки

Останній блок $T_O^{(q)}$ (за необхідності доповнений нулями до повного *64*-розрядного блока) додають за модулем *2* з результатом обчислень на кроці *q-1*, після чого він шифрується в режимі простої заміни, використовуючи перетворення $\widetilde{A}(\bullet)$.

З отриманого *64*-розрядного числа вибирають безперервний відрізок I_L (імітовставку) завдовжки L старших (лівих) бітів (не менше одного й не більше *64* бітів)

$$I_L = (\grave{a}_{32}^{(q)}{}_{L+1}(16),\ a_{32}^{(q)}{}_{L+2}(16),..., a_{32}^{(q)}(16)),$$

де $a_i^{(q)}(16)$ — *i*-й біт *64*-розрядного числа, отриманого після *16*-го циклу останнього перетворення $\widetilde{A}(\bullet)$, $32–L+1 \le i \le 32$.

Імітовставка I_L передається по каналу зв'язку або в пам'ять ЕОМ у кінці зашифрованих даних, тобто

$$\grave{O}_{\emptyset}^{(1)}, \grave{O}_{\emptyset}^{(2)}, \grave{O}_{\emptyset}^{(3)},..., \grave{O}_{\emptyset}^{(q\ 1)}, \grave{O}_{\emptyset}^{(q)}, I_L.$$

Зашифровані дані, що надійшли до одержувача,

$$\grave{O}_{\emptyset}^{(1)}, \grave{O}_{\emptyset}^{(2)}, \grave{O}_{\emptyset}^{(3)},..., \grave{O}_{\emptyset}^{(q-1)}, \grave{O}_{\emptyset}^{(q)}$$

розшифровуються, і з отриманих блоків відкритих даних $T_O^{(1)}$, $\grave{O}_{\hat{I}}^{(2)}, \grave{O}_{\hat{I}}^{(3)},..., \grave{O}_{\hat{I}}^{(q-1)}, \grave{O}_{\hat{I}}^{(q)}$ аналогічно утворюється імітовставка I^*_L. Ця імітовставка I^*_L порівнюється з імітовставкою I_L, отриманою разом із зашифрованими даними з каналу зв'язку або з пам'яті ЕОМ. У разі неспівпадання імітовставок отримані в процесі розшифрування блоки відкритих даних $\grave{O}_{\hat{I}}^{(1)}, \grave{O}_{\hat{I}}^{(2)}, \grave{O}_{\hat{I}}^{(3)},..., \grave{O}_{\hat{I}}^{(q-1)}, \grave{O}_{\hat{I}}^{(q)}$ вважають помилковими.

8.7. АНАЛІЗ ШИФРУ

8.7.1. Криптографічна стійкість

Вибираючи криптографічний алгоритм для використання в конкретному проекті, одним із визначних факторів є його *стійкість*, тобто здатність протистояти можливим спробам зламати шифр. Питання про стійкість шифру за його найближчого розгляду зводиться до двох взаємопов'язаних питань:

- чи можливо взагалі розкрити даний шифр;
- якщо так, то наскільки це складно зробити практично.

Шифри, які взагалі неможливо розкрити, називаються *абсолютно*, або *теоретично стійкими*. Існування подібних шифрів доводять за допомогою теореми Шеннона [43], однак ціною подібної стійкості є необхідність використання ключа для зашифрування кожного повідомлення, який за розміром повинен бути не меншим за саме повідомлення. Майже завжди, за винятком особливих випадків, така ціна надто висока, тому на практиці використовуються шифри, які не мають абсолютної стійкості. Отже, найуживаніші схеми шифру-

вання можуть бути розкриті протягом певного часу, тобто після певної кількості кроків, кожний із яких є певною операцією з числами. Для них найважливішим є поняття практичної стійкості, що відображає практичну складність їх розкриття. Кількісним виміром цієї складності може служити кількість елементарних арифметичних і логічних операцій, необхідних для того, щоб розкрити шифр, тобто для того, щоб задане зашифроване повідомлення перетворити, з достатнім значенням імовірності, у відповідне відкрите повідомлення. При цьому на додаток до масиву даних, який дешифрується, криптографічний аналітик може скористатися блоками відкритих даних і відповідних до них зашифрованих даних, або навіть можливістю отримати для будь-яких вибраних ним відкритих даних відповідні зашифровані дані. Залежно від перерахованих і також багатьох невказаних умов розрізняють окремі види криптографічного аналізу.

Усі сучасні криптографічні схеми побудовані за принципом Керкгоффса, тобто секретність зашифрованих повідомлень визначається секретністю ключа. Це означає, що навіть якщо сам алгоритм відомий криптографічному аналітику, той все одно не зможе розшифрувати повідомлення, якщо не має відповідного ключа. Шифр вважається добре спроектованим, якщо немає можливості його розкрити більш ефективним способом, ніж повним перебором по всьому ключовому простору, тобто по всім можливим значенням ключа. ДСТУ ГОСТ 28147:2009, ймовірно, відповідає цьому принципу – за роки інтенсивних досліджень не було запропоновано жодного результативного способу його криптографічного аналізу. Щодо стійкості він на багато перевершує попередній американський стандарт шифрування *DES*.

У ДСТУ ГОСТ 28147:2009 використовується *256*-бітний ключ, 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. Ця величина стала фактичним стандартом розміру ключа для симетричних криптографічних алгоритмів у наші дні, тому, новий стандарт шифрування *США* також його підтримує. Попередній американський стандарт *DES* із його реальним розміром ключа в *56* бітів і обсягом ключового простору лише 2^{56} уже не є достатньо стійким щодо можливостей сучасних обчислювальних засобів. Це було продемонстровано в кінці 90-х рр.

XX ст. кількома вдалими спробами зламу *DES* шляхом перебору. Крім того, виявилося, що *DES* схильний до спеціальних способів криптографічного аналізу таких, як диференційний і лінійний. У зв'язку із цим *DES* може становити швидше дослідницький або науковий, ніж практичний інтерес. 1998 р. його криптографічну уразливість було визнано офіційно: Національний інститут стандартів *США* рекомендував використовувати триразове шифрування за *DES*. А наприкінці 2001 р. було офіційно затверджено новий стандарт шифрування *США*, *AES*, побудований на інших принципах і вільний від недоліків свого попередника [36].

8.7.2. Зауваження до архітектури

Загальновідомо, що вітчизняний стандарт шифрування є представником цілої родини шифрів, побудованих на одних принципах. Найвідомішим його "родичем" є колишній американський стандарт шифрування, алгоритм *DES*. Усі ці шифри, подібно до ДСТУ ГОСТ 28147:2009, містять алгоритм трьох рівнів. В основі завжди лежить певний "*основний крок*", на базі якого аналогічно вибудовуються "*базові цикли*", а вже на їх основі побудовано всі практичні процедури шифрування й вибір імітовставки. Отже, специфіка кожного із шифрів цієї родини криється саме в "*основному кроці*", точніше, в його частині.

Алгоритми "*основних кроків криптографічних перетворювань*" для шифрів, подібних до ДСТУ ГОСТ 28147:2009, побудовані ідентично, і ця архітектура називається *збалансованою мережею Фейстеля (Balanced Feistel Network),* названа на ім'я людини, яка вперше її запропонувала [2, 6]. Схему перетворення даних в одному циклі, або, як його заведено називати, раунді, наведено на рис. 8.3.

На вхід основного кроку подається блок парного розміру, старша й молодша половини якого обробляються окремо одна від одної. У процесі перетворення молодша половина блока встановлюється на місце старшої, а старша, скомбінована за допомогою операції побітного *xor* за результатом обчислення певної функції, — на місце молодшої. Ця функція, приймаючи як аргумент молодшу половину блока та певний елемент ключової інформації (X), є змістовою частиною шифру й називається його *функцією шифрування*. З різних міркувань виявилось вигідним розділити блок, що шифрується, на дві однакові за розміром частини: $|N_1| = |N_2|$ — саме цей факт

відображає слово "*збалансована*" в назві архітектури. Втім, шифрувальні незбалансовані мережі також час від часу використовуються, хоча і не так часто, як збалансовані. Крім того, міркування про стійкість шифру вимагають, щоб розмір ключового елемента не був меншим, ніж розмір половини блока: $|N_i| \le |X|$ у ДСТУ ГОСТ 28147:2009 усі три розміри дорівнюють *32* бітам.

Якщо застосувати вищезазначене до схеми основного кроку алгоритму ДСТУ ГОСТ 28147:2009, стане очевидним, що блоки 1, 2, 3 алгоритму (див. рис. 8.2, б) визначають обчислення його функцій шифрування, а блоки 4 і 5 задають формування вихідного блока основного кроку, виходячи із вмісту вхідного блока та значення функції шифрування.

У попередньому пункті вже було порівняно *DES* і ДСТУ ГОСТ 28147:2009 за стійкістю, тепер співставимо їх за функціональним змістом і зручністю реалізації. У циклах шифрування ДСТУ ГОСТ 28147:2009 основний крок повторюється *32* рази, для *DES* ця величина дорівнює *16*. Однак сама функція шифрування ДСТУ ГОСТ 28147:2009 суттєво простіша від аналогічної функції *DES*, в якій присутня множина нерегулярних бітових перестановок. Ці операції вкрай неефективно реалізуються на сучасних неспеціалізованих процесорах. ДСТУ ГОСТ 28147:2009 не містить подібних операцій, тому він набагато зручніший для програмної реалізації.

Жодна з реалізацій *DES* для процесора *Intel x86* не досягає навіть половини продуктивності запропонованої реалізації ДСТУ ГОСТ 28147:2009, незважаючи на вдвічі коротший цикл. Усе вищесказане свідчить про те, що розробники ДСТУ ГОСТ 28147:2009 врахували як позитивні, так і негативні сторони *DES*, а також реальніше оцінили існуючі та перспективні можливості криптографічного аналізу. Втім, брати *DES* за основу в ході порівняння швидкодії реалізацій шифрів уже не актуально. У новому стандарті шифрування *США* ефективність набагато краща: за такого самого як у ДСТУ ГОСТ 28147:2009 розміру ключа у *256* бітів *AES* працює швидше за нього приблизно на *14%* – це, якщо порівняти кількість елементарних операцій. Крім того, ДСТУ ГОСТ 28147:2009 майже не вдається розділити на паралельні потоки, а в *AES* можливостей щодо цього набагато більше. У деяких архітектурах ця перевага *AES* може бути меншою, а в інших – більшою. Так, на процесорі *Intel Pentium* вона досягає 28%. Подробиці можна знайти в [1, 36, 44, 45].

8.7.3. Вимоги до якості ключової інформації та джерела ключів

Не всі ключі й таблиці замін забезпечують максимальну стійкість шифру. Для кожного алгоритму шифрування існують свої критерії оцінки ключової інформації. Так, для алгоритму *DES* відомо існування так званих *уразливих ключів*, використовуючи які, зв'язок між відкритими й зашифрованими даними маскується недостатньо й шифр порівняно легко розкривається.

Вичерпну відповідь на питання про критерії якості ключів і таблиць замін ДСТУ ГОСТ 28147:2009 якщо й можна де-небудь отримати, то тільки в розробників алгоритму. Відповідні дані не були опубліковані відкрито. Однак, відповідно до встановленого порядку, для шифрування інформації, що має гриф, потрібно використовувати ключові дані, отримані від уповноваженої організації. Опосередковано це може свідчити про наявність методик перевірки ключових даних на достовірність. Якщо наявність уразливих ключів у ДСТУ ГОСТ 28147:2009 – дискусійне питання, то наявність уразливих вузлів заміни не викликає сумніву. Очевидно, що *"тривіальна"* таблиця замін, за якою будь-яке значення замінюється ним же, є настільки вразливою, що за її використання шифр зламується елементарно, яким би не був ключ.

Як вже було зазначено вище, критерії оцінки ключової інформації недоступні, однак щодо них все ж можна висловити деякі загальні міркування.

Ключ

Ключ повинен являти собою масив статично незалежних бітів, які приймають із рівною ймовірністю значення *0* і *1*. Неможливо повністю виключити при цьому те, що деякі конкретні значення ключа можуть виявитися *вразливими*, тобто шифр може не забезпечувати заданий рівень стійкості у випадку їх використання. Однак, імовірно, частка таких значень у загальній масі всіх можливих ключів мізерно мала. Принаймні інтенсивні дослідження шифру до цього часу не виявили жодного такого ключа для жодної з відомих (тобто запропонованих ФАУЗІ) таблиць замін. Тому ключі, отримані за допомогою деякого датчика істинно випадкових чисел будуть якісними з імовірністю, що відрізняється від одиниці на мізерно малу величину. Якщо ж ключі отримуються за допомогою генератора псевдовипадкових чисел, то генератор, який використовується,

повинен забезпечувати зазначені вище статистичні характеристики і, крім того, володіти високою криптографічною стійкістю, не меншою, ніж у самому ДСТУ ГОСТ 28147:2009. Інакше кажучи, завдання визначення відсутніх членів, отриманих генератором послідовності елементів, не повинне бути простішим, ніж завдання розкриття шифру. Крім того, для відбраковування ключів із поганими статистичними характеристиками можуть бути використані різноманітні статистичні критерії. На практиці зазвичай вистачає двох критеріїв: для перевірки рівноймовірного розподілу бітів ключа між значеннями *0* і *1* зазвичай використовується *критерій Пірсона* (“*хі-квадрат*”), а для перевірки незалежності бітів ключа – критерій серій.

Найкращим підходом для отримання ключів було б використання апаратних датчиків випадкових чисел, однак це не завжди прийнято з економічних міркувань. При генерації невеликого за обсягом масиву ключової інформації розумною альтернативою використання такого датчика є і широко використовується на практиці *метод* “*електронної рулетки*”, коли чергова порція випадкових бітів, що утворюються, залежить від моменту часу натиснення оператором деякої клавіші на клавіатурі комп'ютера. У цій схемі джерелом випадкових даних є користувач комп'ютера, а саме – часові характеристики його реакції. За одне натискання клавіші може бути сформовано кілька бітів випадкових даних, тому загальна швидкість формування ключової інформації при цьому невелика – до кількох бітів за секунду. Очевидно, що даний підхід не для отримання великих масивів ключів.

У разі, коли необхідно сформувати великий за обсягом масив ключової інформації можливе й дуже поширене використання різних програмних датчиків псевдовипадкових чисел. Оскільки від подібного датчика вимагають високих показників криптографічної стійкості природним є використання як датчика генератора гами самого шифру – просто “нарізаємо” гаму, що формується шифром, на “шматки” потрібного розміру, для ДСТУ ГОСТ 28147:2009 – по *32* байти. Звичайно, для такого підходу потрібен *майстер-ключ*, який можна отримати описаним вище методом “електронної рулетки”, а з його допомогою, використовуючи шифр у режимі генератора гами, отримати масив ключової інформації потрібного обсягу. Отже ці два способи вироблення ключів — *ручний* і а*лгоритмічний* — працюють у тандемі, допомагаючи один одному. Схеми генерації

ключів у *малобюджетних* системах криптографічного захисту інформації майже завжди побудовані за таким принципом.

Таблиця замін

Таблиця замін є довготривалим ключовим елементом, тобто діє протягом більш тривалого терміну, ніж окремий ключ. Передбачається, що вона є спільною для всіх вузлів шифрування в рамках однієї системи криптографічного захисту. Навіть у разі порушення конфіденційності таблиці замін, стійкість шифру залишається надзвичайно високою й не знижується нижче допустимого рівня. Тому немає особливої потреби тримати таблицю в таємниці, та в більшості комерційних застосувань ДСТУ ГОСТ 28147:2009 так все й відбувається. З іншого боку, таблиця замін є критично важливим елементом для забезпечення стійкості всього шифру. Вибір неналежної таблиці може призвести до того, що шифр буде легко розкриватися відомими методами криптографічного аналізу. Критерії вироблення вузлів замін – "таємниця за сімома печатками" та ФАУЗІ навряд чи нею поділяться з громадськістю в найближчому осяжному майбутньому. У кінцевому підсумку, для того, щоб сказати, чи є дана конкретна таблиця замін вдалою чи невдалою, необхідно зробити величезний обсяг робіт – багато тисяч людино- і машиногодин. Один раз обрана й використана таблиця підлягає заміні в тому, і тільки в тому випадку, якщо шифр з її використанням виявляється вразливим до того чи іншого виду криптографічного аналізу. Тому кращим вибором для пересічного користувача шифру — взяти одну з декількох таблиць, які стали надбанням публічності, наприклад, зі стандарту на хеш-функцію "*центробанківську*". Інформацію про ці таблиці можна знайти відкрито, навіть, в Інтернеті, якщо добре пошукати.

Для тих, хто не звик йти легким шляхом, нижче наведено схему отримання якісних таблиць:

1. За допомогою тієї чи іншої методики створити комплект із восьми вузлів замін із гарантованими характеристиками нелінійності. Таких методик існує декілька, одна з них – використання так званих бент-функцій [31].

2. Перевірити виконання найпростіших критеріїв якості, наприклад, тих, що опубліковані для вузлів заміни *DES*. Ось ще декілька загальних міркувань із цього приводу: кожний вузол заміни може бути описаний чотирма логічними функціями від чотирьох логічних

елементів. Якщо ці функції, записані в мінімальній формі (тобто мінімальною можливою довжиною вираження), виявляються недостатньо складними, вузол заміни відкидається. Крім того, окремі функції в межах усієї таблиці замін повинні відрізнятися одна від одної достатньою мірою. На цьому етапі відсіюються багато наперед неякісних таблиць.

3. Для шифру з вибраними таблицями необхідно будувати різноманітні моделі раунду, що відповідають різним видам криптографічного аналізу, і вимірювати відповідні *профільні* характеристики. Отже, для лінійного криптографічного аналізу необхідно будувати лінійний статистичний аналог раунду шифрування й обчислювати профільну характеристику – показник нелінійності. Якщо вона виявляється недостатньою, то таблиця замін відкидається.

4. Нарешті, використовуючи результати попереднього пункту, потрібно ретельно дослідити шифр з обраною таблицею (спроба криптографічного аналізу всіма відомими методами). Саме цей етап є найскладнішим. Але якщо він зроблений якісно, то з високим ступенем ймовірності можна констатувати, що шифр із вибраними таблицями не буде розкритий будь-ким, навіть спецслужбам.

Проте можна діяти набагато простіше: чим більше в шифрі раундів, тим менший вплив на стійкість цього шифру мають характеристики стійкості одного раунду, у ДСТУ ГОСТ 28147:2009 — *32* раунди, це більше, ніж майже в усіх шифрах з аналогічною архітектурою; тому для більшості випадків побутового і комерційного застосування буває достатньо отримати вузли заміни як незалежні випадкові перестановки чисел від *0* до *15*. Це може бути реалізовано практично, наприклад, за допомогою перемішування колоди із *16* карт, за кожною з яких закріплене одне зі значень зазначеного діапазону.

Стосовно таблиці замін необхідно зазначити ще один цікавий факт. Для оберненості циклів зашифрування й розшифрування не потрібно, щоб вузли заміни були перестановками чисел від *0* до *15*. Усе працює навіть у тому випадку, якщо у вузлі заміни є елементи, що повторюються, і заміна, обумовлена таким вузлом, необернена, проте, у цьому випадку знижується стійкість шифру. Для цього достатньо спробувати спочатку зашифрувати, а потім розшифрувати блок даних, використовуючи таку "*неповноцінну*" таблицю замін, вузли якої містять повторювані значення.

Використання ДСТУ ГОСТ 28147:2009

Часто для використання в системі криптографічного захисту даних з'являється потреба в алгоритмі з більшою, ніж *у* ДСТУ ГОСТ 28147:2009*, швидкодією реалізації, і при цьом*у немає потреби в такій високій криптографічній стійкості. Типовим прикладом подібних завдань є різні електронні біржові торгові системи, що управляють торговими сесіями в реальному часі. Тут від використання алгоритмів шифрування вимагається, щоб було неможливо розшифрувати оперативні дані системи протягом сесії (дані про виставлені заявки, про укладені угоди тощо), по її закінченні ці дані, як правило, вже не приносять користі для зловмисників. Інакше кажучи, є потреба у гарантованій стійкості лише на декілька годин — такою є типова тривалість торговельної сесії. Зрозуміло, що використовувати повноцінний ДСТУ ГОСТ 28147:2009 у цій ситуації було б недоречно.

Як вчинити в цьому й аналогічному випадках, щоб збільшити швидкодію шифрування? Слід використовувати модифікацію шифру з меншою кількістю основних кроків (раундів) в базових циклах. У скільки разів зменшується кількість раундів шифрування, у стільки ж разів зростає швидкодія. Зазначеної зміни можна досягти двома шляхами: зменшенням довжини ключа та зменшенням кількості *циклів перегляду* ключа. Відомо, що кількість основних кроків у базових циклах шифрування дорівнює $N = n \cdot m$, де n — кількість *32*-бітових елементів у ключі, m – кількість циклів використання ключових елементів: згідно зі стандартом $n = 8$, $m = 4$. Можна зменшити будь-яке із чисел, але найпростіший варіант — зменшувати довжину ключа, не торкаючись схеми його використання.

Зрозуміло, що наслідком прискорення роботи буде зниження стійкості шифру. Основна складність полягає в тому, що досить складно більш-менш точно оцінити ступінь цього зниження. Очевидно, що єдиний можливий спосіб зробити це — провести повне дослідження всіх варіантів шифру зі скороченими циклами криптографічного перетворення, але, по-перше, це вимагає використання закритої інформації, якою володіють тільки розробники ДСТУ ГОСТ 28147:2009, і, по-друге, є надто трудомістким. Тому спробуємо дати оцінку, виходячи лише із загальних закономірностей.

Що стосується стійкості шифру до зламу *екстенсивними* методами, тобто до атаки "грубої сили", то тут все більш-менш зрозуміло: ключ розміром *64* біти знаходиться десь на межі доступності цього виду атаки, шифр із ключем *96* бітів і вище (ключ повинен

містити ціле число *32*-бітових елементів) цілком стійкий проти нього. Дійсно, кілька років тому колишній стандарт шифрування США, *DES*, був неодноразово зламаний шляхом перебору: спочатку його зламала обчислювальна мережа, організована на базі глобальної мережі Інтернет, а потім — спеціалізована, тобто сконструйована спеціально для цього, обчислювальна машина. Припустимо, що стандартний варіант ДСТУ ГОСТ 28147:2009, за програмної реалізації на сучасних процесорах, працює у чотири рази швидше за *DES*. Тоді *8*-раундовий "*скорочений ГОСТ*" буде працювати в *16* разів швидше за *DES*. Припустимо також, що за минулий, з моменту зламу *DES*, час, продуктивність обчислювальної техніки згідно із *законом Мура* зросла у чотири рази. Звідси перевірка одного *64*-бітового ключа для "скороченого *ГОСТ*" з вісьмома циклами здійснюється в *64* рази швидше, ніж свого часу виконувалася перевірка одного ключа *DES*. Отже, перевага такого варіанту ДСТУ ГОСТ 28147:2009 перед *DES* за трудомісткістю атаки "грубої сили" скорочується з $2^{64-56} = 2^8 = 256$ до $256/64 = 4$ разів. Це дуже мізерна відмінність, майже нічого.

Набагато складніше оцінити стійкість ослаблених модифікацій ДСТУ ГОСТ 28147:2009 до *інтенсивних* способів криптографічного аналізу. Однак загальну закономірність можна простежити й тут. *Профільні* характеристики багатьох найсильніших на сьогоднішній момент видів криптографічного аналізу експоненціально залежать від кількості раундів шифрування. Отже, для лінійного криптографічного аналізу це буде характеристика лінійності L:

$$L = C \cdot e^{-a \cdot r},$$

де C і α – константи; r – кількість раундів.

Аналогічна залежність існує і для диференційного криптографічного аналізу. За своїм *фізичним змістом* усі характеристики такого роду ймовірностні. 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 кількості основних кроків шифрування. Тому за зниження кількості раундів у кілька разів трудомісткість найбільш відомих видів аналізу зміниться (дуже наближено та грубо) як корінь цього ступеня з початкової кількості. Це надто велике падіння стійкості.

З іншого боку, ДСТУ ГОСТ 28147:2009 проектувався з великим запасом міцності й на сьогодні стійкий до всіх відомих видів

криптографічного аналізу, включаючи диференційний і лінійний. Стосовно лінійного криптографічного аналізу це означає, що для його вдалого проведення потрібно більше пар "*відкритий блок—зашифрований блок*", ніж існує, тобто більше 2^{64}. З урахуванням сказаного вище це означає, що для вдалого лінійного криптографічного аналізу *16*-раундового ДСТУ ГОСТ 28147:2009 потребує не менше $\sqrt{2^{64}} = 2^{32}$ блоків або 2^{35} байтів або *32* Гбайти даних, а для *8*-раундового — не менше $\sqrt[4]{2^{64}} = 2^{16}$ блоків або 2^{19} байтів або *0,5* Мбайта.

Висновки з усього сказаного вище наведено в табл. 8.2, узагальнюючої характеристики скорочених варіантів ДСТУ ГОСТ 28147:2009.

Два останні варіанти, з *12* і *8* раундами, здатні забезпечити дуже обмежений у часі захист. Їх використання виправдане тільки в задачах, де потрібна лише короткострокова секретність закритих даних, приблизно на кілька годин. Можлива область застосування цих слабких варіантів шифру — закриття *UDP*-трафіка електронних біржових торгових систем. У цьому випадку кожний пакет даних (*datagram*, середня *D* з абревіатури *UDP*) шифрується на окремому *64*-бітовому ключі, а сам ключ шифрується на сеансовому ключі (ключі, область дії якого — один сеанс зв'язку між двома комп'ютерами) і передається разом із даними.

Таблиця 8.2

Узагальнюючі характеристики скорочених варіантів ДСТУ ГОСТ 28147:2009

Кількість раундів	Розмір ключа, біт	Індекс швидкодії	Ймовірні характеристики шифру (надто груба оцінка)
24	192	1,33	Стійкий до більшості відомих видів КА або знаходиться на межі стійкості. Практична реалізація КА неможлива через високі вимоги до вихідних даних і трудомісткості
16	128	2	Теоретично нестійкий до деяких видів криптографічного аналізу, проте, їх практична реалізація в більшості випадків утруднена через високі вимоги до вихідних даних і трудомісткості

Закінчення табл. 8.2

Кількість раундів	Розмір ключа, біт	Індекс швидкодії	Ймовірні характеристики шифру (надто груба оцінка)
12	95	2,67	Нестійкий до деяких відомих видів криптографічного аналізу, однак, підходить для забезпечення секретності невеликих обсягів даних (до десятків-сотень Кбайтів) на короткий термін
8	64	4	Нестійкий до деяких відомих видів криптографічного аналізу, однак, підходить для забезпечення секретності невеликих обсягів даних (до десятків Кбайтів) на короткий термін

Примітка. Розмір ключа за незмінної схеми його використання, що припускає спочатку перегляд ключа *3* рази в прямому напрямку, потім один раз у зворотному. Індекс швидкодії показує у скільки разів зросте швидкодія шифрування в порівнянні зі стандартним *32*-раундовим варіантом.

Розглянуте питання має і зворотний бік. Що якщо швидкість шифрування некритична, а вимоги до стійкості надто жорсткі? Підвищити стійкість ДСТУ ГОСТ 28147:2009 можна двома шляхами: умовно назвемо їх "*екстенсивний*" і "*інтенсивний*". Перший із них — це просте збільшення кількості раундів шифрування. Адже вітчизняний стандарт і без цього забезпечує необхідну стійкість. Зазначений підхід дозволяє реально збільшити стійкість шифру: якщо раніше криптографічний аналіз був просто неможливим, то тепер він взагалі неможливий.

Дещо цікавішим є питання про те, чи можна збільшити стійкість шифру, не змінюючи кількості та структури основних кроків шифрування. Як не дивно, відповідь на це питання є позитивною. У ДСТУ ГОСТ 28147:2009 на основному кроці перетворення передбачається виконання заміни *4* на *4* біти, а на практиці всі програмні реалізації виконують заміну побайтно, тобто *8* на *8* бітів — так робиться з міркувань ефективності. Якщо відразу спроектувати таку заміну як *8*-бітову, то суттєво покращаться характеристики одного раунду. По-перше, збільшиться *дифузна* характеристика або показник "*лавинності*" — один біт вхідних даних та/або ключа буде впливати на більшу кількість бітів результату. По-друге, для великих

за розміром вузлів заміни можна отримати найнижчі диференціальну й лінійну характеристики, зменшивши тим самим схильність шифру до однойменних видів криптографічного аналізу. Особливо актуально це для скорочених циклів ДСТУ ГОСТ 28147:2009, а для *8*- і *12*-раундових варіантів такий крок просто необхідний. Це дещо компенсує втрату стійкості в них від зменшення кількості раундів. Ускладнює використання цього прийому — те, що конструювати подібні *збільшені* вузли заміни доведеться самостійно, а також те, що більш великі вузли взагалі конструювати набагато важче, ніж менші за розміром.

8.7.4. Нестандартне використання стандарту

Безумовно, основне призначення криптографічного алгоритму ДСТУ ГОСТ 28147:2009 — це шифрування й імітозахист даних. Однак йому можна знайти й інші застосування, пов'язані, звісно, із захистом інформації. Коротко про них:

1. Для шифрування в режимі гамування ДСТУ ГОСТ 28147:2009 передбачає вироблення криптографічної гами — послідовності бітів із якісними статистичними характеристиками, що володіє високою криптографічної стійкістю. Далі ця гама використовується для модифікації відкритих даних, внаслідок чого виходять зашифровані дані. Однак це не єдине можливе застосування криптографічної гами: алгоритм її формування — це *генератор послідовності псевдовипадкових чисел* (ГППВЧ) з якісними характеристиками. Звичайно, використовувати такий ГППВЧ там, де потрібно тільки отримання статистичних характеристик послідовності, що виробляється, а криптографічна стійкість не потрібна (для цих випадків існують набагато ефективніші генератори). Але для різного застосування, пов'язаного із захистом інформації, таке джерело буде доречним.

Як уже зазначалося вище, гаму можна використовувати як “сировину” для розробки ключів. Для цього потрібно лише отримати відрізок гами необхідної довжини — *32* байти. Таким способом ключі можна розробляти в міру необхідності та їх не потрібно буде зберігати: якщо такий ключ знадобиться повторно, буде досить легко його розробити знову. Слід буде лише згадати, на якому ключі він був розроблений спочатку, яке використовувалося синхропосилання і з якого байта сформованої гами починався ключ. Уся інформація, крім використаного ключа, несекретна. Даний підхід

дозволить легко контролювати досить складну й розгалужену систему ключів, використовуючи лише один *майстер-ключ*.

Аналогічно гаму можна використовувати як вихідну "сировину" для формування паролів. Тут може виникнути питання, навіщо взагалі потрібно їх генерувати, чи не простіше в міру потреби їх просто вигадувати. Неспроможність такого підходу було наочно продемонстровано серією інцидентів у комп'ютерних мережах, найбільшим з яких був добовий параліч Інтернету в листопаді 1988 р., викликаний "*черв'яком Моріса*". Одним зі способів проникнення зловмисної програми на комп'ютер був підбір паролів: програма намагалась увійти в систему, послідовно перебираючи паролі зі свого внутрішнього списку, який складався з кількох сотень, причому в значній частці випадків їй це вдавалося зробити. Фантазія людини щодо вигадування паролів виявилася надто бідною. Саме тому в тих організаціях, де безпеці приділяється належна увага, паролі генерує й роздає користувачам системний адміністратор із безпеки. Розробка паролів є дещо складнішою, ніж формування ключів, оскільки при цьому "сиру" двійкову гаму необхідно перетворити у символьний вигляд, а не просто "нарізати на шматки". Крім того, окремі значення, можливо, доведеться відкинути, щоб забезпечити рівну ймовірність появи всіх символів алфавіту в паролі.

Ще один спосіб використання криптографічного гами — гарантоване затирання даних на магнітних носіях: навіть при перезаписі інформації на магнітному носії залишаються сліди попередніх даних, які можливо відновити відповідною експертизою. Для знищення цих слідів такий перезапис потрібно здійснити багато разів. Виявилося, що необхідно перезаписувати інформацію на носій менше разів, якщо під час такої процедури використовувати випадкові або псевдовипадкові дані, які залишаться невідомими експертам, які намагаються відновити затерту інформацію. Гама шифру тут буде доречною.

2. Не тільки криптографічна гама, але й саме криптографічне перетворення може бути використано для потреб, безпосередньо не пов'язаних із шифруванням.

Відомо, що один із таких варіантів використання ДСТУ ГОСТ 28147:2009 — формування імітовставки для масивів даних. Однак на базі будь-якого блокового симетричного шифру, в тому числі ДСТУ ГОСТ 28147:2009, досить легко побудувати схему обчислення односторонньої хеш-функції, так званої *MDC*, що в різних джерелах роз-

шифровується як код виявлення змін/маніпуляцій (англ. *Modification/ Manipulation Detection Code*) або *дайджест повідомлення* (англ. *Message Digest Code*). *MDC* може безпосередньо використовуватися в системах імітозахисту як аналог імітовставки, що не залежить, однак, від секретного ключа. Крім того, *MDC* широко використовується в схемах *електронного цифрового підпису* (ЕЦП), адже більшість таких схем сконструйовані таким способом, що підписувати зручно блок даних фіксованого розміру. Як відомо, на базі обговорюваного стандарту ДСТУ ГОСТ 28147:2009 побудований стандарт Російської Федерації обчислення односторонньої хеш-функції ГОСТ Р34.11-94.

Контрольні питання та завдання

1. Для яких цілей може використовуватися алгоритм криптографічного перетворення даних за ДСТУ ГОСТ 28147:2009?

2. Перерахувати основні параметри алгоритму симетричного шифрування за ДСТУ ГОСТ 28147:2009.

3. Які операції використовуються в блоковому алгоритмі шифрування за ДСТУ ГОСТ 28147:2009?

4. Чим відрізняється останній цикл (*32*-й) шифрування й розшифрування даних від усіх попередніх?

5. Для чого використовується таблиця замін в алгоритмі ДСТУ ГОСТ 28147:2009?

6. Які основні відмінності алгоритму шифрування за ДСТУ ГОСТ 28147:2009 від алгоритму *DES*?

7. Яка інформація, крім секретного ключа, за умови використання стандартних алгоритмів ДСТУ ГОСТ 28147:2009 необхідна для розшифрування повідомлення?

8. В яких режимах може проводитися шифрування даних за допомогою алгоритму криптографічного перетворення за ДСТУ ГОСТ 28147:2009?

9. Чим відрізняються режими роботи гамування та гамування зі зворотним зв'язком в алгоритмі ДСТУ ГОСТ 28147:2009?

10. Що таке імітовставка? З якою метою може бути використана імітовставка?

11. У регістрі N_1 алгоритму ДСТУ ГОСТ 28147:2009 знаходяться дані, які у шістнадцятковому численні мають вигляд: N_1 — *191a2ab8*, а в N_2 — *434665b2*. Ключ для шифрування в шістнадцятковому численні має вигляд: *eb8a7159 7ce5d63d 4ac1d6e0 bafe4731 a3deb025 8bb389ac 10d3b61a e9ac340f*. Цикл зашифрування — *25*. Що буде знаходиться в N_1 та N_2 після завершення циклу? Як блоки заміни використовувати табл. 8.1.

12. У регістрі N_1 алгоритму ДСТУ ГОСТ 28147:2009 знаходяться дані, які у шістнадцятковому численні мають вигляд: N_1 — *434665b2*, а в N_2 — *191a2ab8*. Ключ для шифрування в шістнадцятковому численні має вигляд: *7ce5d63d 4ac1d6e0 bafe4731 a3deb025 8bb389ac 9ea2923b 9a62e045 e9ac340f*. Цикл розшифрування — *9*. Що буде знаходиться в N_1 та N_2 після завершення циклу? Як блоки заміни використовувати табл. 8.1.

13. У регістрі N_1 алгоритму ДСТУ ГОСТ 28147:2009 знаходяться дані, які у шістнадцятковому численні мають вигляд: N_1 — *10d3b61a*, а в N_2 — *eb8a7159*. Ключ для шифрування в шістнадцятковому численні має вигляд: *e9ac340f a3deb025 8bb389ac 9ea2923b 9a62e045 bafe4731 7ce5d63d 4ac1d6e0*. Цикл зашифрування — *32*. Що буде знаходиться в N_1 та N_2 після завершення циклу? Як блоки заміни використовувати табл. 8.1.

14. У регістрі N_1 алгоритму ДСТУ ГОСТ 28147:2009 знаходяться дані, які в шістнадцятковому численні мають вигляд: N_1 — *0d3b61a5*, а в N_2 — *eb8a7159*. Ключ для шифрування в шістнадцятковому численні має вигляд: *e9ac340f 9a62e045 10d3b61a 8bb389ac 9ea2923b 7ce5d63d eb8a7159 bafe4731*. Цикл розшифрування — *32*. Що буде знаходиться в N_1 та N_2 після завершення циклу? Як блоки заміни використовувати табл. 8.1.

15. В алгоритмі ДСТУ ГОСТ 28147:2009 імовірність нав'язування помилкових перешкод повинна бути не менше $P_{ЛП} = 2{,}4 \cdot 10^{-10}$. Визначити довжину імітовставки в бітах.

16. В алгоритмі ДСТУ ГОСТ 28147:2009 формується імітовставка довжиною *24* біти. Визначити ймовірність нав'язування помилкових перешкод.

Розділ 9

СТАНДАРТ СИМЕТРИЧНОГО АЛГОРИТМУ БЛОКОВОГО ШИФРУВАННЯ ДАНИХ ADVANCED ENCRYPTION STANDARD (Rijndael)

9.1. ІСТОРІЯ СТВОРЕННЯ СТАНДАРТУ

1997 р. Національний інститут стандартів і технологій США (*NIST*) оголосив про початок програми з прийняття нового стандарту криптографічного захисту — стандарту XXI ст. для закриття важливої інформації урядового рівня на заміну алгоритму *DES*, що існує з 1974 р., найпоширенішому криптографічному алгоритму в світі. *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 зламу цього шифру, що істотно відрізняються за ефективністю від повного перебору по ключовому простору.

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 такі 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 кількість 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.

У жовтні 2000 р. 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 цей шифр таким, що ідеально підходить для вбудованих систем. Авторами шифру є *Йон Демен* і *Вінсент Реймен*, початкові букви прізвищ яких і утворюють назву алгоритму — *Rijndael* (*Rijmen*, *Daemen*).

Після цього *NIST* почав підготовку попередньої версії Федерального Стандарту Обробки Інформації (*Federal Information Processing Standard — FIPS*) і в лютому 2001 р. опублікував його на сайті *http://csrc.nist.gov/encryption/aes/*. Протягом 90-денного періоду відкритого обговорення попередню версію *FIPS* переглядали з врахуванням коментарів, після чого почався процес виправлень і твердження. Нарешті, 26 листопада 2001 р. було опубліковано остаточну версію стандарту *FIPS-197*, що описує новий американський стандарт шифрування ADVANCED ENCRYPTION STANDARD (*AES*). Згідно з цим документом стандарт набрав чинності з 26 травня 2002 р.

9.2. ПРИНЦИПИ ПОБУДОВИ АЛГОРИТМУ

Матеріали даного розділу значною мірою ґрунтуються на авторському описі алгоритму *Rijndael* [36] і документі *FIPS-197* [36] Скрізь у подальшому викладі, де стандарт *AES* збігається з алгоритмом *Rijndael*, згадується лише останній. Усі відмінності прийнятого стандарту від криптографічного алгоритму *Rijndael* розглядаються окремо.

Оскільки криптосистема належить до блокових шифрів, вона має багато спільного з *DES*, хоча й не є безпосереднім узагальненням шифру Фейстеля. Для забезпечення криптографічної стійкості алгоритм *Rijndael* включає раунди, що повторюються, кожний із яких складається із замін, перестановок і додавання з ключем. Крім того, *Rijndael* використовує сильну математичну структуру: більшість його операцій заснована на арифметиці поля $GF(2^8)$. Проте на відміну від *DES*, шифрування й розшифрування в цьому алгоритмі — процедури різні.

Алгоритм *Rijndael* оперує байтами, які розглядаються як елементи кінцевого поля $GF(2^8)$. Арифметичні операції в полі відповідають операціям над двійковими багаточленами з $GF(2)$ за модулем незвідного полінома.

Елементами поля є багаточлени степеня не більше 7, які можуть бути задані рядком своїх коефіцієнтів. Якщо представити байт у вигляді

$$\{ a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0 \},\ a_i \in \{ 0,1 \},\ i = 0,1,\ldots,7,$$

то елемент поля описується поліномом

$$a_7 \cdot x^7 + a_6 \cdot x^6 + a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 .$$

Наприклад, байту *{ 11001011 }* (або *{ cb }* у шістнадцятковій формі) відповідає поліном $x^7 + x^6 + x^3 + x + 1$.

Для елементів кінцевого поля визначені адитивні й мультиплікативні операції.

9.2.1. Адитивні операції

Складання елементів кінцевого поля — суть операція порозрядного *xor* і тому позначається як $\oplus$. Приклад виконання операції складання:

- у вигляді поліномів

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

- у вигляді двійкового представлення

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

- у вигляді шістнадцяткового представлення

$$\{ 57 \} \oplus \{ 83 \} = \{ d4 \}.$$

У кінцевому полі для будь-якого ненульового елемента α існує зворотний елемент $-\alpha$, при цьому $\alpha + (-\alpha) = 0$, де нульовий елемент — це *{ 00 }*. У кінцевому полі $GF(2^8)$ справедливо $\alpha + \alpha = 0$, тобто кожний ненульовий елемент є своєю власною адитивною інверсією.

Складання двох поліномів із коефіцієнтами з поля $GF(2^8)$ — суть операції складання поліномів із приведенням подібних членів у полі $GF(2^8)$, тобто

$$a(x) + b(x) = (a_7 \oplus b_7) \cdot x^7 + (a_6 \oplus b_6) \cdot x^6 + (a_5 \oplus b_5) \cdot x^5 +$$
$$+ (a_4 \oplus b_4)\, x^4 + (a_3 \oplus b_3) \cdot x^3 + (a_2 \oplus b_2) \cdot x^2 +$$
$$+ (a_1 \oplus b_1) \cdot x + (a_0 \oplus b_0).$$

Отже, складання двох поліномів — суть операції порозрядного *xor* коефіцієнтів цих поліномів із приведенням подібних членів у полі $GF(2^8)$.

9.2.2. Мультиплікативні операції

Множення елементів кінцевого поля, що позначається далі як •, складніша операція. Множення в полі $GF(2^8)$ — це операція множення багаточленів з узяттям результату за модулем незвідного полінома $p(x)$ восьмого степеня та з використанням операції *xor* за приведення подібних членів.

У *Rijndael* вибраний незвідний поліном $p(x) = x^8 + x^4 + x^3 + x + 1$, або в шістнадцятковій формі *1{ 1b }*. Запис *1{1b }* означає, що присутній "*зайвий*" дев'ятий біт.

Поліном $p(x) = x^8 + x^4 + x^3 + x + 1$, використовуваний для побудови поля $GF(2^8)$, є першим незвідним поліномом восьмого степеня, який наведений у більшості довідників, тобто його вибір досить довільний.

Приклад операції множення:

$$\{ 57 \} \bullet \{ 83 \} = \{ c1 \},$$

оскільки

$$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod p(x) = x^7 + x^6 + 1,$$

де

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 =$$

$$= (x^5 + x^3) \cdot (x^8 + x^4 + x^3 + x + 1) \oplus (x7 + x^6 + 1),$$

а шістнадцяткове представлення полінома $x^7 + x^6 + 1$ відповідає значенню *{ c1 }*.

Для будь-якого ненульового елемента α справедливо $\alpha \times 1 = \alpha$. Мультиплікативної одиницею в полі $GF(2^8)$ є елемент *{ 01 }*.

Операція множення може бути описана й реалізована по-іншому. Помноживши довільний поліном сьомого степеня

$$a_7 \cdot x^7 + a_6 \cdot x^6 + a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

на x, отримуємо

$$a_7 \cdot x^8 + a_6 \cdot x^7 + a_5 \cdot x^6 + a_4 \cdot x^5 + a_3 \cdot x^4 + a_2 \cdot x^3 + a_1 \cdot x^2 + a_0 \cdot x.$$

Приведення отриманого полінома за модулем $p(x) = 1\{1b\}$, дає результат добутку в кінцевому полі $GF(2^8)$. Для цього за $a_7 = 1$ достатньо відняти (застосувати операцію порозрядного *xor*) $p(x)$ з отриманого вище полінома.

Якщо ж $a_7 = 0$, то результат уже виходить звідним. Тоді множення на x (тобто $\{00000010\}$ у двійковій або $\{02\}$ шістнадцятковій формі) виходить із зсувом вліво та можливо подальшим використанням *xor* із поліномом $1\{1b\}$.

Нехай функція *xtime()* здійснює операцію множення на x вищеописаним способом. Застосовуючи функцію *xtime()* n разів, можна отримати результат множення на x^n, а підсумовуючи різні степені x, можна отримати будь-який елемент поля. Наприклад:

$$\{57\} \bullet \{13\} = \{fe\},$$

оскільки

$$\{57\} \bullet \{02\} = xtime(\{57\}) = \{ae\};$$
$$\{57\} \bullet \{04\} = xtime(\{ae\}) = \{47\};$$

$$\{57\} \bullet \{08\} = xtime(\{47\}) = \{8e\};$$

$$\{57\} \bullet \{10\} = xtime(\{8e\}) = \{07\}.$$

Звідки

$$\{57\} \bullet \{13\} = \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) =$$

$$= (\{57\} \bullet \{01\}) \oplus (\{57\} \bullet \{02\}) \oplus (\{57\} \bullet \{10\}) =$$

$$= \{57\} \oplus \{ae\} \oplus \{07\} = \{fe\}.$$

Отже, добуток у полі $GF(2^8)$ – це звичайна операція множення поліномів із взяттям результату за модулем деякого незвідного полінома й із використанням операції *xor* за приведення подібних членів [1].

Раундові перетворення *Rijndael* оперують *32*-розрядними словами. В алгоритмі *32*-бітові слова ототожнюються з поліномами степеня *3* з $GF(2^8)$ [36]. Ототожнення робиться у форматі "*перевертиш*", тобто старший (найбільш значущий) біт відповідає молодшому коефіцієнту полінома. Так, наприклад, слово

$$a_0||a_1||a_2||a_3$$

відповідає поліному

$$a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 .$$

Арифметика в алгоритмі збігається з арифметичними діями в кільці поліномів $GF(2^8)$ за модулем полінома $m(x) = x^4 + 1$. Зазначимо, що поліном $m(x) = x^4 + 1 = (x + 1)^4$ є звідним, і, отже, арифметичні дії в алгоритмі відрізняються від операцій поля $GF(2^8)$, зокрема, бувають пари ненульових елементів, добуток яких дорівнює 0 [36].

Множення двох поліномів із коефіцієнтами з поля $GF(2^8)$ — складніша операція. Припустимо, що помножуються два поліноми

$$a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

і

$$b(x) = b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0 .$$

Результатом множення

$$c(x) = a(x) \otimes b(x)$$

буде поліном

$$c(x) = c_6 \cdot x^6 + c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0,$$

де

$$c_0 = a_0 \cdot b_0;$$

$$c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1;$$

$$c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2;$$

$$c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3;$$

$$c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3;$$

$$c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3;$$

$$c_6 = a_3 \cdot b_3.$$

Для того, щоб результат множення міг бути представлений *4*-байтовим словом, необхідно взяти результат за модулем полінома степеня не більше *4*. Автори шифру вибрали поліном

$$m(x) = x^4 + 1$$

для якого справедливо [30]

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}.$$

Отже, результатом $d(x)$ множення $\otimes$ двох поліномів за модулем $m(x) = x^4 + 1$ є

$$d(x) = a(x) \otimes b(x)$$

буде поліном

$$d(x) = d_3 \cdot x^3 + d_2 \cdot x^2 + d_1 \cdot x + d_0,$$

де $d_0 = a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$;

$d_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3$;

$d_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_0 \cdot b_3$;

$d_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$.

У матричній формі це може бути записано так:

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

Нехай $b(x) = b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0$.

Множенню на x багаточлена $b(x)$ із коефіцієнтами з поля $GF(2^8)$ за модулем $x^4 + 1$, враховуючи властивості останнього, відповідає циклічний зсув байтів у межах слова в бік старшого байта, оскільки

$$x \otimes b(x) = b_2 \cdot x^3 + b_1 \cdot x^2 + b_0 \cdot x + b_3.$$

9.2.3. Операції знаходження мультиплікативно обернених величин

У кінцевому полі для будь-якого ненульового елемента $a(x)$ існує обернений (інверсний) адитивний елемент $-a(x)$, при цьому

$$a(x)+(-a(x)) \bmod p(x) = 0.$$

У полі $GF(2^8)$ справедливо $a(x) + a(x) = 0$, де нульовий елемент — це $\{ 00 \}$, тобто кожний ненульовий елемент є своєю власною адитивною інверсією.

У шифрі *Rijndael* також використовується процедура знаходження елемента мультиплікативно оберненого (інверсного) до даного, тобто такого, для якого виконується рівність

$$a(x) \bullet a^{-1}(x) \bmod p(x) = 1, \qquad (9.1)$$

де $a(x)$ – деякий елемент поля $GF(2^8)$; $a^{-1}(x)$ – його мультиплікативно обернений елемент.

Добуток (9.1) у полі $GF(2^8)$ просто виконувати, якщо розглядати ненульові елементи поля як степені деякого примітивного елемента ω. Для цього визначимо структуру пристрою (генератора елементів поля), який дозволяє поставити у відповідність до кожного ненульового елемента поля відповідний степінь примітивного елемента.

Маємо

$$\begin{aligned} p(x+1) &= (x+1)^8 + (x+1)^4 + (x+1)^3 + (x+1) + 1 = \\ &= (x^8 + 1) + (x^4 + 1) + (x^3 + x^2 + x + 1) + (x + 1) + 1 = \\ &= x^8 + x^4 + x^3 + x^2 + 1 = \tilde{p}(x). \end{aligned} \qquad (9.2)$$

Поліном $\tilde{p}(x)$ примітивний, а тому, відповідність між різними формами представлення елементів у полі $GF(2^8)$ можна отримати, якщо модулювати роботу пристрою, наведеного на рис. 9.1.

Наприклад,

$$\begin{aligned} &(x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1) \bmod p(x) = \\ &= (x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod p(x) = \\ &= x^7 + x^6 + 1 \end{aligned}$$

або

$$\omega^{98} \cdot \omega^{80} = \omega^{178},$$

а тому

$$\{ 57 \} \bullet \{ 83 \} = \{ c1 \}.$$

У полі $GF(2^8)$ справедливо $\omega^{255} = 1$, а тому, ненульові елементи ω^i і ω^j ($i \neq j$) є мультиплікативно оберненими тоді й тільки тоді, якщо $i + j = 255$ [7, 28].

Наприклад, $\omega^4 \cdot \omega^{251} = 1$, тобто $\{11\} \bullet \{b4\} = 1$ або

$$(x^4 + 1) \cdot (x^7 + x^5 + x^4 + x^2) \bmod p(x) = 1.$$

00000001	– {01}	($\omega^0 = 1$)
00000011	– {03}	(ω^1)
00000101	– {05}	(ω^2)
00001111	– {0f}	(ω^3)
00010001	– {11}	(ω^4)
00110011	– {33}	(ω^5)
01010101	– {55}	(ω^6)
11111111	– {ff}	(ω^7)
00011010	– {1a}	(ω^8)
00101110	– {2e}	(ω^9)
. . .	. . .	. . .
11110111	– {f7}	(ω^{25})
00000010	– {02}	(ω^{26})
00000110	– {06}	(ω^{27})
. . .	. . .	. . .
11000111	– {b4}	(ω^{251})
11000111	– {c7}	(ω^{252})
01010010	– {52}	(ω^{253})
11110110	– {f6}	(ω^{254})
00000001	– {01}	(ω^{255})

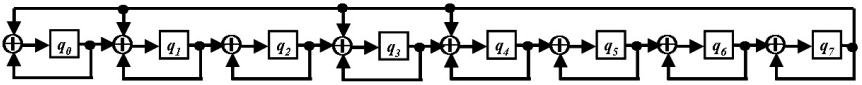


Рис. 9.1. Генератор елементів поля $GF(2^8)$ із примітивним поліномом $\tilde{p}(x) = x^8 + x^4 + x^3 + x + 1$

Тут i та j — кількість тактів, необхідних для переводу генератора елементів поля $GF(2^8)$ у стан зі значенням ω^i і ω^j відповідно.

Визначення мультиплікативно обернених величин у полі $GF(2^8)$ можливо з використанням поданого нижче прикладу.

Приклад 9.1. Нехай деякий байт дорівнює {2d} у полі $GF(2^8)$. У поліноміальному уявленні це $a(x) = x^5 + x^3 + x^2 + 1$.

Для знаходження мультиплікативно оберненого елемента $a^{-1}(x)$ до $a(x)$ у полі $GF(2^8)$ із незвідним багаточленом $p(x) = x^8 + x^4 + x^3 + x + 1$ будемо ділити багаточлени до тих пір, поки максимальна степінь залишку від ділення не буде меншим за максимальний степінь дільника. Якщо максимальна степінь залишку від ділення більше *1*, ділення триває, тільки як ділене береться дільник, а як дільник — залишок від попереднього ділення.

1.

	x^8	$+x^4$	$+x^3$	$+x$	$+1$	x^5	$+x^3$	$+x^2$	$+1$
$\oplus$	x^8	$+x^6$	$+x^5$	$+x^3$		x^3	$+x$	$+1$	
	x^6	$+x^5$	$+x^4$	$+x$	$+1$				
$\oplus$	x^6	$+x^4$	$+x^3$	$+x$					
	x^5	$+x^3$	$+1$						
$\oplus$	x^5	$+x^3$	$+x^2$	$+1$					
				x^2					

Результат проведеного ділення можна представити у вигляді

$$p(x) = x^8 + x^4 + x^3 + x + 1 = (x^5 + x^3 + x^2 + 1) \times x (x^3 + x + 1) + x^2 = a(x)\cdot(x^3 + x + 1) + x^2. \quad (9.3)$$

У зв'язку з тим, що максимальний степінь залишку від ділення (x^2) менше максимального степеня дільника (x^5), процес поточного ділення закінчується. Оскільки залишок від ділення x^2 більше *1*, то ділення триває, тільки як ділення береться дільник ($x^5 + x^3 + x^2 + 1$), а як дільник — залишок від попереднього ділення (x^2).

2.

	x^5	$+x^3$	$+x^2$	$+1$	x^2			
$\oplus$	x^5				x^3	$+x$	$+1$	
	x^3	$+x^2$	$+1$					
$\oplus$	x^3							
	x^2	$+1$						
$\oplus$	x^2							
			1					

Як результат цього ділення вийшов залишок *1*, тому процес ділення на цьому припиняється.

Результат проведеного ділення можна представити у вигляді

$$a(x) = x^5 + x^3 + x^2 + 1 = (x^3 + x + 1)\cdot x^2 + 1. \quad (9.4)$$

Представимо вираз (9.4) у такому вигляді:

$$a(x) + (x^3 + x + 1)\cdot x^2 = 1, \tag{9.5}$$

а вираз (9.3)

$$p(x) + a(x)\cdot(x^3 + x + 1) = x^2. \tag{9.6}$$

Підставляючи (9.5) у (9.6) отримуємо:

$$\begin{aligned} &a(x) + (x^3 + x + 1)\cdot(p(x) + a(x)\cdot(x^3 + x + 1)) = \\ &= a(x) + (x^3 + x + 1)\cdot p(x) + s(x)\cdot(x^3 + x + 1)^2 = \\ &= a(x)\cdot(1 + (x^3 + x + 1)^2) + (x^3 + x + 1)\cdot p(x) = 1. \end{aligned} \tag{9.7}$$

Обчислюючи від лівої та правої частини (9.7) модульне значення в $GF(2^8)$ із незвідним поліномом $p(x) = x^8 + x^4 + x^3 + x + 1$ отримаємо:

$$\begin{aligned} &(a(x)\cdot(1 + (x^3 + x + 1)^2) + (x^3 + x + 1)\cdot p(x)) \bmod p(x) = \\ &= a(x)\cdot(1 + (x^3 + x + 1)^2) \bmod p(x) + \\ &+ ((x^3 + x + 1)\cdot p(x)) \bmod p(x) = \\ &= a(x)\cdot(1 + (x^3 + x + 1)^2 \bmod p(x) = 1, \end{aligned} \tag{9.8}$$

оскільки $((x^3 + x + 1)\cdot p(x)) \bmod p(x) = 0$.

Із виразу (9.8) виходить, що

$$a(x)\cdot(1 + (x^3 + x + 1)^2) \bmod p(x) = 1. \tag{9.9}$$

Зіставимо (9.9) і (9.1) отримаємо

$$a^{-1}(x) = (1 + (x^3 + x + 1)^2). \tag{9.10}$$

Роблячи перетворення в (9.10), остаточно отримаємо

$$a^{-1}(x) = x^6 + x^2. \tag{9.11}$$

У шістнадцятковій системі числення значення байта $a^{-1}(x)$ буде дорівнювати *{ 44 }*.

Перевіримо правильність знаходження мультиплікативно оберненого елемента до $a^{-1}(x)$ у полі $GF(2^8)$ з незвідним поліномом $p(x) = = x^8 + x^4 + x^3 + x + 1$ до елемента $a(x) = x^5 + x^3 + x^2 + 1$.

Для цього підставляючи значення елементів $a(x)$ і $a^{-1}(x)$ у вираз (9.1) отримаємо:

$$a(x)\cdot a^{-1}(x) \bmod p(x) = (x^5 + x^3 + x^2 + 1)\cdot(x^6 + x^2) = \\ = (x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^2) \bmod p(x) = 1. \quad (9.12)$$

Як випливає з (9.12) елементи $a(x)$ і $a^{-1}(x)$ є мультипликативно оберненими в полі $GF(2^8)$ з незвідним поліномом $p(x)$.

9.3. ФОРМАТ ДАНИХ АЛГОРИТМУ

Rijndael — це ітераційний блоковий шифр, що має архітектуру "*Квадрат*". Шифр має змінну довжину блоків і різні довжини ключів. Довжина ключа й довжина блока можуть дорівнювати незалежно один від одного *128*, *192* або *256* бітам. У стандарті *AES* визначена довжина даних, які обробляються, що дорівнює *128* бітам.

AES використовує п'ять одиниць для представлення даних: *біти*, *байти*, *слова*, *блоки* й *масиви станів*. *Біт* — найменша й елементарна одиниця; інші одиниці можуть бути виражені в термінах менших одиниць. В *AES* біт — двійкова цифра зі значенням *0* або *1*. Для позначення бітів будемо використовувати малі літери.

Рис. 9.2 показує одиницю (формат) даних – байт.

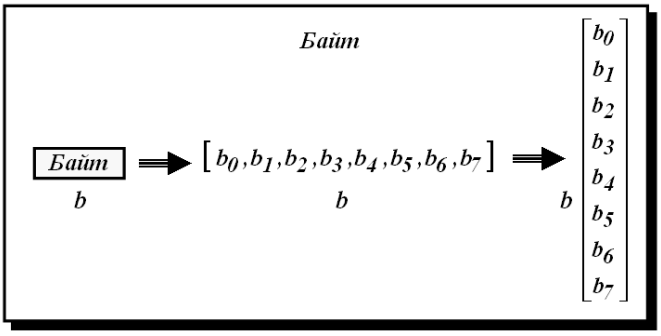


Рис. 9.2. Одиниця (формат) даних – байт

Байт — група з восьми бітів, яка може бути оброблена як єдиний об'єкт: матриця з одного рядка восьми бітів (1×8) або стовпець матриці з восьми бітів (8×1). Коли інформація байта обробляється як матриця рядка, то біти вставляються в матрицю зліва направо. Коли байт

обробляється як матриця стовпця, біти вставляються в матрицю зверху вниз. Для позначення байта будемо використовувати малу літеру *b*.

Рис. 9.3 показує одиницю (формат) даних – слово.

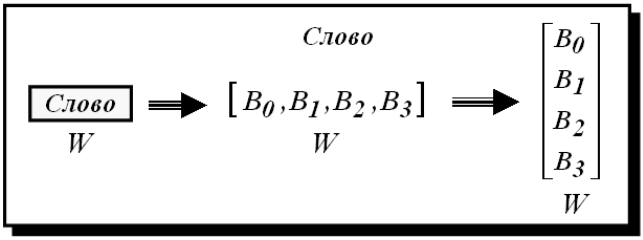


Рис. 9.3. Одиниця (формат) даних – слово

Слово — група з *32* бітів, яка може бути оброблена як єдиний об'єкт. Це рядок або стовпець матриці із чотирьох байтів. Коли слово обробляється як матриця-рядок, байти вставляються зліва направо. Коли слово представлене матрицею-стовпцем, байти вставляються зверху вниз. Для позначення слова будемо використовувати велику літеру *W*.

Рис. 9.4 показує одиницю (формат) даних – блок.

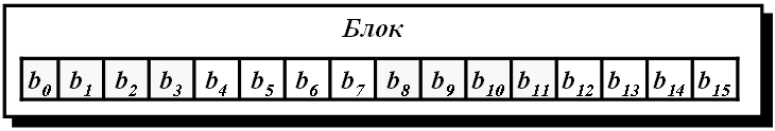


Рис. 9.4. Одиниця (формат) даних – блок

AES зашифровує й розшифровує блоки даних. *Блок* в *AES* — група з *128* бітів. Однак блок може бути представлений як матриця-рядок з *16* байтів.

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 станів*. Для позначення цієї матриці будемо використовувати велику літеру *S*. Хоча матриця станів на різних каскадах зазвичай позначається *S*, іноді будемо застосовувати велику літеру *T*, щоб позначити тимчасову матрицю станів.

Рис. 9.5 показує одиницю (формат) даних – матрицю стану.

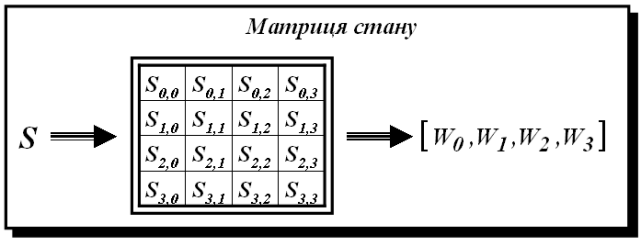


Рис. 9.5. Одиниця (формат) даних – матрицю стану

Матриці стану, подібно до блоків, складаються з *16* байтів, але зазвичай обробляються як матриці 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 в матриці стану витягуються, як це показано на рис. 9.6.

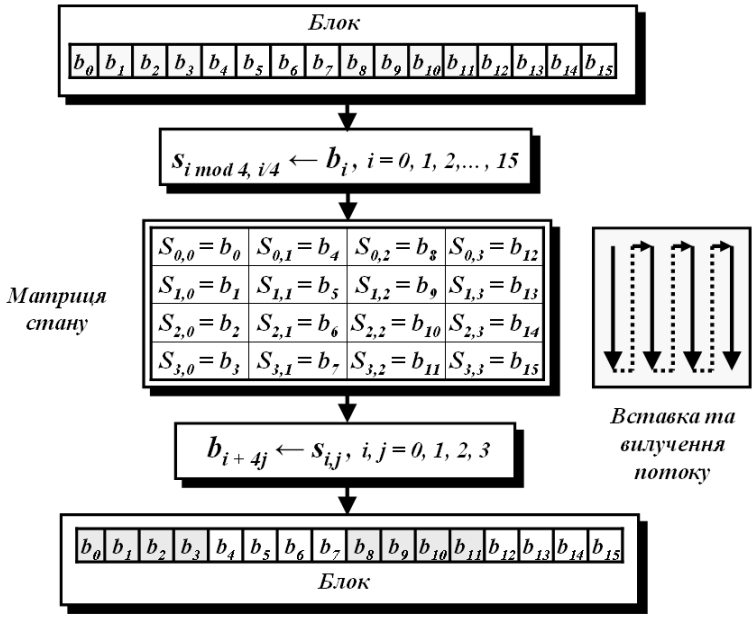


Рис. 9.6. Перетворення блока в матрицю стану й матрицю стану в блок

Приклад 9.2. Розглянемо, як можливо представити блок із *16* символами у вигляді матриці розміром 4×4. Приймемо, що текстовий блок — "*AES uses a matrix*". Додамо в кінець блока два фіктивних символи та отримаємо блок "*AESUSESAMATRIXZZ*". Тепер проведемо заміну кожного символу його цілим десятковим порядком (числом) між *00* і *25*.

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 наведено на рис. 9.7.

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*	*23*
12	*12*	*13*	*19*
14	*00*	*11*	*19*

Рис. 9.7. Перехід вхідного тексту в матрицю стану

9.4. СТРУКТУРА АЛГОРИТМУ Й РАУНДІВ

9.4.1. Структура алгоритму

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.

Рис. 9.8 показує загальну схему: алгоритм зашифрування (званий шифром); алгоритм розшифрування (званий зворотним шифром), для якого застосовуються ті самі ключі, але у зворотному порядку.

На рис. 9.8 *Nr* визначає кількість раундів. Рисунок також показує відношення між кількістю раундів і розміром ключа. Це означає, що

існують три різні версії *AES*; вони позначаються як *AES-128*, *AES-192* і *AES-256*. Проте ключі раунду, які створені алгоритмом розширення ключів завжди *128* бітів, мають той самий розмір, що й блоки зашифрованих або вихідних даних.

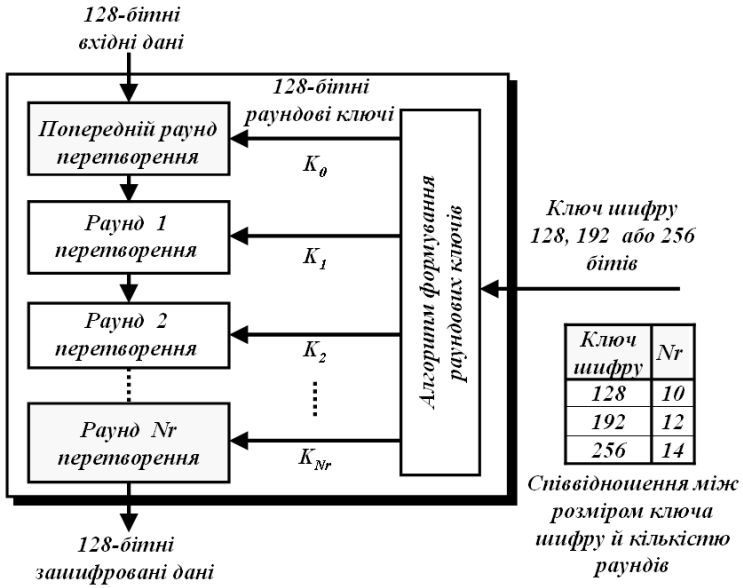


Рис. 9.8. Загальна побудова схеми шифрування *AES*

AES визначив три версії, з *10*, *12* і *14* раундами. Кожна версія використовує різний розмір ключа шифру (*128*, *192* або *256*), але ключ раунду — завжди *128* бітів.

Число ключів раунду, згенерованих алгоритмом розширення ключів, завжди на одне більше, ніж число раундів. Іншими словами, маємо кількість раундових ключів $Nr + 1$. Позначимо раундові ключі як: $K_0, K_1, K_2, ..., K_{Nr}$.

9.4.2. Структура раундів алгоритму

Рис. 9.9 показує структуру кожного раунду на стороні зашифрування.

Кожний раунд, крім останнього, використовує чотири перетворення, які є оберненими. Останній раунд має тільки три перетворення.

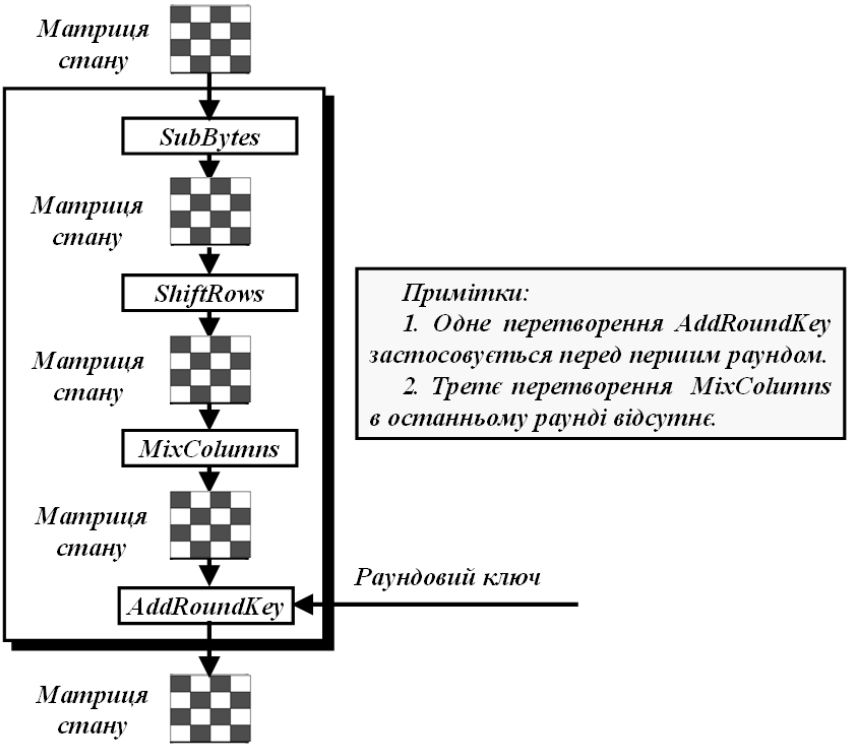


Рис. 9.9. Структура кожного раунду *AES* на стороні зашифрування

Попередній раунд використовує тільки одне перетворення (*AddRoundKey*); останній раунд використовує тільки три перетворення (*MixColumns* — перетворення відсутнє).

9.4.3. Кількість раундів алгоритму

Для виконання перетворень у блоці використовується раундовий ключ *W*, що отримується із секретного ключа *K*. Раундовий ключ у стандарті *AES* складається з блоків (по *128* бітів). Також у стандарті *AES* (*Rijndael*) визначено відповідність між розміром ключа, розміром блока даних і кількістю раундів шифрування, як показано в табл. 9.1 і 9.2.

Таблиця 9.1

Кількість раундів *Nr* як функція від довжини ключа *Nk* і довжини блока *Nb*

Nr	*Nb = 4*	*Nb = 6*	*Nb = 8*
Nk = 4	*10*	*12*	*14*
Nk = 6	*12*	*12*	*14*
Nk = 8	*14*	*14*	*14*

Таблиця 9.2

Відповідність між довжиною ключа, розміром блока даних і кількістю раундів у стандарті *AES*

Стандарт	Довжина ключа (*Nk* слів)	Розмір блока даних (*Nb* слів)	Кількість раундів (*Nr*)
AES-128	*4*	*4*	*10*
AES-192	*6*	*4*	*12*
AES-256	*8*	*4*	*14*

Автори шифру визначили кількість раундів, враховуючи максимальну кількість раундів, для яких можлива ще ефективніша атака, ніж повний перебір по всьому ключовому простору (так звана скорочена атака), і додали запас у вигляді додаткових раундів.

Для спрощеної версії шифру із *6* раундів зі *128*-бітовими блоками даних і ключем не було виявлено жодної ефективної скороченої атаки. Додавання ще *4* раундів з урахуванням таких міркувань є більш ніж достатнім запасом міцності: два раунди шифру забезпечують повне розсіювання в тому сенсі, що кожний біт блока *State* залежить від усіх бітів цього самого блока "*дворaундової давності*", або, інакше кажучи, зміна одного біта блока *State* з великою ймовірністю відіб'ється на половині бітів цього блока через *2* раунди. Отже, додаткові *4* раунди можуть розглядатися як додавання кроку повного розсіювання на початку й у кінці процедури шифрування.

Високий ступінь розсіювання шифру обумовлений цілісним характером алгоритму, що перетворює відразу всі біти вхідних даних.

За лінійного й диференціального криптографічного аналізу при проведенні атаки методом скорочених диференціалів використовують залежності, які можна простежити крізь *n* раундів для того, щоб

атакувати $(n+1)$-й або $(n+2)$-й раунди. Це також правильно й для *Square*-атак, що використовують залежності крізь *4* раунди для атаки на шостий раунд. Для останніх додаткові *4* раунди означають подвоювання кількості раундів, крізь які необхідно відстежувати залежності.

Для версій шифру з довшим ключем кількість раундів збільшується на одиницю для кожних додаткових *32* розрядів ключа шифрування, виходячи з таких міркувань: одна з головних цілей — неможливість скорочених атак, оскільки із зростанням довжини ключа трудомісткість простого перебору збільшується, то і скорочена атака може бути більш трудомісткою.

Атаки по відомому (чи частково відомому) ключу, атаки з використанням еквівалентних ключів засновані відповідно на наявності інформації про біти початкового ключа шифрування й можливості застосування декількох різних ключів шифрування. Якщо ключ подовжується, то й можливостей у криптографічного аналітика стає більше.

Оскільки ніяких ефективних атак по відомому ключу або з використанням еквівалентних ключів для шифру із шістьма раундами знайдено не було, то інші раунди можуть розглядатися як додатковий запас стійкості.

Для інших версій шифру *Rijndael* (не включених у стандарт) із блоками вхідних даних, розмір яких більше *128* розрядів, кількість раундів також збільшується на одиницю з кожним додатковим *32*-розрядним словом, виходячи з таких міркувань: для блоків розміром більше *128* розрядів повне розсіювання відбудеться не раніше, ніж після проведення третього раунду перетворень, тобто відносне розсіювання раунду зменшується із зростанням розміру блока вхідних даних.

Із зростанням розміру блока вхідних даних також збільшується кількість можливих комбінацій залежностей вхідних/вихідних даних. А це, своєю чергою, іноді дозволяє продовжити ефективність атаки ще на один раунд.

Автори шифру стверджують, що загроза розповсюдження ефективності атаки ще на один раунд навіть для *256*-розрядного блока малоймовірна. Тому всі раунди, номери яких більше шести, можна розглядати як додатковий запас стійкості шифру.

В основу розробки *Rijndael* покладено три критерії:

- стійкість відносно всіх відомих атак;
- швидкість і компактність коду;
- простота дизайну.

На відміну від попередніх розглянутих нами симетричних блокових шифрів, *Rijndael* не використовує який-небудь аналог структури Фейстеля. Кожний раунд складається з трьох різних обернених перетворень, що називаються *шарами*:

- лінійний змішуючий шар гарантує високий ступінь взаємопроникнення символів блока для маскування статистичних зв'язків;
- нелінійний шар реалізований за допомогою *S*-боксів, що мають оптимальну нелінійність, і запобігає можливості використання диференціального, лінійного й інших сучасних методів криптографічного аналізу;
- шар складання з ключем безпосередньо виконує шифрування.

Шифр починається й закінчується складанням із ключем. Це дозволяє закрити вхід першого раунду під час атаки за відомим текстом і зробити криптографічно значущим результат останнього раунду.

9.5. РАУНДОВІ ПЕРЕТВОРЕННЯ АЛГОРИТМУ

Для шифрування даних алгоритм *AES* (*Rijndael*) використовує так звану колову функцію, яка складається із чотирьох різних байт-зорієнтованих (раундових) перетворень:

- заміни байтів масиву станів із використанням таблиці підстановки — *SubBytes (State)*, тобто побайтової підстановки в *S*-блоках із фіксованою таблицею замін розмірністю 8×256;
- зсув рядків у масиві станів на різну кількість позицій — *ShiftRows (State)*;
- перемішування стовпців масиву станів — *MixColumns (State)* (множення стовпців стану, що розглядаються як багаточлени над $GF(2^4)$, на багаточлен третього степеня $g(x)$ за модулем $x^4 + 1$);
- додавання раундового ключа до масиву станів — *AddRoundKey (State, RoundKey)*, тобто порозрядного *xor* масиву станів із поточним фрагментом розгорнутого ключа.

Для розшифрування даних алгоритм *AES* (*Rijndael*) використовує також колову функцію, яка складається із чотирьох різних байт-зорієнтованих (раундових) перетворень, які є оберненими (інверсними) до перетворень при зашифруванні відповідно: *InvSubBytes*

(State); *InvShiftRows (State)* та *InvMixColumns(State)*. Інверсне перетворення *AddRoundKey(State, RoundKey)* аналогічне самому перетворенню, тому що використовує операцію порозрядного *xor*.

9.5.1. Заміна байтів масиву станів — *SubBytes (State)* і *InvSubBytes (State)*

Процедура *SubBytes()* (заміна байтів) використовується на стороні шифрування й реалізує шар нелінійного перетворення (є нелінійною заміною байтів), що виконується незалежно з кожним байтом стану. Таблиці заміни *S*-блока є такими, що інвертуються й побудовані з композиції таких двох перетворень вхідного байта:

- отримання мультиплікативно оберненого елемента відносно множення в полі $GF(2^8)$, тобто розв'язання рівняння:

$$s(x)\cdot s^{-1}(x) \bmod p(x) = 1 \qquad (9.13)$$

відносно елемента $s^{-1}(x)$ (нульовий елемент *{ 00 }* переходить сам у себе);

- застосування афінного перетворення над $GF(2^8)$ визначеного таким чином [1, 2, 9]:

$$d(x)\cdot = (a(x)\cdot s^{-1}(x) + b(x)) \bmod (x^8 + 1), \qquad (9.14)$$

де $a(x)$ і $b(x)$ – поліноми (багаточлени), які мають фіксовані значення для алгоритму *AES* (*Rijndael*) і дорівнюють

$$a(x) = x^7 + x^6 + x^5 + x^4 + 1; \quad b(x) = x^6 + x^5 + x + 1.$$

Блоки замін у шифрі *AES* (*Rijndael*) відіграють важливу роль. Згідно із засадничими принципами, сформульованими ще К. Шенноном, перетворення даних, використовуваних у шифрі, повинні надавати останньому дві основні властивості: розсіювання й перемішування. Розсіювання припускає поширення впливу кожного біта відкритого повідомлення, а також кожного біта ключа на значну кількість бітів зашифрованого повідомлення. Перемішування призводить до втрати в процесі шифрування будь-яких залежностей між бітами відкритого повідомлення. Тобто на виході ми отримуємо дані, начебто ми вибрали їх абсолютно випадково, а не як значення

функцій перетворення шифруючого алгоритму. Саме ці дві властивості забезпечують захист від двох можливих загроз: підробки повідомлення та його розкриття.

За забезпечення цих двох властивостей відповідають функції *MixColumns()* і *SubBytes()*. Розсіювання в шифрі *Rijndael*, забезпечується здебільше функцією *MixColumns()*, перемішування — здебільше функцією *SubBytes()*. Отже, критерії вибору й розробки таблиці замін *S*-блоків обумовлені, з одного боку, обліком можливостей диференціального й лінійного криптографічного аналізу (будуть розглянуті далі), а з іншого — обліком можливих алгебраїчних маніпуляцій, наприклад, атаки методом інтерполяції (будуть розглянуті далі).

Отже, основними критеріями вибору таблиці замін *S*-блоків стали:

- оберненість;

- мінімізація найдовшої можливої нетривіальної кореляції між лінійними комбінаціями бітів вхідних і вихідних даних; інакше кажучи, для усіх випадково вибраних вхідних даних кількість раундів, в яких ще можна простежити залежності між вхідними і вихідними даними, буде мінімальна; саме дослідження таких залежностей є основою лінійного криптографічного аналізу;

- мінімізація найбільшого нетривіального значення *xor* таблиці характеризує стійкість до диференціального криптографічного аналізу;

- складність представлення алгебри в полі $GF(2^8)$, важливо для стійкості до атак алгебри;

- простота опису.

У [7, 28, 36] наведено декілька методів конструювання таблиць *S*-блоків, що задовольняють перші три критерії. Для оберненої байтової таблиці замін максимальна кореляція вхідних/вихідних даних може бути мінімізована до 2^{-3}, а максимальна величина *xor* таблиці — мінімізована до *4* (що відповідає коефіцієнту диференціального проникнення 2^{-6}).

Для формування таблиці замін було вибрано відображення $s \rightarrow s^{-1}$ (мультиплікативна інверсія) у полі $GF(2^8)$. Проте очевидно, що вибране відображення має занадто просте представлення алгебри. Це робить можливим використання маніпуляцій алгебри в таких атаках на шифр, як, наприклад, атака методом інтерполяції [36]. Тому результат перетворення зазнає додаткової (цілком оберненої) зміни.

Ця зміна не зменшує властивостей *S*-блока відносно перших трьох критеріїв, але дозволяє задовольнити вимогу відповідності

таблиці замін четвертому критерію вибору. Було вибрано таку зміну, яка сама по собі дуже проста в описі, але при цьому, у поєднанні зі знаходженням мультиплікативно оберненого елемента в полі $GF(2^8)$, має складне представлення алгебри.

Воно може бути представлене як перемножування багаточленів за модулем $x^8 + 1$ із подальшим складанням:

$$\begin{aligned} s(x) &= (a(x)\cdot d(x) + b(x)) \bmod (x^8 + 1) = \\ &= ((x^7 + x^6 + x^5 + x^4 + 1)\cdot d(x)\cdot + \\ &+ (x^6 + x^5 + x + 1)) \bmod (x^8 + 1). \end{aligned} \tag{9.15}$$

Тут $d(x)$ — перетворюваний байт у вигляді багаточлена, $s(x)$ — результуючий байт. Модуль $x^8 + 1$ був вибраний як найпростіший із можливих. Співмножник $x^7 + x^6 + x^5 + x^4 + 1$ ($a(x)$ у виразі (9.15)) був обраний із набору багаточленів, взаємно простих із модулем $x^8 + 1$, як той, що має найбільш просте представлення. А багаточлен $x^6 + x^5 + x + 1$ ($b(x)$ у виразі (9.15)) був обраний так, щоб отримана в результаті таблиця замін не мала точок симетрії ($S(d(x)) = d(x)$) і точок оберненої симетрії ($S(d(x)) = d^{-1}(x)$).

Іншими словами, суть перетворення (9.15) може бути описана рівняннями:

$$\begin{aligned} s_i = d_i \oplus d_{(i+7) \bmod 8} \oplus d_{(i+6) \bmod 8} \oplus d_{(i+5) \bmod 8} \oplus \\ \oplus d_{(i+4) \bmod 8} \oplus b_i, \end{aligned} \tag{9.16}$$

де d_i і s_i — відповідно початкове й перетворене значення i-го біта, $i = 0,1,...,7$; $b_0 = b_1 = b_5 = b_6 = 1$, $b_2 = b_3 = b_4 = b_7 = 0$.

Застосування перетворення (9.16) можна описати в матричному вигляді таким чином:

$$S(x) = A(x)\cdot D(x) \oplus B(x), \tag{9.17}$$

де $S(x)$, $D(x)$ і $B(x)$ — вектор-стовпці розмірністю 8×1; $A(x)$ — матриця розмірністю 8×8.

Вираз (9.17) можна представити в розгорнутому вигляді:

$$s(x)=\begin{vmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix} \cdot \begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{vmatrix} \oplus \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix}. \qquad (9.18)$$

Як видно з виразу (9.18) матриця *A(x)* дорівнює:

$$A(x)=\begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}. \qquad (9.19)$$

Матриці, подібні до матриці *A(x)*, називаються *матрицею Тепліца* (теплицевими матрицями) або діагонально-постійними матрицями. У лінійній алгебрі, матриця Тепліца, названа на честь німецького математика *Отто Тепліца*, — це матриця, в якій на усіх діагоналях (включаючи головну), паралельних головній, стоять рівні елементи.

Примітка. Існують також інші *S*-блоки, що задовольняють вищеперелічені критерії. Тому в разі підозри на існування в цій таблиці "*чорного ходу*" вона може бути легко замінена на іншу. Більше того, структура шифру й вибрана кількість раундів дозволяють застосувати таку таблицю замін, яка не задовольняє критерії *2* й *3*. Навіть середня в цьому відношенні таблиця замін буде забезпечувати стійкість до диференціального й лінійного криптографічного аналізу.

Приклад 9.3. Нехай деякий байт стану на вході функції *SubBytes()* дорівнює *{2a}*. У поліноміальному уявленні це

$s(t) = x^5 + + x^3 + x$ (ω^{166}). Необхідно визначити для нього мультиплікативно обернене значення в полі $GF(2^8)$.

Розв'язання

У полі $GF(2^8)$ для знаходження мультиплікативно оберненого елемента повинна виконуватися рівність відносно примітивних елементів: $\omega^i \cdot \omega^j = 1$ ($i + j = 255$). За $i = 166$ (ω^{166}) j буде дорівнювати 89 (ω^{89}). У такому випадку примітивному елементу ω^{89} [9] буде відповідати поліном $s^{-1}(t) = x^7 + x^4 + x^3$, який є мультиплікативно оберненим до полінома $s(t) = x^5 + x^3 + x$. Для перевірки цього твердження помножимо поліноми для $s(t)$ і $s^{-1}(t)$ у полі $GF(2^8)$ з незвідним багаточленом $p(x) = x^8 + x^4 + x^3 + x + 1$. Якщо добуток буде дорівнювати одиниці, то $s(t)$ і $s^{-1}(t)$ дійсне є мультиплікативно оберненими.

$$s(x)\cdot s^{-1}(x) \bmod p(x) = (x^7 + x^4 + x^3)\cdot(x^5 + x^3 + x) = \\ = (x^{12} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4) \bmod p(x) = 1. \qquad (9.20)$$

Отже, мультиплікативно оберненим значенням байта стану *{ 2a }* на вході функції *SubBytes()* у полі $GF(2^8)$ буде дорівнювати байт *{ 98 }* (еквівалент полінома $s^{-1}(x) = x^7 + x^4 + x^3$).

Приклад 9.4. Нехай деякий байт стану отриманий внаслідок обчислення мультиплікативно оберненого значення дорівнює *{ 98 }*. У поліноміальному уявленні це $d(x) = s^{-1}(x) = x^7 + x^4 + x^3$ (див. приклад 9.1). Необхідно визначити для нього афінне перетворення в полі $GF(2^8)$.

Розв'язання

Підставляючи значення полінома $d(x)$ у вираз (9.15) або (9.18), роблячи перетворення і обчислюючи, отримаємо

$$S(x) = \{ s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0 \} = \{ 1, 1, 1, 0, 0, 1, 0, 1 \}. \qquad (9.21)$$

Роблячи представлення (9.21), отримаємо $s(x)$:

- у двійковій системі — $s(x) =$*{ 11100101 }*;
- у шістнадцятковій системі — $s(x) =$*{ e5 }*.

Якщо розрахувати мультиплікативні інверсії й афінні перетворення для всіх можливих станів (байтів) від *{ 00 }* до *{ ff }*, то виходить набір даних, які можна звести в табл. 9.3.

Таблиця 9.3

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*	*fo*	*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*	*ao*	*52*	*3b*	*d6*	*b3*	*29*	*e3*	*2f*	*84*
5	*53*	*d1*	*00*	*ed*	*20*	*fc*	*b1*	*5b*	*6a*	*cb*	*be*	*39*	*4a*	*4c*	*58*	*cf*
6	*do*	*ef*	*aa*	*fb*	*43*	*4d*	*33*	*85*	*45*	*f9*	*02*	*f7*	*50*	*3c*	*9f*	*a8*
7	*51*	*a3*	*40*	*8f*	*92*	*9d*	*38*	*f5*	*bc*	*b6*	*da*	*21*	*10*	*ff*	*f3*	*d2*
8	*cd*	*0c*	*13*	*ec*	*5f*	*97*	*44*	*17*	*c4*	*a7*	*7e*	*3d*	*64*	*5d*	*19*	*73*
9	*60*	*81*	*4f*	*dc*	*22*	*2a*	*90*	*88*	*46*	*ee*	*b8*	*14*	*de*	*5e*	*0b*	*db*
a	*e0*	*32*	*3a*	*0a*	*49*	*06*	*24*	*5c*	*c2*	*d3*	*ac*	*62*	*91*	*95*	*e4*	*79*
b	*e7*	*cb*	*37*	*6d*	*8d*	*d5*	*4e*	*a9*	*6c*	*56*	*f4*	*ea*	*65*	*7a*	*ae*	*08*
c	*ba*	*78*	*25*	*2e*	*1c*	*a6*	*b4*	*c6*	*e8*	*dd*	*74*	*if*	*4b*	*bd*	*8b*	*8a*
d	*70*	*3e*	*b5*	*66*	*48*	*03*	*f6*	*oe*	*61*	*35*	*57*	*b9*	*86*	*c1*	*1d*	*9e*
e	*e1*	*f8*	*98*	*11*	*69*	*d9*	*8e*	*94*	*9b*	*1e*	*87*	*e9*	*ce*	*55*	*28*	*df*
f	*8c*	*a1*	*89*	*0d*	*bf*	*e6*	*42*	*68*	*41*	*99*	*2d*	*0f*	*b0*	*54*	*bb*	*16*

Перетворення *SubBytes(state)* зводиться до виконання для кожного байта *s* у *state* операції $s \leftarrow S(s)$. Щоб застосувати підстановку до байта, необхідно інтерпретувати байт як дві шістнадцяткові цифри. Ліва цифра визначає рядок, а права — колонку в табл. 9.3. На перетині рядка й колонки, позначених цими шістнадцятковими цифрами, знаходиться новий байт. Рис. 9.10 ілюструє цю ідею.

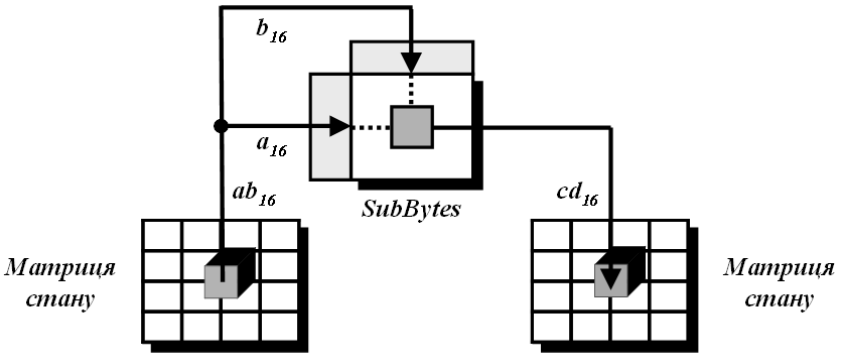


Рис. 9.10. Пояснення ідеї табличної заміни в перетворенні *SubBytes*()

Наприклад, результат перетворення байта *{ 53 }* знаходиться на перетині *5*-го рядка й *3*-го стовпця та дорівнює *{ ed }*.

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 перетворень байт у байт.

Приклад 9.5. Нехай на вхід функції *SubBytes()* алгоритму *AES* (*Nb = 4*) надходить масив стану, який дорівнює:

$$State = 193de3bea0f4e22b9ac68d2ae9f84808_{16}.$$

Якщо зробити табличну заміну кожного байта (див. табл. 9.3), то вийде (рис. 9.11):

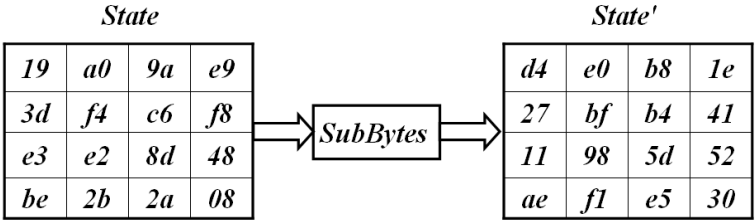


Рис. 9.11. Результат виконання табличної заміни кожного байта масиву стану

Перетворення *SubBytes()* забезпечує ефект перемішування. Наприклад, два байти, $5a_{16}$ і $5b_{16}$, які відрізняються тільки одним бітом (крайній правий біт), перетворені в be_{16} і 39_{16} відповідно, які відрізняються чотирма бітами.

Процедура *InvSubBytes()* (інверсна заміна байтів) реалізує шар нелінійного перетворення та є інверсною нелінійною заміною байтів, що виконується незалежно з кожним байтом стану. Таблиці заміни *S*-блока процедури *InvSubBytes()* є такими, що інвертуються, і побудовані з композиції таких двох перетворень вхідного байта:

1. Застосування інверсії до афінного перетворення (9.14) над $GF(2^8)$ визначеного таким чином [1, 7, 36]:

$$d^{-1}(x) \cdot = (s(x) + b(x)) \cdot a^{-1}(x) \bmod (x^8 + 1), \qquad (9.22)$$

де $a^{-1}(x) = (x^7 + x^6 + x^5 + x^4 + 1)^{-1} \bmod (x^8 + 1) = x^7 + x^5 + x^2$.

Іншими словами, суть перетворення (9.22) може бути описана рівняннями:

$$d_i = c_{(i+7) \bmod 8} \oplus c_{(i+5) \bmod 8} \oplus c_{(i+2) \bmod 8}, \qquad (9.23)$$

де $c_i = s_i \oplus b_i$; $b_0 = b_1 = b_5 = b_6 = 1$, $b_2 = b_3 = b_4 = b_7 = 0$, s_i і d_i відповідно початкове й перетворене значення i-го біта, $i = 0,1,\ldots,7$.

Застосування перетворення (9.22) можна описати в матричному вигляді таким чином:

$$d^{-1}(x) = \begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{vmatrix} \cdot \left(\begin{vmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{vmatrix} \oplus \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix} \right). \qquad (9.24)$$

2. Отримання мультиплікативно оберненого елемента відносно множення в полі $GF(2^8)$, тобто розв'язання рівняння

$$d(x) \cdot d^{-1}(x) \bmod p(x) = 1 \qquad (9.25)$$

відносно елемента $d^{-1}(x)$ (нульовий елемент $\{00\}$ також переходить сам у себе).

Приклад 9.6. Нехай деякий байт стану на вході функції *InvSubBytes()* дорівнює $\{e5\}$. У поліноміальному уявленні це $s(t) = x^7 + x^6 + x^5 + x^2 + 1$. Необхідно визначити для нього інверсне афінне перетворення в полі $GF(2^8)$.

Розв'язання

Підставляючи значення полінома $s(x)$ у вираз (9.23) або (9.24), роблячи перетворення й обчислюючи, отримаємо

$$d^{-1}(x) = \{ d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0 \} = \{ 1,0,0,1,1,0,0,0 \}. \qquad (9.26)$$

Роблячи представлення (9.26), отримаємо $d^{-1}(x)$:

- у двійковій системі – $d^{-1}(x) = \{ 10011000 \}$;
- у шістнадцятковій системі — $d^{-1}(x) = \{ 98 \}$.

Приклад 9.7. Нехай деякий байт стану отриманий внаслідок обчислення інверсного афінного перетворення функції *InvSubBytes()* дорівнює *{ 98 }*. У поліноміальному уявленні це $d^{-1}(x) = x^7 + x^4 + x^3$ (див. приклад 9.6). Необхідно визначити для нього мультиплікативно обернене значення в полі $GF(2^8)$.

Розв'язання

У полі $GF(2^8)$ для знаходження мультиплікативно оберненого елемента повинна виконуватися рівність відносно примітивних елементів: $\omega^i \cdot \omega^j = 1$ ($i + j = 255$). За $i = 89$ (ω^{89}) j буде дорівнювати 166 (ω^{166}). У такому випадку примітивному елементу ω^{166} [7] буде відповідати поліном $s(x) = x^5 + x^3 + x$, який є мультиплікативно оберненим до полінома $d^{-1}(x) = x^7 + x^4 + x^3$.

Отже, мультиплікативно оберненим значенням байта стану *{ 98 }* на вході функції *InvSubBytes()* в полі $GF(2^8)$ буде дорівнювати байт *{ 2a }* (еквівалент полінома $s(x) = x^5 + x^3 + x$).

Якщо розрахувати інверсії афінних та мультиплікативно обернених перетворень для всіх можливих станів (байтів) від *{ 00 }* до *{ ff }*, то виходить набір даних, які можна звести в табл. 9.4. Тоді перетворення *InvSubBytes()* зводиться до виконання для кожного байта s у *state* операції $s \leftarrow S(s)$.

Табл. 9.3 і 9.4 взаємообернені. Наприклад, якщо виконати перетворення над байтом *{ 2d }* за допомогою *SubBytes()* (див. табл. 9.3), то отримаємо *{ d8 }*. Якщо тепер застосувати до байта *{ d8 }* перетворення *InvSubBytes()* (табл. 9.4), то отримаємо *{ 2d }*.

Таблиця 9.4

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*

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
9	*96*	*ac*	*74*	*22*	*e7*	*ad*	*35*	*85*	*e2*	*f9*	*37*	*e8*	*1c*	*75*	*de*	*6e*
a	*47*	*e1*	*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*	*d*	*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 9.8. Рис. 9.12 показує, як матриця станів перетворюється з використанням *SubBytes()* та *InvSubBytes()*.

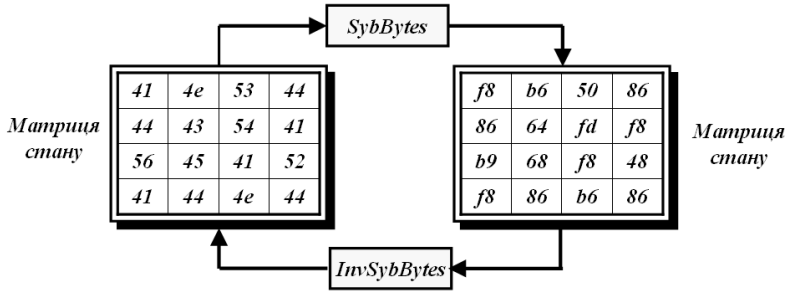


Рис. 9.12. Перетворення *SubBytes()* та *InvSubBytes()* за прикладом 9.8

Рисунок також показує, що *InvSubBytes()* однозначно відтворює оригінал. Зауважимо, що якщо два байти мають однакове значення, то вони перетворяться однаково. Наприклад, два байти $4e_{16}$ і $4e_{16}$ у лівій матриці станів перетворюються в $b6_{16}$ і $b6_{16}$ у правій матриці станів і навпаки. Причина в тому, що кожний байт використовує ту саму таблицю перетворень.

Хоча можна використовувати таблиці 9.3 і 9.4 для перестановки кожного байта, *AES* дає визначення алгебраїчним перетворенням на основі поля $GF(2^8)$ за допомогою поліномів, як це показано на рис. 9.13.

У процедурі *SubBytes()* байт (двійковий рядок на *8* бітів) знаходиться в полі $GF(2^8)$ за допомогою незвідного полінома $p(x) = x^8 + x^4 + x^3 + x + 1$. Зауважимо, що байт 00_{16} сам є власним оберненням. Обернений байт потім інтерпретується як матриця-стовпець із

наймолодшим бітом угорі та найстаршим бітом унизу. Ця матриця-стовпець множиться на постійну квадратну матрицю *A(x)* і результат, який є матрицею-стовпцем, складається з постійною матрицею-стовпцем *B(x)*, що дає новий байт. Зауважимо, що множення або сума бітів відбувається в $GF(2^8)$. *InvSubBytes()* робить ті самі дії у зворотному порядку.

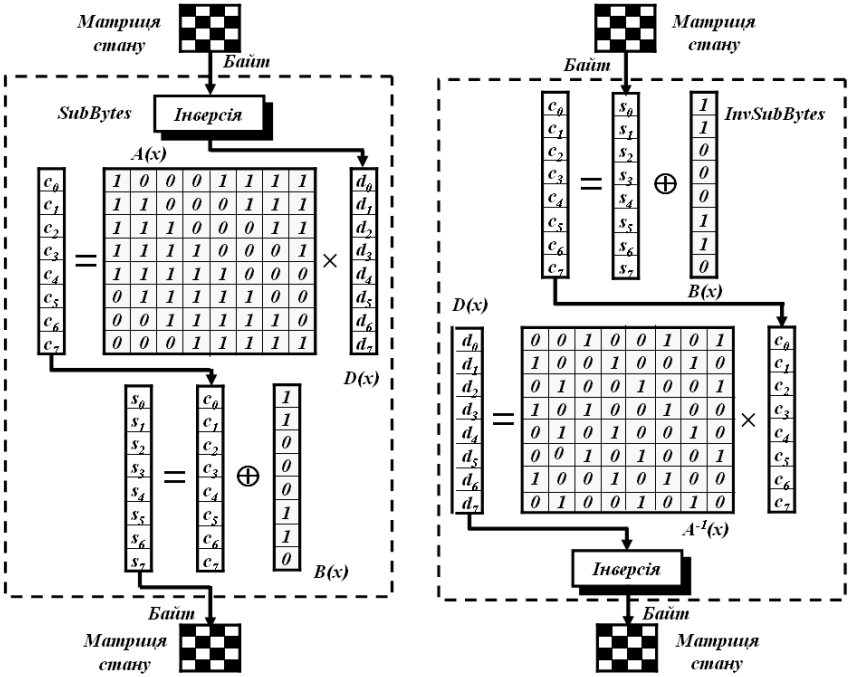


Рис. 9.13. Перетворення *SubBytes()* і *InvSubBytes()*

У процесі зашифрування множення є першою операцією, додавання — другою. У ході розшифрування віднімання (додавання інверсією) є першим, а ділення (множення з інверсією) — другим.

Хоча множення й додавання матриць у процедурі *SubBytes()* і *InvSubBytes()* — перетворення афінного типу й лінійне, заміна байта його мультиплікативно оберненим значенням в $GF(2^8)$ — нелінійна. Цей крок робить все перетворення нелінійним.

9.5.2. Зсув рядків масиву станів — *ShiftRows()* і *InvShiftRows()*

Процедура *ShiftRows(State)* реалізує шар лінійного перетворення. Вибір із можливих комбінацій зсувів у рядках перетворюваного блока був зроблений на підставі таких критеріїв:

- усі чотири зсуви повинні бути різними для кожного рядка і $C_0 = 0$;
- стійкість до атак, що використовують скорочені диференціали (див. нижче, а також [19]);
- стійкість до атаки *Square* (буде описано нижче, також див. [28]);
- простота.

Для деяких комбінацій значень зсувів атаки з використанням скорочених диференціалів стають ефективні для більш ніж одного раунду.

Для інших комбінацій ефективніші атаки типу *Square*. Із набору комбінацій зсувів, що якнайкраще задовольняють пункти *2* і *3*, було вибрано просту комбінацію.

Таблиця 9.5

Величина зсувів для різної довжини блоків

N_b	C_1	C_2	C_3
4	*1*	*2*	*3*
6	*1*	*2*	*3*
8	*1*	*3*	*4*

Останні три рядки стану циклічно зсуваються вліво на різне число байтів. Рядок *1* зсувається на C_1 байтів, рядок *2* — на C_2 байтів і рядок *3* — на C_3 байтів. Значення зсувів C_1, C_2 і C_3 в *Rijndael* залежать від довжини блока *Nb*. Їх величини наведено в табл. 9.5.

У стандарті *AES*, де визначений єдиний розмір блока ($Nb = 4$), дорівнює *128* бітам, $C_1 = 1$, $C_2 = 2$ і $C_3 = 3$.

Операція зсуву останніх трьох рядків стану позначена як *ShiftRows()*. Рис. 9.14 показує вплив перетворення на рядки масиву стану.

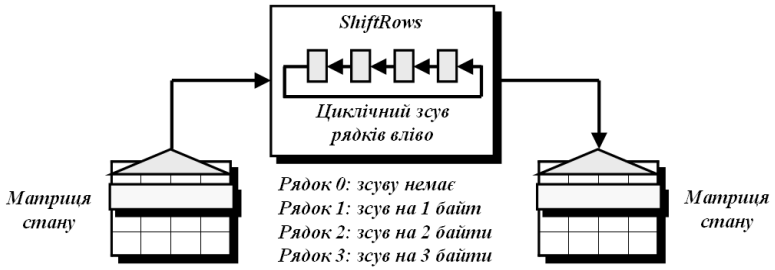


Рис. 9.14. Дія перетворення *ShiftRows()* на рядки масиву стану

Зауважимо, що перетворення *ShiftRows()* працює одночасно тільки з одним рядком.

Перетворення *InvShiftRows (State)* обернене перетворенню *ShiftRows (State)* і діє на кожний рядок r_i у *State* за таким правилом (рис. 9.15): перший рядок *State* не зсувається, тобто $C'_0 = 0$, останні три рядки стану циклічно зсуваються вправо на різне число байтів; другий рядок зсувається на C'_1 байтів, третій — на C'_2 байтів і четвертий — на C'_3 байтів; значення зсувів C'_1, C'_2 і C'_3 в *Rijndael* залежать від довжини блока *Nb*, їх величини наведено в табл. 9.5.

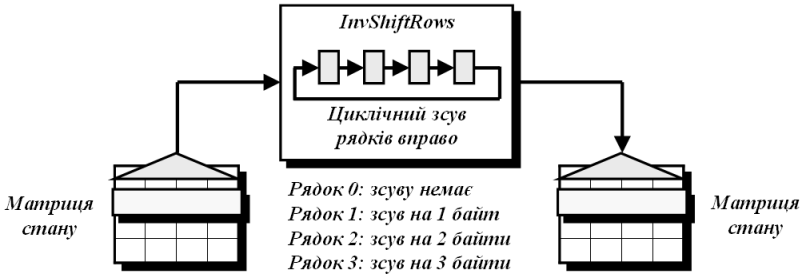


Рис. 9.15. Дія перетворення *InvShiftRows()* на рядки масиву стану

Приклад 9.9. Нехай на вхід функції *ShiftRows()* алгоритму *AES* (*Nb = 4*) надходить масив стану, який дорівнює:

$$State = f886b9f8b664688650fdf8b686f84886_{16}.$$

На рис. 9.16 показано результат виконання зсувів рядків масиву станів уліво за допомогою функції *ShiftRows()* і вправо допомогою функції *InvShiftRows()*.

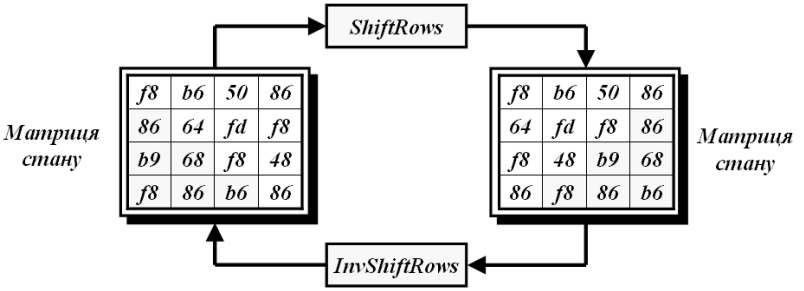


Рис. 9.16. Результат виконання перетворення *ShiftRows()* і *InvShiftRows()* на рядки масиву стану

9.5.3. Перемішування стовпців масиву станів – *MixColumns()* і *InvMixColumns()*

Перетворення *MixColumns (State)* реалізує також шар лінійного перетворення й діє на кожний стовпець s_i *(i = 0,1,...,3)* у блоці *State*, тобто на кожне машинне слово, за правилом

$$s'_i(x) = g(x) \otimes s_i(x) \bmod (x^4 + 1). \qquad (9.27)$$

У цьому перетворенні стовпці стану (чотири байти) є як багаточлени над $GF(2^8)$ і множаться за модулем $m(x) = x^4 + 1$ на багаточлен $g(x)$, що виглядає таким чином [28]:

$$g(x) = \{03\} \cdot x^3 + \{01\} \cdot x^2 + \{01\} \cdot x + \{02\}. \qquad (9.28)$$

Елементи поля $GF(2^8)$ записуються або як бітовий вектор, або як байт. Зокрема, байт $\{03\}$ — це елемент поля $GF(2^8)$, що представлений багаточленом $x + 1$ за модулем $m(x)$.

Оскільки множення на багаточлен — лінійна операція, її можна представити у вигляді дії матриці:

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix}, \qquad (9.29)$$

де c — номер стовпця масиву *State* ($c = 0,1,...,3$).

Як результат такого множення байти стовпця $s_{0,c}$, $s_{1,c}$, $s_{2,c}$ і $s_{3,c}$ замінюються відповідно на байти:

$$\begin{aligned} s'_{0,C} &= \{02\} \bullet s_{0,C} \oplus \{03\} \bullet s_{1,C} \oplus s_{2,C} \oplus s_{3,C}; \\ s'_{1,C} &= s_{0,C} \oplus \{02\} \bullet s_{1,C} \oplus \{03\} \bullet s_{2,C} \oplus \bullet s_{3,C}; \\ s'_{2,C} &= s_{0,C} \oplus s_{1,C} \oplus \{02\} \bullet s_{2,C} \oplus \{03\} \bullet s_{3,C}; \\ s'_{3,C} &= \{03\} \bullet s_{0,C} \oplus s_{1,C} \oplus s_{2,C} \oplus \{02\} \bullet s_{3,C}. \end{aligned} \qquad (9.30)$$

Застосування цієї операції до всіх чотирьох стовпців стану позначене як *MixColumns(State)*. Рис. 9.17 демонструє застосування перетворення *MixColumns()* до стовпців масиву стану.

Функцію розсіювання *MixColumns()* було вибрано з можливих лінійних перетворень *4 bytes → 4 bytes* (чотири байти стовпця масиву стану в чотири байти) виходячи з таких критеріїв вибору:

- оборотність;

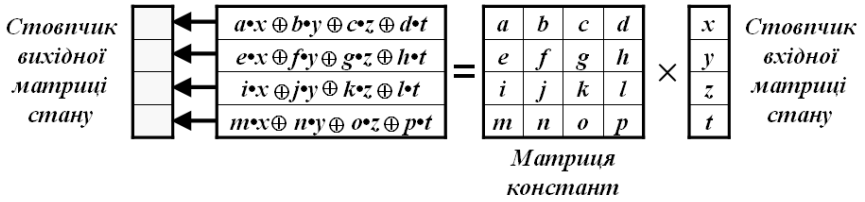


Рис. 9.17. Дія *MixColumns()* — перетворення до стовпців масиву стану

- лінійність у полі $GF(2^4)$; ця властивість функції *MixColumns()* дозволяє застосовувати алгоритм прямого розшифрування;

- досить сильне розсіювання;

- швидкість виконання на *8*-розрядних процесорах;

- симетричність, тобто *MixColumns()* повинна працювати з усіма даними однаково;

- простота опису.

Критерії *2*, *5* і *6* обумовили вибір множення багаточленів за модулем $x^4 + 1$. Критерії *1*, *3* і *4* визначили вибір коефіцієнтів співмножників. Критерій *4* визначав вибір коефіцієнтів (як перевагу) — *{ 00 }*, *{ 01 }*, *{ 02 }*, *{ 03 }*, ... Значення *{ 00 }* не припускає взагалі ніякого перетворення, *{ 01 }* — не потрібно робити множення, *{ 02 }* — множення може бути виконане за допомогою функції *xtime()* і *{ 03 }* — за допомогою функції *xtime()* і подальшого *xor*.

Третій критерій визначає складніші умови вибору коефіцієнтів. Стратегія розробників шифру враховувала такі вимоги до перетворення *MixColumns()*: *Z* повинне бути лінійним перетворенням, що виконується над векторами байтів; байтовою вагою вектора повинна бути кількість ненульових байтів у ньому. Тоді коефіцієнтом поширення лінійного перетворення, що характеризує його силу розсіювання, назвемо величину

$$min_{a \neq 0} \{ Z(a) + Z(F(a)) \},$$

де *Z(a)* — байтова вага.

Байт, що не дорівнює нулю, назвемо *активним байтом*. *MixColumns()* за одного активного байта вхідного *32*-розрядного

масиву *State* на виході буде максимум *4* активних байти, оскільки ця функція перетворить стовпці окремо. Отже, верхня межа коефіцієнта поширення дорівнює *5*. Коефіцієнти *{ 01 }*, *{ 02 }* і *{ 03 }* були підібрані так, щоб отримати це максимально можливе значення коефіцієнта поширення.

Це значення дорівнює *5* і вказує на те, що різниця у вхідних даних в один байт розсіюється (поширюється) на чотири байти вихідних даних, різниця у два байти вхідних даних відобразиться, як мінімум, на трьох байтах вихідних даних. Більше того, можна сказати, що лінійна залежність між бітами у вхідних і вихідних даних завжди охоплює біти як мінімум п'яти різних байтів на вході й виході функції *MixColumns()*.

Приклад 9.10. Нехай на вхід функції *MixColumns()* алгоритму *AES* (*Nb = 4*) надходить масив стану, який дорівнює:

$$State = d4bf5d30e0b452aeb84111f11e2798e5_{16},$$

при цьому

$$\begin{array}{llll} s_{0,0} = d4; & s_{0,1} = e0; & s_{0,2} = b8; & s_{0,3} = 1e; \\ s_{1,0} = bf; & s_{1,1} = b4; & s_{1,2} = 41; & s_{1,3} = 27; \\ s_{2,0} = 5d; & s_{2,1} = 52; & s_{2,2} = 11; & s_{2,3} = 98; \\ s_{3,0} = 30; & s_{3,1} = ae; & s_{3,2} = f1; & s_{3,3} = e5. \end{array}$$

Як відомо масив стану на виході буде визначатися виразом (9.30). Визначимо стовпець масиву стану на виході для випадку *c = 0*, використовуючи вираз (9.30). У даному випадку

$$s'_{0,0} = \{02\} \bullet s_{0,0} \oplus \{03\} \bullet s_{1,0} \oplus s_{2,0} \oplus s_{3,0} =$$

$$= \{02\} \bullet \{d4\} \oplus \{03\} \bullet \{bf\} \oplus \{5d\} \oplus \{30\} =$$

$$= \{02\} \bullet \{d4\} \oplus \{02\} \bullet \{bf\} \oplus \{bf\} \oplus \{5d\} \oplus \{30\}.$$

Представимо значення *{ d4 }*, *{ bf }*, *{ 5d }* і *{ 30 }* у вигляді поліномів:

$$\{d4\} \rightarrow x^7 + x^6 + x^4 + x^2,$$

$$\{bf\} \rightarrow x^7 + x^5 + x^4 + x^3 + x^2 + x + 1,$$

$$\{5d\} \rightarrow x^6 + x^4 + x^3 + x^2 + 1,$$

$$\{30\} \rightarrow x^5 + x^4.$$

Вирахуємо $\{02\}\bullet\{d4\}$:

$$\{02\}\bullet\{d4\} \rightarrow x \cdot (x^7 + x^6 + x^4 + x^2) \bmod p(x) =$$
$$= (x^8 + x^7 + x^5 + x^3) \bmod (x^8 + x^4 + x^3 + x + 1) =$$
$$= x^7 + x^5 + x^4 + x + 1.$$

Вирахуємо $\{02\}\bullet\{bf\}$:

$$\{02\}\bullet\{bf\} \rightarrow x \cdot (x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \bmod p(x) =$$
$$= (x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1) =$$
$$= x^6 + x^5 + x^2 + 1.$$

Вирахуємо елемент $s'_{0,0}$:

$$s'_{0,0} = (x^7 + x^5 + x^4 + x + 1) \oplus (x^6 + x^5 + x^2 + 1) \oplus$$
$$\oplus (x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \oplus (x^6 + x^4 + x^3 + x^2 + 1) \oplus$$
$$\oplus (x^5 + x^4) = x^2 \rightarrow \{04\}.$$

Рахуючи аналогічно елементи $s'_{1,0}$, $s'_{2,0}$ і $s'_{3,0}$, отримаємо:

$$s'_{1,0} = x^6 + x^5 + x^2 + x \rightarrow \{66\};$$
$$s'_{2,0} = x^7 + 1 \rightarrow \{81\};$$
$$s'_{3,0} = x^7 + x^6 + x^5 + x^2 + 1 \rightarrow \{e5\}.$$

Представляючи значення стовпців масиву стану за $c = 1, 2$ і 3 у вигляді поліномів і роблячи аналогічні перетворення, отримаємо масив стану на виході функції *MixColumns()* (рис. 9.18).

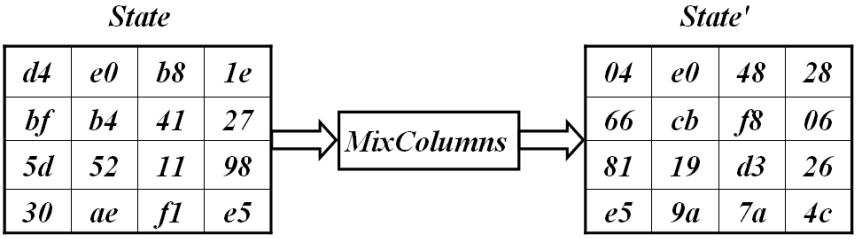


Рис. 9.18. Результат виконання перетворення стовпців масиву стану

У перетворенні *MixColumns()* матриця коефіцієнтів, що входить у (9.29), невироджена над $GF(2^8)$, тому операція *MixColumns()* є оберненої. Обернена до неї дія реалізується матрицею, оберненою до матриці, вказаної в (9.29).

У перетворенні *InvMixColumns()* стовпці стану розглядаються як багаточлени над $GF(2^8)$ і множаться за модулем $x^4 + 1$ на багаточлен

$$s'(x) = (s(x) \otimes g^{-1}(x)) \bmod (x^4 + 1),$$

де $g^{-1}(x) = \{0b\} \cdot x^3 + \{0d\} \cdot x^2 + \{09\} \cdot x + \{0e\}$ багаточлен, який виходить розв'язанням рівняння

$$g(x) \otimes g^{-1}(x) \bmod (x^4 + 1) = 1 \qquad (9.31)$$

відносно $g^{-1}(x)$ за $g(x) = \{03\} \cdot x^3 + \{01\} \cdot x^2 + \{01\} \cdot x + \{02\}$.

Це може бути представлено в матричному вигляді таким чином:

$$\begin{vmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{vmatrix} = \begin{vmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{vmatrix} \begin{vmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{vmatrix},$$

де c — номер стовпця масиву *State* ($c = 0,1,\ldots, 3$).

Внаслідок такого множення байти стовпця $s_{o,c}$, $s_{1,c}$, $s_{2,c}$ і $s_{3,c}$ замінюються відповідно на байти:

$$s'_{0C} = (\{0e\} \bullet s_{0C}) \oplus (\{0b\} \bullet s_{1C}) \oplus (\{0d\} \bullet s_{2C}) \oplus (\{09\} \bullet s_{3C});$$
$$s'_{1C} = (\{09\} \bullet s_{0C}) \oplus (\{0e\} \bullet s_{1C}) \oplus (\{0b\} \bullet s_{2C}) \oplus (\{0d\} \bullet s_{3C});$$
$$s'_{2C} = (\{0d\} \bullet s_{0C}) \oplus (\{09\} \bullet s_{1C}) \oplus (\{0e\} \bullet s_{2C}) \oplus (\{0b\} \bullet s_{3C});$$
$$s'_{3C} = (\{0b\} \bullet s_{3C}) \oplus (\{0d\} \bullet s_{0C}) \oplus (\{09\} \bullet s_{1C}) \oplus (\{0e\} \bullet s_{2C}).$$

Рис. 9.19 показує матриці констант, що використовуються для перетворень *MixColumns()* і *InvMixColumns()*. Ці дві матриці обернені одна одній, коли елементи інтерпретуються як слова з *8* бітів (або поліноми) з коефіцієнтами в $GF(2^8)$ [7, 28, 36].

Перетворення *MixColumns()* працює на рівні стовпця; воно перетворює кожний стовпець матриці станів у новий стовпець. Це перетворення — фактично матричне множення стовпця матриці станів і квадратної матриці констант. Байти в стовпці матриці станів

і в матриці констант інтерпретуються як слова по *8* бітів (або поліномами) з коефіцієнтами в $GF(2^8)$. Множення байтів виконується в $GF(2^8)$ за модулем $p(x) = x^8 + x^4 + x^3 + x + 1$. Додавання — це застосування операції *xor* до слів по *8* бітів.

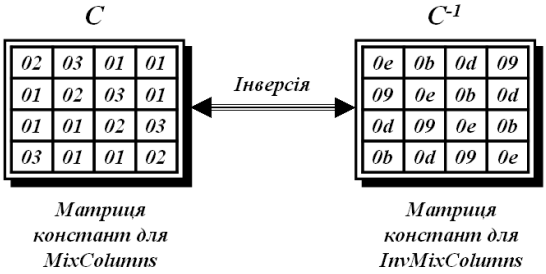


Рис. 9.19. Матриці констант, що використовують перетворення *MixColumns()* і *InvMixColumns()*

Перетворення *InvMixColumns()* схоже на *MixColumns()*-перетворення. Рис. 9.20 показує принцип перетворення з використанням *MixColumns()* або *InvMixColumns()*.

Приклад 9.11. Нехай на вхід перетворення *MixColumns()* надходить масив стану, який дорівнює:

$$State = 63f27dd4c963d4fafe26c96330f2c982_{16}.$$

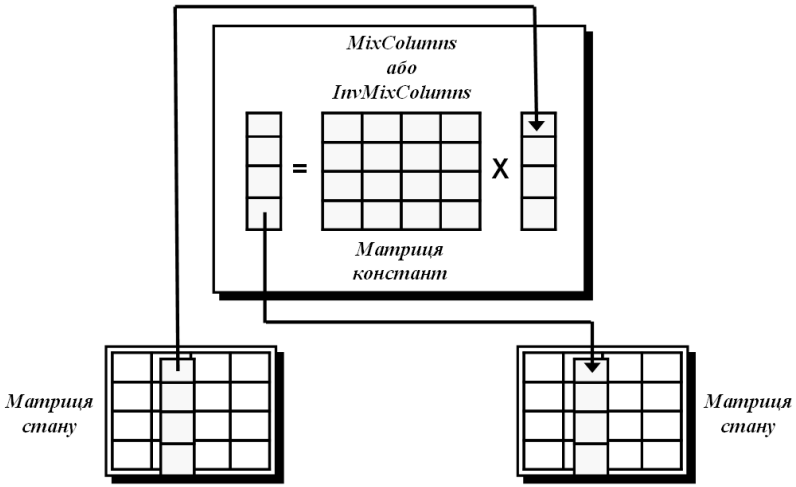


Рис. 9.20. Принцип перетворення з використанням *MixColumns()* або *InvMixColumns()*

Рис. 9.21 показує, як масив стану перетвориться, якщо використовувати перетворення *MixColumns()*. Рисунок також показує, що перетворення *InvMixColumns()* створює первісне значення.

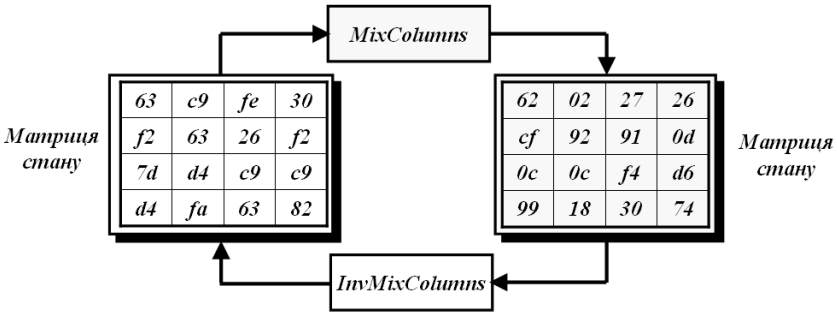


Рис. 9.21. Перетворення *MixColumns()* і *InvMixColumns()* для прикладу 9.11

Зауважимо, що байти, які рівні між собою в старій матриці станів, більше не рівні в новій матриці станів. Наприклад, два байти *f2* у другому рядку змінені на *cf* і *0d*.

9.5.4. Додавання раундового ключа до масиву станів — *AddRoundKey(State, RoundKey)*

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, зловмисник дуже просто визначить вхідне повідомлення за даним йому зашифрованим. У цьому випадку охоронець таємниці — один єдиний 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 станів. У *MixColumns()* застосовується матричне множення, в *AddRoundKey()* — операції додавання й віднімання. Оскільки додавання й віднімання в кінцевому полі ті самі, то *AddRoundKey()* обернена сама до себе. Рис. 9.22 показує принцип перетворення *AddRoundKey()*.

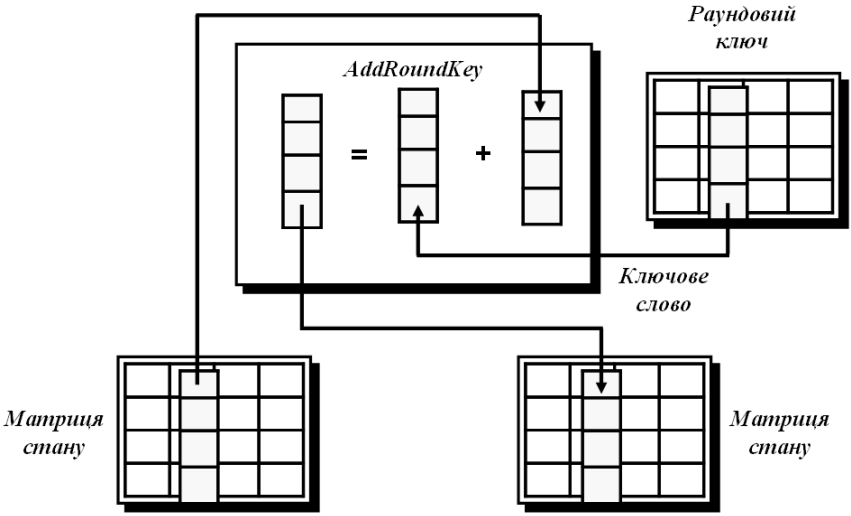


Рис. 9.22. Принцип перетворення масиву стану за допомогою *AddRoundKey()*

Перетворення *AddRoundKey()* реалізує шар складання оброблюваних даних (станом) із раундовим ключем. У цій операції раундовий ключ додається до матриці стану за допомогою простого порозрядного *xor*. Раундовий ключ *W* виробляється з ключа шифрування *K* за допомогою алгоритму вироблення ключів. Довжина раундового ключа (у *32*-розрядних словах) дорівнює довжині блока *Nb*.

Приклад 9.12. Нехай на вхід функції *AddRoundKey()* алгоритму *AES* (*Nb = 4*) надходить:

- масив стану:

$$State = d4bf5d30e0b452aeb84111f11e2798e5_{16};$$

- масив раундового ключа:

$$RoundKey = 046681e5e0cb199a48f8d37a2806264c_{16}.$$

Як відомо, масив стану на виході буде визначатися шляхом порозрядного підсумовування за модулем *2* (виконання операції *xor*) стовпців масиву стану й масиву раундового ключа.

Виконуючи такі дії над стовпцями масиву стану й масиву раундового ключа, отримаємо масив стану на виході функції *AddRoundKey()* (рис. 9.23).

Підстановка, яка робиться перетворенням *SubByte()*, що змінює значення байта, засноване тільки на початковому значенні та вході в таблицю; процес не включає сусідні байти. Можна сказати, що *SubByte()* – внутрішньобайтове перетворення. Перестановка, яка робиться *ShiftRows()* – перетворенням, міняє місцями байти, не переставляючи біти в байтах.

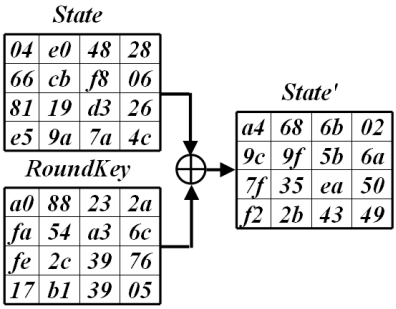


Рис. 9.23. Дія *AddRoundKey()* на байти масиву стану для прикладу 9.12

Можна сказати, що *ShiftRows()* – перетворення обміну байтами. Тепер нам потрібно внутрішньобайтове перетворення, що змінює біти в байтах та 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 байти. 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 залежить від усіх чотирьох елементів старої матриці.

9.6. АЛГОРИТМ РОЗГОРТАННЯ КЛЮЧА ДЛЯ ШИФРУВАННЯ ДАНИХ

Алгоритм розгортання ключа визначає порядок отримання раундових ключів *W* із початкового ключа шифрування *K*. Раундові ключі виходять із ключа шифрування за допомогою алгоритму вироблення ключів. Він містить два компоненти:

- розширення ключа (*Key Expansion*);
- вибір раундового ключа (*Round Key Selection*).

Засадничі принципи алгоритму виглядають так:

- загальна кількість бітів раундових ключів дорівнює довжині блока, помноженому на кількість раундів, плюс *1* (наприклад для

довжини блока *128* бітів і *10* раундів потрібно *1408* бітів раундових ключів);

- ключ шифрування розгортається в розширений ключ (*Expanded Key*);

- раундові ключі беруться з розширеного ключа таким чином: перший раундовий ключ містить перші *Nb* слів, другий — наступні *Nb* слів і так далі.

9.6.1. Розширення ключа (KeyExpansion)

Для того, щоб створити ключ для кожного раунду, *AES* використовує процес ключового розширення. Якщо кількість раундів *Nr*, процедура розширення ключів створює *Nr + 1* раундових ключів на *128* бітів кожний від єдиного *128*-бітового ключа шифрування. Перші раундові ключі використовуються для перетворення перед початком раундів (раунд *0*) із використанням перетворення *AddRoundKey()*; раундові ключі, що залишаються, застосовуються для подальших перетворень *AddRoundKey()* наприкінці кожного раунду.

Процедура розширення ключів створює слово за словом, з яких надалі будуть формуватися раундові ключі. Слово — це масив із чотирьох байтів. Процедура розширення ключів створює $4 \times (Nr + 1)$ слів, які позначаються

$$W = w_0, w_1, w_2 \ldots, w_{4 \times (Nr + 1)}, w_{4 \times (Nr + 1) - 1}.$$

Іншими словами, у версії *AES-128* ($Nr = 10$) буде $W = 44$ слова; у версії *AES-192* ($Nr = 12$) – $W = 52$ слова; і у версії *AES-256* ($Nr = 14$) — $W = 60$ слів. Кожний раундовий ключ складається із чотирьох слів. Табл. 9.6 показує відношення між раундами та словами.

Таблиця 9.6

Слова для кожного раунду для 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}
. . .	. . .	. . .	. . .	. . .
Nr	$w_{4 \times Nr}$	$w_{4 \times Nr+ 1}$	$w_{4 \times Nr+2}$	$w_{4 \times Nr+3}$

Розглянемо процес розширення ключа для версії *AES-128*; процеси для інших версій за винятком невеликих змін – такі самі. Рис. 9.24 показує, як з початкового ключа отримати *44* слова.

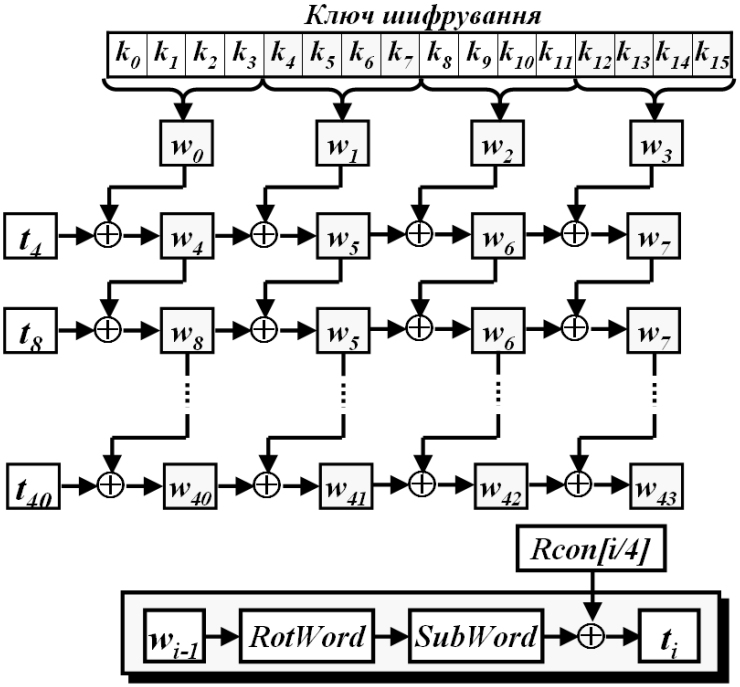


Рис. 9.24. Пояснення процесу розширення ключів в *AES-128*

Процес розширення ключів в *AES-128* такий:

1. Перші чотири слова (w_0, w_1, w_2, w_3) виходять із ключа шифрування. Ключ шифрування 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 шифрування.

2. Решта частини слів w_i від $i = 4$ до $i = 43$ виходить так:

а) якщо $(i \bmod 4) \neq 0$, то $w_i = w_{i-1} \oplus w_{i-4}$ (згідно з рис. 9.24 це означає, що кожне слово отримане з одного лівого й одного верхнього);

б) якщо $(i \bmod 4) = 0$, то $w_i = t_i \oplus w_{i-4}$ (тут t_i — тимчасове слово, результат застосування двох процесів, *SubWord* і *RotWord*, зі словом w_{i-1} і застосування операції *xor* з константою раунду *Rcon[i/4])*.

Іншими словами, маємо

$$t_i = SubWord(RotWord(w_{i-1})) \oplus Rcon[i/4]. \qquad (9.32)$$

RotWord (*Rotate Word*) — процедура, подібна до перетворення *ShiftRows()*, але застосовується тільки до одного стовпця. Процедура приймає слово як масив із чотирьох байтів і зсуває кожний байт угору з конвертацією (циклічний зсув байтів).

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 є завжди нульовими.

Значення константи раунду *Rcon[i/4]* визначається виразом

$$Rcon[i/4] = R_{con}[j] = (2^{j-1}) \bmod p(x). \qquad (9.33)$$

Значення раундових констант *Rcon*, розрахованих за формулою (9.33), наведено в табл. 9.7.

Таблиця 9.7

Значення констант раундів *Rcon (Nk = 4)*

i	*i/Nk*	*Rcon[i/Nk]*	*i*	*i/Nk*	*Rcon[i/Nk]*
4	*1*	*{ 01000000 }*	*24*	*6*	*{ 20000000 }*
8	*2*	*{ 02000000 }*	*28*	*7*	*{ 40000000 }*
12	*3*	*{ 04000000 }*	*32*	*8*	*{ 80000000 }*
16	*4*	*{ 08000000 }*	*36*	*9*	*{ 1b000000 }*
20	*5*	*{ 10000000 }*	*40*	*10*	*{ 36000000 }*

Процедура розширення ключа може використовувати або наведену вище таблицю, коли обчислює слова, або поле $GF(2^8)$, коли обчислює крайні ліві біти динамічно, як це показано нижче.

$$Rcon[i = 4/Nk]|_{Nk=4} = Rcon[1] = 2^{1-1} \bmod p(x) = 01_{16};$$
$$Rcon[i = 8/Nk]|_{Nk=4} = Rcon[2] = 2^{2-1} \bmod p(x) = 02_{16};$$
$$Rcon[i = 12/Nk]|_{Nk=4} = Rcon[3] = 2^{3-1} \bmod p(x) = 04_{16};$$
$$Rcon[i = 16/Nk]|_{Nk=4} = Rcon[4] = 2^{4-1} \bmod p(x) = 08_{16};$$
$$Rcon[i = 20/Nk]|_{Nk=4} = Rcon[5] = 2^{5-1} \bmod p(x) = 10_{16};$$
$$Rcon[i = 24/Nk]|_{Nk=4} = Rcon[6] = 2^{6-1} \bmod p(x) = 20_{16};$$
$$Rcon[i = 28/Nk]|_{Nk=4} = Rcon[7] = 2^{7-1} \bmod p(x) = 40_{16};$$
$$Rcon[i = 32/Nk]|_{Nk=4} = Rcon[8] = 2^{8-1} \bmod p(x) = 80_{16};$$

$$Rcon[i = 36/Nk]|_{Nk = 4} = Rcon[9] = 2^{9-1} \bmod p(x) = 1b_{16};$$
$$Rcon[i = 40/Nk]|_{Nk = 4} = Rcon[10] = 2^{10-1} \bmod p(x) = 36_{16}.$$

Далі наведено псевдокод програми, яка показує процес розгортання ключа (*Key Expansion*).

```
//==============================================
//= Псевдокод функції розгортання ключа KeyExpansion() =
//==============================================
KeyExpansion (byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+l], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb (Nr+1))
        temp = w[i-l]
        if (i mod Nk = 0)
                temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4) temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
//==============================================
```

Примітка. Необхідно враховувати, що з метою повноти опису тут наводиться алгоритм для усіх можливих довжин ключів, на практиці ж повна його реалізація потрібна не завжди.

Перші *Nk* слів містять ключ шифрування. Усі інші слова визначаються рекурсивно зі слів із меншими індексами. Алгоритм вироблення ключів залежить від величини *Nk*.

Приклад 9.13. Нехай на вхід функції *KeyExpansion()* алгоритму *AES* (*Nb = 4*, *Nk = 4*) надходить 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 чотирибайтові слова розширеного ключа: w_4, w_5, w_6 і w_7.

Розв'язання

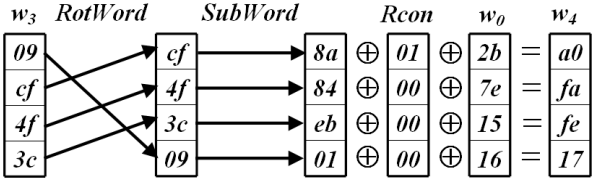
Сформуємо слово w_4. У зв'язку з тим, що

$$i \bmod N_k = 4 \bmod 4 = 0,$$

то слово w_4 буде сформовано з використанням (9.32)

$$w_4 = SubWord(RotWord(w_3)) \oplus w_0 \oplus R_{con}[1]$$

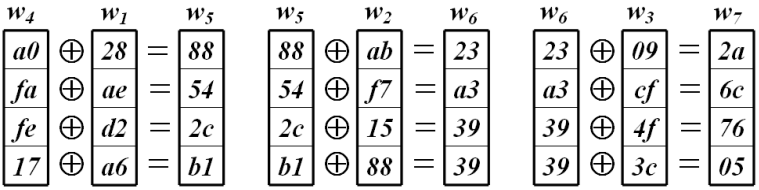
або



Сформуємо слова w_5, w_6 і w_7. Оскільки за $i = 5, 6, 7$ $i \bmod Nk \neq 0$, то слова будуть сформовані з використанням (9.32) таким чином:

$$w_5 = w_4 \oplus w_1; \; w_6 = w_5 \oplus w_2; \; w_7 = w_6 \oplus w_3$$

або



Отже, з урахуванням чотирибайтових слів ключа шифрування K і сформованих чотирибайтових слів: w_4, w_5, w_6 і w_7, слова розширеного ключа набудуть вигляду

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

9.6.2. Вибір раундового ключа (*Round Key Selection*)

Раундовий ключ для *i*-го раунду виходить зі слів масиву раундового ключа від w_{Nb-i} і до $w_{Nb(i+1)-1}$, як показано на рис. 9.25.

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	. . .
Раундовий ключ 0				*Раундовий ключ 1*				*Раундовий ключ 2*				. . .	

Рис. 9.25. Процедури розширення ключа й вибору раундового ключа для *Nk = 4*

Приклад 9.14. Нехай після виконання функції *KeyExpansion()* алгоритму *AES* (*Nb = 4*, *Nk = 4*) отримано такі чотирибайтових слова розширеного ключа:

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}
2b	28	ab	09	a0	88	23	2a	f2	7a	59	73
7e	ae	f7	cf	fa	54	a3	6c	c2	96	35	59
15	d2	15	4f	fe	2c	39	76	95	b9	80	f6
16	a6	88	3c	17	b1	39	05	f2	43	7a	7f

Сформувати раундові ключі для нульового, першого й іншого раундів шифрування даних.

Розв'язання

Раундовий ключ для нульового раунду (K_0) виходить зі слів масиву раундового ключа від $w_{Nb\cdot 0} = w_0$ і до $w_{Nb\cdot(0+1)-1} = w_3$. Раундовий ключ для першого раунду (K_1) виходить зі слів масиву раундового ключа від $w_{Nb\cdot 1} = w_4$ і до $w_{Nb\cdot(1+1)-1} = w_7$. Раундовий ключ для другого раунду (K_2) виходить зі слів масиву раундового ключа від $w_{Nb\cdot 2} = w_8$ і до $w_{Nb\cdot(2+1)-1} = w_{11}$. Отже, раундові ключі з урахуванням його представлення, як показано на рис. 9.25, набудуть вигляду

Раундовий ключ 0

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

Раундовий ключ 1

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

Раундовий ключ 2

f2	7a	59	73
c2	96	35	59
95	b9	80	f6
f2	43	7a	7f

Примітка. Алгоритм вироблення ключів можна здійснювати також без використання масиву w_i. Для реалізацій, в яких істотна вимога до займаної пам'яті, раундові ключі можуть обчислюватися відразу за допомогою використання буфера з *Nk* слів. Розширений ключ повинен завжди виходити з ключа шифрування й ніколи не вказується безпосередньо. Немає ніяких обмежень щодо вибору ключа шифрування.

Алгоритм розгортання ключа повинен забезпечувати стійкість проти таких 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 компресії даних при хешуванні;
- атак "еквівалентних ключів" [28]; необхідною умовою стійкості щодо таких атак є відсутність занадто великих однакових наборів раундових ключів, отриманих із двох різних початкових ключів шифрування.

Процедура розгортання ключа також відіграє важливу роль у виключенні:

- симетрії однораундового перетворення, яке обробляє усі вхідні байти однаково; для її усунення в процедурі розгортання ключа при отриманні кожного першого *32*-розрядного слова раундового ключа використовуються функція *SubWord(RotWord())* і раундова константа;
- міжраундовій симетрії (раундове перетворення однакове для всіх ітерацій циклу перетворення); для її порушення в алгоритм розгортання ключа введені константи, значення яких різні для кожного раунду.

Отже, алгоритм розгортання вибраний відповідно до таких критеріїв:

- оборотність використовуваних перетворень, тобто з будь-яких *Nk* послідовних слів розгорнутого ключа можна однозначно відновити увесь розгорнутий 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 бітів раундового або ключа шифрування вирахувати значну частину інших бітів;
- достатня нелінійність для попередження повного визначення міжбітових залежностей розгорнутого раундового ключа лише на підставі знання таких залежностей у початковому ключі шифрування;
- простота опису.

Було обрано байт-зорієнтовану схему алгоритму, яка може ефективно реалізовуватися на *8*-розрядних процесорах. Застосування *SubBytes()* забезпечує нелінійність перетворення та вимагає невеликого додаткового простору пам'яті на *8*-розрядних процесорах.

Раундові ключі для розшифрування використовуються у зворотному порядку відносно раундових ключів для зашифрування.

9.7. ЗАШИФРУВАННЯ ДАНИХ

Структура шифру *AES* (*Rijndael)* складається (рис. 9.26):

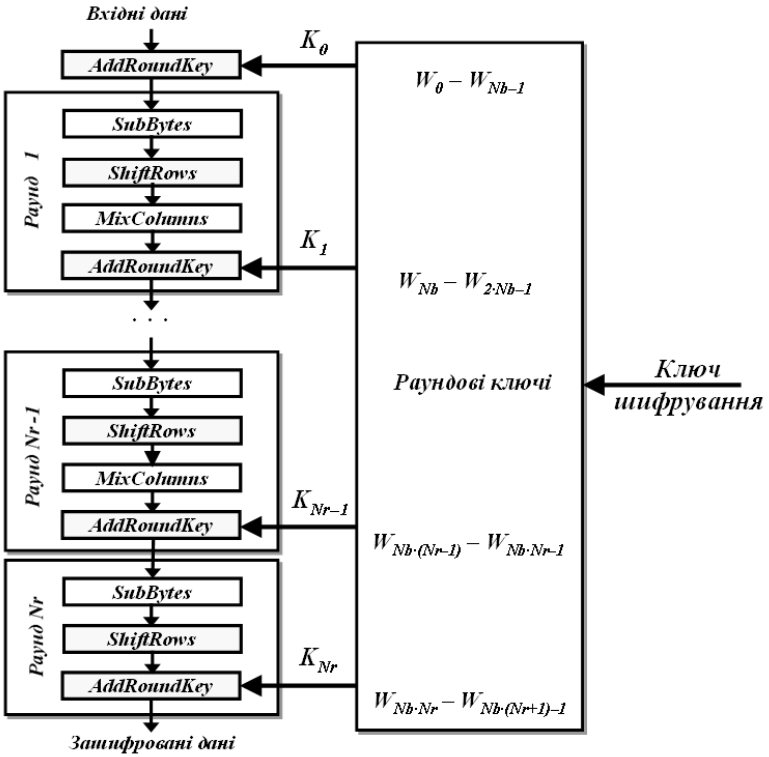


Рис. 9.26. Схема процедури шифрування в криптографічному алгоритмі *AES* (*Rijndael*)

- із початкового додавання раундового ключа;
- із *Nr – 1* раундів;
- із заключного раунду, який відрізняється від попередніх тим, що в ньому відсутня операція *MixColumns()*.

На вхід алгоритму подаються блоки даних *State* (вхідні дані). На рис. 9.27 *in* позначено байти вхідної матриці стану, а *out* – вихідної. У ході перетворень зміст блока змінюється й на виході утворюється зашифроване повідомлення, організоване у вигляді блоків *State*.

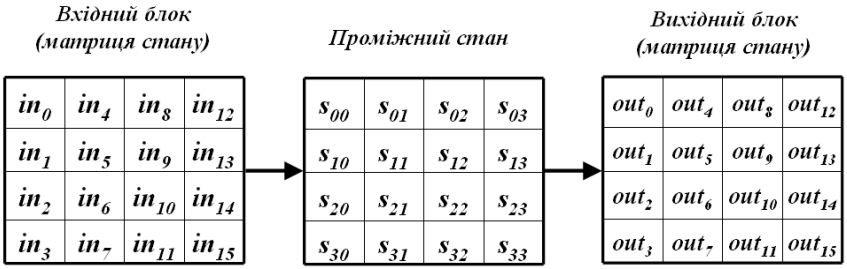


Рис. 9.27. Хід перетворення даних, організованих у вигляді блоків *State*

Перед початком першого раунду відбувається підсумовування за модулем *2* із ключем шифрування, потім перетворення масиву байтів *State* впродовж *10*, *12* або *14* раундів залежно від довжини ключа. Останній раунд дещо відрізняється від попередніх тим, що не використовує функцію перемішування байтів у стовпцях *MixColumns()*.

Рис. 9.28 демонструє також розсіювальні й перемішувальні властивості шифру. Видно, що навіть два раунди забезпечують повне розсіювання й перемішування.

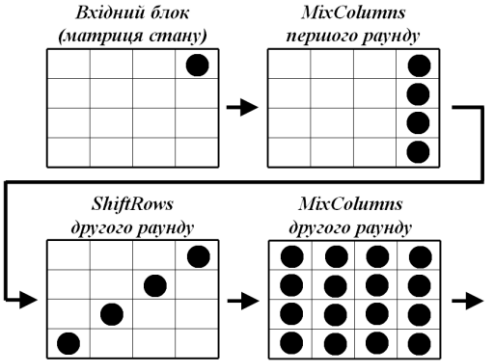


Рис. 9.28. Пояснення розсіювальних властивостей алгоритму *AES* (*Rijndael*): ■ — змінений байт

Нижче наведено псевдокод програми, яка показує процес шифрування даних (*Cipher*).

```
//===========================================
//======== Процедура зашифрування в псевдокоді ======
//===========================================
Cipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
//===========================================
```

9.8. РОЗШИФРУВАННЯ ДАНИХ

9.8.1. Обернене (інверсне) розшифрування даних

Звернемо увагу на те, що в останньому раунді зашифрування відсутнє перетворення *MixColumns()*. На перший погляд це здається дивним рішенням, що погіршує структуру шифру. Але це не так. Позначимо перетворення *SubBytes()*, *ShiftRows()*, *MixColumns()* і *AddRoundKey()* алгоритму зашифрування відповідно через B, R, C і K. Запишемо усю послідовність дій алгоритму зашифрування для випадку довжини ключа *128* бітів у вигляді лінійного ланцюжка:

$$\underbrace{KBRCK}_{\text{1-й раунд}}\underbrace{BRCK}_{\text{2-й раунд}}\ldots\underbrace{BRCK}_{\text{9-й раунд}}\underbrace{BRK}_{\text{10-й раунд}}. \qquad (9.34)$$

Якщо замість *SubBytes()*, *ShiftRows()*, *MixColumns()* і *AddRoundKey()* у послідовності (9.34) виконати інверсні їм перетворення, можна побудувати функцію оберненого розшифрування (рис. 9.29). При цьому порядок використання раундових ключів є оберненим у відношенні до того, який використовується при зашифруванні.

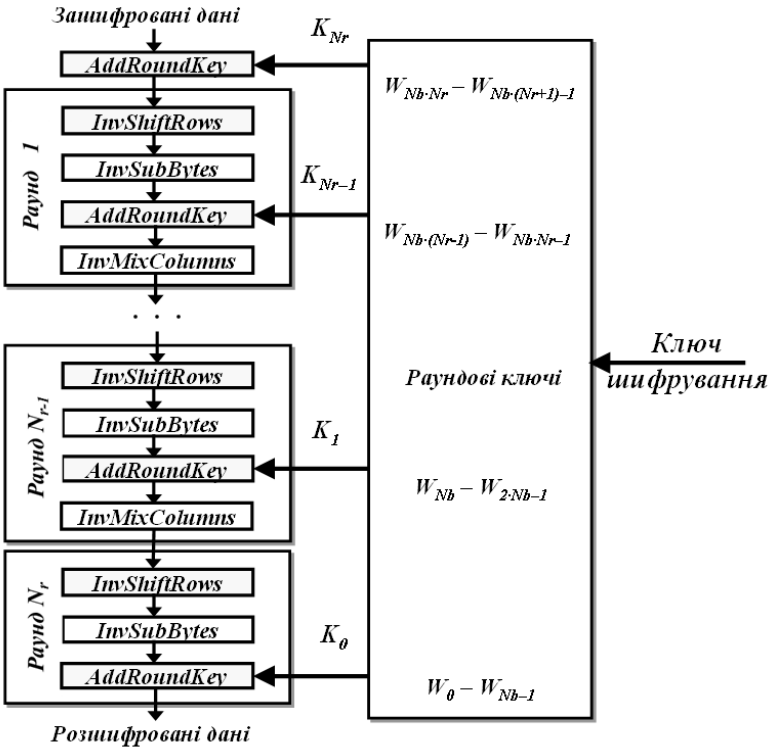


Рис. 9.29. Схема процедури оберненого (інверсного) розшифрування в криптографічнму алгоритмі *AES* (*Rijndael*)

Нижче наведено псевдокод програми, яка показує процес оберненого (інверсного) розшифрування даних (*InvDecipher*).

```
//=============================================
//= Процедура оберненого розшифрування в псевдокоді =
//=============================================
InvDecipher (byte in[4*Nb], word w[Nb*(Nr+1)], byte out[4*Nb])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey (state, w[Nr*Nb,(Nr+1)*Nb–1])
    InvShiftRows (state)
    InvSubBytes (state)
    for round = 1 step 1 to Nr–1
```

```
        AddRoundKey (state, w[(Nr-round)*Nb,
        ( Nr+1–round)*Nb–1])
        InvMixColumns (state)
        InvShiftRows (state)
        InvSubBytes (state)
    end for
    AddRoundKey (state, w[0,Nb–1])
    out = state
end
//=============================================
```

На рис. 9.29 та в тексті псевдокоду програми перетворення *InvAddRoundKey()* у початковому перетворенні та останньому раунді замінено на *AddRoundKey()*. Це обумовлено тим, що операції додавання та віднімання за модулем *2* є однаковими.

У першому (початковому) проекті *AES* порядок перетворень у кожному раунді при зашифруванні та розшифруванні не співпадав (рис. 9.30).

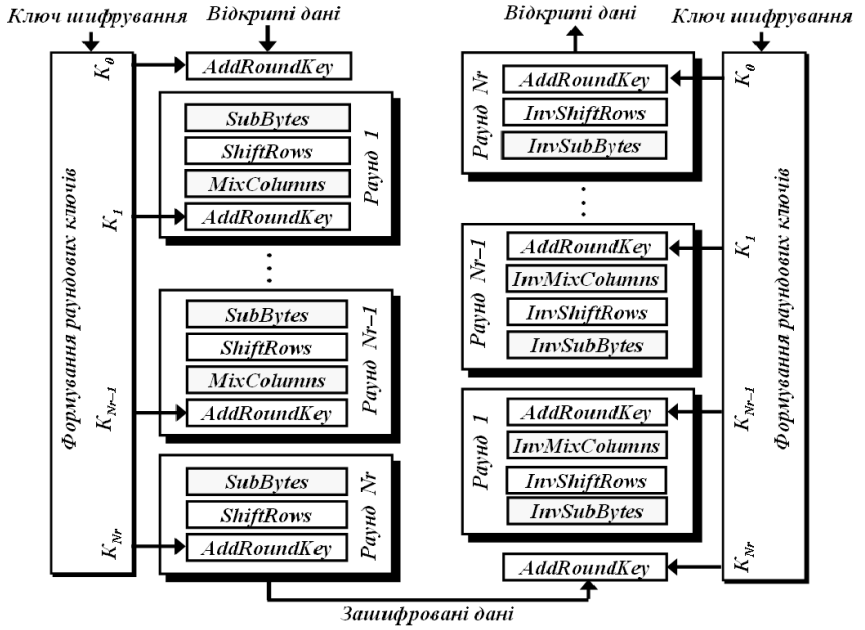


Рис. 9.30. Пояснення зашифрування й оберненого розшифрування в криптографічному алгоритмі *AES* (*Rijndael*)

Як видно з рис. 9.30, по-перше, при розшифруванні змінюється порядок виконання перетворень *SubBytes* (*InvSubBytes*) і *ShiftRows* (*InvShiftRows*); по-друге, при розшифруванні змінено порядок виконання перетворень *MixColumns* (*InvMixColumns*) і *AddRoundKey*.

Ці зміни в порядку необхідні, щоб при розшифруванні зробити порядок роботи перетворень інверсним у відношенні до порядку роботи при зашифруванні. Отже, алгоритм розшифрування в цілому — інверсія алгоритму зашифрування. На рис. 9.30 показано тільки три раунди, але інші мають той самий вигляд.

Необхідно звернути увагу, що ключі раунду при розшифруванні використовуються в зміненому (зворотному) порядку, а також на те, що алгоритми зашифрування й розшифрування в первинному проекті не збігаються.

9.8.2. Пряме (еквівалентне) розшифрування даних

Для тих додатків, яким потрібні співпадаючі алгоритми для зашифрування й розшифрування, було розроблено пряме (еквівалентне) розшифрування. У такій версії, яка називається *альтернативною*, перетворення при розшифрування побудоване так, щоб зробити порядок перетворень таким самим, як і при зашифруванні.

Якщо перетворення *B* і *R*, а також *K* і *C* поміняти місцями без зміни результату, то послідовність (9.34), прочитану справо наліво, можна записати у вигляді

$$\underbrace{KBRCK}_{1\text{-é раунд}}\underbrace{BRCK}_{2\text{-é раунд}}\ldots\underbrace{BRCK}_{9\text{-é раунд}}\underbrace{BRK}_{10\text{-é раунд}}. \qquad (9.35)$$

Послідовність (9.35) тепер точно співпадає з (9.34). Це означає, що блок (9.34) можна розшифрувати, використовуючи ту саму послідовність дій, що й при його зашифруванні, але з використанням раундових ключів у зворотному порядку.

Дійсно, операцію *SubBytes()* можна поміняти місцями із *ShiftRows()*. Те саме буде правильним й для операцій *InvSubBytes()* й *InvShiftRows()*. Це відбувається тому, що функції *SubBytes()* і *InvSubBytes()* працюють із байтами, а операції *ShiftRows()* і *InvShiftRows()* зсувають цілі байти, не зачіпаючи їх значень.

Перетворення *SubBytes()* змінює зміст кожного байта, не змінюючи порядок байтів матриці станів; *ShiftRows()* змінює порядок байтів

у матриці станів, не змінюючи зміст байтів. Маємо на увазі, що можна змінити порядок цих двох перетворень при розшифруванні, не зачіпаючи оборотність цілого алгоритму; рис. 9.31 ілюструє цю ідею.

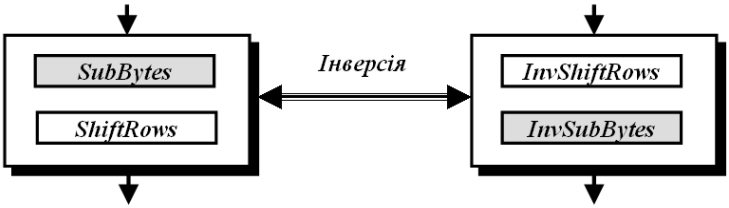


Рис. 9.31. Заміна порядку виконання перетворень *SubBytes()* і *InvShiftRows()*, а також *ShiftRows()* і *InvSubBytes()*

Зауважимо, що комбінації двох перетворень при зашифруванні й розшифруванні інверсні одна одній.

Алгоритм оберненого (інверсного) розшифрування, описаний вище, має порядок перетворень, обернений порядку в алгоритмі зашифрування, але використовує ті самі параметри (розгорнутий ключ). Проте деякі властивості алгоритму зашифрування *AES* (*Rijndael*) дозволяють застосувати для розшифрування такий самий порядок перетворень (використовуваний для зашифрування) завдяки зміні деяких параметрів, а саме — розгорнутого ключа:

$$\begin{aligned} &InvMixColumns(AddRoundKey()) = \\ &= InvMixColumns(State \oplus RoundKey) = \\ &= InvMixColumns(State) \oplus InvMixColumns(RoundKey). \end{aligned} \tag{9.36}$$

Ці властивості функцій алгоритму розшифрування дозволяють змінити порядок застосування перетворень *AddRounKey()* і *InvMixColumns()*. Така зміна порядку виконання, як слідує з (9.36), можлива за умови, що всі раундові ключі розшифрування (крім початкового раундового ключа — K_0 і останнього — K_{Nr}) заздалегідь пропущені (перетворені) через функцію *InvMixColumns()*.

Перетворення *InvMixColumns()* 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, використовуваної в перетворенні *InvMixColumns()*. Назвемо нове перетворення *InvAddRoundKey()*. Рис. 9.32 показує нову конфігурацію.

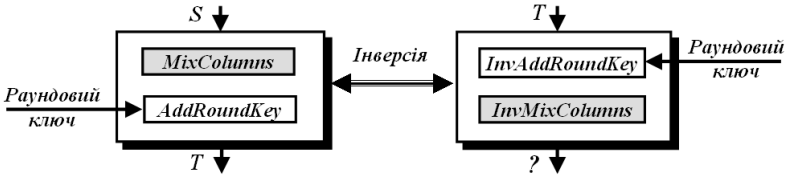


Рис. 9.32. Заміна порядку виконання перетворень *MixColumns()* і *InvAddRoundKey()*, а також *AddRoundKey()* і *InvMixColumns()*

Можна показати, що ці два перетворення *MixColumns()* і *AddRoundKey()* тепер інверсні один одному. При зашифруванні вхідну матрицю станів позначимо як *S*, а вихідну — *T*, а при розшифрування вхідну матрицю станів позначимо *T*. Покажемо, що в такому випадку вихідна матриця станів — *S*. Необхідно звернути увагу на те, що перетворення *MixColumns ()* — фактично результат множення матриці констант *C* на матрицю станів *S*. Отже, результат зашифрування — матриця станів *T*

$$T = C \bullet S \oplus K.$$

При розшифруванні

$$C^{-1} \cdot T \oplus C^{-1} \cdot K = C^{-1} \cdot (C \cdot S \oplus K) \oplus C^{-1} \cdot K =$$

$$= C^{-1} \cdot C \cdot S \oplus C^{-1} \cdot K \oplus C^{-1} \cdot K = S,$$

оскільки $C^{-1} \cdot C \cdot S = I \cdot S = S$, а $C^{-1} \cdot K \oplus C^{-1} \cdot K = 0$. Тут *I* — одинична матриця.

Слід звернути увагу, що необхідно використовувати два перетворення *AddRoundKey()* і дев'ять *InvAddRoundKey()*, як це показано на рис. 9.33.

Якщо в схемі на рис. 9.33 раундові ключі від K_1 до K_{Nr-1} перетворити за допомогою функції *InvMixColumns()*, тобто провести обчислення з використанням виразу:

$$K_i^* = InvMixColumns(K_i), \; i = 1 \ldots Nr-1,$$

то для розшифрування можна застосувати такий самий порядок функцій, що й при зашифруванні, але зворотний.

Нижче наведено псевдокод програми, яка показує процес прямого (еквівалентного) розшифрування даних (*EqDeCipher*).

```
//================================================
//=== Процедура прямого розшифрування в псевдокоді ===
//================================================
EqDeCipher (byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb–1])
    for round = Nr–1 step –1 downto 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
    end for
    InvSubBytes(state)
    InvShiftRows(state)
    AddRoundKey(state, dw[0, Nb–1])
    out = state
end
//================================================
```

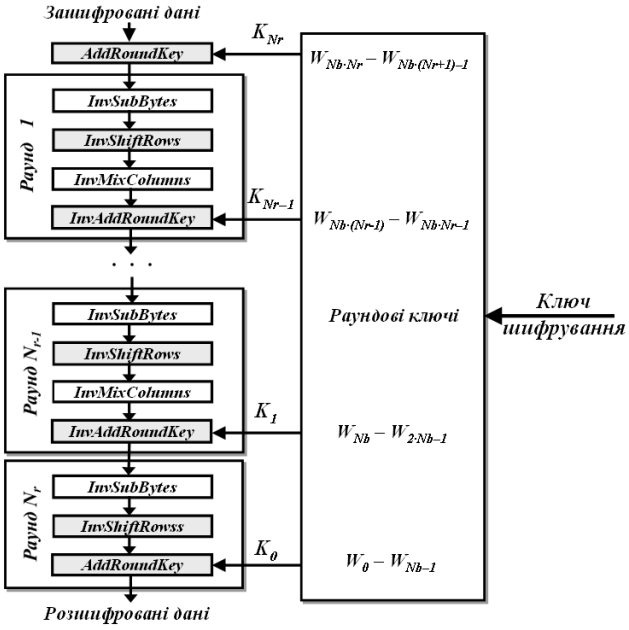


Рис. 9.33. Схема процедури прямого (еквівалентного) розшифрування в криптографічному алгоритмі *AES* (*Rijndael*)

У такому випадку процеси при зашифруванні й прямому розшифруванні в криптографічному алгоритмі *AES* (*Rijndael*) можна показати на рис. 9.34.

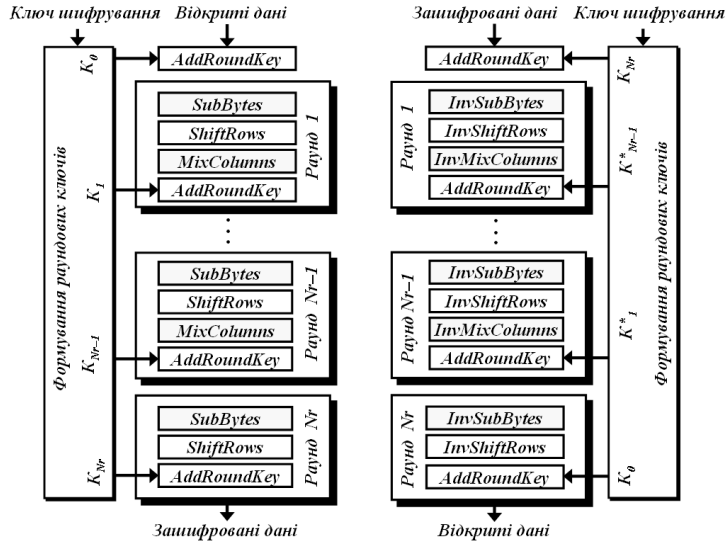


Рис. 9.34. Пояснення зашифрування й прямого розшифрування в криптографічному алгоритмі *AES* (*Rijndael*)

Для формування розгорнутого ключа розшифрування з урахуванням (9.37), до процедури розгортання ключа необхідно додати такий код:

```
//=============================================
EqKeyExpansion (byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
    for i = 0 step 1 to (Nr+1)*Nb–1
        dw[i] = w[i]
    end for
    for round = 1 step 1 to Nr–1
        InvMixColumns(dw[round*Nb, (round+1)*Nb–1])
    end for
//=============================================
```

Примітка. В останньому операторі (у функції *InvMixColumn()*) відбувається перетворення типу даних, оскільки розгорнутий ключ зберігається

у вигляді лінійного масиву *32*-розрядних слів, тоді як вхідний параметр функції — двовимірний масив байтів.

У табл. 9.8 наведено процедуру зашифрування, а також два еквівалентні варіанти процедури розшифрування при використанні двораундового варіанту *AES* (*Rijndael*).

Таблиця 9.8

Послідовність перетворень у двораундовому варіанті *AES* (*Rijndael*)

Перетворення для двораундового варіанту *AES* (*Rijndael*)		
зашифрування	обернене розшифрування	пряме розшифрування
AddRoundKey	*AddRoundKey*	*AddRoundKey*
SubBytes	*InvShiftRows*	*InvSubBytes*
ShiftRows	*InvSubBytes*	*InvShiftRows*
MixColumns	*AddRoundKey*	*InvMixColumns*
AddRoundKey	*InvMixColumns*	*AddRoundKey*
SubBytes	*InvShiftRows*	*InvSubBytes*
ShiftRows	*InvSubBytes*	*InvShiftRows*
AddRoundKey	*AddRoundKey*	*AddRoundKey*

Перший варіант функції розшифрування — звичайна інверсія функції зашифрування. Другий варіант функції розшифрування отриманий з першого після зміни порядку дотримання операцій у трьох парах перетворень: *InvShiftRows — InvSubBytes* (двічі) і *AddRoundKey — InvMixColumns*.

Очевидно, що результат перетворення за переходу від початкової до оберненої послідовності виконання операцій у вказаних парах не зміниться.

Видно, що процедура зашифрування та другий варіант процедури розшифрування співпадають з точністю до порядку використання раундових ключів (у ході виконання операцій *AddRoundKey()*), таблиць замін (у ході виконання операцій *SubBytes()* і *InvSubBytes()*) і матриць перетворення (у ході виконання операцій *MixColumns()* і *InvMixColumns()*). Цей результат так легко узагальнити на будь-яке інше число раундів [36].

9.9. АНАЛІЗ ТА БЕЗПЕКА ШИФРУ

9.9.1. Властивості симетричності та вразливі ключі

Враховуючи значний ступінь симетричності (однорідності), що притаманна шифру *AES* (*Rijndael*), у роботі з даними, було вжито спеціальні заходи, що дозволили зменшити її вплив. Це досягається завдяки різним для кожного раунду константам (див. обґрунтування вибору алгоритму розгортання ключа). Той факт, що алгоритми зашифрування й розшифрування складаються з різних операцій-функцій, майже унеможливлює існування уразливих і напівуразливих ключів шифрування (що відбувалося в *DES*).

Нелінійність процедури розгортання ключа також майже унеможливлює отримання еквівалентних раундових ключів після розгортання різних початкових ключів шифрування.

9.9.2. Диференціальний і лінійний криптографічний аналіз

Диференціальний криптографічний аналіз був уперше описаний Е. Біхамом і А. Шаміром [3, 19]. Лінійний криптографічний аналіз був уперше описаний М. Мацуї [25].

Диференціальний криптографічний аналіз

Диференціальний криптографічний (*ДК*) *аналіз* — це атака на шифр, що базується на можливості вільного підбору відкритих даних для подальшого їх зашифрування. Аналізу підлягають залежності між парами блоків даних до і після застосування шифру. ДК-атаки стають можливі тоді, коли можна встановити проникнення таких залежностей через майже усі (за винятком *2–3*) раунди шифруючого перетворення.

При цьому відносна кількість фіксованих пар бітів даних (назвемо це число *коефіцієнтом проникнення*), для яких можна встановити залежність відмінностей на виході від відмінностей на вході при зашифруванні, має бути значно вищою, ніж 2^{1-n}, де n — довжина вхідних даних у бітах. Спробуємо пояснити цей критерій успішності диференціального криптографічного аналізу.

Уявімо собі, що ми можемо довільно вибирати два різні блоки відкритих даних завдовжки по n бітів кожний для зашифрування та подальшого аналізу двох отриманих блоків зашифрованих даних.

Нагадуємо, будемо шукати пари бітів у вхідних даних, відмінності між якими підказали б нам, яка буде відмінність між бітами деякої певної пари бітів у вихідних даних.

Отже, вибираємо по одному певному біту з кожного з двох блоків відкритих даних і аналізуємо результати зашифрування за усіх можливих значень вхідних даних. Оскільки тепер у кожному блоці залишилося $n–1$ бітів, які можуть набувати різних значень, то нам слід проаналізувати лише 2^{n-1} значень кожного блока вхідних даних.

Якщо виявиться, що у вихідних даних існує пара бітів така, що відмінності між цими бітами можна передбачити, виходячи з відмінностей між бітами вибраної пари у вхідних даних, і ця подія не випадкова (тобто його ймовірність більша, ніж $1/2^{n-1}=2^{1-n}$), то можна вважати, що перетворення зашифрування має явну вразливість і може зазнавати диференціального криптографічного аналізу.

Раунд зашифрування при диференціальному криптографічному аналізі зручно розглядати як окрему різницеву сесію перетворення даних зі своїми вхідними й вихідними даними. Отже, повна різницева сесія складається з окремих сесій — раундів зашифрування, а повний коефіцієнт проникнення при зашифруванні є добутком коефіцієнтів проникнення складових сесій.

Отже, для того, щоб забезпечити стійкість до ДК-атак, достатньо щоб коефіцієнт проникнення після деякого числа раундів став не більше 2^{1-n}. Далі буде наведено доказ того, що вже після чотирьох раундів перетворення коефіцієнт проникнення стає не більше, ніж 2^{-150} (а після 8 раундів — менше 2^{-300}), що враховуючи табл. 9.1, доводить стійкість шифру за різних n.

Лінійний криптографічний аналіз

Лінійний криптографічний (*ЛК*) *аналіз* також застосовується в атаках із можливістю підбору відкритих даних для їх подальшого зашифрування. Він заснований на аналізі лінійних залежностей між бітами вхідних даних, які обумовлюють цілком певні залежності між бітами вихідних даних.

Нехай $A[i_1, i_2, i_3, \ldots, i_n]$ — сума за модулем 2 бітів масиву даних A, індексами яких є значення $i_1, i_2, i_3, \ldots, i_n$, тобто

$$A[i_1, i_2, i_3, \ldots, i_n] = A[i_1] \oplus A[i_2] \oplus A[i_3] \oplus \ldots \oplus A[i_n].$$

Тоді лінійною залежністю будемо називати вираз вигляду

$$P[i_1, i_2, i_3, \ldots, i_n] = C[i_1, i_2, i_3, \ldots, i_n] \oplus K[i_1, i_2, i_3, \ldots, i_n].$$

де P, C і K означають, відповідно, масиви бітів вхідних даних, вихідних даних і ключа.

Отже, якщо для досить великої кількості різних P і C вдається підібрати K, що задовольняє цьому виразу, то можна говорити про розкриття одного біта інформації про ключ на підставі існування лінійної залежності вхідних і вихідних даних. Існують також методи визначення більш ніж одного біта інформації про ключ на підставі аналізу лінійних залежностей.

ЛК-атаки стають ефективними тоді, коли можна встановити існування таких лінійних залежностей (інакше — кореляцій) після майже усіх (за винятком *2–3*) раундів шифруючого перетворення. При цьому відносне число (коефіцієнт) таких кореляцій у даних має бути значно вищим ніж $2^{-n/2}$ (тобто більше, ніж одна пара бітів). Кореляції мають знак, тобто можуть бути негативними або позитивними, залежно від того, чи співпадає залежність у вхідних даних із залежністю у вихідних або ж обернена їй.

Це дає ключ (гра слів) до знаходження бітів раундового ключа. Вважається, що загальна кореляція при шифруванні є композицією (чи сумою) усіх кореляцій, спостережуваних у кожній з окремих лінійних сесій (тобто в кожному раунді перетворення), де є передбачувані комбінації бітів у вхідних і вихідних даних.

Достатньою умовою стійкості до ЛК-атаки є відсутність будь-яких лінійних сесій із коефіцієнтом загальної кореляції вище, ніж $2^{-n/2}$. Далі буде доведено, що вже після чотирьох раундів перетворення коефіцієнт кореляції стає меншим ніж 2^{-75} (а після *8* раундів менше 2^{-150}).

Ефективність різницевих і лінійних сесій

У [36] показано, що:

- повний коефіцієнт проникнення різницевої сесії може бути апроксимований до добутку коефіцієнтів проникнення усіх активних у цій різницевій сесії таблиць замін *S*-блоків;

- загальна кореляція лінійної сесії може бути апроксимована до створення коефіцієнтів кореляції між вхідними й вихідними даними усіх активних у цій лінійній сесії таблиць замін *S*-блоків.

Отже, стратегія сесії перетворення будь-якого шифру така:

- вибрати *S*-блок такий, де максимальний коефіцієнт проникнення й максимальний коефіцієнт кореляції між вхідними й вихідними даними були б якомога меншими; для *S*-блоків *Rijndael* вони дорівнюють відповідно 2^{-6} і 2^{-3};

- сконструювати розсіювання так, щоб не було багатораундових перетворень із малою кількістю активних *S*-блоків.

Далі буде показано, що мінімальна кількість активних *S*-блоків у будь-якій *4*-раундовій різницевій або лінійній сесії дорівнює *25*. Це дає коефіцієнт проникнення 2^{-150} для будь-якої *4*-раундової різницевої сесії та найбільше 2^{-75} для коефіцієнта кореляції будь-якої *4*-раундової лінійної сесії. Це правильно для усіх розмірів блоків шифру й не залежить від значення раундового ключа.

Примітка. Нелінійність *S*-блока, вибраного навмання з набору можливих обернених *8*-розрядних блоків заміни, буде, швидше за все, меншою за оптимальну. Типові значення від 2^{-5} до 2^{-4} для максимального коефіцієнта проникнення і 2^{-2} для коефіцієнта кореляції між вхідними й вихідними даними таблиць заміни [36].

Проникнення образів активності

Для диференціального криптографічного аналізу кількість активних *S*-блоків в одному раунді визначається кількістю ненульових байтів, що задають відмінності у вхідних даних раунду (кількість ненульових байтів після застосування операції *xor* до двох масивів вхідних даних). Нехай образ (*pattern*) масиву *State*, що визначає позиції активних *S*-блоків, позначається терміном *"різницевий образ активності"*, і нехай *різницева байтова вага* означає кількість активних (ненульових) байтів в образі.

Для лінійного криптографічного аналізу кількість активних *S*-блоків визначається кількістю ненульових байтів у масиві вхідних даних. Нехай образ, що визначає позиції активних *S*-блоків, позначається терміном *"кореляційний образ активності"*, і нехай *кореляційна байтова вага Z(a)* буде кількістю активних (ненульових) байтів в образі *a*. Більше того, нехай стовпець в *образі активності* називається *активним*, якщо він містить хоча б один активний байт. Нехай *Zc(a)* — кількість активних стовпців в *a*, тобто, по суті, вага стовпця цього образу.

Байтова вага стовпця *j* образу *a* (*Zj(a)*) задає кількість активних байтів у цьому стовпці.

Загальна вага сесії дорівнює сумі ваги образів активності вхідних даних для кожного раунду, що входить до складу цієї сесії.

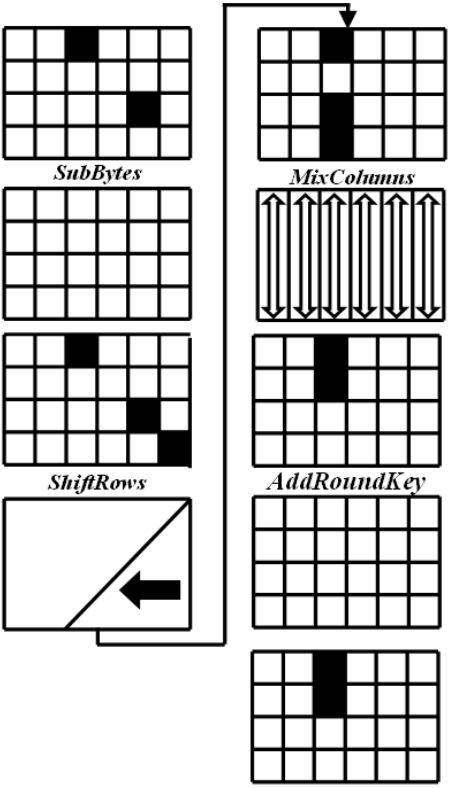


Рис. 9.35. Проникнення образів активності через одноразундове перетворення

Можна розглядати проникнення різницевих (лінійних) образів активності крізь раунди перетворення як проникнення залежностей вхідних/вихідних даних у різницевій (лінійній) сесії. Це проілюстровано на рис. 9.35, де чорним кольором виділені активні байти.

SubBytes() і *AddRoundKey()*. Образи активності, байтова вага й вага стовпця не змінюються.

ShiftRows(). Байтова вага не змінюється, оскільки значення самих байтів взагалі не змінюються.

MixColumns(). Оскільки стовпці перетворяться кожний окремо, вага стовпця не змінюється.

Отже, функції *SubBytes() і AddRoundKey()* не відіграють ніякої ролі в проникненні образів активності, і тому в нашому розгляді ефект раунду перетворень може бути зведений до ефекту від функцій *ShiftRows()* і *MixColumns()*.

Надалі *SubBytes() і AddRoundKey()* до уваги братися не будуть.

Функція *MixColumns()*, як було зазначено вище, має коефіцієнт проникнення, що дорівнює 5. Це означає, що для будь-якого активного стовпця з образу вхідних (чи вихідних) даних, сума байтової ваги на вході й виході раунду буде обмежена знизу значенням 5.

Функція, своєю чергою, має такі властивості:

- вага стовпців вихідних даних обмежена знизу максимальною байтовою вагою стовпця з вхідних даних;

- вага стовпців вхідних даних обмежена знизу максимальною байтовою вагою стовпця з вихідних даних.

Отже, нехай далі в нашому описі образ активності на вході *i*-го раунду позначається як a_{i-1}, а після додавання функції *ShiftRows()* позначається як b_{i-1}. Нумерація раундів розпочинається з одиниці, тому початковий образ активності має позначення a_0. Тоді a_i і b_i розділені лише функцією *ShiftRows()* і мають ту саму байтову вагу, а b_{i-1} і a_i розділені лише функцією *MixColumns()* і мають однакову вагу стовпців. Загальна вага *m*-раундової сесії дорівнює сумі ваги образів від a_0 до a_{m-1}. Властивості проникнення образів активності проілюстровано на рис. 9.36, де чорним кольором виділені активні байти, а сірим — активні стовпці.

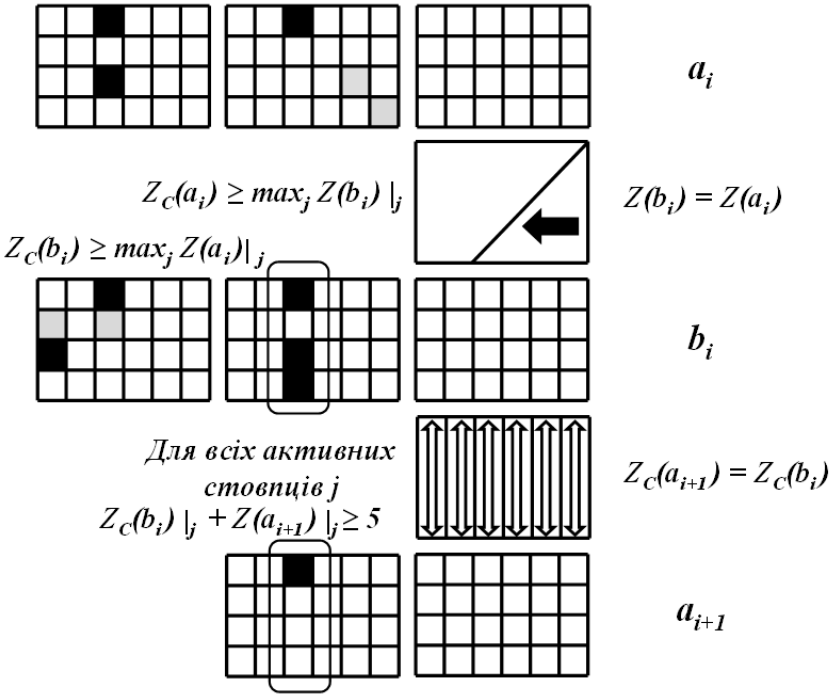


Рис. 9.36. Проникнення образів у перетвореннях одного раунду

Теорема 9.1. Загальна байтова вага двораундової сесії з Q активними стовпцями на вході другого раунду обмежена знизу значенням $5Q$.

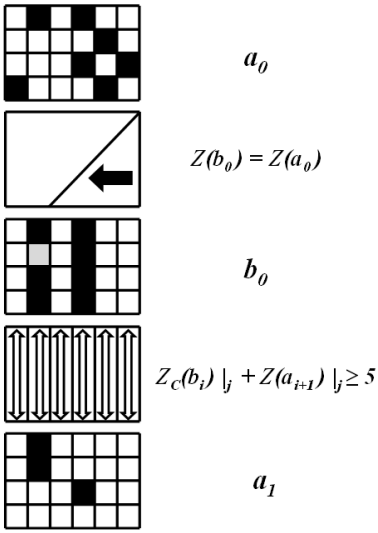


Рис. 9.37. Ілюстрація теореми 9.1 за $Q = 2$

Доведення

Той факт, що функція *MixColumns()* має коефіцієнт поширення, що дорівнює 5, означає, що сума байтової ваги в стовпцях образів b_0 і a_1 для кожного стовпця обмежена знизу значенням 5. Тому, якщо стовпцева вага a_1 дорівнює Q, то це дає обмеження знизу $5Q$ для суми байтових ваг b_0 і a_1. Оскільки a_0 і b_0 мають однакову байтову вагу, це обмеження дійсне також для суми байтової ваги a_0 і a_1, що й потрібно довести. Ілюстрацію теореми 9.1 показано на рис. 9.37.

Звідси витікає, що будь-яка двораундова сесія має як мінімум 5 активних S-блоків.

Лема 9.1. У двораундовій сесії сума активних стовпців у вхідних даних і активних стовпців у вихідних даних не менше 5. Іншими словами, сума стовпцевої ваги образів a_0 і a_2 не менше 5.

Доведення

Функція *ShiftRows()* зсуває байти будь-якого стовпця з образу a_i в різні стовпці образу b_i і навпаки. Звідси витікає, що стовпцева вага образу a_i обмежена знизу байтовою вагою окремих стовпців b_i. Так само стовпцева вага образу b_i обмежена знизу байтовою вагою окремих стовпців образу a_i.

У сесії активний як мінімум один стовпець образу a (чи, що теж саме, образу b_0). Позначимо цей стовпець як стовпець g. Оскільки коефіцієнт поширення функції *MixColumns()* дорівнює 5, то сума байтової ваги стовпця g в образі b_0 і в образі a_1 буде не менше 5. Але стовпцева вага образу a_0 обмежена знизу байтовою вагою стовпця g образу b_0. А стовпцева вага образу b_1 обмежена знизу байтовою вагою стовпця g образу a_1. Отже, сума стовпцевої ваги образів a_0 і b_1 обмежена знизу значенням 5.

Оскільки стовпцева вага образу a_2 дорівнює стовпцевій вазі образу b_1, то лему можна вважати доведеною. Ілюстрацію леми 9.1 наведено на рис. 9.38.

Теорема 9.2. Будь-яка сесія, що складається з *4* раундів, має, як мінімум, *25* активних байтів.

Доведення

Загальна байтова вага *4*-раундової сесії дорівнює сумі байтової ваги $Z(a_0)$, $Z(a_1)$, $Z(a_2)$ і $Z(a_3)$. Застосувавши теорему 9.1 до перших двох раундів (*1* і *2* на рис. 9.39) і до останніх двох раундів (*3* і *4* на рис. 9.39), отримуємо, що загальна байтова вага *4*-раундової сесії обмежена знизу добутком суми стовпцевої ваги образів a_1 і a_3 на *5*. А згідно з лемою 9.1 сума стовпцевої ваги для образів a_1 і a_3 не менше *5*. Звідси витікає, що байтова вага *4*-раундової сесії обмежена знизу значенням *25*. Теорему 9.2 проілюстровано на рис. 9.39.

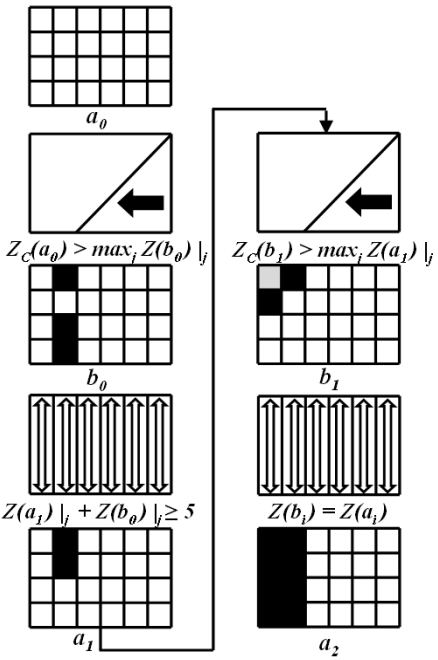


Рис. 9.38. Ілюстрація леми 9.1 з одним активним стовпцем в образі a_1

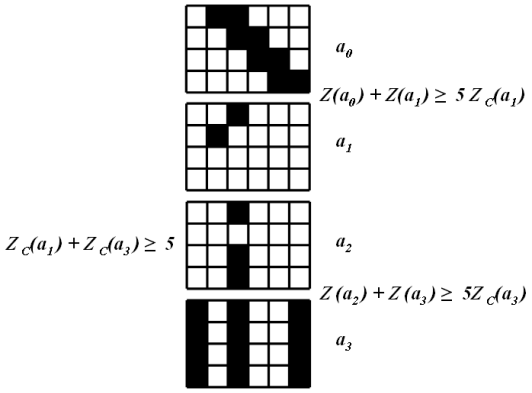


Рис. 9.39. Ілюстрація теореми 9.2

9.9.3. Атака методом скорочених диференціалів

Концепція скорочених диференціалів була вперше опублікована в роботі *Л. Кнудсена* [7, 19]. Атаки цього класу засновані на тому, що для деяких шифрів різницеві сесії мають тенденцію до кластеризації [36]. Це означає, що для деяких комбінацій лінійних залежностей у вхідних і вихідних даних кількість раундів, крізь які можна відстежити значну кількість таких залежностей, стає занадто великою. Іншими словами, ефективність диференціального криптографічного аналізу для таких комбінацій даних різко зростає.

Шифри, в яких усі перетворення виконуються над вирівняними блоками масиву вхідних даних, мають тенденцію бути особливо схильними до такого роду атак. Оскільки *Rijndael* якраз має ту властивість, що її функції перетворення оперують цілими байтами, то його схильність до цього виду атак була спеціально досліджена. Установлено, що для числа раундів, більшого за 6, не існує ефективнішої атаки, ніж повний перебір по усьому ключовому простору.

9.9.4. Атака "Квадрат"

Атаку "*Квадрат*" було спеціально розроблено для однойменного шифру *Square* (автори Й. Демен, Л. Кнудсен, В. Реймєн). Під час атаки використовується байт-зорієнтована структура шифру. Опис її був опублікований разом із самим шифром у роботі [36]. Враховуючи, що *Rijndael* успадкував багато властивостей шифру *Square*, ця атака застосовна й до нього. Далі наведено опис атаки "*Квадрат*" стосовно *Rijndael*.

Атака "*Квадрат*" заснована на можливості вільного підбору тим, що атакує деякий набір відкритих даних для подальшого їх зашифрування. Вона незалежна від таблиць замін *S*-блоків, багаточлена функції *MixColumns()* і способу розгортання ключа. Ця атака для *6*-раундового шифру *Rijndael*, ефективніша, ніж повний перебір по усьому ключовому простору. Після опису базової атаки на *4*-раундовий *Rijndael*, буде показано, як цю атаку можна продовжити на *5* і навіть *6* раундів.

Передумови

Нехай λ-набор — набір із *256* вхідних блоків (масивів *State*), кожний з яких має байти (назвемо їх активними), значення яких різні

для усіх *256* блоків. Інші байти (будемо називати їх пасивними) залишаються однаковими для усіх *256* блоків з λ-набору. Тобто для усіх x і y:

$$x,y \in \lambda : \begin{cases} x_{i,j} \neq y_{i,j}, & \text{якщо байт із номером } ij \text{ активний;} \\ x_{i,j} = y_{i,j}, & \text{в іншому випадку.} \end{cases}$$

Блоки λ-набору, що зазнали обробки функціями *SubBytes()* і *AddRoundKey()*, дадуть внаслідок іншого λ-набір з активними байтами в тих самих позиціях, що й початковий. Функція *ShiftRows()* змістить ці байти відповідно до заданих у ній зміщень у рядках масивів *State*. Після функції *MixColumns()* λ-набір у загальному випадку необов'язково залишиться λ-набором (тобто результат перетворення перестане задовольняти визначення λ-набору). Але оскільки кожний байт результату функції *MixColumns()* є лінійною комбінацією (з оберненими коефіцієнтами) чотирьох вхідних байтів того самого стовпця

$$b_{ij} = 2 \cdot a_{ij} \oplus 3 \cdot a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j},$$

стовпець з єдиним активним байтом на вході дасть на виході стовпець з усіма чотирма активними байтами.

Базова атака "Квадрат" на 4 раунди

Розглянемо λ-набір, в усіх блоках якого активний тільки один байт. Інакше кажучи, значення цього байта різне в усіх *256* блоках, а інші байти однакові (скажімо, дорівнюють нулю). Простежимо еволюцію цього байта впродовж трьох раундів. У першому раунді функція *MixColumns()* перетворить один активний байт у стовпець з *4* активних байтів. У другому раунді ці *4* байти розійдуться по 4 різних стовпцях внаслідок перетворення функцією *ShiftRows()*.

Функція *MixColumns()* наступного, третього, раунду перетворить ці байти в *4* стовпці, що містять активні байти. Цей набір все ще залишається λ-набором до того самого моменту, коли він надходить на вхід функції *MixColumns()* третього раунду.

Основною властивістю λ-набору, використовуваного тут, є те, що порозрядна сума за модулем *2* всіх блоків такого набору завжди дорівнює нулю. Дійсно, порозрядна сума неактивних (з однаковими

значеннями) байтів дорівнює нулю за визначенням операції порозрядного *xor*, а активні байти, пробігаючи усі *256* значень, також за порозрядного підсумовування дадуть нуль. Розглянемо тепер результат перетворення функцією *MixColumns()* в третьому раунді байтів вхідного масиву даних *a* у байти вихідного масиву даних *b*. Покажемо, що і в цьому випадку порозрядна сума всіх блоків вихідного набору буде дорівнює нулю, тобто:

$$\bigoplus_{b=MixColumns(a),a\in\lambda} b_{ij} = \bigoplus_{a\in\lambda} (2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j} =$$

$$= 2 \bigoplus_{a\in\lambda} a_{ij} \oplus 3 \bigoplus_{a\in\lambda} a_{(i+1)j} \oplus \bigoplus_{a\in\lambda} a_{(i+2)j} \oplus \bigoplus_{a\in\lambda} a_{(i+3)j} .$$

Отже, усі дані на вході четвертого раунду збалансовані (тобто їх повна сума дорівнює нулю). Цей баланс у загальному випадку порушується подальшим перетворенням даних функцією *SubBytes()*.

Припустимо далі, що четвертий раунд є останнім, тобто в ньому немає функції *MixColumns()*. Тоді кожний байт вихідних даних цього раунду залежить тільки від одного байта вхідних даних. Якщо позначити через *a* байт вихідних даних четвертого раунду, через *b* байт вхідних даних і через *k* відповідний байт раундового ключа, то можна записати:

$$a_{ij} = SubBytes(b_{ij}) \oplus k_{ij} .$$

Звідси, припускаючи значення k_{ij} можна за відомим a_{ij} обчислити b_{ij}, а потім перевірити правильність припущення про значення k_{ij}: якщо значення байта b_{ij} отримані при цьому k_{ij} не будуть збалансовані по усіх блоках (тобто не дадуть за порозрядного підсумовування нульовий результат), означає, що припущення неправильне.

Перебравши максимум *28* варіантів байта раундового ключа, знайдемо його істинне значення.

За таким самим принципом можуть бути визначені й інші байти раундового ключа. Завдяки тому, що пошук може здійснюватися окремо (тобто паралельно) для кожного байта ключа, швидкість підбору усього значення раундового ключа дуже велика; а за значенням повного раундового ключа, за відомого алгоритму його розгортання, не становить труднощів відновити сам ключ шифрування.

Додавання шостого раунду на початку базової атаки "Квадрат"

Основна ідея полягає в тому, щоб підібрати такий набір блоків відкритих даних, який на виході після першого раунду давав би λ-набір з одним активним байтом. Це вимагає припущення про значення чотирьох байтів ключа, використовуваних функцією *AddRoundKey()* перед першим раундом.

Для того, щоб на вході другого раунду був тільки один активний байт достатньо, щоб у першому раунді один активний байт залишався на виході функції *MixColumns()*. Це означає, що на вході *MixColumns()* першого раунду повинен бути такий стовпець, байти a якого для набору з 256 блоків внаслідок лінійного перетворення такі:

$$b_i = 2 \cdot a_i \oplus 3 \cdot a_{i+1} \oplus a_{i+2} \oplus a_{i+3}, \quad 0 \le i \le 3,$$

де i — номер рядка, для одного певного i давали 256 різних значень, тоді як для кожного з інших трьох значень i результат цього перетворення повинен залишатися постійним. Слідуючи назад по порядку додатка функцій перетворення в першому раунді, до *ShiftRows()* цю умову треба застосувати до відповідно до рознесених по стовпцях 4 байтам. З урахуванням застосування функції *SubBytes()* і складання з передбачуваним значенням 4-байтового раундового ключа можна складати рівняння й підбирати потрібні значення байтів відкритих даних, що подаються для зашифрування та подальшого аналізу результату:

$$b_{ij} = 2 \cdot SubBytes(a_{ij} \oplus k_{ij}) \oplus 3 \cdot SubBytes(a_{(i+1)(j+1)} \oplus$$

$$\oplus k_{(i+1)(j+1)}) \oplus SubBytes(a_{(i+2)(j+2)} \oplus k_{(i+2)(j+2)}) \oplus$$

$$\oplus SubBytes(a_{(i+3)(j+3)} \oplus k_{(i+3)(j+3)}), \quad 0 \le i, j \le 3.$$

Отже, отримуємо такий алгоритм зламу: маємо всього 2^{32} різних значень a для певних i і j; інші байти для усіх блоків однакові (пасивні байти); припустивши значення чотирьох байтів k ключа першого раунду, підбираємо (виходячи з вищеописаної умови) набір з 256 блоків; ці 256 блоків стануть λ-набором після першого раунду; до цього λ-набору застосовано базову атаку для 4 раундів. Підібраний з її допомогою один байт ключа останнього раунду фіксується. Тепер підбираємо новий набір із 256 блоків для того самого значен-

ня *4* байтів *k* ключа першого раунду. Знову здійснюється базова атака, що дає один байт ключа останнього раунду. Якщо після декількох спроб значення цього байта не змінюється, це означає, що злам знаходиться на правильному шляху. Інакше необхідно змінити припущення про значення *4* байтів *k* ключа першого раунду. Такий алгоритм дій досить швидко приведе до повного відновлення усіх байтів ключа останнього раунду.

9.9.5. Атака методом інтерполяцій

У роботі [7, 19, 36] *Т. Якобсен* і *Л. Кнудсен* представили новий вид атаки на блокові шифри. Суть її полягає в тому, що той, який атакує, намагається отримати багаточлен із відомих йому пар вхідних/вихідних даних. Це стає можливим лише в тих випадках, коли функції, що входять до складу перетворення, можуть бути досить компактно виражені алгебраїчно, а все перетворення представлено у вигляді вираження алгебри вирішуваної складності.

Атака ґрунтується на такому факті: якщо отриманий багаточлен (чи раціональне вираження) має невелику степінь, то достатньо декількох пар вхідних/вихідних даних для того, щоб підібрати коефіцієнти багаточлена, значення яких прямо пов'язані зі значенням ключа. Складність представлення алгебри підстановки *SubBytes()* у полі $GF(2^8)$ у поєднанні з розсіюванням функцією *MixColumns()* робить неможливим застосування цього типу атак до багатораундового шифру ***Rijndael***.

Приклад алгебраїчного виразу для функції *SubBytes()*:

$$\{63\} + \{8f\}\cdot x^{127} + \{b5\}\cdot x^{191} + \{01\}\cdot x^{223} + \{f4\}\cdot x^{239} +$$

$$+ \{25\}\cdot x^{247} + \{f9\}\cdot x^{251} + \{09\}\cdot x^{253} + \{05\}\cdot x^{259}.$$

9.9.6. Про існування вразливих ключів

Уразливими вважаються ключі, використання яких призводить до невдалого перетворення вхідних даних (недостатнє розсіювання або перемішування). Найбільш відомий випадок існування вразливих ключів для шифру *IDEA* описано в [5, 20]. Як правило, такий недолік властивий шифрам, в яких є нелінійні операції, залежні від значення ключа. У разі *Rijndael* — додавання ключа в процедуру

шифрування здійснюється з використанням операції *xor*, а нелінійне перетворення реалізують *S*-блоки з фіксованими таблицями замін.

Отже, у *Rijndael* відсутні обмеження на вибір ключів.

9.9.7. Атака еквівалентних ключів

У [38] Е. Біхам описав *атаку еквівалентних ключів*. Пізніше Д. Келсі, Б. Шнайєр і Д. Вагнер показали в роботі [26], що деякі шифри є недостатньо стійкими до цієї атаки.

В атаці еквівалентних ключів криптографічний аналітик виконує шифрування, використовуючи різні (повністю або частково невідомі) ключі, які мають задані взаємозалежності. Враховуючи високий ступінь розсіювання при розгортанні ключа, можливість цієї атаки на *Rijndael* малоймовірна.

Контрольні питання та завдання

1. Перерахувати параметри (розмір блока, розмір ключа та кількість раундів) для трьох версій *AES*.

2. Скільки перетворень є в кожній версії *AES*? Скільки ключів необхідно для кожної версії?

3. Порівняти *DES* і *AES*. Який із них орієнтований на роботу з бітом, а який – на роботу з байтом?

4. Визначити матрицю станів в *AES*. Скільки матриць станів є в кожній версії *AES*?

5. Які із чотирьох перетворень, визначених для *AES*, змінюють зміст байтів, а які не змінюють?

6. Порівняти підстановку в *DES* і *AES*. Чому в *AES* є тільки одна таблиця підстановки і декілька в *DES* (*S*-блоків)?

7. Порівняти перестановки в *DES* і *AES*. Чому потрібно мати розширення й стиснення перестановки в *DES* і не потрібно в *AES*?

8. Порівняти ключі раунду в *DES* і *AES*. В якому шифрі розмір ключа раунду дорівнює розміру блока?

9. Чому перетворення *MixColumns*, яке змішує дані, потрібне в *AES*, але не потрібне в *DES*?

10. Перерахувати режими роботи для алгоритму *AES*.

11. Зробити складання двох елементів кінцевого поля *GF(2)* x^7+x^3+x+1 і $x^6+x^3+x^2+1$.

12. Визначити частку й залишок від ділення елемента $x^{13}+x^{11}+x^9+x^7+x^5+x+1$ на елемент $x^6+x^3+x^2+1$ в кінцевому полі *GF(2)*.

13. Зробити складання двох багаточленів із коефіцієнтами кінцевого поля $GF(2^8)$ $\{f5\}\cdot x^7+\{03\}\cdot x^3+\{09\}\cdot x+\{02\}$ і $\{07\}\cdot x^7+\{1b\}\cdot x^5+\{1e\}\cdot x+\{1f\}$. Незвідний багаточлен $p(x) = x^8+x^4+x^3+x+1$.

14. Зробити множення двох багаточленів із коефіцієнтами з кінцевого поля $GF(2^8)$ $\{02\}\cdot x^3+\{05\}\cdot x^2+\{03\}\cdot x+\{04\}$ і $\{03\}\cdot x^3+\{05\}\cdot x^2+\{04\}\cdot x+\{01\}$. Незвідний багаточлен $p(x) = x^8+x^4+x^3+x+1$.

15. Зробити множення двох багаточленів із коефіцієнтами з кінцевого поля $GF(2^8)$ $\{02\}\cdot x^3+\{05\}\cdot x^2+\{03\}\cdot x+\{04\}$ і $\{03\}\cdot x^3+\{05\}\cdot x^2+\{04\}\cdot x+\{01\}$. Незвідний багаточлен $p(x) = x^8+x^4+x^3+x+1$. Результат множення повинен бути представлений 4-байтовим словом, при цьому використати багаточлен $m(x) = x^4+1$.

16. Визначити адитивну інверсію багаточлена завдовжки в один байт із кінцевого поля $GF(2^8)$ $x^6+x^3+x^2+1$.

17. Визначити мультиплікативну інверсію багаточлена завдовжки в один байт із кінцевого поля $GF(2^8)$ $x^6+x^3+x^2+1$. Незвідний багаточлен $p(x) = x^8+x^4+x^3+x+1$.

18. Визначити результат афінного перетворення багаточлена довжиною в один байт x^5+x^2+1 з кінцевого поля $GF(2^8)$ в алгоритмі *AES*. Незвідний багаточлен $p(x) = x^8+x^4+x^3+x+1$.

19. Значення байтів матриці стану даних на вході функції *SubBytes()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *a49c7ff2689f352b6b5bea43026a5049*. Визначити значення байтів матриці стану даних на виході функції *SubBytes()*.

20. Значення байтів матриці стану даних на вході функції *InvSubBytes()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *f196db453b53027789d2de491a87397f*. Визначити значення байтів матриці стану даних на виході функції *InvSubBytes()*.

21. Значення байтів матриці стану даних на вході функції *ShiftRows()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *49ded28945db96f17f39871a7702533b*. Визначити значення байтів матриці стану даних на виході функції *ShiftRows()*.

22. Значення байтів матриці стану даних на вході функції *InvShiftRows()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *f196db453b53027789d2de491a87397f*. Визначити значення байтів матриці стану даних на виході функції *InvShiftRows()*.

23. Значення байтів матриці стану даних на вході функції *MixColumns()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *49db873b453953897f02d2f177de961a*. Визначити значення байтів матриці стану даних на виході функції *MixColumns()*.

24. Значення байтів матриці стану даних на вході функції *InvMixColumns()* криптографічного системи *AES-128* у шістнадцятковій

системі дорівнюють: *31adb22485c967caedd5e4a2ff9e4ac3*. Визначити значення байтів матриці стану даних на виході функції *InvMixColumns()*.

25. Значення байтів матриці стану даних на вході функції *AddRoundKey()* криптографічного системи *AES-128* у шістнадцятковій системі дорівнюють: *584dcaf11b4b5aacdbe7caa81b6db0e5*. Значення байтів матриці стану раундового ключа на вході функції *AddRoundKey()* криптографічного системи *AES-128* в шістнадцятковій системі дорівнюють: *f2c295f27a9bb9435935807a7359f67f*. Визначити значення байтів матриці стану даних на виході функції *AddRoundKey()*.

26. Значення байтів матриці стану раундового ключа криптографічного системи *AES-128* для дев'ятого раунду зашифрування у шістнадцятковій системі дорівнюють: *584dcaf11b4b5aacdbe7caa81b6db0e5*. Визначити значення байтів матриці стану раундового ключа для десятого раунду.

27. Значення перших шести слів розгорнутого ключа *AES-192* в шістнадцятковій системі дорівнюють: *w[0] = 00010203*; *w[1] = 04050607*; *w[2] = 08090a0b*; *w[3] = 0c0d0e0f*; *w[4] = 10111213*; *w[5] = 14151617*. Визначити значення других шести слів розгорнутого ключа зашифрування *AES-192*.

28. Значення перших восьми слів розгорнутого ключа *AES-256* у шістнадцятковій системі дорівнюють: *w[0] = 00010203*; *w[1] = 04050607*; *w[2] = 08090a0b*; *w[3] = 0c0d0e0f*; *w[4] = 10111213*; *w[5] = 14151617*; *w[6] = 18191a1b*; *w[7] = 1c1d1e1f*. Визначити значення других восьми слів розгорнутого ключа зашифрування *AES-256*.

Розділ 10

АСИМЕТРИЧНІ КРИПТОГРАФІЧНІ СИСТЕМИ ШИФРУВАННЯ

10.1. ПЕРЕДІСТОРІЯ Й ОСНОВНІ ІДЕЇ

Розглянемо три завдання, вирішення яких допоможе нам краще зрозуміти ідеї та методи криптографії з відкритими ключами. Усі ці завдання мають важливе практичне значення.

Перше завдання — збереження паролів у комп'ютері [10]. Відомо, що кожний користувач у мережі має свій таємний пароль. Входячи в мережу, користувач вказує своє ім'я (несекретне) і потім вводить пароль. Проблема в тому, що: 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.

Друге завдання виникло з появою радіолокаторів і *системи протиповітряної оборони* (ППО). При перетині кордону літаком радіолокатор запитує пароль. Якщо пароль правильний, то літак "*свій*", в іншому випадку — "*чужий*". Тут виникає така проблема: оскільки пароль повинен передаватися по відкритому каналу (повітряному середовищі), то противник може прослуховувати всі переговори й взнати правильний пароль. Потім "*чужий*" літак у разі запиту повторить перехоплений раніше "*правильний*" пароль як відповідь локатора й буде пропущений.

Третє завдання схоже на попереднє й виникає в комп'ютерних мережах із віддаленим доступом, наприклад, при взаємодії банку й клієнта. Зазвичай на початку сеансу банк запитує в 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 всіх цих завдань ґрунтується на важливому понятті односторонньої функції (*one-way function*).

Визначення 10.1. Нехай дана функція

$$y = f(x), \tag{10.1}$$

визначена на кінцевій множині X ($x \in X$), для якої існує обернена функція:

$$x = f^{-1}(y). \tag{10.2}$$

Функція називається *односторонньою*, якщо обчислення за формулою (10.1) — проста задача, що вимагає трохи часу, а обчислення за (10.2) — завдання складне, 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 [10, 17, 25], але для наших цілей достатньо й вищенаведеного.

Як приклад односторонньої функції розглянемо таку:

$$y = a^x \bmod p, \tag{10.3}$$

де p — деяке просте число (тобто таке, яке ділиться без залишку тільки на себе й на одиницю); x — ціле число з множини $\{1, 2, ..., p–1\}$.

Обернена функція позначається

$$x = \log_a y \bmod p \tag{10.4}$$

і називається дискретним логарифмом.

Для того щоб забезпечити труднощі обчислення за (10.4) 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.

Спочатку покажемо, що обчислення за (10.3) може бути виконане достатньо швидко. Почнемо з прикладу обчислення числа $a^{16} \bmod p$. Можна записати:

$$a^{16} \bmod p = (((a^2)^2)^2)^2 \bmod p,$$

тобто значення даної функції обчислюється лише за чотири операції множення замість 15 за "*наївного*" варіанта $a \cdot a \cdot \ldots \cdot a$. На цьому заснований загальний алгоритм.

Для опису алгоритму введемо величину $t = [\log_2 x]$ — цілу частину $\log_2 x$ (далі всі логарифми будуть двійкові, тому надалі не будемо вказувати основу логарифма *2*). Обчислюємо числа ряду:

$$a, a^2, a^4, a^8, \ldots, a^{2^t} \pmod p. \tag{10.5}$$

У ряду (10.5) кожне число виходить шляхом множення попереднього числа самого на себе за модулем p. Запишемо показник ступеня x у двійковій системі числення:

$$x = (x_t x_{t-1} \ldots x_1 x_0)_2.$$

Тоді число $y = a^x \bmod p$ може бути обчислене як:

$$y = \prod_{i=0}^{t} a^{x_i \cdot 2^i} \bmod p. \tag{10.6}$$

Усі обчислення проводяться за модулем p.

Приклад 10.1. Нехай потрібно обчислити $3^{100} \bmod 7$.

Маємо $t = [\log 100] = 6$. Обчислюємо числа ряду (10.5):

a	a^2	a^4	a^8	a^{16}	a^{32}	a^{64}
3	*2*	*4*	*2*	*4*	*2*	*4*

(10.7)

У загальному випадку справедливе твердження.

Твердження 10.1 (про складність обчислень (10.3)). Кількість операцій множення за обчислення (10.3) за описаним методом не перевищує $2 \cdot \log x$.

Доведення

Для обчислення чисел ряду (10.5) потрібно t множень, для обчислення y за (10.6) не більше, ніж t множень (див. приклад 10.1). З умови $t = [\log x]$, враховуючи, що $[\log x] \le \log x$, робимо висновок про справедливість доказуваного твердження.

Зауваження. Як буде показано надалі, підносячи до степеня за модулем p, є сенс використовувати лише показники $x < p$. У цьому випадку ми можемо сказати, що кількість операцій множення за обчислення (10.3) не перевищує $2 \cdot \log p$.

Важливо зазначити, що настільки ж ефективні алгоритми обчислення оберненої функції (10.4) невідомі. Один із методів обчислення (10.4), який називається "*крок немовлятi, крок велетня*" (буде

детально описаний у підрозділі 10.2). Цей метод вимагає порядку $2\sqrt{p}$ операцій.

Покажемо, що за великих p функція (10.3) дійсно одностороння, якщо для обчислення оберненої функції використовується метод "крок немовляти, крок велетня". Отримуємо такий результат (табл. 10.1).

Таблиця 10.1

Кількість множень для обчислення прямої й оберненої функцій

Кількість десяткових знаків у записі p	Обчислення (10.3) ($2 \cdot \log p$ множень)	Обчислення (10.4) ($2 \cdot \sqrt{p}$ 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 цієї таблиці бачимо, що якщо використовувати модулі, що складаються з *50–100* десяткових цифр, то "*пряма*" функція обчислюється швидко, а обернена — майже не обчислюється. Розглянемо, наприклад, суперкомп'ютер, який множить два *90*-значних числа за 10^{-14} с (для сучасних комп'ютерів це поки не доступно). Для обчислення (10.3) такому комп'ютеру потрібно:

$$T_{ОБЧ.ПР} = 600 \cdot 10^{-14} = 6 \cdot 10^{-12}\ с,$$

а для обчислення (10.4)

$$T_{ОБЧ.ОБЕРН} = 10^{45} \cdot 10^{-14} = 10^{31}\ с,$$

тобто більше 10^{22} років.

З цього виходить, що обчислення обернених функцій майже неможливе за довжини чисел близько *90* десяткових цифр, і використання паралельних обчислень і комп'ютерних мереж істотно не змінює ситуацію. У розглянутому прикладі ми припускали, що обернена функція обчислюється за $2\sqrt{p}$ операцій. У даний час відомі також більш швидкі методи обчислення дискретного логарифма, однак загальна картина та сама: кількість необхідних операцій більше $2 \cdot \log p$. Отже, можна стверджувати, що функція (10.3) дійсно одностороння, але із застереженням. Ніким не доведено, що обернена функція (10.4) не може бути обчислена так само швидко, як і пряма.

Використовуємо односторонню функцію (10.3) для вирішення всіх трьох завдань, описаних на початку даного 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 завдання ґрунтується на тому, що паролі взагалі не зберігаються! Точніше, реєструючись у мережі, користувач набирає своє ім'я й пароль: наприклад, його ім'я — *"фрукт"*, а пароль — *"абрикос"*. Комп'ютер розглядає слово *"абрикос"* як двійковий запис числа x і обчислює (10.3), 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 (10.3) за $x = $ *пароль*. За всіх подальших входів цього користувача після введення пари (*"фрукт"*— *"абрикос"*), комп'ютер обчислює за (10.3) нове значення $y_{НОВ}$ з $x =$ *"абрикос"* і порівнює зі збереженим у пам'яті раніше обчисленим значенням y. Якщо $y_{НОВ}$ збігається зі збереженим у пам'яті y, відповідним даному імені, то це законний користувач. В іншому випадку це зловмисник.

Зловмисник міг би спробувати знайти x за y. Однак вже при *90*-значних числах для цього буде потрібно більше, ніж 10^{22} років.

Отже, надана система зберігання пароля досить надійна й у даний час використовується в багатьох реальних операційних системах.

Розглянемо рішення другого завдання (ППО і літак). Можна використовувати такий метод: кожному своєму літаку присвоюється секретне ім'я, відоме системі ППО і льотчику, точніше, бортовому комп'ютеру. Нехай, наприклад, одному з літаків присвоєно секретне ім'я *СОКІЛ*, і цей літак наближається до кордону *1 лютого 2012 р. о 12.45*. Тоді перед наближенням до кордону бортовий комп'ютер літака формує слово:

СОКІЛ	*2012*	*02*	*01*	*12*	*45*
ім'я	*рік*	*місяць*	*день*	*години*	*хвилини*

Іншими словами, комп'ютер на літаку та станція ППО 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 несекретні. Потім літак повідомляє число y станції ППО. Станція порівнює обчислене нею число y з отриманим від літака. Якщо обчислене й отримане значення співпали, то літак визнається як "*свій*".

Противник не може зламати цю систему. Справді, з одного боку, він не знає секретного слова *СОКІЛ* 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 завдання ППО вимагає точної синхронізації годин у літаку й локаторі. Ця проблема досить легко вирішується. Наприклад, служба навігації постійно передає позначки часу у відкритому вигляді (ще нетаємно), і всі літаки й локатори використовують ці мітки для синхронізації своїх годин. Але є дещо інші складні проблеми. Позначка часу додається до слова *x* для того, щоб усі обчислювані значення *y* були різні й противник не міг їх повторно використовувати. Однак противник може спробувати миттєво повторити *y* в межах поточної хвилини. Як цьому запобігти? Це перше питання. Інше ускладнення виникає в ситуації, коли літак посилає число *y* в кінці *45*-ї хвилини, а локатор приймає його на початку *46*-ї. Надаємо читачеві можливість самостійно запропонувати варіант вирішення цих проблем.

Інший спосіб вирішення завдання ППО можливий, якщо ми будемо використовувати додатковий відкритий канал передачі даних від локатора до літака. Як описано вище, кожний свій літак і локатор знають секретне слово (типу *СОКІЛ*), яке не замінюється. Виявивши ціль, локатор посилає їй випадково згенероване число *a* (*"виклик"*). Літак обчислює $y = a^x \bmod p$, де *x* — секретне слово (*СОКІЛ*), 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 це завдання, напевно, було історично першим, вирішуючи яке, використовувалися односторонні функції.

Третє завдання вирішується аналогічно, і обидва розглянутих методи формування пароля застосовуються й використовуються в реальних мережевих протоколах.

10.2. ПЕРША КРИПТОГРАФІЧНА СИСТЕМА З ВІДКРИТИМ КЛЮЧЕМ – СИСТЕМА ДІФФІ–ХЕЛЛМАНА

Ця криптосистема була відкрита в середині *70*-х років американськими вченими Діффі та Хеллманом і 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 потрібно організувати секретний зв'язок для кожної пари з них. Якщо використовувати звичайну систему розподілу секретних ключів, то кожна пара абонентів повинна бути забезпечена своїм секретним ключем, тобто всього буде потрібно

$$\tilde{N}_N^2 = \frac{N(N-1)}{2} \approx \frac{N^2}{2}$$

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 та Хеллман вирішили цю проблему завдяки відкритому поширенню й обчисленню ключів. Перейдемо до опису запропонованої ними системи.

Нехай потрібно 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 $g \bmod p$ (відомі різні підходи для знаходження таких чисел g, один із них буде представлений нижче). Числа p і g відомі всім абонентам.

Абоненти вибирають великі числа x_A, x_B, x_C, які зберігають у таємниці (зазвичай такий вибір рекомендується проводити випадково, використовуючи датчики випадкових чисел). Кожний абонент обчислює відповідне число y, яке відкрито передається іншим абонентам,

$$\begin{cases} y_A = g^{x_A} \bmod p; \\ y_B = g^{x_B} \bmod p; \\ y_C = g^{x_C} \bmod p. \end{cases} \quad (10.9)$$

Як результат складемо табл. 10.2.

Таблиця 10.2

Ключі користувачів у системі Діффі–Хеллмана

Абонент	Секретний ключ	Відкритий ключ
A	x_A	y_A
B	x_B	y_B
C	x_C	y_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:

$$Z_{AB} = (y_B)^{x_A} \bmod p. \quad (10.10)$$

Ніхто інший окрім *A* цього зробити не може, тому що число x_A секретне.

Своєю чергою, абонент *B* обчислює число:

$$Z_{BA} = (y_A)^{x_B} \bmod p. \quad (10.11)$$

Схему криптографічної системи Діффі–Хеллмана наведено на рис. 10.1.

Твердження 10.2. $Z_{AB} = Z_{BA}$.

Доведення

Дійсно,

$$Z_{AB} = (y_B)^{x_A} \bmod p = (g^{x_B})^{x_A} \bmod p = \\ = g^{x_A x_B} \bmod p = (y_A)^{x_B} = Z_{BA}.$$

Тут перша рівність випливає з (10.10), друга й четверта — з (10.9), остання — з (10.11).

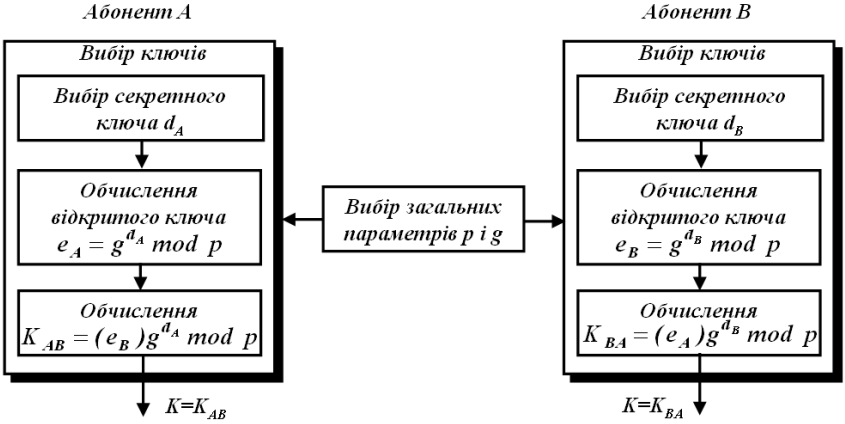


Рис. 10.1. Схема криптографічної системи Діффі–Хеллмана

Зазначимо головні властивості системи:

1) абоненти A і B отримали те саме число $Z = Z_{AB} = Z_{BA}$, яке не передавалося по відкритому каналу зв'язку;

2) зловмисник не знає секретних чисел x_A і x_B, тому не може обчислити число Z_{AB} або Z_{BA} (взагалі, він міг би спробувати знайти секретне число x_A за y_A (див. (10.9)), проте за великих p це майже неможливо (потрібні мільйони років)).

Абоненти A і B можуть використовувати $Z = Z_{AB} = Z_{BA}$ як секретний ключ для зашифрування й розшифрування даних. Так само будь-яка пара абонентів може обчислити секретний ключ, відомий тільки їм.

Зупинимося тепер на згаданій вище задачі вибору числа g. За довільно заданого p це може виявитися важким завданням, пов'язаним із розкладанням на прості множники числа $p–1$. Для забезпечення високої стійкості розглянутої системи число $p–1$ повинне обов'язково містити великий простий множник (в іншому випадку *алгоритм Поліга–Хеллмана*, описаний, наприклад, у [23], швидко обчислює дискретний логарифм). Тому часто рекомендують використовувати такий підхід: просте число p вибирається таким, щоб виконувалася рівність:

$$p = 2 \cdot q + 1,$$

де q — також просте число,

тоді як g можна взяти будь-яке число, для якого справедливі нерівності:

$$1 < g < p - 1 \quad \text{і} \quad g^q \bmod p \neq 1.$$

Приклад 10.2. Нехай

$$p = 23 = 2 \cdot 11 + 1 \quad (q = 11).$$

Виберемо параметр g. Спробуємо взяти $g = 3$. Перевіримо:

$$g^q \bmod p = 3^{11} \bmod 23 = 1,$$

а отже, таке g не підходить. Візьмемо $g = 5$. Перевіримо:

$$g^q \bmod p = 5^{11} \bmod 23 = 22 \neq 1.$$

Отже, можна вибрати параметри $p = 23$, $g = 5$.

Тепер кожний абонент вибирає секретне число й обчислює відповідне йому відкрите число. Нехай вибрано $x_A = 7$, $x_B = 13$. Обчислюємо

$$y_A = 5^7 \bmod 23 = 17, \quad y_B = 5^{13} \bmod 23 = 21.$$

Нехай A і B вирішили сформувати спільний секретний ключ. Для цього A обчислює

$$Z_{AB} = (y_B)^{x_A} \bmod p = 21^7 \bmod 23 = 10,$$

а B обчислює

$$Z_{BA} = (y_A)^{x_B} \bmod p = 17^{13} \bmod 23 = 10.$$

Тепер вони мають загальний ключ $Z = Z_{AB} = Z_{BA} = 10$, який не передавався по каналу зв'язку.

10.3. ЕЛЕМЕНТИ ТЕОРІЇ ЧИСЕЛ

Багато криптографічних алгоритмів базуються на результатах класичної теорії чисел. Розглянемо необхідний мінімум з цієї теорії. Класичні *теореми Ферма*, *Ейлера* і ряд інших результатів із теорії чисел будуть дані без доведень, які можуть бути знайдені майже в будь-якому підручнику з теорії чисел (див., наприклад, [14]).

Визначення 10.2. Ціле позитивне число p називається *простим*, якщо воно не ділиться ні на яке інше число, крім самого себе й одиницю.

Приклад 10.3. Числа *11* і *23* — прості; числа *27* і *33* — складені (*27* ділиться на *3* і на *9*, *33* ділиться на *3* і на *11*).

Теорема 10.3 (основна теорема арифметики). Будь-яке ціле позитивне число може бути представлене у вигляді добутку простих чисел, причому єдиним чином.

Приклад 10.4. 27 = 3·3·3, 33 = 3·11.

Визначення 10.3. Два числа називаються *взаємно простими*, якщо вони не мають жодного спільного дільника крім одиниці.

Приклад 10.5. Числа *27* і *28* взаємно прості (у них немає спільних дільників крім одиниці), числа *27* і *33* — ні (у них є спільний дільник *3*).

Визначення 10.4 (*функція Ейлера*). Нехай дано ціле число $n \geq 1$. Значення функції Ейлера $\varphi(n)$ дорівнює кількості чисел у ряду *1*, *2*, *3*, … , $n-1$, взаємно простих із n.

Приклад 10.6.

$\varphi(10)$	$\varphi(12)$
1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
$\varphi(10) = 4$	$\varphi(12) = 4$

Тут підкреслено подвійною лінією числа, невзаємно прості з аргументом n.

Твердження 10.3. Якщо p — просте число, то $\varphi(p) = p-1$.

Доведення

У ряду *1, 2, 3,... , p–1* усі числа взаємно прості з p, оскільки p — просте число й за визначенням не ділиться ні на яке інше число.

Твердження 10.4. Нехай p і q — два різних простих числа ($p \neq q$). Тоді

$$\varphi(p \cdot q) = (p-1) \cdot (q-1).$$

Доведення

У ряду *1*, *2*,…, $p \cdot g - 1$ невзаємно прості з $p \cdot q$ будуть числа

$$p,\ 2 \cdot p,\ 3 \cdot p,\ \ldots,\ (q-1) \cdot p \quad \text{і} \quad q,\ 2 \cdot q,\ 3 \cdot q,\ \ldots,\ (p-1) \cdot q.$$

Усього таких чисел буде $(q-1)+(p-1)$. Отже, кількість чисел, взаємно простих з $p \cdot q$, буде [24]

$$p \cdot q - 1 - (p-1) - (q-1) = p \cdot q - q - p + 1 = (p-1) \cdot (q-1).$$

Теорема 10.6 (теорема Ферма). Нехай p — просте число і $0<a<p$. Тоді

$$a^{p-1} \bmod p = 1.$$

Приклади 10.7.

$p = 13, a = 2;$

$$2^{12} \bmod 13 = (2^2)^2 \cdot ((2^2)^2)^2 \bmod 13 = 3 \cdot 9 \bmod 13 = 1,$$

$p = 11, a = 10;$

$$10^{10} \bmod 11 = 10^2 \cdot ((10^2)^2)^2 \bmod 11 = 1 \cdot 1 \bmod 11 = 1.$$

$P = 10, a = 10;$

$$10^9 \bmod 11 = 10 \cdot ((10^2)^2)^2 \bmod 11 = 10 \cdot 1 \bmod 11 = 10 \neq 1.$$

Теорема 10.7 (Теорема Ейлера). Нехай a і b — взаємно прості числа. Тоді

$$a^{\varphi(b)} \bmod b = 1.$$

Теорема Ферма є окремим випадком теореми Ейлера, коли b — просте число.

Приклад 10.8.

Нехай $a = 5$ і $b = 12$. Раніше (у прикладі 10.6) було визначено

$$\varphi(b) = \varphi(12) = 4.$$

Тоді

$$a^{\varphi(b)} \bmod b = 5^4 \bmod 12 = (5^2)^2 \bmod 12 = (-1)^2 \bmod 12 = 1.$$

Нехай $a = 2$ і $b = 21$. Обчислюючи $\varphi(b)$, отримаємо

$$\varphi(b) = \varphi(21) = 12,$$

$$a^{\varphi(b)} \bmod b = 2^{12} \bmod 21 = 2^4 \cdot (2^4)^2 \bmod 21 = 16 \cdot 4 \bmod 21 = 1.$$

Нам знадобиться ще одна теорема, близька до теореми Ейлера.

Теорема 10.8. Якщо p і q — прості числа, $p \neq q$ і k — довільне ціле число, то

$$a^{k \cdot \varphi(p \cdot q)+1} \bmod (p \cdot q) = a. \qquad (10.12)$$

Приклад 10.9. Візьмемо $p = 5$, $q = 7$. Тоді $p \cdot q = 35$, а функція Ейлера — $\varphi(35) = 4 \cdot 6 = 24$. Розглянемо випадок $k = 2$, тобто будемо підносити числа до степеня $k \cdot \varphi(p \cdot q) + 1 = 2 \cdot 24 + 1 = 49$.

Нехай $a = 9$. Як результат отримаємо

$$a^{k\cdot\varphi(p\cdot q)+1} \bmod (p\cdot q) = 9^{49} \bmod 35 = 9.$$

Нехай $a = 23$. Як результат отримаємо

$$a^{k\cdot\varphi(p\cdot q)+1} \bmod (p\cdot q) = 23^{49} \bmod 35 = 23.$$

Це не дивно, тому що кожне з чисел 9 і 23 взаємно просте з модулем 35, і за теоремою Ейлера $9^{24} \bmod 35 = 1$, $23^{24} \bmod 35 = 1$. Проте теорема 10.8 залишається правильною і для таких чисел:

$$10^{49} \bmod 35 = 10, \quad 28^{49} \bmod 35 = 28,$$

у той час як теорема Ейлера для них не може бути застосована (кожне з чисел 10 і 28 не взаємно просте з модулем 35 і $10^{24} \bmod 35 = 15$, $28^{24} \bmod 35 = 21$).

Визначення 10.5. Нехай a і b — два цілих позитивних числа. Найбільший загальний дільник чисел a і b є найбільше число c, яке ділить і a, і b:

$$c = gcd(a, b).$$

Приклад 10.10.

$$gcd(10,15) = 5; \quad gcd(8,28) = 4.$$

У даному прикладі значення числа a дорівнює 10 (8), числа b – 15 (28), а c – 5 (4).

Для знаходження найбільшого загального дільника можна використовувати алгоритм, відомий як алгоритм Евкліда.

Алгоритм 10.1. Алгоритм Евкліда (використовується псевдокод):

ВХІД: Позитивні цілі числа a, b, $a \geq b$.

```
1. while (b! = 0)
2. {
       r = a mod b; a = b; b = r;
3. }
```

ВИХІД: Найбільший загальний дільник a.

Приклад 10.11. Покажемо, як за допомогою алгоритму Евкліда обчислюється $gcd(28,8)$:

a:	28	8	4
b:	8	4	0
r:	4	0	

.

Тут кожний стовпець являє собою чергову ітерацію алгоритму. Процес продовжується до тих пір, поки b не стане таким, що дорівнює нулю. Тоді в значенні змінної a міститься відповідь (*4*).

Для багатьох криптографічних систем актуальний так званий узагальнений алгоритм Евкліда, з яким пов'язана така теорема:

Теорема 10.9. Нехай a і b — два цілих позитивних числа. Тоді існують цілі (необов'язково позитивні) числа x і y, такі, що

$$a \cdot x + b \cdot y = gcd(a, b). \qquad (10.13)$$

Узагальнений алгоритм Евкліда призначений для відшукування $gcd(a, b)$ і x, y, які задовольняють (10.13). Уведемо три рядки $U = (u_1, u_2, u_3)$, $V = (v_1, v_2, v_3)$ і $T = (t_1, t_2, t_3)$.

Тоді алгоритм записується таким чином:

Алгоритм 10.2. Узагальнений алгоритм Евкліда (використовується псевдокод):

ВХІД: Позитивні цілі числа a, b, $a \geq b$.

```
1.   U = { a, 1, 0 };  V = { b, 0, 1 }.
2.   while ( v1 != 0 )
3.   {
         q = u1 div v1;
         T = { u1 div v1; u2 - q·v2 ; u3 - q·v3 };
         U = V; V = T;
4.   }
5.   U = { gcd(a, b); x; y }
```

ВИХІД: $gcd(a, b)$; x; y, які задовольняють (10.13).

Результат міститься в рядку U.

Операція *div* в алгоритмі — це цілочисельне ділення

$$a \; div \; b = [\, a / b \,].$$

Доведення коректності алгоритму 10.2 може бути знайдено в [14, 24].

Приклад 10.12. Нехай $a = 28$, $b = 19$. Знайдемо числа x і y, що задовольняють (10.13).

U				*28*	*1*	*0*	
V	*U*			*19*	*0*	*1*	
T	*V*	*U*		*9*	*1*	*–1*	*q = 1*
	T	*V*	*U*	*1*	*–2*	*3*	*q = 2*
		T	*V*	*0*	*19*	*–28*	*q = 9*

Пояснимо представлену схему. Спочатку в рядок U записуються числа (*28,1,0*), а в рядок V — числа (*19,0,1*) (це перші два рядки на схемі). Обчислюється рядок T (третій рядок у схемі). Після цього як рядок U береться другий рядок у схемі, а як V — третій, і знову обчислюється рядок T (четвертий рядок у схемі). Цей процес продовжується до тих пір, поки перший елемент рядка V не стане таким, що дорівнює нулю. Тоді передостанній рядок у схемі містить відповідь. У нашому випадку $gcd(28,19) = 1$, $x = -2$, $y = 3$. Виконаємо перевірку: $28 \cdot (-2) + 19 \cdot 3 = 1$.

Розглянемо одне важливе застосування узагальненого алгоритму Евкліда. У багатьох задачах криптографії для заданих чисел c і m потрібно знаходити таке число $d < m$, що

$$c \cdot d \bmod m = 1. \tag{10.14}$$

Зазначимо, що таке d існує тоді й тільки тоді, коли числа c і m взаємно прості.

Визначення 10.6. Число d, що задовольняє (10.14), називається *мультиплікативною інверсією c* за модулем m і часто позначається $c^{-1} \bmod m$.

Дане позначення для мультиплікативної інверсії досить природне, оскільки можна тепер переписати (10.14) у вигляді

$$c \cdot c^{-1} \bmod m = 1.$$

Множення на c^{-1} відповідає діленню на c за обчислень за модулем m. За аналогією можна ввести довільні негативні ступені за обчислень за модулем m:

$$c^{-e} \bmod m = (c^{e})^{-1} \bmod m = (c^{-1})^{e} \pmod m.$$

Приклад 10.13. $3 \cdot 4 \bmod 11 = 1$, тому число 4 — це мультиплікативна інверсія числа 3 за модулем 11. Можна записати $3^{-1} \bmod 11 = 4$. Число $5^{-2} \bmod 11$ може бути знайдено двома способами:

$$5^{-2} \bmod 11 = (5^{2} \bmod 11)^{-1} \bmod 11 = 3^{-1} \bmod 11 = 4,$$

$$5^{-2} \bmod 11 = (5^{-1} \bmod 11)^{2} \bmod 11 = 9^{2} \bmod 11 = 4.$$

Обчислюючи другим способом, використовують рівність

$$5^{-1} \bmod 11 = 9.$$

Дійсно, $5 \cdot 9 \bmod 11 = 45 \bmod 11 = 1$.

Покажемо, як можна обчислити мультиплікативну інверсію за допомогою узагальненого алгоритму Евкліда. Рівність (10.14) означає, що для деякого цілого k

$$c \cdot d - k \cdot m = 1. \qquad (10.15)$$

Враховуючи, що c і m взаємно прості, перепишемо (10.15) у вигляді

$$m \cdot (-k) + c \cdot d = gcd(m, c) = 1, \qquad (10.16)$$

що повністю відповідає (10.13), тут тільки по-іншому позначені змінні. Тому, щоб обчислити $c^{-1} \bmod m$, тобто знайти число d, потрібно просто використовувати узагальнений алгоритм Евкліда для розв'язання рівняння (10.16). Зауважимо, що значення змінної k нас не цікавить, тому можна не обчислювати другий елемент рядків U, V, T. Крім того, якщо число d виходить негативним, то слід додати до нього m, оскільки за визначенням число $a \bmod m$ береться з множини $\{0,1, ..., m-1\}$.

Приклад 10.14. Обчислимо $7^{-1} \bmod 11$. Використовуємо таку саму схему запису обчислень, як у прикладі 10.12:

U					11	0	
V	U				7	1	
T	V	U			4	-1	$q = 1$
	T	V	U		3	2	$q = 1$
		T	V	U	1	-3	$q = 1$
			T	V	0	11	$q = 3$

Отримуємо $d = -3$ і $d \bmod 11 = (11 - 3) \bmod 11 = 8$, тобто

$$7^{-1} \bmod 11 = 8.$$

Перевіримо результат: $7 \cdot 8 \bmod 11 = 56 \bmod 11 = 1$.

Однією з найважливіших операцій у криптографії з відкритими ключами є операція піднесення до степеня за модулем. Ідея побудови ефективного алгоритму піднесення до степеня була раніше проілюстрована за допомогою (10.5) і (10.6). Розглянутий алгоритм можна реалізувати й без зберігання в пам'яті ряду чисел (10.5). Дамо опис цього алгоритму у формі, придатній для безпосередньої програмної реалізації. У назві алгоритму відображений той факт, що біти показника степеня проглядаються справа-наліво, тобто від молодшого до старшого.

Алгоритм 10.3. Піднесення до степеня за модулем (справа-наліво):

ВХІД: Цілі числа a, $x = (x_t x_{t-1} \ldots x_0)_2$, p.
ВИХІД: Число $y = a^x \bmod p$.
1. $y \leftarrow 1$; $s \leftarrow a$;
2. FOR $i = 0,1,\ldots,t$ DO
3. IF $x_i = 1$ THEN $y \leftarrow y \cdot s \bmod p$;
4. $s \leftarrow s \cdot s \bmod p$
5. RETURN y

Щоб показати, що за представленим алгоритмом дійсно обчислюється y згідно з (10.6), запишемо степені змінних після кожної ітерації циклу. Нехай $x = 100 = (1100100)_2$, як у прикладі 10.1, тоді

i:	0	1	2	3	4	5	6
x_i:	0	0	1	0	0	1	1
y:	1	1	a^4	a^4	a^4	a^{36}	a^{100}
s:	a^2	a^4	a^8	a^{16}	a^{32}	a^{64}	a^{128}.

У деяких ситуаціях більш ефективним виявляється наступний алгоритм, в якому біти показника степеня проглядаються зліва-направо, тобто від старшого до молодшого.

Алгоритм 10.4. Піднесення до степеня за модулем (зліва-направо):

ВХІД: Цілі числа a, $x = (x_t x_{t-1} \ldots x_0)_2$, p.
ВИХІД: Число $y = a^x \bmod p$.
1. $y \leftarrow 1$;
2. FOR $i = t, t-1, \ldots, 0$ DO
3. $y \leftarrow y \cdot y \bmod p$;
4. IF $x_i = 1$ THEN $y \leftarrow y \cdot a \bmod p$;
5. RETURN y.

Щоб переконатися в тому, що алгоритм 10.4 обчислює те саме, що й алгоритм 10.3, запишемо степінь змінної y після кожної ітерації циклу для $x = 100$:

i:	6	5	4	3	2	1	0
x_i:	1	1	0	0	1	0	0
y:	a	a^3	a^6	a^{12}	a^{25}	a^{50}	a^{100}.

Наведених у даному розділі відомостей із теорії чисел буде достатньо для опису основних криптографічних алгоритмів і методів.

10.4. КРИПТОГРАФІЧНА СИСТЕМА ШАМІРА

Ця криптографічна система, запропонована Шаміром, була першою, що дозволяє організувати обмін секретними повідомленнями по відкритій лінії зв'язку для осіб, які не мають жодних захищених каналів і секретних ключів і, можливо, ніколи не бачили один одного. Нагадаємо, що криптосистема Діффі–Хеллмана дозволяє сформувати тільки секретне слово, а передача повідомлення потребує використання деякої криптосистеми, де це слово буде використовуватися як ключ.

Перейдемо до опису системи. Нехай є два абоненти A і B, з'єднані лінією зв'язку. A хоче передати повідомлення M абоненту B так, щоб ніхто не взнав його зміст. A вибирає випадкове велике просте число p і відкрито передає його B. Потім A вибирає два числа e_A і d_A, такі, що

$$e_A \cdot d_A \bmod (p-1) = 1. \qquad (10.17)$$

Ці числа A тримає в таємниці й передавати не буде. B теж вибирає два числа e_B і d_B, такі, що

$$e_B \cdot d_B \bmod (p-1) = 1, \qquad (10.18)$$

і тримає їх у таємниці.

Після цього A передає своє повідомлення M, використовуючи триступеневий протокол. Якщо $M < p$ (M розглядається як число), то повідомлення M передається відразу, якщо ж $M > p$, то повідомлення представляється у вигляді $m_1, m_2, \ldots, m_t$, де всі $m_i < p$, і потім передаються послідовно $m_1, m_2, \ldots, m_t$. При цьому для шифрування кожного m_i краще вибирати випадково нові пари (e_A, d_A) і (e_B, d_B), у противному випадку надійність системи знижується. У наш час така криптографічна система, як правило, використовується для передачі чисел, наприклад, секретних ключів, значення яких менше за p. Отже, будемо розглядати тільки випадок $M < p$. Дамо опис протоколу.

Крок 1. A обчислює число

$$x_1 = M^{e_A} \bmod p, \qquad (10.19)$$

де M — повідомлення, яке передається,
і пересилає x_1 до B.

Крок 2. B, отримавши x_1, обчислює число

$$x_2 = x_1^{e_B} \bmod p \qquad (10.20)$$

і передає x_2 до A.

Крок 3. A обчислює число

$$x_3 = x_2^{d_A} \bmod p \tag{10.21}$$

і передає його *B*.

Крок 4. B, отримавши x_3, обчислює число

$$x_4 = x_3^{d_B} \bmod p\,. \tag{10.22}$$

Схему криптографічної системи Шаміра, яка показує процеси шифрування даних при їх передачі від абонента *A* абонентові *B*, надано на рис. 10.2.

Схема криптографічної системи Шаміра, яка показує процеси шифрування даних при їх передачі від абонента *B* абонентові *A*, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент *B*.

Твердження 10.5 (властивості *протоколу Шаміра*):

1) $x_4 = M$, тобто внаслідок реалізації протоколу від *A* до *B* дійсно передається вихідне повідомлення;

2) зловмисник не може дізнатися, яке повідомлення було передане.

Доведення

Спочатку зауважимо, що будь-яке ціле число $c \geq 0$ може бути представлено у вигляді

$$c = k\cdot(p-1) + r,$$

де $r = c \bmod (p-1)$.

Тому на підставі теореми Ферма ($x^{p-1} \bmod p = 1$)

$$\begin{aligned} x^c \bmod p &= x^{k\cdot(p-1)+r} \bmod p = (1^k \cdot x^r) \bmod p = \\ &= x^{c \bmod (p-1)} \bmod p. \end{aligned} \tag{10.23}$$

Справедливість першого пункту твердження випливає з ланцюжка рівностей:

$$\begin{aligned} x_4 = x_3^{d_B} \bmod p = (x_2^{d_A})^{d_B} \bmod p = (x_1^{e_B})^{d_A\cdot d_B} \bmod p = \\ = (M^{e_A})^{e_B\cdot d_A\cdot d_B} \bmod p = M^{e_A\cdot e_B\cdot d_A\cdot d_B} \bmod p = \\ = M^{(e_A\cdot e_B\cdot d_A\cdot d_B) \bmod (p-1)} \bmod p = M. \end{aligned}$$

Передостання рівність випливає з (10.23), а остання виконується завдяки (10.17) і (10.18).

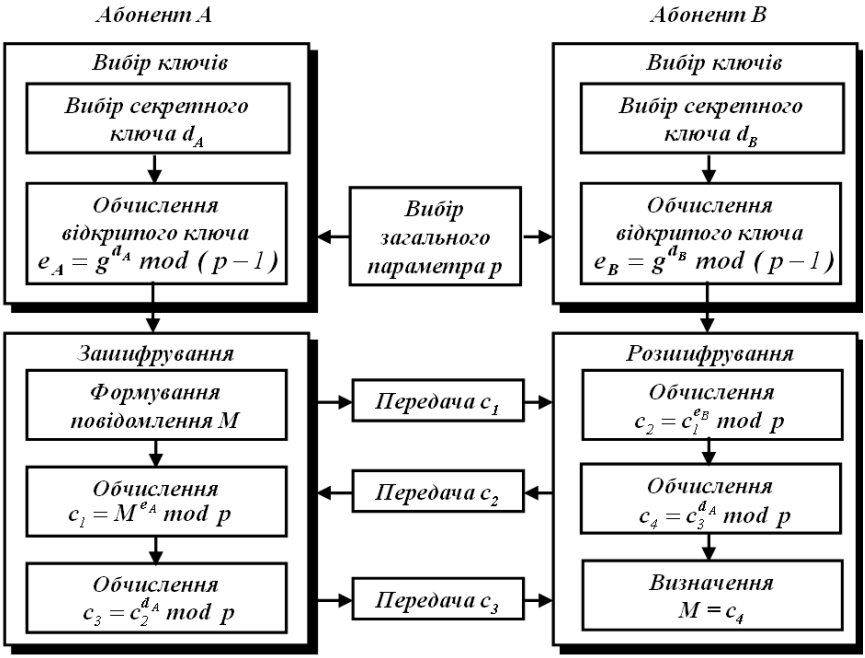


Рис. 10.2. Схема криптографічної системи Шаміра

Доведення другого пункту твердження засноване на припущенні, що для зловмисника, який намагається визначити M, не існує стратегії більш ефективної, ніж така: спочатку він обчислює e_B з (10.20), потім знаходить d_B і, нарешті, обчислює $x_4 = M$ за (10.22). Але для здійснення цієї стратегії зловмисник повинен розв'язати задачу дискретного логарифмування (10.20), що майже неможливо за великих p.

Опишемо метод знаходження пар e_A, d_A і e_B, d_B, які задовольняють (10.17) і (10.18). Достатньо описати тільки дії для абонента A, оскільки дії для B абсолютно аналогічні. Число e_A вибираємо випадково так, щоб воно було взаємно простим з $p–1$ (пошук доцільно вести серед непарних чисел, оскільки $p–1$ парне). Потім обчислюємо d_A за допомогою узагальненого алгоритму Евкліда, як це було пояснено в підрозділі 10.3.

Приклад 10.15. Нехай A хоче передати B повідомлення $M = 10$. A вибирає $p = 23$, $e_A = 7$ ($gcd(7, 22) = 1$) і обчислює $d_A = 19$. Аналогічно, B вибирає параметри $e_B = 5$ (взаємно просте з 22) і $d_B = 9$. Переходимо до протоколу Шаміра:

Крок 1. x_1 згідно з (10.19) дорівнює $x_1 = 10^7 \bmod 23 = 14$.

Крок 2. x_2 згідно з (10.20) дорівнює $x_2 = 14^5 \bmod 23 = 15$.

Крок 3. x_3 згідно з (10.21) дорівнює $x_3 = 15^{19} \bmod 23 = 19$.

Крок 4. x_4 згідно з (10.22) дорівнює $x_4 = 19^9 \bmod 23 = 10$.

Отже, B отримав передане повідомлення $M = 10$.

10.5. КРИПТОГРАФІЧНА СИСТЕМА ЕЛЬ-ГАМАЛЯ

Нехай є абоненти A, B, C, ..., які хочуть передавати один одному зашифровані повідомлення, не маючи ніяких захищених каналів зв'язку. Розглянемо криптографічну систему, запропоновану *Ель-Гамалем*, яка вирішує цю задачу, використовуючи, на відміну від криптосистеми Шаміра, тільки одне пересилання повідомлення. Фактично, тут 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 криптосистеми.

Для всієї групи абонентів вибираються деяке велике просте число p і число g, такі, що різні степені g різні числа за модулем p (див. підрозділ 10.2). Числа p і g передаються абонентам у відкритому вигляді (вони можуть використовуватися всіма абонентами мережі).

Потім кожний абонент групи обирає своє секретне число d_i, яке задовольняє вимоги

$$1 < d_i < p-1$$

і обчислює відповідне йому відкрите число e_i

$$e_i = g^{d_i} \bmod p. \qquad (10.24)$$

Як результат складемо табл. 10.3.

Таблиця 10.3

Ключі користувачів у криптографічної 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 Шаміра, що повідомлення представлено у вигляді числа $M < p$.

Крок 1. Абонент *A* формує випадкове число k ($1 < k < p–2$) і обчислює числа

$$r = g^k \bmod p; \tag{10.25}$$

$$c = (M \cdot e_B^{\,k}) \bmod p \tag{10.26}$$

та передає пару чисел (r, c) абоненту *B*.

Крок 2. Абонент *B*, отримавши (r, c), обчислює

$$M' = (c \cdot r^{p-1-d_B}) \bmod p. \tag{10.27}$$

Схему криптографічної системи Ель-Гамаля, яка показує процеси шифрування даних при їх передачі від абонента *A* абонентові *B*, надано на рис. 10.3.

Твердження 10.6 (властивості криптографічної системи Ель-Гамаля):

1) абонент *B* отримав повідомлення, тобто $M' = M$;

2) зловмисник, знаючи p, g, e_B, r і c, не може обчислити *M*.

Доведення

Підставимо в (10.27) значення c з (10.26)

$$M' = (M \cdot e_B^{\,k} \cdot r^{p-1-d_B}) \bmod p.$$

Тепер замість r підставимо (10.25), а замість e_B — (10.24)

$$M' = (M(g^{d_B})^k (g^k)^{p-1-d_B}) \bmod p =$$

$$= (M \cdot g^{d_B \cdot k + k \cdot (p-1) - k \cdot d_B}) \bmod p = (M \cdot g^{k \cdot (p-1)}) \bmod p.$$

За теоремою Ферма

$$g^{k \cdot (p-1)} \bmod p = 1^k \bmod p = 1,$$

звідси отримуємо першу частину твердження.

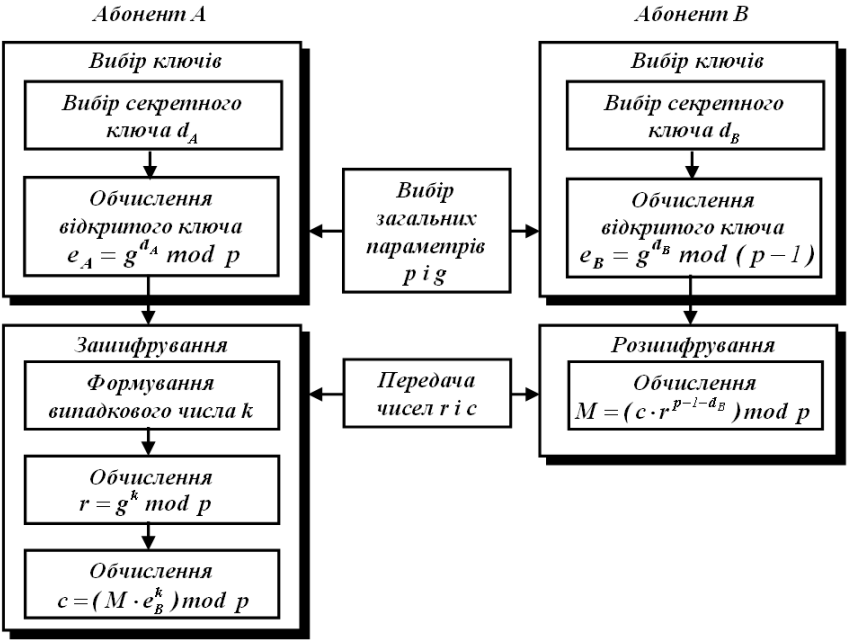


Рис. 10.3. Схема криптографічної системи Ель-Гамаля

Для доведення другої частини зауважимо, що зловмисник не може обчислити k у рівності (10.25), оскільки це завдання дискретного логарифмування. Отже, він не може обчислити M у рівності (10.26), оскільки M було помножено на невідоме йому число. Зловмисник також не може відтворити дії законного одержувача повідомлення (абонента B), оскільки йому не відоме секретне число c_B (обчислення c_B на підставі (10.24) — також задача дискретного логарифмування).

Схема криптографічної системи Ель-Гамаля, яка показує процеси шифрування даних при їх передачі від абонента B абонентові A, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент B.

Приклад 10.16. Передамо повідомлення $M = 15$ від абонента A абоненту B. Виберемо параметри аналогічно тому, як це було зроблено в прикладі 10.2. Візьмемо $p = 23$, $g = 5$. Нехай абонент B вибрав для себе секретне число $d_B = 13$ і обчислив за (10.24)

$$e_B = g^{d_B} \bmod p = 5^{13} \bmod 23 = 21.$$

Абонент *A* вибирає випадково число *k*, наприклад *k = 7*, і обчислює за (10.25), (10.26):

$$r = g^k \bmod p = 5^7 \bmod 23 = 17,$$

$$c = (M \cdot e_B^{\,k}) \bmod p = (15 \cdot 21^7) \bmod 23 = (15 \cdot 10) \bmod 23 = 12.$$

Тепер абонент *A* посилає абоненту *B* зашифроване повідомлення у вигляді пари чисел (*r, c*) = (*17, 12*).

Абонент *B* обчислює за (10.27)

$$M' = (c \cdot r^{p-1-d_B}) \bmod p = (12 \cdot 17^{23-1-13}) \bmod 23 =$$

$$= (12 \cdot 17^9) \bmod 23 = (12 \cdot 7) \bmod 23 = 15.$$

З цього видно, що абонент *B* зміг розшифрувати передане повідомлення.

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).

10.6. КРИПТОГРАФІЧНА СИСТЕМА *RSA*

Криптографічна система *RSA* названа на честь його розробників Рівеста, Шаміра й Адлемана. Ця криптографічна система до цих пір є однією з найбільш широко використовуваних [16].

Як вказано вище криптографічна система Шаміра повністю вирішує задачу обміну повідомленнями, закритими для прочитання, у випадку, коли абоненти можуть користуватися тільки відкритими каналами зв'язку. Однак при цьому повідомлення пересилається три рази від одного абонента до іншого, що є недоліком. Криптографічна система Ель-Гамаля дозволяє вирішити ту саму задачу за одне пересилання даних, але обсяг зашифрованих даних у два рази перевищує обсяг повідомлення, яке передається. Криптографічна система *RSA* не

має подібних недоліків. Цікаво те, що вона базується на інший односторонньої функції, відмінній від дискретного логарифма. Крім того, тут ми зустрінемося зі ще одним винаходом сучасної криптографії — *односторонньою функцією* з "*лазівкою*" (*trapdoor function*).

Ця система базується на таких двох фактах із теорії чисел:

- завдання перевірки числа на простоту є порівняно легкою;
- завдання розкладання чисел вигляду $n = p \cdot q$ (p і q — прості числа) на множники є надто складною, якщо ми знаємо тільки n, а p і q — великі числа (це так звана задача факторизації).

Нехай у криптографічній системі є абоненти A, B, C,... Кожний абонент вибирає випадково два великих простих числа p та q. Потім він обчислює число

$$n = p \cdot q. \tag{10.28}$$

Число n є відкритою інформацією, доступною іншим абонентам.

Після цього абоненти обчислюють число $\varphi(n) = (p-1) \cdot (q-1)$ і вибирають деякі числа $e < \varphi(n)$, взаємно прості з $\varphi(n)$, і за узагальненим алгоритмом Евкліда знаходять числа d, такі, що

$$e \cdot d \bmod \varphi(n) = 1. \tag{10.29}$$

Усю інформацію, пов'язану з абонентами і є їх відкритими й секретними ключами, надано в табл. 10.4.

Таблиця 10.4

Ключі користувачів у системі *RSA*

Абонент	Відкритий ключ	Секретний ключ
A	e_A, n_A	d_A
B	e_B, n_B	d_B
C	e_C, n_C	d_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 абоненту B).

Крок 1. Абонент A шифрує повідомлення за формулою

$$C = (M^{e_B}) \bmod n_B, \tag{10.30}$$

використовуючи відкриті параметри абонента B, і пересилає e по відкритій лінії.

Крок 2. Абонент *B*, який отримав зашифроване повідомлення, обчислює

$$M' = (C^{d_B}) \bmod n_B . \qquad (10.31)$$

Схему криптографічної системи *RSA*, яка показує процеси шифрування даних при їх передачі від абонента *A* абонентові *B*, надано на рис. 10.4.

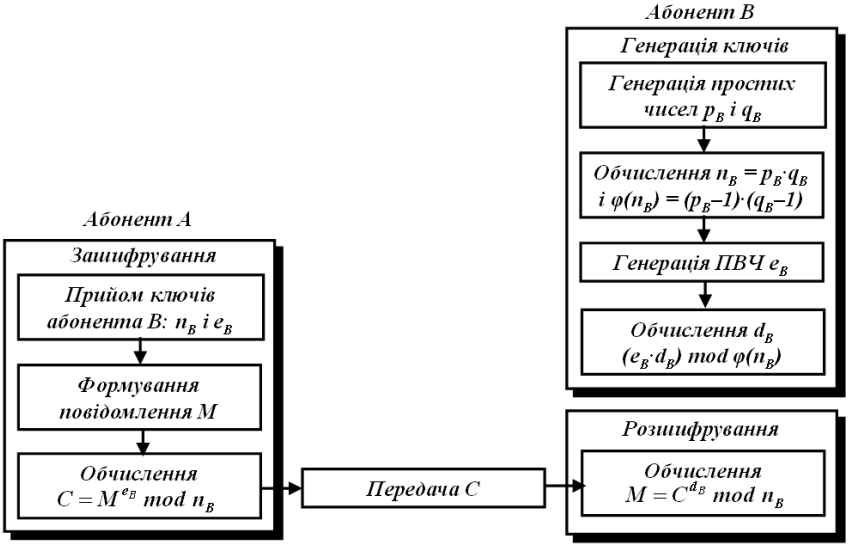


Рис. 10.4. Схема криптографічної системи *RSA*

Схема криптографічної системи *RSA*, яка показує процеси шифрування даних при їх передачі від абонента *B* абонентові *A*, аналогічна представленій, тільки в цьому випадку ініціатором передачі повідомлення буде абонент *B*.

Твердження 10.7. Для описаного протоколу $M' = M$, тобто абонент *B* отримує від *A* повідомлення.

Доведення

За протоколом

$$M' = (C^{d_B}) \bmod n_B = (M^{e_B \cdot d_B}) \bmod n_B .$$

Рівність (10.29) означає, що для деякого k

$$e_B \cdot d_B = k \cdot \varphi_B + 1.$$

Відповідно до твердження 10.5

$$\varphi_B = (p_B - 1)\cdot(q_B - 1) = \varphi(n_B),$$

де $\varphi(n_B)$ — функція Ейлера.

Звідси, а також з теореми 10.8 випливає

$$M' = (M^{e_B \cdot d_B}) \bmod n_B = (M^{k\cdot\phi(n_B)+1}) \bmod n_B = M .$$

Твердження 10.8 (властивості протоколу *RSA*):

1) протокол забезпечує зашифрування й розшифрування повідомлень M коректно;

2) зловмисник, що перехоплює всі зашифровані повідомлення і знає всю відкриту інформацію відносно криптографічної системи, не зможе знайти повідомлення M за великих p і q.

Доведення

Перша властивість протоколу випливає з твердження 10.8. Для доказу 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, своєю чергою, йому потрібно знати p і q. Взагалі, він може знайти p і q, розклавши n на множники, однак це складне завдання (див. підрозділ 10.2). Зазначимо, що вибір великих випадкових p і q можливий за прийнятний час, тому що справедливий пункт 1.

Одностороння функція $y = x^d \bmod n$, що застосовується 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 й в інших розділах криптографії.

Зазначимо, що для схеми *RSA* важливо, щоб кожний абонент вибирав власну пару простих чисел p і q, тобто всі модулі n_A, n_B, n_C, … повинні бути різні (в іншому випадку один абонент міг би читати зашифровані повідомлення, призначені для іншого абонента). Однак цього не потрібно від другого відкритого параметра e. Параметр e може бути однаковим у всіх абонентів. Часто рекомендується вибирати $e = 3$ (за відповідного вибору p і q, див. [25]). Тоді зашифрування виконується максимально швидко, лише за два множення.

Приклад 10.17. Припустимо, абонент *A* хоче передати абоненту *B* повідомлення *M = 15*. Нехай абонент *B* вибрав такі параметри:

$$p_B = 3,\ q_B = 11,\ n_B = 33,\ e_B = 3.$$

Секретний ключ абонента *B* — d_B за допомогою узагальненого алгоритму Евкліда буде дорівнювати: $d_B = 7$.

Перевіримо: $e_B \cdot d_B \bmod \varphi(n) = 3 \cdot 7 \bmod 20 = 1$.

Зашифруємо *M* за формулою (10.30):

$$C = M^{e_B} \bmod n_B = 15^3 \bmod 33 = 15^2 \cdot 15 \bmod 33 =$$

$$= 27 \cdot 15 \bmod 33 = 9.$$

Число $C = 9$ абонент *A* передає абоненту *B* по відкритому каналу зв'язку. Тільки абонент *B* знає секретний ключ $d_B = 7$, тому він розшифровує отримане повідомлення, використовуючи (10.31):

$$M' = C^{d_B} \bmod n_B = 9^7 \bmod 33 = (9^2)^2 \cdot 9^2 \cdot 9 \bmod 33 =$$

$$= 15^2 \cdot 15 \cdot 9 \bmod 33 = 15.$$

Отже, абонент *B* розшифрував повідомлення абонента *A*.

Розглянуту систему неможливо зламати за великих *p* і *q*, але вона має такий недолік: абонент *A* передає повідомлення абоненту *B*, використовуючи відкриту інформацію абонента *B* (числа n_B і e_B). Зловмисник не може читати повідомлення, призначені для абонента *B*, проте він може передати повідомлення для абонента *B* від імені абонента *A*. Уникнути цього можна, використовуючи більш складні протоколи, наприклад, такий:

абонент *A* хоче передати абоненту *B* повідомлення *M*: спочатку абонент *A* обчислює число $C = M^{d_A} \bmod n_A$; зловмисник не може цього зробити, тому що d_A таємне; потім абонент *A* обчислює число $F = C^{e_B} \bmod n_B$ і передає *F* абоненту *B*; абонент *B* отримує *F* і обчислює послідовно числа:

$$U = F^{d_B} \bmod n_B \text{ і } W = U^{e_A} \bmod n_A.$$

Як результат абонент *B* отримує повідомлення *W = M*. Як і в криптографічній системі *RSA*, зловмисник не може прочитати передане повідомлення, але тут, на відміну від *RSA*, він не може також послати повідомлення від імені абонента *A* (оскільки не знає секретного ключа d_A).

Тут виникає нова ситуація: абонент *B* знає, що повідомлення надійшло від абонента *A*, тобто *A* ніби "*підписав*" його, зашифрувавши своїм секретним ключем d_A. Це приклад так званого електронного або цифрового підпису. Він є одним з широко використовуваних на практиці винаходів сучасної криптографії.

10.7. КРИПТОГРАФІЧНА СИСТЕМА РАБІНА

Безпека *криптографічної системи Рабіна* заснована на ускладненні пошуку квадратних коренів за модулем складеного числа. Із позиції теорії чисел ця задача аналогічна задачі факторизації модуля. Розглянемо одну з реалізацій системи. Уявімо, що користувачі *A* і *B* бажають обмінюватися зашифрованими повідомлення між собою. Тоді за схемою Рабіна:

1. Користувачі A і B генерують для себе відкриті та закриті ключі, для цього:

- кожний із користувачів вибирає по два великих простих числа p_A (p_B) и q_A (q_B) (далі алгоритм розшифрування буде особливо простим, якщо ці числа порівняні із числом *3* за модулем *4*, тобто $p_A \bmod 4 = 3$, $q_A \bmod 4 = 3$, $p_B \bmod 4 = 3$ і $q_B \bmod 4 = 3$). Числа p_A (p_B) і q_A (q_B) будуть закритими ключами системи (користувачів *A* і *B*);
- кожний із користувачів визначає свої відкриті ключі: $n_A = p_A \cdot q_A$ і $n_B = p_B \cdot q_B$;
- користувачів *A* і *B* обмінюються відкритими ключами.

2. Перед зашифруванням повідомлень абоненти розбивають їх на блоки m_i, довжина яких менше відкритого ключа n_A (якщо відбувається передача повідомлення від користувача *B* користувачеві *A*) або n_B (якщо відбувається передача повідомлення від користувача *A* користувачеві *B*). Припустимо, що $M < n_A$ або $M < n_B$.

3. Для зашифрування повідомлень абоненти *A* або *B* перетворюють дані згідно з рівнянням зашифрування

$$C = E_n(M) = M^2 \bmod n \qquad (10.32)$$

і після цього пересилають *C* іншому абоненту *B* або *A*.

4. Із виразу (10.32) витікає, що *M* — корінь квадратний із числа *C* за модулем *n*. Якщо $gcd(C,n) = 1$ (числа *C* і *n* взаємно прості), то кожний квадратичний залишок має в мультиплікативній групі залишків Z_n^* рівно чотири різних корені [14]. Оскільки отримувач

знає множники p і q числа n, то він розв'язує два порівняння. Передусім

$$r_1 = C^{(p+1)/4} \bmod p; \quad r_2 = -C^{(p+1)/4} \bmod p;$$
$$r_3 = C^{(q+1)/4} \bmod q; \quad r_4 = -C^{(q+1)/4} \bmod q; \tag{10.33}$$

$$a = q \cdot (q^{-1} \bmod p); \quad b = p \cdot (p^{-1} \bmod q). \tag{10.34}$$

Чотири можливих корені із числа C за модулем n — це

$$M_1 = (a \cdot r_1 + b \cdot r_3) \bmod n; \quad M_2 = (a \cdot r_2 + b \cdot r_4) \bmod n;$$
$$M_3 = (a \cdot r_1 + b \cdot r_4) \bmod n; \quad M_4 = (a \cdot r_2 + b \cdot r_3) \bmod n. \tag{10.35}$$

Схему криптографічної системи Рабіна, яка показує процеси шифрування даних під час їх передачі від абонента B абонентові A, надано на рис. 10.5.

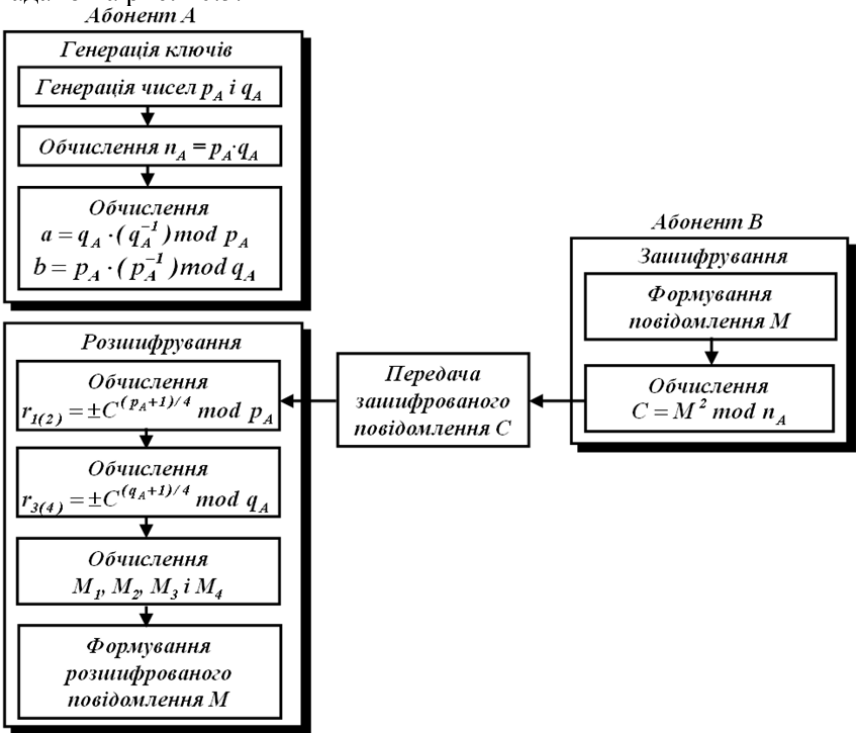


Рис. 10.5. Схема криптографічної системи Рабіна

Схема криптографічної системи Рабіна, яка показує процеси шифрування даних під час їх передачі від абонента *A* абонентові *B*, аналогічна наданій, 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. Якщо повідомлення написане звичайною мовою, то вибрати правильне *M* неважко. Але якщо шифрувалася сукупність випадкових бітів, способу визначення правильного *M* не існує.

Приклад 10.18. Згенерувати ключі та зашифрувати за допомогою криптографічної системи Рабіна повідомлення "*протокол*" при передачі його від користувача *B* користувачеві *A*.

Розв'язання

Нехай закритий ключ користувача *A* утворює пара простих чисел $p_A = 79$ і $q_A = 83$, тоді відкритий ключ — число $n_A = p_A \cdot q_A = 6557$. Кожну літеру відкритого повідомлення замінимо її номером в українській абетці

"*протокол*" => *18 19 17 21 17 13 17 14.*

Наведена числова послідовність — це відкрите повідомлення, записане в цифровій формі. Розіб'ємо послідовність на чотири блоки m_i, кожний з яких — деяке натуральне число, яке повинне бути меншим, ніж число $n_A = 6557$:

$$M \Rightarrow 1819 - 1721 - 1713 - 1714.$$

Зашифруємо відкрите повідомлення за формулою (10.32):

$$C_1 = E_n(m_1) = 1819^2 \bmod 6557 = 4033;$$
$$C_2 = E_n(m_2) = 1721^2 \bmod 6557 = 4634;$$
$$C_3 = E_n(m_3) = 1713^2 \bmod 6557 = 3390;$$
$$C_4 = E_n(m_4) = 1714^2 \bmod 6557 = 0260.$$

Як результат отримаємо зашифроване повідомлення

$$C \Rightarrow 4033 - 4634 - 3390 - 0260.$$

Приклад 10.19. Законний користувач A згенерував власну пару ключів для криптографічної системи Рабіна: закритий ключ — прості числа $p_A = 59$, $q_A = 67$, які він зберігає в таємниці, та відкритий ключ — загальнодоступне число $n_A = p_A \cdot q_A = 3953$. Розшифрувати отримане зашифроване повідомлення $C \Rightarrow 0743–2204$ від користувача B.

Розв'язання

Розшифрування — це добуття квадратного кореня із чисел $C_1 = 743$ та $C_2 = 2204$ за модулем $n_A = 3953$ і зводиться до добування коренів за простими модулями $p_A = 59$ і $q_A = 67$.

Обчислимо

$$r_1 = C_1^{(p_A+1)/4} \bmod p_A = 0743^{(59+1)/4} \bmod 59 = 25;$$

$$r_2 = -C_1^{(p_A+1)/4} \bmod p_A = -0743^{(59+1)/4} \bmod 59 = -25 \bmod 79 = 34;$$

$$r_3 = C_1^{(q_A+1)/4} \bmod q_A = 2204^{(67+1)/4} \bmod 67 = 26;$$

$$r_4 = -C_1^{(q_A+1)/4} \bmod q_A = -2204^{(67+1)/4} \bmod 67 = -26 \bmod 79 = 41.$$

Отримані розв'язки скомбінуємо між собою в системи та за китайською теоремою про залишки визначимо значення квадратних коренів. $gcd(p_A,q_A) = 1$ і за розширеним алгоритмом Евкліда

$$1 = 37 \cdot 67 + 25 \cdot 59,$$

тобто $(q_A^{-1} \cdot q_A) \bmod p_A = 1$, звідки $q_A^{-1} = 37$; $(p_A^{-1} \cdot p_A) \bmod q_A = 1$, звідки $p_A^{-1} = 25$.

Відповідно до значень q_A, q_A^{-1}, p_A і p_A^{-1}

$$a = q_A(q_A^{-1} \bmod p_A) = 67 \cdot 37 = 2479;$$

$$b = p_A(p_A^{-1} \bmod q_A) = 59 \cdot 25 = 1475.$$

Перший корінь M знайдемо з (10.33), (10.34) і (10.35), звідки

$$M_1 = (a \cdot r_1 + b \cdot r_3) \bmod n_A = 1500.$$

Другий корінь протилежний до знайденого першого кореня $M_1 = 1500$ знайдемо з (10.33), (10.34) і (10.35), звідки

$$M_2 = (a \cdot r_2 + b \cdot r_4) \bmod n_A = 2453.$$

Корені M_1 і M_2 протилежні, тобто $(M_1 + M_2) \bmod n = 0$:

$$(M_1 + M_2) \bmod n = (1500 + 2453) \bmod 3953 = 3953 \bmod 3953 = 0.$$

Третій корінь M знайдемо з (10.33), (10.34) і (10.35), звідки

$$M_3 = (a \cdot r_1 + b \cdot r_4) \bmod n_A = 3860.$$

Четвертий корінь протилежний знайденому третьому кореню $M_3 = 3860$ знайдемо з (10.33), (10.34) і (10.35), звідки

$$M_4 = (a \cdot r_2 + b \cdot r_3) \bmod n_A = 93.$$

Корені M_3 і M_4 протилежні, тобто $(M_3 + M_4) \bmod n = 0$:

$$(M_3 + M_4) \bmod n = (3860 + 93) \bmod 3953 = 3953 \bmod 3953 = 0.$$

Отже, із числа $C_1 = 0743$ добуто чотири корені за модулем *3953*: *1500*, *2453*, *3860*, *93*. Аналогічно добудемо чотири корені з другого числа $C_2 = 2204$ отриманого зашифрованого повідомлення: *2033*, *1920*, *1384*, *2569*. Лише корені *1500* і *1920* — числовий еквівалент повідомлення української абетки, потужність якої *32* літери.

Оскільки числу *15* української абетки відповідає літера "*м*", *00* — "*а*", *19* — "*р*", а *20* — "*с*", то розшифроване слово — "*марс*".

10.8. МЕТОДИ ЗЛАМУ КРИПТОГРАФІЧНИХ СИСТЕМ, ЗАСНОВАНИХ НА ДИСКРЕТНОМУ ЛОГАРИФМУВАННІ

10.8.1. Постановка завдання

Для створення надійної криптографічної системи необхідно брати до уваги ті методи зламу, які може застосувати зловмисник, і вибирати параметри криптографічної системи (зокрема довжини чисел) так, щоб зробити ці методи практично нереалізованим. У цьому пункті розглянемо два таких методи, для того щоб дати читачеві певне уявлення про цю "*таємничу*" область.

Багато криптографічних систем, які розглядаються, ґрунтуються на односторонній функції

$$y = a^x \bmod p. \qquad (10.36)$$

y можна обчислити, якщо дано a, x та p, витративши не більше, ніж $2 \cdot \log x$ операцій (твердження 10.1). Однак пошук x за відомими

a, y і p, тобто обчислення дискретного логарифма, — завдання набагато складніше.

Як уже було показано під час розгляду криптографічної системи Шаміра (див. (10.23)), на підставі теореми Ферма при піднесенні до степеня за простим модулем p показники степеня наводяться за модулем $p - 1$. Тому нам достатньо розглянути лише показники x, що задовольняють нерівність $0 \le x \le p - 1$.

Позначимо через t_y кількість операцій множення, необхідних для обчислення y в (10.1) за a, x і p, і будемо для стислості називати t_y часом обчислення. Час піднесення до степеня за алгоритмами з підрозділу 10.1 не більше $2 \cdot \log x$, причому $x < p$. Звідси

$$t_y \le 2 \cdot \log x \qquad (10.37)$$

за будь-якого показника степеня x.

Тепер перейдемо до задачі пошуку x в (10.36) за даними a та y. Спочатку оцінимо складність прямого перебору. Для цього можна було б спочатку обчислити a^1 і перевірити, чи правильна рівність $a^1 \bmod p = y$. Якщо ні, то перевіряємо $a^2 \bmod p = y$, якщо ні, то $a^3 \bmod p = y$ і так далі до a^{p-1}. У середньому потрібно $(p-1)/2$ разів множити на a й перевіряти рівність. Отже, час прямого перебору

$$t_{п.п.} \approx p/2.$$

В описуваному нижче методі "крок немовляти, крок велетня" час пошуку x істотно менше

$$t_{к.н.к.в.} \approx 2 \cdot \sqrt{p}\,,$$

а в методі обчислення порядку цей час ще менший:

$$t_{о.п.} \approx c_1 \cdot 2^{c_2 \sqrt{\log p \ \log(\log p)}}\,,$$

де c_1 і c_2 — деякі позитивні константи.

Щоб зробити порівняння більш наочним, виразимо час обчислення через довжину числа p у (10.36). Позначимо цю довжину в бітах через n. При обчисленнях за модулем p маємо $n \approx \log p$. Тому порядок трудомісткості (тобто кількості операцій) згаданих алгоритмів буде такий:

$$t_y \approx n, \quad t_{п.п.} \approx 2^{n-1}, \quad t_{к.м\ к.в.} \approx 2^{n/2}, \quad t_{о.п.} \approx 2^{c_2 \sqrt{n \cdot \log n}}\,,$$

де $\approx$ означає *пропорційно*.

Як бачимо, кількість операцій при піднесенні до степеня зростає лінійно із зростанням довжини числа n, а час розв'язання оберненої задачі різними методами зростає експоненціально або субекспоненціально (для методу обчислення порядку). Питання про існування більш швидких алгоритмів для обчислення дискретних логарифмів, як і для розв'язання інших обернених задач, що виникають у криптографічному аналізі, залишається відкритим.

10.8.2. Метод "крок немовляти, крок велетня"

У відкритій літературі цей метод був вперше описаний *Д. Шенксом*; посилання на нього відомі з 1973 р. [28]. Це був один із перших методів, який показав, що завдання обчислення дискретного логарифма може бути вирішене значно швидше, ніж методом перебору. Перейдемо до опису цього методу пошуку x у (10.36).

Крок 1. Спочатку беремо два цілих числа m і k, такі, що

$$m \cdot k > p. \tag{10.38}$$

Крок 2. Обчислимо два ряди чисел:

$$y,\ a \cdot y,\ a^2 \cdot y,\ \ldots,\ a^{m-1} \cdot y \ (\mathrm{mod}\ p); \tag{10.39}$$

$$a^m,\ a^{2 \cdot m},\ \ldots,\ a^{k \cdot m} \ (\mathrm{mod}\ p). \tag{10.40}$$

Усі обчислення проводяться за модулем p.

Крок 3. Знайдемо такі i та j, для яких виконується рівність

$$a^{i \cdot m} = a^j \cdot y. \tag{10.41}$$

Твердження 10.9. Число

$$x = i \cdot m - j \tag{10.42}$$

є розв'язком рівняння (10.36). Крім того, цілі числа i та j, які задовольняють (10.41), існують.

Доведення

Справедливість (10.42) випливає з наведеного нижче символічного ланцюга рівностей, де всі обчислення дані за модулем p, а поділ відповідає множенню на обернений елемент:

$$a^x = a^{i\ m-j} = \frac{a^{i\ m}}{a^j} = \frac{a^{i\ m} y}{a^j y} = \frac{a^{i\ m} y}{a^{i\ m}} = y\,.$$

Доведемо тепер, що числа i та j, які задовольняють (10.41), існують. Для цього зведемо всі числа вигляду (10.42) у табл. 10.5.

Таблиця 10.5

Розподіл чисел вигляду $i \cdot m - j$

	$j = 0$	$j = 1$	$j = 2$	...	$j = m - 1$
$i = 1$	m	$m - 1$	$m - 2$	...	1
$i = 2$	$2 \cdot m$	$2 \cdot m - 1$	$2 \cdot m - 2$	...	$m + 1$
...	...	...	...	...	...
$i = k$	$k \cdot m$	$k \cdot m - 1$	$k \cdot m - 2$	...	$(k - 1) \cdot m + 1$

Бачимо, що в таблиці містяться всі числа від 1 до $k \cdot m$. Отже, з (10.42) випливає, що в таблиці містяться всі числа від 1 до p. Тому, будь-який показник степеня $x < p$ буде міститися в таблиці, тобто число x, що задовольняє (10.36), може бути представлено у вигляді (10.42) і завжди знайдеться в таблиці, тому рівняння (10.41) завжди має розв'язок.

Приклад 10.20. Знайдемо розв'язок рівняння $2^x \bmod 23 = 9$, використовуючи метод "крок немовляти, крок велетня".

Виберемо m і k. Нехай $m = 6$ і $k = 4$. Бачимо, що (10.38) виконується. Обчислимо числа (10.39) і (10.40)

$9, 18, 13, 3, 6, 12$;

18.

Подальші обчислення не проводимо, тому що вже знайшлися однакові числа в (10.39) і (10.40) за $i = 1, j = 1$. За (10.42) отримуємо

$$x = i \cdot m - j = 1 \cdot 6 - 1 = 5.$$

Перевіримо: $2^5 \bmod 23 = 10$. Дійсно, $x = 5$ є розв'язком.

Пояснимо походження назви розглянутого методу. Відомо, що в криптографії p — велике просте число, отже m і k теж великі. У ряді (10.39) степінь збільшується на 1 ("крок немовляти"), а в ряді (10.40) степінь збільшується на m ("крок велетня"). Оцінимо складність цього методу.

Твердження 10.10. Час обчислень за цим методом за великих p задовольняє нерівність [25]

$$t_{к.н.к.в.} \le const \cdot \sqrt{p} \cdot log^2\, p\,. \qquad (10.43)$$

Тут йдеться про повний час обчислень, а не про кількість множень.

Доведення

Можна взяти

$$k = m = \left\lfloor \sqrt{p} \right\rfloor + 1, \tag{10.44}$$

тому, очевидно, (10.38) виконується. Тоді в (10.39) і (10.40) буде потрібно не більше $2\sqrt{p}$ операцій множення. Відомо, що для *звичайних* (*шкільних*) методів множення й ділення час обчислення результату для двох *r*-значних чисел пропорційний r^2. У нашому випадку всі числа беруться з множини $\{ 1, \ldots, p \}$, отже, $r < \log p$, і час обчислення пропорційний $\log^2 p$. Звідси відразу отримуємо час, витрачений на обчислення рядів (10.39) і (10.40). Проте не враховано всі етапи алгоритму та час, необхідний для знаходження рівних чисел у цих рядах. За великих *k* і *m* це не просте завдання. Його можна вирішити у такий спосіб: спочатку кожному числу припишемо його номер у ряду та ще один біт, в якому вказано приналежність до ряду (10.39) або (10.40), потім перетворимо обидві послідовності в список і відсортуємо (впорядкуємо за величиною). Довжина загального ряду дорівнює $k + m \approx 2 \cdot \sqrt{p}$. Для кращих методів сортування потрібно $S \cdot \log S$ операцій порівняння, де *S* — число елементів у списку (див., наприклад, [2]). У цьому випадку $S = 2 \cdot \sqrt{p}$ і, отже, потрібно

$$2\sqrt{p} \cdot \log(2 \cdot \sqrt{p}) \approx \sqrt{p} \cdot \log p$$

операцій порівняння над словами довжини $\log p$ бітів, тобто всього потрібно близько $\sqrt{p} \cdot \log^2 p$ операцій. Після сортування об'єднаного ряду його слід переглянути та знайти два рівних числа з різних рядів (10.39), (10.40), використовуючи бітову ознаку. Отже, підсумовуючи час обчислення на всіх етапах, отримуємо (10.43).

10.8.3. Алгоритм обчислення порядку

Основні ідеї *алгоритму обчислення порядку* (*index-calculus algorithm*) були відомі в теорії чисел ще з 20-х рр. XX ст. Однак тільки 1979 р. *Адлеман*, один із творців *RSA*, вказав на цей алгоритм як на засіб розв'язання рівняння (10.36) і досліджував його трудомі-

сткість. У даний час алгоритм обчислення порядку та його поліпшені варіанти дають найбільш швидкий спосіб обчислення дискретних логарифмів у рівняннях типу (10.36).

Для зручності опису алгоритму введемо таке поняття.

Визначення 10.7. Число n називається p-гладким, якщо воно розкладається лише на прості множники, менші або рівні p.

Приклад 10.21. Числа *15*, *36*, *45*, *270*, *2025* є *5*-гладкими (в їх розкладанні беруть участь тільки множники *2*, *3*, і *5*).

Перейдемо безпосередньо до опису алгоритму.

Крок 1. Формуємо множину базових множників

$$S = \{ p_1, p_2, \ldots, p_t \},$$

яка складається з перших t простих чисел (зауваження про вибір значення t буде дано нижче).

Крок 2. Задаючи послідовно значення $k = 1,2,3,\ldots$, знаходимо $t + \varepsilon$ (ε — невелике ціле число, див. нижче) p^t-гладких чисел виду $a^k \bmod p$, перевіряючи гладкість шляхом поділу на елементи множини S. Кожне зі знайдених p^t-гладких чисел записується через утворення базових множників:

$$a^k \bmod p = \prod_{i=1}^{t} p_i^{c_i} , \quad c_i \geq 0. \tag{10.45}$$

Для кожного значення k отримуємо свій набір чисел c_i.

Крок 3. Переходимо до логарифмів у (10.47):

$$k = \sum_{i=1}^{t} c_i \cdot \log_a p_i , \tag{10.46}$$

для кожного p^t-гладкого числа, знайденого на кроці 2. Ми отримали систему з $t + \varepsilon$ рівнянь виду (10.46) з t невідомими. Як невідомі тут виступають величини $\log_a p_i$, при цьому кількість рівнянь на ε більше кількості невідомих, що підвищує ймовірність отримання розв'язку системи у випадку, якщо деякі з рівнянь виявляться лінійно залежними. Розв'язуємо систему методами лінійної алгебри, проводячи всі обчислення за модулем $p-1$ (нагадаємо, що показники степеня, а отже й логарифми, наводяться за модулем $p-1$). Як результат отримуємо значення логарифмів чисел із множини S: $\log_a p_1$, $\log_a p_2$, ..., $\log_a p_t$.

Крок 4. Випадково вибираючи r, знаходимо p^t-гладке число вигляду $(y \cdot a^r)$:

$$y \cdot a^r \bmod p = \prod_{i=1}^{t} p_i^{\varepsilon_i}, \quad \varepsilon_i \geq 0. \tag{10.47}$$

Крок 5. Логарифмуючи (10.49), отримуємо кінцевий результат

$$x = \log_a y = \left(\sum_{i=1}^{t} e_i \log_a p_i - r \right) \bmod (p-1), \tag{10.48}$$

величина r віднімається від усієї суми, а не з кожного доданка.

Справедливість описаного методу досить очевидна з побудови алгоритму, а його ефективність пов'язана з таким спостереженням. Якщо вибирати навмання число з нескінченної кількості цілих чисел, то з імовірністю *1/2* воно ділиться на *2*, з імовірністю *1/3* — на *3*, з імовірністю *1/5* — на *5* і т.д. Тому можна очікувати, що в проміжку від *1* до $p-1$ існує досить багато чисел, у розкладанні яких беруть участь тільки маленькі прості множники з множини S. Саме такі числа відшукуються на кроках 2 і 4 алгоритму. Чим більше t, тобто кількість простих множників у S, тим менше невдач у ході пошуку гладких чисел відбувається на кроках 2 і 4, тобто ці кроки виконуються швидше. Проте за великих t різко збільшується трудомісткість кроку 3, коли доводиться розв'язувати систему з $t + \varepsilon$ рівнянь. Знаходження значення дає мінімальний загальний час обчислень, зазвичай може бути виконано з використанням чисельних методів. Аналітичні вирази отримати досить складно. Параметр ε приймається таким, що дорівнює невеликому цілому числу для того, щоб збільшити ймовірність існування розв'язку системи рівнянь на кроці 3. Отримана система може містити лінійно залежні рівняння (як це буде показано в наведеному нижче прикладі). Вважається, що за великих p значення ε порядку *10* гарантує існування єдиного розв'язку системи з високою ймовірністю (див. [28]). Якщо все ж таки отримана система має нескінченно багато розв'язків, то необхідно повернутися до кроку 2 і використовувати інші значення k.

Адлеман показав, що за оптимального значення t для трудомісткості алгоритму маємо

$$t_{u.n.} < c_1 \cdot 2^{(c_2+o(1))\cdot\sqrt{\log p \cdot \log(\log p)}},$$

де c_1 і c_2 — деякі позитивні константи.

Приклад 10.22. Розв'яжемо за допомогою алгоритму обчислення порядку рівняння

$$37 = 10^x \bmod 47. \tag{10.49}$$

Маємо $y = 37$, $a = 10$, $p = 47$. Візьмемо множину базових множників $S = \{ 2, 3, 5 \}$, $t = 3$, і приймемо $\varepsilon = 1$, тобто будемо будувати систему із чотирьох рівнянь. Позначимо логарифми чисел із S через u_1, u_2 і u_3 відповідно, наприклад, $u_3 = \log_{10} 5 \bmod 47$. Виконано перший крок алгоритму, перейдемо до другого.

Проведемо пошук чотирьох 5-гладких чисел:

$$\begin{aligned}
10^1 \bmod 47 &= 10 = 2\cdot 5, && \curlyvee\\
10^2 \bmod 47 &= 6 = 2\cdot 3, && \curlyvee\\
10^3 \bmod 47 &= 13 = 13, &&\\
10^4 \bmod 47 &= 36 = 2\cdot 2\cdot 3\cdot 3, && \curlyvee\\
10^5 \bmod 47 &= 31 = 31, &&\\
10^6 \bmod 47 &= 28 = 2\cdot 2\cdot 7, &&\\
10^7 \bmod 47 &= 45 = 3\cdot 3\cdot 5. && \curlyvee
\end{aligned}$$

Знайдено чотири 5-гладких числа, що відповідають степеням 1, 2, 4 і 7.

Починаємо третій крок алгоритму. Перейдемо до логарифмів і складемо систему рівнянь із рівностей, позначених на попередньому кроці символом $\curlyvee$:

$$1 = u_1 + u_3; \tag{10.50}$$

$$2 = u_1 + u_2; \tag{10.51}$$

$$4 = 2\cdot u_1 + 2\cdot u_2; \tag{10.52}$$

$$7 = 2\cdot u_2 + u_3. \tag{10.53}$$

Бачимо, що в отриманій системі рівняння (10.51) і (10.52) лінійно залежні, отже, ми не даремно знайшли четверте гладке число. Щоб розв'язати систему, віднімемо (10.50) від (10.51). Отримаємо

$$1 = u_2 - u_3. \tag{10.54}$$

Додамо (10.56) до (10.55) та отримаємо

$$8 = 3 \cdot u_2 - u_3. \tag{10.55}$$

З (10.50) безпосередньо знаходимо u_2:

$$u_2 = (8/3) \bmod 46 = 8 \cdot 3^{-1} \bmod 46 = 8 \cdot 31 \bmod 46 = 18.$$

Ми можемо зробити перевірку, обчисливши $10^{18} \bmod 47 = 3$, отже, u_2 — дійсно логарифм числа 3. Тепер з (10.54) знаходимо u_3:

$$u_3 = u_2 - 1 = 18 - 1 = 17.$$

Дійсно, $10^{17} \bmod 47 = 5$.

Нарешті, з (10.51) знаходимо u_1:

$$u_1 = 2 - u_2 = (2 - 18) \bmod 46 = -16 \bmod 46 = 30, \ 10^{30} \bmod 47 = 2.$$

Отже, тепер знаємо логарифми чисел із S. Найтрудомісткіший етап алгоритму позаду. Переходимо до четвертого кроку. Почнемо з $k = 3$:

$$37 \cdot 10^3 \bmod 47 = 37 \cdot 13 \bmod 47 = 11,$$
$$37 \cdot 10^4 \bmod 47 = 37 \cdot 36 \bmod 47 = 16 = 2 \cdot 2 \cdot 2 \cdot 2. \quad \blacktriangledown$$

Переходимо з останньої рівності до логарифмів (це п'ятий крок) і отримуємо кінцевий результат:

$$\log_{10} 37 = 4 \log_{10} 2 - 4 = (4 \cdot 30 - 4) \bmod 46 = 24.$$

Отже, знайдено розв'язок рівняння (10.49) $x = 24$. Можемо зробити перевірку: $10^{24} \bmod 47 = 37$.

Найшвидшим на даний час вважається варіант розглянутого алгоритму обчислення порядку, званий *Number Field Sieve* (*NFS* – загальний метод решета числового поля — метод факторизації цілих чисел). Цей метод використовує тонкі алгебраїчні конструкції й досить складний для опису. Його трудомісткість дається оцінкою

$$t_{\text{î.ï.}} < c_1 \cdot 2^{(c_2 + o(1)) \cdot \sqrt[3]{\log p \log(\log p)^2}}, \tag{10.56}$$

де c_1 і c_2 — деякі позитивні константи.

Саме цей метод диктує сьогодні умови для вибору довжин модулів криптографічних систем, стійкість яких заснована на складності обчислення дискретних логарифмів (це система Діффі–Хеллмана, Шаміра й Ель-Гамаля). Для досягнення довгострокової стійкості цих криптографічних систем рекомендується брати модулі довжиною не менше *1024* бітів [18, 40].

На закінчення зазначимо, що в навчальному посібнику не розглядаються методи зламу криптографічних систем, заснованих на факторизації чисел (таких, як *RSA*). Через те, що опис сучасних алгоритмів розкладання числа на множники, що вимагав би введення додаткових понять і алгоритмів з теорії чисел, ніде більше в книзі не використовується. Однак на сьогодні [28] швидкі методи розкладання чисел на множники характеризуються такою самою оцінкою часу, яку дає вираз (10.56). Як наслідок, для забезпечення стійкості криптографічної системи *RSA* довжина модуля повинна також бути не менше *1024* бітів (тобто прості числа, що дають при множенні модуль *RSA*, повинні бути довжиною мінімум по *512* бітів).

Контрольні питання та завдання

1. Дати визначення простого й складеного числа. Навести по три приклади простих і складених чисел.
2. Дати визначення поняття “взаємнопрості числа”. Навести приклади взаємно простих чисел і чисел, які не є взаємно простими. Що таке інверсія за модулем *n*?
3. У чому полягає завдання факторизації? Дати визначення найбільшого спільного дільника.
4. З якою метою може застосовуватися алгоритм *RSA*?
5. Описати процес шифрування з використанням алгоритму *RSA*.
6. Для чого може застосовуватися алгоритм Діффі–Хеллмана?
7. Описати послідовність дій при використанні алгоритму Діффі–Хеллмана.
8. Для яких цілей може застосовуватися алгоритм Ель-Гамаля?
9. Опишіть послідовність дій при використанні алгоритму Ель-Гамаля.
10. Для яких цілей може застосовуватися алгоритм Рабіна?
11. Опишіть послідовність дій при використанні алгоритму Рабіна.
12. Які атаки можливі при використанні алгоритмів шифрування з відкритим ключем?
13. Знайти результат виразів *5*, *16*, *27*, *-4*, *-13*, *3 + 8*, *3 - 8*, *3·8* і *3·8·5*: а) за модулем 10; б) за модулем 11.
14. Обчислити, використовуючи швидкі алгоритми піднесення до степеня, $2^8 \bmod 10$, $2^7 \bmod 10$, $7^{19} \bmod 100$ і $7^{57} \bmod 100$.

15. Розкласти на прості множники числа *108*, *77*, *65*, *30* і *159*.

16. Визначити, які з пар чисел (*25,12*), (*25,15*), (*13,39*) і (*40,27*) взаємно прості.

17. Знайти значення функції Ейлера $\varphi(14)$ і $\varphi(15)$.

18. Використовуючи властивості функції Ейлера, обчислити $\varphi(53)$, $\varphi(21)$ і $\varphi(159)$.

19. Використовуючи теорему Ферма, обчислити $3^{13} \bmod 13$, $5^{22} \bmod 11$ і $3^{17} \bmod 5$.

20. Використовуючи теорему Ейлера, обчислити $3^{9} \bmod 20$, $2^{14} \bmod 21$ і $2^{107} \bmod 159$.

21. За допомогою алгоритму Евкліда знайти $gcd(21,12)$, $gcd(30,12)$, $gcd(24,40)$ і $gcd(33,16)$.

22. За допомогою узагальненого алгоритму Евкліда знайти значення x і y в рівняннях:

а) $21 \cdot x + 12 \cdot y = gcd(21,12)$; б) $30 \cdot x + 12 \cdot y = gcd(30,12)$;

в) $24 \cdot x + 40 \cdot y = gcd(24,40)$; г) $33 \cdot x + 16 \cdot y = gcd(33,16)$.

23. Обчислити $3^{-1} \bmod 7$, $5^{-1} \bmod 8$, $3^{-1} \bmod 53$ і $10^{-1} \bmod 53$.

24. Виписати всі прості числа, менші *100*. Які з них відповідають вигляду $p = 2 \cdot q + 1$, де q також просте?

25. Знайти всі допустимі варіанти вибору параметра g в системі Діффі–Хеллмана за $p = 11$.

26. Обчислити секретні ключі y_A, y_B і загальний ключ Z_{AB} для системи Діффі–Хеллмана з параметрами:

а) $p = 23$, $g = 5$, $x_A = 5$, $x_B = 7$;

б) $p = 19$, $g = 2$, $x_A = 5$, $x_B = 7$;

в) $p = 23$, $g = 7$, $x_A = 3$, $x_B = 4$;

г) $p = 17$, $g = 3$, $x_A = 10$, $x_B = 5$;

д) $p = 19$, $g = 10$, $x_A = 4$, $x_B = 8$.

27. Для криптосистеми Шаміра із заданими параметрами p, c_A і c_B знайти відсутні параметри й описати процес передачі повідомлення M від A до B:

а) $p = 19$, $c_A = 5$, $c_B = 7$, $M = 4$;

б) $p = 23$, $c_A = 15$, $c_B = 7$, $M = 6$;

в) $p = 19$, $c_A = 11$, $c_B = 5$, $M = 10$;

г) $p = 23$, $c_A = 9$, $c_B = 3$, $M = 17$;

д) $p = 17$, $c_A = 3$, $c_B = 13$, $M = 9$.

28. Для криптосистеми Ель-Гамаля із заданими параметрами p, g, c_B і k знайти відсутні параметри й описати процес передачі повідомлення M користувачеві B:

а) $p = 19$, $g = 2$, $c_B = 5$, $k = 7$, $M = 5$;

б) $p = 23$, $g = 5$, $c_B = 8$, $k = 10$, $M = 10$;

в) $p = 19$, $g = 2$, $c_B = 11$, $k = 4$, $M = 10$;

г) $p = 23$, $g = 7$, $c_B = 3$, $k = 15$, $M = 5$;

д) $p = 17$, $g = 3$, $c_B = 10$, $k = 5$, $M = 10$.

29. У системі *RSA* із заданими параметрами p_A, q_A і d_A знайти відсутні параметри й описати процес передачі повідомлення M користувачеві A:

а) $p_A = 5$, $q_A = 11$, $d_A = 3$, $M = 12$;

б) $p_A = 5$, $q_A = 13$, $d_A = 5$, $M = 20$;

в) $p_A = 7$, $q_A = 11$, $d_A = 7$, $M = 17$;

г) $p_A = 7$, $q_A = 13$, $d_A = 5$, $M = 30$;

д) $p_A = 3$, $q_A = 11$, $d_A = 3$, $M = 15$.

30. Користувачеві системи *RSA* з параметрами $n = 187$ і $d = 3$ передано зашифроване повідомлення $e = 100$. Розшифрувати це повідомлення, зламавши систему *RSA* користувача.

31. Згенерувати відкритий ключ та, вживши криптографічну систему *Рабіна*, зашифрувати відкритий текст: "*криптографія*", $p_A = 53$, $q_A = 71$. При переході до цифрового запису тексту кожну літеру замінити її двоцифровим десятковим номером в українській абетці (нумерацію починати з *00*) та розбити текст на блоки з чотирьох цифр.

32. Користувач A згенерував відкритий ключ $n_A = 4189$ і за допомогою криптографічної системи *Рабіна* отримав на свою адресу зашифроване повідомлення $C = 3272\ 1516\ 0166$. Відновити закриті ключі і розшифрувати зашифроване повідомлення (у цифровому запису розшифрованого повідомлення кожне двозначне десяткове число є номером відповідної літери в українській абетці, нумерація починається з *00* і закінчується числом *31*).

33. Використовуючи метод *"крок немовлять, крок велетня"*, вирішити такі рівняння:

а) $2^x \bmod 29 = 21$;

б) $3^x \bmod 31 = 25$;

в) $2^x \bmod 37 = 12$;

г) $6^x \bmod 41 = 21$;

д) $3^x \bmod 43 = 11$.

34. Використовуючи алгоритм обчислення порядку, розв'язати такі рівняння:

а) $2^x \bmod 53 = 24$;

б) $2^x \bmod 59 = 13$;

в) $2^x \bmod 61 = 45$;

г) $2^x \bmod 67 = 41$;

д) $7^x \bmod 71 = 41$.

ІМЕННИЙ ПОКАЖЧИК

ПРЕДМЕТНИЙ ПОКАЖЧИК

П

Р

С

Т

СПИСОК ЛІТЕРАТУРИ

1. Аграновский А. В. Практическая криптография: алгоритмы и их программирование / А. В. Аграновский, Р. А. Хади. – М. : СОЛОН-Пресс, 2009. – 256 с.: ил.

2. Бабаш А. В. Криптография / А. В. Бабаш, Г. П. Шанкин; под редакцией В. П. Шерстюка, Э. А. Применко. – М. : СОЛОН-Пресс, 2007. – 512 с.: ил.

3. Блінцов В. С. Математичні основи криптології + CD : Навчальний посібник для студ. вищих навч. закл. / В. С. Блінцов, Ю. Л. Гальчевський. – Миколаїв : Національний ун-т кораблебудування ім. адмірала Макарова, 2006. – 232 с.: іл.

4. Безпека інформаційних систем і технологій: Навч. посібник / В. І. Єсін, О. О. Кузнецов, Л. С. Сорока. – Х. : ХНУ імені В. Н. Каразіна, 2013. – 632 с.

5. Богуш В. М. Криптографічні застосування елементарної теорії чисел : Навч. посібник / В. М. Богуш, В. А. Мухачов. – К. : Державний ун-т інформаційно-комунікаційних технологій, 2006. – 126 с.: іл.

6. Введение в криптографию / Н. П. Варновский, Ю. В. Нестеренко, Г. А. Кабатянский и др.; под ред. В. В. Ященко. – М. : МЦНМО-ЧеРо, 1998. – 272 с.: ил.

7. Горбенко І. Д. Прикладна криптологія. Теорія. Практика. Застосування : Монографія / І. Д. Горбенко, Ю. І. Горбенко. – Харків : Видавництво "Форт", 2012. – 880 с.: іл.

8. Горбенко І. Д. Захист інформації в інформаційно-телекомунікаційних системах : Навч. посіб. для студ. Ч. 1. Криптографічний захист інформації / І. Д. Горбенко, Т. О. Гріненко. – Х. : Харк. нац. ун-т радіоелектрон., 2004. – 368 с.: іл.

9. Грайворонський М. В. Безпека інформаційно-комунікаційних систем : Підручник / М. В. Грайворонський, О. М. Новіков. – К. : Видавнича група BHV, 2009. – 608 с.: іл.

10. Задірака В. К. Комп'ютерна криптологія : підручник / В. К. Задірака, О. С. Олексюк. – К. : Тернопільська академія народного господарства; НАН України; Інститут кібернетики ім. В. М. Глушкова, 2002. – 504 с.: іл.

11. Захист інформації в мережах передачі даних: Підручник / Юдін О. К., Корченко О. Г., Конахович Г. Ф. – К. : Вид-во ТОВ "НВП" ІНТЕРСЕРВІС", 2009. – 716 с.

12. Защита информации в компьютерных системах и сетях / Романец Ю. В., Тимофеев П. А., Шаньгин В. Ф.; под ред. В. Ф. Шаньгина. – М. : Радио и связь, 2001. – 376 с.: ил.

13. Иванов М. А. Криптографические методы защиты информации в компьютерных системах и сетях / Иванов М. А. – М. : КУДИЦ-ОБРАЗ, 2001. – 363 с.: ил.

14. Коблиц Н. Курс теории чисел и криптографии / Н. Коблиц – М. : Научное издательство ТВП, 2001. – 272 с.: ил.

15. Корченко О. Г. Охорона конфіденційної інформації підприємства : Навч. посіб. / О. Г. Корченко, Ю. О. Дрейс. – Житомир: ЖВІ НАУ, 2011. – 172 с.

16. Коутинхо С. Введение в теорию чисел. Алгоритм RSA. – М. : Постмаркет, 2001. – 328 с.: ил.

17. Мао В. Современная криптография: теория и практика / Мао В.; пер. с англ. – М. : Издательский дом "Вильямс", 2005. – 768 с.: ил.

18. Математичні основи криптографії : Навч. посіб. / Г. В. Кузнецов, В. В. Фомичов, С. О. Сушко, Л. Я. Фомичова. – Д. : Національний гірничий університет, 2004. – 391 с.: іл.

19. Математичні основи криптоаналізу : Навч. посіб. / С. О. Сушко, Г. В. Кузнецов, Л. Я. Фомичова, А. В. Корабльов. – Д. : Національний гірничий університет, 2010. – 465 с.: іл.

20. Математические и компьютерные основы криптологии : Учебное пособие / Ю. С. Харин, В. И. Берник, Г. В. Матвеев, С. В. Агиевич. – Минск : Новое издание, 2003. – 382 с.: ил.

21. Математические основы криптологии : Учебное пособие / Ю. С. Харин, В. И. Берник, Г. В. Матвеев. – Минск: БГУ, 1999. – 319 с.: ил.

22. Методи та алгоритми симетричної криптографії: Навч. посіб. / Кузнецов О. О., Євсеев С. П., Смірнов О. А., Мелешко Є. В., Король О. Г. – Кіровоград : Вид. КНТУ, 2012. – 316 с.

23. Молдовян Н.А. Криптография с открытым ключом /Н.А. Молдовян, А.А. Молдовян. – СПб. : БХВ-Петербург, 2005. – 288 с.: ил.

24. Мукачев В. А. Методы практической криптографии / В. А. Мукачев, А. А. Хорошко. – К.: ООО "Полиграф-Консалтинг", 2005. – 215 с.: ил.

25. Ожиганов А. А. Основы криптоанализа симметричных шифров : Учебное пособие / Ожиганов А. А.– СПб. : СПбГУ ИТМО, 2008. – 44 с.: ил.

26. Основы криптографии : Учебное пособие / А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин. – М. : Гелиос АРВ, 2002. – 480 с.: ил.

27. Основы современной криптографии / С. Г. Баричев, В. В. Гончаров, В. Е. Серов. – М. : "Горячая линия-Телеком", 2001. – 154 с.

28. Панасенко С. П. Алгоритмы шифрования. Специальный справочник / Панасенко С. П. – СПб. : БХВ-Петербург, 2009. – 576 с.: ил.

29. Петров А. А. Компьютерная безопасность. Криптографические методы защиты / Петров А. А. – М. : Издательство ДМК, 2000. – 448 с.: ил.

30. Поточные шифры / А. В. Асосков, М. А. Иванов, А. А. Мирский и др. — М. : КУДИЦ-ОБРАЗ, 2003. – 336 с.: ил.

31. Ростовцев А. Г. Теоретическая криптография / А. Г. Ростовцев, Е. Б. Маховенко. – СПб. : АНО НПО Профессионал, 2005. – 480 с.: ил.

32. Рябко Б. Я. Криптографические методы защиты информации : Учебное пособие для вузов / Б. Я. Рябко, А. Н. Фионов. – М. : Горячая линия – Телеком, 2005. – 229 с.: ил.

33. Саломаа А. Криптография с открытым ключом : пер. с англ / Саломаа А. – М.: Мир, 1996. – 304 с.: ил.

34. "Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования" ДСТУ ГОСТ 28147 : 2009 / Державний Комітет України з питань технічного регулювання та споживчої політики, Наказ № 495 від 22.12.2008 р.– К., 2009.

35. Смарт Н. Криптография : пер. с англ / Смарт Н. – М. : Техносфера, 2005. – 528 с.: ил.

36. Стандарт криптографической защиты – AES. Конечные поля / А.С. Зензин, М.А. Иванов ; под ред. М.А. Иванова. – М. : КУДИЦ-ОБРАЗ, 2002. – 176 с.: ил.

37. Стеганографія: Навч. посіб. / О. О. Кузнецов, С. П. Євсеєв, О. Г. Король. – Х. : Вид. ХНЕУ, 2011. – 232 с.

38. Тилборг ван Х. К. А. Основы криптологии. Профессиональное руководство и интерактивный учебник : пер. с англ / Тилборг ван Х. К. А. – М. : Мир, 2006. – 471 с.: ил.

39. Фергюссон Н. Практическая криптография : пер. с англ. / Н. Фергюссон, Б. Шнайер. – М. : издательский дом "Вильямс", 2005. – 424 с.: ил.

40. Фомичев В.М. Дискретная математика и криптология. – М. : Диалог-МИФИ, 2003. – 400 с.: ил.

41. Фороузан Б. А. Криптография и безопасность сетей: Учебное пособие / Фороузан Б. А.; пер. с англ. Под ред. А. Н. Берлина. – М. : Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. – 784 с.: ил., табл.

42. Черемушкин А. В. Криптографические протоколы. Основные свойства и уязвимости : Учебное пособие / Черемушкин А. В. – М. : Издательский центр "Академия", 2009. – 272 с.: ил.

43. Шеннон К. Теория связи в секретных системах // В книге работы по теории информации и кибернетики / Шеннон К. – М. : ИЛ, 1963. – 830 с.

44. Щербаков А. Ю. Прикладная криптография. Использование и синтез криптографических интерфейсов / А. Ю. Шербаков, А.В. Домашев. – М. : Издательско-торговый дом "Русская редакция", 2003. – 416 с.: ил.

45. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Шнайер Б. – М. : Триумф, 2002. – 797 с.: ил.

ВІДПОВІДІ НА КОНТРОЛЬНІ ЗАВДАННЯ

Розділ 2

18. “*НКПРГХГ*”.
19. “*АБРИКОС*”.
20. *К = 7*.
21. “*ЄЮЯЛШХНА*”.
22. “*АЛГОРИТМ*”.
23. “*ЖЄУТПАФН*”.
24. “*ГІПОТЕЗА*”.
25. “*НСЄФЛЮИП*”.
26. “*ВИНОГРАД*”.
27. “*ТФМЕ.Ь_Щ*”.
28. “*КОМБІНАТ*”.
29. “*НАВУЯЛЛП*”.
30. “*ГВОЗДІКА*”.
31. “*OLYMVQKGZDGAXET*”.
32. “*WANT TO TRAVEL*”

Розділ 3

10. а) *32* біти; б) *188* блоків.
11. а) *24*; б) *5* бітів.
12. а) *20 922 789 888 000*; б) *43* біти.
13. $(11011100)_2$.
14. $(11111010)_2$.
15. а) $(00000000)_2$; б) $(11111111)_2$; в) $(01001101)_2$; г) $(10110010)_2$.
16. $(00111)_2$.
17. $(011)_2$.
18. $(11110)_2$.

19. Прямим *P*-блоком.

20. *P*-блоком розширення.

21. *P*-блоком стискання.

22. Прямим *P*-блоком.

23.

Вхід: лівий біт \ Вхід: правий біт	0	1
0	10	00
1	11	01

24. а) *10*; б) *00*.

25. а) *110*; б) *011*.

Розділ 4

10. а) *7*; *8*; *13*; *4*; *10*; *6*; *3*; *5*; *15*; *14*;
б) *3*; *21*; *14*; *18*; *19*; *2*; *15*; *1*; *9*; *11*.

11. а) $k_3 = 0{,}1$; $k_4 = 0{,}2$; $k_5 = 0{,}3$; $k_6 = 0{,}8$; $k_7 = 0{,}4$; $k_8 = 0{,}9$; $k_9 = 0{,}9$; $k_{10} = 0{,}5$; $k_{11} = 0{,}4$; $k_{12} = 0{,}5$;
б) $k_4 = 0{,}4$; $k_5 = 0{,}4$; $k_6 = 0{,}1$; $k_7 = 0{,}5$; $k_8 = 0{,}3$; $k_9 = 0{,}9$; $k_{10} = 0{,}8$; $k_{11} = 0{,}6$; $k_{12} = 0{,}5$; $k_{12} = 0{,}3$.

12. $a = 5$; $b = 3$; $c = 11$.

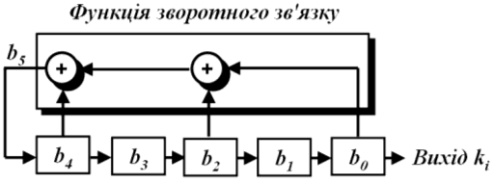
13. а) $x_{11} = 35$; б) $x_{11} = 679$.

14. а) $(110001011001)_2$; б) $(110111100011)_2$.

15. а) $x_1 = 81$; $x_2 = 6$; $x_3 = 36$; $x_4 = 422$; $x_5 = 225$; $x_6 = 370$; $x_7 = 119$; $x_8 = 177$;
б) $x_1 = 81$; $x_2 = 144$; $x_3 = 59$; $x_4 = 629$; $x_5 = 639$; $x_6 = 485$; $x_7 = 648$; $x_8 = 660$.

16. *18*, *4*, *232*, *61*, *37*, *201*, *252*, *166*, *151*, *162*, *244*, *66*, *137*, *130*, *28*, *248*, *127*, *77*, *205*, *226*.

17. а)



б) *31* біт.

18. а) $x^4 + x^3 + x^2 + 1$; б) *15* бітів.

19. $(1111000100010111000)_2$.

20. *5* розрядів.

21. Максимальний період двійкової послідовності лінійних регістрів зсуву: R_1 — *524287* бітів; R_2 — *4194303* біти; R_3 — *8388607* бітів.

22. а) $f(x, y, z) = f(1, 0, 0) = 0$; б) $f(x, y, z) = f(0, 1, 1) = 1$; синхронізуються лінійні регістри зсуву R_2 і R_3; в) $f(x, y, z) = f(0, 0, 0) = 0$; г) $f(x, y, z) = f(1, 1, 1) = 1$; синхронізуються лінійні регістри зсуву R_1, R_2 і R_3.

Розділ 5

11. $L_0||R_0 = cc1fc6e0f0aae8a5_{16}$.

12. $R_0^E = fa155575150b_{16}$.

13. $R_1^E = 8a2b57d029a6_{16}$.

14. $R_1^{\oplus} = cf430fca9568_{16}$.

15. $R_{12}^{\oplus} = 5dfd4bd5b555_{16}$.

16. а) 5_{16}; б) e_{16}; в) f_{16}; г) 0_{16}.

17. $R_{13}^S = 75140940eb51_{16}$.

18. $R_2^P = 1680e976_{16}$.

19. $R_3^L = 48a3638f_{16}$.

20. $L_5 = 48a3638f_{16}$ і $R_5 = 4178632b_{16}$.

21. $L_6 = 4178632b_{16}$ і $R_6 = 48a9b329_{16}$.

22. $C_0 = c3c033a_{16}$ і $D_0 = 33f0cfa_{16}$.

23. а) $C_2 = 0f00ceb_{16}$ і $D_2 = cfc33e8_{16}$; б) $C_3 = 3c033ac_{16}$ і $D_3 = 3f0cfa3_{16}$; в) $C_7 = 033ac3c_{16}$ і $D_7 = 0cfa33f_{16}$; г) $C_{12} = 7587806_{16}$ і $D_{12} = f467e19_{16}$.

24. $k_2 = 4568581abcce_{16}$.

25. а) $k_4 = da2d032b6ee3_{16}$; б) $k_9 = 84bb4473dccc_{16}$;
в) $k_{13} = 99c31397c91f_{16}$; г) $k_{16} = 181c5d75c66d_{16}$.

26. $f07704d0741eb2c2_{16}$.

Розділ 6

15. У першому блоці розшифрованих даних буде зруйновано близько половини бітів.

16. У першому блоці розшифрованих даних буде зруйновано близько половини бітів; у другому – один (5) біт.

17. Помилка у відкритих даних вплине на всі подальші зашифровані дані, але самоусунеться в ході розшифрування.

18. Першим ефектом буде збій зруйнованого другого біта першого блока розшифрованих даних. Другим ефектом буде збій зруйнованих із другого по дев'ятий блоки розшифрованих даних.

19. Буде збій другого біта першого блока розшифрованих даних. Подальша послідовність блоків буде розшифрована коректно.

20. Буде збій тільки другого біта другого блока розшифрованих даних.

Розділ 7

8. $k_2^{(1)} = 96a2_{16}$, $k_2^{(2)} = 978e_{16}$, $k_2^{(3)} = b820_{16}$, $k_2^{(4)} = b940_{16}$, $k_2^{(5)} = 1af6_{16}$, $k_2^{(6)} = 7b7d_{16}$.

9. $k_2^{(1)} = e8d8_{16}$, $k_2^{(2)} = 5099_{16}$, $k_2^{(3)} = 4c25_{16}$, $k_2^{(4)} = 95d7_{16}$, $k_2^{(5)} = eb43_{16}$, $k_2^{(6)} = a108_{16}$, $k_2^{(7)} = 8382_{16}$, $k_2^{(8)} = 47e0_{16}$, $k_2^{(9)} = b324_{16}$.

10. $k_3^{(1)} = fca2_{16}$, $k_3^{(2)} = 6bff_{16}$, $k_3^{(3)} = ff29_{16}$, $k_3^{(4)} = 9427_{16}$, $k_3^{(5)} = 9b4b_{16}$, $k_3^{(6)} = a576_{16}$, $k_3^{(7)} = bad1_{16}$, $k_3^{(8)} = 6872_{16}$, $k_3^{(9)} = efa4_{16}$.

11. $k_5^{(1)} = 92d4_{16}$, $k_5^{(2)} = c7e8_{16}$, $k_5^{(3)} = f424_{16}$, $k_5^{(4)} = 1266_{16}$, $k_5^{(5)} = 3370_{16}$, $k_5^{(6)} = 8035_{16}$, $k_5^{(7)} = 1af6_{16}$, $k_5^{(8)} = 7b3d_{16}$.

12. $k_1^{(1)} = 5773_{16}$, $k_1^{(2)} = 7f25_{16}$, $k_1^{(3)} = ab30_{16}$, $k_1^{(4)} = e2ef_{16}$, $k_1^{(5)} = 529a_{16}$, $k_1^{(6)} = 20e2_{16}$, $k_1^{(7)} = c77b_{16}$, $k_1^{(8)} = 9a70_{16}$, $k_1^{(9)} = 7bcc_{16}$.

13. $X_1^{(1)} = abc3_{16}$, $X_2^{(1)} = 5c2d_{16}$, $X_3^{(1)} = e77a_{16}$, $X_4^{(1)} = 1c2d_{16}$.

14. $y_1^{(1)} = 5c92_{16}$, $y_2^{(1)} = 62b8_{16}$.

15. $C = 4825e238d0fb9c46_{16}$.

16. $f_1^{(1)} = 4cb9_{16}$, $f_2^{(1)} = 4000_{16}$.

Розділ 8

11. N_1 — $0cbd4791_{16}$, N_2 — $191a2ab8_{16}$.

12. N_1 — $cc29563b_{16}$, N_2 — $434665b2_{16}$.

13. N_1 — $10d3b61a_{16}$, N_2 — $02af83f6_{16}$.

14. N_1 — $0d3b61a5_{16}$, N_2 — $3153698e_{16}$.

15. Довжина імітовставки повинна бути не менше 32 бітів.

16. Імовірність нав'язування помилкових перешкод $P_{ЛП} \approx 5{,}96 \cdot 10^{-10}$.

Розділ 9

11. $x^7 + x^6 + x^2 + x$.

12. *Частка від ділення* — $x^7 + x^5 + x^4 + x^2 + 1$. *Залишок від ділення:* $x^5 + x^3 + x$.

13. $\{ f2 \} \cdot x^7 + \{ 1b \} \cdot x^5 + \{ 03 \} \cdot x^3 + \{ 17 \} \cdot x + \{ 1d \}$.

14. $\{ 06 \} \cdot x^6 + \{ 05 \} \cdot x^5 + \{ 19 \} \cdot x^4 + \{ 15 \} \cdot x^3 + \{ 1d \} \cdot x^2 + \{ 13 \} \cdot x + \{ 04 \}$.

15. $\{ 19 \} \cdot x^3 + \{ 11 \} \cdot x^2 + \{ 1c \} \cdot x + \{ 0a \}$.

16. $x^7 + x^6 + x^3 + x^2 + x$.

17. $x^5 + x^2 + 1$.

18. $x^7 + x^6 + x^5 + x + 1$.

19. *49ded28945db96f17f39871a7702533b*.

20. $2b359f6849506a02f27f9ca443ea5b6b_{16}$.

21. $49db873b453953897f02d2f177de961a_{16}$.

22. *f187de773b9639498953db7f1ad2245*.

23. *584dcaf11b4b5aacdbe7caa81b6bb0e5*.

24. *72dbe546102a2bf01ad3ef5836ab483d*.

25. *aa8f5f0361dde3ef82d24ad26832469a*.

26. *d014f9a8c9ee2589e13f0cc8b6630ca6*.

27. *w[6] = 5846f2f9*; *w[7] = 5c43f4fe*; *w[8] = 544afef5*; *w[9] = 5847f0fa*; *w[10] = 4856e2e9*; *w[11] = 5c43f4fe*.

28. *w[8] = a573c29f*; *w[9] = a176c498*; *w[10] = a97fce93*; *w[11] = = a572c09c*; *w[12] = 1651a8cd*; *w[13] = 0244beda*; *w[14] = 1a5da4c1*; *w[15] = = 0640bade*.

Розділ 10

13. *а) 5 mod 10 = 5, 16 mod 10 = 6, 27 mod 10 = 7, –4 mod 10 = 6, –13 mod 10 = –3 mod 10 = 7, (3 + 8) mod 10 = 1, (3 – 8) mod 10 = 5, (3·8) mod 10 = 4, (3·8·5) mod 10 = (4·5) mod 10= 0.*

б) 5 mod 11 = 5, 16 mod 11 = 5, 27 mod 11 = 5, –4 mod 11 = 7, –13 mod 11 = –2 mod 11 = 9, (3 + 8) mod 11 = 0, (3 – 8) mod 11 = 6, (3·8) mod 11 = 2, (3·8·5) mod 11= (2·5) mod 11= 10.

14. $2^8 \bmod 10 = 6$, $2^7 \bmod 10 = 7$, $7^{19} \bmod 100 = 43$, $7^{57} \bmod 100 = 7$.

15. *108 = 2·2·3·3·3*, *77 = 7·11*, *65 = 5·13*, *30 = 3·3·5*, *159 = 3·53*.

16. *Пари (25,12) і (40,27) взаємно прості, решта – ні (числа (25,15) поділяються на 5, (13,39) поділяються на 13)*.

17. $\varphi(14) = 6$, $\varphi(15) = 8$.

18. $\varphi(53) = 52$, $\varphi(21) = \varphi(7) \cdot \varphi(3) = 6 \cdot 2 = 12$, $\varphi(159) = \varphi(3) \cdot \varphi(53) = 2 \cdot 52 = 104$.

19. $3^{13} \bmod 13 = 3 \cdot 3^{12} \bmod 13 = 3$, $5^{22} \bmod 11 = 5^2 \cdot 5^{10} \cdot 5^{10} \bmod 11 = 25 \bmod 11 = 3$, $3^{17} \bmod 5 = 3$.

20. $3^9 \bmod 20 = 3 \cdot 3^8 \bmod 20 = 3$, $2^{14} \bmod 21 = 2^2 \cdot 2^{12} \bmod 21 = 4$, $2^{107} \bmod 159 = 2^3 \cdot 2^{104} \bmod 159 = 8$.

21. $gcd(21,12) = 3$, $gcd(30,12) = 6$, $gcd(24,40) = gcd(40,24) = 8$, $gcd(33,16) = 1$.

22. а) $x = -1, y = 2$. б) $x = 1, y = -2$. в) $x = 2, y = -1$. г) $x = 1, y = -2$.

23. $3^{-1} \bmod 7 = 5$, $5^{-1} \bmod 8 = 5$, $3^{-1} \bmod 53 = 18$, $10^{-1} \bmod 53 = 16$.

24. Прості числа, які менше *100*: *2*, *3*, *5*, *7*, *11*, *13*, *17*, *19*, *23*, *29*, *31*, *37*, *41*, *43*, *47*, *53*, *59*, *61*, *67*, *73*, *79*, *83*, *89*, *97*, із них числа *5*, *7*, *11*, *23*, *47*, *59* і *83* відповідають вигляду $p = 2 \cdot q + 1$.

25. При $p = 11$ як параметр g можуть бути вибрані числа *2*, *6*, *7* і *8*.

26. а) $y_A = 20, y_B = 17, Z_{AB} = 21$.
б) $y_A = 13, y_B = 14, Z_{AB} = 10$.
в) $y_A = 21, y_B = 9, Z_{AB} = 16$.
г) $y_A = 8, y_B = 5, Z_{AB} = 9$.
д) $y_A = 6, y_B = 17, Z_{AB} = 16$.

27. а) $d_A = 11, d_B = 13, x_1 = 17, x_2 = 5, x_3 = 6, x_4 = 4$;
б) $d_A = 3, d_B = 19, x_1 = 8, x_2 = 12, x_3 = 3, x_4 = 6$;
в) $d_A = 5, d_B = 11, x_1 = 14, x_2 = 10, x_3 = 3, x_4 = 10$;
г) $d_A = 5, d_B = 15, x_1 = 7, x_2 = 21, x_3 = 14, x_4 = 17$;
д) $d_A = 11, d_B = 5, x_1 = 15, x_2 = 2, x_3 = 8, x_4 = 9$.

28. а) $d_B = 13, r = 14, e = 12, M' = 5$;
б) $d_B = 16, r = 9, e = 15, M' = 10$;
в) $d_B = 15, r = 16, e = 14, M' = 10$;
г) $d_B = 21, r = 14, e = 12, M' = 5$;
д) $d_B = 8, r = 5, e = 12, M' = 10$.

29. а) $n_A = 55, \varphi(n_A) = 40, c_A = 27, e = 23, M' = 12$;
б) $n_A = 65, \varphi(n_A) = 48, c_A = 29, e = 50, M' = 20$;
в) $n_A = 77, \varphi(n_A) = 60, c_A = 43, e = 52, M' = 17$;
г) $n_A = 91, \varphi(n_A) = 72, c_A = 29, e = 88, M' = 30$;
д) $n_A = 33, \varphi(n_A) = 20, c_A = 7, e = 9, M' = 18$.

30. $M = 111$.

31. Відкритий ключ $n_A = 3763$, зашифроване повідомлення $C = 1255\ 3575\ 3719\ 0160\ 0529\ 1795$.

32. $p_A = 59, q_A = 71;$ “*захист*”.

33. а) $x = 17$; б) $x = 10$; в) $x = 28$; г) $x = 14$; д) $x = 30$.

34. а) $x = 10000$; б) $x = 20000$; в) $x = 1000$; г) $x = 12345$; д) $x = 25000$.

Навчальне видання

Корченко Олександр Григорович
Сіденко Володимир Павлович
Дрейс Юрій Олександрович

ПРИКЛАДНА КРИПТОЛОГІЯ:
системи шифрування

Підручник

Редактор **І. М. Амірова**
Технічний редактор **О. В. Белікова**
Комп'ютерне верстання – **О. В. Белікова**

Підписано до друку 10.09.2014.
Формат 60x84/16. Друк офсетний.
Папір офсетний. Надруковано в Україні.
Наклад 500 прим.

Надруковано в друкарні ТОВ «Наш формат»,
01042, Україна, м. Київ, вул. Фрунзе, 84