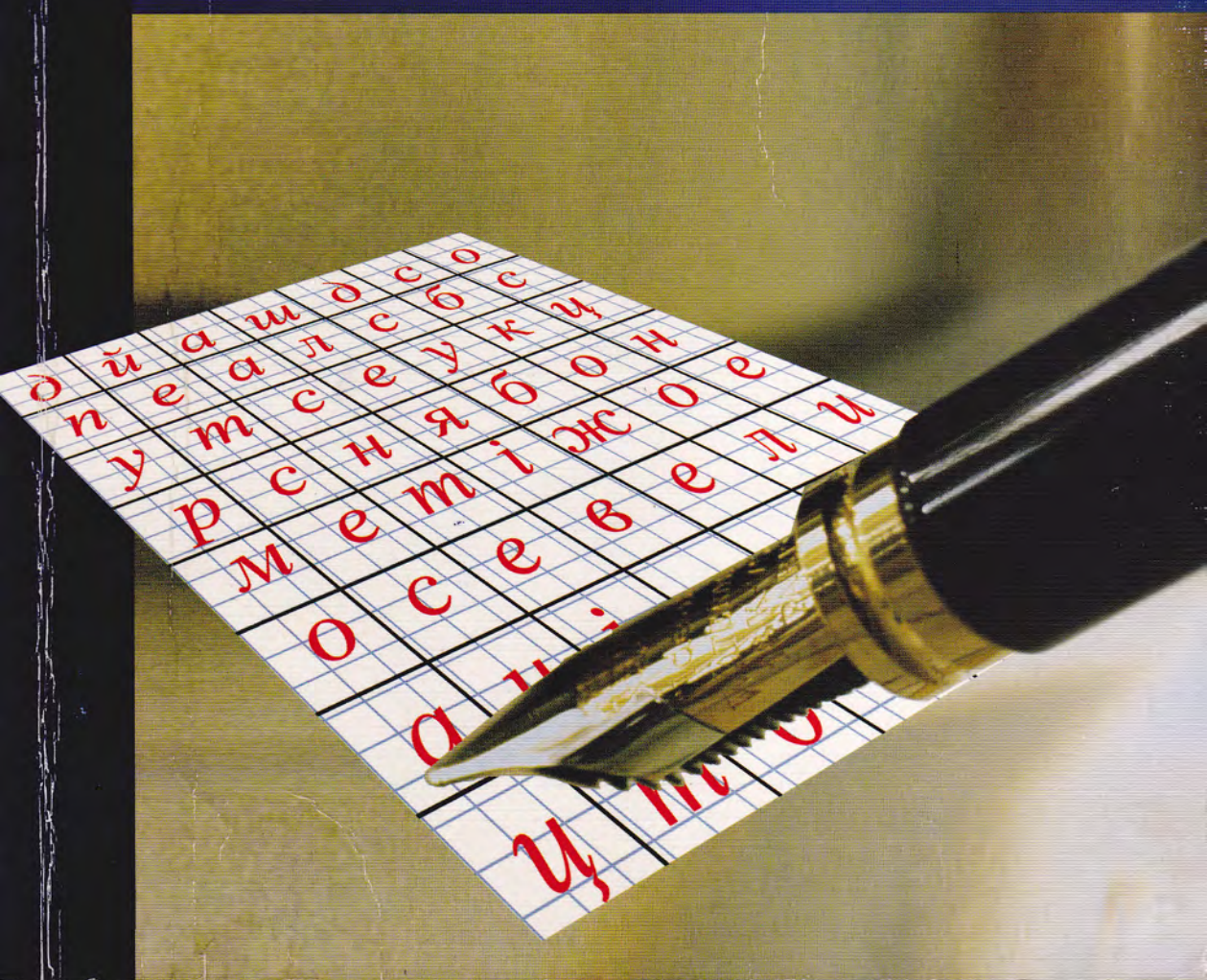
Остапов С. Е.
Валь Л. О.

# ОСНОВИ КРИПТОГРАФІЇ



**С. Е. Остапов**
**Л. О. Валь**

# Основи криптографії

*Рекомендовано Міністерством освіти і науки України*
*як навчальний посібник для студентів вищих навчальних закладів*

**Чернівці**
**Книги – ХХІ**
**2008**

**УДК 681.3.07**
**ББК 32.81**
**О 76**

*Рекомендовано Міністерством освіти і науки України*
*як навчальний посібник для студентів вищих навчальних закладів*
(Лист № 1.4/18-Г-1336 від 11 червня 2008 р.)

**Рецензенти:** доктор фізико-математичних наук, професор *Тарасенко В. П.*;
доктор технічних наук, професор *Локазюк В. М.*;
кандидат фізико-математичних наук, доцент *Чернов В. М.*

**Остапов С. Е., Валь Л. О.**

**О 76** Основи криптографії: Навчальний посібник. – Чернівці: Книги – XXI, 2008. – 188 с.

**ISBN 978-966-2147-35-3**

Викладено основи сучасної криптографії, які не пов'язані з державною таємницею. Посібник складається з теоретичної частини та криптографічного практикуму. У теоретичній частині висвітлені деякі історичні аспекти розвитку криптографічної науки, класичні техніки шифрування, симетричні й асиметричні криптографічні алгоритми, електронний цифровий підпис і пов'язані з ним алгоритми хешування; розглянуті проблеми розповсюдження криптографічних ключів і генерування випадкових та псевдовипадкових послідовностей. Криптографічний практикум містить задачі й лабораторні роботи, які сприятимуть набуттю студентами навичок створення програмного забезпечення з урахуванням вимог до криптографічного захисту інформації.

Для студентів, які навчаються за напрямами «Комп'ютерна інженерія», «Програмна інженерія», «Інформатика», студентів інших спеціальностей, де вивчається цикл дисциплін захисту інформації, а також для бажаючих самостійно опанувати основи криптографії.

**ББК 32.81**

**ISBN 978-966-2147-35-3**

## Зміст


ВСТУП ..... 5

**Розділ І. З історії криптографії ..... 9**

Класичні техніки шифрування ..... 16

Шифри підстановок ..... 18

**Розділ ІІ. Елементи теорії зв'язку в секретних системах ..... 21**

Клод Шеннон. «Теорія зв'язку в секретних системах» ..... 21

Класифікація сучасних криптосистем. Вимоги до сучасних криптосистем ..... 24

Комбіновані криптосистеми ..... 27

**Розділ ІІІ. Симетричні криптосистеми ..... 30**

Блокові шифри ..... 30

Загальні відомості про блокові шифри ..... 30

Особливості стандарту DES ..... 35

Шифрування DES ..... 36

Початкова перестановка ..... 38

Перестановка з розширенням ..... 39

Підстановка з допомогою S-блоків ..... 39

Перетворення ключа ..... 42

Стійкість DES ..... 43

Потрійний DES ..... 44

DESX ..... 45

S-DES ..... 45

Особливості стандарту ГОСТ 28147-89 ..... 49

Міжнародний стандарт шифрування даних IDEA ..... 57

Особливості стандарту AES ..... 60

Опис AES ..... 62

Потокові шифри ..... 66

Загальні відомості про потокові шифри ..... 66

**Розділ IV. Асиметричні криптосистеми ..... 70**

Елементи теорії чисел ..... 71

Криптосистема RSA ..... 74

Криптостійкість RSA ..... 78

Криптосистема Ель-Гамаля ..... 79

Криптостійкість системи Ель-Гамаля ..... 80

**Розділ V. Електронний цифровий підпис ..... 81**

Функції хешування ..... 82

Алгоритми хешування сімейства MD ..... 83

Алгоритм MD2 ..... 83

Алгоритми MD4-MD5 ..... 84

Безпека MD4-MD5 ..... 85

Алгоритм SHA-1 ..... 86

Безпека SHA ..... 87

Алгоритми електронного цифрового підпису ..... 88

Алгоритм цифрового підпису DSA ..... 88

Безпека DSA ..... 89

Стандарти цифрового підпису ГОСТ Р 34.10-94
і ГОСТ Р 34.10-2001 ... 89
**Розділ VI. Елементи криптоаналізу ... 92**
Атаки на криптосистему RSA ... 95
Елементи частотного криптоаналізу ... 97
Різницевий криптоаналіз ... 99
Лінійний криптоаналіз ... 99
**Розділ VII. Проблеми розподілу криптографічних ключів ... 101**
Генерування ключів ... 101
Ієрархія ключів ... 102
Накопичення ключів ... 103
Розподіл ключів ... 103
Протокол прямого обміну ключами ... 110
Протокол Діффі-Хеллмана ... 111
Протокол обчислення ключа парного зв'язку ECKEP ... 114
**Розділ VIII. Проблеми генерування випадкових і псевдовипадкових послідовностей ... 116**
Генератори випадкових послідовностей ... 116
Генерування псевдовипадкових послідовностей ... 118
Лінійний регістр зсуву зі зворотними зв'язками ... 119
Генератор BBS ... 121
Генератор Блум-Мікалі ... 122
Генератор RSA ... 122
**Криптографічний практикум ... 125**
Практичні заняття з криптографії ... 126
Вивчення шифру Цезаря ... 127
Вивчення афінної системи шифрування Цезаря ... 127
Вивчення матричних способів шифрування ... 129
Задачі з криптографії ... 131
Класичні техніки шифрування ... 131
Симетричні криптосистеми ... 132
Асиметричні криптосистеми ... 133
Електронний цифровий підпис ... 135
Генерування псевдовипадкових послідовностей ... 138
**Лабораторний практикум із криптографії ... 139**
Лабораторна робота № 1 ... 140
Лабораторна робота № 2 ... 143
Лабораторна робота № 3 ... 145
Лабораторна робота № 4 ... 150
Лабораторна робота № 5 ... 154
Лабораторна робота № 6 ... 159
Лабораторна робота № 7 ... 164
Лабораторна робота № 8 ... 167
Лабораторна робота № 9 ... 171
Лабораторна робота № 10 ... 173
**Додатки ... 183**
Список літератури ... 187

# ВСТУП

Задача захисту інформації в комп'ютерних системах перетворюється сьогодні в одну з найактуальніших унаслідок широкої розповсюдженості таких систем, а також розширення локальних і глобальних комп'ютерних мереж, якими передаються величезні об'єми інформації державного, військового, комерційного, приватного характеру, власники якої часто були б категорично проти ознайомлення з цією інформацією сторонніх осіб.

Не менш важливим завданням видається широке впровадження в різні сфери діяльності людини електронного документообігу, який повинен забезпечуватися юридичною чинністю підписаних електронних документів.

Усі ці та багато інших задач захисту інформації покликана розв'язувати криптографія.

Криптографічні механізми настільки тісно пов'язані із сучасними інформаційними технологіями, що разом із підвищенням комп'ютерної грамотності необхідно опановувати основи криптографії.

Грецьке слово *cryptos* перекладається як «таємниця», а отже, криптографія означає тайнопис. Звідси випливає, що початковим завданням криптографії було розроблення методів, спрямованих на приховування змісту переданої або збереженої інформації. І хоча сьогодні сфера застосування криптографічних механізмів значно розширилася, основні ідеї можна проілюструвати саме на прикладі забезпечення конфіденційності інформації.

Існує безліч публікацій, які містять історичні огляди з криптографії (див., наприклад, [1-5]). Зазначимо лише, що на кожному етапові розвитку цивілізації використовувалися відповідні криптографічні пристрої. Тривалий час шифрування текстів виконувалося вручну. Їх створення скоріше нагадує мистецтво, ніж якусь стандартну процедуру. Існують два протилежні погляди на шифри. Відповідно до першого, можна створити шифр, який неможливо розкрити. Другий погляд відбивав таку точку зору: малоймовірно, що «загадку», яка лежить в основі створеного шифру, не можна розгадати. Згодом ***науку про перетворення інформації в незрозумілу для сторонніх осіб форму*** стали називати ***криптографією***, методи пошуку «розгадки» – *криптоаналітичними методами,* а відповідну галузь досліджень – *криптоаналізом*. Отже, ***криптоаналіз – це наука, спрямована на подолання криптографічного захисту***. Тепер дедалі ширше використовують термін ***криптологія,*** тобто наука

про шифри. Вважають, що криптологію складають дві великі частини, які доповнюють одна одну, – ***криптографія*** та ***криптоаналіз***.

Процес криптографічного перетворення інформації ми будемо називати ***шифруванням*** (або ***зашифруванням***, як це часто використовується у криптологічній літературі). Зашифровану інформацію повинні прочитати ті особи, для яких призначена ця інформація. Перш ніж прочитати, її треба перетворити у зрозумілу форму. Цей процес, який ми будемо називати ***розшифруванням***, виконується за допомогою певної секретної частини криптографічної системи – ***криптографічного ключа*** (або просто ***ключа***).

Супротивник, який перехопив зашифровану інформацію, як правило, не має такого ключа. Тому він намагається подолати криптографічний захист за допомогою криптоаналітичних методів. Такий спосіб, тобто розшифрування повідомлення без знання ключа, називатимемо ***дешифруванням***. Отже, можна сказати, що розшифровують «свої», а дешифрують – «чужі».

Методи криптографічного захисту інформації можуть реалізовуватися як апаратно, так і програмно. Апаратна реалізація має значно більшу вартість, однак водночас більшу продуктивність і захищеність. Програмна реалізація практичніша, дешевша та гнучкіша у використанні.

Цю книгу присвячено основам сучасної криптографії, які не пов'язані з державною таємницею.

Книга складається зі вступу, восьми розділів і криптографічного практикуму.

Перший розділ містить короткий екскурс в історію криптографії. Тут висвітлено питання про класичні техніки шифрування, шифри підстановок і перестановок, які далі використовуються у криптографічному практикумі.

У другому розділі розглянуто основні положення, викладені у праці Клода Шеннона «Теорія зв'язку в секретних системах», та сформульовано основні вимоги до сучасних криптосистем.

У третьому розділі обговорюються симетричні криптосистеми (DES, ГОСТ 28147-89) і подано загальні відомості про потокові шифри.

Асиметричні криптосистеми подаються у четвертому розділі. Тут викладено основні поняття теорії чисел і двоключові криптосистеми RSA та Ель-Гамаля.

Алгоритми хешування сімейства MD і стандарти електронного цифрового підпису США й Росії розглянуто в п'ятому розділі.

Шостий розділ книжки присвячено елементам криптоаналізу. Розглянуто атаки на криптосистему RSA, елементи частотного криптоаналізу, подано поняття про різницевий і лінійний криптоаналіз.

Проблеми розповсюдження криптографічних ключів і генерування випадкових та псевдовипадкових послідовностей висвітлено в сьомому й восьмому розділах відповідно.

Задачі та десять лабораторних робіт із курсу подано у криптографічному практикумі. Задачі практикуму охоплюють практично всі основні поняття криптографії: шифри підстановок і перестановок, блокові й потокові шифри, асиметричні криптосистеми, електронний цифровий підпис, елементи

криптоаналізу, генерування криптографічних послідовностей, елементи криптоаналізу.

Лабораторний практикум побудований за принципом «від простого – до складного», у такий спосіб, що сприяє набуттю студентами навичок створення програмного забезпечення з урахуванням вимог до криптографічного захисту інформації. Перші три роботи присвячені різним модифікаціям шифру Цезаря, далі розглядаються питання створення шифру гаммування.

Сучасні алгоритми шифрування представлено спрощеними системами DES і RSA, які використовуються для шифрування й електронного підпису повідомлень.

Роботи 8 та 9 присвячено узгодженню криптографічних ключів за алгоритмом Діффі-Хеллмана та створенню простого потокового шифру на основі генератора псевдовипадкових послідовностей BBS відповідно.

Остання лабораторна робота має, скоріше, прикладний характер. Вона знайомить студентів із криптографічним інтерфейсом CryptoAPI від Microsoft, в якому реалізовано всі найпопулярніші криптографічні алгоритми. Студенти повинні розробити навчальне програмне забезпечення, яке використовує відповідні функції криптографічного інтерфейсу.

Для створення курсу лабораторних робіт із криптографії викладач може обрати 5-6 робіт залежно від мети та рівня підготовки студентів.

# Розділ І
# З історії криптографії

Проблема захисту інформації шляхом перетворення її в незрозумілу для сторонніх осіб форму хвилювала людство споконвіку. Історія криптографії така давня, як історія людської мови. Більше того, сама письмова мова була своєрідною криптографічною системою, оскільки нею на ранніх етапах розвитку людства володіли одиниці. Отже, історія криптографії, як це показано у [1-2], налічує не менше трьох тисяч років.

Прийнято вважати [2], що вона складається з чотирьох великих етапів:

1. Наївна криптографія (до початку XVI сторіччя).
2. Формальна криптографія (поч. XVI сторіччя – поч. XX ст.).
3. Наукова криптографія (30-60-ті рр. XX ст.).
4. Комп'ютерна криптографія (60-ті рр. XX ст. – теперішній час).

Для ***наївної криптографії*** характерне використання будь-яких методів приховування інформації, зокрема *кодування* та *стеганографії*.

Серед криптографічних методів використовувалися *перестановки* та *моноалфавітні підстановки* (заміни). *Шифрами перестановки* називають перетворення, які лише змінюють порядок розташування літер у відкритому тексті. *Шифрами заміни* будемо називати такі перетворення, коли кожна літера вихідного тексту замінюється іншими символами, причому порядок розміщення символів у повідомленні не змінюється.

Одним із перших зафіксованих прикладів використання шифрів є т. зв. шифр сцитала спартанського царя Лісандра. Цей шифр відомий із часів війни між Спартою та Афінами у V ст. до н. е. Для його реалізації використовувалась сцитала – циліндричний жезл певного діаметра. На сциталу намотували вузьку папірусну стрічку, на якій писали повідомлення вздовж осі сцитали. Коли стрічку знімали, на ній залишалися незрозумілі літери. Для розшифровки повідомлення адресат намотував стрічку на жезл такого самого діаметра й читав повідомлення.

Найстаршим шифром підстановки вважається шифр Цезаря, коли кожна літера вихідного тексту замінюється іншою літерою тієї ж абетки. Цезарь використовував третю літеру, розташовану в абетці праворуч від тієї, яку треба замінити (див. лабораторні роботи). Іншим прикладом підстановки є «магічний квадрат», авторство якого приписують грецькому письменнику Полібію.

*Сцитала*

Етап ***формальної криптографії*** (поч. XVI ст. – поч. XX ст.) звичайно пов'язують із появою формалізованих алгоритмів, досить стійких для

*Йоганн Тритеміус*

ручного криптоаналізу. В європейських країнах це сталося в період епохи Відродження, коли розвиток науки й торгівлі потребував надійних способів захисту інформації. На цьому етапі активно використовуються т. зв. матричні шифри, перестановки в яких задаються одним або двома ключовими словами. Важлива роль на цьому етапі належить Леонові Баптисті Альберті, італійському архітектору, який одним із перших запропонував багатоалфавітну підстановку. Цей шифр пізніше названо на честь французького дипломата XVI ст. Блеза Віженера. Метод шифрування полягав у послідовному «додаванні» літер відкритого тексту з ключовим словом. Цю процедуру можна полегшити використанням спеціальної таблиці – таблиці Віженера (див. Додаток А). Лише в 60-х роках XIX ст. офіцер прусських військ Касіскі виявив, що цей шифр піддається частотному аналізу [4].

Однією з перших наукових праць, де сформульовано й узагальнено існуючі на той момент алгоритми шифрування, є «Поліграфія» (1508 рік) німецького абата Йоганна Тритеміуса. Йому належать два важливих відкриття: спосіб заповнення полібіанського квадрата за допомогою ключової фрази та шифрування пар літер (біграм).

Простим та ефективним способом багатоалфавітної заміни (шифрування біграм) є шифр Play-fair (його ще іноді називають шифром Плейфера), який було запропоновано Чарльзом Уїтстоном на початку XIX ст. Цей шифр застосовувався аж до Першої світової війни, оскільки погано піддавався криптоаналізу вручну.

*Блез Віженер*

У XIX ст. голландець Аугуст Керкхоффс сформулював головне правило сучасної криптографії: ***стійкість криптосистеми повинна визначатися секретністю ключа, а не алгоритму***.

У роки Першої світової війни з'явилися подрібнювальні шифри, найвідомішим представником яких був шифр ADFGVX. Шифр ADFGVX був поєднанням перестановок і підстановок. Цей алгоритм розкрито французьким криптоаналітиком Жоржем Пенвеном [1].

1917 рік ознаменований появою багатообіцяючого шифру одноразового блокнота. Його розробили інженери фірми AT&T Г. Вернам та Дж. Моборн і використовували для захисту

телетайпного зв'язку. Цей шифр можна вважати узагальненням шифру Віженера у випадку, коли довжина ключа збігається з довжиною повідомлення. Шифр одноразового блокнота абсолютно надійний, однак незручний у користуванні. Назва шифру походить від того, що агент, який використовував цей шифр, після кожного сеансу зв'язку знищував сторінку шифроблокнота з ключем, на якому виконував шифрування поточного повідомлення.

*Аугуст Керкхоффс*

Найвищим етапом розвитку формальної криптографії були *роторні шифрувальні машини*. Використання таких машин значно ускладнило, а з їх удосконаленням і унеможливило ручний криптоаналіз. Однією з перших подібних систем була винайдена майбутнім президентом США Томасом Джеферсоном у 1790 році механічна роторна машина.

*Джозеф Моборн*

Попередником сучасних роторних шифрувальних пристроїв була машина, винайдена у 1917 році Едвардом Хепберном з Окленда, штат Каліфорнія. Роторними машинами користувалася вся військова криптографія впродовж майже 50 років. Базовими елементами машини є сам ротор і механічне колесо, яке служить для виконання операції підстановки.

У початковому варіанті роторна машина мала клавіатуру та чотири колеса, які оберталися на одній осі. На кожному колесі з лівого і правого боків було по 25 електричних контактів, що відповідали 25 літерам латинської абетки (літери I та J ототожнили). Контакти з лівого і правого боку були попарно з'єднані всередині колеса в певному порядку. Наприклад, ротор можна використовувати для заміни A на L; B на U; C на Z і т. д. Під час роботи машини колеса складалися разом, і їхні контакти, торкаючись один одного, забезпечували проходження електричного сигналу через усі чотири колеса. Наприклад [4-5], у чотирьохроторній машині перший ротор міг замінювати A на L, другий – L на V, третій – V на D; четвертий – D на S. Літера S і є остаточним результатом шифрування відкритого тексту (рис. 1.1). При натисканні літер на клавіатурі ротори проверталися подібно до електролічильника, забезпечуючи в такий спосіб величезний період таблиці замін.

*Едвард Хепберн*

Найвідомішою роторною шифрувальною машиною, звичайно, була німецька *Енігма* (в перекладі – *загадка*), сконструйована Артуром Шербіусом. *Енігма*

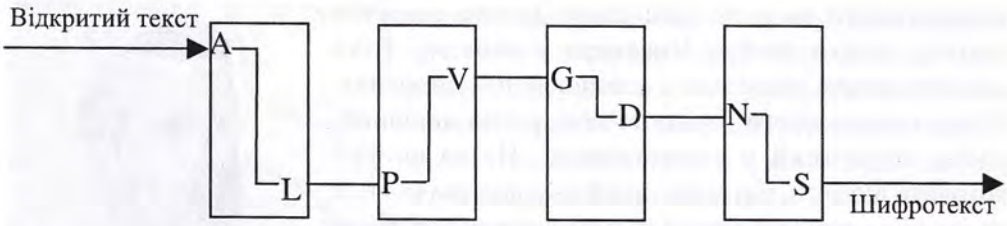



*Рис. 1.1. Схема роторної шифрувальної машини*

складалася з батареї, щоби не залежати від джерела живлення, групи роторів (від трьох до п'яти), які можна було замінювати, та комутаційної панелі, за допомогою якої вхідний текст піддавали перестановкам.

*Німецька шифрувальна машина «Енігма»*

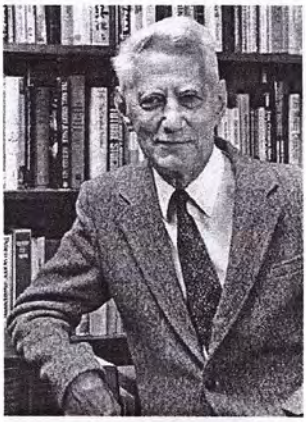
Перед роботою машину треба було налаштувати. Налаштування зводилося до встановлення потрібної групи роторів у певному порядку, який визначався статичним ключем (наприклад, 5-3-4-1-2). Ротори провертали один відносно одного так, щоб у віконці на передній панелі пристрою утворилося кодове слово, і приводили в контакт. Після цього з'єднували потрібні контакти комутаційної панелі для первісного перемішування вхідного тексту. Спосіб з'єднання задавався ще одним статичним ключем. Далі необхідно було встановити ротори в певне початкове положення та перевірити правильність налаштувань, для чого на клавіатурі набирали контрольний текст (який також можна вважати своєрідним ключем) і стежили, щоб отримати на виході системи контрольний зашифрований (або розшифрований) текст. Тепер машина готова до роботи. Працювали на цій машині два шифрувальники. Один набирав на клавіатурі відкритий текст (або шифротекст), а другий записував на папері літери, які висвічувалися на передній панелі пристрою.

Роторні машини активно використовувалися не лише німецькою армією. Відомі аналогічні апарати *Sigaba* (США), *Typex* (Велика Британія), *Red*, *Orange*, *Purple* (Японія). Роторні шифрувальні системи – вершина формальної криптографії, оскільки відносно просто реалізовували дуже стійкі, як на той час, шифри. Успішні криптоатаки на роторні машини стали можливими лише з винаходом перших комп'ютерів у 40-х роках XX ст.

Етап ***наукової криптографії*** характеризується появою криптосистем із математично обґрунтованою криптостійкістю. До початку 30-х років XX ст. остаточно сформувалися такі розділи математики, як теорія ймовірностей і математична статистика, загальна алгебра, теорія чисел; почали активно розвиватися теорія алгоритмів, теорія інформації та кібернетика.

Епохальною в цьому розумінні стала книга ***Клода Шеннона «Теорія зв'язку в секретних системах»*** (1949), де сформульовано основні теоретичні принципи

криптографічного захисту інформації. К. Шеннон увів поняття «розсіювання», «перемішування», «надмірності мови», обґрунтував можливість створення абсолютно стійких криптосистем [6]. Матеріал статті первісно був викладений у таємній доповіді «Математична теорія криптографії» (1945 р.), яку розсекретили після закінчення Другої світової війни.



*Клод Шеннон*

Стаття розділена на три частини. У першій наведено математичні основи побудови секретних систем. Секретну систему визначено абстрактно, як деяку множину відображень одного простору (множина можливих повідомлень) в іншому (множину можливих шифрограм). Кожне конкретне відображення з цієї множини відповідає способу шифрування за допомогою конкретного криптографічного ключа. З кожною мовою пов'язаний деякий параметр, який К. Шеннон назвав надмірністю мови. Надмірність визначає, наскільки можна скоротити певне повідомлення без втрати будь-якої частини інформації. Надмірність мови відіграє центральну роль у криптології. Зокрема, саме надмірність мови дозволяє використовувати для розкриття шифрів принципи частотного криптоаналізу.

У другій частині статті розглянуто проблему теоретичної секретності, тобто наскільки легко конкретна криптосистема піддається розкриттю за умови, що для аналізу перехоплених шифрограм супротивник має необмежені часові та матеріальні ресурси. «Ідеальну секретність» К. Шеннон визначає такими вимогами до системи: щоб апостеріорні ймовірності різних повідомлень, отримані після перехоплення супротивником певної шифрограми, точно дорівнювали апріорним імовірностям тих же повідомлень до перехоплення. Ідеальний шифр, за твердженням К. Шеннона, існує, однак вимагає у випадку кінцевої кількості повідомлень такої самої кількості ключів. При цьому довжина ключа повинна дорівнювати довжині повідомлення. Крім того, кожний ключ повинен використовуватися лише один раз. У цьому ж розділі К. Шеннон уводить поняття «інтервал єдиності», тобто мінімальної довжини шифрограми, за якої можливе однозначне за змістом її дешифрування.

Третя частина присвячена «практичній секретності». Дві криптосистеми з однаковою довжиною ключа можуть бути розв'язані в один спосіб, однак значно відрізняються необхідними для цього часовими та матеріальними ресурсами. На основі аналізу недоліків алгоритмів пропонуються методи побудови систем, для розкриття яких потрібні великі часові та матеріальні витрати. Розглядається й проблема несумісності різних вимог до криптосистем.

У 60-х роках XX ст. провідні криптографічні школи впритул підійшли до створення *блокових шифрів*, ще стійкіших за роторні машини, однак їх практична реалізація можлива лише за допомогою комп'ютерної техніки.

*Хорст Фейстель*

*Уїтфілд Діффі*

*Вінсент Ріймен*

***Комп'ютерна криптографія*** зумовлена появою потужних і компактних обчислювальних пристроїв, використання яких уможливило розробки блокових шифрів. З-поміж перших був американський стандарт шифрування DES (прийнятий 23 листопада 1977 р.). Один із його авторів, Хорст Фейстель з IBM розробив алгоритм, призначений для апаратної реалізації (т. зв. *сітка Фейстеля*). Характерні риси: обробка за один цикл лише половини блока тексту; робота з бітами, а не з байтами інформації (що сповільнює його програмну реалізацію); можливість використання одного набору мікросхем як для зашифрування, так і для розшифрування повідомлень. Первісний алгоритм під назвою *Люцифер*, розроблений математиками фірми IBM на чолі з Х.Фейстелем, був поданий до Агентства національної безпеки США (АНБ) для аналізу. АНБ, модифікувавши статичні ключі алгоритму, скоротило ключ шифрування з 128 до 64 бітів (8 з яких використовувалися як біти парності, так що секретна частина ключа складала всього 56 бітів). Такий модифікований алгоритм і було прийнято як стандарт шифрування DES. При цьому, як випливає з повідомлень у пресі, АНБ недооцінило темпи розвитку комп'ютерної техніки, що призвело до надто швидкого «старіння» алгоритму. Вже на початку 90-х років постала загроза організації атаки прямого перебору ключів за допомогою розподілених обчислень. Отже, алгоритм вже не міг використовуватися як стандарт шифрування США. АНБ оголосило всесвітній конкурс криптографічних алгоритмів на звання нового стандарту шифрування. У 2000 році як новий стандарт, який отримав назву AES, було прийнято алгоритм Rijndael бельгійських криптографів Вінсента Ріймена та Йоана Даймена. Цей алгоритм використовує групу точок еліптичної кривої над кінцевим полем Галуа. Увесь блок вхідного тексту повністю обробляється за один цикл. Окрім того, AES працює зі змінною довжиною блока та ключа, що дає можливість зміни криптостійкості в залежності від ступеня секретності інформації.

Стандарт DES був не єдиним алгоритмом, який використовував сітку Фейстеля. Одним із найбільш стійких таких алгоритмів є ГОСТ 28147-89 [7]. Його взято за стандарт шифрування СРСР у 1989 році. При проектуванні цього алгоритму криптографи КДБ узагальнили досвід десятирічного використання

DES, значно збільшивши криптостійкість за допомогою використання 32 циклів шифрування та 256-бітного ключа. Це дало змогу спростити раундову функцію та процедуру розгортання ключа. Необхідно сказати, що досі немає повідомлень про успішний криптоаналіз ГОСТу 28147-89. Алгоритм і сьогодні використовується для захисту інформації в Росії та Україні, в тому числі для інформації з найвищим ступенем секретності.

*Мартін Хеллман*

У 1976 році сталася ще одна видатна подія в галузі криптографії: було розроблено основні принципи *асиметричної криптографії*. Започаткувала це праця Уїтфілда Діффі та Мартіна Хеллмана «Нові напрямки сучасної криптографії» [8]. Тут вперше сформульована можливість обміну зашифрованими повідомленнями без обміну секретними ключами шифрування. Кількома роками пізніше Рональд Рівест, Аді Шамір і Леонард Аделман розробили асиметричну криптосистему RSA (названа за першими літерами прізвищ авторів), крипостійкість якої ґрунтується на складності розкладання великого числа на прості множники. Асиметрична криптографія, крім розв'язання проблеми розповсюдження ключів, основної слабкості симетричних криптосистем, відкрила одразу кілька нових прикладних напрямків, зокрема системи *електронного цифрового підпису* та *електронних грошей*.

*Рональд Рівест*

*Аді Шамір*

У 80-90-х роках XX ст. розробляються абсолютно нові революційні ідеї в галузі криптографічного захисту інформації: квантова криптографія та ймовірнісне шифрування, які й сьогодні є авангардом криптографічної науки.

Актуальним залишається й удосконалення симетричних та асиметричних криптосистем. Було розроблено нефейстелівські симетричні крипто-алгоритми (SAFER, RC4-RC6 тощо), альтернативні асиметричні системи (Ель-Гамаль, Меркл-Хеллман тощо).

*Леонард Аделман*

В Україні зараз очікується бум криптографічних технологій. Окрім ГОСТу 28147-89, сертифікованого ще за часів Радянського Союзу симетричного алгоритму, починається розробка асиметричних алгоритмів для виконання електронного цифрового

підпису та створення умов електронного документообігу. Наприклад, програмно-технічний комплекс центру сертифікації ключів професора Горбенка І. Д. був нагороджений почесною відзнакою Департаменту спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України і рекомендований до використання державними службами України, які переходять до електронної звітності. Заданий криптографічний алгоритм використовує групу точок еліптичної кривої над кінцевим полем Галуа.

Дізнатися більше про історію криптографії можна, прочитавши чудову книжку Девіда Кана «Взломщики кодов» [1].

**Класичні техніки шифрування**

Розглянемо деякі класичні техніки шифрування.

**Шифри перестановок**

1. Шифр частоколу

Одним із найпростіших шифрів перестановки є т. зв. шифр частоколу. Він дуже схожий на матричний шифр. Наприклад, зашифруємо цим шифром із висотою частоколу 2 слово «*зашифрування*». Для цього запишемо його так:

а и р в н я
з ш ф у а н

Тепер зчитуємо спочатку верхній рядок: «*аирвня*», а потім нижній: «*зшфуан*». Отже, зашифроване повідомлення «*аирвнязшфуан*».

Для частоколу висотою 3 отримаємо такий зашифрований текст:

ш р а я
а ф в н – «*шраяафвнзиун*»
з и у н

Шифр частоколу дуже нагадує інший шифр перестановок – матричний. Власне, це і є матричний шифр у два чи три рядки.

2. Матричний шифр

Відкритий текст записують послідовно рядок за рядком у матрицю. Літери криптограми виписують із цієї ж матриці по стовпчиках. Для прикладу зашифруємо текст «*використовуйте канал зв'язку номер чотири*» на матриці 6 × 6 [4].

Зчитуючи літери по стовпчиках, отримаємо шифрограму: «*встлуи итезно коквот оваями рунзер ийакри*». Зрозуміло, що криптостійкість такого шифру зовсім незначна, однак її можна легко підсилити. Для цього використовують ключові слова. Зашифруємо цю ж фразу на ключах «*літера*» та «*захист*». Ключі записуємо згори та зліва від матриці тексту, після чого

| | | | | | |
|---|---|---|---|---|---|
| в | и | к | о | р | и |
| с | т | о | в | у | й |
| т | е | к | а | н | а |
| л | з | в | я | з | к |
| у | н | о | м | е | р |
| ч | о | т | и | р | и |

переставляємо рядки та стовпчики згідно з позицією кожної літери ключів у абетці. Отже, матриця з відкритим текстом перетворюється так:

| | л | і | т | е | р | а |
|---|---|---|---|---|---|---|
| з | в | и | к | о | р | и |
| а | с | т | о | в | у | й |
| х | т | е | к | а | н | а |
| и | л | з | в | я | з | к |
| с | у | н | о | м | е | р |
| т | ч | о | т | и | р | и |

→

| | а | е | ш | л | р | т |
|---|---|---|---|---|---|---|
| а | й | в | т | с | у | о |
| з | и | о | и | в | р | к |
| и | к | я | з | л | з | в |
| с | р | м | н | у | е | о |
| т | и | к | о | и | р | т |
| х | а | а | е | т | н | к |

Отримаємо шифротекст «*йикриа воямка тизное свлуит урзерн оквотк*».

Розшифрувати шифрограму можна, записавши слова в матрицю по стовпчиках згідно з порядком розташування літер ключів, а потім, переставивши стовпчики та рядки так, щоби ключі утворили зв'язні слова, зчитуємо розшифрований текст по рядках.

Дешифрувати повідомлення, не знаючи ключів, можна, якщо проаналізувати записаний по стовпчиках шифротекст за частотою появи пар літер, зважаючи на те, що деякі пари літер ніколи не зустрічаються або зустрічаються в українській мові дуже рідко.

3. Шифр Першої світової війни (ADFGVX-шифр)

Модифікацією матричного шифру можна вважати ADFGVX-шифр, який використовували під час Першої світової війни. Це комбінація підстановки та перестановки за ключовим словом. Припустимо, для прикладу, що необхідно зашифрувати текст «*monday is the key change day*" (понеділок – день зміни ключа). Використаємо таблицю 6 × 6, в яку впишемо всі латинські літери та цифри від 0 до 9.

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

Для зашифрування фрази знаходимо почергово її літери в таблиці та вибираємо літери, які стоять у заголовках рядка та стовпчика. Отже, *m* перетворюється в *AD*, *o* – в *DA*, *n* – в *AF* і т.д. У результаті отримуємо таку шифрограму:

*ADDAAFXFGVGFFFDGGAVGAXDDAXGFAAVGGVAFXXAXXFGVGF*.

Аналогічно можна створити схожі шифри для української і російської мов.

**Шифри підстановок**

1. Шифр Цезаря

Цей шифр реалізує таке перетворення відкритого тексту: кожна літера замінюється третьою після неї літерою тієї ж абетки, яка вважається написаною по колу, тобто після „я" йде „а". Зауважимо, що Юлій Цезарь замінював кожну літеру третьою за нею літерою, але можна замінювати й будь-якою іншою. Головне, щоб адресат повідомлення знав величину і напрямок цього зсуву.

Криптостійкість шифру Цезаря можна підсилити, побудувавши таблицю заміни за ключовим словом або використавши т.зв. афінну систему Цезаря, коли величина зсуву різна для кожної літери повідомлення. Цим шифрам присвячена робота № 3 лабораторного практикуму з криптографії та практичне заняття.

2. Шифр пар

Шифр пар використовує ключову фразу, яка легко запам'ятовується та містить, як правило, близько половини літер абетки. Для того, щоб зашифрувати повідомлення, роблять так. Ключову фразу записують в один рядок, причому літери, які зустрічаються кілька разів, вилучаються. У випадку української мови, коли в абетці 32 літери (г та ґ ототожнюються), 16 літер ключової фрази записують у верхньому рядку, а решту – в нижньому рядку в порядку їх розміщення в абетці. Отже, ми створили таблицю заміни, яку тепер можна використати для зашифрування повідомлення.

Наприклад, створимо таблицю заміни на ключовій фразі «Роторна шифрувальна машина Енігма».

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| р | о | т | н | а | ш | и | ф | у | в | л | ь | м | е | і | г |
| б | д | є | ж | з | ї | й | к | п | с | х | ц | ч | щ | ю | я |

Зашифруємо за допомогою цієї таблиці заміни повідомлення «*Чекайте літак завтра опівночі*». Шифротекст буде мати вигляд: «*Мщфзиєщ хюєзф азсєбз дуюсждмю*». Для розшифрування криптограми використовують таку саму таблицю, замінюючи за нею літери шифротексту.

3. Квадрат Полібія

Ще одним прикладом шифрів підстановки є квадрат, що отримав ім'я грецького письменника Полібія, – полібіанський квадрат [9]. Уперше він був розроблений для грецької абетки. Для української мови квадрат Полібія може мати такий вигляд:

| і | ж | ю | о | в | а |
|---|---|---|---|---|---|
| з | н | и | ш | ь | я |
| б | д | л | ф | є | ч |
| м | е | щ | й | п | г |
| к | у | т | х | ї | с |
| ц | р | . | , | - | ; |

Для зашифрування повідомлення квадрат використовується так. Кожна літера відкритого тексту замінюється на літеру, що знаходиться саме під нею. Якщо літера стоїть в останньому рядочку, вона замінюється такою, що стоїть у першому рядочку саме над нею. Отже, повідомлення «Зустріч перенесено на завтра» перетворюється у «*Бр;.жзг їужуду;удш дя бяь.жя*».

4. Шифр Play-fair

Шифр Play-fair розроблено для англійської абетки [10]. Двадцять п'ять літер (*i* та *j* ототожнювалися) у випадковому порядку розміщувалися у квадраті розміром 5×5. Для зашифрування використовують від одного до чотирьох квадратів. Якщо використовується один квадрат, він називається «магічним». Із чотирма квадратами криптостійкість шифру зростає, оскільки невідомим залишається розміщення літер у чотирьох, а не в одному квадраті. Крім того, не буває критичних ситуацій, коли обидві літери знаходяться в одному рядку або стовпчику.

Реалізуємо варіант цього шифру для української абетки. Використаємо для цього чотири квадрати 6×6, заповнивши їх у випадковому порядку літерами та знаками пунктуації.

Для прикладу зашифруємо повідомлення «*Чекайте літак завтра опівночі*». Розіб'ємо його на пари літер: «*Че ка йт ел іт ак за вт ра оп ів но чі*». Щоби зашифрувати пару «*Че*», шукаємо «*Ч*» у першому квадраті, «*е*» – в четвертому. Вони позначені в таблиці штриховкою. Ці літери утворюють прямокутник і знаходяться на його діагоналі (вона позначена стрілкою). На іншій діагоналі знаходяться літери «*Д*» та «*ц*». Отже, пара «*Че*» замінюється на «*Дц*». Подібним чином «*ка*» переходить в «*іщ*», «*йт*» – у «*нп*» і так далі.

У результаті отримаємо криптограму:

«*Дціщнпсвутаф-щнтцщйьфчшцйш*».

| а | р | т | к | з | ю | г | щ | ї | ф | ю | с |
|---|---|---|---|---|---|---|---|---|---|---|---|
| п | й | є | х | м | с | ґ | ж | д | а | п | й |
| ш | у | ч | б | н | ґ | є | х | м | ш | ц | і |
| ц | е | г | щ | ї | ф | я | у | е | в | и | . |
| і | в | о | л | - | ж | ь | о | ч | є | т | к |
| я | и | ь | . | , | д | , | л | н | ї | р | - |
| а | ю | д | я | р | з | й | ц | у | к | ш | н |
| , | и | т | к | . | ь | г | ш | щ | з | х | ї |
| п | с | ж | і | ш | ґ | є | ж | д | л | о | р |
| ф | ц | й | м | - | в | п | а | в | і | ф | я |
| у | н | ї | е | ч | б | ч | с | м | и | т | ь |
| щ | г | х | є | о | л | б | ю | . | , | ґ | - |

5. Шифр Віженера

Шифр Віженера, як уже згадувалося, належить до багатоалфавітної заміни. Використовується відкрите повідомлення та ключове слово або фраза. Якщо довжина ключового слова не збігається з довжиною повідомлення, слово повторюють кілька разів. Зашифровують повідомлення за допомогою так званої таблиці Віженера, яку легко скласти самостійно (див. Додаток А).

Припустимо, треба зашифрувати повідомлення «*Переходьте до виконання плану номер два*» на ключі «*резидент*».

Як видно з наведеної таблиці, шифрування виконується так. Літеру повідомлення, наприклад «п», шукають у лівому стовпчику таблиці Віженера. Літеру ключового слова, наприклад «*р*», шукають у верхньому рядку. На перетині рядка з літерою «*п*» і стовпчика з літерою «*р*» знаходиться літера «*е*». Отже, після зашифрування отримаємо такий шифротекст: «*еішлдукпзійчєльєгехцгфютгшхчрігцте*».

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| р | е | х | о | д | ь | т | е | д | о | в | и | к | о | н | а | н | н | я | п | л | а | н | у | н | о | м | е | р | д | в | а |
| з | и | д | е | н | т | р | е | з | и | д | е | н | т | р | е | з | и | д | е | н | т | р | е | з | и | д | е | н | т | р | е | |
| ш | л | щ | у | к | п | з | і | й | ч | є | л | ь | є | г | е | х | ц | г | ф | ю | т | г | ш | х | ч | р | і | г | ц | т | е | |

Процес розшифрування виконується так. У стовпчику зліва шукаємо літеру ключа, наприклад «*р*». Далі шукаємо в рядочку із цією «*р*» літеру шифротексту «*е*» і дивимося, в якому стовпчику вона стоїть. Заголовок цього стовпчика, в нашому випадку «*п*», і буде розшифрованим текстом.

Замість таблиці Віженера можна скористатися такими міркуваннями. Відкритий текст $M$ зобразимо як сукупність літер $\{m_i\}$. Аналогічно ключ також запишемо як сукупність літер $\{k_i\}$. Тоді кожна літера шифрограми $c_i$ отримується зі співвідношення

$$c_i = (m_i + k_i) \bmod 32 .$$

Легко переконатися, що результат шифрування буде той самий, що й за допомогою таблиці. Формулу для розшифрування легко розробити самостійно.

Як бачимо, навіть однаковим літерам відкритого тексту можуть відповідати різні літери шифротексту, що значно ускладнює дешифрування. Однак це не означає, що цей шифр не можна дешифрувати за допомогою аналізу частот появи різних літер та їх блоків у зашифрованому тексті.

Класичні техніки шифрування сьогодні втратили свою цінність як самостійні шифри. Криптостійкість їх дуже незначна, до того ж існує багато програмних засобів, які легко їх розкривають. Однак вони не втратили своєї актуальності, оскільки, по-перше, з них варто починати вивчення криптографії, а по-друге, вони можуть бути складовими частинами складніших сучасних систем шифрування, які розглядаються далі.

# Розділ II
# Елементи теорії зв'язку в секретних системах

## Клод Шеннон. «Теорія зв'язку в секретних системах»

Клод Елвуд Шеннон, а точніше, його книга «Теорія зв'язку в секретних системах» [6, 9], внесла визначальний внесок у сучасну криптографічну науку. Вважається, що зазначена праця, яка була спершу його секретною доповіддю «Математична теорія криптографії» (1945 р. – розсекречено після Другої світової війни, у 1949 році), визначила основи та сформувала обличчя сучасної криптографії. Дехто порівнює його внесок із впливом на фізику праць Ісаака Ньютона.

К. Шеннон народився у 1916 році в м. Гейлорді (штат Мічиган, США). У 1936 році він закінчив Масачусетський технологічний інститут, спеціалізуючись одночасно в математиці та електротехніці. Саме таке поєднання дозволило у 1940 році захистити докторську дисертацію, де він уперше застосував для опису роботи реле та перемикачів булеву алгебру. Зараз такий аналіз лежить в основі сучасної цифрової схемотехніки, а на той час це було майже революційною справою. Сам К.Шеннон на це скромно зауважував: «Просто так склалося, що ніхто інший не був знайомий з обома галузями одночасно».

У 1941 році К.Шеннона запросили на роботу в Bell Laboratories, де в роки війни він займався розробкою криптографічних систем, що пізніше допомогло йому відкрити методи кодування з корекцією помилок. Метою К.Шеннона була оптимізація передавання інформації телефонними та телеграфними лініями. Для того, щоби розв'язати цю проблему, йому довелося сформулювати, що ж таке інформація, чим визначається її кількість. У працях 1948-1949 років він визначив кількість інформації через ентропію – величину, яка використовується в термодинаміці та статистичній фізиці як міра розупорядкованості системи, а за одиницю інформації – величину, яку потім було названо «бітом», тобто вибором з двох рівноймовірних варіантів.

К. Шеннон розглядає шифрування як відображення відкритого тексту в шифрограмі [6, 11]:

$$C = F_i(M),$$

де C – шифрограма, $M$ – відкритий текст, $F_i$ – відповідне відображення. Індекс $i$ відповідає конкретному криптографічному ключу, використаному при шифруванні. Для того, щоб існувала можливість однозначного розшифрування повідомлення, відображення $F_i$ повинно мати єдине обернене відображення $F_i^{-1}$, таке, що $F_i F_i^{-1}=I$, де $I$ – тотожне перетворення:

$$M= F_i^{-1}(C).$$

Тут враховано, що джерело ключів є статистичним процесом або пристроєм, що задає відображення $F_1, F_2, \ldots, F_N$ з імовірностями $p_1, p_2, \ldots, p_N$.

Розглянемо найпростіший шифр, де вихідний алфавіт повідомлень збігається з множинами знаків ключа та криптограми, а шифрування виконується послідовною заміною знаків відкритого тексту знаками криптограми залежно від чергового значення знаків ключа [12]. В такому разі відкритий текст, ключ і криптограма є послідовностями літер того самого алфавіту: $M = (m_1 m_2 m_3 \ldots m_n)$; $K = (k_1 k_2 k_3 \ldots k_n)$; $C = (c_1 c_2 c_3 \ldots c_n)$. Кожен крок шифрування визначається співвідношенням $c_i = f(m_i, k_i)$.

У практичних криптосистемах довжина ключа значно менша за довжину відкритого тексту, тому часто ключова послідовність обчислюється за допомогою деякого первісного ключа меншого розміру або навіть може бути періодичною.

Задача криптоаналітика полягає в обчисленні відкритого тексту за криптограмою, знаючи множину відображень $F_1, F_2, \ldots, F_N$. Існують криптосистеми, для яких будь-який об'єм перехопленої інформації недостатній для знаходження шифрувального відображення, причому ситуація не залежить від обчислювальної потужності обладнання криптоаналітика. Шифри такого типу називаються ***безумовно стійкими*** (за К. Шенноном – ***ідеально секретними***). Строго кажучи, безумовно стійкими будуть такі шифри, для яких криптоаналітик (навіть маючи безмежні обчислювальні ресурси) не зможе поліпшити оцінку вихідного повідомлення (відкритого тексту) $M$, знаючи криптограму $C$, порівняно з оцінкою при невідомій криптограмі. Це можливо лише тоді, коли $M$ і $C$ статистично незалежні. Безумовно стійкі криптосистеми існують, що легко показати [6, 12]. Нехай у розглянутому вище простому шифрі використовується алфавіт із $L$ літер, а поточні знаки криптограми генеруються за законом

$$c_i = f(m_i, k_i) = (m_i + k_i) \bmod L,$$

де кожному знакові $c_i$, $m_i$, $k_i$ поставлено у відповідність їх порядковий номер у алфавіті.

Оберемо як ключ послідовність із $n$ випадкових знаків $k_1 k_2 k_3 \ldots k_n$, тобто оберемо випадковий ключ, розмір якого дорівнює довжині повідомлення. Для генерування ключа використаємо деякий фізичний генератор випадкових чисел, що забезпечує рівну ймовірність кожного елемента з множини чисел $\{1, 2, \ldots, N\}$. Обране нами джерело забезпечує рівноймовірність вибору будь-якого ключа довжини $n$. У такому разі ймовірність вибору ключа довжини $n$ складає:

$$P(K=K_i) = L^{-n}.$$

Із цього виразу видно, що для довільних $M$ і $C$ виконується аналогічне співвідношення:

$$P(M=M_i / C=C_i) = L^{-n}.$$

А це, у свою чергу, означає, що криптограмі довжини $n$ з імовірністю $L^{-n}$ може відповідати будь-який відкритий текст довжини $n$. Для шифрування іншого повідомлення оберемо інший випадковий ключ. Така процедура шифрування

забезпечує безумовну стійкість. Криптосистеми, що використовують рівноймовірний випадковий ключ, який має однакову з відкритим текстом довжину, називаються шифрами зі ***стрічкою одноразового використання*** – одноразовим блокнотом або шифрами з ***безмежною ключовою гамою***. На практиці такі системи отримали лише обмежене використання, оскільки досить незручні у використанні.

Криптосистеми іншого типу характеризуються тим, що при зростанні кількості доступної для криптоаналітика інформації, при певному значенні $n = n_0$ існує єдиний розв'язок криптоаналітичної задачі. Мінімальний об'єм криптограми, для якого існує єдиний розв'язок, називається ***інтервалом єдиності***. У випадку стрічки одноразового використання $n_0 \to \infty$. За кінцевої довжини криптографічного ключа значення $n_0$ кінцеве. Відомо, що за криптограмою довжиною, більшою за інтервал єдиності, можна знайти цей єдиний розв'язок. Однак для криптоаналітика з обмеженими обчислювальними ресурсами ймовірність знайти цей розв'язок за скінченний проміжок часу (поки інформація має ще якусь цінність) надзвичайно мала ($10^{-30}$ і менша). Шифри такого типу називаються ***умовно стійкими*** (практично стійкими, за К. Шенноном). Їх стійкість ґрунтується на значній обчислювальній складності розв'язку криптоаналітичної задачі.

Мета розробника умовно стійких криптосистем полягає в тому, щоби зменшити витрати на процедури шифрування та розшифрування й одночасно задати такий рівень складності криптоаналітичної задачі, щоб для успішного розв'язку її потрібно було залучити ресурси, вартість яких перетворювала б знаходження рішення в економічно невигідну задачу.

Завдання такого об'єму обчислень називаються важкими або обчислювально складними, а про їх розв'язок говорять, що вони обчислювально нездійсненні. Шифри, які ґрунтуються на обчислювально нездійсненних задачах, називаються ***обчислювально стійкими***. Найбільшу практичну розповсюдженість мають саме обчислювально стійкі криптосистеми.

Під стійкістю криптосистем такого роду будемо розуміти складність розв'язку криптоаналітичної задачі при певних умовах. К. Шеннон увів поняття ***робочої характеристики*** $W(n)$ шифру як середню кількість роботи для знаходження ключа за відомими $n$ знаками криптограми з використанням найкращого алгоритму криптоаналізу. Кількість роботи можна виміряти, наприклад, кількістю операцій, які необхідно виконати для обчислення ключа. Цей параметр безпосередньо пов'язаний з алгоритмом обчислення ключа. Складність визначення $W(n)$ пов'язана зі складністю знаходження найкращого способу розкриття. Особливо цікаве граничне значення $W(n)$ для $n \to \infty$. Сьогодні невідомі обчислювально стійкі криптосистеми, для яких обчислено $W(\infty)$. Унаслідок складності такої оцінки, практичні шифри характеризують досягнутою оцінкою робочої характеристики, яку отримують для найкращого з відомих сьогодні методів обчислення ключа.

К. Шеннон запропонував також модель для оцінки інтервалу єдиності, з якої отримано співвідношення

$$n_0 = H(K)/D,$$

де $H(K)$ – ентропія ключа, яка для випадкового ключа дорівнює довжині ключа в бітах; $D$ – надмірність мови, що вимірюється у бітах на знак. Це співвідношення можна записати у вигляді

$$H(K) \le nD,$$

де $H(K)$ характеризує кількість невідомих у двійковому представленні ключа; $nD$ – кількість рівнянь для обчислення ключа. Якщо кількість рівнянь менша за кількість невідомих, розв'язок системи буде неоднозначним. В таких умовах криптосистема буде безумовно стійкою. Якщо кількість рівнянь більша від кількості невідомих, то існує єдиний розв'язок, а криптосистема не вважається безумовно стійкою. Однак вона може бути обчислювально стійкою, якщо $n >> n_0$. Рівень стійкості обчислювально стійких криптосистем залежить від типу шифрувальних процедур (за винятком вибору дуже малого ключа шифрування, коли складність повного перебору можливих ключів дуже мала). Конкретні процедури перетворення також визначає хід робочої характеристики, тобто явний вигляд залежності $W(n)$.

## Класифікація сучасних криптосистем. Вимоги до сучасних криптосистем

Позначимо процес зашифрування відкритого повідомлення $M$ на ключі $K$ через $E_K(M)$ (від англійського *encryption* – зашифрування). Тоді отримання шифрограми $C$ можна зобразити як

$$C = E_K(M).$$

У такому разі процес розшифрування (*decryption*), тобто отримання відкритого повідомлення $M$ із криптограми $C$ за допомогою відомого ключа $K$, можна позначити:

$$M = D_K(C).$$

Функція $D_K$ повинна бути оберненою до $E_K$, тобто $D_K = E^{-1}_K$ в тому розумінні, що при правильному ключі $K$ дозволяє отримати відкритий текст $M$.

Отже, процес інформаційного обміну схематично можна зобразити так:

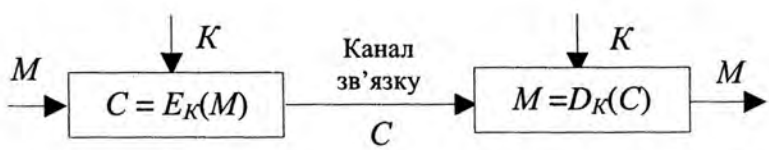



*Рис. 2.1. Схема роботи симетричної криптосистеми*

На передавальній стороні виконують шифрування відкритого повідомлення $M$ за допомогою функції шифрування $E_K(M)$ на ключі $K$ та отримують криптограму $C$, яку передають відкритим каналом зв'язку. На приймальній стороні, отримавши зашифроване повідомлення $C$, застосовують до нього обернене перетворення $D_K(C)$ і отримують відкрите повідомлення $M$ при використанні такого самого ключа $K$, як і на передавальній стороні, але якщо зашифроване повідомлення не було змінено під час проходження каналом зв'язку.

Як бачимо, для успішної роботи такої криптосистеми учасники інформаційного обміну повинні завчасно домовитися про алгоритми шифрування та розшифрування, а також про те, як приймальна сторона дізнається про криптографічний ключ. Оскільки ключ, що використовується для зашифрування та розшифрування повідомлень, однаковий, такі криптосистеми отримали назву ***симетричних***. Симетричні криптосистеми здебільшого використовують перетворення тексту, які є комбінацією ***перестановок*** і ***замін***. Симетричні криптосистеми використовуються вже багато років і вважаються найбільш дослідженими.

Основними *перевагами* найпопулярніших симетричних криптосистем вважають [10, 13]:

- високу швидкість роботи (отже, можливість швидкого шифрування великих потоків інформації в каналах зв'язку);
- забезпечення високого ступеня секретності, оскільки симетричні криптосистеми давно відомі та добре досліджені, ймовірність знайти нові вразливі місця в цих системах незначна;
- можливість використання однакових апаратних засобів для зашифрування та розшифрування інформації.

Однак застосування симетричних криптосистем на практиці потребує подолання двох значних *недоліків*:

- проблеми розповсюдження та зберігання криптографічних ключів, оскільки він є секретною частиною криптосистеми як на передавальній, так і на приймальній сторонах. Компрометація ключа створює загрозу для всієї системи;
- велика кількості криптографічних ключів, необхідних для створення шифрованого інформаційного обміну між багатьма учасниками. Для того, щоби $n$ осіб могли обмінюватися секретною інформацією, необхідно згенерувати та розповсюдити $n(n\text{-}1)/2$ криптографічних ключів.

Одним із найефективніших способів подолання недоліків симетричних криптосистем стало винайдення асиметричних способів шифрування, коли для зашифрування й розшифрування використовуються різні криптографічні ключі. Криптостійкість таких систем ґрунтується на «*односторонніх функціях*», які легко обчислюються в прямому напрямку й утворюють математичну проблему надзвичайної обчислювальної складності при спробі розв'язку оберненої задачі.

Як правило, для зашифрування інформації використовують так званий ***відкритий,*** або ***публічний ключ*** (хоча в деяких випадках його можна використати і для розшифрування), який непотрібно приховувати. Навпаки, цей ключ розміщують на ресурсі, доступному для всіх учасників інформаційного обміну та захищають від підміни. Кожен, хто хоче написати конфіденційного листа власникові публічного ключа, використає його для зашифрування, а власник ключа, маючи парний до цього публічного ***приватний*** (або ***секретний***) ключ, зможе розшифрувати повідомлення. Особливістю такої криптосистеми є те, що за допомогою публічного ключа неможливо розшифрувати зашифроване на ньому повідомлення. Таким чином, ніхто, крім власника приватного ключа, не зможе прочитати призначене йому повідомлення. Приватний ключ непотрібно розповсюджувати, він зберігається після генерування у власника й використовується лише для розшифрування (хоча можливий і обернений варіант використання криптосистеми в залежності від задач).

Отже, процес такого інформаційного обміну схематично можна зобразити так:

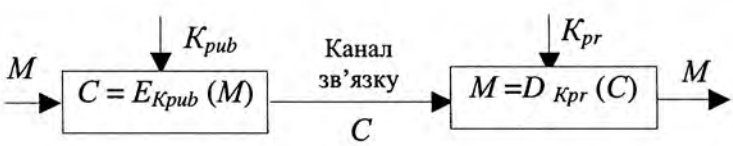



*Рис. 2.2. Схема роботи асиметричної криптосистеми*

Криптосистеми, в яких для зашифрування та розшифрування повідомлень використовуються різні криптографічні ключі, називаються ***асиметричними***.

Асиметричні криптосистеми розв'язали основну проблему симетричних криптосистем, проблему розповсюдження криптографічних ключів, адже для зашифрування інформації багатьма користувачами потрібно мати лише одну пару ключів: публічний і парний йому приватний.

Однак, подолавши недоліки розповсюдження ключів, ми отримали додаткові *хиби, що притаманні асиметричним системам*:

- мала швидкість роботи асиметричних криптоалгоритмів (прийнято вважати, що ці системи приблизно в 1000 разів повільніші за симетричні [14]);
- нестійкість асиметричних криптосистем проти атаки підміни публічного ключа;
- досі математично строго не доведено, що не існує простого алгоритму для отримання приватного ключа з публічного.

Цей перелік потребує коментарів. Перший пункт є наслідком специфічних математичних перетворень, на базі яких ґрунтуються всі асиметричні криптосистеми, та великої довжини ключів: такі задачі досить повільно виконуються на комп'ютері.

Розміщення публічного ключа на відкритому ресурсі може призвести до наступної атаки. Супротивник генерує власну пару ключів і заміняє публічний ключ системи своїм. Зрозуміло, що тепер він може читати всі повідомлення, які зашифровані його відкритим ключем, контролюючи канал обміну інформацією. Прочитавши повідомлення, він ***перешифровує*** їх старим відкритим ключем і відправляє адресату. Той, у свою чергу, розшифровує повідомлення своїм приватним ключем і, оскільки сеанс розшифрування проходить успішно, не помічає ***підміни публічного ключа***. Для подолання такого значного недоліку створено так звану ***інфраструктуру публічних ключів***, яка ґрунтується на системі сертифікатів, де вказано, кому належить той чи інший публічний ключ. Про структуру та принципи організації такої системи піде мова далі.

Стосовно останнього недоліку зауважимо, що всі асиметричні криптосистеми, як уже згадувалося, ґрунтуються на односторонніх функціях. Отже, генерування пари криптографічних ключів відбувається легко, а при спробі криптоаналітика обчислити приватний ключ, знаючи публічний, він наштовхується на математичну проблему надзвичайної обчислювальної складності. Цікаво, що таке твердження немає жодного математично строгого доведення, воно ґрунтується лише на досвіді практичного використання односторонніх функцій. Не виключено, що з часом, при подальшому розвитку математичної науки, знайдеться спосіб легкого знаходження приватних ключів за парними до них публічними, що може спричинити повний крах асиметричної криптографії [10].

Асиметрична криптографія, звичайно, має й істотні *переваги*. До них можна віднести [14, 15]:

- відсутність проблеми розповсюдження криптографічних ключів;
- непотрібність великої кількості ключів для створення багатокористувацької системи обміну зашифрованою інформацією;
- можливість створення зовсім нових криптографічних процедур, зокрема ***електронного цифрового підпису***, які принципово неможливі з використанням симетричної криптографії.

## Комбіновані криптосистеми

Підведемо підсумки двох останніх розділів. З одного боку, симетричні криптосистеми, завдяки високій швидкодії та надійності, якнайкраще підходять для захисту інформації в комп'ютерних системах. Вони можуть застосовуватися як для шифрування мережного трафіку, так і локальної інформації. Однак симетричним криптосистемам притаманний суттєвий недолік, проблема розповсюдження ключів.

З іншого боку, цього недоліку позбавлені асиметричні криптосистеми. Вони використовують для шифрування публічний ключ, який не потрібно розповсюджувати. Однак швидкодія цих систем, не в останню чергу внаслідок використання дуже довгих ключів, занадто мала (приблизно у 1000 разів менша порівняно із симетричними), щоб їх можна було застосовувати для шифрування

потоків інформації. Асиметричну криптосистему було б доцільно використати для обміну короткою інформацією, коли швидкодія системи не відіграє визначальної ролі, наприклад криптографічних ключів.

Таким чином, ми приходимо до ідеї комбінованого використання криптосистем: безпосереднє шифрування інформації виконують симетричною криптосистемою, а для розповсюдження криптографічних ключів використовують асиметричну.

Сеанс зв'язку при цьому виглядає приблизно так. Після встановлення сеансу передавальна сторона вибирає публічний ключ асиметричної криптосистеми приймальної сторони й зашифровує ним сеансовий ключ симетричного алгоритму. Зашифрований сеансовий ключ відправляється відкритим каналом зв'язку на приймальну сторону, де розшифровується приватним ключем. Тепер можна починати шифрування каналу зв'язку за допомогою симетричної криптосистеми, оскільки ключ шифрування вже відомий обом учасникам інформаційного обміну. Схематично роботу такої системи можна зобразити так:

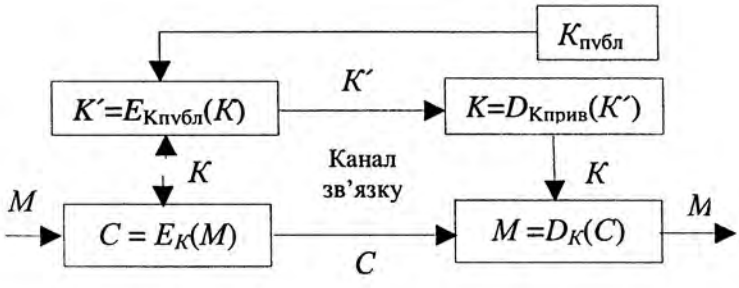



*Рис. 2.3. Схема роботи комбінованої криптосистеми*

Перевагами такої криптосистеми будуть як найвища швидкодія симетричних алгоритмів, так і зручність використання асиметричних криптосистем. Необхідно сказати, що комбіновані криптосистеми внаслідок своєї універсальності сьогодні дуже популярні. Наприклад, вони використовуються в захищених протоколах SET, SSL, TLS та інших.

Незалежно від того, який алгоритм використовує криптосистема, симетричний чи асиметричний, вона повинна задовольняти такі вимоги:

- зашифрований текст можна прочитати лише, знаючи ключ розшифрування;
- трудомісткість обчислення криптографічного ключа шифрування за фрагментом криптограми та відповідним йому фрагментом відкритого тексту повинна бути занадто великою для її практичної реалізації;
- кількість операцій, необхідних для розшифрування інформації підбиранням ключа, повинна мати чітку нижню оцінку та виходити за межі можливостей сучасних комп'ютерних систем, а також мати достатній запас з урахуванням прогресу обчислювальної техніки;

- знання зловмисником алгоритму шифрування не повинно впливати на надійність системи захисту;
- незначна зміна ключа або відкритого тексту повинна приводити до суттєвої зміни зашифрованого тексту;
- в процесі шифрування необхідно забезпечувати постійний контроль за ключем шифрування та даними, що зашифровуються;
- довжина зашифрованого тексту не повинна бути набагато більшою, ніж довжина відкритого тексту;
- не повинно бути простих або легко встановлюваних залежностей між ключами, що послідовно використовуються в процесі шифрування;
- будь-який ключ із множини можливих повинен забезпечувати однаково надійний захист інформації;
- додаткові біти, що додаються до криптограми в процесі зашифрування, мають бути повністю та надійно приховані в зашифрованому тексті;
- криптосистема повинна забезпечувати гарантовану швидкість шифрування за довільних параметрів відкритого тексту та ключів шифрування.

Криптосистеми можуть реалізовуватися як програмно, так і апаратно. Апаратна реалізація, без сумніву, має значно більшу вартість, однак і значні переваги, зокрема високу продуктивність, простоту використання, захищеність тощо. Програмна реалізація практичніша, гнучкіша у використанні, простіше модифікується.

# Розділ III
# Симетричні криптосистеми

Під симетричними криптосистемами розуміються такі криптосистеми, в яких для шифрування та розшифровки використовується той самий ключ.

Для користувачів це означає, що перед початком використання системи необхідно отримати загальний секретний ключ так, щоб унеможливити доступ до нього потенційного злочинця.

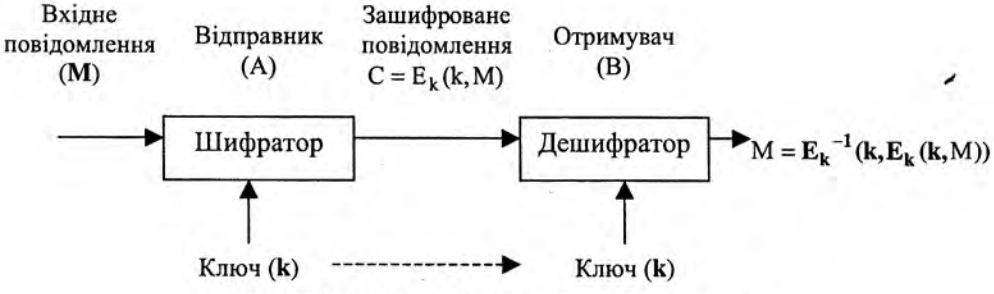



*Рис. 3.1. Схема симетричної криптосистеми*

**Блокові шифри**

Є сімейством обернених перетворень, блоків (частин фіксованої довжини) вихідного тексту. Фактично блоковий шифр – це система підстановки на алфавіті блоків (вона може бути моно- чи багатоалфавітною в залежності від режиму блокового шифру). В наш час блокові шифри найпоширеніші на практиці.

Російський ГОСТ 28147-89 та американський DES належать саме до цього класу шифрів.

**Загальні відомості про блокові шифри**

Під *N-розрядним блоком* будемо розуміти послідовність із нулів і одиниць довжиною $N^3$ [10]:

$$x = (x_0, x_1, ..., x_{N-1}) \in Z_{2,N};$$

$x$ у $Z_{2,N}$ можна інтерпретувати як вектор і як двійкове представлення цілого числа

$$\|x\| = \sum_{i=0}^{N-1} x_i 2^{N-i-1}.$$

Наприклад, якщо N = 4, то

| | | | |
|---|---|---|---|
| $(0,0,0,0)\to 0$ | $(0,0,0,1)\to 1$ | $(0,0,1,0)\to 2$ | $(0,0,1,1)\to 13$ |
| $(0,1,0,0)\to 4$ | $(0,1,0,1)\to 5$ | $(0,1,1,0)\to 6$ | $(0,1,1,1)\to 7$ |
| $(1,0,0,0)\to 8$ | $(1,0,0,1)\to 9$ | $(1,0,1,0)\to 10$ | $(1,0,1,1)\to 11$ . |
| $(1,1,0,0)\to 12$ | $(1,1,0,1)\to 13$ | $(1,1,1,0)\to 14$ | $(1,1,1,1)\to 15$ |

Блоковим шифром будемо називати елемент

$$\pi \in SYM(Z_{2,N}) : \pi : x \to y = \pi(x),$$

де $x = (x_0, x_1, ..., x_{N-1})$, $y = (y_0, y_1, ..., y_{N-1})$. Незважаючи на те, що блокові шифри є частковими випадками підстановок (тільки на алфавітах дуже великої потужності), їх потрібно розглядати особливим чином, оскільки, по-перше, більшість симетричних шифрів, що використовуються в системах передачі інформації, є блоковими і, по-друге, блокові шифри найзручніше описувати в алгоритмічному вигляді, а не як звичайні підстановки.

Нехай

$$\pi(x_i) = y_i, 0 \le i < m,$$

для деякого $\pi \in SYM(Z_{2,N})$, вихідного тексту $X = \{x_i : x_i \in Z_{2,N}\}$ і шифрованого тексту $Y = \{y_i\}$. Що можна сказати про $\pi(x)$, якщо $x \notin \{x_i\}$? Оскільки $\pi$ є перестановкою на $Z_{2,N}$, то $\{y_i\}$ різні і $\pi(x) \notin \{y_i\}$ при $x \notin \{x_i\}$.

$(2^N - m)!$ з $(2^N)!$ перестановок у $SYM(Z_{2,N})$ задовольняє рівність

$$\pi(x_i) = y_i,\ 0 \le i < m.$$

Подальша специфікація $\pi(x)$ за відсутності додаткової інформації неможлива. Це визначається в основному тією обставиною, що р є елементом, що належить $SYM(Z_{2,N})$. Якщо відомо, що р належить невеликій підмножині $\Pi$ з $SYM(Z_{2,N})$, то можна зробити висновок. Наприклад, якщо

$$\Pi = \{\pi_j : 0 \le j < 2^N\}, \pi_j(i) = (i + j) (\mathrm{mod}\, 2^N), 0 \le i < 2,$$

то значення $\pi(x)$ при заданому значенні $x$ однозначно визначає $p$. У цьому випадку $X$ є підмножиною підстановок Цезаря на $Z_{2,N}$.

Криптографічне значення цієї властивості повинно бути очевидним: якщо вихідний текст шифрується підстановкою $p$, вибраною з повної симетричної групи, то зловмисник, який вивчає відповідність між підмножинами вихідного і шифрованого текстів

$$x_i \leftrightarrow y_i, 0 \le i < m,$$

не в змозі на основі цієї інформації визначити вихідний текст, що відповідає $y \notin \{y_i\}$.

Якщо для шифрування вихідного тесту використовується підсистема $p$ із $П \in SYM(Z_{2,N})$, то отриману в результаті систему підстановок $П$ будемо називати системою блокових шифрів чи системою блокових підстановок. Блоковий шифр являє собою випадок моноалфавітної підстановки з алфавітом $Z_{2^N} = Z_{2,N}$. Якщо інформація вихідного тексту не може бути представлена N-розрядними блоками, як у випадку стандартного алфавітно-цифрового тексту, то перше, що потрібно зробити, це перекодувати вихідний текст саме в цей формат. Перекодування можна здійснити декількома способами і з практичного погляду неважливо, який зі способів був обраний.

В установках обробки інформації блокові шифри будуть використовуватися багатьма користувачами. Ключовою системою блокових шифрів є підмножина П[К] симетричної групи

$$П[K] = \{\pi\{k\} : k \in K\},$$

що індексується параметром $k \in K$; $k$ – ключ, а К – простір ключів. При цьому непотрібно, щоб різні ключі відповідали різним підстановкам $Z_{2,N}$.

Ключова система блокових шифрів П[К] використовується так. Користувач $i$ та користувач $j$ у певний спосіб укладають угоду відносно ключа $k$ із $K$, вибираючи елемент із П[К] і передаючи текст, зашифрований із використанням вибраної підстановки. Запис

$$y = \pi\{k, x\}$$

будемо використовувати для позначення $N$-розрядного блока шифрованого тексту, який отримується в результаті шифрування $N$-розрядного блока вихідного тексту $x$ із використанням підстановки $\pi\{k\}$, що відповідає ключу $k$. Припустимо, що зловмиснику:

- відомий простір ключів $K$;
- відомий алгоритм визначення підстановки $\pi\{k\}$ за значенням ключа $k$;
- невідомо, який саме ключ $k$ обрав користувач.

На основі цих даних зловмисник може:

- отримати ключ унаслідок недбалості користувача $i$ чи користувача $j$;
- перехопити (шляхом перехоплення телефонних і комп'ютерних повідомлень) шифрований текст $y$, який передається користувачу $j$ користувачем $i$, і використовувати всі можливі ключі з $K$ до отримання повідомлення вихідного тексту;
- отримати відповідний вихідний і шифрований тексти ($x \to y$) і скористатися методом проби на ключ;
- отримати відповідний вихідний і шифрований тексти та дослідити співвідношення вихідного тексту $x$ і шифрованого тексту $y$ для визначення ключа $k$;

- організувати каталог $N$-розрядних блоків із записами частот їх появи у вихідному чи шифрованому тексті. Каталог дає можливість проводити пошук найбільш імовірних слів, використовуючи, наприклад, таку інформацію:
  - лістінг на мові асемблера характеризується сильно вираженим структурованим форматом;
  - цифрове представлення графічної і звукової інформації має обмежений набір символів.

Припустимо, що $N = 64$ і кожен елемент $SYM(Z_{2,N})$ може бути використаний як підстановка, так що $K = SYM(Z_{2,N})$. Тоді існує $2^{64}$ 64-розрядних блоки; зловмисник не може підтримувати каталог із $2^{64} \approx 1{,}8 \cdot 10^{19}$ рядками; проба на ключ за кількості ключів, що дорівнює $(2^{64})!$, практично неможлива; відповідність вихідного і шифрованого текстів деяких $N$-розрядних блоків $\pi\{k, x_i\} = y_i$, $0 \le i < m$, не дає зловмиснику інформації відносно значення $\pi\{k, x\}$ для $x \in \{x_i\}$.

Системи шифрування з блоковими шифрами, алфавітом $Z_{2,64}$ і простором ключів $K = SYM(Z_{2,64})$ неподільні в тому розумінні, що підтримання каталогу частот появи букв для 64-розрядних блоків чи проба на ключ при кількості ключів $2^{64}$ виходять за межі можливостей зловмисника.

Отже, вимоги для якісного блокового шифру формуються так:

1. Необхідно досить велике $N$ (64 чи більше) для того, щоб завадити створенню й підтримці каталогу.
2. Необхідний досить великий простір ключів, аби уникнути можливості підбору ключа.
3. Необхідні складні співвідношення $\pi\{k, x\} : x \to y = \pi\{k, x\}$ між вихідним і шифрованим текстами для того, щоб аналітичні і статистичні методи визначення вихідного тексту і ключа на основі відповідності вихідного і шифрованого текстів було б неможливо (принаймні важко) реалізувати.

Одним із найпоширеніших способів задання блокових шифрів є використання так званих мереж Фейстеля. Мережа Фейстеля являє собою загальний метод перетворення довільної функції (її зазвичай називають $F$-функцією) в перестановку на множині блоків. Ця конструкція винайдена Хорстом Фейстелем і була використана у великій кількості шифрів, включаючи DES і ГОСТ 28147-89. $F$-функція являє собою основний складовий блок мережі Фейстеля і завжди вибирається нелінійною та практично в усіх випадках незворотною.

Формально $F$-функцію можна подати у вигляді відображення

$$\mathrm{F} : Z_{2,N/2} \times Z_{2,k} \to Z_{2,N/2},$$

де $N$ – довжина перетворюваного блока тексту (повинна бути парною), $k$ – довжина використовуваного блока ключової інформації.

Нехай $M$ – блок тексту, зобразимо його у вигляді двох підблоків однакової довжини $M = \{A, B\}$. Тоді один цикл (ітерацію) мережі Фейстеля визначають так:

$$M_{i+1} = B_i \| (F(B_i, k_i) \oplus A_i),$$

де $M_i = \{A_i, B_i\}$, $\|$ – операція конкатенації, а $\oplus$ – побітне виключаюче АБО.

Структура циклу мережі Фейстеля показана на рис. 3.2.

Мережа Фейстеля складається з певної фіксованої кількості циклів, яку визначають із міркувань стійкості шифру, що його розробляють. У цьому разі в останньому циклі переставляння місцями половин блоків даних не виконують, бо це не впливає на стійкість шифру. Така структура шифрів має низку переваг, а саме:

- процедури шифрування й розшифрування збігаються, лише ключову інформацію під час розшифрування використовують у зворотному порядку;
- для розшифрування можна використовувати ті ж апаратні або програмні блоки, що й для шифрування.

Недоліком мережі Фейстеля є те, що в кожному циклі змінюється лише половина блока тексту, який опрацьовують. Це призводить до збільшення кількості циклів для досягнення бажаної стійкості.

Стосовно вибору $F$-функції чітких рекомендацій немає, проте найчастіше ця функція, яка повністю залежить від ключа, складається з нелінійних замін, перестановок і зсувів.

Інший підхід до побудови блокових шифрів – використання зворотних, залежних від ключа перетворень. У цьому випадку на кожній ітерації змінюється весь блок і, відповідно, загальна кількість циклів може бути зменшена. Кожен цикл є послідовністю перетворень (так званих шарів), кожне з яких виконує свою функцію. Звичайно використовують шар нелінійної оберненої заміни, шар лінійного перемішування й один або два шари підмішування ключа. Недоліком цього підходу є те, що для процедур шифрування й розшифрування в загальному

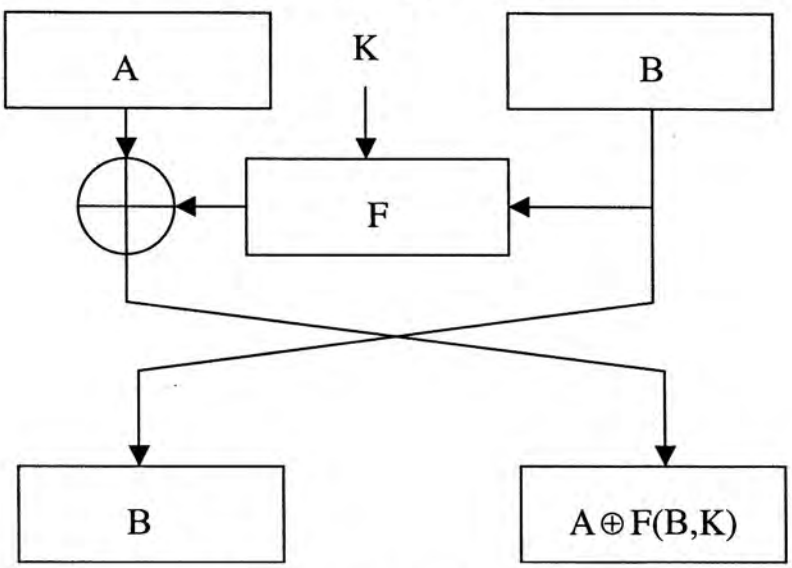



*Рис. 3.2. Структура циклу мережі Фейстеля*

випадку не можна використовувати одні й ті ж блоки, а це збільшує апаратні або програмні витрати на реалізацію.

## *Особливості стандарту DES*

Найпопулярніша сучасна схема шифрування базується на стандарті DES (Data Encryption Standard – стандарт шифрування даних), прийнятому в 1977 році Національним бюро стандартів (NBS) США (сьогодні називається Національним інститутом стандартів і технологій NIST). Цей стандарт отримав офіційне ім'я Federal Information Processing Standard 46. Відповідно до цього стандарту дані шифруються 64-бітовими блоками з використанням 56-бітного ключа. Багатошаровий алгоритм перетворює 64-бітові блоки тексту, які надходять на вхід у 64-бітні блоки шифрованого тексту. Цей же алгоритм із тим же ключем служить для зворотного перетворення шифрованого тексту у відкритий.

Стандарт DES завоював широку популярність, неодноразово ставши при цьому об'єктом полеміки на тему його безпеки. Щоб зрозуміти суть цієї полеміки, давайте коротко розглянемо історію створення і становлення DES.

У кінці 60-х IBM почала науково-дослідний проект у галузі комп'ютерної криптографії, який очолив Хорст Фейстель. У результаті роботи над проектом до 1971 року був створений алгоритм під кодовою назвою LUCIFER, який продали банку Ллойда (Lloyd's of London) для використання в системі управління оборотом готівкових коштів, теж розробленій IBM. Шифр LUCIFER являв собою блоковий шифр Фейстела, що оперував блоками розміром 64 біти, використовуючи ключ довжиною 128 бітів. Базуючись на багатообіцяючих результах проекту LUCIFER, IBM взяла курс на створення комерційного варіанта шифру, який, в ідеалі, можна було б розмістити в одній мікросхемі. Цей напрямок очолили Уолтер Тачман і Карл Мейер, залучивши не тільки спеціалістів з IBM, але й консультантів і технічних спеціалістів. У результаті їхніх зусиль була створена вдосконалена версія шифру LUCIFER, що мала більшу криптостійкість, але зменшений до 56 бітів ключ, щоб алгоритм міг вміститися в одну мікросхему.

У 1973 році Національне бюро стандартів оголосило конкурс на найкращий проект по створенню загальнодержавного стандарту шифрування. Компанія IBM подала на конкурс результати проекту Тачмана-Мейера. Їхній алгоритм виявився безумовно найкращим з усіх запропонованих, і в 1977 році він був затверджений як стандарт шифрування даних (DES) [10, 14].

Але перш ніж стати офіційним стандартом, запропонований IBM алгоритм зазнав жорстокої критики, яка не вщухає до сьогодні. Нападок зазнають, в основному, дві особливості шифру. По-перше, в початковому алгоритмі LUCIFER фірми IBM використовувалися ключі довжиною 128 бітів, а в запропонованому стандарті довжина ключа зменшена до 56 бітів. Критики побоювалися (і побоюються досі), що такий розмір ключа занадто малий для того, щоб шифр міг гарантовано протистояти спробам криптоаналізу з простим перебором всіх можливих варіантів (сьогодні, з бурхливим розвитком комп'ютерної техніки, реалізація такої атаки стала фактом). Другий серйозний випад критиків

спрямований проти факту засекреченості внутрішньої структури DES, а саме структури S-матриць. Тому користувачі не могли бути впевнені в тому, що у внутрішній структурі DES немає якихось дефектів, за допомогою яких спеціалісти АНБ могли б розшифрувати повідомлення, не маючи ключа. Подальші дослідження, зокрема останні праці з різницевого криптоаналізу, дозволяють із більшою впевненістю стверджувати, що DES має досить надійну внутрішню структуру. Більше того, за ствердженнями учасників проекту з боку IBM, єдиними змінами, внесеними в представлений ними проект стандарту, були запропоновані АНБ зміни в структурі S-матриць із метою укріплення ймовірних слабких місць алгоритму, знайдених у ході оцінки проекту.

Так чи інакше, DES побачив світ і став дуже популярним, особливо у фінансових застосуваннях. В 1994 році NIST продовжив використання DES як федерального стандарту ще на п'ять років і рекомендував його до застосування в комерційних структурах.

**Шифрування DES**

*Опис DES*

DES являє собою блоковий шифр, він шифрує дані 64-бітними блоками. На вхід алгоритму подається 64-бітний блок відкритого тексту, а на виході отримується 64 біти шифрограми. DES є симетричним алгоритмом: для шифрування і розшифрування використовується однаковий алгоритм і ключ.

Довжина ключа дорівнює 56 бітам. Ключ зазвичай представляється 64-бітним числом, але кожний восьмий біт використовується для перевірки парності й ігнорується. Біти парності є найменшими значущими бітами байтів ключа. Деякі числа вважаються слабкими ключами, але їх можна легко уникнути. Безпека системи повністю визначається ключем.

На найпростішому рівні алгоритм не являє собою нічого, крім комбінації двох основних методів шифрування: зміщення й дифузії. Фундаментальним блоком DES є застосування до тексту одиничної комбінації цих методів (підстановка, а за нею – перестановка), які залежать від ключа. Такий блок називається етапом. DES складається з 16 етапів, однакова комбінація методів застосовується до відкритого тексту 16 разів [10].

DES працює з 64-бітним блоком відкритого тексту. Після початкової перестановки блок розбивається на праву й ліву половини довжиною 32 біти. Після цього виконується 16 етапів однакових дій, що називаються цикловою функцією, в яких дані об'єднуються з ключем. Після шістнадцятого етапу права і ліва половини об'єднуються й алгоритм завершується кінцевою перестановкою (оберненою по відношенню до початкової).

На кожному етапі біти ключа зсуваються, а потім із 56 бітів вибираються 48. Права половина даних збільшується до 48 бітів за допомогою перестановки з розширенням, об'єднується з допомогою XOR із 48 бітами зміщеного й переставленого ключа, проходить через 8 S-блоків, утворюючи 32 нових біти, і переставляється знову. Ці чотири операції і виконуються функцією *F*. Потім

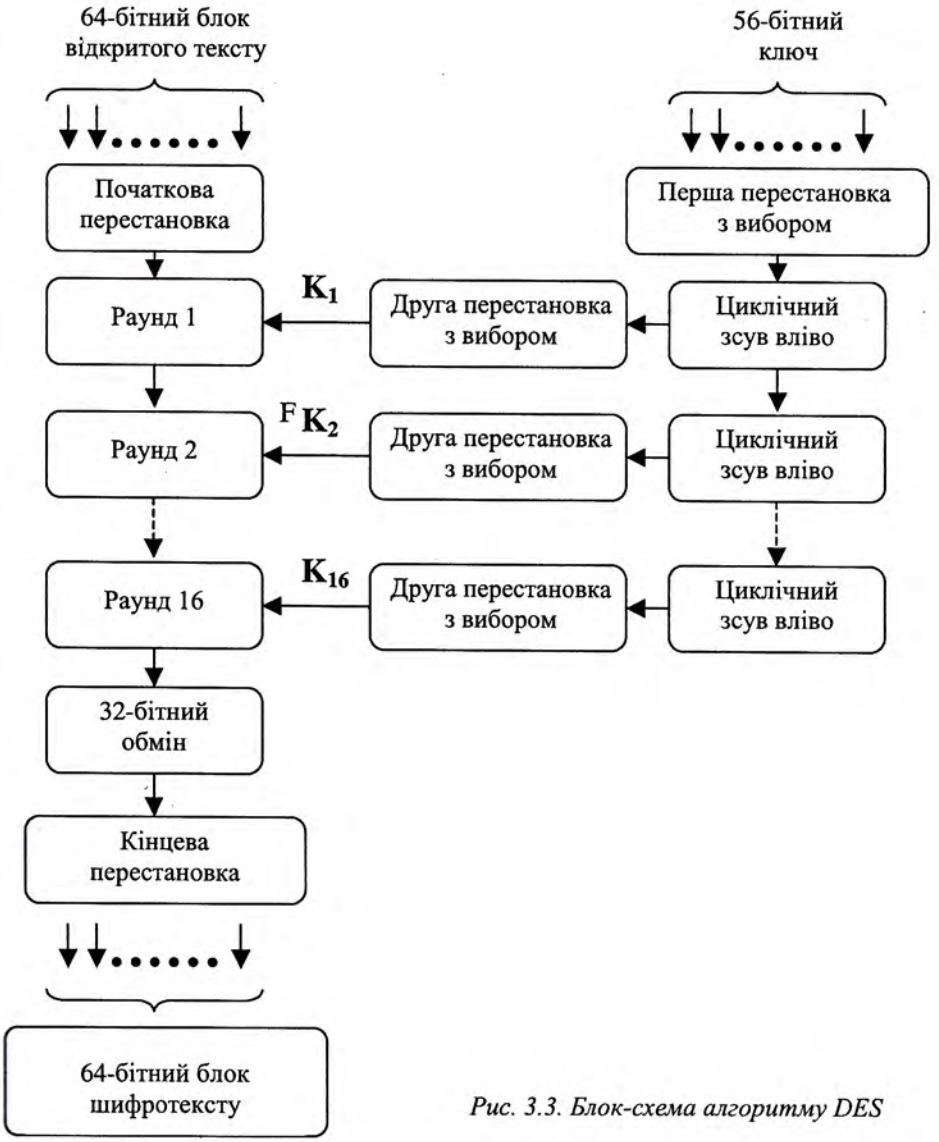



*Рис. 3.3. Блок-схема алгоритму DES*

результат функції $F$ об'єднується з лівою половиною з допомогою іншого XOR. У результаті цих дій з'являється нова права половина, а стара права половина стає новою лівою. Ці дії повторюються 16 разів, утворюючи 16 етапів DES [10].

Якщо $B_i$ – це результат $i$-тої ітерації, $L_i$ і $R_i$ – ліва і права половини $B_i$, $K_i = C_i + D_i$ – 48-бітний ключ для етапу $i$, а $F$ – це функція, яка виконує усі підстановки, перестановки і XOR із ключем, то етап можна зобразити як

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

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i).$$

Тепер розглянемо порядок обробки текстового блока алгоритмом DES.

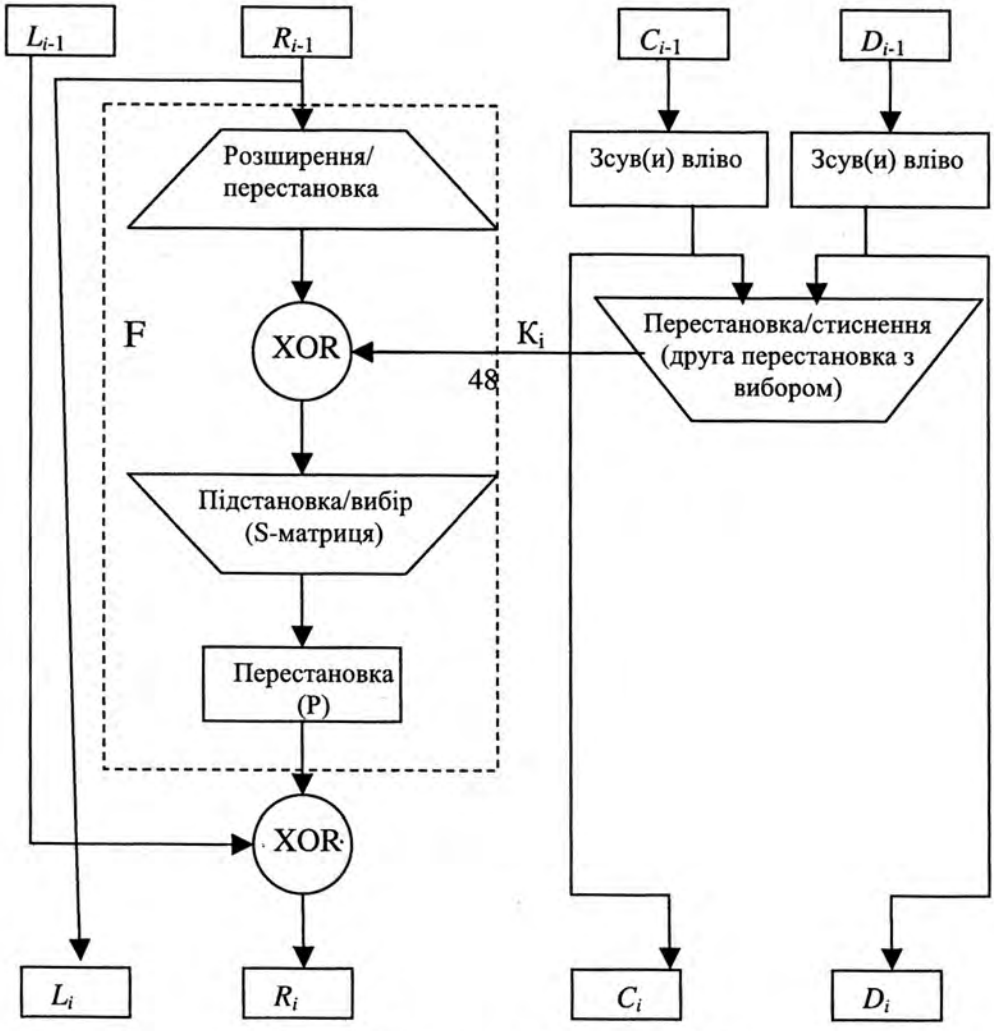



*Рис. 3.4. Блок-схема одного етапу DES*

**Початкова перестановка**

Початкова перестановка виконується ще до першого етапу шифрування. Початкова перестановка і відповідна їй кінцева перестановка не впливають на безпеку DES. Оскільки програмна реалізація цієї багатобітної перестановки нелегка, в багатьох програмних реалізаціях DES початкова й кінцева перестановки не використовуються (за таких умов цей алгоритм перестає бути алгоритмом DES).

Таблиця 3.1

**Початкова перестановка**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Таблицю слід читати так: перший біт переставляється на 58-ме місце, другий – на 50-те, третій – на 42-ге, ..., 64-й – на 7-ме.

**Перестановка з розширенням**

Ця операція розширює праву половину даних від 32 до 48 бітів. Оскільки при цьому не просто повторюються певні біти, а й змінюється їхній порядок, ця операція називається перестановкою з розширенням. У неї дві задачі – привести розмір правої половини у відповідність із ключем для операції XOR і отримати довший результат, який можна буде стиснути в ході операції підстановки. Проте основний зміст зовсім інший. За рахунок впливу одного біта на дві підстановки швидше зростає залежність бітів результату від бітів вихідних даних. Це називається лавинним ефектом. DES спроектовано так, щоб якомога скоріше можна було досягти залежності кожного біта шифротексту від кожного біта відкритого тексту й кожного біта ключа.

Перестановку з розширення ще іноді називають Е-блоком. Для кожного 4-бітного вхідного блока перший і четвертий біти являють собою два біти вихідного блока, а другий і третій біти – один біт вхідного блока.

Таблиця 3.2

**Перестановка з розширенням**

| | | | | | |
|---|---|---|---|---|---|
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

**Підстановка з допомогою S-блоків**

Після об'єднання стисненого блока з розширеним блоком із допомогою XOR над 48-бітним результатом виконується операція підстановки. Підстановки

проводяться у восьми блоках підстановки, чи S-блоках. У кожного S-блока 6-бітний вхід і 4-бітний вихід, всього використовується вісім різних S-блоків. 48 бітів діляться на вісім 6-бітних підблоки. Кожний окремий підблок обробляється окремим S-блоком: перший підблок – S-блоком 1, другий підблок – S-блоком 2 і т.д. [4, 10].

Кожен S-блок являє собою таблицю з 2 рядків і 16 стовпців. Кожний елемент в блоці є 4-бітним числом. За 6 вхідними бітами S-блока визначається, під якими номерами стовпців і рядків шукати вихідні значення.

Таблиця 3.3

**S-блоки алгоритму DES**

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
| $S_2$ | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
| $S_3$ | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
| $S_4$ | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
| $S_5$ | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 1 |
| | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
| $S_6$ | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
| $S_7$ | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |
| $S_8$ | 13 | 5 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

Вхідні біти в особливий спосіб визначають елемент S-блока. Наприклад, розглянемо 6-бітний вхід S-блока: $b_1$, $b_2$, $b_3$, $b_4$, $b_5$ і $b_6$. Біти $b_1$ і $b_6$ об'єднуються, утворюючи 2-бітне число від 0 до 3, яке відповідає рядку таблиці. Середні 4 біти, з $b_2$ по $b_5$, об'єднуються, утворюючи 4-бітне число від 0 до 15, яке відповідає стовпцю таблиці.

Наприклад, на вхід шостого S-блока (тобто біти функції XOR з 31 по 36) потрапляє 110011. Перший і останній біти, об'єднуючись, утворюють 11, що відповідає рядку 3 шостого S-блока. Середні 4 біти утворюють 1001, а це стовпець 9 того ж S-блока. Елемент S-блока 6, який знаходиться на перетині рядка 3 і стовпця 9, – 14. Замість 110011 підставляється 1110.

Підстановка з допомогою S-блоків є ключовим етапом DES. Інші дії алгоритму лінійні й легко піддаються криптоаналізу. S-блоки нелінійні, й саме на них здебільшого ґрунтується безпека DES.

У результаті цього етапу підстановки отримуються вісім 4-бітних блоки, які знову об'єднуються в єдиний 32-бітний блок. Цей блок надходить на вхід наступного етапу – підстановки з допомогою P-блоків.

*Підстановка з допомогою P-блоків*

32-бітний вихід підстановки з допомогою S-блоків перемішується відповідно до P-блока. Ця перестановка переміщує кожний вхідний біт в іншу позицію, жоден біт не використовується двічі і жоден не ігнорується. Цей процес називається прямою перестановкою.

Таблиця 3.4

**Перестановка з допомогою P-блоків**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 |
| 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 |
| 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

Нарешті результат перестановки з допомогою P-блока об'єднується з допомогою XOR з лівою половиною початкового 64-бітного блока. Потім ліва і права половини міняються місцями, і починається наступний етап.

*Кінцева перестановка*

Кінцева перестановка обернена по відношенню до початкової. Слід звернути увагу, що ліва і права половини не міняються місцями після останнього етапу DES, замість цього об'єднаний блок $R_{16}L_{16}$ використовується як вхід кінцевої перестановки. Це робиться для того, щоб алгоритм можна було використовувати як для шифрування, так і для дешифрування. Якби ж використовувалася перестановка половинок із наступним циклічним зсувом, результат був би таким самим.

Таблиця 3.5

**Кінцева перестановка**

Тепер розглянемо детальніше алгоритм розгортання кляча в DES.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 39 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

**Перетворення ключа**

Спочатку 64-бітний ключ DES зменшується до 56-бітного ключа відкиданням бітів. 56 бітів ключа переставляються відповідно до табл. 3.6. Після перестановки 56-бітного ключа для кожного з 16 етапів DES генерується новий 48-бітний підключ. Ці підключі визначаються так. По-перше, 56-бітний ключ ділиться на дві 28-бітні половинки. Потім половинки циклічно зсуваються вліво на один чи два біти, залежно від етапу.

Таблиця 3.6

**Перестановка ключа**

| | | | | | | |
|---|---|---|---|---|---|---|
| 57 | 49 | 41 | 33 | 25 | 17 | 9 |
| 1 | 58 | 50 | 42 | 34 | 26 | 18 |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 |
| 19 | 11 | 3 | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 |
| 7 | 62 | 54 | 46 | 38 | 30 | 22 |
| 14 | 6 | 61 | 53 | 45 | 37 | 29 |
| 21 | 13 | 5 | 28 | 20 | 12 | 4 |

Таблиця читається аналогічно до табл. 3.1.

Таблиця 3.7

**Кількість бітів зсуву ключа залежно від етапу**

| Етап | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Число | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Після зсуву вибирається 48 з 56 бітів. Оскільки при цьому не тільки вибирається підмножина бітів, а й змінюється їх порядок, ця операція називається перестановкою зі стисненням (її ще називають перестановкою з вибором). Її результатом є набір із 48 бітів. Наприклад, біт зсунутого ключа в позиції 33 переміщається в позицію 35 результату, а 18-й біт зсунутого ключа відкидається.

Таблиця 3.8

**Перестановка зі стисненням**

| | | | | | |
|---|---|---|---|---|---|
| 14 | 17 | 11 | 24 | 1 | 5 |
| 3 | 28 | 15 | 6 | 21 | 10 |
| 23 | 19 | 11 | 4 | 26 | 8 |
| 16 | 7 | 27 | 20 | 13 | 2 |
| 41 | 52 | 31 | 37 | 47 | 55 |
| 30 | 40 | 51 | 45 | 33 | 48 |
| 44 | 49 | 39 | 56 | 34 | 53 |
| 46 | 42 | 50 | 36 | 29 | 32 |

Через зсув для кожного підключа використовується різна підмножина бітів ключа. Кожен біт використовується приблизно в 14 з 16 підключів, хоча не всі біти використовуються однакову кількість разів.

*Операція розшифрування DES*

Після всіх підстановок, перестановок, операцій XOR і циклічних зсувів можна подумати, що алгоритм дешифрування суттєво відрізняється від алгоритму шифрування і так само заплутаний. Навпаки, різні компоненти були підібрані так, щоб виконувалося дуже корисна властивість: для шифрування й розшифрування використовувався один і той самий алгоритм.

DES дозволяє використовувати для шифрування і розшифрування блока одну функцію. Єдиною відмінністю є те, що ключі повинні використовуватися в зворотному порядку. Тобто, якщо на етапах шифрування використовувалися ключі $K_1, K_2, K_3, ..., K_{16}$, то ключами дешифрування будуть $K_{16}, K_{15}, K_{14}, ..., K_1$. Алгоритм, який створює ключ для кожного етапу, також циклічний. Ключ зсувається вправо, а кількість позицій зсуву становить 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

**Стійкість DES**

Обговорюючи DES, неможливо обминути тему безпеки цього алгоритму та можливих атак на нього. Багаторічний досвід експлуатації DES і його відкритість зумовили те, що DES став одним із найпопулярніших алгоритмів із погляду перевірки тих чи інших методів криптоаналізу. За весь час існування алгоритму на нього було здійснено багато атак; при цьому уважно вивчалися та враховувалися його слабкості, виявлені за довгий термін експлуатації. Треба мати на увазі, що деякі атаки можна реалізувати, лише виходячи з припущення, що зловмисник володіє певними обчислювальними (або часовими) ресурсами. У більшості випадків такі спроби мають лише теоретичний характер, однак не виключено, що з розвитком комп'ютерної техніки та криптології як науки ці атаки можна буде реалізувати на практиці.

До основних недоліків DES відносять такі [14]:

- наявність «слабких» ключів, викликана тим, що для генерування ключової послідовності виконується два незалежних регістри зсуву. Прикладом слабкого ключа може служити *1F1F1F1F0E0E0E0E*. При цьому результатом генерування будуть ключові послідовності, однакові з вихідним ключем, в усіх 16 раундах. Існують також різновиди слабких ключів, що дають усього чотири ключові послідовності. Існують також «зв'язані» ключі, які отримуються один з одного інверсією одного біта;
- невелика довжина ключа в 56 бітів. При сучасному рівні розвитку комп'ютерних засобів ця довжина ключа не може забезпечувати потрібного захисту для деяких типів інформації;
- надмірність ключа, що має біти контролю парності. Біхам і Шамір запропонували досить ефективну атаку на реалізацію DES у смарт-картах або банківських криптографічних модулях, що використовують EEPROM-пам'ять для зберігання ключів. Наявність бітів контролю парності дозволяє відновити ключ при втраті частини ключа, викликаній втратою інформації в комірках пам'яті;
- використання статичних підстановок у S-блоках, що, незважаючи на велику кількість раундів, дозволяє криптоаналітикам атакувати цей алгоритм.

Зауважимо, що авторам не відомі успішні атаки на 16-раундовий DES, які використовують останній факт, однак успішні атаки на 8-раундовий DES трапляються. Так, М.Хеллман запропонував атаку на основі робочої станції SUN-4, яка визначає 10 бітів ключа за 10 с. У випадку вибору 512 відкритих текстів імовірність успіху становить 80%, а при виборі 768 відкритих тактів – 95%. Відновивши 10 бітів ключа, решту можна знайти методом повного перебору наступних 46 бітів.

Отже, враховуючи вищезазначене, можна стверджувати, що сьогодні використання DES для критичної, з погляду секретності, інформації є досить небезпечною справою.

Криптологи давно намагалися модифікувати DES із метою збільшення його криптостійкості. Розглянемо такі модифікації, як 3DES, DESX і S-DES.

**Потрійний DES**

Найвідомішою модифікацією DES є потрійний DES (3DES), один з варіантів якого визначається формулою [2]:

$$C = 3DES_{K_1K_2K_3}(M) = DES_{K_3}(DES^{-1}_{K_2}(DES_{K_1}(M))).$$

Тут використовуються три ключі $K_1$, $K_2$, $K_3$, сумарна довжина яких складає *56* x *3=186* бітів. 64-бітний блок відкритого повідомлення $M$ спочатку шифрується на ключі $K_1$, потім розшифровується на ключі $K_2$ і знову зашифровується на $K_3$.

Причиною того, що на на другому кроці використовується $DES^{-1}_{K_2}$, а не $DES_{K_2}$, є сумісністю з DES. Дійсно, якщо $K = K_1 = K_2 = K_3$, то $3DES_K = DES_K$.

Причиною того, чому обрано саме три ітерації, а не дві, є існування атаки «зустріч посередині» на подвійний DES.

Розшифрування інформації в 3DES використовується обернено:

$$M = DES_{K_3}^{-1}(DES_{K_2}(DES_{K_1}^{-1}(C))).$$

Отже, кожен блок повідомлення $M$ обробляється DES-машиною тричі на різних ключах, що, звичайно, поліпшує криптостійкість. Однак проблемою 3DES є його повільність – швидкість його втричі менша за DES. У багатьох випадках це неприпустимо, особливо для шифрування каналів зв'язку.

**DESX**

У 1984 р. Рон Рівест запропонував інший варіант модифікації DES, позбавлений цього недоліку 3DES.

DESX (DES extended – розширений DES) можна визначити так:

$$C = DESX_{KK_1K_2} = K_2 \oplus DES_K(K_1 \oplus M).$$

Отже, 64-бітний блок повідомлення $M$ підсумовується за mod 2 з першим «зашумлюючим» ключем $K_1$, потім обробляється DES-машиною з ключем $K$, після чого підсумовується за mod 2 з другим «зашумлюючим» ключем $K_2$.

Таким чином, DESX має усього на дві операції XOR більше за оригінальний DES-алгоритм, що не вимагає значних витрат часу.

Водночас наявність цих двох операцій XOR значно поліпшує стійкість до повного перебирання ключів (загальна довжина ключа дорівнює $K + K_1 + K_2 = 56 + 64 + 64 = 184$ біти), а також робить його стійкішим до диференційного та лінійного криптоаналізів, збільшуючи необхідну кількість спроб із вибраним текстом до $2^{60}$ [2].

**S-DES**

Спрощений DES – це алгоритм шифрування, який має, скоріше, навчальне, ніж практичне значення. За своїми властивостями він подібний до DES, але має значно менше параметрів [1].

Цей алгоритм приймає на вході 8-бітний блок відкритого тексту та 10-бітний ключ, а на виході генерує 8-бітний блок шифрованого тексту. При розшифруванні на вхід алгоритму подається 8-бітний блок шифротексту і 10-бітний ключ, а на виході генерує 8-бітний блок відкритого тексту.

Алгоритм шифрування передбачає послідовне виконання п'яти операцій: початкової перестановки $IP$; раундової функції, що складається з перестановок і підстановок; перестановки $SW$, коли дві половинки блока по 4 біти переставляються місцями; ще одного застосування раундової функції; і, нарешті, перестановки $IP^{-1}$, оберненої до початкової. Послідовне використання кількох перестановок і підстановок, як це показано у [1-2], значно ускладнюють криптоаналіз.

Раундова функція приймає на вході не лише блок тексту, а й 8-бітний цикловий підключ, який утворюється з 10-бітного ключа.

Блок-схему алгоритму подано на рис. 3.5. З цього рисунка видно, що, оскільки це симетричний криптоалгоритм, він використовує для шифрування

та розшифрування той самий ключ. Тому ключ має бути як на передавальній, так і на приймальній стороні. З цього ключа на певних етапах шифрування та розшифрування генеруються два 8-бітних раундових підключі.

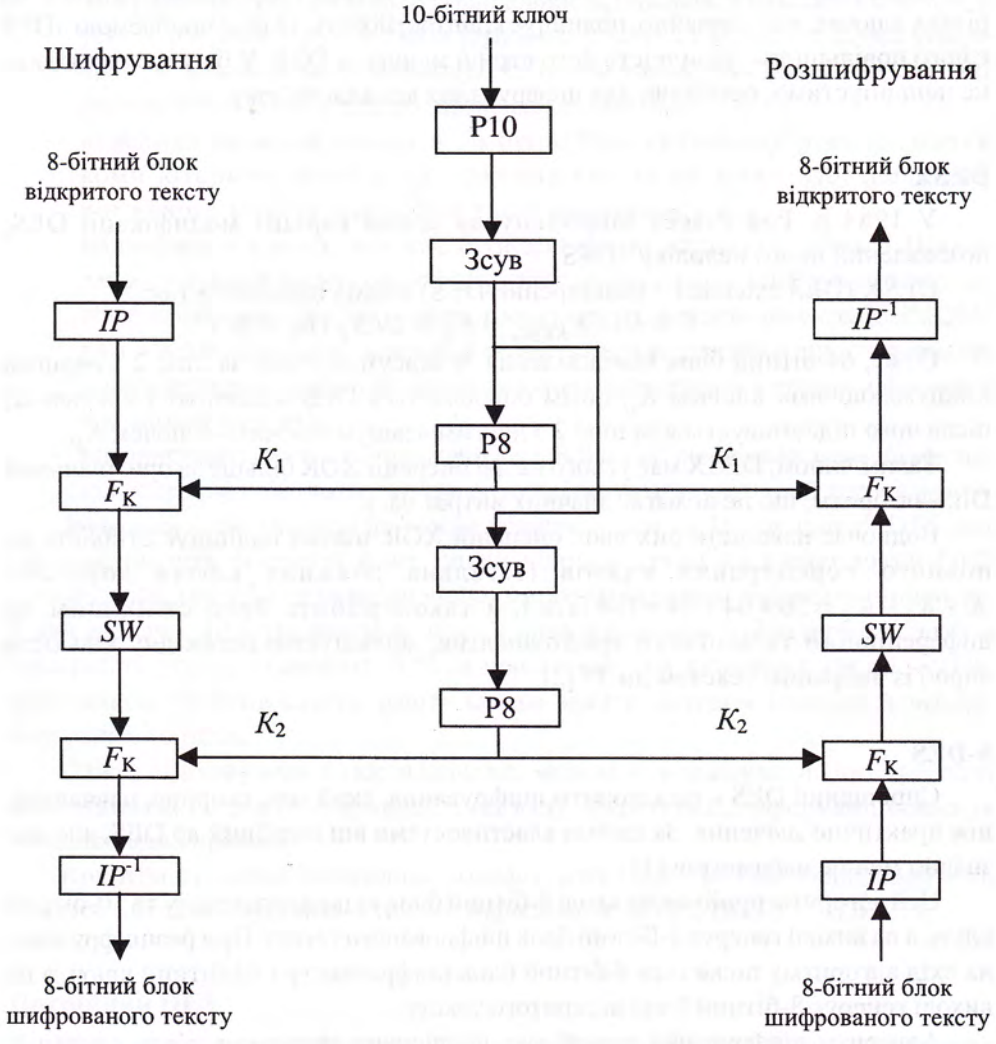



*Рис. 3.5. Схема спрощеного алгоритму S-DES*

***Процедура генерування раундових підключів***

1. Спочатку біти ключа переставляються так. Якщо 10-бітний ключ подати у вигляді $k_1,k_2,\ldots,k_{10}$, то перестановка P10 задається формулою

$$P(k_1,k_2,k_3,k_4,k_5,k_6,k_7,k_8,k_9,k_{10}) =$$
$$= (k_3,k_5,k_2,k_7,k_4,k_{10},k_1,k_9,k_8,k_6).$$

Можна також зобразити перестановку P10 у вигляді таблиці:

| P10 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 2 | 7 | 4 | 10 | 1 | 9 | 8 | 6 |

Ця таблиця символізує позицію біта вхідних даних у вихідній послідовності: першим стає 3-й біт, другим – 5-й, третім – 2-й і т.д. Наприклад, ключ (1010000010) відповідно до цієї перестановки перетворюється в послідовність (1000001100).

2. Ключ розділяється на дві 5-бітні половини. Окремо перша половина й окремо друга піддаються циклічному зсуву ліворуч на одну позицію. У нашому прикладі в результаті буде отримана послідовність (00001 11000).

3. Отримана послідовність піддається перестановці P8, у результаті якої з 10-бітного ключа обирається 8-бітна послідовність за таким правилом:

| P8 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 7 | 4 | 8 | 5 | 10 | 9 |

У результаті цієї операції ми отримаємо перший раундовий підключ ($K_1$). У нашому прикладі він буде мати вигляд (10100100).

4. Для генерування другого раундового підключа $K_2$ необхідно повернутися на крок назад, до двох 5-бітних рядків до застосування P8 та виконати для кожного з цих рядків циклічний зсув ліворуч на дві позиції. У нашому прикладі значення підключів (00001 11000) перетворяться у (00100 00011).

5. Нарешті, застосувавши до цієї послідовності перестановку P8, отримаємо другий раундовий підключ $K_2$. Для нашого прикладу результатом буде (01000011).

***Шифрування S-DES***

1. *Початкова й кінцева перестановки (IP та $IP^{-1}$).* На вхід алгоритму подається 8-бітний блок відкритого тексту, до якого застосовується початкова перестановка IP:

| IP | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 6 | 3 | 1 | 4 | 8 | 5 | 7 |

На завершальній стадії алгоритму виконується обернена перестановка $IP^{-1}$:

| $IP^{-1}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 3 | 5 | 7 | 2 | 8 | 6 |

Можна пересвідчитися, що ці дві таблиці дійсно обернені одна до одної, тобто $IP^{-1}(IP(M))=M$.

2. *Раундова функція $F_K$.* Розіб'ємо вхідний блок тексту після IP-перестановки на два 4-бітні підблоки. Лівий 4-бітний блок позначимо $L$, а правий – $R$. Тоді циклову функцію можна записати у вигляді формули

$$F_K(L,R) = (L \oplus F(R,K_i),R). \quad (1)$$

Тут $K_i$ означає цикловий підключ, $K_1$ або $K_2$; $\oplus$ – побітне XOR.

Тепер опишемо саму циклову функцію. На вході вона отримує 4-бітне значення $(n_1,n_2,n_3,n_4)$, тобто праву половину вхідного блока. Перша операція – операція розширення та перестановки. Її можна також зобразити таблицею

| Розширення з перестановкою | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 2 | 3 | 2 | 3 | 4 | 1 |

Зручніше цю операцію зобразити у вигляді матриці

$$n_4\ n_1\ n_2\ n_3$$
$$n_2\ n_3\ n_4\ n_1.$$

До цього значення додається 8-бітний підключ за допомогою операції XOR. Це можна зобразити так:

$$n_4+k_1\ \ n_1+k_2\ \ n_2+k_3\ \ n_3+k_4$$
$$n_2+k_5\ \ n_3+k_6\ \ n_4+k_7\ \ n_1+k_8.$$

Перейменуємо отримані елементи:

$$p_{00}\ p_{01}\ p_{02}\ p_{03}$$
$$p_{10}\ p_{11}\ p_{12}\ p_{13}.$$

Перші чотири біти (тобто перший рядок цієї матриці) далі подаються на вхід модуля заміни ($S$-матриці), $S_0$, на виході якого отримується 2-бітна послідовність. Другий рядок матриці подається на вхід другого модуля заміни, $S_1$, на виході якого також отримується 2-бітна послідовність.

Модулі $S_0$ і $S_1$ задаються так:

$$S_0 = \begin{matrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 1 \end{matrix}; \qquad S_1 = \begin{matrix} 1 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{matrix}.$$

Рядки та стовпчики нумеруються, починаючи з нуля.

Ці модулі заміни працюють так. Перший і четвертий біти вхідної послідовності вважаються двійковим представленням номера рядка, а другий і третій – номерами стовпця. Елемент, що знаходиться на перетині цих рядка і стовпця, задає двобітне вихідне значення. Наприклад, якщо $(p_{00},p_{03}) = (00)$ та $(p_{01}, p_{02})=(10)$, то вихідні два біти задаються значенням, яке знаходиться на перетині 0-го рядка та 3-го стовпців, тобто це буде число 3, а у двійковому представленні –11.

Аналогічну операцію виконують і з другим рядком $p_{10}\ p_{11}\ p_{12}\ p_{13}$.

Після застосування матриць заміни результат піддають перестановці P4 за таким законом:

| P4 | | | |
|---|---|---|---|
| 2 | 4 | 3 | 1 |

Результат перестановки P4 і буде результатом функції $F_K$. Отримана послідовність бітів додається за модулем 2 з лівою половиною L вхідного блока і буде новою лівою половиною. Права половина передається на вихід циклу без змін.

3. *Перестановка підблоків*. Як бачимо, за один раунд раундовою функцією обробляється лише ліва половина відкритого тексту, права половина залишається без змін. Для того, щоби зашифрувати й праву половину, використовується другий цикл, однак на його вхід треба подати переставлені підблоки: $L$ і $R$ поміняти місцями. Для цього й служить функція $SW$ – перемикач блоків.

Після переставляння підблоків один раунд алгоритму закінчено.

До переставлених підблоків знову застосовується циклова функція, як це описано вище. При другому викликові раундової функції розширення з перестановкою модулі $S_0$, $S_1$ і $P4$ залишаються тими ж, тільки використовується підключ $K_2$.

По закінченні другого раунду виконується $IP^{-1}$-перестановка, й роботу алгоритму закінчено, тобто на виході маємо зашифрований текст.

***Розшифрування зашифрованого тексту***

Як видно з рис. 3.5, розшифрування зашифрованого тексту виконується аналогічно шифруванню, за винятком того, що ключі подаються у зворотному порядку.

## *Особливості стандарту ГОСТ 28147-89*

Стандарт шифрування даних Радянського Союзу, ГОСТ 28147-89, було прийнято у 1989 році. В основі цього стандарту лежить DES-подібний алгоритм симетричного шифрування.

Розробники вказаного стандарту могли врахувати всі недоліки DES і успішно це зробили. ГОСТ 28147-89 й сьогодні залишається одним із найстійкіших алгоритмів шифрування.

Авторам цієї книжки не відомі успішні атаки на цей алгоритм. Його і досі успішно використовують на теренах колишнього Радянського Союзу для шифрування даних різного ступеня секретності.

*Опис ГОСТу*

ГОСТ є 64-бітним алгоритмом із 256-бітним ключем. У процесі роботи алгоритму на 32 етапах послідовно виконується простий алгоритм шифрування.

Для шифрування текст спочатку розбивається на ліву половину L і праву половину R. На етапі $i$ використовується підключ $K_i$. На $i$-му етапі алгоритму ГОСТу виконується таке:

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

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

Етап ГОСТу показаний на рис. 3.6. Функція f досить проста. Спочатку права половина додається за модулем $2^{32}$ з $i$-м підключем. Результат розбивається на вісім 4-бітних частин, кожна з яких надходить на вхід свого S-блока. ГОСТ використовує вісім різних S-блоків, перші 4 біти потрапляють у перший S-блок, другі 4 біти – в другий S-блок і т. д. Кожен S-блок є перестановкою чисел від 0 до 15. Наприклад, S-блок може виглядати так:

7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11.

У цьому випадку, якщо на вході S-блока 0, то на виході 7. Якщо на вході 1, на виході 10 і т. д. Усі вісім S-блоків різні, вони фактично є додатковим ключовим матеріалом. S-блоки зберігаються в таємниці.

Виходи всіх восьми S-блоків об'єднуються в 32-бітне слово, потім все слово циклічно зсувається вліво на 11 бітів. Нарешті результат об'єднується з допомогою XOR із лівою половиною, й отримується нова права половина, а права половина стає новою лівою половиною. Виконавши це 32 рази, отримаємо результат.

Генерація підключів досить проста. 256-бітний ключ розбивається на вісім 32-бітних блоків: $k_1$, $k_2$, ...$k_8$. На кожному етапі використовується свій підключ, як показано в табл. 3.9. Дешифрування виконується так само, як і шифрування, але інвертується порядок підключів $k_i$.

Стандарт ГОСТу визначає не спосіб генерації S-блоків, а лише спосіб їх зображення. Виробник створює перестановки S-блока самостійно за допомогою генератора випадкових чисел. Час від часу S-блоки повинні змінюватися.

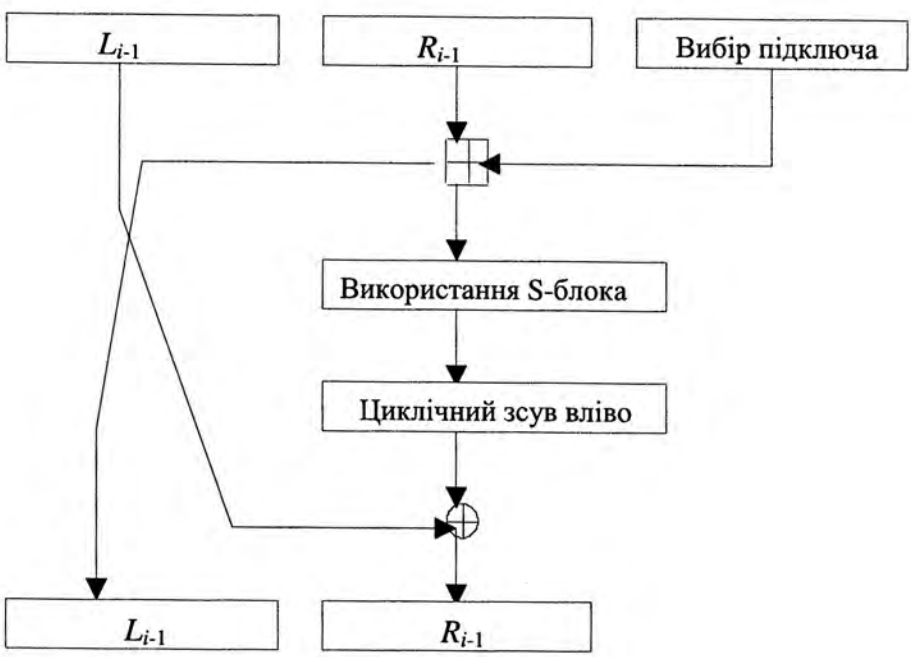



*Рис. 3.6. Блок-схема одного етапу алгоритма ГОСТ 28147-89*

Таблиця 3.9

| Етап | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Підключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Етап | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Підключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Етап | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Підключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Етап | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| Підключ | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

**Використання підключів на різних етапах ГОСТу**

***Порівняння крипостійкості DES і ГОСТу***

Ось головні відмінності між DES і ГОСТом:

– DES використовує складну процедуру для генерації підключів із ключів. У ГОСТ ця процедура дуже проста;

– у DES 56-бітний ключ, а в ГОСТі – 256-бітний. Якщо додати секретні перестановки S-блоків, то повний об'єм секретної інформації ГОСТу складе приблизно 610 бітів;

– у S-блоків DES 6-бітні входи і 4-бітні виходи, а у S-блоків ГОСТу 4-бітні і входи, і виходи. В обох алгоритмах використовується по вісім S-блоков, але розмір S-блока ГОСТу дорівнює одній четвертій розміру S-блока DES;

– у DES використовуються нерегулярні перестановки, названі P-блоком, а в ГОСТі – 11-бітний циклічний зсув вліво;

– у DES 16 етапів, а в ГОСТі – 32.

Якщо найкращим способом зламу ГОСТу є метод «грубої сили», то це дуже безпечний алгоритм. ГОСТ використовує 256-бітний ключ, а якщо враховувати секретні S-блоки, то довжина ключа зростає. ГОСТ, очевидно, стійкіший до диференційного і лінійного криптоаналізу, ніж DES. Хоча випадкові S-блоки ГОСТу слабкіші від фіксованих S-блоків DES, їх секретність збільшує стійкість ГОСТу до диференційного і лінійного криптоаналізу. До того ж ці способи зламу вразливі до кількості етапів – чим більше етапів, тим важчий злам. ГОСТ використовує в два рази більше етапів, ніж DES. Саме це робить неспроможними і диференційний, і лінійний криптоаналізи.

Інші частини ГОСТ такі ж, як у DES, або навіть слабкіші. ГОСТ не використовує перестановку з розширенням, на відміну від DES. Видалення цієї перестановки з DES послаблює його через зменшення лавинного ефекту, тому вважається, що відсутність такої операції в ГОСТі послаблює цей алгоритм. Додавання, що використовується в ГОСТі, не менш безпечне, ніж використовувана в DES операція XOR.

Найбільшою відмінністю є використання в ГОСТі циклічного зсуву замість перестановки. Перестановка DES збільшує лавинний ефект. У ГОСТі зміна

одного вхідного біта впливає на один S-блок одного етапу, який потім впливає на два S-блоки наступного етапу, три блоки наступного етапу і т.д. У ГОСТі потрібно 8 етапів, перш ніж зміна одного вхідного біта вплине на кожен біт результату, алгоритму DES для цього потрібно тільки 5 етапів. Це, звичайно ж, слабке місце. Але ГОСТ складається з 32 етапів, а DES – тільки з 16.

Розробники ГОСТу намагалися досягти рівноваги між безпекою і ефективністю. Вони змінили ідеологію DES так, щоб створити алгоритм, який більше підходить для програмної реалізації. Вони, можливо, менш упевнені в безпеці свого алгоритму і спробували компенсувати це дуже великою довжиною ключа, збереженням у секреті S-блоків і подвоєнням кількості ітерацій.

Наведемо порівняння стійкості DES і ГОСТу 28147-89 до повного перебору ключів (атака «грубою» силою) [11].

Нехай зловмисник володіє обчислювальними потужностями, здатними виконувати $W = 10^9$ спроб ключів на секунду. Тоді ймовірність успішності такої атаки можна оцінити так:

$$P_k(T) \approx \frac{T \cdot W}{K},$$

де $T$ – час, витрачений на підбір ключів ($C$), $W$ – кількість спроб за секунду, $K$ – потужність простору ключів.

Оцінимо ймовірність того, що за 1 рік роботи система зловмисника знайде правильний ключ:

$$P_k(1\text{ рік}) \approx \frac{3 \cdot 10^7\text{ с} \cdot 10^9}{2^{56}} \approx 0{,}44.$$

Ми бачимо, що $P_k(1\text{ рік}) < 0{,}5$. Результати такого розрахунку для DES і ГОСТ 28147-89 подано в табл. 3.10.

*Таблиця 3.10*

| Час | DES | ГОСТ |
|---|---|---|
| 1 рік | 0,44 | $2{,}72 \cdot 10^{-61}$ |
| 2 роки | 0,88 | $5{,}4 \cdot 10^{-61}$ |
| 10 років | 1 | $2{,}72 \cdot 10^{-60}$ |

Із таблиці видно, що велика довжина криптографічного ключа робить спроби атак «грубою силою» проти ГОСТу 28147-89 неефективними, а якщо врахувати наявність додаткових статичних ключів (S-блоки), ця атака взагалі видається неймовірною.

### *Режими застосування блокових шифрів*

Для шифрування вихідного тексту довільної довжини блокові шифри можуть використовуватися в декількох режимах. Існує чотири режими застосування

блокових шифрів, які найчастіше зустрічаються в системах криптографічного захисту інформації, а саме режими: електронної кодової книги (ECB – Electronic Code Book), зчеплення блоків шифрованого тексту (CBC – Cipher Block Chaining), оберненого зв'язку по шифрованому тексту (CFB – Cipher Feedback) і оберненого зв'язку по виходу (OFB – Output Feedback).

У режимі електронної кодової книги кожен блок вихідного тексту шифрується блоковим шифром незалежно від інших. Стійкість цього режиму рівна стійкості самого шифру. Але структура вихідного тексту при цьому не приховується. Кожний однаковий блок вихідного тексту приводить до появи однакового блока шифрованого тексту. Вихідним текстом можна легко маніпулювати шляхом видалення, повторення чи перестановки блоків. Швидкість шифрування дорівнює швидкості блокового шифру. Режим ECB допускає просте розпаралелювання для збільшення швидкості шифрування. Проте жодна обробка неможлива до надходження блока, за винятком генерації ключів.

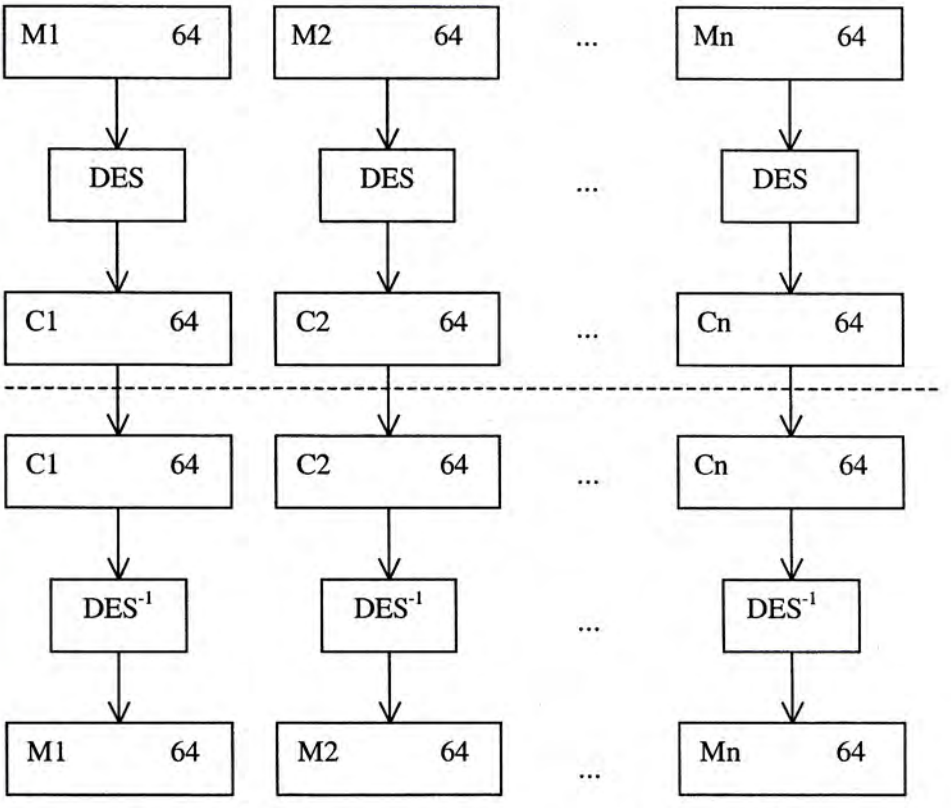



*Рис. 3.7. Блок-схема роботи блокового шифру в режимі ECB*

У режимі зчеплення блоків шифрованого тексту (CBC) кожний блок вихідного тексту додається порозрядно за модулем 2 з попереднім блоком шифрованого тексту, а потім шифрується. Для початку процесу шифрування використовується

синхропосилка (початковий вектор), яка передається по каналу зв'язку у відкритому вигляді. Стійкість даного режиму рівна стійкості блокового шифру, який лежить у його основі. Крім того, структура вихідного тексту приховується за рахунок додавання попереднього блока шифрованого тексту з черговим блоком відкритого тексту. Стійкість шифрованого тексту збільшується, оскільки стає неможливою пряма маніпуляція вихідним текстом, хіба що шляхом видалення блоків із початку чи кінця шифрованого тексту. Однією з потенційних проблем режиму CBC є можливість внесення контрольованих змін у наступний розшифрований блок вихідного тексту. Наприклад, якщо зловмисник змінить хоча б один біт в блоці, то весь блок буде розшифровано неправильно, але в наступному блоці з'явиться

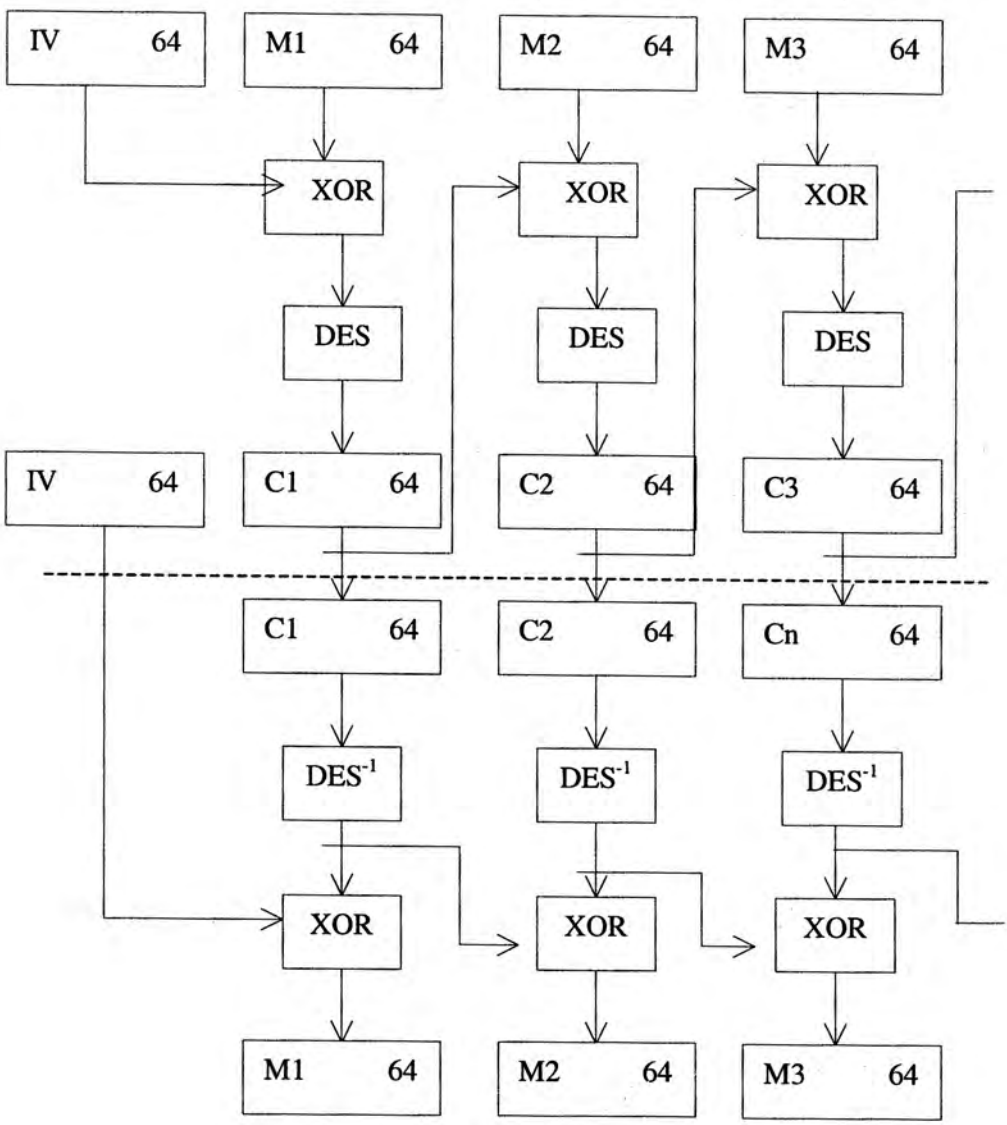



*Рис. 3.8. Блок-схема роботи блокового шифру в режимі CBC*

помилка у відповідному положенні. Для боротьби із загрозами вихідний текст повинен містити певну надлишковість.

Відомі певні модифікації режиму CBC. Розглянемо деякі з них. Режим зчеплення блоків шифрованого тексту з поширенням (PCBC – Propogating CBC) відрізняється тим, що за модулем 2 додається як попередній блок шифрованого, так і вихідного текстів.

Режим зчеплення блоків шифрованого тексту з контрольною сумою (CBCC – CBC with Checksum) відрізняється тим, що до попереднього блока вихідного тексту перед шифруванням додається сума за модулем 2 всіх попередніх блоків вихідного тексту. Це дає можливість проконтролювати цілісність переданого тексту з невеликими додатковими витратами.

У режимі оберненого зв'язку по шифрованому тексту (CFB) передавальний блок шифрованого тексту шифрується ще раз, і для отримання чергового блока шифрованого тексту результат додається порозрядно за модулем 2 з блоком вихідного тексту. Для початку процесу шифрування також використовується початковий вектор. Стійкість даного режиму рівна стійкості блокового шифру, який лежить у його основі. Структура вихідного тексту приховується за рахунок використання операції додавання за модулем 2. Маніпулювання вихідним текстом шляхом видалення блоків із початку до кінця шифрованого тексту стає неможливим. У режимі CFB, якщо два блоки шифрованого тексту

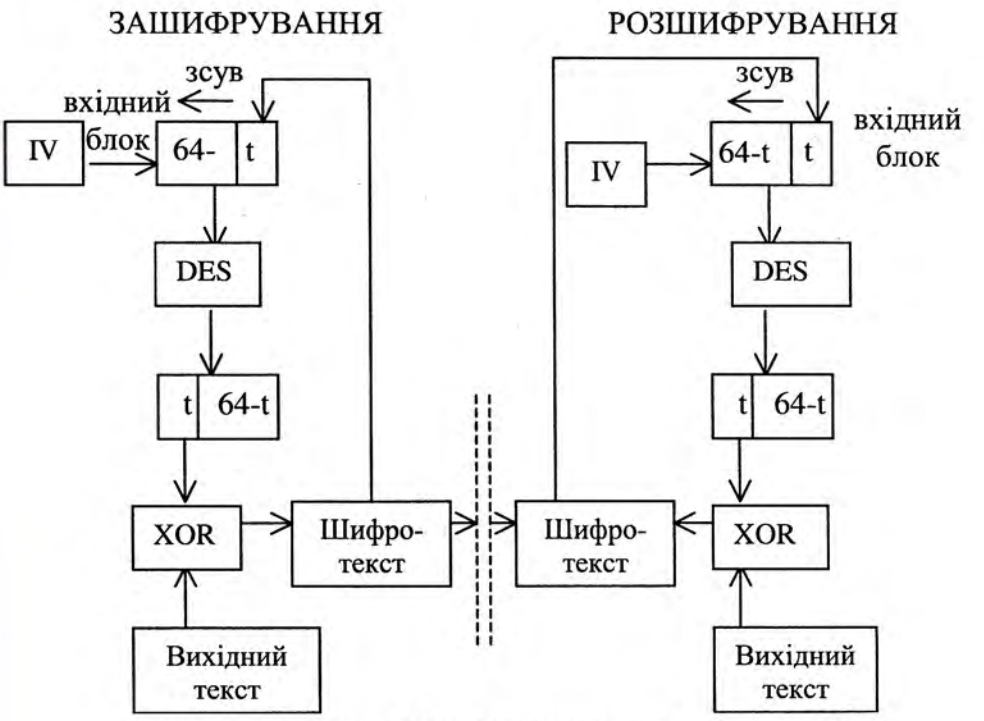



*Рис. 3.9. Блок-схема роботи блокового шифру в режимі CFB*

ідентичні, то результати їх шифрування на наступному кроці також будуть ідентичними, а це створює можливість витоку інформації про вихідний текст. Швидкість шифрування дорівнює швидкості роботи блокового шифру і простого способу розпаралелювання процесу шифрування також не здійснюється.

Режим оберненого зв'язку по виходу (OFB) подібний до режиму CFB, за винятком того, що величини, які додаються за модулем 2 з блоками вихідного тексту, генеруються незалежно від вихідного чи шифрованого тексту. Для початку процесу шифрування також використовується початковий вектор. Режим OFB має переваги над режимом CFB у тому сенсі, що будь-які бітні помилки, які виникають у процесі передачі, не впливають на розшифрування наступних блоків. Проте можлива проста маніпуляція вихідним текстом шляхом зміни шифрованого тексту. Існує модифікація цього режиму, що називається режимом оберненого зв'язку по виходу з нелінійною функцією (OFBNLF – OFB with a NonLinear Function). В цьому випадку на кожному кроці змінюється й ключ шифрування. Хоч у цьому випадку простого способу розпаралелювання процесу шифрування не існує, час можна зекономити, виробивши ключову послідовність заздалегідь.

У стандарті ГОСТ 28147-89 передбачається використання однойменного алгоритмув режимі генерування імітовставки.

Імітовставкою (або криптографічною контрольною сумою) називають контрольну комбінацію, що додається до повідомлення з метою захисту системи

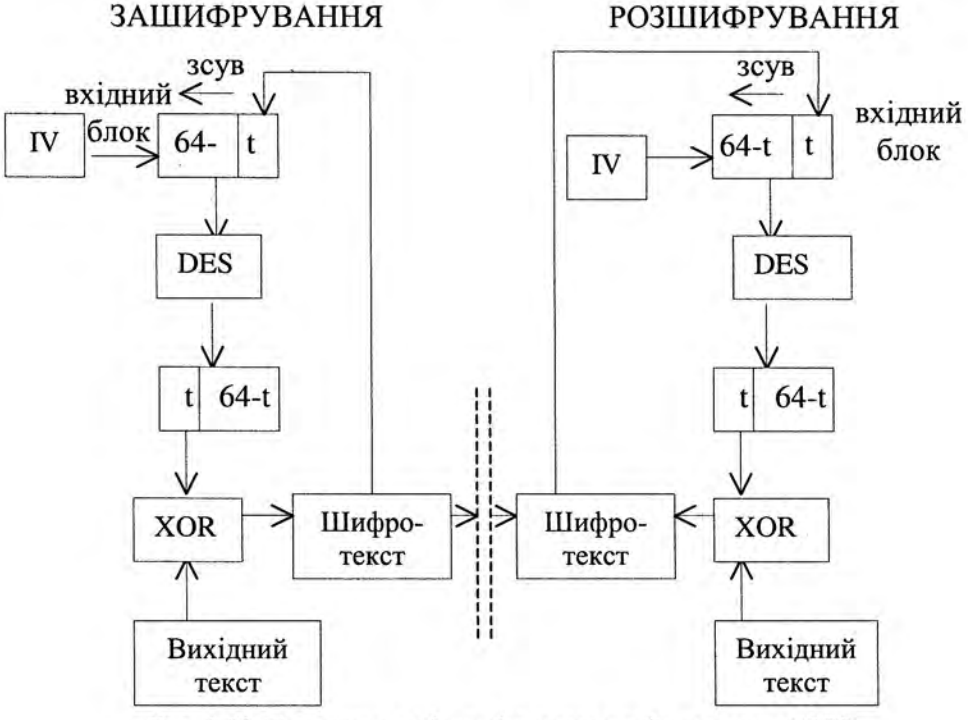



*Рис. 3.10. Блок-схема роботи блокового шифру в режимі OFB*

зв'язку від нав'язування хибних даних. Імітовставка залежить від змісту всього повідомлення та ключів шифрування.

Наявність імітовставки робить практично нереальним розв'язання двох таких задач без знання ключа:

– обчислення імітовставки для заданої відкритої інформації;

– підбір відкритої інформації під задану імітовставку.

Імітовставка обчислюється для відкритих текстів довжиною від 128 бітів. Як імітовставку здебільшого обирають 32 молодших біти результуючого блока. Вихідний текст $M$ розбивається на блоки $M_i$ по 64 біти. Перший блок $M_1$ подається на ГОСТ-машину. Отриманий блок шифротексту $C_1$ додається за модулем 2 з другим блоком $M_2$ відкритого тексту; результат $C_2$ – із третім блоком $M_3$ і т.д. до закінчення відкритого тексту. В результаті на виході отримується один 64-бітний блок, який залежить від усього повідомлення $M$ та ключа шифрування. Імітовставкою обирають 32 молодших біти цього блока.

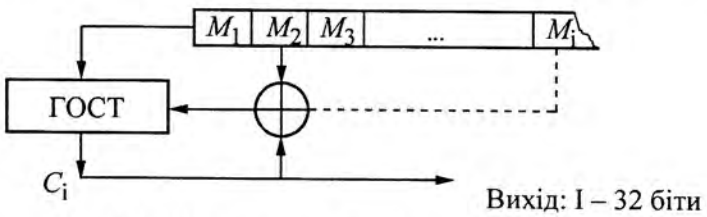



*Рис. 3.11. Імітовставка стандарту ГОСТ 28147-89*

## *Міжнародний стандарт шифрування даних IDEA*

Перший варіант шифру IDEA, запропонований Ксуеджа Лай і Джеймсом Массі, з'явився в 1990 році. Він називався PES (Proposed Encryption Standard, запропонований стандарт шифрування). Наступного року, після демонстрації Біхамом і Шаміром можливостей диференційного криптоаналізу, автори підсилили свій шифр проти такого злому й назвали новий алгоритм IPES (Improved Proposed Encryption Standard, поліпшений запропонований стандарт шифрування). В 1992 році назва IPES змінена на IDEA (International Data Encryption Algorithm, міжнародний алгоритм шифрування даних).

IDEA є блоковим шифром, він працює з 64-бітними блоками відкритого тексту. Довжина його ключа – 128 бітів. Для шифрування й розшифрування використовує той самий алгоритм [10].

В основі алгоритму лежить змішування трьох алгебраїчних груп, які можна легко реалізувати як апаратно, так і програмно. Ці групи такі:

– XOR;

– додавання за модулем $2^{16}$;

– множення за модулем $2^{16}+1$. Цю операцію можна вважати S-блоком IDEA.

Усі названі операції працюють із 16-бітними підблоками.

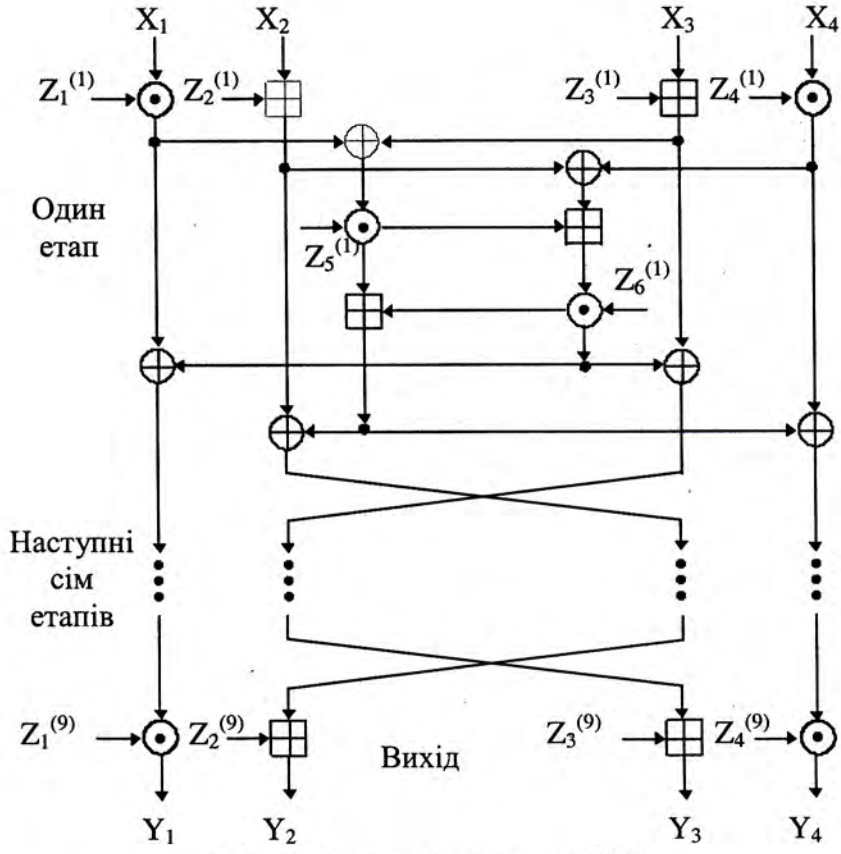



*Рис. 3.12. Блок-схема алгоритму IDEA*:

$X_i$ – 16-бітний підблок вікритого тексту;

$Y_i$ – 16-бітний підблок шифротексту;

$Z_i^{(r)}$ – 16-бітний підблок ключа;

⊕ – побітне «виключаюче АБО» (XOR) 16-бітних підблоків;

□ – додавання за модулем $2^{16}$ 16-бітних цілих;

⊙ – множення за модулем $2^{16}+1$ 16-бітних цілих за умови, що нульовий підблок відповідає $2^{16}$

*Опис IDEA*

Схема IDEA наведена на рис. 3.12. 64-бітний блок даних ділиться на чотири 16-бітних підблоки: $X_1$, $X_2$, $X_3$ і $X_4$. Ці чотири підблоки стають вхідними даними для першого етапу алгоритму. Всього в алгоритмі вісім етапів. На кожному етапі чотири підблоки підлягають операції додавання і множення один з одним і з шістьма 16-бітними підключами. Між етапами міняються місцями другий і третій підблоки. Нарешті чотири підблоки об'єднуються з чотирма підключами в кінцевому перетворенні. На кожному етапі події відбуваються в такій послідовності:

1. Перемножуються $X_1$ і перший підключі.
2. Додаються $X_2$ і другий підключі.
3. Додаються $X_3$ і третій підключі.
4. Перемножуються $X_4$ і четвертий підключі.
5. Виконується XOR над результатами кроків 1 і 3.
6. Виконується XOR над результатами кроків 2 і 4.
7. Перемножуються результати кроку 5 і п'ятий підключ.
8. Додаються результати кроків 6 і 7.
9. Перемножуються результати кроку 8 і шостий підключ.
10. Додаються результати кроків 7 і 9.
11. Виконується XOR над результатами кроків 1 і 9.
12. Виконується XOR над результатами кроків 3 і 9.
13. Виконується XOR над результатами кроків 1 і 10.
14. Виконується XOR над результатами кроків 4 і 10.

Виходом етапу є чотири підблоки – результати дій 11, 12, 13 і 14. Помінявши місцями два внутрішні підблоки (але не на останньому етапі), отримуються початкові дані для наступного етапу.

Після восьмого етапу виконується завершальне перетворення:

1. Перемножуються $X_1$ і перший подключі.
2. Додаються $X_2$ і другий подключі.
3. Додаються $X_3$ і третій подключі.
4. Перемножуються $X_4$ і четвертий подключі.

Нарешті чотири підблоки знову з'єднуються, утворюючи шифротекст.

Нескладно створювати й підключі. Алгоритм використовує 52 з них (шість для кожного з восьми етапів і ще чотири для завершального перетворення). Спочатку 128-бітний ключ ділиться на вісім 16-бітних підключів. Це перші вісім підключів алгоритму (шість для першого етапу і два – для другого). Потім ключ циклічно зсувається вліво на 25 бітів і знову ділиться на вісім підключів. Перші чотири використовуються на етапі 2, а решта чотири – на етапі 3. Ключ циклічно зсувається вліво на 25 бітів для отримання наступних восьми підключів, і так до кінця алгоритму.

Розшифрування здійснюється так само, за винятком того, що підключі інвертуються і трохи змінюються. Підключі при розшифруванні є оберненими значеннями ключів шифрування по відношенню до операцій чи то додавання, чи то множення. Ці обчислення можуть зайняти якийсь час, але їх потрібно виконати один раз для кожного ключа дешифрування. В табл. 3.11 подані підключі шифрування і відповідні підключі дешифрування.

*Таблиця 3.11*

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $Z_1^{(1)}$ | $Z_2^{(1)}$ | $Z_3^{(1)}$ | $Z_4^{(1)}$ | $Z_5^{(1)}$ | $Z_6^{(1)}$ | $Z_1^{(9)-1}$ | $-Z_2^{(9)}$ | $-Z_3^{(9)}$ | $Z_4^{(9)-1}$ | $Z_5^{(8)}$ | $Z_6^{(8)}$ |
| 2 | $Z_1^{(2)}$ | $Z_2^{(2)}$ | $Z_3^{(2)}$ | $Z_4^{(2)}$ | $Z_5^{(2)}$ | $Z_6^{(2)}$ | $Z_1^{(8)-1}$ | $-Z_2^{(8)}$ | $-Z_3^{(8)}$ | $Z_4^{(8)-1}$ | $Z_5^{(7)}$ | $Z_6^{(7)}$ |
| 3 | $Z_1^{(3)}$ | $Z_2^{(3)}$ | $Z_3^{(3)}$ | $Z_4^{(3)}$ | $Z_5^{(3)}$ | $Z_6^{(3)}$ | $Z_1^{(7)-1}$ | $-Z_2^{(7)}$ | $-Z_3^{(7)}$ | $Z_4^{(7)-1}$ | $Z_5^{(6)}$ | $Z_6^{(6)}$ |
| 4 | $Z_1^{(4)}$ | $Z_2^{(4)}$ | $Z_3^{(4)}$ | $Z_4^{(4)}$ | $Z_5^{(4)}$ | $Z_6^{(4)}$ | $Z_1^{(6)-1}$ | $-Z_2^{(6)}$ | $-Z_3^{(6)}$ | $Z_4^{(6)-1}$ | $Z_5^{(5)}$ | $Z_6^{(5)}$ |
| 5 | $Z_1^{(5)}$ | $Z_2^{(5)}$ | $Z_3^{(5)}$ | $Z_4^{(5)}$ | $Z_5^{(5)}$ | $Z_6^{(5)}$ | $Z_1^{(5)-1}$ | $-Z_2^{(5)}$ | $-Z_3^{(5)}$ | $Z_4^{(5)-1}$ | $Z_5^{(4)}$ | $Z_6^{(4)}$ |
| 6 | $Z_1^{(6)}$ | $Z_2^{(6)}$ | $Z_3^{(6)}$ | $Z_4^{(6)}$ | $Z_5^{(6)}$ | $Z_6^{(6)}$ | $Z_1^{(4)-1}$ | $-Z_2^{(4)}$ | $-Z_3^{(4)}$ | $Z_4^{(4)-1}$ | $Z_5^{(3)}$ | $Z_6^{(3)}$ |
| 7 | $Z_1^{(7)}$ | $Z_2^{(7)}$ | $Z_3^{(7)}$ | $Z_4^{(7)}$ | $Z_5^{(7)}$ | $Z_6^{(7)}$ | $Z_1^{(3)-1}$ | $-Z_2^{(3)}$ | $-Z_3^{(3)}$ | $Z_4^{(3)-1}$ | $Z_5^{(2)}$ | $Z_6^{(2)}$ |
| 8 | $Z_1^{(8)}$ | $Z_2^{(8)}$ | $Z_3^{(8)}$ | $Z_4^{(8)}$ | $Z_5^{(8)}$ | $Z_6^{(8)}$ | $Z_1^{(2)-1}$ | $-Z_2^{(2)}$ | $-Z_3^{(2)}$ | $Z_4^{(2)-1}$ | $Z_5^{(1)}$ | $Z_6^{(1)}$ |
| Заключне перетворення | $Z_1^{(9)}$ | $Z_2^{(9)}$ | $Z_3^{(9)}$ | $Z_4^{(9)}$ | | | $Z_1^{(1)-1}$ | $-Z_2^{(1)}$ | $-Z_3^{(1)}$ | $Z_4^{(1)-1}$ | | |

**Підключі шифрування й дешифрування IDEA**

Певним недоліком алгоритму IDEA можна вважати неможливість зміни таких параметрів, як довжина ключа й довжина блока шифрування.

## *Особливості стандарту AES*

У кінці 1996 р. Національним інститутом стандартів США (NIST) було оголошено конкурс на створення нового загальнонаціонального стандарту шифрування, який повинен прийти на заміну DES. Розробленому стандарту присвоїли робочу назву AES (Advanced Encryption Standard).

2 жовтня 2000 року прийнято остаточне рішення. Як запропонований стандарт обрали алгоритм Rijndael. Цей алгоритм розроблений Вінсентом Рійманом і Йоаном Дайменом і є алгоритмом, який не використовує сітку Фейстеля.

На відміну від DES і ГОСТу, алгоритм Rijndael використовує кінцеву групу точок еліптичної кривої.

Із теорії чисел відомі операції цілочислового ділення та залишку від ділення [16,17]. Наприклад, 7:5 = 1 і 2 у залишку. Взяття залишку від цілочислового ділення записується так: 2 = 7mod5. Залишки за mod $p$ утворюють кінцеву послідовність цілих чисел від 0 до $p$-1. Наприклад, група залишків за mod 7 складається з чисел 0, 1, 2, 3, 4, 5, 6.

Для таких чисел можна ввести операції додавання та множення за модулем. Наприклад,

$$(4+5) \bmod 7 = 9 \bmod 7 = 2,$$
$$(3 \times 5) \bmod 7 = 15 \bmod 7 = 1.$$

Як бачимо, і сума, і добуток двох елементів такої групи також є членами цієї групи (детальніше див. Розділ IV).

Тепер можна визначити обернені елементи групи. Елемент $a^{-1}$ називається оберненим до $a$, якщо $a \times a^{-1} = 1 \bmod p$.

Наприклад, обереним до числа 4 за mod7 буде число 2mod7. Дійсно, $(4 \times 2) \bmod 7 = 8 \bmod 7 = 1$. Тепер можна просто ввести операцію ділення, як множення на обернений елемент, операцію віднімання і т.д. Виявляється, якщо $p$ – просте число, то оберений елемент існує для всіх елементів множини, крім 0. Множини залишків, для яких уведено операції додавання, множення та розкриття дужок, і всі члени якої, крім 0, мають оберений елемент, будемо називати *кінцевим полем* або полем Галуа *GF(p)*.

Можна утворити поле Галуа з точок т.зв. еліптичної кривої. Еліптичною кривою назвемо множину пар точок *(x,y)*, що задовольняють рівняння

$$y^2 = ax^2 + bx + c.$$

Обмежуючи *x, y, a,b,c*, ми можемо утворити кінцеве поле над точками кривої:

$$y^2 = (ax^2 + bx + c) \bmod p.$$

Наприклад, якщо оберемо утворююче рівняння $y^2 = x^3 - x$, то, обмежившись полем дійсних чисел, будемо мати криву (див. рис. 3.13):

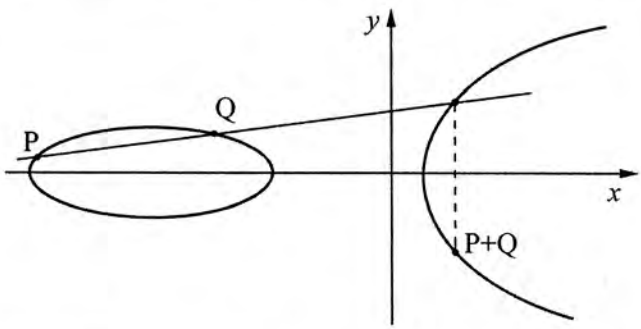



*Рис. 3.13. Еліптична крива, обмежена полем дійсних чисел*

Для криптографічних застосувань найчастіше використовують еліптичну криву над простим кінцевим полем Галуа *GF(p)*, яка визначається як множина пар *(x,y)*, таких, що $x, y \equiv GF(p)$ і задовольняють рівняння [10]:

$$y^2 = (x^3 + ax + b) \bmod p,$$

причому $a$ і $b$ також належать *GF(p)*. Пару *(x,y)* будемо називати точкою. Для точок еліптичної кривої можна побудувати операції додавання та множення, визначити нульовий та обернений елементи. Наприклад, для кривої $y^2 = x^3 - x$ їх можна визначити так, як показано на рис. 3.13 для точок $P$ і $Q$.

Множину точок еліптичної кривої разом із нульовим елементом і визначеною

операцією додавання будемо називати групою. Для кожної еліптичної кривої група, тобто кількість точок, скінченна, хоча й дуже велика.

Важливу роль у криптографічних алгоритмах на основі еліптичних кривих відіграють т.зв. точки кратності.

Точку $Q$ будемо називати точкою кратності $k$, якщо для деякої точки $P$ виконується рівність

$$P=\underbrace{Q+Q+...+Q}_{k\text{–разів}}=kQ.$$

Точки кратності еліптичної кривої вважають аналогами степенів чисел у простому полі. Задача обчислення точок кратності еліптичної кривої є складною математичною задачею, на якій і ґрунтується криптостійкість більшості криптографічних алгоритмів, що використовують еліптичні криві. Розв'язок цієї задачі вважається складнішим за дискретне логарифмування (див. алгоритм Ель-Гамаля), тому довжина ключів «еліптичних» алгоритмів менша при більшій криптостійкості.

***Опис AES***

В алгоритмі AES використовується поле Галуа $GF(2^8)$, побудоване як розширення поля $GF(2)$ за коренями многочлена $m(x)=x^8+x^4+x^3+x+1$. Цей многочлен обрано з міркувань ефективності представлення елементів поля. Елементарні операції, які використовуються в алгоритмі, виконуються у вказаному полі.

Алгоритм Rijndael являє собою блоковий шифр зі змінною довжиною блока і змінною довжиною ключа. Довжини блока і ключа можуть бути обрані незалежно такими, що дорівнюють 128, 192 чи 256 бітам.

Робота алгоритму Rijndael базується на прямому перетворенні блока, який зображається як матриця байтів. На кожному кроці обробляється блок у цілому, й незмінних частин блока не залишається. Оскільки за один раунд шифрується новий блок, таких раундів треба менше. Блок-схему алгоритму подано на рис. 3.14.

Вхідні дані M подаються у вигляді прямокутної матриці байтів $4\times n$, де $n=4, 6, 8$ в залежності від розміру блока. Позаяк AES використовує розмір блока в 128 бітів, то матриця байтів має розмір $4\times 4$.

Раундова функція $F(x_i)$ складається з чотирьох етапів:

1) BS (Byte Sub) – побайтова заміна:

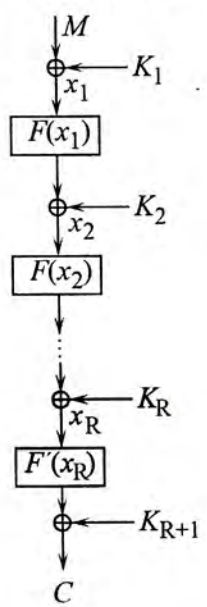



*Рис. 3.14. Блок-схема алгоритму Rijndael*

$$a_{ij} \to b_{ij}.$$

Це блок нелінійної оберненої байтової заміни (S-блок), складається з двох ітерацій.

Кожен байт замінюється на мультикативний обернений до нього в полі $GF(2^8)$. Байт зі значенням 00h відображається в себе.

Над кожним байтом виконується афінне перетворення в полі GF(2), яке задається таким рівнянням:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Це афінне перетворення може бути описане в поліномному вигляді як $b(x) = (x^7 + x^6 + x^2 + x) + a(x) \cdot (x^7 + x^6 + x^5 + x^4 + 1) \bmod (x^8 + 1)$. Поліном, на який відбувається множення, вибраний взаємопростим із модулем, так що множення обернене.

Оберненим до ByteSub буде перетворення, що складається з оберненого афінного перетворення й отримання мультиплікативного зворотного в $GF(2^8)$.

Для заміни використовується загальний вектор заміни.

2) SR (Shift Row) – зсув рядків:

$$b_{00}b_{01}b_{02}b_{03} \to b_{00}b_{01}b_{02}b_{03}$$
$$b_{10}b_{11}b_{12}b_{13} \to b_{11}b_{12}b_{13}b_{10}$$
$$b_{20}b_{21}b_{22}b_{23} \to b_{22}b_{23}b_{20}b_{21}$$
$$b_{30}b_{31}b_{32}b_{33} \to b_{33}b_{30}b_{31}b_{32}\ .$$

Як бачимо, перший рядок матриці $4 \times 4$ не зсувається, 2-й – зсувається на 1 байт вліво, 3-й – на 2 байти, 4-й – на 3 байти.

3) MC (Mix Column) – операція над стовпчиками, коли кожен стовпчик множиться за правилом поля Галуа на точках еліптичної кривої на фіксовану матрицю.

У цьому перетворенні рядки масиву стану розглядаються як поліноми над полем *GF(2⁸)*. Перетворення полягає в множенні стовпця по модулю $x^4 + 1$ на фіксований поліном

$$c(x) = '03h' x^3 + '01h' x^2 + '01h' x + '02h'.$$

Цей поліном є взаємно простим з $x^4+1$ і тому множення обернене. В матричній формі дане перетворення можна представити як

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02\,03\,01\,01 \\ 01\,02\,03\,01 \\ 01\,01\,02\,03 \\ 03\,01\,01\,02 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

Обернене перетворення являє собою множення на поліном, мультиплікативно обернений до $c(x)$ по модулю $x^4+1$:

$$d(x) = '0Bh'\,x^3 + '0Dh'\,x^2 + '09h'\,x + '0Eh'.$$

Останній раунд не використовує Mix Column.

4) AK (Add Round Key) – додавання раундового ключа.

Додавання ключа ітерації виконується шляхом простого побітного додавання за модулем 2 кожного байта масиву з відповідним байтом ключа. Це перетворення обернене самому собі.

*Алгоритм обробки ключа*

Ключі ітерації отримуються з ключа шифрування з допомогою алгоритму обробки ключа, який складається з двох компонентів – розширення ключа й вибору ключа ітерації. Основними принципами його побудови є:

1. Загальна кількість біт ключів ітерації дорівнює довжині блока, помноженій на кількість ітерацій плюс одиниця (наприклад, для блока розміром 128 бітів і 10 ітерацій потрібно 1408 бітів ключів ітерації).
2. Ключ шифрування збільшується до розширеного ключа.
3. Ключі ітерації беруться з розширеного ключа в такий спосіб: перший ключ ітерації складається з перших Nb слів, другий – із наступних Nb слів і т.д.

*Алгоритм розширення ключа*

Розширений ключ являє собою одновимірний масив 4-байтових слів і позначається $W[Nb \cdot (Nr+1)]$. Перші $Nk$ слів збігаються з ключем шифру. Усі наступні слова визначені рекурсивно через слова з меншими індексами. Алгоритм розширення ключа залежить від значення $Nk$. Для $Nk$ меншого або такого, що дорівнює 6, використовують один варіант алгоритму, а для $Nk$, більшого від 6, – інший.

У випадку, коли $Nk \le 6$, використовують функцію Byte Sub, яка повертає слово, в якому кожен байт є результатом застосування блока заміни шифру, до байта, який знаходиться на відповідній позиції у вхідному слові. Далі здійснюють циклічний зсув байтів у слові, так що вхідне слово (a, b, c, d) перетворюється в слово (b, c, d, a).

Перші $Nk$ слів заповнюють із ключа шифру. Кожне наступне слово $W[i]$ дорівнює сумі за модулем 2 попереднього слова $W[i-1]$ та слова $W[i-Nk]$, що

на $Nk$ позицій лівіше від слова $W[i]$. Для слів у позиціях, які кратні числу $Nk$, перед додаванням за модулем 2 зі словом $W[i-Nk]$ слово $W[i-1]$ спочатку піддають певному перетворенню, а потім ще додають за модулем 2 з раундовою сталою. Перетворення полягає в тому, що спочатку виконують циклічний зсув байтів у слові, а потім застосовують до кожного з чотирьох байтів слова побітову заміну.

*Для $Nk>6$ різниця у виконанні алгоритму полягає в такому:* якщо $i-4$ кратне $Nk$, то перед додаванням за модулем 2 зі словом $W[i-Nk]$ до слова $W[i-1]$ застосовують побайтову заміну.

*Криптостійкість алгоритму Rijndael*

Дослідження показали, що криптоаналіз 16-раундового DES у принципі реальний, однак вимагає великої кількості вихідних даних. Криптоаналіз при 20-24 циклах стає навіть теоретично неможливим.

Алгоритм ГОСТу 28147-89 має 32 раунди. Як ми бачили, його криптостійкість має значний запас і сьогодні. У відкритих публікаціях відсутні повідомлення про успішні атаки на ГОСТ.

За оцінками розробників Rijndael [7,10], вже на чотирьох раундах шифрування цей алгоритм досить стійкий для сучасних використань. Теоретичною межею вважається 6-8 раундів. Отже, 10-14 раундів, передбачених у цьому алгоритмі, мають значний запас щодо криптостійкості.

До недоліків алгоритму можна віднести нетрадиційну, недостатньо вивчену схему, яка може містити приховані вразливості, невідомі на час конструювання алгоритму.

Насамкінець, подамо порівняльну таблицю характеристик блокових алгоритмів.

*Таблиця 3.12*

| **Характеристика** | **DES** | **ГОСТ** | **IDEA** | **Rijndael** |
|---|---|---|---|---|
| Довжина блока за 1 раунд, біти | 64 | 64 | 64 | 128, 192, 256 |
| Довжина ключа, біти | 64 (56) | 256 | 128 | 128, 192, 256 |
| Розмір раундового ключа, біти | 48 | 32 | 16 | 128, 192, 256 |
| Операції | Адитивні підстановки та зсуви | | Додавання та множення | Операції над кінцевими полями |
| Еквівалентність прямого і обернених перетворень | До порядку розміщення ключових елементів | | До використання ключового елемента | До вектора ключових елементів, вузла замін та інших констант |
| Кількість раундів | 16 | 32 | 8 | 10, 12, 14 |

Як стандарт AES було обрано варіант Rijndael із розміром блока у 128 бітів. Кількість раундів залежить, як бачимо з таблиці, від розміру блока й ключа. Якщо максимальний із них дорівнює 128 бітам, то використовують 10 раундів, якщо 192 бітам – 12, для 256 бітів – 14 раундів. Це дозволяє обирати оптимальний варіант у залежності від потрібної криптостійкості та швидкості обробки.

## Потокові шифри

### Загальні відомості про потокові шифри

Потокові шифри являють собою різновид гаммування й перетворюють відкритий текст у шифрований послідовно по 1 біту. Генератор ключової послідовності видає послідовність бітів $k_1, k_2, ..., k_i, ...$ Ця ключова послідовність додається за модулем 2 з послідовністю бітів вихідного тексту $p_1, p_2, ..., p_i, ...$ для отримання шифрованого тексту:

$$p_i \oplus k_i = c_i .$$

Стійкість системи цілком залежить від внутрішньої структури генератора ключової інформації. Якщо генератор видає послідовність із невеликим періодом, то стійкість системи буде невеликою. І навпаки, якщо генератор буде видавати нескінченну послідовність істинно випадкових (але не псевдовипадкових) бітів, отримаємо одноразовий блокнот з ідеальною стійкістю.

Реальна стійкість потокових шифрів лежить десь посередині між стійкістю простої моноалфавітної підстановки й одноразового блокнота. Генератор ключової послідовності видає потік бітів, який виглядає випадковим, але в дійсності є детермінованим і може бути точно відтворений на приймальній стороні. Чим більше згенерований потік схожий на випадковий, тим більше зусиль докладе криптоаналітик для взлому шифру.

Але, якщо щоразу при запуску генератор буде видавати одну й ту ж послідовність, злом криптосистеми стане досить легкою задачею. Перехопивши два шифрованих тексти, зловмисник може додати їх за модулем 2 і отримати два вихідних тексти, доданих також за модулем 2. Таку систему розкрити дуже просто. Якщо в руках зловмисника з'явиться пара вихідний текст-шифрований текст, задача взагалі стане елементарною.

Саме з цієї причини всі потокові шифри передбачають використання ключа. Вихід генератора ключової послідовності залежить від цього ключа. В такому випадку простий криптоаналіз стане неможливим.

Потокові шифри придатні для шифрування неперервних потоків, наприклад у мережах передачі даних.

Структуру генератора ключової послідовності можна зобразити у вигляді кінцевого автомата з пам'яттю, який складається з трьох блоків: блока пам'яті, що зберігає інформацію про стан генератора; вихідної функції, яка генерує біт ключової послідовності в залежності від стану, і функції переходів, що задають новий стан, куди перейде генератор на наступному кроці.

У 1946 році в США була запатентована базова ідея так званих потокових шифрів, які самосинхронізуються (чи шифрування з автоключем – CipherText Auto Key (СТАК)). Вона полягає в тому, що внутрішній стан генератора є функцією фіксованого числа передуючих бітів шифрованого тексту. Оскільки внутрішній стан залежить тільки від $n$ бітів шифрованого тексту, генератор на приймальній стороні синхронізується з передаваючою стороною після отримання $n$ бітів.

Реалізація цього підходу виглядає так. Кожне повідомлення називають випадковим заголовком (довжиною $n$ бітів). Цей заголовок шифрується й передається в лінію. На приймальній стороні заголовок розшифровується. Результат розшифрування буде неправильний, але після обробки $n$ бітів заголовка обидва генератори будуть синхронізовані.

Недоліком системи є поширення помилок. При спотворенні одного біта генератор на приймальній стороні видасть $n$ неправильних бітів ключової послідовності, поки помилковий біт не буде виштовхнутий із пам'яті, що призведе до помилкового розшифрування n бітів вихідного тексту.

Окрім того, шифри, які самосинхронізуються, вразливі до атак типу «відтворення». Зловмисник записує деяку кількість бітів шифрованого тексту. Згодом він замінює біти трафіку записаними – відтворює їх. Після деякої кількості «сміття», поки приймальна сторона не синхронізується, старий шифрований текст буде розшифровуватися нормально. У приймальної сторони немає жодних засобів визначення того, що дані, які приймаються, неактуальні.

Потокові шифри, які самосинхронізуються, можуть бути реалізовані у вигляді блокових шифрів, використовуваних у режимі оберненого зв'язку по шифрованому тексту. При цьому за один раз може шифруватися довільна кількість бітів, яка менша або дорівнює довжині блока. Це можна проілюструвати на наступному прикладі шифрування по одному байту за цикл.

Блоковий шифр працює над чергою розміром, який дорівнює довжині блока. Початково черга заповнюється синхропосилкою. Потім черга шифрується, й ліві 8 бітів додаються до перших 8 бітів вихідного тексту. Отримані 8 бітів шифрованого тексту передаються в лінію, черга зсувається вліво на 8 бітів, ліві біти відкидаються, а праві заповнюються 8 бітами шифрованого тексту, переданими в лінію. Потім процедура повторюється. Число 8 взято тільки для прикладу. За один цикл роботи блокового алгоритму може шифруватися і 1 біт, хоча це буде не надто ефективним з точки зору швидкості роботи схеми. В режимі CFB (режим оберненого зв'язку по шифрованому тексту) синхропосилка повинна бути унікальною для кожного повідомлення протягом терміну дії ключа. Якщо це буде не так, зловмисник відновить вихідний текст.

У випадку виникнення помилки в шифрованому тексті на приймальній стороні можуть виникнути помилки при розшифруванні поточного й наступних $[m/n]$ блоків, де $m$ – розмір блока, $n$ – кількість бітів, які шифруються за 1 цикл, тобто поки помилковий біт шифрованого тексту не буде витіснений із пам'яті.

У цьому випадку вихідні значення генератора не залежать від вихідного чи шифрованого текстів. Такі потокові шифри називаються синхронними.

Основна складність у цьому підході полягає в необхідності синхронізації генераторів ключа на передаючій і приймальних сторонах. Якщо в процесі передачі відбулася втрата одного біта, то вся послідовність бітів шифрованого тексту після помилкового біта не зможе бути розшифрована. Якщо таке станеться, сторони повинні провести повторну синхронізацію. При цьому синхронізація має бути проведена так, щоб жоден відрізок ключової інформації не повторився, тобто повернення до деякого попереднього стану генератора неможливе.

Позитивною властивістю синхронних потокових шифрів є відсутність ефекту поширення помилок. Один спотворений біт при передачі призведе до спотворення тільки одного біта тексту при розшифруванні.

Синхронні шифри також захищають від вставок і викидів відрізків шифрованого тексту з потоку. Такі операції спричинять порушення синхронізації, що буде одразу ж виявлено на приймальній стороні.

Проте такі шифри вразливі до змін окремих бітів. Якщо зловмисник знає вихідний текст, то він зможе змінювати біти в потоці шифрованого тексту так, що він буде розшифровуватися, як необхідно зловмиснику.

Синхронний потоковий шифр може бути реалізований у вигляді блокового шифру, який працює в режимі оберненого зв'язку по виходу. Режим OFB (режим оберненого зв'язку по виходу) може бути реалізований з будь-яким розміром оберненого зв'язку, меншим від розміру блока. Проте даний спосіб не рекомендується, оскільки при цьому знижується період генератора до приблизно $2^{\frac{m}{2}}$. При довжині блока 64 біта це буде приблизно $2^{32}$, чого явно недостатньо.

У цьому випадку вихідна функція часто вибирається простою, наприклад, сумою за модулем 2 декількох бітів стану чи взагалі одним бітом стану. Криптографічна стійкість забезпечується функцією переходу до наступного стану, яка залежить від ключа. Деколи такий режим називають режимом із внутрішнім оберненим зв'язком, оскільки обраний зв'язок внутрішній порівняно з алгоритмом генерації ключової послідовності.

Варіантом цього режиму є схема, коли ключ задає початковий стан генератора, після чого останній працює без подальшого втручання.

Ще один спосіб побудови потокового шифру – використання лічильника як вхідного значення для блокового шифру. Після кожного циклу шифрування блока значення лічильника збільшується, найчастіше на одиницю. Властивості даного режиму щодо поширення помилок і синхронізації будуть такими ж, як і для режиму OFB. Як лічильник може використовуватися будь-який генератор псевдовипадкових чисел, незалежно від його крипостійкості.

При використанні потокового шифру в режимі лічильника вибирається проста функція переходу і складна, залежна від ключа, функція виходу. Функція переходу може бути простим лічильником, який збільшується на одиницю на кожному такті.

***Алгоритм потокового шифру RC4***

RC4 є потоковим шифром зі змінною довжиною ключа. Він розроблений у 1987 р. Роном Рівестом для компанії RSA Data Security, Inc. Упродовж семи років цей шифр ліцензіювався компанією тільки на умовах нерозголошення. Проте в 1994 р. він був анонімно опублікований в Інтернеті й відтоді став доступним для незалежного аналізу.

Описується шифр досить просто. Алгоритм працює в режимі OFB. Ключова послідовність не залежить від вихідного тексту. Структура алгоритму містить блок заміни розміром 8×8: $S_0, \ldots, S_{255}$. Блок заміни є залежною від ключа змінної довжини перестановкою чисел 0, ..., 255. Алгоритм містить два лічильники – $i$ та $j$, які початково дорівнюють 0. Для генерування псевдовипадкового байта виконуються такі дії:

$$i = (i + 1) \bmod 256,$$

$$j = (j + S_i) \bmod 256.$$

Далі – переставити $S_i$ і $S_j$

$$t = (S_i + S_j) \bmod 256$$

$$k = S_t.$$

Потім байт $k$ додається за модулем 2 з байтом вихідного тексту для отримання шифрованого.

Ініціалізація блока заміни також проста. Спочатку він заповнюється лінійно: $S_0 = 0, S_1 = 1, \ldots S_{255} = 255$. Потім ключем заповнюється ще один 256-байтний масив, при цьому ключ може повторюватися необхідну кількість разів для заповнення всього масиву: $k_0, \ldots, k_{255}$. Лічильник $j$ встановлюється в 0, після чого проводяться такі дії:

for i=0 to 255,

$j=(j+k_i+S_i) \bmod 256$,

переставити $S_i$ і $S_j$.

Шифрування за цим алгоритмом приблизно в 10 разів швидше, ніж шифрування DES за програмної реалізації.

Можливе узагальнення алгоритму для більшої довжини слова і розміру блока заміни. Так, можна побудувати шифр із блоком заміни розміром 16×16 (для цього потрібно 128 Кбайт пам'яті) і довжиною слова 16 бітів.

# Розділ IV
# Асиметричні криптосистеми

Якими би надійними не були симетричні криптоалгоритми, слабким місцем їх практичної реалізації залишається проблема розподілу криптографічних ключів. Для безпечного обміну інформацією між двома суб'єктами, один з них повинен згенерувати ключ та в якийсь спосіб конфіденційно передати іншому. Отже, для розповсюдження криптографічного ключа необхідно використати або існуючу криптосистему, або захищений інформаційний канал.

Для розв'язання цієї проблеми на основі нових результатів сучасної алгебри було запропоновано системи з відкритим ключем (***асиметричні криптосистеми***).

Суть таких систем, як уже зазначалося, полягає в тому, що кожним суб'єктом інформаційного обміну генеруються два ключі, зв'язані між собою певними правилами. Один ключ оголошується ***публічним*** (***відкритим***), а інший – ***приватним*** (***секретним***). Публічний ключ розміщується на доступному всім ресурсі (публікується). Приватний ключ зберігається суб'єктом, який його створив, і недоступний для інших суб'єктів.

Відкритий текст зашифровується на публічному ключі та передається адресатові. Зашифрований текст принципово не може бути розшифрований на публічному ключі. Розшифрування повідомлення можливе лише на відповідному приватному ключі, відомому тільки безпосередньо адресату.

Асиметричні криптосистеми, як уже зазначалося, використовують так звані односторонні функції, які мають таку властивість: при заданому значенні $x$ відносно легко обчислити значення $f(x)$, однак якщо відомо значення $y = f(x)$, то не існує простого способу обчислення значення $x$. Велика кількість класів незворотних функцій і породжує всю різноманітність криптосистем із відкритим ключем. Однак у самому означенні є деяка невизначеність: що означає «не існує простого способу»?

Тому до асиметричних криптосистем ставляться дві важливих вимоги:

- перетворення відкритого тексту повинно бути незворотним без можливості його відновлення на публічному ключі;
- обчислення приватного ключа на основі публічного також має бути неможливим на сучасному технологічному рівні. При цьому бажана точна нижня оцінка трудомісткості розкриття шифру.

Алгоритми шифрування з відкритим ключем використовують у трьох напрямках:

1) як самостійні засоби захисту інформації;
2) як засоби аутентифікації користувачів;
3) як засоби розповсюдження ключів.

Як відомо, асиметричні криптоалгоритми значно повільніші за симетричні. Тому часто на практиці доцільно використати перші для шифрування невеликої

кількості інформації, а потім за допомогою симетричних алгоритмів виконувати шифрування великих інформаційних потоків.

Для кращого розуміння принципів функціонування асиметричних криптосистем розглянемо необхідні елементи теорії чисел [10,16-17].

## Елементи теорії чисел

Нехай $n$ – довільне натуральне число, $x$ та $y$ – цілі числа. Будемо називати числа $x$ та $y$ ***конгруентними за модулем n***, якщо залишки від їх ділення на число $n$ однакові, тобто $x \bmod n \equiv y \bmod n$. Наприклад, $2 \bmod 7 \equiv 9 \bmod 7$ або $11 \bmod 8 \equiv 33 \bmod 30$, оскільки залишок від ділення у цих чисел однаковий.

Операція взяття числа за модулем має три основні властивості:

- адитивності: $(a+b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$;
- мультиплікативності: $(a \times b) \bmod n = ((a \bmod n) \times (b \bmod n)) \bmod n$;
- збереження степеня: $(a \bmod n)^k \bmod n = a^k \bmod n$.

***Найбільший спільний дільник*** (НСД) чисел $A$ і $B$ – це найбільше з чисел, на яке обидва цих числа діляться без залишку. Наприклад, НСД(56,98)=14; НСД(76,190)=38; НСД(150,19)=1.

***Взаємно прості числа*** – це такі числа $A$ та $B$, які самі по собі, можливо, не є простими, але не мають іншого спільного дільника, на який вони діляться без залишку, крім 1. Наприклад, числа 32 і 13 – взаємно прості, хоча 32 саме по собі не просте, оскільки ділиться ще на 2, 4, 8, 16.

***Лишок числа A за модулем n*** – цілий залишок від ділення числа $A$ на число $n$.

***Набір лишків числа A за модулем n*** – це сукупність усіх різних цілих залишків від ділення числа $A \times i$ на число $n$, де $i$ набирає значення від 1 до $n$-1. Наприклад, набір лишків числа 3 за модулем 5 буде таким: {3,1,4,2}. Зауважимо, що довжина вектора лишків максимальна й дорівнює $n$-1, якщо числа $A$ та $n$ взаємно прості.

***Обчислення мультиплікативного оберненого числа***. Мультиплікативним оберненим числом до числа $A$ за модулем $n$ будемо називати таке число $A^{-1}$, що $(AA^{-1}) \bmod n = 1$. Зазначимо, що розв'язок такої задачі існує не завжди, а лише тоді, коли A та $n$ – взаємно прості числа. Операція знаходження мультиплікативного оберненого дуже часто використовується у двоключовій криптографії.

Мультиплікативне обернене число можна знайти (для невеликих чисел) безпосередньо з розв'язку рівняння $(AA^{-1}) \bmod n = 1$. Це рівняння можна записати так: $AA^{-1} - 1 = i \times n$, де $i$=1, 2, 3,... – натуральне число. Тоді $AA^{-1} = 1 + i \times n$, звідки $A^{-1} = (1 + i \times n)/A$.

Наприклад, треба знайти мультиплікативне обернене числа 32 за модулем 29. Для цього отримаємо: $A^{-1} = (1 + 11 \times 29)/32 = 10$.

Класичним алгоритмом знаходження мультиплікативного оберненого є розв'язок еквівалентного рівняння *Діофанта* за допомогою методики *Евкліда*.

Вираз $32\times32^{-1} \bmod 29 = 1$ еквівалентний діофантовому рівнянню $32x+29y = 1$. Застосуємо до нього методику Евкліда. Суть її полягає ось у чому. Нехай є рівняння типу $ax+by=1$, де $a$ та $b$ – взаємно прості. Представимо:

$$\frac{a}{b} = \frac{P_K}{Q_K},$$

причому $P_{-2} = 0, P_{-1} = 1, Q_{-2} = 1, Q_{-1} = 0$.

Для довільних $n \geq 0$ $P_n = q_n P_{n-1} + P_{n-2}$; $Q_n = q_n Q_{n-1} + Q_{n-2}$.

Для обчислення наступних членів P і Q складемо таблицю:

| $n$ | -2 | -1 | 0 | 1 | 2 | … | $k$-1 | $k$ |
|---|---|---|---|---|---|---|---|---|
| $q_n$ | | | $q_0$ | $q_1$ | $q_2$ | … | $q_{k-1}$ | $q_k$ |
| $P_n$ | 0 | 1 | $P_0$ | $P_1$ | $P_2$ | … | $P_{k-1}$ | $P_k$ |
| $Q_n$ | 1 | 0 | $Q_0$ | $Q_1$ | $Q_2$ | … | $Q_{k-1}$ | $Q_k$ |

Тоді для обчислення $x$ та $y$ використовуються формули

$$x = (-1)^{k-1}Q_{k-1}; y = (-1)^{k-1}P_{k-1}, \text{ причому } P_k = a, Q_k = b.$$

Для обчислення значень таблички використаємо ланцюжкові дроби. Отже, для нашого прикладу отримаємо:

$32=29\times1+3$, звідси $q_0 = 1$;

$29=3\times9+2$, звідси $q_1 = 9$;

$3 = 2\times1+1$, звідси $q_2 = 1$;

$2 = 1\times2+0$, звідси $q_3 = 2$.

Тоді таблиця набуває вигляду:

| $n$ | -2 | -1 | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|---|---|
| $q_n$ | | | 1 | 9 | 1 | 2 | |
| $P_n$ | 0 | 1 | 1 | 10 | 11 | 32 | $P_k=a$ |
| $Q_n$ | 1 | 0 | 1 | 9 | 10 | 29 | $Q_k=b$ |

$$x = (-1)^{k-1}Q_{k-1} = (-1)^2 \times 10 = 10;$$

$$y = (-1)^{k-1}P_{k-1} = (-1)^2 \times 11 = 11.$$

Таким чином, ми отримали, $x = 10$, тобто $(32)^{-1} \bmod 29 = 10$.

Перевірка дає $10\times32 \bmod 29 = 320 \bmod 29 = 1$.

Уведемо ***функцію Ейлера*** $\varphi(n)$, яка визначає кількість цілих чисел, взаємно простих із $n$, із множини [1, $n$-1]. Доведено, що для простого $n$ $\varphi(n)= n-1$. Продемонструємо це для множини [2,11]:

| $n$ | 2 | **3** | 4 | **5** | 6 | **7** | 8 | 9 | 10 | **11** |
|---|---|---|---|---|---|---|---|---|---|---|
| $\varphi(n)$ | 1 | **2** | 2 | **4** | 2 | **6** | 4 | 6 | 4 | **10** |

Як бачимо з таблички, для чисел 3, 5, 7, 11 $\varphi(n) = n-1$.

***Теорема 1. Мала теорема Ферма***. Якщо $n$ – просте число, то $(x^{n-1} \bmod n) = 1$ для будь-яких $x$, взаємно простих із $n$. Ілюстрацію цієї теореми див. далі.

***Теорема 2***. Нехай число $n$ – просте. Для будь-якого $A$ та $1 \le B \le (n-1)$ знайдеться таке $1 \le X \le (n-1)$, що $A^x \bmod n = B$. Іншими словами, стверджується, що функція $A^x \bmod n = B$ однозначна на проміжку $0 \ldots n-1$.

Виняток складають лише степені числа 2 (тобто $A^2, A^4, \ldots$), які не утворюють однозначного перетворення.

Для прикладу розглянемо функцію $y = a^x \bmod n$, $n = 7$. У таблиці подано всі можливі значення $y$ для $a$ та $x$ від 0 до 6. Як бачимо, тут є кілька варіантів. Для $a = 1$ усі значення $y = 1$. Для $a = 2, 4$ та 6 бачимо багатозначне відображення $\{x\} \to \{y\}$ ($y$ – періодичне). Водночас для $a = 3$ і $a = 5$ існує взаємно однозначне відображення $\{x\} \to \{y\}$. Це означає, що ці числа, 3 та 5, ми можемо використати для практичних потреб. Такі числа називаються ***первісними коренями*** за модулем $n$.

| $a \backslash x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | | | | | | | |
| 1 | | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | | 2 | 4 | 1 | 2 | 4 | 1 |
| 3 | | 3 | 2 | 6 | 4 | 5 | 1 |
| 4 | | 4 | 2 | 1 | 4 | 2 | 1 |
| 5 | | 5 | 4 | 6 | 2 | 3 | 1 |
| 6 | | 6 | 1 | 6 | 1 | 6 | 1 |

Зауважимо, що 3 та 5 взаємно прості з 7. Крім того, 5 – взаємно просте число з 6, тобто з $n$-1. Ця табличка також ілюструє *малу теорему Ферма*: $a^6 \bmod 7 = 1$ (тобто $a^{n-1} \bmod n = 1$).

Зрозуміло, що у випадку таких малих чисел, які подано в табличці, розв'язати обернену задачу знаходження числа $x$, якщо відомо $y$ та $n$, дуже просто. Однак для великих цілих чисел розв'язок оберненої задачі (яка називається задачею дискретного логарифмування в кінцевому полі) стає обчислювально складним. Така функція називається ***односторонньою***. Аналогічна одностороння функція використовується у криптосистемі Ель-Гамаля.

Іншим представником односторонніх функцій є розкладання великого цілого числа на прості множники. Обчислити добуток двох простих чисел дуже просто.

Однак для розв'язку оберненої задачі, розкладання заданого числа на прості множники, ефективного алгоритму сьогодні не існує. Така одностороння функція використовується у криптосистемі RSA.

В усіх двоключових криптосистемах застосовують так звані ***односторонні функції з пасткою***. Це означає, що публічний ключ криптосистеми визначає конкретну реалізацію функції, а приватний ключ дає певну інформацію про пастку. Той, хто знає приватний ключ, може легко обчислювати функцію в обох напрямках, але той, хто не має такої інформації, може обчислювати функцію лише в одному напрямку. Прямий напрямок використовується для зашифрування інформації та верифікації цифрового підпису. Зворотний напрямок застосовують для розшифрування та створення електронного цифрового підпису.

В усіх криптосистемах із відкритим ключем чим більша довжина ключів, тим більша різниця в зусиллях, необхідних для обчислення функції у прямому та зворотному напрямках (для тих, хто не має інформації про пастку).

Усі практичні криптосистеми з відкритим ключем використовують функції, які вважаються односторонніми, однак цю властивість не доведено для жодної з них. Це означає, що теоретично можливе створення алгоритму, який дозволить легко обчислити обернену функцію без знання інформації про пастку. Однак так само ймовірне теоретичне доведення неможливості такого обчислення. В першому випадку це призведе до повного краху відповідної криптосистеми, а у другому – значно зміцнить її позиції.

## Криптосистема RSA

Незважаючи на досить велику кількість різних асиметричних криптосистем, широке практичне застосування отримала RSA, назва якої походить від перших літер прізвищ її авторів, Рональда Рівеста (Ron Rivest), Аді Шаміра (Adi Shamir) та Лео Аделмана (Leonard Addleman).

Вони використали той факт, що обчислення добутку двох великих простих чисел значно простіше, ніж розкладання великого цілого числа на прості множники. Доведено (теорема Рабіна), що розкриття криптосистеми RSA еквівалентне такому розкладанню. Звідси випливає, що можна дати нижню оцінку кількості операцій, необхідних для розкриття шифру, а враховуючи продуктивність сучасних комп'ютерів, обчислити потрібний для цього час.

Звичайно, наявність гарантованої оцінки криптостійкості алгоритму створила сприятливі умови для розповсюдження криптосистеми RSA, на відміну від інших алгоритмів. Це й стало однією з причин її популярності в банківських системах для роботи з віддаленими клієнтами.

Розглянемо алгоритм шифрування за схемою RSA [15]. Звичайно, в навчальних цілях ми будемо використовувати приклад із малими числами. Для реальної роботи ці числа не підходять, оскільки криптостійкість такої системи практично дорівнює нулю.

Отже, для обміну інформацією, зашифрованою за допомогою криптосистеми RSA, необхідно виконати такі кроки.

***Крок 1***. *Підготовчі обчислення*. Отримувач генерує два великих простих числа $p$ і $q$ (мінімум 128-бітних [посилання]). Для нашого прикладу візьмемо $p = 7, q = 11$. Знайдемо добуток, ***модуль*** криптосистеми, $n = p \times q = 77$. Далі необхідно обчислити функцію Ейлера для цього модуля. Ми знаємо, що для простих чисел $\varphi(n) = (p\text{-}1)(q\text{-}1)$. Отже, $\varphi(77) = 6 \times 10 = 60$. Тепер необхідно згенерувати ціле число, взаємно просте як із $n$, так і з $\varphi(n)$, наприклад, $e = 13$.

Пара чисел $(e, n)$ служитиме ***публічним ключем*** криптосистеми. У нашому випадку це буде пара (13, 77).

Тепер необхідно обчислити ***приватний ключ*** $d$, парний до обраного публічного. Для цього треба розв'язати рівняння $(d \times e) \bmod \varphi(n) = 1$. Відповідно до обчислення мультиплікативного оберненого, будемо мати: $d = (1 + k \times \varphi(n))/e$. Для $k = 8$ отримуємо $d = 37$. Отже, приватним ключем служитиме пара чисел $(d,n)$ – (37,77).

***Крок 2***. *Розповсюдження ключів*. Для шифрування інформації використовують публічний ключ (хоча можна використовувати і приватний). Для використання його розміщують на ресурсі, до якого мають доступ усі учасники інформаційного обміну. Зауважимо, що одним публічним ключем можуть користуватися всі, хто бажає обмінюватися з отримувачем зашифрованою інформацією. На відміну від симетричних криптосистем, тут немає необхідності для кожної пари учасників генерувати окрему пару ключів, адже розшифрувати інформацію за допомогою публічного ключа неможливо (принаймні, обчислювально складно). Отже, конфіденційність обміну інформації гарантується самим принципом обробки інформації. Єдине, що необхідно зробити, це захистити публічний ключ від підміни (про атаки на асиметричні криптосистеми дивись далі). Найпростіше, що можна зробити, це захистити каталог, де знаходяться ключі, від запису, однак найнадійнішим способом вважається сертифікація публічних ключів. Кожен, хто бажає захистити свій публічний ключ від підміни, повинен отримати сертифікат довірчого центру інфраструктури відкритих ключів, який прив'язує ключ до його власника.

Приватний ключ не розповсюджується. Він використовується для розшифрування інформації та створення ***електронного цифрового підпису*** і повинен бути відомим лише його власникові.

На цьому підготовчі операції закінчено, і можна починати обмін захищеною інформацією.

***Крок 3***. *Шифрування інформації*. Криптосистемою RSA можна зашифрувати числа (коди літер) у діапазоні від 0 до $n$. Відправник повідомлення, використовуючи публічний ключ $(e,n)$, у нашому випадку – (13,77), за допомогою формули $C_i = (M_i)^e \bmod n$ зашифровує своє повідомлення, де $M_i$ –

числове представлення чергової літери повідомлення, $C_i$ – черговий символ криптограми. Наприклад, слово «БАНК» (яке має числове зображення «02 01 17 14» за таблицею заміни українського алфавіту, яка починається з 01) зашифрується так:

$$C_1 = 2^{13} \bmod 77 = 30;$$
$$C_2 = 1^{13} \bmod 77 = 1;$$
$$C_3 = 17^{13} \bmod 77 = 73;$$
$$C_4 = 14^{13} \bmod 77 = 49.$$

Отже, криптограма матиме вигляд: «30 01 73 49». Очевидно, що шифр у нашому прикладі є шифром заміни. Як бачимо, літера «А», яка має код «01», не змінилася. Таку ж властивість мають «0» та $n$-1. Отже, не всі числа доцільно вибирати як коди літер. Вважається правильним надавати для шифрування числа з діапазону [2, $n$-2]. Це дещо ускладнює розкриття шифру [18].

Зашифрований текст пересилається отримувачу відкритими каналами зв'язку.

Із наведеного способу шифрування може скластися враження, що піднесення великого цілого числа до великого степеня та взяття залишку за модулем третього великого числа є надзвичайно складною математичною задачею. Однак насправді це зовсім не так. Для ілюстрації цього наведемо алгоритм обчислення виразу $432^{678} \bmod 987$ [11].

Число 678 можна зобразити так: 678=512+128+32+4+2. Тоді $432^{678} \bmod 987 = (432^{512} \times 432^{128} \times 432^{32} \times 432^{4} \times 432^{2}) \bmod 987$. Використовуючи основні властивості, наведені вище, можемо записати:

$$432^{678} \bmod 987 = ((432^{2} \bmod 987) \times (432^{4} \bmod 987) \times (432^{32} \bmod 987) \times$$
$$\times (432^{128} \bmod 987) \times (432^{512} \bmod 987)) \bmod 987.$$

Тепер обчислимо степені числа 432:

$$432^{2} \bmod 987 = 81;$$
$$432^{4} \bmod 987 = 81^{2} \bmod 987 = 639;$$
$$432^{8} \bmod 987 = 639^{2} \bmod 987 = 690;$$
$$432^{16} \bmod 987 = 690^{2} \bmod 987 = 366;$$
$$432^{32} \bmod 987 = 366^{2} \bmod 987 = 711;$$
$$432^{64} \bmod 987 = 711^{2} \bmod 987 = 177;$$
$$432^{128} \bmod 987 = 177^{2} \bmod 987 = 732;$$
$$432^{256} \bmod 987 = 732^{2} \bmod 987 = 870;$$
$$432^{512} \bmod 987 = 870^{2} \bmod 987 = 858.$$

Підставляючи в наш вираз ці значення, отримаємо: $(81 \times 639 \times 711 \times 732 \times 858) \bmod 987 = 204$.

***Крок 4***. *Розшифрування інформації*. Отримувач розшифровує зашифроване повідомлення «30 01 73 49», використавши тільки йому відомий приватний

ключ $(d,n)$ та формулу $M_i = (C_i)^d \bmod n$. Доведемо принципову можливість розшифрування зашифрованої на публічному ключі інформації:

$$(C)^d \bmod n = (M^e \bmod n)^d \bmod n = (M^e)^d \bmod n =$$
$$= (M^{ed}) \bmod n = (M^{1+k\varphi(n)}) \bmod n = M(M^{\varphi(n)}) \bmod n = M.$$

Таким чином, операція зашифрування на публічному та розшифрування на приватному ключі – взаємно обернені.

У нашому випадку приватним ключем служить пара (37,77). Тоді отримаємо:

$$M_1 = 30^{37} \bmod 77 = 2;$$
$$M_2 = 1^{37} \bmod 77 = 1;$$
$$M_3 = 73^{37} \bmod 77 = 17;$$
$$M_4 = 49^{37} \bmod 77 = 14.$$

Маючи таблицю заміни, за кодами літер отримаємо «БАНК». Отже, ми з'ясували методи зашифрування та розшифрування інформації у криптосистемі RSA. Залишилося виявнити один цікавий факт, характерний для цієї криптосистеми.

Спробуємо використати для зашифрування приватний ключ. Нехай повідомлення для зашифрування те ж саме: «БАНК» або «02 01 17 14». Зашифруємо її на приватному ключі (37,77). Отримаємо таке:

$$C_1 = 2^{37} \bmod 77 = 51;$$
$$C_2 = 1^{37} \bmod 77 = 1;$$
$$C_3 = 17^{37} \bmod 77 = 52;$$
$$C_4 = 14^{13} \bmod 77 = 42.$$

Тепер розшифруємо зашифроване повідомлення «51 01 52 42» на публічному ключі (13, 77):

$$M_1 = 51^{13} \bmod 77 = 2;$$
$$M_2 = 1^{13} \bmod 77 = 1;$$
$$M_3 = 52^{13} \bmod 77 = 17;$$
$$M_4 = 42^{13} \bmod 77 = 14.$$

Як бачимо, ми отримали відкрите повідомлення «БАНК» і таким методом. Отже, криптосистема RSA симетрична відносно застосування парних ключів: можна зашифровувати інформацію на публічному ключі та розшифровувати на приватному і, навпаки, зашифровувати на приватному, а розшифровувати – на парному до нього публічному ключі.

Чим цікава така особливість? Оскільки приватний ключ знає лише його власник, то отримувач зашифрованої на приватному ключі інформації може бути впевненим у тому, що її отримано саме від власника приватного ключа, і він згоден з цією інформацією. Отже, маємо ***аналог підписаного автором документа***. До цієї проблеми ми повернемось у наступних розділах.

ключ $(d,n)$ та формулу $M_i = (C_i)^d \bmod n$. Доведемо принципову можливість розшифрування зашифрованої на публічному ключі інформації:

$$(C)^d \bmod n = (M^e \bmod n)^d \bmod n = (M^e)^d \bmod n =$$
$$= (M^{ed}) \bmod n = (M^{1+k\varphi(n)}) \bmod n = M(M^{\varphi(n)}) \bmod n = M.$$

Таким чином, операція зашифрування на публічному та розшифрування на приватному ключі – взаємно обернені.

У нашому випадку приватним ключем служить пара (37,77). Тоді отримаємо:

$$M_1 = 30^{37} \bmod 77 = 2;$$
$$M_2 = 1^{37} \bmod 77 = 1;$$
$$M_3 = 73^{37} \bmod 77 = 17;$$
$$M_4 = 49^{37} \bmod 77 = 14.$$

Маючи таблицю заміни, за кодами літер отримаємо «БАНК». Отже, ми з'ясували методи зашифрування та розшифрування інформації у криптосистемі RSA. Залишилося вияснити один цікавий факт, характерний для цієї криптосистеми.

Спробуємо використати для зашифрування приватний ключ. Нехай повідомлення для зашифрування те ж саме: «БАНК» або «02 01 17 14». Зашифруємо її на приватному ключі (37,77). Отримаємо таке:

$$C_1 = 2^{37} \bmod 77 = 51;$$
$$C_2 = 1^{37} \bmod 77 = 1;$$
$$C_3 = 17^{37} \bmod 77 = 52;$$
$$C_4 = 14^{13} \bmod 77 = 42.$$

Тепер розшифруємо зашифроване повідомлення «51 01 52 42» на публічному ключі (13, 77):

$$M_1 = 51^{13} \bmod 77 = 2;$$
$$M_2 = 1^{13} \bmod 77 = 1;$$
$$M_3 = 52^{13} \bmod 77 = 17;$$
$$M_4 = 42^{13} \bmod 77 = 14.$$

Як бачимо, ми отримали відкрите повідомлення «БАНК» і таким методом. Отже, криптосистема RSA симетрична відносно застосування парних ключів: можна зашифровувати інформацію на публічному ключі та розшифровувати на приватному і, навпаки, зашифровувати на приватному, а розшифровувати – на парному до нього публічному ключі.

Чим цікава така особливість? Оскільки приватний ключ знає лише його власник, то отримувач зашифрованої на приватному ключі інформації може бути впевненим у тому, що її отримано саме від власника приватного ключа, і він згоден з цією інформацією. Отже, маємо ***аналог підписаного автором документа***. До цієї проблеми ми повернемось у наступних розділах.

**Криптостійкість RSA**

Припустимо, що інформацію, яку перехопив зловмисник, зашифровано на публічному ключі (13,77). Якою інформацією володіє в такому разі зловмисник? По-перше, він має криптограму «30 01 73 49»; по-друге, має публічний ключ (13,77). Які зусилля треба йому докласти для обчислення відкритого тексту? Як уже згадувалося, задача розшифрування для RSA еквівалентна задачі розкладання великого числа (а у нашому випадку – малого числа 77) на прості множники. У монографії [14] виконано розрахунок MIPS років (1 MIPS рік = 1 млн. інструкцій за секунду протягом 1 року = $3{,}1\times10^{13}$ інструкцій), необхідних для розкладання великих чисел на прості множники. Ці дані можна побачити в таблиці.

| Кількість розрядів $n$ | Значення функції $L(n)$ | Кількість MIPS років |
|---|---|---|
| 512 | $6{,}7\times10^{19}$ | $2{,}1\times10^{6}$ |
| 576 | $1{,}7\times10^{21}$ | $5{,}5\times10^{7}$ |
| 960 | $3{,}7\times10^{28}$ | $1{,}2\times10^{15}$ |
| 1024 | $4{,}4\times10^{29}$ | $1{,}4\times10^{16}$ |

Тут як функцію $L(n)$, яка задає апроксимацію швидкості найкращого на сьогодні алгоритму розкладання чисел на прості множники, методу решета числового поля, взято:

$$L(n) = \exp(1+\varepsilon)\sqrt{(\ln n)(\ln\ln n)^2}\ ,$$

де $n$ – кількість двійкових розрядів у числі; $\varepsilon$ – мала величина.

У кінці 1995 року лише єдиний раз вдалося практично реалізувати розкриття шифру RSA для 500-бітного ключа. Для цього за допомогою Інтернет були задіяні 1600 комп'ютерів упродовж 5 місяців неперервної роботи. Тому автори RSA рекомендують використовувати таку довжину модуля $n$ [7]:

- 768 бітів – для приватних осіб;
- 1024 біти – для комерційної інформації;
- 2048 бітів – для особливо таємної інформації.

Наведемо також таблицю порівняння криптостійкості симетричних та асиметричних криптосистем. У таблиці вказано, за яких довжин ключів досягається приблизно однакова стійкість симетричних та асиметричних систем до методу суцільного перебору ключів (метод «грубої сили»).

| Довжина ключа симетричної криптосистеми, біти | Довжина відкритого ключа асиметричної криптосистеми, біти |
|---|---|
| 56 | 384 |
| 64 | 512 |
| 80 | 768 |
| 112 | 1792 |
| 128 | 2304 |

Як бачимо, для досягнення однакової стійкості асиметричні криптосистеми використовують значно довший ключ (від 7 до 18 разів). Зрозуміло, що такий довгий ключ значно зменшує швидкодію асиметричних алгоритмів.

І, нарешті, визначимо, які типи інформації вимагають більшої криптостійкості, а отже, й більшої довжини ключів [14]:

| Тип інформації | Час життя | Довжина ключа, біти |
|---|---|---|
| Тактична військова інформація | хв/год | 56-64 |
| Оголошення про нову продукцію, злиття компаній | дні/тижні | 64 |
| Довготривалі бізнес-плани | роки | 64 |
| Торговельні секрети (наприклад, рецептура) | 10-річчя | 112 |
| Секрети водневої бомби | >40 років | 128 |
| Особи шпигунів | >50 років | 128 |
| Дипломатичні конфлікти | >60 років | 128 |
| Дані перепису населення | >100 років | >128 |

Із таблиці видно, що з огляду на терміни зберігання інформації різних типів таємності, довжини ключів асиметричних криптосистем у 2048 бітів не видаються занадто параноїдальними.

## Криптосистема Ель-Гамаля

Стійкість криптосистеми Ель-Гамаля, розробленої у 1985 році, ґрунтується на складності задачі дискретного логарифмування у скінченному полі [10,15,19].

Для встановлення зашифрованого інформаційного обміну необхідно виконати такі кроки.

***Крок 1***. *Попередні обчислення*. За допомогою криптографічно стійкого генератора випадкових чисел генерують просте число $n$, таке, що обчислення логарифму за mod $n$ практично важко реалізувати.

Обирають також випадкові числа $g$ та $a$ з діапазону $[1, n\text{-}1]$ та обчислюють $h = g^a \bmod n$.

Маємо публічний $(n, g, h)$ і приватний ключі $(n, a)$.

***Крок 2***. *Шифрування інформації*. Зашифровують числа $m$ від 0 до $n$. Для шифрування виконують таке:

- Обирають випадкове число $r$, яке належить інтервалу $[1, n\text{-}1]$ та взаємно просте з $n$-1.
- Обчислюють пару чисел $C_1$ і $C_2$ за формулами $C_1 = g^r \bmod n$; $C_2 = mh^r \bmod n$.

Пара чисел $C_1$ і $C_2$ утворює шифрограму для числа $m$.

***Крок 3***. *Розшифрування інформації*. Розшифрування виконується за формулою $m = C_2(C_1^a)^{-1} \bmod n$. Доведемо це. Підставимо сюди значення $C_1$ і $C_2$: $m = mh^r(g^a)^{-1} \bmod n$. Оскільки $h = g^a \bmod n$, то:

$$m = mh^r(g^a)^{-1} \bmod n = mh^r(h^r)^{-1} \bmod n = m.$$

Отже, ми довели еквівалентність прямого та оберненого перетворення.

*Розглянемо приклад*. Нехай $n = 29$, $g = 2$, $a = 5$.

Обчислимо $h = 2^5 \bmod 29 = 3$.

Таким чином, ***публічний ключ*** – (29, 2, 3), ***приватний ключ*** – (29, 5).

Нехай повідомлення $m = 11$. Оберемо $r = 8$ та обчислимо $C_1 = 2^8 \bmod 29 = 24$ і $C_2 = (11 \times 3^8) \bmod 29 = 19$. Отже, в результаті обчислень маємо криптограму $C = (24,19)$.

Розшифруємо отриману криптограму: $D(C) = 19 \times (24^5)^{-1} \bmod 29 = 11 \equiv m$.

**Криптостійкість системи Ель-Гамаля**

Як правило, використовують модуль $n$ криптосистеми довжиною 1024 біти, $g$ – порядку 160 бітів.

Безпосередня атака на систему Ель-Гамаля, атака обчислення приватного ключа за публічним, потребує обчислення дискретного логарифму, що для таких великих чисел, як $n$ і $g$, перетворюється в математичну задачу надзвичайної обчислювальної складності.

Однак імовірна вразливість криптосистеми Ель-Гамаля полягає в тому, що саме повідомлення міститься лише у $C_2$. Тому теоретично можливою видається атака, коли, помноживши $C_2$ на $g^u$ ($u \neq 0$), ми отримаємо шифротекст для повідомлення $m = mg^u$.

Удосконалений алгоритм Ель-Гамаля, відомий під назвою Ель-Гамаля-Шнорра, використовується для цифрового підпису DSA та російського стандарту цифрового підпису ГОСТ Р 34.10-94, про які піде мова в наступних розділах.

# Розділ V
# Електронний цифровий підпис

У кінці звичайного листа або документа його автор або відповідальна особа звичайно ставлять свій підпис. Така дія має на меті таке:

- по-перше, отримувач може впевнитися в справжності документа, порівнявши підпис зі зразком;
- по-друге, особистий підпис є юридичною гарантією авторства документа.

Останній аспект особливо важливий під час укладання різного роду комерційних угод, створення доручень та інших документів, коли сторони можуть не довіряти одна одній.

Підробити підпис людини на папері досить непросто, і зловмисник повинен мати дуже серйозні аргументи для виконання такої операції, бо встановити авторство підпису сучасними криміналістичними методами – справа техніки.

А як бути у випадку, коли вигідніше використати електронний документообіг і не витрачати величезні кошти та час на неодноразові відрядження?

Для цього розроблено зовсім новий криптографічний механізм, який став можливим лише після винайдення асиметричної криптографії. Цей механізм У. Діффі та М. Хеллман назвали *цифровим підписом*. Його суть пояснимо на прикладі системи RSA. До повідомлення $M$ застосуємо перетворення за допомогою приватного ключа $d$ і назвемо його *цифровим підписом*, тобто

$$S = M^d \bmod n.$$

Повідомлення $M$ і його електронний підпис $S$ відправляють за призначенням.

Отримувач, маючи $(M, S)$ та відкритий ключ відправника повідомлення $e$, може перевірити виконання співвідношення

$$S^e \bmod n = M.$$

Якщо обчислене $M$ збігається з отриманим повідомленням, то підпис справжній.

Така схема приводить до такого:

- отримувач, перевіривши справжність підпису, впевнений у тому, що це повідомлення $M$ сформував саме власник приватного ключа $d$ (оскільки більше ніхто не має до нього доступу);
- відправник не зможе відмовитися від цього листа з тієї ж самої причини. Отже, існує можливість створення юридично чинних документів на основі такого механізму електронного підписування.

Ця схема має суттєвий недолік: цифровий підпис має ту ж довжину, що й документ, ним підписаний. Отже, каналом зв'язку пересилається вдвічі більше інформації, ніж це потрібно для самого документа.

Таким чином, підписання довгих повідомлень потребує видозміни схеми. Для досягнення цієї мети було запропоновано застосовувати до повідомлень криптографічне перетворення, назване хешуванням.

**Функції хешування**

*Хеш-функцією назвемо криптографічне перетворення двійкової послідовності довільної довжини у двійкову послідовність фіксованої довжини.*

Для використання в криптографії хеш-функції повинні задовольняти такі властивості:

1) перетворення має бути одностороннім, тобто за хеш-образом $h(M)$ неможливо (принаймні обчислювально складно) визначити повідомлення $M$;

2) для фіксованого $M$ складно знайти $M_1 \neq M$, щоби $h(M_1) = h(M)$.

Додатковою бажаною властивістю є складність визначення довільних $M_1$ і $M_2$, для яких $h(M_1) = h(M_2)$. Таку властивість називають відсутністю колізій (безколізійністю). Зауважимо, що для довгих повідомлень завжди існують $M_1$ і $M_2$, такі що $h(M_1) = h(M_2)$ унаслідок фіксованої довжини значення хеш-функції. Мова йде лише про складність їх знаходження.

У випадку використання хеш-функцій схема цифрового підпису така. Спершу хешують повідомлення, тобто обчислюють $h(M)$. Потім до отриманого хеш-образу застосовують асиметричне криптографічне перетворення за допомогою приватного ключа. Для RSA – це

$$S = [h(M)]^d \bmod n.$$

Якщо учасники інформаційного обміну завчасно домовилися про алгоритм хешування, то відправник передає каналом зв'язку $M$ і $S$. На приймальній стороні, отримавши підписане повідомлення, перевіряють підпис так:

$$S^e \bmod n = h(M).$$

Одночасно на приймальній стороні обчислюють хеш-образ отриманого повідомлення $h'(M)$ і, якщо $h(M) = h'(M)$, то отримувач впевнений:

- повідомлення $M$ не зазнало жодних змін під час передавання його каналами зв'язку;
- хеш-образ $h(M)$ відповідає повідомленню $M$ і також не змінювався під час «мандрів» каналами зв'язку;
- повідомлення написане власником приватного ключа $d$, він згоден із його змістом;
- відправник не зможе відмовитися від факту існування такого документа, тобто він, цей документ, набуває юридичної чинності.

Кінцева довжина хеш-образів (а їх довжина коливається від 128 до 256 бітів) не загромаджує канали зв'язку, одночасно роблячи неможливою (принаймні обчислювально складною) підробку захищених ними повідомлень.

Наголосимо, що дотримання перших двох умов є обов'язковим чинником правильності використання хеш-функції.

Розглянемо, до чого може призвести недотримання цих умов.

Припустимо, що сторона *А* укладає угоду зі стороною *Б*. Сторона *А* підготувала два проекти угоди, одну «правильну», а другу «неправильну», тобто таку, що зумовлює банкрутство сторони *Б*.

Сторона *А* вносить в обидві версії документів незначні зміни (наприклад, ПРОБІЛ заміняється комбінацією ПРОБІЛ-BACKSPACE-ПРОБІЛ, збільшення

кількості пробілів між словами) та обчислює відповідні хеш-образи.

Сторона *А* робить це доти, поки не знайде таку пару документів, хеш-образи яких збігаються (якщо довжина хеш-образу 64 біти, то для цього потрібно буде отримати по $2^{32}$ кожного документа).

Сторона *А* підписує у сторони *Б* «правильний» документ із пари, після чого, підмінивши підписаний документ на «неправильний», вона дуже просто зможе довести третій стороні, що *Б* підписала «неправильний» документ.

Спосіб розкриття, описаний у цьому прикладі, отримав назву «розкриття у день народження». Він створює помітну проблему, а отже, й довжина хеш-образу, і складність знаходження повідомлень з однаковим образом повинні задовольняти умови, що висуваються до хеш-функцій.

Розглянемо детальніше найпопулярніші алгоритми хешування.

**Алгоритми хешування сімейства MD**

Алгоритми цього сімейства (*Message Digest* – згортка повідомлення) були розроблені Роном Рівестом і утворюють на виході 128-бітні образи.

Рон Рівест так описував вимоги до цих хешувальних алгоритмів [14]:

*Безпека*. Обчислювально неможливо знайти два повідомлення з однаковими хеш-образами. Атака грубою силою найефективніша.

*Пряма безпека*. Безпека алгоритмів не ґрунтується на жодних припущеннях, наприклад на розкладанні чисел на прості множники.

*Швидкість*. Алгоритми розраховані на швидкісну програмну реалізацію.

*Простота та компактність*. Алгоритми MD прості, наскільки це можливо, не містять великих обсягів даних або складних програмних модулів.

*Архітектура*. Алгоритми оптимізовані для мікропроцесорної архітектури. Для більш складних процесорів легко внести відповідні зміни.

***Алгоритм MD2***

Криптостійкість MD2 ґрунтується на випадкових перестановках окремих байтів.

Для того, щоб отримати хеш-образ повідомлення, треба зробити такі кроки:

1. Повідомлення доповнюється *i* байтами, причому *i* повинно бути таким, щоб довжина повідомлення була кратна 16-ти байтам;
2. До повідомлення додається ще 16 байтів контрольної суми;
3. Ініціалізується 48-байтовий блок: $X_0$, $X_1$,..., $X_{47}$. Перші 16 байтів $X$ заповнюються нулями; у другі 16 байтів $X$ копіюються перші 16 байтів повідомлення; останні 16 байтів отримуються за допомогою XOR перших і других 16 байтів.
4. Функція стиску виглядає так:

$t = 0$,
*for* $j = 0$ *to* 17,
*for* $k = 0$ *to* 47,
$t = X_t$ XOR $S_t$,
$X_k = t$,
$t = (t+j) \bmod 256$,

де $S_t$ – 256 байтів випадкового переміщування з цифр числа $\pi$.

5. Наступні 16 байтів повідомлення копіюються у другі 16 байтів блока, а треті 16 байтів отримуються за допомогою XOR перших і других.
6. Етапи 4-5 повторюються до закінчення відкритого тексту.
7. Вихід – перші 16 байтів блока *X*.

Хоча сьогодні у MD2 не знайдено слабких місць, однак він працює значно повільніше за інші хеш-функції.

***Алгоритми MD4-MD5***

Алгоритми MD4 і MD5 дуже схожі: останній є вдосконаленням першого, коли в ньому було знайдено вразливості. Обидві хеш-функції на виході дають 128-бітні значення.

Алгоритми обробляють повідомлення блоками по 512 бітів, які розбиваються на чотири 32-бітних блоки.

Для отримання хеш-образу цим алгоритмом необхідно виконати такі операції:

1. *Підготовчі операції.* Повідомлення доповнюється до величини так, щоби його загальна довжина була на 64 біти менша за число, кратне 512 бітам. Доповнюється повідомлення одиницею, за якою йде стільки нулів, скільки потрібно. До результату додається 64-бітове представлення істинної довжини повідомлення. Якщо довжина повідомлення більша за $2^{64}$, використовують 64 молодших біти довжини.
2. Ініціалізуються чотири 32-бітні регістри, які заповнюються такими початковими значеннями: $a$ = 01234567; $b$ = 89abcdef; $c$ = fedcba98; $d$ = 76543210. Ці змінні називаються ***змінними зчеплення***.
3. Утворення хеш-образу складається з чотирьох етапів (у MD4 – тільки три етапи), кожен з яких містить по 16 циклів. У кожному етапі для обробки блока тексту використовуються деякі нелінійні функції, на кожному етапі інша. Усього є чотири функції:

$$F(X,Y,Z) = XY \vee \overline{X}Z;$$
$$G(X,Y,Z) = XZ \vee Y\overline{Z};$$
$$H(X,Y,Z) = X \oplus Y \oplus Z;$$
$$I(X,Y,Z) = X \vee Y \vee \overline{Z}.$$

Тут $\oplus$ – додавання за модулем 2; $\vee$ – диз'юнкція; риска над змінною позначає інверсію.

За допомогою функції $F(X,Y,Z)$ визначають процедуру для обробки тексту:

$FF(a, b, c, d, M_j, s, t_i) \equiv a := b + \mathrm{Sh}_s(F(b, c, d) + M_j + t_i)$, де $M_j$ – j-й 32-бітний блок тексту; «+» – означає підсумовування за модулем $2^{32}$; $\mathrm{Sh}_s$ – символізує операцію циклічного зсуву ліворуч на $s$ позицій; $t_i$ – ціла частина числа $2^{32}|\sin i|$ у двійковому представленні.

Для утворення процедур *GG*, *HH*, *II* використовують відповідно функції $G(X,Y,Z)$, $H(X,Y,Z)$, $I(X,Y,Z)$.

4. Значення хеш-образу накопичуються в регістрах $a$, $b$, $c$ і $d$.

*Цикл 1.* Послідовно виконуються такі 16 кроків:

$FF(a, b, c, d, M_0, 7, t_1)$;
$FF(d, a, b, c, M_1, 12, t_2)$;
$FF(c, d, a, b, M_2, 17, t_3)$;
$FF(b, c, d, a, M_3, 22, t_4)$;
……
$FF(b, c, d, a, M_{15}, 22, t_{16})$.

Кожен раз порядок регістрів $a$, $b$, $c$ і $d$ циклічно зсувається на одну позицію праворуч. Окрім цього, аргумент зсуву $s$ набуває ряд значень: 7, 12, 17, 22, 7, 12, …, 22.

*Цикл 2.* Виконується аналогічно до першого циклу, однак функція *FF* замінюється на *GG*; замість послідовності 7,12,17,22 використовують послідовність 5,9,14,20; $i$-й виклик процедури *GG* використовує $t_{16+i}$ замість $t_i$ і $M_{(1+5i) \bmod 16}$ замість $M_i$ ($i$ = 1,2,3…,16).

*Цикл 3.* Виконується з аналогічними замінами: використовують процедуру *HH*, послідовність 4,11,16,23, а також $t_{32+i}$ та $M_{(5+3i) \bmod 16}$ під час $i$-го виклику процедури *HH* ($i$ = 1,2,3…,16).

*Цикл 4.* Використовують процедуру *II*; послідовність 6,10,15,21, а також $t_{48+i}$ та $M_{(3+7i) \bmod 16}$ під час $i$-го виклику процедури *II* ($i$ = 1,2,3…,16).

5. Остаточне значення хеш-образу накопичується в регістрах $a$, $b$, $c$ і $d$ та утворюється їх конкатенацією.

**Безпека MD4-MD5**

Різниця між MD4 та MD5 полягає ось у чому:

1. З'явився четвертий етап.
2. У MD5 у кожному циклі використовується унікальна константа, яка додається до результату ($t_i$).
3. Функція G на другому етапі була замінена (з $XY \vee XZ \vee YZ$ на $XZ \vee Y\overline{Z}$) для досягнення меншої симетрії.
4. У MD5 результат кожного етапу додається до результатів попереднього для досягнення швидшого лавинного ефекту.
5. Змінився порядок використання підблоків повідомлення на етапах 2 та 3 для зменшення симетрії шаблонів.
6. Значення циклічного зсуву оптимізовано для швидшого досягнення лавинного ефекту.

Проти MD5 (проти кожного з етапів) були спроби застосування різницевого криптоаналізу, але він виявився неефективним [14], і було виявлено існування

колізій. З практичної точки зору існування колізій не відіграє суттєвої ролі, це лише означає, що одна із задач, яку ставив перед собою Р. Рівест, а саме розробка безколізійного алгоритму хешування, не була розв'язана.

**Алгоритм SHA-1**

Алгоритм SHA-1 (Secure Hash Algorithm – безпечний алгоритм хешування) було прийнято як стандарт США у 1992 році. Принципи, на яких ґрунтується цей алгоритм, аналогічні тим, за якими розроблявся MD4 [14]. Тому ці алгоритми дуже схожі, однак SHA дає 160-бітний образ.

Для того, щоби створити хеш-образ за алгоритмом SHA-1, треба виконати такі кроки:

1. *Підготовчі операції.* Повідомлення доповнюється до величини, кратної 512 бітам. Використовується те ж доповнення, що й в MD5: спочатку додається 1, а потім стільки нулів, щоби довжина повідомлення стала на 64 біти менше числа, кратного 512; потім додається 64-бітна довжина оригінального повідомлення.
2. Ініціалізують п'ять 32-бітних регістрів, які заповнюються такими константами:

$a$ = 67452301; $b$ = efcdab89; $c$ = 98badcfe; $d$ = 10325476; $e$ = c3d2e1f0.

Після цього починається головний цикл програми. Він обробляє блоки тексту по 512 бітів та продовжується, поки не вичерпаються всі блоки тексту.

3. Головний цикл складається з чотирьох етапів по 20 операцій у кожному (у MD5 – чотири етапи по 16 операцій у кожному). Кожна операція являє собою нелінійну функцію над трьома з п'яти регістрів, а потім виконується зсув і додавання, аналогічно до MD5. У SHA використовується такий набір нелінійних функцій:

$$F_i(X,Y,Z) = XY \vee \overline{X}Z \text{ для } i = 0\text{-}19;$$

$$F_i(X,Y,Z) = X \oplus Y \oplus Z \text{ для } i = 20\text{-}39;$$

$$F_i(X,Y,Z) = XY \vee XZ \vee YZ \text{ для } i = 40\text{-}59;$$

$$F_i(X,Y,Z) = X \oplus Y \oplus Z \text{ для } i = 60\text{-}79.$$

В алгоритмі використовуються такі константи:

$$K_i = 5a827999 \text{ для } i = 0\text{-}19;$$

$$K_i = 6ed9eba1 \text{ для } i = 20\text{-}39;$$

$$K_i = 8f1bbcdc \text{ для } i = 40\text{-}59;$$

$$K_i = ca62c1d6 \text{ для } i = 60\text{-}79.$$

512-бітний блок повідомлення перетворюється з шістнадцяти 32-бітних блоків ($M_0$-$M_{15}$) у вісімдесят 32-бітних слів ($W_0$-$W_{79}$) за допомогою алгоритму

$$W_i = M_i \text{ для } i = 0\text{-}15,$$

$$W_i = \mathrm{Sh}_1(W_{i-3} \oplus W_{i-8}\ W_{i-14}\ W_{i-16}) \text{ для } i = 16\text{-}79.$$

4. Основний цикл обробки виглядає так:

$For\ i = 0\ to\ 79$
$Temp = \mathrm{Sh}_5(a)+F_i(b,c,d)+e+W_i+K_i$
$e = d$
$d = c$
$c = \mathrm{Sh}_{30}(b)$
$b = a$
$a = Temp.$

У кінці кожного циклу отримані значення *a,b,c,d,e* додають до відповідних їм початкових даних і переходять до обробки наступного блоку відкритого тексту *M*.

Остаточне значення хеш-образу довжиною у 160 бітів утворюється конкатенацією регістрів *a, b, c, d, e*.

**Безпека SHA**

SHA дуже схожа на MD4, однак утворює 160-бітне значення хеш-образу. Головною відмінністю є наявність розширювального перетворення та додавання результатів обробки до результатів попередніх циклів.

Порівняння SHA з MD5 виявляє такі відмінності:

1. MD5 має чотири етапи обробки. Однак, на відміну від MD5, у SHA на четвертому етапі використовується функція другого етапу.
2. У MD5 в кожному циклі використовується унікальна константа. SHA повторно використовує константи для кожної групи з двадцяти етапів.
3. В MD5 було замінено функцію другого етапу для досягнення більшої асиметрії. SHA використовує функцію з MD4.
4. І в MD5, і у SHA результати обробки після кожного циклу додаються до результатів попереднього. Відмінність полягає в тому, що у SHA добавлено п'яту змінну, яка робить атаки, застосовані проти MD5, неможливими.
5. Змінився порядок виборки блоків тексту на етапах 2-3. SHA абсолютно відрізняється цим від MD5, оскільки використовує циклічний код виправлення помилок.
6. SHA, на відміну від MD5, використовує постійне значення зсувів на кожному етапі. Це число – взаємно просте з розміром слова, як і в MD4.

Підсумовуючи, можна сказати, що SHA – це MD4 з додатковим етапом, розширювальним перетворенням і поліпшеним лавинним ефектом.

Відомості про успішні атаки на SHA відсутні [14]. Оскільки вона утворює 160-бітний хеш-образ, то вона стійкіша до атак методом «грубої сили» порівняно з тими 128-бітними хеш-функціями, які розглянуто вище.

# Алгоритми електронного цифрового підпису

## Алгоритм цифрового підпису DSA

Алгоритм DSA було прийнято як федеральний стандарт цифрового підпису в США в 1991 році. Після кількох модифікацій він був остаточно затверджений як стандарт для несекретних застосувань у 1994 році. Оскільки він уводився безкоштовно, то проти нього активно виступали прихильники міжнародного стандарту цифрового підпису ISO 9796, зокрема Рон Рівест та Аді Шамір, які були фінансово не зацікавлені у введенні нового стандарту, оскільки міжнародний використовував розроблену ними криптосистему RSA. Суттєві зауваження криптоаналітиків були враховані розробниками алгоритму, і він був уведений у дію [14,20].

Алгоритм DSA (Digital Signature Algorithm) являє собою варіант цифрового підпису Шнорра-Ель Гамаля та використовує такі параметри:

$p$ – просте число довжиною $L$ бітів, де $L$ може набувати значень від 512 до 1024 бітів (у першому варіанті алгоритму $p$ мало фіксоване значення у 512 бітів, що критикувалося криптографами як мале значення модуля). Значення $p$ має бути кратним 64;

число $q$, що є дільником $p$-1 (щонайменше 160 бітів у двійковому представленні);

елемент $h$ з множини $\{1,p\}$ порядку $q$;

випадкове число $a < q$;

число $b = h^a \bmod p$.

Алгоритм також використовує хеш-функцію $H(M)$. Стандарт вимагає використання SHA.

Величини $p$, $q$ ,$h$ і $b$ відкриті та утворюють ***публічний ключ***.

***Приватним ключем*** у такому разі буде число $a$.

Для ***формування електронного підпису*** повідомлення $M$ власник приватного ключа виконує такі дії:

А) обирає випадкове число $r<q$.

Б) обчислює $r' = r^{-1} \bmod q$;

В) обчислює $s_1 = (h^r \bmod p) \bmod q$;

Г) обчислює $s_2 = r'(H(M)+as_1) \bmod q$;

Д) надає пару чисел $(s_1, s_2)$ як підпис для повідомлення $M$.

***Перевірка електронного підпису*** зводиться до такого:

А) отримувач обчислює $s' = s_2^{-1} \bmod q$;

Б) обчислює $u_1 = H(M)\, s' \bmod q$;

В) обчислює $u_2 = s' s_1 \bmod q$;

Г) обчислює $t = (h^{u_1} b^{u_2} \bmod p) \bmod q$.

Якщо $t = s_1$, – підпис справжній.

**Доведення коректності процедури підписування.** Коректність процедури ґрунтується на тому, що $h^r = h^{u_1} b^{u_2} \bmod p$.

Справді, оскільки $b = h^a \bmod p$, то $h^r = h^{u_1} b^{u_2} \bmod p$; $h^r = h^{u_1 + au_2} \bmod p$. Для доведення треба показати, що $r = (u_1 + au_2) \bmod q$. Це рівнозначне доведенню істинності співвідношення $1 \equiv r'(u_1 + au_2) \bmod q$.

Підставивши сюди значення $u_1$ $u_2$, будемо мати:

$r'(u_1 + au_2) \equiv r'(H(M)s' + as's_1) \bmod q \equiv r'(H(M) + as_1)s' \bmod q \equiv s_2 s' \bmod q \equiv s_2 s_2^{-1} \bmod q \equiv 1.$

Отже, процедура утворення та перевірки підпису коректні.

**Безпека DSA**

Як уже згадувалося, основним недоліком першої версії DSA була замала довжина модуля $p$ – усього 512 бітів. Тому в подальших редакціях цього алгоритму було вирішено надати можливість користувачам змінювати довжину модуля від 512 до 1024 бітів, що значно підсилило криптостійкість алгоритму до атаки «грубою силою».

Як видно з самого алгоритму, випадкові числа, що використовуються для створення цифрового підпису, відкриті. Це уможливлює аналіз стійкості цих чисел усьому криптографічному загалові. Цей факт свідчить на користь алгоритму DSA, бо найбільш популярний алгоритм RSA зберігає генеровані прості числа в таємниці, і перевірити їх якість можна, лише знаючи приватний ключ. Отже, як стверджується у [14], алгоритм цифрового підпису DSA можна вважати стійкишим, ніж решта алгоритмів, що утворюють 128-бітний хеш-образ. Відомості про успішні атаки на DSA досі невідомі.

**Стандарти цифрового підпису ГОСТ Р 34.10-94 і ГОСТ Р 34.10-2001**

Алгоритм ГОСТ Р 34.10-94 дуже схожий на DSA та використовує такі параметри:

$p$ – просте число, довжина якого може бути між 509 та 512 або 1020-1024 біти;

$q$ – просте число, множник $p$-1, довжиною 254-256 бітів;

$a$ – будь-яке число, менше за $p$-1, для якого $a^q \bmod p = 1$;

$x$ – довільне число, менше за $q$;

$y = a^x \bmod p$.

Перші три параметри, $p$, $q$, $a$ відкриті та можуть використовуватися всіма користувачами мережі. Величина $y$ складає публічний ключ, а $x$ – приватний.

Щоби підписати повідомлення $M$, необхідно виконати такі кроки:

1. Відправник генерує випадкове число $k$, менше за $q$.

2. Відправник обчислює два числа: $r' = a^k \bmod p$ та $r = r' \bmod q$. Якщо $r = 0$, то генерують інше число $k$.

3. З використанням приватного ключа користувача обчислюється:

$$s = (xr + kH(M)) \bmod q.$$

Якщо $s = 0$, то генерують нове число $k$.

4. Каналом зв'язку відправляють повідомлення $M$ разом із підписом $s$.

Рівняння перевірки підпису в такому разі буде таким:

$$r \equiv (a^{sH(M)^{-1}} y^{-rH(M)^{-1}} \bmod p) \bmod q.$$

Зауважимо, що повідомлення, яке дає нульове значення хеш-образу $H(M)$, не підписується, оскільки в такому разі рівняння перевірки підпису спрощується настільки, що зловмисник може легко обчислити приватний ключ.

Зі збільшенням продуктивності процесорів і розробкою нових типів криптоаналітичних атак з'явилася потреба у зміцненні алгоритму цифрового підпису. Тому стандарт ГОСТ Р 34.10-94 було модифіковано. Новий стандарт, ГОСТ Р 34.10-2001, було введено в дію 1 липня 2002 року замість старого.

У цьому стандарті використовують алгоритм з операціями групи точок еліптичної кривої над кінцевим полем [3]. Використовують еліптичну криву $E$ у формі Вейєрштраса над простим полем, яка задається коефіцієнтами $a$ та $b$ або інваріантом кривої:

$$J(E) \equiv 1728 \frac{4a^3}{4a^3 + 27b^2} \bmod p.$$

Коефіцієнти $a$ та $b$ визначаються за відомим інваріантом так:

$$a \equiv 3k \bmod p;$$
$$b \equiv 2k \bmod p,$$

де $k \equiv \dfrac{J(E)}{1728 - J(E)} \bmod p; J(E) \neq 0;1728.$

Точку $Q$ будемо називати точкою кратності $k$, якщо для деякої точки $P$ справджується рівність $Q = kP$.

Параметрами цієї схеми ЕЦП будуть такі значення:

1) $p$ – модуль еліптичної кривої, просте число, $p > 2^{255}$;
2) еліптична крива, яку задано інваріантом $J(E)$ або коефіцієнтами $a$ та $b$;
3) ціле число $m$ – порядок групи точок еліптичної кривої $E$;
4) просте число $q$ – порядок циклічної підгрупи групи точок еліптичної кривої $E$, для якого виконуються такі умови:

$$m=nq,\ n \in Z,\ n \geq 1;\ \ 2^{254} < q < 2^{256};$$

5) базова точка $P \neq 0$ на кривій порядку $q$ (тобто $qP = 0$). Координати цієї точки позначимо через $(x_p, y_p)$;
6) хеш-функція, передбачена стандартом ГОСТ Р 34.11-94, яка перетворює двійкову послідовність довільної довжини у двійкову послідовність довжиною 256 бітів.

Кожен учасник інформаційного обміну повинен мати власну пару ключів:

- приватний ключ, ціле число $d$, $0 < d < q$;

- публічний ключ, точка $Q$ з координатами $(x_a, y_a)$, що задовольняють рівняння $dP = Q$.

Для формування цифрового підпису необхідно виконати такі дії:

1. Обчислити хеш-образ повідомлення $M$, $\boldsymbol{h}(M)$.
2. Двійковому вектору $\boldsymbol{h} = (\alpha_0, \ldots, \alpha_{255})$ поставити у відповідність число

$$\alpha = \sum_{i=0}^{255} \alpha_i 2^i$$

та обчислити число $e \equiv \alpha \bmod q$. Якщо $e = 0$, то встановити $e = 1$.

3. Згенерувати випадкове число $0 < k < q$.
4. Обчислити точку еліптичної кривої $C = kP$ і визначити величину $r \equiv x_C \bmod q$, де $x_C$ – $x$-координата точки $C$. Якщо $r = 0$, то згенерувати наступне випадкове число $r$.
5. Обчислити значення $s \equiv (rd+ke) \bmod q$. Якщо $s = 0$, тоді обираємо нове число $k$.
6. Обчислити двійкові вектори, що відповідають числам $r$ і $s$. Визначити цифровий підпис як конкатенацію цих двійкових представлень: $\xi = \{\boldsymbol{r}||\boldsymbol{s}\}$.

Для перевірки справжності ЕЦП виконують такі дії:

1. За отриманим значенням ЕЦП $\xi$ відновлюють значення $r$ та $s$. Якщо $0 < r < q$, $0 < s < q$, то переходимо до наступного кроку. Якщо ні, то підпис несправжній.
2. Обчислюють хеш-образ отриманого повідомлення $\boldsymbol{h}(M)$.
3. Обчислюють число $\alpha$, двійковим представленням якого є вектор $\boldsymbol{h}$ та число $e \equiv \alpha \bmod q$. Якщо $e = 0$, то встановити $e = 1$.
4. Обчислюють $v \equiv e^{-1} \bmod q$, $z_1 \equiv sv \bmod q$, $z_2 \equiv -rv \bmod q$.
5. Обчислюють точку еліптичної кривої $C = z_1P + z_2Q$ та визначають $R \equiv x_C \bmod q$, де $x_C$ – $x$-координата точки $C$.
6. Якщо виконується рівність $R = r$, то підпис справжній. У протилежному випадку підпис не підтверджено.

Аналогічно працюють й інші системи електронного цифрового підпису, які ґрунтуються на еліптичних кривих (наприклад, ECDSA). Вони привертають увагу криптографів, оскільки дозволяють конструювати «елементи» та «правила об'єднання», які формують групи. Властивості цих груп відомі достатньо добре, щоби використати їх у криптографічних застосуваннях. Однак групи точок еліптичних кривих, на відміну від інших, не мають характерних властивостей, що полегшують криптоаналіз. Наприклад, їм не властиве поняття «гладкості».

За допомогою еліптичних кривих можна реалізувати багато криптографічних алгоритмів із відкритими ключами, наприклад Діффі-Хеллмана, Ель-Гамаля та інші. Відповідна математика дуже складна і виходить за межі цієї книги.

Детальніше про криптографічні системи на еліптичних кривих можна почитати у [13, 14].

# Розділ VI
# Елементи криптоаналізу

Багато років тому, коли сторонами, що обмінювалися конфіденційною інформацією, були переважно дипломати та військові, вони були впевненими у надійності учасників обміну, довіряли один одному. Основна проблема сторін тоді полягала у забезпеченні неможливості втручання в конфіденційний зв'язок третіх сторін. Практично не відомі випадки, коли адресати поводили себе недобросовісно: відмовлялися від отриманих повідомлень або стверджували про якісь зміни в ньому. Іншими словами, справа зводилася до обміну конфіденційною інформацією між «своїми», які користувалися повною довірою один одного. Основне завдання тогочасних криптографів полягало лише в забезпеченні ***конфіденційності*** інформації.

Принципово інакше складається картина сьогодні, коли інформацією обмінюються суб'єкти комерційної діяльності, які можуть не довіряти один одному. І якщо раніше головним було забезпечення лише ***конфіденційності*** інформації, то зараз не менш важливим є забезпечення її ***цілісності*** та ***юридичної чинності***, а також захист від ***нав'язування хибних повідомлень***.

Отже, метою суб'єкта в такому разі є перешкоджання здійсненню намірів законних учасників інформаційного обміну. В загальному випадку супротивник може не лише перехоплювати зашифровані повідомлення, але й модифікувати їх, а також направляти фальсифіковані повідомлення легальним сторонам, що обмінюються інформацією, ніби від імені їх легальних партнерів.

Таким чином, існує чотири можливих типи загроз із боку супротивника [21,22]:

- ***порушення конфіденційності інформації*** – дешифрування, повне чи часткове, переданого повідомлення або отримання додаткової інформації про його зміст;
- ***порушення цілісності інформації*** – внесення змін у повідомлення, що змінюють його зміст;
- забезпечення ***неможливості відмови*** від отриманого чи відправленого повідомлення;
- ***порушення істинності повідомлень*** – формування хибних повідомлень, які легальні учасники інформаційного обміну можуть класифікувати як істинні.

Дешифрування перехопленого повідомлення можливе у випадку обчислення криптографічного ключа або т.зв. «безключового читання», тобто за допомогою знаходження еквівалентного алгоритму, що не вимагає знання ключа.

Коротко охарактеризуємо основні типи атак. Нижче вони перелічені в порядку зростання небезпеки.

**1.** ***Атака на основі лише шифротексту*** (*ciphertext-only attack*). У цьому випадку супротивнику відомі лише шифротексти, зашифровані на одному ключі. Це найслабший тип криптоаналітичної атаки, успіх якої зовсім неочевидний.

Він залежить від багатьох чинників та визначається кваліфікацією аналітика (за умови ручного криптоаналізу) або повністю визначається потужністю комп'ютерів криптоаналітичної системи.

Процес криптоаналізу завжди починається зі збирання інформації про відкритий текст:

- якою мовою написаний оригінал;
- які лінгвістичні особливості цієї мови;
- які слова або фрази може містити оригінал та в якій послідовності;
- якою приблизно може бути довжина оригінального тексту;
- які методи шифрування могли застосувати для зашифрування цього тексту;
- яка службова інформація може міститися в тексті (дата, контрольна сума, адреси тощо);
- якою апаратурою було зашифровано текст.

Ці або подібні дані збираються агентурними або аналітичними засобами і значно полегшують роботу криптоаналітиків.

**2. *Атака на основі невибраного (відомого) відкритого тексту*** (*known-plaintext attack*). Супротивник знає або може встановити деякі частини відкритого тексту у криптограмі. Задача полягає в тому, щоби дешифрувати весь текст. Це можна виконати шляхом покрокового обчислення ключа.

**3. *Атака на основі обраного відкритого тексту*** (*chosen-plaintext attack*). Супротивник може обрати будь-який відкритий текст і отримати для нього зашифрований. Задача полягає у визначенні ключа шифрування. Деякі алгоритми шифрування досить вразливі для атак цього типу. Тому у випадку використання таких систем серйозної уваги вимагає внутрішня безпека використання системи, щоби супротивник ні в якому разі не зміг отримати доступ до відкритих текстів.

Така атака буде називатися ***простою***, коли всі відкриті тексти супротивник отримує до перехоплення першої криптограми, та ***адаптивною***, якщо супротивник обирає черговий відкритий текст, маючи шифровки всіх попередніх.

**4. *Атака на основі обраного шифротексту*** (*chosen-ciphertext attack*). Супротивник може обирати потрібну кількість криптограм та отримати для них відкриті тексти. Тут також, аналогічно до попереднього типу атак, існують різновиди простої та адаптивної атак.

**5. *Атака на основі обраного тексту*** (*chosen-text attack*). Це найнебезпечніший тип атак, оскільки супротивник може передавати спеціально підготовлені відкриті тексти для зашифрування, а потім отримувати відповідні шифровки. Задача полягає в розкритті ключа. Ця атака також може бути простою та адаптивною.

Якщо супротивник здійснює атаку на основі лише шифротексту, в нього є дуже обмежений набір можливостей. До них можна віднести метод грубої сили або частотний криптоаналіз.

У разі атаки на основі відомого відкритого тексту, розкриття шифру стане елементарним після того, коли в текстах зустрінеться більшість літер абетки.

Якщо супротивник має можливість атакувати на основі обраного тексту, то криптосистему буде розкрито вже після шифрування відкритого тексту типу: «*АБВГД...ЬЮЯ*».

**6.** Специфічний тип атаки, ***«посередництво»*** (*Man-in-middle attack*). Ця атака спрямована на злам криптографічних комунікацій і протоколів обміну ключами. Атака здійснюється приблизно так. Припустимо, що зловмисник (назвемо його *З*) має змогу перехоплювати всі повідомлення сторін *А* та *В*. Припустимо також, що сторони хочуть обмінятися криптографічними ключами, тож *А* відправляє свій ключ шифрування $K_{АВ}$ стороні *В*. Зловмисник *З* перехоплює цей ключ, зберігає його та відправляє стороні *В* вже свій криптографічний ключ, $K_{ЗВ}$, ніби від сторони *А*. Сторона *В*, отримавши цей ключ, у відповідь відправляє стороні *А* свій криптографічний ключ $K_{ВА}$. Зловмисник *З* також підміняє цей ключ своїм, $K_{ЗА}$ і надсилає його стороні *А*. Таким чином, сторони *А* та *В* «вірять», що мають криптографічні ключі один одного, хоча насправді вони мають лише два різні ключі зловмисника *З*. Той, у свою чергу, має обидва ключі сторін і може читати всю їхню зашифровану інформацію (див. рис. 6.1).

Спроба обміну інформацією між сторонами приводить до такого. *А* шифрує повідомлення ключем $K_{АВ}$ і надсилає його. *З* перехоплює цього листа, розшифровує його перехопленим ключем сторони *А* ($K_{АВ}$), перешифровує ключем, який він надіслав *В* ($K_{ЗВ}$), та відправляє. Сторона *В*, отримавши зашифрованого листа, розшифровує його і, оскільки сеанс розшифрування пройшов успішно, «вірить», що обмін інформацією триває нормально. При спробі сторони *В* відповісти на лист від *А* зловмисник виконує аналогічну процедуру.

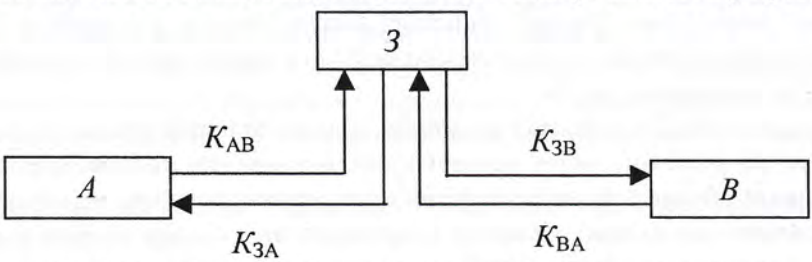



*Рис. 6.1. Схема атаки* Man-in-Middle *при обміні криптографічними ключами*

Отже, зловмисник читає всю інформацію, що курсує між сторонами *А* та *В*, оскільки володіє чотирма криптографічними ключами, і продовжуватися це може доти, поки сторони *А* та *В* фізично не зустрінуться та зрозуміють, що оперують незнайомими криптографічними ключами, або *З* не використає проти них перехоплену інформацію.

Посередництво – одна з найнебезпечніших атак проти систем зв'язку. Захист від нього потребував створення потужної системи – інфраструктури відкритих ключів і застосування хеш-функцій. Успішний захист від посередництва полягає в обчисленні хеш-образу повідомлень і підписуванні цього образу електронним цифровим підписом. У такому разі отримувач має змогу перевірити цілісність

інформації порівнянням хеш-функцій та власника цифрового підпису за допомогою сертифіката довірчого центру.

**Атаки на криптосистему RSA**

Однією з найвідоміших атак на криптосистему RSA є така, що використовує близькі значення $p$ та $q$. Припустимо, що $p > q$, хоча це ніяк не обмежує подальші міркування. Тоді ми можемо записати: $x=(p+q)/2$ $y=(p-q)/2$, а для $n$ справедливе рівняння

$$n = x^2 - y^2. \qquad (*)$$

Для знаходження множників $p$ і $q$ досить підібрати числа $x$ і $y$, що задовольняють (*). Знайдемо $x_0 = \sqrt{n}$ та оберемо найближче до нього більше ціле $x_1$. Підставляючи його у (*), знаходимо відповідне значення $y$. Повторюємо цю операцію доти, поки не отримаємо цілі значення $x$ та $y$, що задовольняють (*). В результаті знаходимо $p = x+y$, $q = x-y$.

Розглянемо ***приклад***. Нехай $n = pq = 851$. Скористаємося описаним вище правилом для знаходження $p$ і $q$. Оскільки $\sqrt{n} = 29.17$, то, припустивши, що $x = 30$, знаходимо $y^2 = 30^2-851 = 49$. Отже, $y = 7$, і ми з першої спроби знаходимо розв'язок (*): $x = 30$, $y = 7$. Тому $p = 30+7 = 37$, а $q = 30-7 = 23$. Обчисливши значення $p$ і $q$, зловмисник може легко обчислити приватний ключ, зламавши в такий спосіб криптосистему.

Іншою атакою на криптосистему RSA є ***атака з вибраним шифротекстом***. Нехай ми перехопили повідомлення $C = E_e(M)$, зашифроване на публічному ключі алгоритму RSA.

Обираємо число $r < n$. Тоді можна обчислити таке:

$x = r^e \bmod n;$

$y = xC \bmod n;$

$t = r^{-1} \bmod n.$

Якщо $x = r^e \bmod n$, тоді $r = x^d \bmod n$.

Тепер просимо власника ключа підписати $y$ приватним ключем $d$. Будемо мати $u = y^d \bmod n$.

Обчислимо

$$tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^dC^d \bmod n = C^d \bmod n = M.$$

Таким чином, ми отримали розшифроване повідомлення $M$.

Захист від такої атаки може здійснюватися двома шляхами. Перший полягає в тому, що пари ключів для шифрування та електронного підпису повинні бути різними. Другий – підписувати завжди потрібно хеш-образ повідомлення, а не відкрите повідомлення.

***Атака на електронний підпис***. Якщо зловмиснику необхідно отримати електронний підпис повідомлення $M$, він може діяти так. Створюються

повідомлення $M_1$ і $M_2$ такі, що $M = M_1M_2 \bmod n$, та підписуються окремо повідомлення $M_1$ та $M_2$.

Якщо ми маємо ЕЦП повідомлень $M_1$ і $M_2$, тоді

$$M_3^{\,d} = (M_1^{\,d} \bmod n)\,(M_2^{\,d} \bmod n).$$

Отже, повідомлення $M$ підписане.

***Атака на спільний модуль***. При реалізації RSA іноді буває вигідним роздати всім користувачам спільний модуль $n$ і різні ключі $e$ та $d$. Однак така схема працює, поки два користувача не зашифрують на різних ключах однакове повідомлення. Якщо $e_1$ та $e_2$ – взаємно прості числа, тоді можлива наступна атака.

Нехай $m$ – повідомлення, $e_1$ та $e_2$ – два публічних ключа, $n$ – спільний модуль. Процес зашифрування можна зобразити рівняннями

$$C_1 = m^{e_1} \bmod n; \qquad C_2 = m^{e_2} \bmod n\,.$$

Отже криптоаналітик знає $n$, $e_1$, $e_2$, $C_1$, $C_2$. За допомогою алгоритму Евкліда він знаходить такі $r$ та $s$, які задовольняють рівняння

$$se_1+re_2=1 \quad (r<0;\ s>0)$$

та обчислюють $C_2^{-1} \bmod n$. Тоді

$$C_1^S (C_2^{-1})^{-r} \bmod n = m\,.$$

Доведемо це, підставивши замість $C_1$ та $C_2$ їх значення:

$$m^{e_1 s}((m^{e_2})^{-1})^{-r} \bmod n = m^{e_2 s} m^{e_2 r} \bmod n = m^{e_2 s+e_2 r} \bmod n = m\,.$$

Зважаючи на таку атаку, задавати спільний модуль для групи користувачів дуже небезпечно.

***Атака дешифрування ітераціями***. Суть методу полягає в тому, що перехоплену криптограму повторно шифрують на публічному ключі і при деякій кількості ітерацій отримують вихідне повідомлення. Кількість ітерацій, звичайно, залежить від довжини модуля та ключа. За невеликої довжини модуля цього можна досягти вже при кількох ітераціях. Розглянемо ***приклад***. Нехай $p = 3$, $q = 11$; $n = 33$; $e = 7$, $d = 3$; повідомлення $m$ = «02». Обчислення криптограми дає: $C = 2^7 \bmod 33 = 29$. Спробуємо перешифрувати криптограму на публічному ключі (7, 33):

перша ітерація – $C_1 = 29^7 \bmod 33 = 17$;

друга ітерація – $C_2 = 17^7 \bmod 33 = 8$;

третя ітерація – $C_3 = 8^7 \bmod 33 = 2 = m$.

Як бачимо, вже третя ітерація дозволила отримати відкритий текст. На практиці це повинно було б виглядати так. Перехоплена шифровка перешифровується на публічному ключі доти, поки на виході не отримується зв'язний текст. Зрозуміло, що це обов'язково станеться, однак час, потрібний для цього, не повинен бути меншим за час для повного перебору ключів чи розкладання модуля на множники [11].

***Елементи частотного криптоаналізу***

Зауважимо, що частотним криптоаналізом користуються вже дуже давно. Перші частотні таблиці, за даними Девіда Кана [1], склали ще стародавні араби. Частотний криптоаналіз неодноразово описувався в художній літературі. Наприклад, Шерлок Холмс, використовуючи принципи частотного криптоаналізу, успішно впорався з розшифровкою записок, які отримувала від зловмисника місіс К'юбіт (А. Конан-Дойл «Танцюючі чоловічки»). Аналогічно головні герої виконували розшифровку повідомлень, зашифрованих методом простої заміни, в оповіданні Едгара По «Золотий жук» і романі Жюля Верна «Подорож до центру Землі» [9].

Однак теоретичне обґрунтування принципи частотного криптоаналізу отримали лише в праці Клода Шеннона «Теорія зв'язку в секретних системах». У ній Клод Шеннон увів поняття надмірності мови, яку визначив так: «*З кожною мовою пов'язана деяка величина D, яку можна назвати надмірністю цієї мови. Надмірність має той зміст, наскільки можна зменшити довжину деякого тексту без втрати будь-якої частини інформації*». К.Шеннон наводить простий приклад: «*...оскільки у словах англійської мови за літерою q завжди йде тільки літера u, то u можна без втрати змісту пропустити. Значних скорочень у англійській мові можна досягти, використовуючи його статистичну структуру, часту повторюваність певних літер і слів. Надмірність мови відіграє центральну роль у вивченні секретних систем*».

У Додатку Б подано частотні таблиці української, російської та англійської мов. Їх створено на основі аналізу текстів загального спрямування. Зрозуміло, що в залежності від змісту текстів частотні таблиці можуть дещо змінюватися, однак у більшості випадків можна користуватися наданими в Додатку. З таблиць видно, що в текстах українською мовою найчастіше зустрічаються літери «*о*», «*н*», «*а*», а найменше – «*ш*», «*щ*», «*ґ*». Російській мові властиве часте використання літер «*о*», «*е*», «*а*» та «*н*», а дуже рідко – «ъ», «ь» та «*ы*».

Англійська мова характеризується тим, що надзвичайно часто використовує літеру «*e*» (у 1,33 разу частіше, ніж наступну за частотою – «*t*»!), причому ці дві літери часто зустрічаються в одному слові, «*the*». Саме завдяки цьому визначеному артиклеві відносно часто зустрічається літера «*h*», яка в інших англійських словах досить рідкісна.

Статистичні особливості мови, якою написаний відкритий текст, дуже часто зберігаються в зашифрованому, особливо, якщо зашифрування виконувалося простим алгоритмом, наприклад звичайною заміною.

Отже, знаючи статистичні особливості мови оригіналу, можна використати їх для аналізу шифрованого тексту.

Спочатку виконують попередній аналіз зашифрованого тексту, тобто дають відповіді на такі питання:

- якою мовою написаний оригінал;
- які слова або фрази може містити оригінал та в якій послідовності;

- які методи шифрування могли застосувати для зашифрування цього тексту.

Коли відповіді відомі, можна підрахувати статистичні особливості зашифрованого тексту, порівняти їх з частотними таблицями відповідної мови та спробувати розшифрувати повідомлення.

Наведемо простий приклад використання частотного криптоаналізу для розшифрування повідомлень.

Нехай перехоплене повідомлення має вигляд

*«15 17 01 30 14 08 06 20 10 19 03 06 20 18 19 17 08 24 14 15 00 15 01 15 15 31 04 11 19 20 00 15 31 20 13 15 00 12 03 14 15 13 20 13 08 18 23 08».*

Припустимо, що оригінал повідомлення написаний українською мовою. Ми бачимо, що, найімовірніше, повідомлення зашифроване методом простої заміни. Використаємо частотний криптоаналіз. Для цього підрахуємо, скільки разів кожне число (а тут використано числа від *00* до *31*, що, до речі, є опосередкованим підтвердженням використання української мови; в англійській мові – 26 символів, а у російській – 33, див. частотні таблиці в Додатку) зустрічається в повідомленні. Найчастіше зустрічається число *15* – 8 разів. Оскільки в повідомленні всього 48 знаків, то ймовірність появи числа *15* дорівнює 8/48 = 0.167. Із частотних таблиць української мови відомо, що найчастіше в українських текстах зустрічається літера «*о*». Тому припустимо, що числом *15* зашифровано літеру «*о*». Оскільки літера «*о*» – 17-та за порядком в українській абетці, то очевидно, що зсув при шифруванні дорівнював -2. Отже, таблиця заміни виглядає так:

| *А* | *Б* | *В* | *Г* | *Д* | *Е* | *Є* | *Ж* | *З* | *И* | *І* | *Ї* | *Й* | *К* | *Л* | *М* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *30* | *31* | *00* | *01* | *02* | *03* | *04* | *05* | *06* | *07* | *08* | *09* | *10* | *11* | *12* | *13* |
| *Н* | *О* | *П* | *Р* | *С* | *Т* | *У* | *Ф* | *Х* | *Ц* | *Ч* | *Ш* | *Щ* | *Ь* | *Ю* | *Я* |
| *14* | *15* | *16* | *17* | *18* | *19* | *20* | *21* | *22* | *23* | *24* | *25* | *26* | *27* | *28* | *29* |

Замінивши числа літерами, отримаємо: «*Організуйте зустріч нового об'єкта в обумовленому місці*».

Звичайно, це штучно створене повідомлення зі словами, що мають велику кількість літер «*о*», не дасть повного враження про частотний криптоаналіз, однак принцип зрозумілий. У реальній практиці часто доводиться здогадуватися про те, які слова та вирази можуть міститися в повідомленні, шукати характерні для цих виразів послідовності символів, а також аналізувати перехоплений текст на предмет наявності характерних пар літер (наприклад, «*th*», «*qu*» в англійській мові, «*нн*» – у російській та українській мовах). Більше того, для гарної роботи частотного криптоаналізу необхідна значна довжина тексту, корисними будуть відомості про відправника й отримувача, тобто деяка додаткова інформація.

Яскраві й повчальні приклади частотного криптоаналізу можна знайти у книзі В. В. Ященка [9].

### *Різницевий криптоаналіз*

Різницевий (диференціальний) криптоаналіз розроблено Біхамом і Шаміром у 1990 році. Однак існують дані про те, що Агентство Національної Безпеки США при побудові блоків заміни для алгоритму DES (нагадаємо, що його було прийнято як стандарт у 1976 році) врахувало можливість використання проти нього різницевого криптоаналізу. Отже, він був відомий криптоаналітикам цієї організації вже тоді, і своє небажання розсекретити вимоги до блоків заміни DES вони пізніше пояснили тим, що це б сприяло б розробці різницевого криптоаналізу ще в 1976 році.

Метод різницевого криптоаналізу придатний для атак на криптосистеми, що ґрунтуються на багаторазовому використанні циклів. Багато криптосистем, які були альтернативою DES, повністю розкрили за допомогою різницевого криптоаналізу, і через це вилучили з практичного вжитку. DES із малою кількістю циклів (наприклад, 8) також нестійкий до різницевого криптоаналізу. Для DES з 16-ма циклами цей метод уже малоефективний бо вимагає $2^{47}$ пар відкритого тексту криптограми, що не реалізується на практиці.

Різницевий криптоаналіз використовують до пар шифротекстів, відкриті тексти яких мають певні відмінності. Еволюцію цих відмінностей при проходженні відкритого тексту крізь цикли DES і досліджують згаданим методом у випадку шифрування тим самим ключем.

Цей криптоаналіз виконують приблизно так [4]. На початку розглядають лише два відкриті тексти з фіксованою різницею. Тексти можна вибирати й випадково. Однак у такому разі вони повинні мати певні відмінності. У випадку DES такими відмінностями вважають порозрядну суму за модулем 2. Для інших алгоритмів відмінності можна визначити інакше. Далі виконують шифрування цих двох текстів на однакових ключах шифрування. Використовують, як правило, шифрування на вибраній сукупності ключів. Наступним кроком, після вивчення відмінностей у відповідних шифротекстах, є присвоєння різної ймовірності різним ключам шифрування. У процесі подальшого дослідження наступних пар шифротекстів один із ключів стає найімовірнішим. Цей ключ і вважають правильним.

Більш детально про різницевий криптоаналіз одного циклу DES і DES у цілому можна прочитати у [4, 14].

### *Лінійний криптоаналіз*

Лінійний криптоаналіз являє собою інший тип криптоаналітичної атаки, яку розробив Міцуру Мацуї [14]. Ця атака використовує лінійне наближення для опису роботи блочного шифру. Це означає, що, якщо Ви виконаєте операцію XOR над деякими бітами відкритого тексту, далі – над бітами шифротексту, а потім – над результатами, то отримаєте біт, що являє собою XOR деяких бітів ключа. Це називається лінійним зміщенням, яке може бути правильним із деякою ймовірністю $p$. Якщо $p \neq 1/2$, то це зміщення можна використовувати. Далі аналізують зв'язані пари відкритий текст-шифротекст і роблять припущення

стосовно деяких бітів ключа. Чим більше у Вас даних, тим правильнішим буде припущення. Чим більше зміщення, тим швидше атака досягне успіху.

Як визначити ефективне лінійне зміщення для DES? Для цього треба обрати гарне наближення для одного циклу й об'єднати їх. Тепер давайте поглянемо на S-блоки. На вході в них 6 бітів, а на виході – 4. Вхідні біти можна об'єднати за допомогою операції XOR 63 способами ($2^6 - 1$), а вихідні – 15 способами. Тепер для кожного блока можна оцінити ймовірність того, що для випадково обраного входу вхідна комбінація XOR дорівнює деякій вихідній комбінації XOR. Якщо існує лінійна комбінація з достатньо великим зміщенням, то лінійний криптоаналіз може спрацювати.

Лінійний криптоаналіз сильно залежить від структури S-блоків і виявилося, що S-блоки DES не оптимізовані проти такого способу розкриття. Дійсно, за даними [14], зміщення у стандартних S-блоках знаходиться між 9 та 16%, що не забезпечує надійного захисту проти лінійного криптоаналізу. Стверджують, що стійкість до лінійного криптоаналізу не входила в число критеріїв при проектуванні DES. Причина цього невідома: або розробники не знали про можливості лінійного криптоаналізу, або віддали перевагу стійкості проти диференціального, найбільш небезпечного з їхнього погляду методу.

Підраховано, що для найкращого лінійного наближення необхідно $2^{47}$ відомих відкритих блоків, результатом буде 1 біт ключа, а це дуже мало. Якщо змінити напрям і використовувати для аналізу шифротекст, а не відкритий текст, та розшифрування замість шифрування, то в результаті лінійного криптоаналізу можна отримати 2 біти ключа. Проте і цього недостатньо.

Однак існують деякі тонкощі. Якщо використати лінійний криптоаналіз паралельно $2^{12}$ разів та обрати правильний варіант, ґрунтуючись на ймовірностях, це дасть 12 бітів ключа. 13 бітів ключа можна отримати, змінивши шифрування на розшифрування, а інших 30 бітів – «грубою силою», тобто повним перебором.

Розкриття повного 16-раундового DES за такою схемою вимагає $2^{43}$ відомих відкритих текстів. Програмна реалізація цього методу на 12 робочих станціях HP9735 розкрила ключ DES за 50 днів [14]. На момент написання книжки Брюса Шнайєра це був найбільш ефективний метод атаки проти DES.

Лінійний криптоаналіз молодший за диференціальний, тому дуже імовірний подальший розвиток цих ідей.

Детальніше про лінійний криптоаналіз можна прочитати у книжках [4, 14].

# Розділ VII
# Проблеми розподілу криптографічних ключів

Під час вибору криптографічної системи найважливішою проблемою є керування ключами. Будь-яка, навіть найнадійніша, криптосистема базується на використанні ключів. Ефективність криптографічного захисту інформації в комп'ютерних системах і мережах визначається стійкістю алгоритмів криптографічних перетворень, які в ній використовуються, й надійністю протоколів керування ключами [8].

У поняття «керування ключами» входить сукупність методів розв'язання таких задач, як

- генерування ключів;
- розподіл ключів;
- виведення з експлуатації та знищення ключів.

Правильне розв'язання названих задач має величезне значення, адже в більшості випадків зловмиснику легше атакувати саме ключову підсистему, а не алгоритм криптографічного захисту. Використання стійкого алгоритму шифрування є необхідною, але далеко не достатньою умовою побудови надійної системи криптографічного захисту інформації. Ключі, які використовуються в процесі інформаційного обміну, потребують однакового захисту на всіх стадіях життєвого циклу.

***Генерування ключів***

Безпека будь-якого криптографічного алгоритму визначається використовуваним криптографічним ключем. Цей ключ повинен бути досить довгим і мати випадкові значення бітів.

Як приклад розглянемо метод генерації сеансового ключа для симетричних криптосистем, який передбачає використання криптографічного алгоритму DES. Схема генерації випадкового сенсового ключа $R_i$ показана на рис. 7.1.

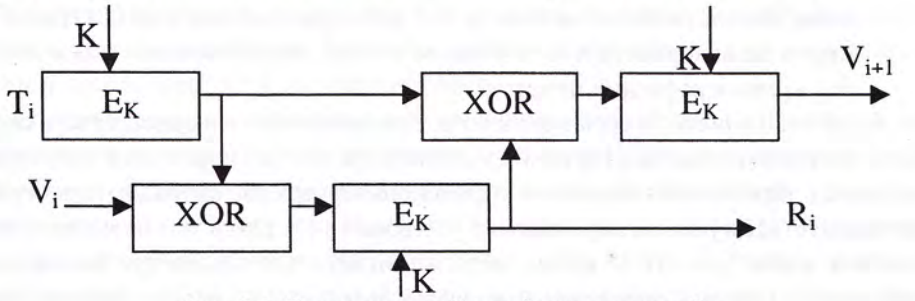



*Рис. 7.1. Схема генерації випадкового ключа $R_i$:*
*$E_K$ – результат шифрування алгоритмом DES значення X; K – ключ, зарезервований для генерації секретних ключів; $T_i$ – часова відмітка*

Випадковий ключ $R_i$ генерують, обчислюючи значення

$$R_i = E_K(E_K(T_i) \oplus V_i,$$

де $V_{i=0}$ – секретне 64-бітне початкове число.

Наступне значення $V_{i+1}$ обчислюють за формулою

$$V_{i+1} = E_K(E_K(T_i) \oplus R_i.$$

У разі необхідності 128-бітного ключа генерують пару ключів $R_i$, $R_{i+1}$ і об'єднують їх.

Для генерації випадкових значень ключів застовують різноманітні програмні й апаратні засоби. Оскільки ступінь випадковості генерації чисел повинен бути досить високим, то використовуються пристрої на основі «натуральних» випадкових процесів, наприклад на основі білого радіошуму. Ще одним прикладом може бути фізичний датчик випадкових чисел, вбудований в ядро процесора Pentium III. Іншим випадковим математичним об'єктом є десяткові знаки трансцендентних чисел, наприклад $\pi$ чи $e$, які обчислюються за допомогою стандартних математичних методів.

***Ієрархія ключів***

Є такі категорії ключів, що використовуються з різною метою [18]:

- Ключі шифрування даних. Ці ключі змінюються досить часто, щоби запобігти накопиченню великої кількості криптограм, зашифрованих на одному ключі. Це полегшує роботу криптоаналітиків супротивника.
- Ключі шифрування ключів. Ці ключі змінюють не так часто, як перші, й використовують лише для шифрування інформації, невеликої за об'ємом: криптографічні ключі, вектори ініціалізації тощо.
- Головний ключ. Цей ключ змінюється найрідше з усіх, тому що використовується лише для генерування інших криптографічних ключів. Виходячи з використання, засоби безпеки при зберіганні цих ключів повинні бути найвищими, оскільки компрометація головного ключа призводить до повного краху всієї криптографічної системи. Для більшої безпеки при зберіганні такого ключа часто використовують принцип «розділення секрету», коли ключ ділять на кілька частин, які або запам'ятовують різні люди, або частини ключа записують у різні пристрої зберігання. Отже, ключ ніколи не використовується з іншою метою; ніколи не зчитується інакше, як у криптографічний пристрій.

Існує також поняття сесійного ключа. Рекомендовано використовувати окремі ключі для шифрування зв'язку, який є в кожній окремій сесії (сесія може охоплювати, наприклад, пересилання масивів із віддаленого сервера або операцію користувача, яка виконується в рамках персональної реєстрації в системі з можливістю доступу багатьох користувачів). У цьому випадку визначення ключа для кожної сесії відбувається з використанням ключів шифрування ключів, а не поточного сесійного ключа. Застосування сесійних ключів потрібне з огляду на таке:

- це обмежує кількість криптограм, які генерують за допомогою одного ключа, чим обмежено кількість криптограм, які зловмисник може аналізувати;

– обмежено час, необхідний для відгадування ключа зловмисником. Після закінчення кожної сесії зловмисник не має змоги виконати активну атаку.

***Накопичення ключів***

Під накопиченням ключів розуміють процес їхнього зберігання, обліку і знищення. Накопиченню ключів слід приділяти особливу увагу, оскільки ключ є найпривабливішим об'єктом для зловмисника, який може відкрити доступ до конфіденційної інформації.

Секретні ключі ніколи не повинні зберігатися на носії, який можна зчитати чи скопіювати.

У досить складній інформаційній системі один користувач із великим об'ємом ключової інформації, й іноді навіть виникає необхідність організації міні-баз даних ключової інформації. Такі бази даних відповідають за прийняття, зберігання і знищення використовуваних ключів.

Уся інформація щодо використовуваних ключів повинна зберігатися в зашифрованому вигляді. Ключі, які зашифровують ключову інформацію, називаються майстер-ключами. Бажано, щоб майстер-ключі кожен користувач пам'ятав, а не зберігав на якихось матеріальних носіях.

Ще однією умовою безпеки інформації є періодичне обновлення ключової інформації й інформаційній системі. При цьому змінюватися повинні як звичайні ключі, так і майстер-ключі. В найважливіших інформаційних системах обновлення ключової інформації слід здійснювати щоденно.

***Розподіл ключів***

Розподіл ключів є найвідповідальнішим процесом у керуванні ключами. До нього ставляться такі вимоги:

1. Оперативність і точність розподілу.
2. Секретність процесу розподілу ключів.

У рамках симетричної (одноключової) системи шифрування двом користувачам, які бажають встановити безпечну взаємодію, необхідно спочатку встановити безпечний спільний ключ. Однією з можливостей є використання третьої сторони, наприклад кур'єра. На практиці, з метою безпеки необхідно час від часу змінювати ключ. Це може зробити використання кур'єра чи іншої подібної схеми дорогою й неефективною.

Альтернативою є отримання двома користувачами спільного ключа від центрального пристрою – центру розподілу ключів (ЦРК), з допомогою якого вони можуть безпечно взаємодіяти. Для організації обміну даними між ЦРК і користувачем останньому при реєстрації виділяється спеціальний ключ, яким шифруються повідомлення, що передаються між ними. Оскільки кожному користувачеві виділяється окремий ключ, його компрометація не призведе до особливо неприємних наслідків. Слабке місце цього підходу: у ЦРК, який володіє доступом до ключів, можливе проникнення зловмисника; внаслідок концентрації довіри одне порушення безпеки скомпрометує всю систему. Крім того, ЦРК може,

наприклад, довгий час брати участь у пасивному підслуховуванні, перш ніж це буде виявлено, хоча й довести це може бути дуже важко.

У великих мережах ця процедура може стати вузьким місцем, оскільки кожній парі користувачів, які потребують ключа, необхідно хоча б один раз звернутися до центрального вузла. Крім того, збій центрального пристрою може зруйнувати систему розподілу ключів. Ієрархічна система з користувачами, які знаходяться в підієрархіях, і центрами розподілу ключів у проміжних вузлах є одним із способів пом'якшення цієї проблеми. Це створює нову проблему безпеки, оскільки множина точок входу для зловмисника збільшується. До того ж дана система буде неефективною, якщо пара користувачів, які часто взаємодіють один з одним, не буде знаходитися в одному піддереві, оскільки в такому випадку корінь дерева знову виявиться вузьким місцем.

Розподіл ключів між користувачами реалізується двома різними підходами:

1. Шляхом створення одного чи декількох центрів розподілу ключів. Недоліком такого підходу є те, що центру розподілу відомо, кому і які ключі призначені, і це дозволяє читати всі повідомлення, які циркулюють в інформаційній системі. Можливі зловживання суттєво впливають на захист.
2. Прямий обмін ключами між користувачами інформаційної системи. В цьому випадку проблема полягає в тому щоб впевнитися в автентичності суб'єктів.

В обох випадках повинна бути гарантована достовірність сеансу зв'язку. Це можна забезпечити двома способами:

1. Механізм запиту-відповіді. Він полягає ось у чому: якщо користувач А хоче бути впевненим, що повідомлення, яке він отримав від В, нехибне, він додає до повідомлення, яке відправляє В, непередбачуваний елемент (запит). Під час відповіді В повинен виконати певну операцію над цим елементом (наприклад, додати 1). Це неможливо здійснити заздалегідь, бо невідомо, яке випадкове число прийде в запиті. Після отримання відповіді з результатами дій користувач А може бути впевнений, що сеанс достовірний. Недоліком цього методу є можливість встановлення, хоч і складної, закономірності між запитом і відповіддю.

2. Механізм відліку часу («часова мітка»). Він передбачає фіксацію часу для кожного повідомлення. Це дозволяє кожному суб'єктові мережі визначити часовий інтервал отриманого повідомлення, і відхилити його, якщо виникне сумнів у його достовірності. При використанні відліків часу необхідно встановити допустимий інтервал затримки.

В обох випадках для захисту елемента контролю використовують шифрування, щоб впевнитися, що відповідь відправлена не зловмисником і мітка часу не змінена.

Таким чином, задача розподілу ключів зводиться до побудови протоколу розподілу ключів, який би забезпечував:

- взаємне підтвердження достовірності учасників сеансу;
- підтвердження достовірності сеансу механізмом запиту-відповіді чи відліку часу;

– використання мінімальної кількості повідомлень при обміні ключами;
– можливість вилучення зловживань зі сторони центру розподілу ключів.

У зв'язку з цим в основу розв'язання задачі розподілу ключів доцільно покласти принцип відділення процедури ідентифікації партнера від процедури розподілу ключів. Метою такого підходу є створення методу, при якому після встановлення автентичності учасники самі формують сеансовий ключ без участі ЦРК задля того, щоб розподілювач ключів не мав можливості «прочитати» зміст повідомлень.

*Симетричні криптосистеми.* В криптосистемах із відкритим ключем використовуються два різних ключі – публічний і приватний. Обчислювальними методами дуже важко визначити приватний ключ за публічним.

Розглянемо процес організації зв'язку між користувачами з допомогою криптосистеми з відкритим ключем:

1) А і В домовляються використовувати криптосистему з публічним ключем;
2) В відправляє А свій публічний ключ;
3) А шифрує своє повідомлення, використовуючи публічний ключ В, і відправляє його В;
4) В розшифровує повідомлення А своїм приватним ключем.

Зрозуміло, що криптографія з публічним ключем усуває актуальну для симетричних криптосистем проблему поширення ключів. При використанні криптосистеми із симетричної А і В повинні були розв'язати проблему таємного розподілу секретного ключа. Криптографія з відкритим ключем значно спрощує цю задачу: А може відправити В секретне повідомлення без будь-якої попередньої підготовки.

Але в цьому випадку використовувана криптосистема повинна заздалегідь узгоджуватися з користувачами, тобто в кожного з них є відкритий і закритий ключі.

*Комбіновані криптосистеми.* Варто зазначити, що алгоритми з відкритим ключем не заміняють симетричні алгоритми. Вони використовуються для шифрування не самих повідомлень, а ключів. На це є дві причини:

1. Алгоритми з відкритим ключем виконують повільно.

2. Криптосистеми з відкритим ключем уразливі до атак на основі підібраного відкритого тексту. Якщо $C = E(M)$, де $M$ – відкритий текст із множини $n$ можливих відкритих текстів, зловмиснику достатньо зашифрувати всі $n$ можливих відкритих текстів і порівняти результати з $C$ (адже ключ зашифрування відкритий). Отже, він не зможе встановити ключ розшифрування, але зуміє визначити $M$.

Криптоаналіз на основі підібраного відкритого тексту особливо ефективний, якщо кількість можливих криптограм цього повідомлення відносно невелика. Симетричні криптосистеми невразливі до дій такого типу, оскільки зловмисник, не знаючи ключ, не зможе виконувати пробні шифрування. Тобто в більшості випадків практичної реалізації криптографія з відкритим ключем застовується тільки для

засекречування й поширення сеансових ключів, а сеансові ключі використовуються симетричними алгоритмами для захисту трафіку повідомлень. Така організація комунікаційного зв'язку називається комбінованою криптосистемою. Розглянемо процес реалізації протоколу обміну повідомленнями між користувачами з використанням комбінованої криптосистеми.

1) В відправляє А свій відкритий ключ;
2) А генерує випадковий сеансовий ключ, шифрує його з допомогою відкритого ключа В і відправляє В;
3) використовуючи свій закритий ключ, В розшифровує повідомлення А, відновлюючи сеансовий ключ;
4) обидві сторони шифрують свої повідомлення з допомогою однакового сеансового ключа.

Використання криптографії з відкритим ключем для поширення ключів розв'язує цю дуже важливу проблему. При використанні цього протоколу для зашифрування повідомлення створюється сеансовий ключ, який після закінчення сеансу зв'язку знищується. Це різко знижує небезпеку компрометації сеансового ключа.

*Асиметричні криптосистеми.* Сертифікатом відкритого ключа називається повідомлення ЦРК, яке засвідчує цілісність деякого відкритого ключа об'єкта. Наприклад, сертифікат відкритого ключа користувача *A* ($C_A$) містить відмітку часу *t*, ідентифікатор $Id_A$, відкритий ключ $K_A$, зашифровані секретним ключем ЦРК $k_{ЦРК}$, тобто

$$C_A = E_{k_{ЦРК}}(t, Id_A, K_A).$$

Секретний ключ $k_{ЦРК}$ відомий тільки менеджеру ЦРК. Відкритий ключ $K_{ЦРК}$ відомий учасникам А і В. Об'єкт учасника А викликається й ініціює стадію встановлення ключа, після чого запитує у ЦРК сертифікат свого відкритого ключа і відкритого ключа учасника В, тобто А відсилає ЦРК $Id_A$ і $Id_B$ (унікальні ідентифікатори відповідно учасників А і В). Менеджер ЦРК відповідає таким повідомленням:

$$E_{k_{ЦРК}}(t, Id_A, K_A), E_{k_{ЦРК}}(t, Id_B, K_B).$$

Учасник А, використовуючи відкритий ключ ЦРК $K_{ЦРК}$, розшифровує відповідь ЦРК і перевіряє обидва сертифікати. Ідентифікатор $Id_B$ переконує А, що особистість правильного учасника зафіксована в ЦРК і $K_B$ – дійсно відкритий ключ учасника В, оскільки обидва зашифровані ключем $k_{ЦРК}$.

Наступний крок протоколу полягає у встановленні зв'язку А з В:

$$C_A, E_{k_A}(t), E_{K_B}(r_1).$$

Тут $C_A$ – сертифікат відкритого ключа користувача А; $E_{k_A}(t)$ – відмітка часу, зашифрована секретним ключем учасника А, і є підписом учасника А, оскільки ніхто інший не може створити такий підпис; $r_1$ – випадкове число, яке генерує учасник А і використовує для обміну з В у ході процедури перевірки достовірності.

Якщо сертифікат $C_A$ і підпис А правильні, то учасник В упевнений, що повідомлення прийшло від А. Частину повідомлення $E_{K_B}(r_1)$ може розшифрувати тільки В, бо більше ніхто не знає секретного ключа $k_B$, який відповідає відкритому ключу $K_B$. Користувач В розшифровує значення числа $r_1$ і, щоб підтвердити свою достовірність, відправляє учаснику А повідомлення: $E_{K_A}(r_1)$.

Учасник А відновлює значення $r_1$, розшифровуючи це повідомлення з використанням свого секретного ключа $k_A$. Якщо це очікуване значення $r_1$, то А отримує підтвердження, що визваний учасник дійсно В.

Якщо множина користувачів відкритого ключа невелика, то такий ключ можна поширити ручним способом, записуючи його на будь-який носій інформації (наприклад, на дискету). Далі варто захистити його від модифікації, застосовуючи, наприклад, організаційні заходи захисту. Проте такий спосіб поширення відкритих ключів неприйнятний для великих систем, особливо для розподілених у просторі. На початку 80-х років XX століття було запропоновано поняття «сертифікат відкритого ключа», використане потім у нормативних документах з інформаційних технологій, першим з яких був ITU X.509, що стосується забезпечення автентичності в електронних довідниках. Сертифікат є документом (у цьому випадку електронним), де розміщується інформація, автентичність якої гарантується органом, що видав цей сертифікат. Саме з цих позицій і варто оцінювати сертифікат відкритого ключа. Характер інформації, поданої в сертифікаті, визначає його формат. Сьогодні найпоширеніший формат X.509, який має три версії (відповідно 1988, 1993 і 1997 рр.). Перша версія сертифіката містила в собі такі поля:

- номер версії;
- порядковий номер сертифіката;
- ідентифікатор алгоритму, використовуваного для цифрового підпису сертифіката;
- найменування (у форматі X.500) органу, що видав сертифікат;
- тепмін придатності сертифіката (дати початку і кінця дії);
- ім'я (у форматі X.500) власника сертифіката;
- відкритий ключ;
- цифровий підпис органу сертифікації.

У 1993 р. було додано два поля, що є необов'язковими; вони містять відповідно ідентифікатори установи, яка видала сертифікат, і власника сертифіката. Пізніше (1997) було вирішено ввести поле «Розширення», кількість підполів якого не фіксується. Воно призначене для подальшої деталізації інформації, що стосується як власника сертифіката, так і сфери застосування даного сертифіката.

Як бачимо, крім числового значення відкритого ключа, його сертифікат містить значну кількість інформації, в тому числі і про власника інформації, що сприяє застосуванню асиметричних криптосистем у системах електронного документообігу.

Сертифікат має такі властивості:

- кожен користувач, що має доступ до публічного ключа центру сертифікації, який видав той чи інший сертифікат, може отримати з цього сертифіката публічний ключ користувача та перевірити його аутентичність;
- жодна зі сторін (окрім центру сертифікації) не може непомітно змінити сертифікат (сертифікат неможливо підробити).

Створення сертифіката починається зі створення пари ключів (приватний-публічний). Це може робити або центр сертифікації, або власне користувач. В останньому випадку публічний ключ надається центру сертифікації. Після цього центр сертифікації генерує сертифікат, який розміщується в Інтернет для того, щоби всі учасники інформаційного обміну могли ним користуватися.

Розглянемо ієрархічну модель сертифікації.

Глобальними, або, як їх ще називають, ***кореневими центрами сертифікації,*** виступають спеціально створені структури, яким довіряють усі без винятку підлеглі центри. Такі структури мають найпотужніші можливості генерування,

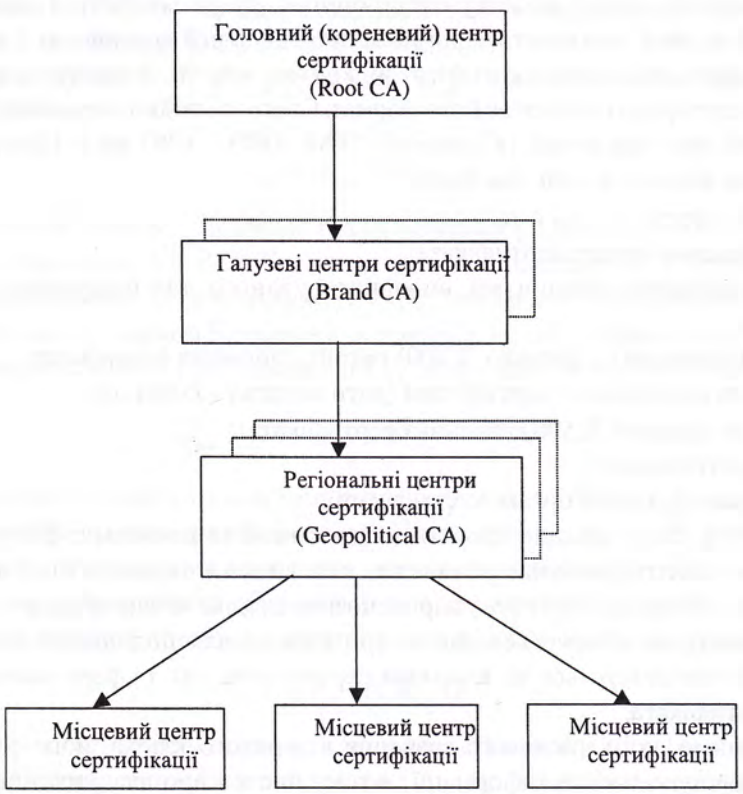



*Рис. 7.2. Ієрархічна модель інфраструктури публічних ключів*

зберігання ключів і сертифікації. Прикладом може служити спеціально створена компанія Verisign. Кореневі центри не працюють безпосередньо з користувачами, а лише з підпорядкованими їм центрами сертифікації та їх сертифікатами.

***Галузеві центри сертифікації*** (або *Brand Certification Authority* – галузеві сертифікаційні авторитети), наприклад фірма Microsoft, сертифікує підпорядковані центри та окремих крупних клієнтів.

На регіональному рівні галузеві центри сертифікації делегують свої права ***регіональним центрам***, які в свою чергу, організовують цілу мережу ***місцевих центрів сертифікації***. Останні вже безпосередньо працюють із місцевими користувачами. Вони генерують пари ключів, забезпечують їх розповсюдження та підтримку сертифікатами. Сертифікати публічних ключів розміщуються на серверах місцевих або, найчастіше, регіональних центрів і забезпечуються захистом від підміни.

Логіка роботи користувачів з сертифікатами публічних ключів приблизно така:

- користувач, отримавши сертифікат партнера, бачить, що його підписав незнайомий центр сертифікації;
- користувач просить партнера надати йому сертифікат цього центру;
- отримавши сертифікат, користувач перевіряє його автентичність за допомогою сертифіката центрального ЦС;
- у випадку успішної перевірки, він починає довіряти цьому ЦС.

Сукупність центрів сертифікації утворює так звану ***інфраструктуру публічних ключів*** *(Public Key Infrastructures – PKI)*, що забезпечує створення та функціонування в системах сертифікатів публічних ключів. Зокрема, нею визначається статус сертифікатів, використання яких може бути як заблоковане, так і анульоване. Необхідно забезпечити постійний доступ до інформації, що стосується сертифікатів публічних ключів, а також можливість відшкодування збитку, завданого внаслідок виявлених вад у системі сертифікації. Значне поширення інформаційних технологій зумовило потребу в наданні електронним документам юридичної чинності. Раніше було згадано про роль цифрового підпису в цьому явищі. У прийнятих на сьогодні відповідних законах велику увагу приділено правовим питанням діяльності органів сертифікації публічних ключів.

Із технічної (і правової) точки зору необхідно звернути увагу на такий аспект проблеми поширення публічних ключів. Через постійне збільшення кількості користувачів асиметричних криптосистем не видається можливим функціонування єдиного для всього населення земної кулі органу сертифікації відкритих ключів. Отже, *PKI* та відповідні закони призначені для організації взаємодії різних органів і систем сертифікації відкритих ключів.

Таким чином, ми можемо визначити основні завдання інфраструктури публічних ключів:

- підтримка життєвого циклу ключів і сертифікатів (тобто генерування ключів, створення та підписування сертифікатів, їх розповсюдження, захист тощо);

- реєстрування фактів компрометації ключів і публікування «чорних» списків відізваних сертифікатів;
- підтримка процесів ідентифікації та аутентифікації користувачів за допомогою сертифікатів;
- реалізація механізму інтеграції існуючого програмного забезпечення безпеки на основі сервісів PKI.

***Протокол прямого обміну ключами***

Два користувачі, які бажають обмінятися криптографічно захищеною інформацією, при використанні для інформаційного обміну криптосистеми з симетричним секретним ключем повинні мати загальний секретний ключ і обмінятися ним по каналу зв'язку в безпечний спосіб. Якщо ключ змінюється досить часто, то його доставка перетворюється в серйозну проблему.

Є два способи розв'язання цієї проблеми:

– використання асиметричної криптосистеми з відкритим ключем для шифрування і передачі секретного ключа симетричної криптосистеми;
– використання системи відкритого розподілу ключів Діффі-Хелмана.

Реалізація першого способу, який ще іноді називають способом електронного цифрового «конверта», здійснюється в рамках комбінованої криптосистеми з симетричними й асиметричними ключами. При такому підході симетрична криптосистема застосовується для шифрування й передачі відкритого тексту, а асиметрична криптосистема з відкритим ключем – для шифрування, передачі й розшифрування тільки секретного ключа симетричної криптосистеми.

Як приклад реалізації способу електронного цифрового «конверта» розглянемо порядок роботи комбінованої критпосистеми із симетричними й асиметричними ключами із застосуванням електронного цифрового підпису і сертифікатів відкритих ключів:

1. Безпечно створюються і поширюються асиметричні публічні й приватні ключі. Закритий асиметричний ключ передається його власнику. Відкритий асиметричний ключ зберігається в базі даних відкритих ключів і адмініструється центром видачі сертифікатів ЦРК. Користувачі повинні довіряти такій системі, де проводиться безпечне створення, розподіл і адміністрування ключів.

2. Створюється електронний цифровий підпис тексту з допомогою обчислення його хеш-функції. Отримане значення шифрується з використанням асиметричного закритого ключа відправника, а потім отриманий рядок символів додається до тексту, який передається (тільки відправник може створити електронний підпис).

3. Створюється секретний симетричний ключ, який буде використовуватися для шифрування тільки даного повідомлення чи сеансу взаємодії (сеансовий ключ); потім за допомогою цього ключа й симетричного алгоритму шифрування шифрується вихідний текст разом із доданим до нього цифровим підписом. У такий спосіб отримується зашифрований текст.

4. Далі необхідно розв'язати проблему з передачею сеансового ключа отримувачу.

5. Відправник повинен мати асиметричний відкритий ключ центру видачі сертифікатів $K_{ЦРК}$. Перехоплення незашифрованих запитів на отримання цього відкритого ключа – це найпоширеніша форма атаки. Може існувати ціла система сертифікатів, які підтверджують достовірність відкритого ключа $K_{ЦРК}$.

6. Відправник запитує у ЦРК асиметричний відкритий ключ отримувача повідомлень. Цей процес вразливий до атаки, під час якої атакуючий суб'єкт втручається у взаємодію між відправником та отримувачем і може модифікувати трафік, який передається між ними. Тому відкритий асиметричний ключ отримувача «підписується» ЦРК. Це означає, що $K_{ЦРК}$ використав свій асиметричний секретний ключ для шифрування асиметричного ключа отримувача. Тільки ЦРК знає асиметричний секретний ключ $K_{ЦРК}$. Це гарантія того, що публічний асиметричний ключ отримувача одержаний саме від ЦРК.

7. Після отримання асиметричний відкритий ключ отримувача розшифровується з допомогою асиметричного відкритого ключа $K_{ЦРК}$ і алгоритму асиметричного шифрування/розшифрування.

8. Відправник шифрує сеансовий ключ з використанням асиметричного ключа отримувача. Цей ключ отримується від ЦРК і розшифровується.

9. Зашифрований сеансовий ключ приєднується до зашифрованого тексту, який містить раніше доданий електронний підпис.

10. Сформований електронний цифровий «конверт» відправляється отримувачу.

11. Отримувач виділяє зашифрований сеансовий ключ із присланого цифрового «конверта» і розв'зує проблему з розшифровкою сеансового ключа.

12. Використовуючи свій секретний асиметричний ключ і такий самий асиметричний алгоритм шифрування, отримувач розшифровує сеансовий ключ.

13. Отримувач застосовує до одержаного зашифрованого тексту розшифрований симетричний (сеансовий) ключ і такий самий симетричний алгоритм шифрування/розшифрування й отримує вихідний текст разом з електронним підписом.

14. Отримувач відділяє електронний підпис від вихідного тексту.

15. Отримувач запитує у ЦРК асиметричний відкритий ключ відправника.

16. Після отримання ключа отримувач розшифровує його з допомогою відкритого $K_{ЦРК}$ і відповідного асиметричного алгоритму шифрування/розшифрування.

17. Потім розшифровується хеш-функція тексту з використанням відкритого відправника й асиметричного алгоритму шифрування/розшифрування.

18. Повторно обчислюється хеш-функція отриманого тексту. Дві ці хеш-функції порівнюються, щоб переконатися, що вихідний текст не було змінено.

Інший спосіб безпечного поширення секретних ключів базується на застосуванні алгоритму відкритого розподілу ключів Діффі-Хелмана. Цей алгоритм дозволяє користувачам обмінюватися ключами незахищеними каналами зв'язку.

***Протокол Діффі-Хеллмана***

Алгоритм відкритого розподілу ключів, винайдений В. Діффі та М. Хеллманом, дозволяє користувачам обмінюватися ключами по незахищених

каналах зв'язку. Його безпека зумовлена важкістю обчислення дискретних логарифмів у кінцевому полі, на відміну від легкості розв'язання прямої задачі дискретного піднесення до степеня в тому ж кінцевому полі.

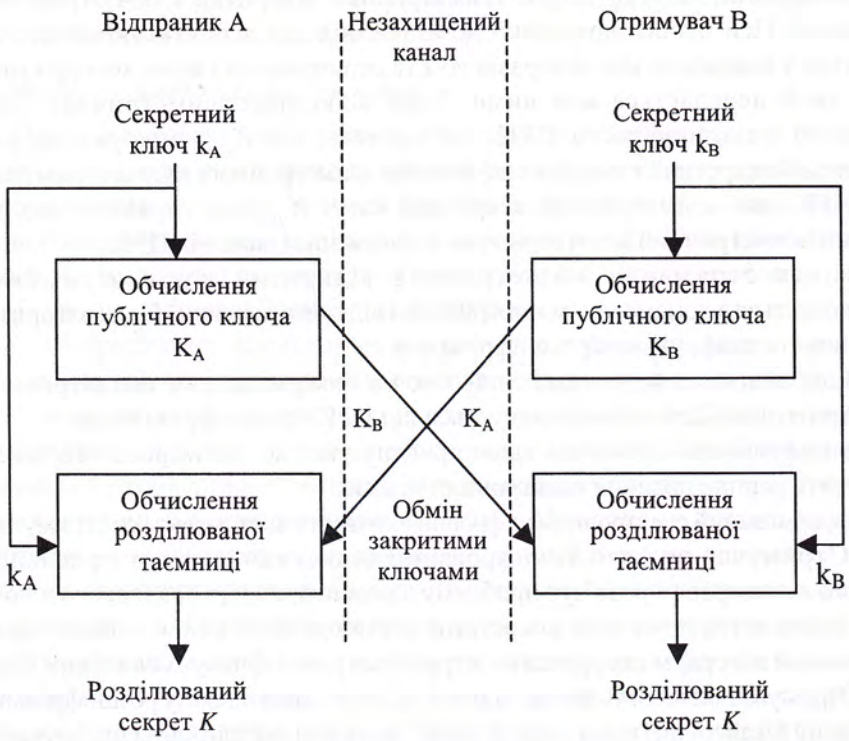



*Рис. 7.3. Схема відкритого розподілу ключів Діффі-Хеллмана*

Принцип алгоритму Діффі-Хеллмана полягає ось у чому (рис. 7.3).

Користувачі A і B, які беруть участь в обміні інформації, генерують незалежно один від одного свої випадкові секретні ключі $k_A$ і $k_B$ ($k_A$ і $k_B$ – випадкові великі числа, які зберігаються користувачами в таємниці).

Потім користувач A обчислює на основі свого секретного ключа $k_A$ публічний ключ:

$$K_A = g^{k_A} (\mathrm{mod}\, N).$$

Одночасно користувач B обчислює на основі свого секретного ключа $k_B$ публічний ключ:

$$K_B = g^{k_B} (\mathrm{mod}\, N).$$

Тут $N$ і $g$ – великі цілі прості числа. Арифметичні дії виконуються зі зведенням за модулем $N$. Числа $N$ і $g$ можуть не зберігатися в таємниці. Як правило, ці значення спільні для всіх користувачів мережі чи системи.

Потім користувачі А і В обмінюються своїми публічними ключами $K_A$ і $K_B$ по незахищеному каналу і використовують їх для обчислення загального сесійного ключа $K$.

користувач А: $K = (K_B)^{k_A} \equiv (g^{k_B})^{k_A} (\mathrm{mod}\, N)$;

користувач В: $K' = (K_A)^{k_B} \equiv (g^{k_A})^{k_B} (\mathrm{mod}\, N)$.

При цьому $K = K'$, оскільки

$$(g^{k_B})^{k_A} (\mathrm{mod}\, N) = (g^{k_A})^{k_B} (\mathrm{mod}\, N).$$

Отже, результатом цих дій є спільний сесійний ключ, який є функцією обох секретних ключів $k_A$ і $k_B$.

Зловмисник, перехопивши значення відкритих ключів $K_A$ і $K_B$, не може обчислити сесійний ключ $K$, тому що він не має відповідних значень секретних ключів $k_A$ і $k_B$. Завдяки використанню однонапрямленої функції операція обчислення відкритого ключа незворотна, тобто за значенням публічного ключа абонента неможливо обчислити його секретний ключ.

Унікальність алгоритму Діффі-Хеллмана полягає в тому, що пара абонентів має можливість отримати відоме тільки їм секретне число, передаючи відкритою мережею публічні ключі. Після цього абоненти можуть приступити до захисту передавальної інформації вже відомим перевіреним способом – застосовуючи симетричне шифрування з використанням отриманої розділюваної таємниці.

Схема Діффі-Хеллмана дає можливість шифрувати дані при кожному сеансі зв'язку на нових ключах. Це дозволяє не зберігати таємниці на дискетах чи інших носіях. Не варто забувати, що подібне зберігання таємниць підвищує ймовірність їх попадання в руки конкурентів чи зловмисників.

Протокол Діффі-Хеллмана дозволяє також реалізувати метод комплексного захисту конфіденційності й автентичності даних, які передаються. Алгоритм дає користувачу можливість сформувати та використати один ключ для виконання цифрового підпису і симетричного шифрування даних.

Для одночасного захисту цілісності та конфіденційності даних варто застосовувати шифрування й електронний підпис у комплексі. Проміжні результати роботи алгоритму Діффі-Хеллмана можуть бути використані як вихідні дані для реалізації методу комплексного захисту цілісності й конфіденційності передавальних даних. Справді, згідно з даним алгоритмом, користувачі А і В спочатку генерують свої секретні ключі $k_A$ і $k_B$ та обчислюють свої публічні ключі і $K_B$. Потім абоненти А і В використовують ці проміжні результати для одночасного обчислення загального секретного ключа $K$, який може використовуватися для симетричного шифрування даних.

Метод комплексного захисту конфіденційності й аутентичності передавальних даних працює за такою схемою:

– абонент А підписує повідомлення $M$ за допомогою свого секретного ключа $k_A$, використовуючи стандартний алгоритм цифорового підпису;

– абонент А обчислює сумісно розділюваний ключ $K$ за алгоритмом Діффі-Хеллмана зі свого секретного ключа $k_A$ і відкритого ключа $K_B$ абонента В;

– абонент А зашифровує повідомлення $M$ на отриманому сумісно розділюваному секретному ключі $K$, використовуючи узгоджений із партнером по обміну алгоритм симетричного шифрування;

– абонент В при отриманні зашифрованого повідомлення $M$ обчислює за алгоритмом Діффі-Хеллмана сумісно розділюваний секретний ключ $K$ зі свого секретного ключа $k_B$ і публічного ключа $K_A$ абонента А;

– абонент В розшифровує отримане повідомлення $M$ на ключі $K$;

– абонент В перевіряє підпис розшифрованого повідомлення $M$ із допомогою публічного ключа абонента $K_A$.

***Протокол обчислення ключа парного зв'язку ECKEP***

Протокол ECKEP (Elliptic Curve Key Establishment Protocol) призначений для організації захищеного комунікаційного каналу. Припустимо, що для конфіденційного інформаційного обміну користувачі А і В мають обмінятися спільним секретним ключем по незахищених каналах зв'язку. При цьому користувачі повинні мати свої секретний і відкритий (публічний) ключі. Користувач А має секретний ключ $K_{cA}$ і відкритий ключ $K_{вA} = K_{cA}$ $P = (x_A, y_A)$. Аналогічно користувач В має секретний ключ $K_{cB}$ і відкритий ключ $K_{вB} = K_{cB}$ $P = (x_B, y_B)$, де $P$ – вибрана точка еліптичної кривої.

Обчислення ключа парного зв'язку проходить у чотири етапи:

1. Дії користувача А:
   - вибирає випадкове ціле число $k_A$, $l \le k_A \le n-1$;
   - обчислює $R_A = k_A P$;
   - обчислює $(x_1, y_1) = k_A K_{вB}$;
   - обчислює $s_A = k_A + K_{cA} x_A x_1 (\mathrm{mod}\, n)$;
   - $R_A$ відправляється користувачу В.
2. Дії користувача В:
   - вибирає випадкове ціле число $k_B$, $l \le k_B \le n-1$;
   - обчислює $R_B = k_B P$;
   - обчислює $(x_2, y_2) = k_B K_{вA}$;
   - обчислює $s_B = k_B + K_{cB} x_B x_2 (\mathrm{mod}\, n)$;
   - $R_B$ відправляє користувачу А.
3. Дії користувача А:
   - обчислює $(x_2, y_2) = K_{cA} R_B$;
   - обчислює ключ парного зв'язку $K = s_A (R_B + x_B x_2 K_{вB})$.
4. Дії користувача В:
   - обчислює $(x_1, y_1) = K_{cB} R_A$;
   - обчислює ключ парного зв'язку $K = s_B (R_A + x_A x_1 K_{вA})$, що еквівалентно значенню $s_A (R_B + x_B x_2 K_{вB})$.

Важливою перевагою схеми розподілу ключів Діффі-Хеллмана і протокола обчислення парного зв'язку ECKEP є те, що вони дозволяють обійтися без захищеного каналу для передачі ключів. Проте потрібно мати гарантію того, що

користувач А отримав публічний ключ саме від користувача В, і навпаки. Ця проблема розв'язується за допомогою сертифікатів відкритих ключів у рамках інфраструктури керування відкритими ключами PKI [22].

# Розділ VIII
# Проблеми генерування випадкових і псевдовипадкових послідовностей

Генерування випадкових і псевдовипадкових послідовностей є необхідним і надзвичайно важливим елементом криптографічних застосувань. Більше того, від якості генераторів, апаратних чи програмних, багато в чому залежить якість самої криптосистеми. Неякісний генератор випадкових послідовностей може піддаватися атакам зловмисників [14], бути причиною утворення слабких криптографічних ключів. Тому важливість якісного генерування випадкових криптографічностійких послідовностей важко переоцінити.

Криптографічні послідовності використовують у разі:

- генерування ключів симетричних та асиметричних криптосистем;
- генерування цифрових підписів;
- реалізації переважної кількості криптографічних протоколів;
- аутентифікації, що ґрунтується на криптографічних засобах;
- потокового шифрування;
- інших криптографічних застосувань.

Послідовності можна розділити на два класи: ***істинно випадкові*** (далі будемо називати їх ***випадковими***) та ***псевдовипадкові***.

Випадкові послідовності, як правило, породжуються апаратними або комбінованими, програмно-апаратними засобами, які використовують як генеруючий пристрій датчики фізичних величин. Дані, які постачають датчики фізичних величин, обробляються або електронними засобами, або програмними рішеннями, які, власне, й виробляють двійкові послідовності.

Псевдовипадкові послідовності, як правило, генеруються програмно.

Відмінності між істинно випадковими та псевдовипадковими послідовностями дуже значні. Основна відмінність – найважливіша, полягає в тому, що псевдовипадкова послідовність – періодична. Іншою суттєвою відмінністю є те, що за однакових початкових умов псевдовипадкова послідовність також буде однаковою.

Цих два недоліки псевдовипадкових послідовностей необхідно враховувати при проектуванні криптографічних систем, що використовують такі генератори.

Проектування криптографічно стійких псевдовипадкових послідовностей – дуже важлива криптографічна задача, що стала вже цілою індустрією. Прийнято вважати [18], що стійкий псевдовипадковий генератор, який можна сміливо використовувати в найскладніших криптографічних системах, повинен мати період не менший за $2^{256}$.

### *Генератори випадкових послідовностей*

На ринку немає дешевих апаратних засобів генерування справді випадкових послідовностей, що використовують результати вимірювання певної фізичної

величини. Як правило, для побудови таких генераторів можна використати датчик практично довільної фізичної величини, що вимірює її значення з великою точністю:

- температури оточуючого середовища;
- рівня радіоактивності;
- рівня сейсмічної активності;
- рівня освітленості;
- опору еталонного резистора;
- інших фізичних величин.

Схема обробки опитує датчик із деякою періодичністю та використовує отримані дані для формування двійкової випадкової послідовності.

У подальшому сигнал може використовувати програмна (або апаратна) реалізація криптографічного пристрою для різних використань.

Перевагами фізичних генераторів випадкових послідовностей можна вважати:

1) відсутність періодичності послідовності, що генерується;
2) неможливість отримання однакових послідовностей навіть при однакових зовнішніх умовах.

Отже, проектування та реалізація генератора випадкових послідовностей такої схеми – складна і нетривіальна інженерна задача, причому найскладнішою частиною можна вважати саме датчик фізичної величини, який повинен вимірювати цю величину з максимально можливою точністю.

Однак як датчик можна використати будь-який комп'ютер.

Непоганими схемами вважається генерування випадкових послідовностей з використанням клавіатури або мишки [21].

Перша схема працює так. Програмне забезпечення «просить» користувача набирати на клавіатурі будь-яку інформацію (краще беззмістовний набір символів) деякий проміжок часу. ПЗ аналізує дії користувача і може використовувати їх, як мінімум, у два різних способи. Перший спосіб полягає в тому, що для генерування випадкової послідовності використовуються проміжки часу, виміряні або в «тіках» процесора, або в мілісекундах, між послідовними натисканнями на клавіші. Цей час у двійковому представленні обробляється в певний спосіб (наприклад, виконується додавання всіх розрядів за модулем 2) і на вихід подається черговий біт послідовності. При такому використанні не має значення, що саме набирає користувач на клавіатури. Другий спосіб обробки отриманої інформації полягає в тому, що ПЗ використовує саме символи, які набирає користувач. У цьому разі також буде краще, якщо користувач набиратиме невпорядкований набір символів. Двійкове представлення чергового символу піддається деякій обробці (в найпростішому випадку це може бути та сама сума всіх розрядів за модулем 2 чи вибірка молодших або старших розрядів цього представлення з наступним їх додаванням за модулем 2 тощо) та на вихід подається біт (або кілька бітів) випадкової двійкової послідовності.

Мишку можна використати в такий спосіб. При необхідності генерування випадкової послідовності, наприклад для створення криптографічного ключа, на екран комп'ютера виводиться форма, на якій у випадкових місцях з'являється об'єкт. Користувача просять клацнути мишкою по цьому об'єктові. ПЗ вимірює проміжки часу між послідовними клацаннями мишкою та, обробляючи двійкове представлення цього проміжку, генерує черговий біт випадкової послідовності. Після генерування потрібної кількості двійкових знаків форма зникає.

Перевагами таких способів генерування двійкових послідовностей є відносна простота їх реалізації. Для цього, крім комп'ютера, програмного забезпечення та користувача нічого не потрібно. Водночас якість таких послідовностей практично ідентична фізичним генераторам випадкових послідовностей.

До недоліків таких способів можна віднести неможливість (або принаймні практичну складність) отримання безмежних послідовностей для потокового шифрування, хоча для інших цілей обидва способи достатньо зручні.

***Генерування псевдовипадкових послідовностей***

На перший погляд, як псевдовипадкову послідовність можна використати таку, яка генерується програмним забезпеченням для програмування на зразок вбудованих у мови програмування генераторів. Однак такі послідовності не будуть криптографічно стійкими, тобто вони не можуть бути використані у криптографії. Власне кажучи, вони взагалі не вважаються випадковими, оскільки період їх дуже малий. Наприклад, у популярній мові програмування C++ функція RAND() генерує «випадкові» значення за формулою $N_{i+1} \equiv (N_i \times 1103515245+12345) \bmod 4294967296$ [14].

Окрім цього, що ще важливіше, вони на різних комп'ютерах при однакових вхідних умовах обов'язково дають однакові послідовності. Це абсолютно непридатне для криптографічних застосувань, адже зловмисник може запустити цей генератор на своєму комп'ютері, змоделювавши ті самі вхідні умови, та отримати таку ж послідовність, зламавши в такий спосіб усю криптосистему.

Таким чином, для усунення такого недоліку ми повинні накласти додаткові умови на генератори псевдовипадкових послідовностей при криптографічних застосуваннях.

*По-перше*, необхідно, щоби псевдовипадкові послідовності якнайменше відрізнялися від справді випадкових [4]. Це означає таке. Нехай є дві бінарні послідовності, $X_n$ та $Y_n$, а також $A$ – ймовірнісний алгоритм, який, отримавши на вході двійкову послідовність, видає на виході один двійковий біт. Нехай $[P(X_n)=1]$ – імовірність того, що, отримавши на вході двійкову послідовність $X_n$, алгоритм на виході видає 1.

Вважається, що послідовності $X_n$ та $Y_n$ не розрізняються за поліноміальний час, якщо справджується нерівність

$$[P(X_n)=1] - [P(Y_n)=1] < 1/n^C$$

для довільної сталої $C$ за достатньо великих $n$. Це й буде означати, що псевдовипадкова послідовність відрізняється від випадкової неістотно.

*По-друге*, псевдовипадкова послідовність не повинна бути передбачуваною. Це означає, що знання всіх попередніх бітів послідовності не дозволяє передбачити наступний біт з імовірністю, більшою за 50%.

Наприклад, якщо ми використовуємо лінійний конгруентний генератор, коли кожен біт послідовності генерується з використанням рівняння $s_i \equiv (as_{i-1}+b) \bmod m$; $m = 2n$; $a \bmod m \equiv 1$, де $a$, $b$, $s_0$ – зародки, знання яких дозволить обчислити будь-який елемент послідовності. Така послідовність, за умови знання зародків, абсолютно передбачувана. Зародки можна визначити, знаючи кілька елементів послідовності та розв'язавши просту систему лінійних рівнянь. Саме тому лінійні й інші поліноміальні конгруентні генератори псевдовипадкових послідовностей не використовуються для потреб криптографії.

***Лінійний регістр зсуву зі зворотними зв'язками***

Послідовності регістрів зсуву зі зворотними зв'язками (Linear Feedback Shift Register - LFSR) використовуються як у криптографії, так і в теорії кодування. Їх теорія прекрасно розроблена, а потокові шифри на їхній основі були «робочою конячкою» військової криптографії задовго до появи електроніки [14].

Лінійний регістр зсуву зі зворотними зв'язками – це пристрій, що складається з регістра зсуву, здатного запам'ятовувати двійкові послідовності кінцевої довжини та схеми, яка реалізує додавання за модулем 2 вибраних бітів із регістра. Ця схема здійснює зворотний зв'язок (див. рис. 8.1).

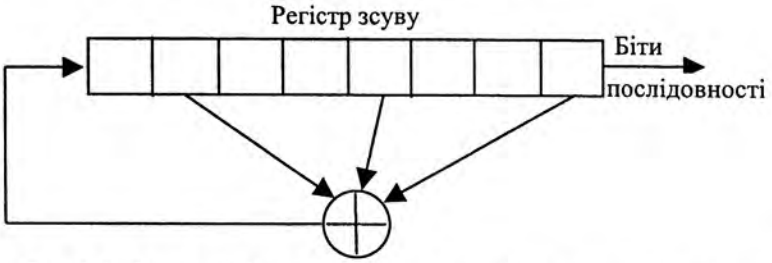



*Рис. 8.1. Схема лінійного регістра зсуву зі зворотними зв'язками*

Зародок послідовності визначається породжувальним поліномом з коефіцієнтами у двійковому представленні. Рівняння визначає зв'язок між тригерами регістра зсуву та схемою додавання за модулем 2. Якщо відповідний коефіцієнт дорівнює 1, то зв'язок є, якщо ж 0 – зв'язку немає. Період повторення послідовностей визначається кількістю тригерів у регістрі $k$ і може досягати $2^k$-1 для нерозкладного поліному ступеня $k$.

Породжувальний поліном має такий загальний вигляд:

$$A_n X^n + A_{n-1} X^{n-1} + \ldots + A_2 X^2 + A_1 X + A_0,$$

причому $A_n = A_0 = 1$, а інші дорівнюють або 0, або 1.

Породжувальні поліноми мають дві властивості:

1) не діляться ні на який інший поліном;
2) ділять поліном $X^{n+1}+1$.

Лінійний регістр зсуву будується так:

- виходячи з потрібного періоду генерації знаходиться породжувальний поліном потрібного степеня;
- будується регістр зсуву відповідної розрядності (в разі необхідності кілька регістрів з'єднуються каскадно);
- виходи тих тригерів регістру, степені яких присутні в поліномі, заводяться на суматор за модулем 2;
- регістр заповнюється початковим значенням;
- додатково на суматор подається константа «1»;
- вихід суматора подається на вхід регістра зсуву;
- бітовим виходом регістра може вважатися або його вихід, або будь-який його біт.

Пристрій працює так. На початку він заповнюється так, як визначено зародком. У черговому такті роботи обчислюється сума за модулем 2 і обчислений біт пересилається в регістр на першу позицію вліво. Водночас решта бітів зсувається на одну позицію вправо. При цьому останній біт праворуч усувається з пристрою й подається у криптографічний пристрій як черговий біт двійкової послідовності.

Зауважимо, що робота такого регістра суттєво залежить від первісного його заповнення. Наприклад, якщо заповнити регістр тільки нулями, то на виході будемо весь час отримувати лише нулі. Саме тому на суматор додатково подається константа, що дорівнює 1.

Наведемо приклад простого LFSR. Припустимо, що ми хочемо побудувати LFSR із періодом 7. Для його реалізації оберемо поліном $X^3+X+1$ (період дорівнює $2^3-1=7$).

Схема такого регістра зсуву подана на рис. 8.2.

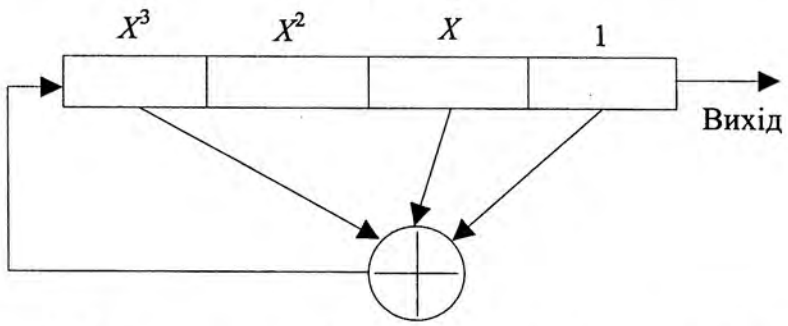



*Рис. 8.2. Регістр зсуву з періодом 7, породжений поліномом* $X^3+X+1$

Псевдовипадкова послідовність, генерована таким регістром, подана в таблиці 8.1.

*Таблиця 8.1*

| $X^3$ | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|
| $X^2$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| $X$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

Самі по собі LFSR можна було би вважати гарними генераторами псевдовипадкових послідовностей, якби не деякі небажані невипадкові властивості [14]. Однією з таких властивостей прийнято вважати те, що послідовні біти – лінійні, що може зробити їх беззмістовними для шифрування. Для генератора з $n$-бітним регістром внутрішній стан є комбінацією $n$ попередніх станів. Це можна використати для ефективної криптоаналітичної атаки [14].

Крім того, великі випадкові числа, які генеруються з використанням послідовних бітів такого генератора, сильно корельовані та іноді зовсім не випадкові.

Так чи інакше, лінійні регістри зсуву зі зворотним зв'язком досить популярні для створення криптографічних алгоритмів. Для поліпшення криптостійкості вихід генераторів ще обробляють хеш-функцією або шифрують DES-подібним алгоритмом із відомим ключем.

***Генератор BBS***

Програмний генератор двійкових послідовностей BBS (назву утворено від перших літер його авторів – Ленори та Мануеля Блум та Майка Шуба, Blum-Blum-Shub) вважають одним із найсильніших програмних генераторів псевдовипадкових послідовностей. Він криптографічно стійкий і може мати серйозні криптографічні застосування [14].

Нехай є два простих числа, $p$ і $q$, причому $p \equiv q \equiv 3 \bmod 4$. Добуток цих чисел $n=pq$ називається цілим числом Блума. Оберемо ще одне випадкове число, $x$, взаємно просте з $n$ та обчислимо $x_0 \equiv x \bmod n$. Це число вважається стартовим числом генератора.

Далі можна обчислити наступні біти послідовності за формулами $x_i \equiv x_{i-1}^2 \bmod n$ і $s_i \equiv x_i \bmod 2$. Останнє визначає, що як вихід генератора обирається молодший біт числа $x_i$. Отже, можемо записати:

$x_0 = x^2 \bmod n,$
for i = 1 to ∞,
  $x_i = (x_{i-1})^2 \bmod n,$
  $s_i = x_i \bmod 2.$

Найцікавішою властивістю генератора BBS є те, що для визначення значення $i$-го біта зовсім необов'язково знати всі попередні $i$-1 бітів. Для безпосереднього обчислення значення $i$-го біта достатньо знати $p$ і $q$.

Безпека цієї схеми ґрунтується на складності розкладання $n$ на множники. Число $n$ можна опублікувати, так що кожен зможе генерувати біти за допомогою

цього генератора. Однак поки криптоаналітик не розкладе $n$ на множники, він не зможе передбачити вихід генератора.

Більше того, генератор BBS непередбачуваний як у правому, так і в лівому напрямках. Це означає, що, отримавши послідовність бітів, криптоаналітик не зможе передбачити ні наступний, ні попередній біти послідовності. Причиною цього є не якийсь заплутаний механізм генерації, а математика розкладання $n$ на множники.

Приклад:

$p = 19$; $q = 23$,

$p = q \equiv 3 \bmod 4$,

$n = 437$,

$x = 233$.

| I | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $X_i$ | 101 | 150 | 213 | 358 | 123 | 271 | 25 | 188 |
| $S_i$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

Обов'язковою умовою, що накладається на зародок $x$, повинно бути таке: а) $x$ – просте; б) $x$ не ділиться ні на $p$, ні на $q$.

Цей генератор повільний, але є спосіб його прискорення. Як вказано у [14], як біти псевдовипадкової послідовності можна використовувати не один молодший біт, а $\log_2 m$ молодших бітів, де $m$ – довжина числа $x_i$. Порівняна повільність цього генератора не дозволяє використовувати його для потокового шифрування (цей недолік зі зростанням швидкодії комп'ютерів стає менш актуальним), а от для високонадійних застосувань, як, наприклад, генерування ключів, він вважається кращим за багато інших.

***Генератор Блум-Мікалі***

Безпека цього генератора базується на проблемах обчислення дискретного логарифма в кінцевому полі.

Для реалізації цього генератора генерують два простих числа: $g$ та $p$. Зародок $x_0$ породжує такий процес генерації: $x_{i+1} \equiv g^{x_i} \bmod p$. Виходом генератора буде 1, якщо $x_i < (p\text{-}1)/2$, і 0 – якщо ця нерівність не виконується.

Якщо модуль $p$ достатньо великий для того, щоб обчислення дискретного логарифму було обчислювально складною задачею, цей алгоритм безпечний.

Детальніше про нього можна дізнатися у книзі Брюса Шнаєра [14].

***Генератор RSA***

Генератор RSA використовує алгоритм RSA для утворення псевдовипадкових послідовностей. Застосовується публічний ключ $(e, n)$:

1) оберемо випадкове число $X_0 < n$;
2) обчислимо $X_i = (X_{i-1})^e \bmod n$

For $i$=1 to $\infty$,
$X_i = (X_{i-1})^e \bmod n$,
$B_i = X_i \bmod 2$.

Безпека цього генератора ґрунтується на складності розкладання великого числа на множники, тобто на тій самій задачі, яка лежить в основі криптостійкості самої системи RSA.

***Використання симетричних алгоритмів*** для генерування псевдовипадкових послідовностей також може бути виправданим, оскільки вони сконструйовані так, щоб максимально перемішувати й розсіювати особливості вхідного тексту. Як показує аналіз, наприклад алгоритму DES, уже при 8 циклах бінарна послідовність на виході практично мало чим відрізняється від випадкової.

Що стосується алгоритму ГОСТ, то навіть у рекомендаціях по його використанню є режим, що називається «гаммуванням зі зворотним зв'язком», де сам алгоритм використовується для генерування т.зв. «гамми», яка побітово накладається на вхідний текст.

Детальніше про генератори випадкових і псевдовипадкових послідовностей можна почитати в багатьох джерелах, наприклад у [14].

# Криптографічний практикум

Криптографічний практикум складається з практичних занять, які допомагають студентам розібратися в деталях як класичних алгоритмів шифрування, так і сучасних криптографічних методів та лабораторного практикуму.

Лабораторний практикум побудований у такий спосіб, що дозволяє студентам набути навичок створювання програмного забезпечення з урахуванням вимог до криптографічного захисту інформації.

Лабораторний практикум побудований від простого до складного. Перші три лабораторні роботи присвячені шифру Цезаря в різних його модифікаціях, далі розглядаються питання генерування псевдовипадкової послідовності та використання шифру гамування.

Сучасні алгоритми шифрування представлено спрощеними системами DES та RSA, які використовуються для шифрування та електронного підпису документів відповідно.

Інші лабораторні роботи присвячені відкритому розподілу криптографічних ключів і створенню простого потокового шифру на основі генератора псевдовипадкових послідовностей BBS.

Для створення курсу лабораторних робіт із криптографії викладач може обрати 5-6 робіт із запропонованих у залежності від мети курсу та рівня підготовки студентів.

**Практичні заняття з криптографії**

Практичні заняття присвячені вивченню особливостей класичних технік шифрування, а також полегшених сучасних криптоалгоритмів. Окрім цього, широко застосовуються (також, по можливості, спрощено) методи частотного криптоаналізу. Для дослідження методів частотного криптоаналізу група студентів розбивається на дві підгрупи, кожна з яких спочатку виконує шифрування спеціально відібраних або довільних повідомлень, а потім підгрупи обмінюються криптограмами і намагаються їх розшифрувати методом частотного криптоаналізу.

**Вивчення шифру Цезаря**

Необхідно зашифрувати або розшифрувати повідомлення за допомогою алгоритму Цезаря. В задачах відомий ключ, тобто величина зсуву, або таблиця заміни.

Спробуємо зашифрувати відкритий текст «*Організуйте зустріч нового об'єкта в обумовленому місці*» алгоритмом Цезаря зі зсувом -2.

Таблиця заміни

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І | Ї | Й | К | Л | М |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

Отримаємо криптограму «*15 17 01 30 14 08 06 20 10 19 03 06 20 18 19 17 08 24 14 15 00 15 01 15 15 31 04 11 19 20 00 15 31 20 13 15 00 12 03 14 15 13 20 13 08 18 23 08*».

Для дешифрування методом частотного криптоаналізу отриману «шифровку», в якій усього 48 знаків, піддають такому аналізу: найчастіше в ній зустрічається число 15 – 8 разів (імовірність тоді можна обчислити так: 8/48=0,167). Порівнявши з частотною таблицею української мови, можна припустити, що 15=«О». Тоді отримуємо таку таблицю заміни:

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І | Ї | Й | К | Л | М |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 31 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 |
| Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |

Замінивши числа літерами, отримаємо відкритий текст.

Звичайно, для реального застосування частотного криптоаналізу такої малої кількості відкритого тексту, як правило, замало, і мова може йти лише про спеціально підібрані фрази, що відповідають статистичним особливостям української мови.

Робота з довільними короткими фразами методами частотного криптоаналізу може й не дати бажаного результату.

**Вивчення афінної системи шифрування Цезаря**

У системі шифрування Цезаря використовувалися лише адитивні властивості множини цілих $Z_n$. Однак символи множини $Z_n$ можна також множити за модулем $n$. Застосовуючи одночасно операції додавання та множення за модулем $n$ над елементами множини $Z_n$, можна отримати систему підстановок, яку називають ***афінною системою підстановок Цезаря***.

$$E_{a,b}(t) = at + b \ (mod\ n),$$

де $a$, $b$ – цілі числа, $0 < a,b < n$, НСД $(a,n) = 1$.

Зауважимо, що перетворення $E_{ab}(t)$ є взаємно однозначним відображенням на множині $Z_n$ тільки в тому випадку, якщо найбільший спільний дільник (НСД($a,n$)), дорівнює одиниці, тобто $a$ і $n$ повинні бути взаємно простими числами.

Наприклад, для англійського алфавіту, нехай $m = 26$, $a = 3$, $b = 5$. Тоді, очевидно, НСД (3,26) = 1, і ми отримуємо таке співвідношення між числовими кодами літер:

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t + 5$ | 5 | 8 | 11 | 14 | 17 | 20 | 23 | 0 | 3 | 6 | 9 | 12 | 15 |

| $t$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t + 5$ | 18 | 21 | 24 | 1 | 4 | 7 | 10 | 13 | 16 | 19 | 22 | 25 | 2 |

Перетворюючи числа в літери англійської мови, отримаємо такі співвідношення між літерами відкритого тексту та шифротексту:

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

Відкрите повідомлення **NIGHT** перетворюється в шифротекст **SDXAK.**

Перевагою афінної системи, крім вищої порівняно зі звичайною системою Цезаря крипостійкістю, є зручне керування ключами – ключі подаються в компактній формі у вигляді пари чисел ($a$, $b$).

Тепер попрацюємо з українською мовою. Вважаємо, що потужність алфавіту $n = 32$.

Для нашого практичного заняття будемо використовувати відкритий текст «*Лабораторна робота з криптографії*».

Використаємо такі числа: $n = 32$, $a = 3$, $b = 5$. НСД(3,32) = 1, так що ми можемо використати їх для створення афінної системи Цезаря.

Отже, отримаємо таку таблицю заміни:

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t$ +5 | 5 | 8 | 11 | 14 | 17 | 20 | 23 | 26 | 29 | 0 | 3 | 6 | 9 |

| $t$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t$ +5 | 12 | 15 | 18 | 21 | 24 | 27 | 30 | 1 | 4 | 7 | 10 | 13 |

| $t$ | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|
| $3t$ +5 | 16 | 19 | 22 | 25 | 28 | 31 | 2 |

На основі цієї таблиці отримаємо таку таблицю відповідності літер української мови:

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І | Ї | Й | К | Л | М |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Е | З | Ї | Л | О | С | Ф | Ч | Ь | А | Г | Є | И | Й | М | П |

| Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Т | Х | Ш | Ю | Б | Д | Ж | І | К | Н | Р | У | Ц | Щ | Я | В |

Отже маємо таку шифрограму для нашого відкритого тексту:

*«МЕЗХЮЕДХЮТЕЮХЗХДЕЬЙЮАШДХЛЮЕІГЄ».*

Для криптоаналізу підрахуємо частоти появи літер: *Ю, Х* та *Е* зустрічаються по 5 разів. Ототожнивши *Ю* з *О*, не отримаємо зв'язного тексту, а якщо ототожнимо *Х* з *О*, тоді отримаємо рівняння $24 = (a \times 17 + b) \bmod 32$, яке треба розв'язати відносно $a$ та $b$. Для розв'язку замало даних: потрібне друге рівняння. Другим рівнянням може бути НСД$(a, n) = 1$. Тоді маємо $a = 1, 3, 5, 7,...$ Одиницю навряд чи використовували б, пробуємо 3. Тоді $24 = (3 \times 17 + b) \bmod 32$, $24 = (51 + b) \bmod 32$. Звідки $b = 5$. Тепер отримуємо таблицю заміни:

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І | Ї | Й | К | Л | М |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Е | З | Ї | Л | О | С | Ф | Ч | Ь | А | Г | Є | И | Й | М | П |

| Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Т | Х | Ш | Ю | Б | Д | Ж | І | К | Н | Р | У | Ц | Щ | Я | В |

Застосовуючи таблицю, одержимо розшифроване повідомлення.

**Вивчення матричних способів шифрування**

Матричний шифр має в основі таку ідею. Наприклад, у нас є повідомлення *«Припиніть скуповувати акції»*. Довжина його – 25 символів. Будуємо таблицю 5 х 5 і вписуємо туди по рядках наше повідомлення.

| *П* | *Р* | *И* | *П* | *И* |
|---|---|---|---|---|
| *Н* | *І* | *Т* | *Ь* | *С* |
| *К* | *У* | *П* | *О* | *В* |
| *У* | *В* | *А* | *Т* | *И* |
| *А* | *К* | *Ц* | *І* | *Ї* |

Зчитуємо символи по стовпчиках, наприклад зліва-направо. Тоді отримаємо шифрограму

*«ПНКУАРІУВКИТПАЦПЬОТІИСВИЇ».*

Для розшифровки шифрограми її записують у матрицю 5 х 5 (наприклад, домовились використовувати саме 5 х 5) уже по вертикалі. Тоді читають відкритий текст по горизонталі.

Матричний шифр можна ускладнити за допомогою ключового слова – ключа. Нехай ключ – «*БІРЖА*». Створюємо таку таблицю:

| *Б* | *І* | *Р* | *Ж* | *А* |
|---|---|---|---|---|
| *2* | *4* | *5* | *3* | *1* |
| *П* | *Р* | *И* | *П* | *И* |
| *Н* | *І* | *Т* | *Ь* | *С* |
| *К* | *У* | *П* | *О* | *В* |
| *У* | *В* | *А* | *Т* | *И* |
| *А* | *К* | *Ц* | *І* | *Ї* |

А тепер записуємо стовпчики в порядку розташування цифр у другому рядочку, тобто в алфавітному порядку літер у ключовому слові. Отримуємо «*ИСВИЇПНКУАПЬОТ ІРІУВКИТПАЦ*».

Криптоаналіз матричного шифру ґрунтується на частотних таблицях пар літер. Нехай ми використовуємо лише ключі довжиною 5 символів. Це означає, що матриця завжди буде мати 5 стовпчиків.

Тоді криптоаналіз зводиться до записування криптограми по вертикалі в матрицю і переставляння стовпчиків. При цьому дуже важливо зрозуміти, що деякі літери української мови поруч стояти ніяк не можуть. У нашому прикладі отримуємо наведену матрицю.

| *И* | *П* | *П* | *Р* | *И* |
|---|---|---|---|---|
| *С* | *Н* | *Ь* | *І* | *Т* |
| *В* | *К* | *О* | *У* | *П* |
| *И* | *У* | *Т* | *В* | *А* |
| *Ї* | *А* | *І* | *К* | *Ц* |
| *1* | *2* | *3* | *4* | *5* |

Звідси видно, що:

- стовпчики 3 і 4 поруч стояти ніяк не можуть, бо літери «*Ь*» та «*І*» поруч не зустрічаються;
- стовпчики 2 і 3 також поруч не можуть стояти, бо літера «*П*» в українських словах не подвоюється;
- стовпчики 1 і 2 також не можуть бути поруч, бо «*Ї*» та «*А*» поруч не зустрічаються.

З іншого боку, дуже зручний перший рядок 3, 4, 5 стовпчиків («*ПРИ*»). Однак такий самий рядок ми отримаємо, якщо порядок стовпчиків буде 2, 4, 5.

| *П* | *Р* | *И* | *П* | *И* |
|---|---|---|---|---|
| *Н* | *І* | *Т* | *Ь* | *С* |
| *К* | *У* | *П* | *О* | *В* |
| *У* | *В* | *А* | *Т* | *И* |
| *А* | *К* | *Ц* | *І* | *Ї* |
| *2* | *4* | *5* | *3* | *1* |

Таким чином, пробуємо порядок 2, 4, 5, 3, 1.

Читаючи по рядках, отримаємо відкрите повідомлення.

Ці приклади допоможуть студентам розв'язати більшість задач із криптографічного практикуму. Пропонуються також завдання підвищеної складності, розв'язання яких вимагає від студентів ґрунтовних знань і кмітливості.

# Задачі з криптографії

***Класичні техніки шифрування***

1. Розшифруйте повідомлення, використовуючи частотні таблиці української мови: «15 17 *01 30 14 08 06 20 10 19 03 06 20 18 19 17 08 24 14 15 00 15 01 15 15 31 04 11 19 20 00 15 31 20 13 15 00 12 03 14 15 13 20 13 08 18 23 08*». Потужність алфавіту – 32 символи.

2. Розшифруйте криптограму «*ЙЦЄЄІ ЧГЇСВ ЛСЕУН ЗЕЧЄЕ ЮВ*», зашифровану шифром пар на ключовій фразі «*ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК*». Потужність алфавіту – 32 символи.

3. Зашифруйте текст «*ЮСТАС – АЛЕКСУ. ПЕРЕХОДЬТЕ НА ЗАПАСНУ РАДІОХВИЛЮ*» за допомогою алгоритму Цезаря, використавши зсув 5.

4. Розшифруйте криптограму «*ЬЯЛЕБ ІШНЧЩ РВЯІЛ ТЙЛНМ ЕСТЬЬ Й*», зашифровану на ключі «*ШТІРЛІЦ*» шифром Віженера.

5. Розшифруйте криптограму «*АХАІЛ ІЮОТИ ЛВЙЛЕ ЦЛМІН НЯЧБЕ ЕИТЬФ ККЕНО*», зашифровану матричним шифром на ключі «*САМСУНГ*».

6. Зашифруйте повідомлення «*ДАЙТЕ ШПАРГАЛКУ ДО П'ЯТОГО БІЛЕТУ*» матричним шифром на ключі «*ШИФР*».

7. Розшифруйте криптограму «*ШЄДИУ ЯЮРНВ РЗБГД ФБВЖЖ ССЖПЧЖ*», зашифровану шифром Віженера на ключі «*ГОРБУНКОВ*».

8. Розшифруйте криптограму «*НСТДЮ УДГЮМ ГНЖТС ЗАИДС КХМДХ ПЕШМ ВУЙХЯ*», зашифровану афінним методом Цезаря на ключі (3,5). Потужність алфавіту – 32 символи.

9. Зашифруйте текст «*МИ ВИВЧАЄМО КРИПТОГРАФІЮ*» шифром Віженера на ключі «*ЗАХИСТ*».

10. Розшифруйте криптограму «*КЛІХТ ВЦТАК ФГШМК НШФЗВ БРЩЕЛ КЦР*», зашифровану на ключі «*ШИФРОВКА*» шифром Віженера.

11. Зашифруйте повідомлення «*КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ*» за допомогою афінної системи Цезаря. Ключ виберіть самостійно та обґрунтуйте його вибір. Потужність алфавіту – 32 символи.

12. Розшифруйте криптограму «*ВКХРЖ ШВДЩЬ ЙЮАШД ХЛЮЕІ ГЄ*», зашифровану афінним методом Цезаря на ключі (3,5). Потужність алфавіту – 32 символи.

13. Зашифруйте повідомлення «*ЗУСТРІЧ ПРИЗНАЧЕНО В КАФЕ «СЛОН*» шифром пар, скориставшись як ключем фразою »*ВИКОНАЙТЕ ЗАВДАННЯ І ПОВЕРТАЙТЕСЬ*». Потужність алфавіту – 32 символи.

14. Зашифруйте текст «*ЮСТАС – АЛЕКСУ. ВИЙДІТЬ НА ЗВ'ЯЗОК З БОРМАНОМ*» за допомогою матричного шифру на ключі «*РАЦІЯ*».

15. Зашифруйте повідомлення «*Я НАВЧАЮСЯ НА ФАКУЛЬТЕТІ КОМП'ЮТЕРНИХ НАУК*» за допомогою матричного шифру на ключі «*ЦЕЗАРЬ*».

16. Розшифруйте повідомлення:
«*ЇАЙХЮАБДЕИДСЙЕТЕМЬЇВЬЙЖТХПСЮШВДЩ*», зашифроване афінною системою Цезаря на ключі (3, 5).

17. Складіть аналог шифру Першої світової війни для української мови. Використайте для цього квадрат 6 × 6. За допомогою створеного шифру зашифруйте повідомлення «*ДЛЯ ЗАВЕРШЕННЯ ПРОЕКТУ НЕОБХІДНО ЗБІЛЬШИТИ ФІНАНСУВАННЯ*».

18. За допомогою шифру, створеного в задачі 17, зашифруйте повідомлення «*ВИКОРИСТАННЯ ЦЬОГО КАНАЛУ ЗВ'ЯЗКУ НЕБЕЗПЕЧНЕ*» на ключовому слові «*ПРОЕКТ*».

19. Складіть квадрат Полібія для української мови та зашифруйте за його допомогою повідомлення «*ЕЛЕКТРОННИЙ ПІДПИС ОТРИМАНОГО ДОКУМЕНТА ПІДТВЕРДЖЕНО*». Оцініть, яка стійкість цього алгоритму відносно атаки «грубою силою».

20. Створіть власну систему шифрування для української мови, аналогічну до шифру Play-fair. Зашифруйте за її допомогою повідомлення «*З НАСТУПНОГО ПОНЕДІЛКА ВИКОРИСТОВУЄМО КЛЮЧ НОМЕР П'ЯТЬ*». Оцініть криптостійкість цього алгоритму відносно атаки «грубою силою».

21. Розробіть формули, еквівалентні таблиці Віженера, для зашифрування та розшифрування інформації.

***Симетричні криптосистеми***

1. Використовуючи ключ (0111111101) вручну розшифруйте криптограму (10100010), зашифровану алгоритмом S-DES. Продемонструйте проміжні результати, отримані на виході кожної функції (IP, FK, SW, FK, $IP^{-1}$). Перетворіть отриману двійкову послідовність у літери, вважаючи, що A = 0000, B = 0001,…,P = 1111. Підказка: для контролю результату зазначимо, що після застосування функції SW двійковий рядок має вигляд (00010011). Мова – англійська.

2. Доведіть, що процес розшифрування в алгоритмі DES обернений до процесу шифрування.

3. Доведіть, що в алгоритмі DES перші 24 біти кожного циклового підключа обираються з однієї 28-бітної підмножини бітів вихідного ключа, а інші 24 біти – з іншої, причому обидві підмножини не перетинаються.

4. Якщо в режимі ECB алгоритму DES деякий біт шифротексту, що передається каналом зв'язку, спотворено, в результаті після розшифрування виявиться спотвореним лише один блок відкритого тексту. У режимі CBC таке спотворення вплине й на наступні блоки, $M_{i+1}$ і $M_{i+2}$. Чи зміняться наступні за $M_{i+2}$ блоки?

Припустимо, що помилка виникла в одному біті відкритого тексту $M_1$. Яка кількість блоків зашифрованого тексту зазнає при цьому змін? Яке повідомлення отримає адресат?

5. Припустимо, що DES використовується для шифрування 8-бітного потоку даних як потоковий шифр у режимі CFB. На скільки блоків шифротексту розповсюджується спотворення одного біта вхідного тексту в цьому режимі?

6. За якої умови потрійний DES з трьома ключами сумісний з однократним DES? Доведіть це.

7. Проаналізуйте процедуру генерування циклових підключів алгоритму S-DES і дайте відповіді на такі запитання:

- Наскільки важлива початкова перестановка P10?
- Наскільки важливі дві функції зсуву LS-1?

***Асиметричні криптосистеми***

1. Зашифруйте повідомлення $M$ за допомогою алгоритму RSA для таких значень параметрів:

- $p$=3; $q$=11; $d$=7; $M$=5.
- $p$=5; $q$=11; $d$=3; $M$=9.
- $p$=7; $q$=11; $d$=17; $M$=8.
- $p$=11; $q$=13; $d$=11; $M$=7.
- $p$=17; $q$=31; $d$=7; $M$=2.

2. Асиметрична криптосистема, що використовує RSA, має публічний ключ $e = 5, n = 35$. Зловмисник перехопив зашифроване повідомлення $C$=10. Обчисліть приватний ключ і розшифруйте повідомлення.

3. Обчисліть приватний ключ RSA, якщо публічний ключ користувачів $e = 31$, $n = 3599$.

4. При використанні алгоритму RSA після виконання невеликої кількості операцій повторного шифрування отримується відкритий текст. Що, на Вашу думку, може бути найімовірнішою причиною цього?

5. У криптосистемі, що використовує алгоритм RSA, кожен користувач має публічний і відповідний йому приватний ключ. Припустимо, що деякий користувач дізнається, що його приватний ключ скомпрометовано. Однак замість того, щоб згенерувати новий модуль $n$, він вирішує обчислити нову пару ключів на старому модулі. Наскільки це безпечно?

6. Розглянемо таку схему:

- Обираємо непарне число $E$.

- Обираємо два простих числа $P$ і $Q$ так, щоб $(P\text{-}1)(Q\text{-}1)\text{-}1$ ділилося на $E$.
- Обчислюється модуль $N=PQ$.
- Обчислюється $D=((P\text{-}1)(Q\text{-}1)(E\text{-}1)+1)/E$.

Чи буде така схема еквівалентною RSA? Обґрунтуйте свою відповідь.

7. Адміністратор безпеки ТОВ «Роги і копита» запропонував для захищеного обміну даними нижченаведений протокол. Коли користувач $A$ хоче відіслати повідомлення користувачу $B$, він повинен виконати такі дії:

- сформувати та відправити отримувачу повідомлення у форматі $[A, E_{\text{КпубВ}}(M), B]$, де $A$ та $B$ – імена відправника та отримувача, $E_{\text{КпубВ}}(M)$ – зашифроване на публічному ключі $K_{\text{пубВ}}$ отримувача повідомлення $M$;
- отримувач $B$ підтверджує отримання цього повідомлення від $A$, повертаючи йому $[B, E_{\text{КпубА}}(M), A]$.

Знайдіть уразливість такого протоколу по відношенню до відомої атаки та модифікуйте протокол так, щоб він став стійким до атак такого типу.

8. Еліптичну криву $E_{11}(1,6)$ задано рівнянням $y^2 = (x^3+x+6) \bmod 11$. Визначте всі точки цієї кривої. *Підказка*: почніть з обчислення всіх значень правої частини рівняння для всіх значень $x$.

9. Для випадку $E_{11}(1,6)$ розгляньте точку $G = (2,7)$. Обчисліть числа, кратні $G$, починаючи від $2G$ до $13G$ включно.

10. Доведіть, що схема шифрування Ель-Гамаля дійсно працює, тобто розшифрування справді відтворює відкритий текст.

11. Розгляньте алгоритм шифрування Ель-Гамаля з такими параметрами:

- Загальний модуль $q = 71$.
- Первісний корінь $a = 7$.
- Якщо користувач $B$ має публічний ключ $Y_B = 3$, а користувач $A$ обирає випадкове число $k = 2$, то яким буде зашифрований текст для повідомлення $M = 30$?
- Якщо тепер користувач $A$ обере інше значення $k$, так що повідомлення $M = 30$ буде шифруватися як $C = (59, C_2)$, то яким за таких умов буде $C_2$?

12. У криптосистемі RSA з модулем $n = 5963$, приватним ключем $d = 37$ та публічним $e = 157$ п'ятиразове за шифруванням повідомлення $M$ призводить до криптограми, що збігається з відкритим текстом. Доведіть це та поясніть причину даного явища.

13. Знайдіть усі первісні корені за модулем 25.

14. Число 2 є первісним коренем за модулем 29. Створіть таблицю степенів $x$ за модулем 29 та використайте її для розв'язку таких рівнянь:

- $17x^2 \equiv 10 \bmod 29$;
- $x^7 \equiv 17 \bmod 29$.

15. Сформуйте відповідні пари публічних і приватних ключів для криптосистеми RSA для значень модуля $n_1 = 377$, $n_2 = 451$.

16. Обчисліть приватний ключ криптосистеми RSA за публічним ключем $e = 97$ для значень модуля $n_1 = 299$, $n_2 = 527$.

17. Обчисліть публічний ключ криптосистеми RSA за публічним ключем $e = 91$ для значень модуля $n_1 = 187$, $n_2 = 319$.

18. Обчисліть публічний ключ криптосистеми RSA за публічним ключем $e = 71$ для значень модуля $n_1 = 229$, $n_2 = 451$.

***Електронний цифровий підпис***

1. Опишіть відмінності між алгоритмами хешування MD4 та MD5. Наскільки, на Ваш погляд, захищеність MD5 вища від захищеності MD4?

2. Запропонуйте схему електронного цифрового підпису з рівнянням перевірки $S^3 \equiv H \bmod n$, де $S$ – підпис, $H$ – хеш-образ документа, що підписується.

3. Зловмисник, отримавши підпис $S'$ для хеш-образу $H'$, сформував «несанкціонований» підпис для іншого документа з хеш-образом $H$. Як це йому вдалося, якщо рівняння перевірки підпису $\alpha \equiv S^H \bmod n$, де $(n,б)$ – публічний ключ, а $n$ – RSA-модуль?

4. У алгоритмі DSA передбачено, що, якщо в результаті процесу створення підпису отримано нульове значення числа $r$, необхідно перервати обчислення і згенерувати нове значення підпису. Чому?

5. Що станеться, якщо значення $r$ у схемі цифрового підпису DSA буде скомпрометоване?

6. Відповідно до стандарту DSS, значення $r$ необхідно генерувати всякий раз, коли виробляється новий підпис, навіть якщо другий раз підписується те ж саме повідомлення. Отже, всі підписи будуть різними. Це не так для підписів RSA. Які практичні наслідки такої різниці алгоритмів?

***Елементи криптоаналізу***

1. Нехай відкрите повідомлення $M = 23$. Зашифруйте його за допомогою криптосистеми RSA, що використовує модуль $n = 527$ та публічний ключ $e = 97$. Здійсніть атаку з багатократним перешифруванням: зашифровуйте криптограму доти, поки не отримаєте $C = 23$. Скільки ітерацій Ви здійснили?

2. Розшифруйте повідомлення «*ЕГЮ ДГХГОАМСР ТСЕЙРЗР ЇГМРВХЙ ТСЇЙЩКЛ ЇГЕХУГ ЖС ЖУЦЄСЛ ЄСЖЙРЙ*» методом частотного криптоаналізу.

3. Розшифруйте повідомлення «*ЇГ ЖЕК ЄСЖЙРЙ ЬЗНГИПС ЕГФ Ц ФЦФКЖГ ФТУГЕГ РГ РГУГЖК*» методом частотного криптоаналізу.

4. На дошці об'яв було вивішено таке оголошення:

**«(9(&(.1~~%**

\;5 ;"\01~"7 =1"!1:"(9( $\:;\ ,( #$"7!~( @:#&7 \=#;"_ \ ):#$"7=~7/ 8#~%""%/ 8#/7;"7&7 )%"_ &#@(:#"(:~7/ :(@5" "# @\0\"_ :1$(*1~0(!#~5 !7$&#0#=1* (":7*#^"_ 1$8#*1~#+54 ~\ (+5~$\ 8# :18\&_"#"#*7 :(@("7 ~# ):("%85 ;1*1;":\»

Розшифруйте це повідомлення. *Підказка*: з чого звичайно починаються оголошення?

5. Припустимо, що Ви перехопили повідомлення, зашифроване шифром «сцитала». Автор цього повідомлення, бажаючи, щоби рядочки літер були рівними, проводив горизонтальні лінії, які залишилися на стрічці як рисочки між літерами. Кут нахилу цих рисочок до краю стрічки дорівнює $\alpha$, ширина стрічки – $d$, відстань між рисочками – $h$. Як, користуючись цими даними, прочитати текст?

6. Якось, коли Шерлок Холмс і доктор Ватсон разом насолоджувалися післяобіднею люлькою, в кімнаті з'явилася місіс Хадсон.

– До Вас відвідувач, містере Холмс, – сказала вона, – схожий на студента.

У кімнаті з'явився юнак.

– Доброго дня, містере Холмс, здрастуйте доктор Ватсон! – сказав він.

– Мене звуть Дмитро Паладян, я студент 4-го курсу факультету комп'ютерних наук Чернівецького університету. Мені дуже потрібна Ваша допомога, містер Холмс. Я вже п'ять днів поспіль отримую якісь короткі зашифровані послання. Мені здається, що мені загрожує небезпека!

– Чому Ви так вирішили, Дмитре? – спитав Холмс.

– Для чого ще будуть зашифровувати записки? – відповів юнак.

– А записки ці у Вас з собою? – спитав Холмс.

– Так, ось вони, містер Холмс, – юнак передав Холмсу аркуш паперу, заповнений якимись фігурками.

– Хм-м, – сказав Холмс, вивчаючи аркуш, – Дивіться, Ватсон, щось таке ми вже колись бачили! Пам'ятаєте справу місіс К'юбіт? Там були схожі записки[1]. Тих записок також було п'ять, здається. Зверніть увагу, Ватсон, чотири з п'яти записок закінчуються знаками запитання. Це означає – автор щось запитує в адресата, як Ви думаєте?

– Бачите, деякі фігурки мають прапорці. Пам'ятаєте, Ватсон, тоді також були фігурки з прапорцями? Що вони означали тоді, Ватсон, не пригадуєте?

Шерлок Холмс уважно вивчав записки.

– Цікаво, цікаво, Ватсон! Бачите, дві записки з п'яти починаються зовсім однаково! Скажіть, юначе, може автор записок знає Ваше ім'я?

– Я не знаю, хто автор записок, містер Холмс, – відповів Дмитро, – але не виключено, що він може знати, як мене звуть.

---

[1]див. А.Конан Дойль «Танцюючі чоловічки»

– Бачите, молодий чоловіче, дуже часто, коли запитують щось у людини, звертаються до неї на ім'я, – сказав Холмс.

– Ну, що ж, юначе, можна ще сказати? Без сумніву, це шифр. Судячи з усього, дуже простий. Здається, він називається шифром заміни. А чому, молодий чоловіче, Ви звернулись до нас? Ви казали, що навчаєтеся на факультеті комп'ютерних наук?

– Так.

– Так що, Вам не читають криптографії? Ні, шановний! Я не буду витрачати мій час на такий простий шифр…

– А Ви вважаєте, містер Холмс, що мені нічого не загрожує?

– Абсолютно нічого! Бачите, юначе, коли в кінці речення ставлять знак запитання, то це означає, що того, кому призначена записка, про щось запитують, а не загрожують. Я впевнений, що Ви в цілковитій безпеці!

– От що, – закінчив Шерлок Холмс, – скористайтесь своїми знаннями з криптографії та самостійно розшифруйте ці записки. Розкрийте шифр і дайте відповіді на поставлені запитання (див. Додаток В).

***Проблеми розподілу ключів***

1. «Однак, – заперечив доктор Ватсон, – ваші клієнти використовують у своїй мережі протокол обміну ключами за Діффі та Хеллманом. Цей протокол ґрунтується на тому, що дискретне логарифмування, як відомо, вважається складною математичною проблемою, чи не так?»

«Так, Ватсон, – відповів Холмс, – за правильного вибору відповідних параметрів проблема дискретного логарифмування дійсно виявляється складною. Мої клієнти знають це. Більше того, саме тому вони й обрали цей метод розподілу ключів. На жаль, їх супротивник, професор Моріарті, також знайомий із цим алгоритмом. Однак активний супротивник значно частіше досягає успіху, ніж пасивний. Моріарті не буде чекати, Ватсон!»

«Ви думаєте, Холмс, – здивовано відповів Ватсон, – що Моріарті може знайти спосіб зламати цю схему розповсюдження ключів?»

«Ох, Ватсон, на жаль, це не так складно, як Ви думаєте! – посміхнувся Холмс. – Все, що Моріарті потрібно, – це знайти місце на каналі зв'язку, де він може не тільки перехоплювати, а й змінювати перехоплені повідомлення. Я впевнений, що Моріарті знайде таку можливість. Знаходячись у такому місці, він буде діяти так…»

Як саме буде діяти професор Моріарті? Чому таку атаку можна застосувати до протоколу Діффі-Хеллмана? Яким способом Ви би запропонували перешкодити цій атаці?

2. Припустимо, що деяка особа пропонує Вам такий спосіб підтвердження того, що Ви обидва володієте одним криптографічним ключем. Ви створюєте випадкову послідовність бітів такої ж довжини, як і ключ, об'єднуєте її з ключем за допомогою операції XOR та пересилаєте результат каналом зв'язку. Ваш партнер за допомогою операції XOR об'єднує отриманий блок із ключем (який

повинен збігатися з Вашим) і повертає Вам результат. Якщо отриманий рядок збігається з Вашою випадковою послідовністю, це означає, що ключі (Ваш і Вашого партнера) однакові, хоча жоден з вас не пересилав його. Чи немає в цій схемі прихованих дефектів?

***Генерування псевдовипадкових послідовностей***

1. Обчисліть період повторення послідовності псевдовипадкових чисел у лінійному регістрі зсуву зі зворотним зв'язком, якщо максимальний ступінь породжувального нерозкладного поліному дорівнює 7.
2. Обчисліть період повторення послідовності псевдовипадкових чисел у лінійному регістрі зсуву зі зворотним зв'язком, якщо максимальний ступінь породжувального нерозкладного поліному дорівнює 3.
3. Обчисліть перші 7 бітів псевдовипадкової послідовності, що її згенерує генератор BBS, який ініціалізується значеннями $q = 7$, $p = 11$, $x = 13$.
4. Обчисліть перші 7 бітів псевдовипадкової послідовності, що її згенерує генератор BBS, який ініціалізується значеннями $q = 5$, $p = 13$, $x = 19$.
5. Обчисліть перші 5 чисел псевдовипадкової послідовності генератора на основі RSA з використанням відкритого ключа $e = 5$, $n = 33$.

# Лабораторний практикум із криптографії

## ЛАБОРАТОРНА РОБОТА № 1

### Шифрувальна система на основі шифру Цезаря

***Мета:***

створити просту криптографічну систему на основі шифру Цезаря та дослідити її роботу.

***Обладнання:***

- персональний комп'ютер з встановленою операційною системою Windows;
- будь-яка мова програмування.

***Завдання***

1. Створити просту криптографічну систему на основі шифру заміни.
2. Перевірити її роботу.

***Література***


1. Масленников М. Практическая криптография. – СПб: БХВ, 2003. – 464 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М: ДМК, 2004. – 616 с.


***Теоретичні відомості***

Під *криптографією* будемо розуміти галузь знань, що належить до методів і засобів перетворення повідомлень у незрозумілу для сторонніх осіб форму, а також перевірки істинності цих повідомлень.

Під *криптоаналітикою* будемо розуміти засоби і методи, спрямовані на подолання криптографічного захисту.

Сукупність криптографії та криптоаналітики називається *криптологією*.

*Розшифровуванням* будемо називати відновлення вихідного повідомлення при відомому ключі шифрування.

*Дешифруванням* будемо називати процес відновлення вихідного повідомлення при невідомому ключі шифрування.

Таким чином, ті, кому призначено шифроване повідомлення, його *розшифровують*, а ті, хто перехоплює його, намагаються *дешифрувати*.

Клод Шеннон у своїй праці «Теория связи в секретных системах» узагальнив накопичений до нього досвід розробки шифрів. З'ясувалося, що навіть у дуже складних шифрувальних системах можна виділити як складові частини шифри заміни, шифри перестановки та їх комбінації. Деякі відомості про ці прості шифри можна знайти в художній літературі, зокрема в «Золотому жуці» Едгара По і «Танцюючих чоловічках» Артура Конан Дойла.

Розглянемо два приклади простих шифрів.

*Шифр сцитала*. Цей шифр відомий із часів війни Спарти проти Афін у V ст. до н.е. Для його реалізації використовувалась т. зв. сцитала – циліндричний жезл певного діаметра. На сциталу намотували вузьку папірусну стрічку, на якій писали повідомлення вздовж осі сцитали. Коли стрічку знімали, на ній залишалися незрозумілі літери. Для розшифровки повідомлення адресат намотував стрічку на такий самий жезл і читав повідомлення. В цьому шифрі перетворення оригінального тексту в шифрований зводиться до перестановки літер оригінального тексту. Тому клас таких шифрів отримав назву *шифру перестановки*.

*Шифр Цезаря*. Цей шифр реалізує таке перетворення відкритого тексту: кожна літера замінюється третьою після неї літерою алфавіту, який вважається написаним по колу, тобто після «я» йде «а». Зауважимо, що Цезар замінював її третьою літерою, але можна замінювати й будь-якою іншою. Головне, щоб адресат цього повідомлення знав величину й напрямок зсуву. Клас шифрів, до якого належить шифр Цезаря, називається *шифрами заміни*.

Із цього, напевне, зрозуміло, що створення надійного шифру є задачею непростою. Тому бажано збільшити час життя шифру, але тут зростає ймовірність того, що криптоаналітики супротивника зможуть розкрити шифр і прочитати зашифровані повідомлення. Якщо в шифрі є змінний «ключ», то його заміна приводить до того, що розроблені супротивником методи вже не дадуть ефекту.

Під *ключем* будемо розуміти змінний елемент шифру, який застосовується для шифрування конкретного повідомлення. Наприклад, у шифрі сцитала ключем є діаметр жезлу, а у шифрі Цезаря – величина і напрямок зсуву літер шифротексту відносно літер відкритого.

Описані міркування зумовили те, що безпека шифрованих повідомлень у першу чергу стала забезпечуватися ключем. Сам шифр, шифромашина або принцип шифрування прийнято вважати відомими суперникові й доступними для попереднього вивчення, але в шифрі з'явився невідомий елемент – ключ, від якого істотно залежать застосовувані перетворення інформації. Тепер користувачі, перш ніж обмінятися шифрованими повідомленнями, повинні обмінятися ключем, за допомогою якого можна прочитати зашифроване повідомлення. А для криптоаналітиків, які хочуть прочитати перехоплене повідомлення, основною задачею є знаходження ключа.

*Принципи частотного криптоаналізу.* Встановлено, що в будь-якій мові літери абетки зустрічаються неодноразово. Якщо взяти достатньо великий текст (близько мільйона символів) загального змісту та підрахувати частоту, з якою кожна літера абетки зустрічається в цьому тексті, ми побачимо, що найчастіше в українських текстах зустрічається літера «О» (0.082), а в російських та англійських – «Е» (0.071 та 0.12 відповідно). Звичайно, в залежності від тематики тексту, частотні характеристики його змінюються, але тенденція залишається незмінною.

На цьому факті ґрунтується метод частотного криптоаналізу. Якщо метод шифрування «перехопленої шифровки» не приховує частотних особливостей мови (а саме таким і є шифр Цезаря), то криптоаналітики виконують такі дії:

1. Підраховують відносні частоти, з якими кожна літера абетки зустрічається в «перехопленому» повідомленні. Робиться це за формулою: *частота = кількість / довжина*; де *кількість* – скільки разів літера зустрічається в повідомленні; *довжина* – кількість літер у повідомленні.

2. Літеру з найбільшою відносною частотою ототожнюють із літерою, яка має найбільшу частоту в частотній таблиці.

3. Визначають величину зсуву.

4. Пробують дешифрувати повідомлення з визначеною в п. 3 величиною зсуву. Якщо отримано логічний зв'язний текст, повідомлення вважається дешифрованим. Якщо зв'язного тексту не отриманю, процедуру продовжують.

5. Літеру з найбільшою відносною частотою ототожнюють із літерою, яка має другу найбільшу частоту в таблиці.

6. Пробують дешифрувати повідомлення, перебираючи частотну таблицю, поки не отримують зв'язного тексту.

Наведеним методом Вам необхідно користуватися для криптоаналізу в цій лабораторній роботі.

***Практична частина***

1. Підгрупа розбивається на пари за бажанням.
2. Один із членів пари пише програму шифрування та розшифрування тексту шифром Цезаря. Програма шифрування повинна читати файл із набраним текстом; шифрувати текст із довільним зміщенням, значення якого вводиться з клавіатури; виводити зашифрований текст у файл. Програма розшифровування повинна задовольняти такі умови: читати файл із зашифрованим текстом; розшифровувати його; видавати на екран розшифрований текст.
3. Другий із членів пари пише програму криптоаналізу методом частотного аналізу. Програма криптоаналізу повинна задовольняти такі вимоги: а) читати «перехоплений» зашифрований файл; б) розраховувати відносну частоту, з якою зустрічається кожний символ у «перехопленому» файлі; в) підставити замість літер, щой найчастіше зустрічаються в повідомленні, відповідні літери з частотної таблиці (див. Додаток Б); г) визначати ключ, тобто величину зсуву; д) розшифровувати повідомлення з цим значенням зсуву та виводити розшифроване повідомлення на екран.
4. Перший із пари зашифровує повідомлення і передає його для криптоаналізу другому. Другий дешифрує отримане повідомлення методом частотного криптоаналізу і знаходить правильний ключ шифрування (величину зміщення).

**Звіт із лабораторної роботи повинен містити:**

1. Протоколи дій обох членів пари.
2. Розшифровані тексти та значення ключа, знайдені за допомогою криптоаналізу.

***Контрольні запитання та завдання***

1. Що називається криптографією? Для чого вона використовується?
2. Що називається криптоаналітикою? Для чого вона використовувається?
3. Що таке криптологія?
4. Яка різниця між розшифровкою і дешифровкою?
5. Що називається ключем шифрування? Для чого він використовується?
6. Які Ви знаєте типи шифрів? Наведіть приклади.
7. Охарактеризуйте шифр сцитала.
8. Охарактеризуйте шифр Цезаря.
9. Які б Ви запропонували методи розкриття шифру сцитала?

## ЛАБОРАТОРНА РОБОТА № 2

### Шифрувальна система на основі шифру простої заміни

***Мета:***

створити криптографічну систему на основі шифру простої заміни та дослідити її роботу.

***Обладнання:***

- персональний комп'ютер із встановленою операційною системою Windows;
- будь-яка мова програмування.

***Завдання***

1. Створити криптографічну систему на основі шифру простої заміни.
2. Перевірити її роботу.

***Література***

1. Масленников М. Практическая криптография. – СПб: БХВ, 2003. – 464 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М: ДМК, 2004. – 616 с.

***Теоретичні відомості***

Під *криптографією* будемо розуміти галузь знань, що належить до методів і засобів перетворення повідомлень у незрозумілу для сторонніх осіб форму, а також перевірки істинності цих повідомлень.

Під *криптоаналітикою* будемо розуміти засоби і методи, спрямовані на подолання криптографічного захисту.

Сукупність криптографії та криптоаналітики називається *криптологією*.

*Розшифровуванням* будемо називати відновлення вихідного повідомлення при відомому ключі шифрування.

*Дешифруванням* будемо називати процес відновлення вихідного повідомлення при невідомому ключі шифрування.

Таким чином, ті, кому призначено шифроване повідомлення, його *розшифровують*, а ті, хто перехоплює його, намагаються *дешифрувати*.

Розглянемо невеликий приклад. Припустимо, що відкрите повідомлення складається із символів алфавіту і пробілу. Спробуємо попрацювати з російською мовою, частотну таблицю якої подано в Додатку Б.Нехай у нас є таблиця 1, що задає відповідність між символами і числами від 0 до 32.

*Таблиця 1*

**Таблиця заміни при шифруванні**

| А | Б | В | Г | Д | Е | Ж | З | И | Й | К |
|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Л | М | Н | О | П | Р | С | Т | У | Ф | Х |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я | |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |

Шифрування методом простої заміни полягає в даному випадку в тому, що кожна літера повідомлення замінюється числом згідно з табл. 1. Звичайно, найкращим є той варіант, коли набір чисел у таблиці генерується випадково і служить ключем для розшифровування повідомлення. У нашому простому прикладі повідомлення МОЯ ПЕРВАЯ ШИФРОГРАММА у зашифрованому на табл.1 вигляді буде таким: «12 14 31 32 15 05 16 02 00 31 32 24 08 20 16 14 03 16 00 12 12 00».

Для розшифровування повідомлення необхідно провести обернену заміну згідно з табл. 1. Отже, вона виступає ключем, який використовується як для шифрування, так і для розшифровування повідомлення.

Системи шифрування, яка використовує такий самий ключ для шифрування і розшифровування повідомлень, називається *симетричною*. Якщо повідомлення шифрують за допомогою одного ключа, а розшифровують за допомогою іншого, така система має назву *асиметричної*.

Дуже часто використовують кілька алгоритмів шифрування, наприклад, зсув на певну кількість знаків алфавіту, а потім застосування простої заміни за допомогою табл.1. Таке невелике ускладнення шифрування може привести до значного ускладнення при його дешифровці, чого, в принципі, і прагнуть досягти при застосуванні криптографічних систем.

***Практична частина***

1. Підгрупа розбивається на пари за бажанням.

2. Один з членів пари модифікує програму ЛР № 1 так, щоб вона спочатку використовувала алгоритм Цезаря, як у ЛР № 1, а потім, при другому проході, виконувала просту заміну згідно з генерованою таблицею заміни типу табл. 1.
3. Система шифрування повинна задовольняти такі вимоги: а) читати відкритий текст повідомлення з текстового файла; б) запитувати величину і напрям зсуву; в) генерувати таблицю простої заміни за допомогою генератора випадкових чисел; в) записувати зашифроване повідомлення, величину зсуву і таблицю заміни в текстовий файл для передачі.
4. Система розшифрування має відповідати таким вимогам: а) читати з текстового файла зашифроване повідомлення разом із величиною зсуву; б) виводити розшифроване повідомлення в текстовий файл або на екран монітора.
5. Протокол дій та отримані результати включіть у звіт із лабораторної роботи.

**Звіт з лабораторної роботи повинен містити:**

1. Протоколи дій обох членів пари.
2. Розшифровані тексти та значення ключа, знайдені за допомогою криптоаналізу.

***Контрольні запитання та завдання***

1. Що називається криптографією? Для чого вона використовується?
2. Що називається криптоаналізом? Для чого він використовується?
3. Що називається криптологією?
4. Яка різниця між розшифровкою і дешифровкою?
5. Охарактеризуйте шифр простої заміни. Які його переваги і недоліки?
6. Які Ви знаєте типи шифрів? Наведіть приклади.
7. Які системи шифрування називаються симетричними?
8. Які системи шифрування називаються асиметричними?
9. Як можна, на Вашу думку, модифікувати дану систему шифрування?

## ЛАБОРАТОРНА РОБОТА № 3

### Дослідження афінної системи шифрування Цезаря

***Мета роботи:***

ознайомитися з різними шифрами простої заміни (шифрами підстановки) та методами їх криптоаналізу.

***Теоретичні відомості***

При шифруванні заміною (підстановкою) символи відкритого тексту

замінюються символами того ж або іншого алфавіту за завчасно встановленим правилом заміни. У шифрах простої заміни кожен символ вихідного тексту замінюється символами того ж алфавіту однаково протягом усього тексту. Часто шифри простої заміни називають шифрами моноалфавітної підстановки.

***Афінна система підстановок Цезаря***

У системі шифрування Цезаря використовувалися лише адитивні властивості множини цілих $Z_n$. Однак символи множини $Z_n$ можна також множити за модулем $n$. Застосовуючи одночасно операції додавання та множення за модулем $n$ над елементами множини $Z_n$, можна отримати систему підстановок, яку називають ***афінною системою підстановок Цезаря***.

$$E_{a,b}(t) = at + b \ (mod\ n),$$

де $a$, $b$ – цілі числа, $0 < a,b < n$, НСД $(a,n) = 1$.

Зауважимо, що перетворення $E_{ab}(t)$ є взаємно однозначним відображенням на множині $Z_n$ тільки в тому випадку, якщо найбільший спільний дільник (НСД($a,n$)) дорівнює одиниці, тобто $a$ і $m$ повинні бути взаємно простими числами.

Наприклад, для англійської мови, нехай $n = 26$, $a = 3$, $b = 5$. Тоді, очевидно, НСД (3,26) = 1, і ми отримуємо таке співвідношення між числовими кодами літер:

*Таблиця 1*

**Співвідношення між числовими кодами літер**

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t + 5$ | 5 | 8 | 11 | 14 | 17 | 20 | 23 | 0 | 3 | 6 | 9 | 12 | 15 |

| $t$ | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $3t + 5$ | 18 | 21 | 24 | 1 | 4 | 7 | 10 | 13 | 16 | 19 | 22 | 25 | 2 |

Перетворюючи числа в літери англійської мови, отримаємо такі співвідношення між літерами відкритого тексту та шифротексту:

*Таблиця 2*

**Відповідність між літерами відкритого тексту та шифротексту**

| A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | I | L | O | R | U | X | A | D | G | J | M | P |

| N | O | Q | P | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | V | Y | B | E | H | K | N | Q | T | W | Z | C |

Відкрите повідомлення **NIGHT** перетворюється у шифротекст **SDXAK.**

Перевагою афінної системи є зручне керування ключами – ключі шифрування і розшифровування подаються в компактній формі у вигляді пари чисел ($a$, $b$).

Недоліки афінної системи аналогічні недолікам усіх шифрів підстановки.

Афінна система використовувалася на практиці кілька століть тому, а сьогодні її застосування обмежується більшою мірою ілюстрацією основних положень криптографії.

**Система Цезаря з ключовим словом**

Система шифрування Цезаря ***з ключовим словом*** – це також моноалфавітна система підстановки. Особливістю цієї системи є можливість використання ключового слова для зміщення та зміни порядку символів у алфавіті підстановки.

Виберемо деяке число $k$, $0 < k < 25$ і слово або коротку фразу як ***ключове слово***. Бажано, щоб усі літери ключового слова були різними. Нехай вибрано слово **DIPLOMAT** як ключове слово та число $k = 5$.

Ключове слово записується під літерами абетки, починаючи з літери, числовий код якої збігається з вибраним числом $k$.

*Таблиця 3*

**Підписування ключового слова**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| | | | | | **D** | **I** | **P** | **L** | **O** | **M** | **A** | **T** |

| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | O | Q | P | R | S | T | U | V | W | X | Y | Z |
| | | | | | | | | | | | | |

Решту літер абетки підстановки записують після ключового слова в алфавітному порядку.

*Таблиця 4*

**Записування решти літер алфавіту підстановки**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| V | W | X | Y | Z | **D** | **I** | **P** | **L** | **O** | **M** | **A** | **T** |

| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | O | Q | P | R | S | T | U | V | W | X | Y | Z |
| B | C | E | F | G | H | J | K | N | Q | R | S | U |

Тепер ми маємо підстановку для кожної літери довільного повідомлення.

Отже, відкрите повідомлення **SEND MORE MONEY** («вишліть ще грошей») шифрується як **HZBY TCGZ TCBZS**.

Варто зазначити, що вимога про відмінність усіх літер ключового слова необов'язкова. Можна просто записати ключове слово (або фразу) без повторення однакових літер. Наприклад, ключова фраза

**КАК ДЫМ ОТЕЧЕСТВА НАМ СЛАДОК И ПРИЯТЕН**

і число $k = 3$ породжують таку таблицю підстановок:

*Таблиця 5*

**Таблиця підстановок**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| А | Б | В | Г | Д | Е | Ж | З | И | Й | К |
| Ь | Э | Ю | **К** | **А** | **Д** | **Ы** | **М** | **О** | **Т** | **Е** |

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|
| Л | М | Н | О | П | Р | С | Т | У | Ф | Х |
| **Ч** | **С** | **В** | **Н** | **Л** | **И** | **П** | **Р** | **Я** | Б | Г |

| 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я |
| Ж | З | Й | У | Ф | Х | Ц | Ш | Щ | Ъ |

Перевага сисеми Цезаря з ключовим словом у тому, що кількість можливих ключових слів практично невичерпна.

Недоліком цієї системи є можливість розкриття шифротексту на основі аналізу частот появи літер.

***Підготовка до роботи***

1. Підгрупа розбивається на пари за бажанням.
2. Один із пари пише програму шифрування за афінною системою Цезаря для текстових файлів.
3. Програма повинна задовольняти такі умови: читати файл із відкритим текстом з диска; шифрувати його за допомогою афінної системи Цезаря з ключем, що вводиться з клавіатури або з ключовим словом; зберігати шифрограму в текстовому файлі. Блок-схему такого алгоритму подано на рис. 1. Приклад зовнішнього вигляду форми введення даних такої програми зображено на рис. 2.
4. Другий з учасників пари пише програму розшифрування. Програма повинна відповідати таким умовам: читати з файла отриману


*Рис. 1. Блок-схема алгоритму шифрування*

шифрограму зі значеннями ключа; обчислювати таблицю заміни та виводити розшифрований текст на екран.

***Виконання роботи***

1. Зашифровані першим учасником пари файли передаються для розшифрування другому учаснику.
2. За результатами лабораторної роботи напишіть звіт.
3. Звіт повинен містити:
   а) протоколи дій учасників пари;
   б) розшифрований текст;
   в) ключ афінної системи Цезаря, при якому отримано результат.

***Контрольні запитання та завдання***

1. Охарактеризуйтк афінну систему шифрування Цезаря.
2. Охарактеризуйте систему шифрування Цезаря з ключовим словом.
3. Назвіть переваги і недоліки цих систем.
4. Порівняйте афінну систему зі звичайною системою Цезаря. В чому полягає основна перевага афінної системи?
5. Запропонуйте спосіб криптоаналізу афінної системи Цезаря.
6. Які вимоги висуваються до вибору ключів афінної системи Цезаря?

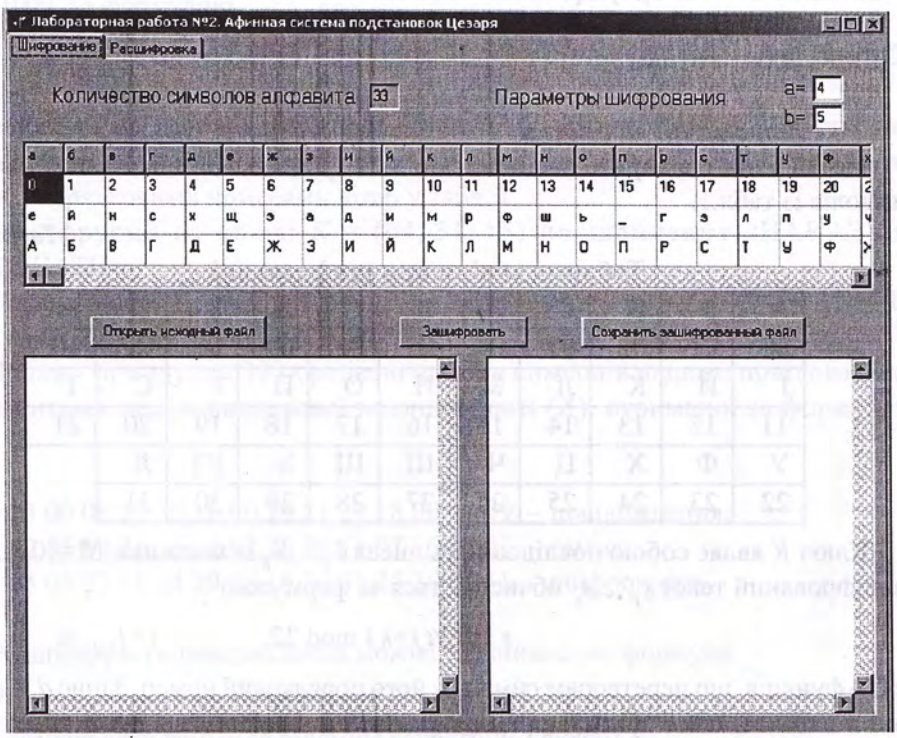



*Рис. 2. Приклад програми шифрування афінним методом Цезаря*

## ЛАБОРАТОРНА РОБОТА № 4

### Шифрувальна система на основі шифру гаммування

***Мета:***

створити криптографічну систему на основі шифру гаммування та дослідити її роботу.

***Обладнання:***

- персональний комп'ютер із встановленою операційною системою Windows;
- будь-яка мова програмування.

***Завдання***

1. Створити криптографічну систему на основі шифру гаммування.
2. Перевірити її роботу.

***Література***


1. Масленников М. Практическая криптография. – СПб: БХВ, 2003. – 464 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М: ДМК, 2004. – 616 с.


***Теоретичні відомості***

Нехай у нас є відкрите повідомлення $t_1,...,t_n$, що являє собою послідовність символів із табл.1.

*Таблиця 1*

**Таблиця заміни при шифруванні**

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І |
|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Ї | Й | К | Л | М | Н | О | П | Р | С | Т |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я | |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | |

Ключ $K$ являє собою послідовність чисел $k_1,...,k_n$ із множини **M**={0,…,31}. Зашифрований текст $s_1,...,s_n$ обчислюється за формулою

$$s_i = (C(t_i)+k_i) \bmod 32, \qquad i=1,...,n, \qquad (1)$$

де С – функція, що перетворює символ у його порядковий номер. Запис $d = (a+b) \bmod n$ означає, що $d$ збігається із залишком від ділення на $n$ суми чисел $a+b$, наприклад, $1 = (4+7) \bmod 10$.

Розшифрувати повідомлення можна за допомогою формули

$$t_i = C^{-1}(s_i + (33 - k_i)) \bmod 32.$$

У даному випадку через $C^{-1}$ позначають функцію, яка виконує обернене перетворення: перетворює порядковий номер із множини 0-31 у символ алфавіту.

Будемо вважати, що елементи ключа $k_i$ вибираються рівноймовірно і незалежно з множини $M$.

Визначений у такий спосіб шифр називається *шифром гаммування з випадковою рівноймовірною гаммою* (*гаммою* прийнято називати послідовність чисел $k_1,...,k_n$, що додається за модулем до шифрованого повідомлення).

При такому методі шифрування довжина шифрограми збігається з довжиною відкритого повідомлення. Це полегшує криптоаналітикам задачу дешифрування повідомлення за частотним словником. Щоб сховати довжину повідомлення, його можна доповнити пробілами до певної фіксованої довжини, яку не перевищує стандартне повідомлення.

Розглянемо приклад використання спрощеного шифру гаммування. Спрощення полягають у заміні випадкової гамми псевдовипадковою на простому ключі з трьох випадкових чисел.

Нехай ключ $K = (Y_1;Y_2;Y_3)$ складається з трьох чисел, які вибрано випадково незалежно і рівноймовірно з множини (0,...,31). За допомогою рекурентного співвідношення $Y_t = (Y_{t-1} + Y_{t-3}) \bmod 32$ формується послідовність $Y_1,..., Y_{n+1}$ для $t > 3$. Далі, за формулою

$$Z_t = (Y_t + Y_{t+1}) \bmod 32, \qquad t = 1,\ldots,n \tag{2}$$

обчислюється псевдовипадкова послідовність $Z_1,..., Z_n$, що використовується як випадкова гамма. Шифрування полягає в додаванні за модулем 32 елементів гамми з порядковими номерами літер у табл. 1.

Зашифруємо на ключі $K$ = (04, 31, 15) повідомлення «НАКАЗУЮ НАСТУПАТИ».

Послідовність $Y_1, \ldots, Y_{n+1}$ у даному випадку буде мати вигляд «04 31 15 19 18 01 20 06 07 27 01 08 03 04 12 15 19».

Додамо за модулем 32 порядкові номери символів нашого повідомлення з елементами псевдовипадкової послідовності ($Z_t$), отриманої за формулою (2):

16 00 13 00 08 22 30 16 00 20 21 22 18 00 21 09 – повідомлення;
03 14 02 05 19 21 26 13 02 28 09 11 07 16 27 02 – гамма;
19 14 15 05 27 11 24 29 02 16 30 01 25 16 16 11 – шифрограма.

Розшифрувати повідомлення можна за допомогою формули

$$t_i = C^{-1}((s_i + (32 - k_i)) \bmod 32, \tag{3}$$

якщо згенерувати гамму за відомим секретним ключем $K$.

Тепер розглянемо метод дешифрування нашого повідомлення. Загальна кількість текстів із 16 літер складе $32^{16}$, а кількість різних ключів у даному випадку $32^3$=32768. Отже, кількість можливих варіантів дешифрування при невідомому ключі не перевищує 32768. Маючи перехоплену шифрограму, методом «грубої сили» (тобто прямого перебору всіх ключів) нам знадобиться не більше 32768 варіантів для дешифрування повідомлення. Зрозуміло, що під час роботи будуть зустрічатися абсолютно «нечитабельні», а всі логічні повідомлення необхідно відфільтрувати за допомогою простої логіки (тобто можливе таке повідомлення чи ні). Згідно з дослідженнями К. Шеннона, кількість змістовних текстів із 16 літер в англійській мові приблизно $10^5$. Приблизно така ж оцінка справедлива і для російської мови. Імовірність появи серед усіх варіантів дешифровок іншого змістовного тексту менша від $2{,}6 \times 10^{-20}$. Для порівняння – ймовірність вгадати 6 чисел із 36 більша за $10^{-10}$. Якщо криптоаналітик буде відкидати по одному неправильному повідомленню за секунду, то йому потрібно буде на те, щоб продивитися всі повідомлення, приблизно 9 годин. Таким чином, трудозатрати ручного дешифрування даного повідомлення методом прямого перебору ключів складуть 500 годин. Процес можна прискорити, якщо застосувати ЕОМ. У цьому випадку результати генерування всіх ключів і дешифрування всіх варіантів буде отримано практично миттєво, і лише 9 годин знадобиться криптоаналітику для відбору істинного повідомлення серед хибних.

У сучасних алгоритмах шифрування використовують ключі набагато більшої довжини, ніж у даному прикладі. Зокрема, в широко відомому алгоритмі DES ключ має об'єм 56 бітів. Із таким ключем на метод «грубої сили» знадобиться 2 млрд. років при швидкості один варіант за секунду. Варіант «грубої сили» можна значно прискорити, якщо врахувати неможливі сполучення літер мови.

Якщо ми маємо частину зашифрованого повідомлення, наприклад, знаємо, що воно починається з «НАК», ми зможемо знайти три перших знаки гамми. Для цього складемо систему з трьох рівнянь-так. У таблиці заміни «Н» має номер 16, «А» – 00, «К» – 13. Це означає, що виконуються співвідношення: $(16+ Z_1) \bmod 32 = 19$; $(00+ Z_2) \bmod 32 = 14$ і $(13+ Z_3) \bmod 32 = 15$. Звідси можна знайти перших три знаки гамми: 3; 14; 2. Тепер можна скласти систему з трьох рівнянь

$$(Y_1+Y_2) \bmod 32 = 3;\ (Y_2+Y_3) \bmod 32 = 14;\ (Y_3+Y_4) \bmod 32 = 2,$$

розв'язуючи яку, ми отримаємо секретний ключ: $Y_1 = 4$; $Y_2 = 31$; $Y_3 = 15$.

Таким чином, ми можемо розділити криптографічні алгоритми на три великих групи.

До першої групи належать досконалі алгоритми, які не піддаються розкриттю при правильному використанні (наприклад, алгоритм одноразових блокнотів, або шифр гаммування випадковою рівноймовірною гаммою).

Другу групу формують шифри, що допускають неоднозначне дешифрування. Наприклад, така ситуація виникає, коли шифрують за допомогою простої заміни коротке повідомлення.

До третьої групи належать шифри, криптограми яких можуть бути

однозначно розшифровані, однак складність дешифрування забезпечується трудомісткістю алгоритму дешифрування. Тобто в останньому випадку стійкість шифру забезпечується складністю алгоритмів дешифрування.

***Практична частина***

1. Створіть просту криптографічну систему, яка використовує шифр гаммування, описаний у теоретичній частині.
2. Система шифрування повинна задовольняти такі вимоги: 1) читати відкрите повідомлення з текстового файла й застосовувати просту заміну за допомогою табл. 1; 2) запитувати сеансовий секретний ключ, що складається з 3 чисел; 3) генерувати гамму за формулою (2), довжина якої дорівнює довжині відкритого повідомлення; 4) шифрувати за допомогою формули (1) відкрите повідомлення й записувати його у файл.
3. Система розшифровування повинна відповідати таким вимогам: 1) читати шифрограму з текстового файла; 2) запитувати сеансовий секретний ключ; 3) генерувати гамму на основі ключа; 4) розшифровувати шифрограму за допомогою формули (3) і виводити її у файл та на екран монітора.

***Контрольні запитання та завдання***

1. Який шифр називається шифром гаммування?
2. Яким умовам повинен відповідати ідеальний шифр гаммування?
3. Що називається додаванням за модулем $n$?
4. Як залежить від ключа даний алгоритм шифрування?
5. Які умови повинен задовольняти ключ шифрування?
6. Оцініть трудомісткість криптоаналітика для дешифрування цього шифру гаммування.
7. Що потрібно знати криптоаналітику для швидкого і точного визначення ключа шифрування?
8. На які групи поділяються криптографічні алгоритми? Чим вони характеризуються?
9. Які Ви знаєте методи розкриття шифру, застосованого у цій ЛР?
10. Як, на Вашу думку, можна ускладнити цю систему шифрування?

## ЛАБОРАТОРНА РОБОТА № 5

### Система блочного шифрування S-DES

***Мета:***

створити просту криптографічну систему на основі спрощеного блочного алгоритму Simple DES (S-DES) та дослідити її роботу.

***Обладнання:***

- персональний комп'ютер з встановленою операційною системою;
- будь-яка мова програмування.

***Завдання***

1. Створити просту криптографічну систему на основі спрощеного блочного алгоритму S-DES.
2. Перевірити її роботу.

***Література***


1. Столлингс В. Криптография и защита сетей. – М.: Вильямс, 2001. – 672 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.


***Теоретичні відомості***

Спрощений DES – це алгоритм шифрування, який має, скоріше, навчальне, ніж практичне значення. За своїми властивостями він подібний до DES, але має значно менше параметрів [1].

Цей алгоритм приймає на вході 8-бітний блок відкритого тексту та 10-бітний ключ, а на виході генерує 8-бітний блок шифрованого тексту. При розшифруванні на вхід алгоритму подається 8-бітний блок шифротексту і 10-бітний ключ, а на виході генерується 8-бітний блок відкритого тексту.

Алгоритм шифрування передбачає послідовне виконання п'яти операцій: початкової перестановки *IP*; циклової функції, що складається з перестановок і підстановок; перестановки *SW*, коли дві половинки блока по 4 біти переставляються місцями; ще одного застосування циклової функції; і, нарешті, перестановки $IP^{-1}$, оберненої до початкової. Послідовне використання кількох перестановок і підстановок, як це показано у [1-2], значно ускладнює криптоаналіз.

Циклова функція приймає на вході не лише блок тексту, а й 8-бітний цикловий підключ, який утворюється з 10-бітного ключа.

Блок-схему алгоритму подано на рис. 1. З цього рисунка видно, що, оскільки це симетричний криптоалгоритм, він використовує для шифрування та дешифрування той самий ключ. Тому ключ має бути як на передавальній, так і на приймальній стороні. З цього ключа на певних етапах шифрування й розшифрування генеруються два 8-бітних циклових підключі.

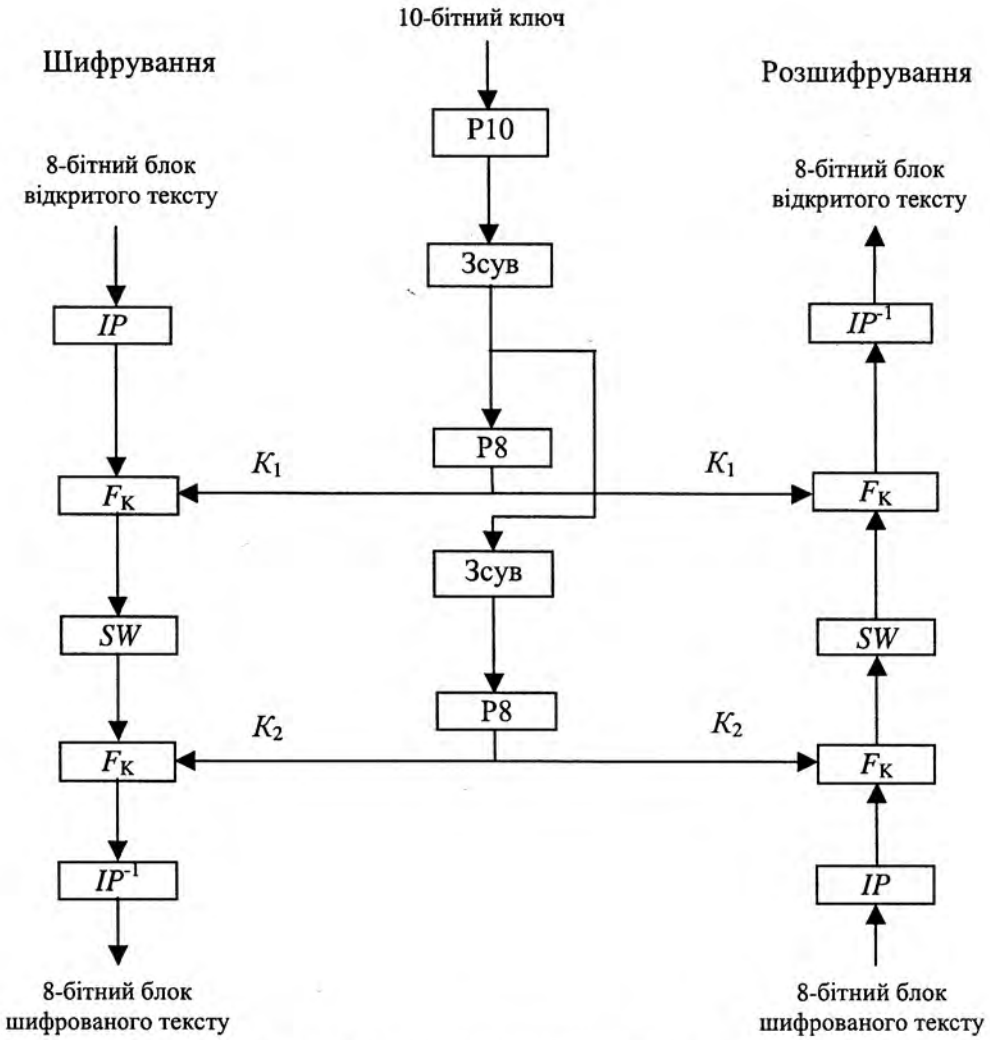



*Рис. 1. Схема спрощеного алгоритму DES*

***Процедура генерування циклових підключів***

1. Спочатку переставляються біти ключа так. Якщо 10-бітний ключ уявити у вигляді $k_1,k_2,\ldots,k_{10}$, то перестановка P10 задається формулою

$$P(k_1,k_2,k_3,k_4,k_5,k_6,k_7,k_8,k_9,k_{10}) =$$
$$=(k_3,k_5,k_2,k_7,k_4,k_{10},k_1,k_9,k_8,k_6).$$

Можна також зобразити перестановку P10 у вигляді таблиці:

| P10 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 2 | 7 | 4 | 10 | 1 | 9 | 8 | 6 |

Ця таблиця символізує позицію біта вхідних даних у вихідній послідовності: першим стає 3-й біт, другим – 5-й, третім – 2-й і т.д. Наприклад, ключ (1010000010) відповідно до цієї перестановки перетворюється в послідовність (1000001100).

2. Ключ розділяється на дві половинки, по п'ять бітів кожна. Окремо перша половина й окремо друга піддаються циклічному зсуву ліворуч на одну позицію. У нашому прикладі в результаті буде отримана послідовність (00001 11000).

3. Отримана послідовність піддається перестановці P8, у результаті якої з 10-бітного ключа обирається 8-бітна послідовність за таким правилом:

| P8 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 7 | 4 | 8 | 5 | 10 | 9 |

У результаті цієї операції ми отримаємо перший цикловий підключ ($K_1$). У нашому прикладі він буде мати вигляд (10100100).

4. Для генерування другого циклового підключа $K_2$ необхідно повернутися на крок назад, до двох 5-бітних рядків для застосування P8 і виконати для кожного з цих рядків циклічний зсув ліворуч на дві позиції. У нашому прикладі значення підключів (00001 11000) перетворяться у (00100 00011).

5. Нарешті, застосувавши до цієї послідовності перестановку P8, отримаємо другий цикловий підключ $K_2$. Для нашого прикладу результатом буде (01000011).

***Шифрування S-DES***

1. *Початкова і кінцева перестановки (IP та IP$^{-1}$).* На вхід алгоритму подається 8-бітний блок відкритого тексту, до якого застосовується початкова перестановка IP:

| IP | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 6 | 3 | 1 | 4 | 8 | 5 | 7 |

На завершальній стадії алгоритму виконується обернена перестановка IP$^{-1}$:

| IP$^{-1}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 3 | 5 | 7 | 2 | 8 | 6 |

Можна пересвідчитися, що ці дві таблиці справді обернені одна до одної тобто IP$^{-1}$(IP(M))=M.

2. *Циклова функція* $F_K$. Розіб'ємо вхідний блок тексту після IP-перестановки на два 4-бітні підблоки. Лівий 4-бітний блок позначимо $L$, а правий – $R$. Тоді циклову функцію можна записати у вигляді формули

$$F_K(L,R) = (L \oplus F(R,K_i),R). \qquad (1)$$

Тут $K_i$ означає цикловий підключ, $K_1$ або $K_2$; $\oplus$ – побітове XOR.

Тепер опишемо саму циклову функцію. На вході вона отримує 4-бітне значення $(n_1,n_2,n_3,n_4)$, тобто праву половину вхідного блока. Перша операція – операція розширення та перестановки. Її можна також зобразити таблицею:

| Розширення з перестановкою | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 2 | 3 | 2 | 3 | 4 | 1 |

Зручніше цю операцію зобразити у вигляді матриці

$$n_4\ n_1\ n_2\ n_3$$
$$n_2\ n_3\ n_4\ n_1.$$

До цього значення додається 8-бітний підключ за допомогою операції XOR. Це можна зобразити так:

$$n_4+k_1\ \ n_1+k_2\ \ n_2+k_3\ \ n_3+k_4$$
$$n_2+k_5\ \ n_3+k_6\ \ n_4+k_7\ \ n_1+k_8.$$

Перейменуємо отримані елементи:

$$p_{00}\ p_{01}\ p_{02}\ p_{03}$$
$$p_{10}\ p_{11}\ p_{12}\ p_{13}.$$

Перші чотири біти (тобто перший рядок цієї матриці) далі подаються на вхід модуля заміни ($S$-матриці) $S_0$, на виході якого отримується 2-бітна послідовність. Другий рядок матриці подається на вхід другого модуля заміни $S_1$, на виході якого також отримується 2-бітна послідовність.

Модулі $S_0$ і $S_1$ задаються так:

$$S_0 = \begin{matrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 1 \end{matrix}; \qquad S_1 = \begin{matrix} 1 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{matrix}.$$

Рядки та стовпчики нумеруються починаючи з нуля.

Ці модулі заміни працюють ось як. Перший і четвертий біти вхідної послідовності вважаються двійковим представленням номера рядка, а другий та третій – номера стовпця. Елемент, що знаходиться на перетині цих рядка і стовпця, задає двобітне вихідне значення. Наприклад, якщо $(p_{00},p_{03}) = (00)$ та $(p_{01}, p_{02})=(10)$, то вихідні два біти задаються значенням, яке знаходиться на перетині 0-го рядка та 3-го стовпця, тобто буде числом 3, а у двійковому представленні –11.

Аналогічну операцію виконують і з другим рядком $p_{10}\ p_{11}\ p_{12}\ p_{13}$.

Після застосування матриць заміни результат піддають перестановці P4 за таким законом:

| P4 | | | |
|---|---|---|---|
| 2 | 4 | 3 | 1 |

Результат перестановки P4 і буде результатом функції $F_K$. Отримана послідовність бітів додається за модулем 2 з лівою половиною L вхідного блока і буде новою лівою половиною. Права половина передається на вихід циклу без змін.

3. *Перестановка підблоків*. Як бачимо, за один цикл цикловою функцією обробляється лише ліва половина відкритого тексту, права половина залишається без змін. Для того щоби зашифрувати й праву половину, використовується другий цикл, однак на його вхід треба подати переставлені підблоки: $L$ і $R$ поміняти місцями. Для цього й служить функція $SW$ – перемикач блоків.

Після переставляння підблоків один цикл алгоритму закінчено.

До переставлених підблоків знову застосовується циклова функція, як це описано вище. При другому виклику циклової функції розширення з перестановкою, модулі $S_0$, $S_1$ та $P4$ залишаються тими ж, тільки використовується підключ $K_2$.

По закінченні другого циклу виконується $IP^{-1}$-перестановка й роботу алгоритму закінчено, тобто на виході маємо зашифрований текст.

***Розшифрування зашифрованого тексту***

Як видно з рис. 1, розшифрування зашифрованого тексту виконується аналогічно шифруванню, за винятком того, що ключі подаються у зворотному порядку.

***Практична частина***

1. Використовуючи довільну мову програмування, створіть програму, яка б шифрувала та розшифровувала текстові повідомлення за допомогою криптосистеми S-DES.
2. Перевірте правильність функціонування криптосистеми, зашифрувавши та розшифрувавши текст. Порівняйте отриманий результат із відкритим текстом.
3. Здійсніть шифрування та розшифрування двох інших текстів на різних криптографічних ключах і зробіть висновок про правильність функціонування створеної програми.
4. Зробіть висновок із лабораторної роботи.

Підготуйте звіт із лабораторної роботи. Звіт повинен містити: а) протокол Ваших дій; б) код програми; в) відкриті та зашифровані тексти, а також криптографічні ключі, на яких виконувалося шифрування; г) результати порівняння розшифрованих текстів із відкритими; д) висновки з лабораторної роботи; е) відповіді на контрольні запитання.

***Контрольні запитання та завдання***

1. До якого класу криптоалгоритмів належить спрощений DES, розглянутий у цій роботі?
2. Які особливості цього алгоритму?
3. Чим він відрізняється від алгоритму DES?
4. Вкажіть особливості обробки криптографічного ключа у S-DES.
5. Опишіть структуру циклової функції у S-DES.
6. Чи не можна було би зробити S-DES з одним циклом? Чому?
7. Для чого потрібні перестановки IP та $IP^{-1}$?

## ЛАБОРАТОРНА РОБОТА № 6

### Дослідження розсіювальних властивостей S-DES

***Мета:***

1. Дослідити властивості розсіювання та перемішування бітів у спрощеній криптосистемі S-DES.
2. Дослідити залежність вихідних параметрів від структури блоків заміни алгоритму.

***Обладнання:***

- персональний комп'ютер із встановленою операційною системою;
- будь-яка мова програмування.

***Завдання***

1. Вивчити розсіювальні та перемішувальні властивості S-DES від структури ключа та вхідного тексту.
2. Вивчити залежність вихідних бітів від структури матриць заміни криптоалгоритму.

***Література:***


1. Столлингс В. Криптография и защита сетей. – М.: Вильямс, 2001. – 672 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.


***Теоретичні відомості***

Спрощений DES – це алгоритм шифрування, який має, скоріше, навчальне, ніж практичне значення. За своїми властивостями він подібний до DES, але має значно менше параметрів [1].

Цей алгоритм приймає на вході 8-бітовий блок відкритого тексту та 10-бітний ключ, а на виході генерує 8-бітний блок шифрованого тексту. При розшифруванні на вхід алгоритму подається 8-бітний блок шифротексту і 10-бітний ключ, а на виході генерується 8-бітний блок відкритого тексту.

Алгоритм шифрування передбачає послідовне виконання п'яти операцій: початкової перестановки *IP*; циклової функції, що складається з перестановок і підстановок; перестановки *SW*, коли дві половинки блока по 4 біти переставляються місцями; ще одного застосування циклової функції; і, нарешті, перестановки $IP^{-1}$, оберненої до початкової. Послідовне використання кількох перестановок і підстановок, як це показано у [1-2], значно ускладнюють криптоаналіз.

Циклова функція приймає на вході не лише блок тексту, а й 8-бітний цикловий підключ, який утворюється з 10-бітного ключа.

Блок-схему алгоритму подано на рис. 1. З цього рисунка видно, що, оскільки це симетричний криптоалгоритм, він використовує для шифрування та

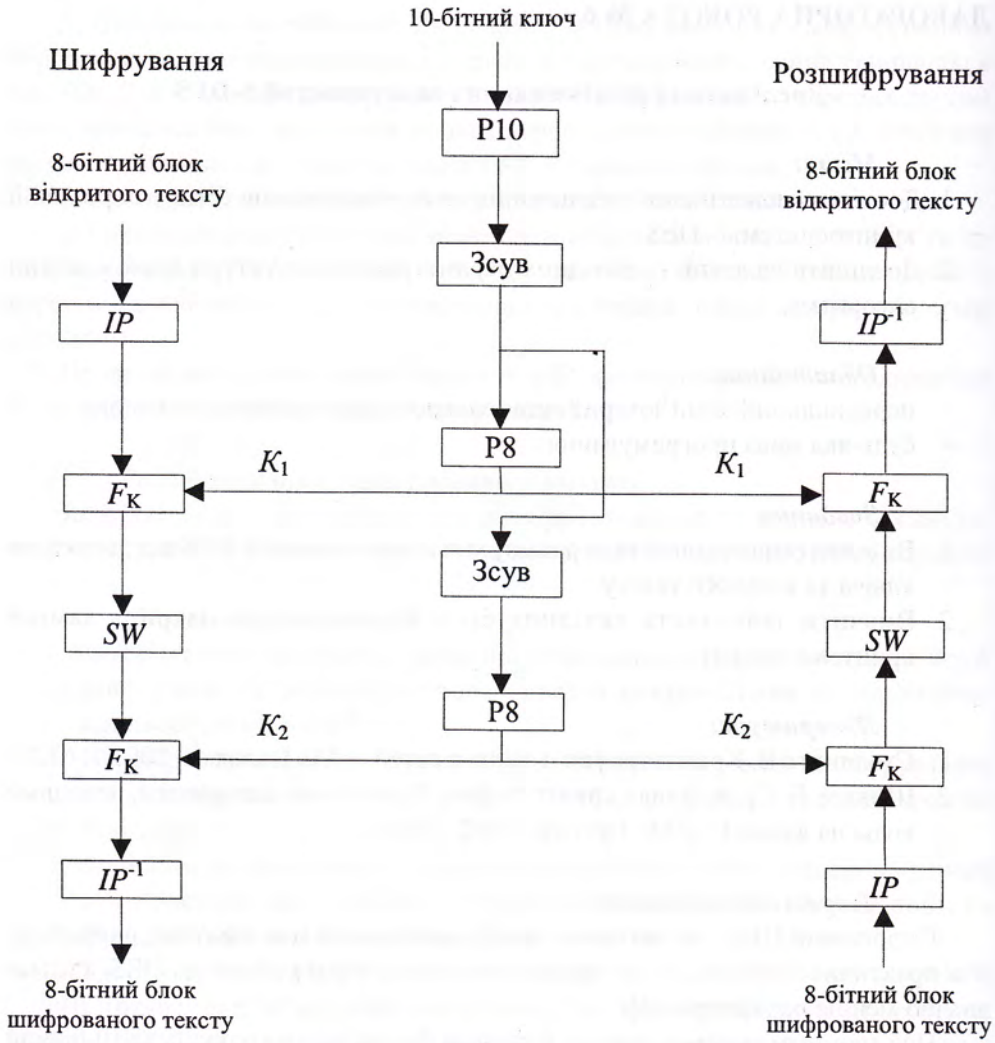



*Рис. 1. Схема спрощеного алгоритму DES*

розшифрування той самий ключ. Тому ключ має бути як на передавальній, так і на приймальній стороні. З цього ключа на певних етапах шифрування та розшифрування генеруються два 8-бітних циклових підключа.

***Процедура генерування циклових підключів***

1. Спочатку переставляються біти ключа так. Якщо 10-бітний ключ уявити у вигляді $k_1,k_2,\ldots,k_{10}$, то перестановка P10 задається формулою

$$P(k_1,k_2,k_3,k_4,k_5,k_6,k_7,k_8,k_9,k_{10}) =$$
$$=(k_3,k_5,k_2,k_7,k_4,k_{10},k_1,k_9,k_8,k_6).$$

Можна також зобразити перестановку P10 у вигляді таблиці:

| P10 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 2 | 7 | 4 | 10 | 1 | 9 | 8 | 6 |

Ця таблиця символізує позицію біта вхідних даних у вихідній послідовності: першим стає 3-й біт, другим – 5-й, третім – 2-й і т.д. Наприклад, ключ (1010000010) відповідно до цієї перестановки перетворюється в послідовність (1000001100).

2. Ключ розділяється на дві половинки по п'ять бітів кожна. Окремо перша половина й окремо друга піддаються циклічному зсуву ліворуч на одну позицію. У нашому прикладі в результаті буде отримана послідовність (00001 11000).

3. Отримана послідовність піддається перестановці P8, в результаті якої з 10-бітного ключа обирається 8-бітна послідовність за таким правилом:

| P8 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 7 | 4 | 8 | 5 | 10 | 9 |

У результаті цієї операції ми отримаємо перший цикловий підключ ($K_1$). У нашому прикладі він буде мати вигляд (10100100).

4. Для генерування другого циклового підключа $K_2$ необхідно повернутися на крок назад, до двох 5-бітних рядків до застосування P8 та виконати для кожного з цих рядків циклічний зсув ліворуч на дві позиції. У нашому прикладі значення підключів (00001 11000) перетворяться у (00100 00011).

5. Нарешті, застосувавши до цієї послідовності перестановку P8, отримаємо другий цикловий підключ $K_2$. Для нашого прикладу результатом буде (01000011).

***Шифрування S-DES***

1. *Початкова і кінцева перестановки (IP та $IP^{-1}$).* На вхід алгоритму подається 8-бітний блок відкритого тексту, до якого застосовується початкова перестановка IP:

| IP | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 6 | 3 | 1 | 4 | 8 | 5 | 7 |

На завершальній стадії алгоритму виконується обернена перестановка $IP^{-1}$:

| $IP^{-1}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 3 | 5 | 7 | 2 | 8 | 6 |

Можна пересвідчитися, що ці дві таблиці дійсно обернені одна до одної, тобто $IP^{-1}(IP(M)) = M$.

2. *Циклова функція* $F_K$. Розіб'ємо вхідний блок тексту після *IP*-перестановки на два 4-бітні підблоки. Лівий 4-бітний блок позначимо $L$, а правий – $R$. Тоді циклову функцію можна записати у вигляді формули

$$F_K(L,R) = (L \oplus F(R,K_i),R). \qquad (1)$$

Тут $K_i$ означає цикловий підключ, $K_1$ або $K_2$; $\oplus$ – побітове XOR.

Тепер опишемо саму циклову функцію. На вході вона отримує 4-бітне значення $(n_1,n_2,n_3,n_4)$, тобто праву половину вхідного блока. Перша операція – операція розширення та перестановки. Її можна також зобразити таблицею:

| Розширення з перестановкою | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 2 | 3 | 2 | 3 | 4 | 1 |

Зручніше цю операцію зобразити у вигляді матриці

$$n_4\ n_1\ n_2\ n_3$$
$$n_2\ n_3\ n_4\ n_1.$$

До цього значення додається 8-бітний підключ за допомогою операції XOR. Це можна зобразити так:

$$n_4+k_1\ \ n_1+k_2\ \ n_2+k_3\ \ n_3+k_4$$
$$n_2+k_5\ \ n_3+k_6\ \ n_4+k_7\ \ n_1+k_8.$$

Перейменуємо отримані елементи:

$$p_{00}\ p_{01}\ p_{02}\ p_{03}$$
$$p_{10}\ p_{11}\ p_{12}\ p_{13}.$$

Перші чотири біти (тобто перший рядок цієї матриці) далі подаються на вхід модуля заміни ($S$-матриці) $S_0$, на виході якого отримується 2-бітна послідовність. Другий рядок матриці подається на вхід другого модуля заміни $S_1$, на виході якого також отримується 2-бітна послідовність.

Модулі $S_0$ і $S_1$ задаються так:

$$S_0 = \begin{matrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 1 \end{matrix}; \quad S_1 = \begin{matrix} 1 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{matrix}.$$

Рядки та стовпчики нумеруються, починаючи з нуля.

Ці модулі заміни працюють так. Перший і четвертий біти вхідної послідовності вважаються за двійковим представленням номера рядка, а другий та третій – номера стовпця. Елемент, що знаходиться на перетині цих рядка і стовпця, задає двобітне вихідне значення. Наприклад, якщо $(p_{00},p_{03})$ = (00) та $(p_{01}, p_{02})$ = (10), то вихідні два біти задаються значенням, яке знаходиться на перетині 0-го рядка та 3-го стовпчика, тобто буде числом 3, а у двійковому представленні –11.

Аналогічну операцію виконують і з другим рядком $p_{10}\ p_{11}\ p_{12}\ p_{13}$.

Після застосування матриць заміни результат піддають перестановці $P4$ за таким законом:

| P4 | | | |
|---|---|---|---|
| 2 | 4 | 3 | 1 |

Результат перестановки $P4$ і буде результатом функції $F_K$. Отримана послідовність бітів додається за модулем 2 з лівою половиною $L$ вхідного блока і буде новою лівою половиною. Права половина передається на вихід циклу без змін.

3. *Перестановка підблоків.* Як бачимо, за один цикл цикловою функцією обробляється лише ліва половина відкритого тексту, права половина залишається без змін. Для того щоби зашифрувати й праву половину, використовується другий цикл, однак на його вхід треба подати переставлені підблоки: $L$ і $R$ поміняти місцями. Для цього й служить функція $SW$ – перемикач блоків.

Після переставляння підблоків один цикл алгоритму закінчено.

До переставлених підблоків знову застосовується циклова функція, як це описано вище. При другому виклику циклової функції розширення з перестановкою, модулі $S_0$, $S_1$ та $P4$ залишаються тими ж, тільки використовується підключ $K_2$.

По закінченні другого циклу виконується $IP^{-1}$-перестановка і роботу алгоритму закінчено, тобто на виході маємо зашифрований текст.

***Розшифрування зашифрованого тексту***

Як видно з рис. 1, розшифрування зашифрованого тексту виконується аналогічно шифруванню, за винятком того, що ключі подаються у зворотному порядку.

***Практична частина***

1. Використовуючи будь-яку мову програмування, створіть програму, яка б шифрувала та розшифровувала текстові повідомлення за допомогою криптосистеми S-DES.

2. Використовуючи створену програму, вивчіть її властивості розсіювання в залежності від вхідних бітів ключа. Для цього оберіть ключ «0000000000» та зашифруйте на ньому вибраний текст. Після цього змініть молодший біт ключа на 1 та виконайте шифрування того ж тексту. Скільки бітів зашифрованого тексту змінилося? Виконайте таку ж операцію з кожним бітом ключа. Поясніть отримані результати. Зміна якого біта ключа приводить до найбільших і найменших змін у зашифрованому тексті? Змініть ключ на «1111111111» та повторіть усі запропоновані операції. Оберіть випадковий ключ і виконайте всі дослідження для нього. Поясніть отримані результати.

3. Зашифруйте на випадковому ключі текст «00000000». Поясніть отримані результати. Змінюйте по одному біту вхідного тексту та послідовно зашифровуйте його на тому ж ключі. Скільки бітів зашифрованого тексту змінилося? Зміна яких бітів відкритого тексту найбільше і найменше впливає на зашифрований? Ті ж операції виконайте для тексту «11111111» та довільного відкритого тексту. Поясніть отримані результати.

4. Виконайте шифрування довільного тексту на довільному ключі. Починаючи з першого рядка, змініть структуру матриць заміни, щоразу

виконуючи шифрування. Як змінюється результат? Визначте, яка зміна структури матриць привела до найкращих і найгірших результатів.

Порівняйте розсіювальні властивості спрощеного алгоритму з такими ж властивостями реального DES. Дані про DES можна знайти у книзі [2] та Інтернеті.

5. Здійсніть атаку методом повного перебору ключів («грубою силою») та визначте скільки часу знадобилося Вашому комп'ютеру для повного перебору ключів. Скільки всього ключів довелося перебрати? Скільки ключів довелося перебрати для досягнення успіху атаки?

Зробіть висновок із лабораторної роботи. Підготуйте звіт із лабораторної роботи. Звіт повинен містити: а) результати досліджень; б) протокол Ваших дій; в) висновки з досліджень впливу зміни ключа, вхідного тексту та матриць заміни на результати шифрування; г) відповіді на контрольні запитання.

***Контрольні запитання та завдання***

1. Оцініть криптостійкість цього алгоритму по відношенню до атаки «грубою силою».
2. Що спільного у цього алгоритму з реальним DES?
3. Які відмінності цього алгоритму від DES?
4. Оцініть швидкодію Вашої реалізації цього алгоритму (в мегабітах або мегабайтах за секунду), виконавши тестове шифрування.
5. Які відмінності у швидкодії порівняно з DES? Дані про DES можна знайти у книзі Б. Шнаєра [2].
6. Якою Ви вважаєте оптимальну структуру матриць заміни цього алгоритму і чому?
7. Що таке «слабкі ключі»? До якого результату приводить шифрування на таких ключах? Наведіть приклади «слабких ключів».

## ЛАБОРАТОРНА РОБОТА № 7

### Використання шифрувальної системи RSA для цифрового підпису

***Мета:***

створити просту криптографічну систему цифрового підпису на основі системи шифрування RSA та дослідити її роботу.

***Обладнання:***

- персональний комп'ютер із встановленою операційною системою Windows;
- будь-яка мова програмування.

***Завдання***

1. Створити просту криптографічну систему цифрового підпису на основі шифру RSA.
2. Перевірити її роботу.

***Література***


1. Масленников М. Практическая криптография. – СПб: БХВ, 2003. – 464 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М: ДМК, 2004. – 616 с.


***Теоретичні відомості***

Криптографічна система RSA (Rivest, Shamir, Adleman) належить до криптографічних систем із відкритим ключем. Її стійкість зумовлена великими проблемами при знаходженні розкладання великих простих чисел на множники.

Для того, щоб організувати передачу шифрованих повідомлень за допомогою криптосистеми RSA, необхідно зробити таке:

1. За допомогою спеціальних алгоритмів згенерувати два великих простих числа $p$ і $q$, які необхідно тримати в таємниці.
2. Повідомити відправнику повідомлень (або розмістити у відкритому каталозі) число $n = pq$, а також випадкове ціле число $E$, взаємно просте з добутком $(p\text{-}1)\times(q\text{-}1)$.
3. Для розшифровки повідомлень, зашифрованих на відкритому ключі $n$, $E$, отримувачу необхідно мати число $D$, яке є мультиплікативним оберненим числа $E$ за модулем $(p\text{-}1)(q\text{-}1)$, тобто $DE=1 \bmod (p\text{-}1)(q\text{-}1)$. Знайти таке число дуже просто, оскільки найбільший спільний дільник $E$ і $(p\text{-}1)(q\text{-}1)$ якраз і дорівнює одиниці за вибором $E$.

Отже, відправник знає свій закритий ключ, $n$, $E$, а отримувач, окрім того, знає ще й свій секретний ключ $D$.

Довільне відкрите повідомлення можна зобразити у вигляді послідовності цілих чисел із деякого інтервалу. Будемо вважати, що відправник передає секретне повідомлення у вигляді $X_1,\ldots,X_n$ $0 < X_i < n\text{-}1$, для всіх $i$ від 1 до $k$.

Відправник для кожного блока $X_i$ вираховує

$$C_i = (X_i^E) \bmod n \qquad (1)$$

і передає $C_i$ відкритим каналом зв'язку.

Маючи $n$, $E$ і $C_i$, отримувач може розшифрувати повідомлення, використовуючи співвідношення

$$X_i = (C_i^D) \bmod n. \qquad (2)$$

Розглянемо як приклад випадок $p = 3$, $q = 11$, $n = 3\times11 = 33$, $E = 7$, $D = 3$. Легко переконатися, що кожне з чисел $E = 7$ і $DxE = 21$ взаємно просте з $(p\text{-}1)(q\text{-}1) = 20$. Для передачі повідомлення M = «02» відправнику треба обчислити

$C = (2^7) \bmod 33 = 29$. Отримувач може розшифрувати повідомлення за допомогою такої операції: $X = 29^3 \bmod 33 = 2$.

Якщо ж ми маємо текстове повідомлення, алфавіт якого пронумеровано від 00 до 32 (з пробілом), тоді можна зашифрувати довільне повідомлення російською мовою. Наприклад, якщо ми маємо повідомлення «ПРОВЕРИМ ЗНАНИЕ АРИФМЕТИКИ», то В зашифрованому вигляді на ключі $n = 33$, $E = 7$ воно буде мати вигляд

27 25 20 29 14 25 02 12 32 28 07 00 07 02 14 32 00 25 02 26 12 14 06 02 10 02

Зрозуміло, що шифром у цьому випадку є шифр простої заміни за табл. 1.

*Таблиця 1*

**Таблиця заміни при шифруванні**

| А | Б | В | Г | Д | Е | Є | Ж | З | И | І |
|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Ї | Й | К | Л | М | Н | О | П | Р | С | Т |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ю | Я | |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |

Система шифрування RSA може бути застосована для цифрового підпису. У випадку підпису повідомлення $M$ відправник обчислює $P=M^E \bmod n$. Отримувач, який має $M$ і $P$, перевіряє справедливість співвідношення $P^D=M \bmod n$ і впевнюється у справжності повідомлення $M$.

***Приклад.*** Нехай $p = 3$, $q = 11$, $n = 3\times11 = 33$, $E = 7$, $D = 3$. Тоді відправник повідомлення $M$ = «02» обчислює цифровий підпис $P = 2^7 \bmod 33 = 29$ і відправляє повідомлення «02, 29» отримувачу. Той, в свою чергу, перевіряє справжність повідомлення «02», обчисливши $M = (29^3) \bmod 33 = 2$.

Насправді підписують не саме повідомлення, а його хеш-образ.

Ми будемо користуватися найпростішою хеш-функцією, яка дуже недосконала й може викликати значні колізії. Однак вона дуже проста і не потребує витрат машинного часу, а також складного програмування. Ця функція просто підсумовує всі значення символів із табл. 1 за модулем 33:

$$H(M) = \sum_{i=1,n} m_i \bmod 33. \qquad (3)$$

До отриманого в такий спосіб числа застосовують алгоритм прикладу, отримуючи зашифрований цифровий підпис.

Отримувач, маючи повідомлення і цифровий підпис, розшифровує текст повідомлення, знаходить хеш-функцію від нього за формулою (3), розшифровує цифровий підпис і порівнює отримані значення. Якщо вони однакові, повідомлення й цифровий підпис істинні.

***Практична частина***

1. Підгрупа розбивається на пари за бажанням. Один студент виконує цифровий підпис повідомлення, а другий цей підпис перевіряє.

2. Перший студент створює криптографічну систему на основі алгоритму RSA, викладеного в теоретичній частині.

3. Система шифрування повинна задовольняти такі вимоги: 1) читати з текстового файла відкрите повідомлення; 2) шифрувати повідомлення за допомогою публічного ключа; 3) обчислювати просту хеш-функцію повідомлення у вигляді (3); 4) обчислювати цифровий підпис знайденої хеш-функції і записувати його значення у файл (той самий, в якому міститься повідомлення, або інший).

4. Другий студент створює систему перевірки електронного підпису. Система повинна задовольняти такі вимоги: 1) читати з текстового файла зашифроване повідомлення та цифровий підпис; 2) розшифровувати повідомлення за допомогою приватного ключа *D* і знаходити хеш-функцію (3); 3) розшифровувати цифровий ключ і порівнювати отримане значення хеш-функції з обчисленим у п.2); 4) робити висновок про істинність отриманого повідомлення й цифрового підпису.

5. Замініть цифровий підпис довільним числом із діапазону 0-32 і знову перевірте, чи «помітить» програма розшифровки заміну.

6. Зробіть висновок про якість роботи Вашої системи електронного підпису.

***Контрольні запитання та завдання***

1. До яких систем шифрування належить система RSA?
2. Який алгоритм шифрування використовується в системі RSA?
3. На чому ґрунтується криптостійкість системи RSA?
4. Які обмеження накладаються на ключі криптосистеми RSA?
5. Які ви знаєте способи розкриття шифру криптосистеми RSA?
6. Як можна застосувати криптосистему RSA для цифрового підпису повідомлень?
7. Як Ви розумієте поняття хеш-функції? Які бувають хеш-функції?
8. Вкажіть недоліки запропонованої в цій ЛР хеш-функції.
9. Для чого, на Вашу думку, застосовують цифровий підпис документів?

## ЛАБОРАТОРНА РОБОТА № 8

### Відкритий розподіл криптографічних ключів

***Мета:***

ознайомитися з відкритим розподілом криптографічних ключів за допомогою алгоритму Діффі-Хеллмана.

***Обладнання:***

- персональний комп'ютер з установленою операційною системою.
- будь-яка мова програмування.

***Завдання***

1. Створити просту систему відкритого розподілу криптографічних ключів за алгоритмом Діффі-Хеллмана.
2. Перевірити її роботу.

***Література***


1. Молдовян Н.А., Молдовян А.А. Введение в криптосистемы с открытым ключом. – СПб: БХВ, 2005. – 288 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М: ДМК, 2004. – 616 с.


***Теоретичні відомості***

Проблема адміністрування криптографічними ключами вважається основним недоліком симетричних криптоалгоритмів. Цю проблему можна розв'язати за допомогою асиметричної криптографії, тобто взагалі не використовувати симетричні криптоалгоритми. Однак такий підхід вважають нераціональним, оскільки асиметричні алгоритми працюють значно повільніше за симетричні й не можуть використовуватися в ряді важливих криптографічних операцій.

Одним із популярних алгоритмів узгодження ключів є алгоритм Діффі-Хеллмана.

Нехай учасники інформаційного обміну, сторони $A$ і $B$, домовилися використати цей алгоритм для обміну ключами. Для цього необхідно виконати такі обчислення.

Спочатку $A$ і $B$ обирають велике просте число $p$, модуль системи. Для цього числа $p$ обирають первісний корінь $a$. Числа $p$ і $a$ відкрито передають каналами зв'язку, так щоб їх мали обидві сторони.

Далі виконується такий протокол:

1) $A$ генерує ціле велике випадкове число $x$ і відправляє $B$ число:

$$X = a^x \bmod p;$$

2) $B$ генерує велике ціле випадкове число $y$ і відправляє $A$ число:

$$Y = a^y \bmod p;$$

3) $A$ обчислює:

$$k = Y^x \bmod p;$$

4) $B$ обчислює:

$$k' = X^y \bmod p.$$

І $k$, і $k'$ дорівнюють $k = k' = a^{xy} \bmod p$. Отже, сторони $A$ і $B$ отримали однаковий криптографічний ключ, не пересилаючи його каналами зв'язку. Ніхто з осіб, що прослуховують цей канал, не зможе обчислити значення ключа. Адже їм відомі тільки $p$, $a$, $X$, $Y$, а для знаходження ключа необхідно розв'язати задачу дискретного логарифмування. Тому А і В мають цілком таємний ключ, який більше ніхто не знає.

Вибір $a$ і $p$ може помітно впливати на безпеку системи. Найголовніше, це те, що $p$ повинно бути великим, таким, щоби задача дискретного логарифмування у скінченному полі була складною обчислювальною проблемою. Можна обирати довільне $a$, яке є первісним коренем за модулем $p$; немає причин, за якими не можна було б обрати $a$ найменшим із можливих, навіть однорозрядним. Навіть необов'язково, щоб $a$ було первісним коренем, воно повинно лише утворювати досить велику підгрупу мультиплікативної групи за модулем $p$.

***Практична частина***

Складіть програму, яка б реалізовувала алгоритм обміну криптографічними ключами за Діффі-Хеллманом. Для цього:

1. Оберіть просте число $p$ і його первісний корінь $a$. Число $p$ повинно бути як мінімум 4-значним. Для пошуку простих чисел можна скористатися кодом, наведеним у додатку.
2. Для обраного $p$ знайдіть первісний корінь $a$. Для цього скористайтеся малою теоремою Ферма.
3. Для обраних $p$ і $a$ виконайте обчислення $k$ і $k'$ та порівняйте їх.
4. Зробіть висновок про якість роботи Вашої системи обміну ключами.
5. Підготуйте звіт із лабораторної роботи. Звіт повинен містити: а) результати досліджень; б) протокол Ваших дій; в) код програми; г) висновок з лабораторної роботи; д) відповіді на контрольні запитання.

***Контрольні запитання та завдання***

1. У чому полягає проблема розподілу ключів у симетричних криптосистемах?
2. Як асиметричні криптосистеми розв'язують цю проблему?
3. Які переваги і недоліки комбінованих криптосистем?
4. Охарактеризуйте метод відкритого розподілу ключів за алгоритмом Діффі-Хеллмана.
5. На чому ґрунтується крипостійкість цього методу?
6. Що таке односторонні функції?
7. Які односторонні функції Ви знаєте? Охарактеризуйте кожну з них.
8. Які ще алгоритми розподілу криптографічних ключів Ви знаєте? Охарактеризуйте їх.

Програма для пошуку простих чисел

```
var
   prime:array[0..1000000]of integer;
   n,i,j:integer;
   o:boolean;
 begin
   assign(input,'input.txt');reset(input);
   assign(output,'output.txt');rewrite(output);
   read(n);
   i:=2;prime[0]:=0;
   while prime[prime[0]]<n do
   begin
     o:=true;
     for j:=1 to prime[0] do
       if prime[j]>trunc(sqrt(i)) then break
       else
         if i mod prime[j] =0 then
         begin
           o:=false;
           break;
         end;
     if o then
     begin
       inc(prime[0]);write(i,' ');
       if prime[0] mod 100 =0 then
         writeln;
       prime[prime[0]]:=i;
     end;
     inc(i)
   end;
 end.
```

## ЛАБОРАТОРНА РОБОТА № 9

### Потоковий шифр на основі генератора BBS

***Мета:***

створити потоковий шифр з використанням програмного генератора псевдовипадкових послідовностей BBS.

***Обладнання:***

- персональний комп'ютер із встановленою операційною системою;
- будь-яка мова програмування.

***Завдання***

1. Створити потоковий шифр на основі генератора BBS.
2. Перевірити його роботу.

***Література:***


1. Молдовян Н.А., Молдовян А.А. Введение в криптосистемы с открытым ключом. – СПб: БХВ, 2005. – 288 с.
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
3. Столлингс В. Криптография и защита сетей. – М.: Вильямс, 2001. – 672 с.


***Теоретичні відомості***

Програмний генератор двійкових послідовностей BBS (назву утворено від перших літер його авторів – Ленори та Мануеля Блум та Майка Шуба, Blum-Blum-Shub) вважають одним із найсильніших програмних генераторів псевдовипадкових послідовностей. Він вважається криптографічно стійким і може використовуватися у серйозних криптографічних застосуваннях [2].

Нехай є два простих числа, $p$ і $q$, причому $p \equiv q \equiv 3 \bmod 4$. Добуток цих чисел $n = pq$ називається цілим числом Блума. Оберемо ще одне випадкове число, $x$, взаємно просте з $n$ та обчислимо $x_0 \equiv x \bmod n$. Це число вважається стартовим числом генератора.

Далі можна обчислити наступні біти послідовності за формулою $x_i \equiv x_{i-1}^2 \bmod n$ та $s_i \equiv x_i \bmod 2$. Останнє означає, що виходом генератора обирається молодший біт числа $x_i$. Отже, можемо записати:

$x_0 = x^2 \bmod n;$
for $i = 1$ to $\infty$;
  $x_i = (x_{i-1})^2 \bmod n;$
  $s_i = x_i \bmod 2.$

Найцікавішою властивістю генератора BBS є те, що для визначення значення $i$-го біта зовсім необов'язково знати всі попередні $i$-1 бітів. Для безпосереднього обчислення значення $i$-го біта достатньо знати $p$ і $q$.

Безпека цієї схеми ґрунтується на складності розкладання $n$ на множники. Число $n$ можна опублікувати, так що кожен зможе генерувати біти за допомогою цього генератора. Однак поки криптоаналітик не розкладе $n$ на множники, він не зможе передбачити вихід генератора.

Більше того, генератор BBS непередбачуваний як у правому, так і в лівому напрямках. Це означає, що отримавши послідовність бітів, криптоаналітик не зможе передбачити ні наступний, ні попередній біти послідовності. Причиною цього є не якийсь заплутаний механізм генерації, а математика розкладання $n$ на множники.

***Приклад***

$p = 19;$ $q = 23;$

$p = q \equiv 3 \bmod 4;$

$n = 437;$

$x = 233.$

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $x_i$ | 101 | 150 | 213 | 358 | 123 | 271 | 25 | 188 |
| $S_i$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

Обов'язковою умовою, що накладається на зародок $x$, повинно бути таке: а) $x$ – просте; б) $x$ не ділиться на $p$ і на $q$.

Цей генератор повільний, але є спосіб його прискорення. Як вказано у [2], як біти псевдовипадкової послідовності можна використовувати не один молодший біт, а $\log_2 m$ молодших бітів, де $m$ – довжина числа $x_i$. Відносна повільність цього генератора не дозволяє використовувати його для потокового шифрування (цей недолік зі зростанням швидкодії комп'ютерів стає менш актуальним), а от для високонадійних застосувань, зокрема генерування ключів, він вважається кращим за багато інших.

Однак у цій лабораторній роботі, яка лише демонструє застосування потокового шифру, де швидкодія не може бути визначальним параметром, ми будемо використовувати саме генератор BBS.

***Практична частина***

Складіть програму, яка б реалізовувала потоковий шифр на основі генератора BBS. Шифрування інформації повинно виконуватися за допомогою побітового XOR двійкового представлення чергового символу відкритого тексту та послідовності генератора BBS.

Оберіть два тризначних числа $p$, $q$, обчисліть модуль $n$ і випадкове число $x$. $n$ та $x$ збережіть, оскільки їх треба передати на приймальну сторону.

За допомогою створеної програми зашифруйте та розшифруйте повідомлення.

Зробіть висновок про якість роботи Вашої системи шифрування.

Підготуйте звіт із лабораторної роботи. Звіт повинен містити: а) результати

досліджень; б) протокол Ваших дій; в) код програми; г) висновок із лабораторної роботи; д) відповіді на контрольні запитання.

***Контрольні запитання та завдання***

1. На чому ґрунтується крипостійкість генератора BBS?
2. Які переваги і недоліки потокових шифрів?
3. Які переваги і недоліки шифрів однократного гаммування?
4. Які вимоги висуваються до генераторів випадкових і псевдовипадкових послідовностей?
5. Запропонуйте кілька реалізацій генераторів випадкових послідовностей за допомогою комп'ютера.

## ЛАБОРАТОРНА РОБОТА № 10

### Використання CryptoAPI операційної системи Windows XP для створення програмного забезпечення захисту інформації

***Мета роботи:***

ознайомити студентів з методами роботи з CryptoAPI ОС Windows 2000/XP.

***Література***


1. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М.: ДМК Пресс, 2004.
2. Щеглов А.Ю. Защита компьютерной информации от несанкционированного доступа. – СПб.: Наука и техника, 2004.
3. Проскурин В.Г., Крутов С.В., Мацкевич И.В. Защита в операционных системах. – М.: Радио и связь, 2000.
4. Щербаков А, Домашев А. Прикладная криптография. Использование и синтез криптографических интерфейсов. – М.: Русская редакция, 2003.
5. Деднев М.А., Дыльнов Д.В., Иванов М.А. Защита информации в банковском деле и электронном бизнесе. – М.: Кудиц-образ, 2004.


***Теоретичні відомості***

Правильне функціонування підсистеми безпеки комп'ютерної системи вимагає реалізації ряду функцій загального призначення, пов'язаних з перетворенням вмісту об'єктів системи (файлів, записів бази даних тощо) або з обчислення деяких спеціальних функцій, які суттєво залежать від вмісту об'єктів. До таких функцій належать алгоритми контролю цілісності об'єктів,

аутентифікації та авторизації об'єктів, що керують процесами, а також алгоритми підтримання конфіденційності інформації, що міститься в об'єктах комп'ютерної системи.

Міжнародні та національні стандарти описують ряд добре відомих і вивчених функцій захисного характеру, зокрема алгоритми хешування MD5, MD2, SHA тощо; алгоритми генерування та перевірки електронного цифрового підпису RSA, DSS та інших. Усі ці алгоритми мають різні механізми викликів (зокрема, різну довжину аргументів). Це, у свою чергу, означає, що вони несумісні між собою.

Тому задача вбудовування тих чи інших захисних механізмів в операційну систему на основі якогось одного алгоритму буде виглядати неефективною, особливо, якщо ця ОС розповсюджується в різних регіонах земної кулі. В цьому випадку логічна побудова «шаруватої» структури, де окремий шар, реалізований, скажімо, як набір динамічних бібліотек, відповідає за захист інформації. Цей спосіб досить універсальний і широко застосовується в сімействі операційних систем Windows. Таким способом можна розв'язати великий клас задач, пов'язаних з універсалізацією ОС: від національних налаштувань системи до реалізації різноманітних засобів безпеки.

Зрозуміло, що такі структури повинні мати «відкритий інтерфейс», тобто бути детально документованими для того, щоби програмісти могли використати засоби цієї структури при створенні прикладного програмного забезпечення, в тому числі й для захисту інформації.

Сьогодні є достатня кількість криптографічних інтерфейсів, однак найбільшої популярності набув інтерфейс від Microsoft - Microsoft CryptoAPI. Зараз використовується CryptoAPI версії 2.0. Причина популярності цього інтерфейсу полягає в тому, що Microsoft інтенсивно впровадила захисні механізми CryptoAPI у свої операційні системи та прикладне програмне забезпечення. Сучасні ОС сімейства Windows містять багато криптографічних підсистем різного призначення як прикладного рівня, так і рівня ядра. Провідну роль у цьому відіграють саме функції CryptoAPI, зокрема базові криптографічні функції, сукупність яких створює інтерфейс CryptoAPI 1.0.

Інтерфейс CryptoAPI 2.0 містить як базові криптографічні функції, так і функції, що реалізують перетворення вищого рівня – роботу із сертифікатами X.509, обробку криптографічних повідомлень PKCS#7 та інші функції, що підтримують інфраструктуру відкритих ключів. Однак набір базових криптографічних функцій цього інтерфейсу утворює CryptoAPI 1.0. Отже, функції CryptoAPI 1.0 утворюють криптографічне ядро прикладного рівня для сучасних операційних систем лінійки Windows.

Загальну архітектуру CryptoAPI 1.0 подано на рис. 1.

Усі функції інтерфейсу зосереджено в бібліотеці advapi32.dll. Ці процедури виконують ряд допоміжних функцій та викликають бібліотеки, де безпосередньо реалізовано відповідні криптографічні перетворення. Такі бібліотеки називають ***криптопровайдерами***. Криптопровайдери мають стандартний набір функцій,

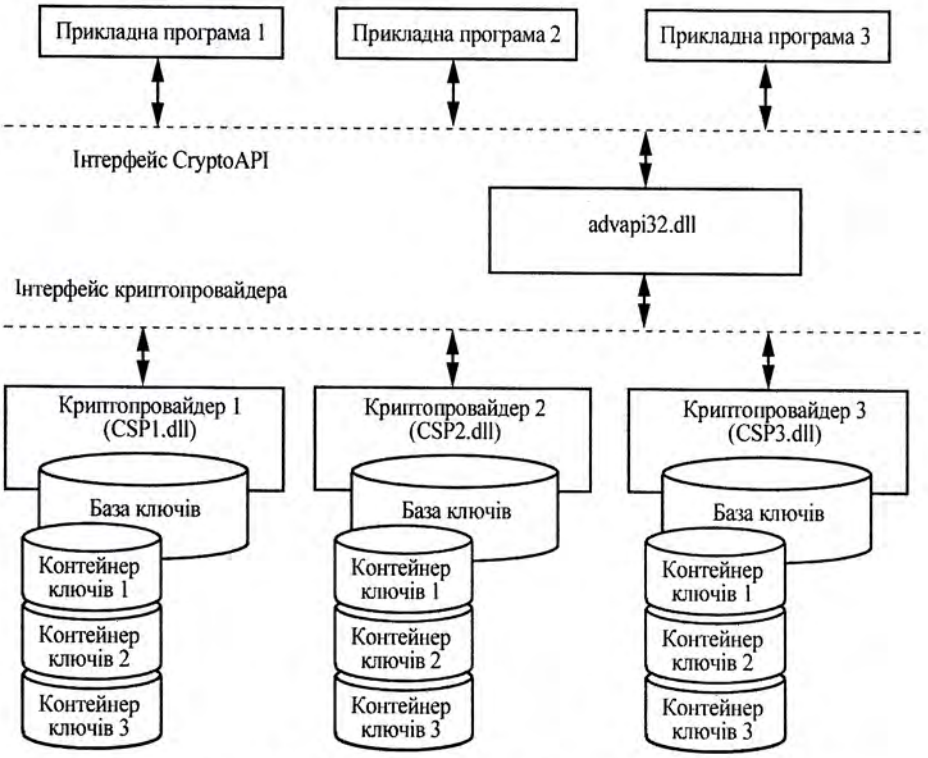



*Рис. 1. Загальна архітектура CryptoAPI 1.0*

який налічує 23 обов'язкових і 2 необов'язкових процедури. Ці процедури подано в табл. 1.

Як бачимо з таблиці, CryptoAPI 1.0 підтримує всі основні методи криптографічного перетворення даних: від генерування криптографічних послідовностей випадкових чисел до операцій з електронним цифровим

*Таблиця 1*

**Функції криптопровайдерів**

| Функція | Короткий опис |
|---|---|
| *CryptAcquireContext* | Використовується для створення дескриптора ключового контейнера в рамках визначеного криптопровайдера (КП) |
| *CryptContextAddRef* | Збільшує на одиницю лічильник посилань на дескриптор КП |
| *CryptEnumProviders* | Використовується для отримання першого та наступних доступних КП |
| *CryptEnumProviderTypes* | Використовується для отримання першого та наступних доступних типів КПє |

| Функція | Короткий опис |
|---|---|
| *CryptGetDefaultProvider* | Повертає КП, встановлений за замовчуванням для вказаного типу КП |
| *CryptGetProvParam* | Повертає параметри КП |
| *CryptReleaseContext* | Вивільняє дескриптор КП |
| *CryptSetProvider*<br>*CryptSetProviderEx* | Задає тип та назву КП за замовчуванням |
| *CryptSetProvParam* | Встановлює параметри КП |
| *CryptDeriveKey* | Створює сесійні криптографічні ключі з ключового матеріалу |
| *CryptDestroyKey* | Звільняє дескриптор ключа |
| *CryptDuplicateKey* | Робить копію криптографічного ключа |
| *CryptExportKey* | Експортує криптографічні ключі із заданого контейнера |
| *CryptGenKey* | Генерує випадкові криптографічні ключі та ключові пари |
| *CryptGenRandom* | Генерує випадкову послідовність та зберігає її в буфері |
| *CryptGetKeyParam* | Повертає параметри ключа |
| *CryptImportKey* | Імпортує криптографічні ключі з ключового блока в контейнер КП |
| *CryptSetKeyParam* | Встановлює параметри ключа |
| *CryptDecrypt* | Виконує операцію розшифрування даних |
| *CryptEncrypt* | Виконує операцію зашифрування даних |
| *CryptCreateHash* | Створює хешований потік даних |
| *CryptDestroyHash* | Знищує об'єкт хеш-функції |
| *CryptDuplicateHash* | Створює точну копію хеш-об'єкта |
| *CryptGetHashParam* | Повертає параметри хеш-об'єкта |
| *CryptHashData* | Додає дані до хеш-об'єкта |
| *CryptHashSessionKey* | Підмішує до хеш-об'єкта сесійний ключ |
| *CryptSetHashParam* | Встановлює параметри хеш-об'єкта |
| *CryptSignHash* | Обчислює значення ЕЦП від значення хешу |
| *CryptVerifySignature* | Перевіряє ЕЦП заданого значення хешу |

підписом. Таким чином, знаючи інтерфейс CryptoAPI 1.0, програміст може досить легко реалізувати всі популярні криптографічні алгоритми у своїх прикладних програмах.

Програміст, який працює з цим інтерфейсом, може отримати всю необхідну інформацію про певного криптопровайдера засобами функції *CryptGetProvParam*. Перше, що необхідно при цьому знати, – це набір криптографічних стандартів, які реалізують встановлені в системі криптопровайдери.

Окрім різниці у стандартах, криптопровайдери відрізняються способом фізичної організації збереження ключової інформації. З точки зору програмування спосіб зберігання ключів значення не має, однак він дуже важливий із погляду експлуатації та безпеки комп'ютерної системи. Існуючі криптопровайдери Microsoft зберігають ключову інформацію на жорсткому диску (у реєстрі або у файлах), а провайдери інших фірм (GemPlus, Schlumberger та Infineon) – на смарт-картках.

Якщо способи фізичної організації збереження ключової інформації у криптопровайдерів різні, то логічна структура, яка визначається інтерфейсами та з якою мають справу програмісти, однакова для будь-якого типу провайдера. Ключова база визначається набором ключових контейнерів, кожен з яких має ім'я, що привласнюється йому при створенні, а потім використовується для роботи з ним. У ключовому контейнері зберігається довготривала ключова інформація, наприклад ключові пари для цифрового підпису або несиметричної системи шифрування.

Тепер розглянемо детально, як функції інтерфейсу CryptoAPI викликають бібліотеки конкретного криптопровайдера. Кожен криптопровайдер має власне ім'я та тип. Його ім'я – просто рядок, за допомогою якого система його ідентифікує. Так, базовий криптопровайдер Microsoft має назву Microsoft Base Cryptographic Provider v1.0. Тип криптопровайдера – ціле число (в нотації C – DWORD), значення якого ідентифікує набір криптографічних алгоритмів, що підтримуються. Криптопровайдер Microsoft має тип 1, цей тип провайдера реалізує як алгоритм цифрового підпису та обміну ключів алгоритм RSA. Інший базовий криптопровайдер Microsoft, „Microsoft Base DSS and Diffie-Hellman Cryptographic Provider", має тип 13. Цей тип криптопровайдера реалізує алгоритм цифрового підпису DSS, а як алгоритм обміну ключами – протокол Діффі-Хелмана.

Отже, для роботи з набором криптопровайдерами в системному реєстрі міститься список імен усіх криптопровайдерів. З кожним ім'ям пов'язаний тип криптопровайдера та ім'я бібліотеки, яка реалізує його алгоритми.

Окрім цього, в системі міститься інформація про те, який криптопровайдер треба застосовувати, якщо користувач явно не вказав конкретне його ім'я, визначивши лише тип провайдера. Такий криптопровайдер називають провайдером за замовчуванням для заданого типу. Наприклад, для типу 1 провайдером за замовчуванням є Microsoft Base Cryptographic Provider v1.0, а

для типу 13 - Microsoft Base DSS and Diffie-Hellman Cryptographic Provider. Для визначення криптопровайдерів за замовчуванням використовують функцію *CryptGetDefaultProvider*, а для зміни цього параметра – функції *CryptSetProvider* або *CryptSetProviderEx*. Функції дозволяють встановити провайдера за замовчуванням як для поточного користувача, так і для системи в цілому (усіх користувачів). Ці параметри зберігаються у вулику реєстру HKEY_LOCAL_MACHINE. Параметри, встановлені для поточного користувача, мають пріоритет над параметрами, встановленими для всієї системи, та зберігаються у вулику реєстру HKEY_CURRENT_USER. Якщо параметри для поточного користувача відсутні, застосовуються загальносистемні.

Тепер розглянемо, як користувач починає працювати з конкретним криптопровайдером і як система викликає конкретну бібліотеку, що відповідає обраному криптопровайдеру.

Робота з певним провайдером починається з виклику функції *CryptAcquireContext*, де користувач визначає тип потрібного криптопровайдера, його назву та назву робочого ключового контейнера. В результаті роботи функція повертає користувачу дескриптор криптопровайдера (handle), за допомогою якого користувач у подальшому буде звертатися до нього та передавати його у процедури для виконання всіх необхідних криптографічних операцій.

Детальний опис контексту роботи з криптопровайдерами та приклади (мовою програмування *C*) дивіться у книжці Щербакова Л.Ю., Домашева А.В. «Прикладная криптография».

Власне бібліотеки CryptoAPI разом із файлами заголовків і допомоги постачаються у складі бібліотек MSDN.

***Практична частина***

Студентам пропонується скласти власну програму з використанням функцій CryptoAPI 1.0. Кожен варіант повинен являти собою закінчений програмний продукт, що розв'язує одну з важливих сучасних задач криптографії.

Мова програмування значення не має.

Перелік завдань подано в таблиці 2. Розподіл завдань між студентами виконує викладач, що проводить лабораторні заняття.

Результатом роботи студента є програмний продукт, що працює без помилок та виконує всі функції, які від нього вимагаються (див. табл. 2).

За результатами виконання лабораторної роботи студенти складають звіт, у якому детально описують правила роботи з розробленою програмою та наводять контрольні приклади для кожної її функціональної можливості.

*Таблиця 2*

**Вимоги до варіантів завдань лабораторної роботи № 7**

| Варіант | Функціональні вимоги |
|---|---|
| 1. Система аутентифікації на основі шифрування випадкових чисел | 1. Пароль, що його вводять при реєстрації користувача в системі, запам'ятовується в базі даних системи та прив'язується до певного ідентифікатора особи – логіна.<br>2. При спробі аутентифікації користувач уводить логін і пароль із клавіатури, користуючись клієнтською частиною програми.<br>3. Клієнтська частина генерує випадкове число, шифрує його одним із симетричних алгоритмів і відправляє логін, випадкове число й результати шифрування серверній частині.<br>4. Серверна частина вибирає з бази даних пароль, що відповідає отриманому логіну, шифрує тим самим алгоритмом отримане випадкове число на паролі з бази даних та порівнює результат з отриманим від клієнтської частини.<br>5. В залежності від результату порівняння зашифрованих чисел система робить висновок про те, допускати користувача до ресурсів системи чи ні. Висновок серверної частини передається клієнтській, яка й повідомляє про це користувача .<br>6. Необхідно передбачити неуспішне завершення аутентифікації в разі триразового введення неправильного пароля.<br>Перевагою такої системи аутентифікації є те, що мережею передається лише випадкове число та результат його зашифрування. Це підвищує стійкість системи аутентифікації по відношенню до перехоплення мережного трафіку.<br>Алгоритм шифрування та довжину ключа, а також спосіб перетворення пароля у ключ шифрування розробіть самостійно. |
| 2. Система аутентифікації на основі хешування парольної інформації | 1. Пароль, що його уводять при реєстрації користувача в системі, хешується певним алгоритмом, до хеш -образу підмішується логін, отриманий образ запам'ятовується в базі даних системи та прив'язується до логіна.<br>2. При спробі аутентифікації користува ч уводить логін і пароль із клавіатури, користуючись клієнтською частиною програми.<br>3. Клієнтська частина обчислює хеш -образ пароля та відправляє логін і хеш -образ серверній частині.<br>4. Серверна частина вибирає з бази даних хеш -образ, що відповідає отриманому логіну, та порівнює його з отриманим образом.<br>5. В залежності від результату порівняння система робить висновок про те, допускати користувача до ресурсів системи чи ні. Висновок серверної частини передається клієнтській, яка й повідомляє про це користувача.<br>6. Необхідно передбачити неуспішне проходження аутентифікації при триразовому неправильному введенні пароля.<br>Перевагою такої системи аутентифікації є те, що мережею передається лише хеш -образ пароля. Це підвищує стійкість системи аутентифікації по відношенню до перехоплення мережного трафіку.<br>Алгоритм хешування оберіть самостійно. |

*Продовження Таблиці 2*

| Варіант | Функціональні вимоги |
|---|---|
| 3. Система шифрування текстових файлів | 1. Створіть програму шифрування текстових файлів або файлів у форматі Microsoft Word за допомогою симетричного блочного криптоалгоритму.<br>2. Програма повинна приймати незашифрований файл на вході та повертати зашифрований файл. Аналогічно вона повинна приймати зашифрований файл та, розшифрувавши його, повертати відкритий текст.<br>3. Генерування ключів та обмін ключовою інформацією розробіть самостійно. |
| 4. Система шифрування мережного трафіку за допомогою потокового симетричного алгоритму | 1. Створіть простий мережний чат, що використовує потокове шифрування мережного трафіку.<br>2. Для шифрування використайте один із потокових симетричних алгоритмів, доступних у CryptoAPI.<br>3. Довжину ключа та алгоритм обміну ключовою інформацією оберіть самостійно.<br>4. Програма повинна передавати введену користувачем із клавіатури інформацію мережею в зашифрованому вигляді.<br>5. Програма повинна розшифровувати отриману мережею інформацію та виводити її на екран для читання користувачем. |
| 5. Система шифрування мережного трафіку за допомогою симетричного блокового алгоритму | 1. Створіть простий мережний чат, що використовує блочне шифрування мережного трафіку.<br>2. Для шифрування використайте один із блочних симетричних алгоритмів, доступних у CryptoAPI.<br>3. Довжину ключа та алгоритм обміну ключовою інформацією оберіть самостійно.<br>4. Програма повинна передавати введену користувачем із клавівтури інформацію мережею в зашифрованому вигляді.<br>5. Програма повинна розшифровувати отриману мережею інформацію та виводити її на екран для читання користувачем. |
| 6. Система шифрування текстових файлів за допомогою асиметричного криптоалгоритму | 1. Створіть програму шифрування текстових файлів або файлів у форматі Microsoft Word за допомогою асиметричного криптоалгоритму.<br>2. Програма повинна приймати незашифрований файл на вході та повертати зашифрований файл. Аналогічно вона повинна приймати зашифрований файл та, розшифрувавши його, повертати відкритий текст.<br>3. Генерування ключів та обмін ключовою інформацією розробіть самостійно. |

*Продовження Таблиці 2*

| Варіант | Функціональні вимоги |
|---|---|
| 7. Система електронного цифрового підпису для текстових файлів | 1. Створіть програму, яка б виконувала процедуру цифрового підпису текстового файла та перевірки цього підпису за допомогою CryptoAPI.<br>2. Програма повинна приймати текстовий файл із відкритою інформацією та виконувати процедуру електронного цифрового підпису цього файла. Цифровий підпис повинен дописуватися в кінці файла.<br>3. Програма повинна приймати текстовий файл із цифровим підписом та перевіряти його істинність.<br>4. Алгоритм електронного цифрового підпису та спосіб обміну ключовою інформацією оберіть самостійно. |
| 8. Система електронного цифрового підпису з хешуванням | 1. Створіть програму, яка б виконувала процедуру електронного цифрового підпису хеш-образу відкритого тексту та перевіряла істинність цього підпису.<br>2. Програма повинна приймати на вході файл із відкритим текстом, створювати хеш-образ і підписувати цей образ за допомогою одного з доступних алгоритмів ЕЦП.<br>3. Програма повинна приймати на вході файл із відкритим текстом, його хеш-образом та електронним цифровим підписом та перевіряти істинність цього підпису.<br>4. Алгоритми хешування та ЕЦП, довжину хеш-образу оберіть самостійно.<br>5. Алгоритм обміну ключовою інформацією також оберіть самостійно. |
| 9. Система аутентифікації на основі хешування парольної інформації (з використанням «токена безпеки») | 1. Пароль, що його вводять при реєстрації користувача в системі, хешується певним алгоритмом, до хеш-образу підмішується логін, отриманий образ запам'ятовується в базі даних системи та прив'язується до логіна.<br>2. Одночасно логін та пароль записуються на дискету або флеш-диск (обов'язково на знімний носій!) у файли з жорстко визначеними іменами.<br>3. При спробі аутентифікації клієнтська частина повинна зчитувати логін і пароль тільки зі знімного носія.<br>4. Система повинна блокувати спроби користувача підмінити читання інформації зі знімного носія читанням із жорсткого диска; не можна також надати користувачеві можливості вибору файлів.<br>5. Клієнтська частина обчислює хеш-образ пароля та відправляє логін і хеш-образ серверній частині.<br>4. Серверна частина вибирає з бази даних хеш-образ, що відповідає отриманому логіну, та порівнює його з отриманим образом.<br>5. Залежно від результату порівняння система робить висновок про те, допускати користувача до ресурсів системи чи ні. Висновок серверної частини передається клієнтській, яка й повідомляє про це користувача.<br>Перевагою такої системи аутентифікації є те, що мережею передається лише хеш-образ пароля. Це підвищує стійкість системи аутентифікації по відношенню до перехоплення мережного трафіку.<br>Алгоритм хешування оберіть самостійно. |

*Продовження Таблиці 2*

| Варіант | Функціональні вимоги |
|---|---|
| 10. Система шифрування текстових файлів за допомогою «токена безпеки» | 1. Створіть програму шифрування текстових файлів або файлів у форматі Microsoft Word за допомогою симетричного блочного криптоалгоритму.<br>2. Програма повинна приймати незашифрований файл на вході та повертати зашифрований файл. Аналогічно вона повинна приймати зашифрований файл та, розшифрувавши його, повертати відкритий текст.<br>3. Ключова інформація повинна зберігатися на знімному носієві (дискета або флешка). При зашифруванні ключ повинен експортуватися на знімний носій (із неможливістю збереження його на жорсткому дискові). При розшифруванні програма повинна зчитувати ключ тільки зі знімного носія. Необхідно заблокувати можливість вибору файлів із жорсткою диска. |

Якщо група налічує більше 10 студентів, кожен із них виконує завдання з цього ж набору, але з використання різних криптографічних алгоритмів. Конкретний алгоритм для кожного студента в такому разі визначає викладач.

***Контрольні запитання та завдання***

1. Які функції виконує CryptoAPI?
2. Розкажіть про архітектуру CryptoAPI.
3. Охарактеризуйте криптографічні алгоритми, які Ви використали для Вашого завдання.
4. Які ще криптографічні алгоритми реалізує CryptoAPI?

# Додатки

## Таблиця Віженера

| | 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 |
| **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 |

**Частотна таблиця літер української мови**

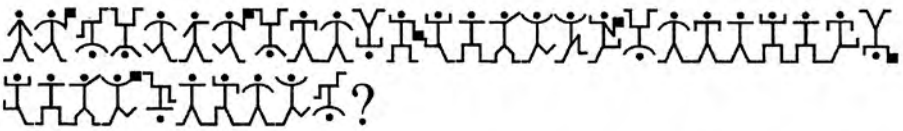
| О | 0,082 | Р | 0,038 | У | 0,028 | Б | 0,010 | Є | 0,006 |
|---|---|---|---|---|---|---|---|---|---|
| Н | 0,070 | І | 0,037 | П | 0,025 | Х | 0,010 | Ф | 0,005 |
| А | 0,070 | С | 0,036 | Я | 0,021 | Ц | 0,009 | Ш | 0,005 |
| И | 0,056 | К | 0,036 | З | 0,019 | Ю | 0,009 | Щ | 0,003 |
| Т | 0,051 | М | 0,033 | Ь | 0,015 | Ж | 0,008 | Ґ | 0,000 |
| В | 0,046 | Д | 0,028 | Г | 0,013 | Й | 0,007 | | |
| Е | 0,043 | Л | 0,028 | Ч | 0,011 | Ї | 0,006 | | |

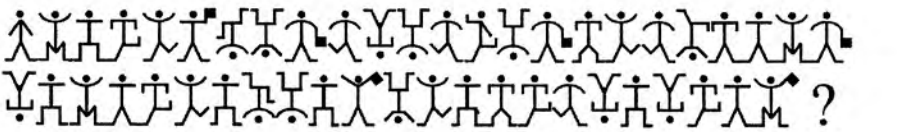
**Частотна таблиця літер англійської мови**

| E | 0,12 | S | 0,06 | P | 0,03 | G | 0,01 | Z | 0,00 |
|---|---|---|---|---|---|---|---|---|---|
| T | 0,09 | R | 0,06 | Y | 0,02 | W | 0,01 | J | 0,00 |
| I | 0,07 | H | 0,04 | U | 0,02 | V | 0,00 | | |
| A | 0,07 | C | 0,04 | M | 0,02 | K | 0,00 | | |
| O | 0,06 | L | 0,04 | F | 0,02 | X | 0,00 | | |
| N | 0,06 | D | 0,04 | B | 0,01 | Q | 0,00 | | |

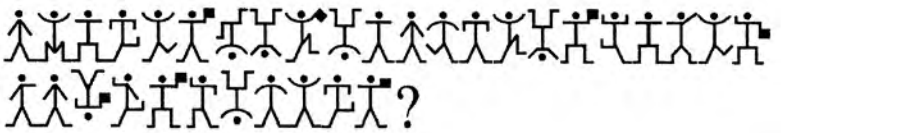
**Частотна таблиця літер російської мови**

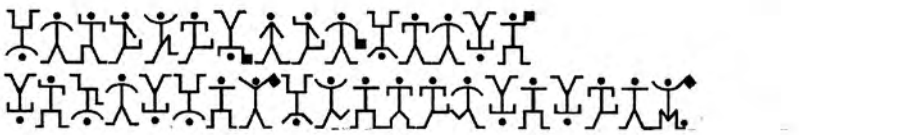
| А | 0,069 | Б | 0,013 | В | 0,038 | Г | 0,014 | Д | 0,024 |
|---|---|---|---|---|---|---|---|---|---|
| Е, Ё | 0,071 | Ж | 0,007 | З | 0,016 | И | 0,064 | Й | 0,010 |
| К | 0,029 | Л | 0,039 | М | 0,027 | Н | 0,057 | О | 0,094 |
| П | 0,026 | Р | 0,042 | С | 0,046 | Т | 0,054 | У | 0,023 |
| Ф | 0,003 | Х | 0,008 | Ц | 0,005 | Ч | 0,012 | Ш | 0,006 |
| Щ | 0,004 | Ъ | 0,001 | Ы | 0,015 | Ь | 0,013 | Э | 0,002 |
| Ю | 0,005 | Я | 0,017 | ПРОБЕЛ | 0,146 | | | | |

1. 

2. 

3. 

4. 

5. 

## Список літератури


1. Кан Д. Взломщики кодов. – М.: Центрполиграф, 2000. – 452 с.
2. Жельников В. Криптография от папируса до компьютера.– М.: ABF, 1994.
3. Баричев С.Г., Серов Р.Е. Основы современной криптографии. М.: Горячая линия-Телеком, 2002. – 152 с.
4. Ємець В., Мельник А., Попович Р. Сучасна криптографія. Основні поняття. – Львів: БаК, 2003. – 144 с.
5. Вербицький О.В. Вступ до криптології. – Львів, 1998. – 247 с.
6. Шеннон К., Теория связи в секретных системах // Шеннон К. Э. Работы по теории информации и кибернетике. – М.: ИЛ, 1963.
7. Винокуров А., Применко Є. Сравнение российского стандарта шифрования, алгоритма ГОСТ 28147-89 и алгоритма Rijndael, выбранного в качестве нового стандарта шифрования США // Системы безопасности. – М.: Гротэк, 2001. – №1,2.
8. Diffie W., Hellman M.E. New directions in cryptography // IEEE Trans. Informat. Theory. – 1976. – V. IT-22. – P. 644–654.
9. Введение в криптографию / Под общ. ред. В.В.Ященко, 2001.
10. Столлинс В. Криптография и защита сетей. – М.: Вильямс, 2004. – 848 с.
11. Гундарь К.Ю., Гундарь А.Ю., Янишевский Д.А. Защита информации в компьютерных системах. – К.: Корнійчук, 2000. – 152 с.
12. Молдовян А, Молдовян Н., Советов Б. Криптография. – СПб.: Лань, 2000. – 224 с.
13. Menezes A. J., Oorshot P. S van. Handbook of applied cryptography. CRC Press. 1997.
14. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные коды на языке С. – М: Триумф, 2002. – 816 с.
15. Молдовян Н.А., Молдовян А.А. Введение в криптосистемы с открытым ключом. – СПб: БХВ-Петербург, 2005. – 288 с.
16. Аветисян Р.Д., Аветисян Д.О. Теоретические основы информатики. – М: РГГУ, 1997.– 167 с.
17. Черемушки А.В. Лекции по арифметическим алгоритмам в криптографии. – М.: МЦНМО, 2002. – 103 с.
18. Фаль О.М. Криптографія: основні ідеї та застосування. – К: Вид-во. НТУУ КПІ, 2004.
19. Ростовцев А. Г., Маховенко Е. Б. Простое усиление схемы цифровой подписи Эль-Гамаля, DSS, ГОСТ Р 34.10–94 // Методы и технические средства обеспечения безопасности информации: Тезисы докладов // Под ред. П. Д. Зегжды. – СПб.: Изд-во СПбГТУ, 2000. – С. 128–130.
20. Деднев М.А., Дыльнов Д.В., Иванов М.А.Защита информации в банковском деле и электронном бизнесе. – М.: Кудиц-образ, 2004. – 512 с.
21. Масленников М. Практическая криптография. – СПб: БХВ, 2003. – 464 с.
22. Галицкий А.В., Рябко С.Д., Шаньгин В.Ф. Защита информации в сети. – М.:ДМК, 2004. – 616 с.
23. Шеховцов В.А. Операційні системи. – К.: BHV, 2005. – 400 с.
24. Щербаков А, Домашев А. Прикладная криптография. Использование и синтез криптографических интерфейсов. – М.: Русская редакция, 2003.
25. Хорошко В.А., Чекатков А.А. Методы и средства защиты информации. – К.: Юниор, 2003. – 501 с.

*Навчальне видання*

**Остапов Сергій Едуардович,**
**Валь Лідія Олександрівна**

# Основи криптографії

Навчальний посібник

Підписано до друку 12.08.2008 р.
Формат 70x100 $^{1}/_{16}$. Папір офсетний. Друк офсетний
Умов. друк. арк. 15,28. Обл.-вид. арк. 17,64.
Замовлення № *317* . Наклад 300 прим.

---

Видавництво "Книги – XXI"
Україна, 59000, м. Сторожинець Чернівецької обл.
вул. О. Кобилянської, 7
Тел./факс: (0372)586021, 586464
e-mail: booksxxi@gmail.com
www.books-xxi.com.ua
*Свідоцтво про державну реєстрацію*
*ДК № 1839 від 10.06.2004 р.*
Друк ТОВ «Друк Арт»
58018, м. Чернівці, вул. Головна, 198а, к.5, тел. (0372) 585-432


ISBN 978-966-2147-35-3


*Видавництво «Книги – XXI»*