В. А. Дем'яненко, Ю. А. Кузнецова

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

В. А. Дем'яненко, Ю. А. Кузнецова

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

Навчальний посібник до лабораторного практикуму

Харків «ХАІ» 2021

УДК 004.056.5
Д32

Рецензенти: канд. техн. наук, доц. О. В. Сєвєрінов,
канд. техн. наук В. М. Ткачов

Дем'яненко, В. А.

Д32 Безпека програм та даних [Електронний ресурс] : навч. посіб. до лаб. практикуму / В. А. Дем'яненко, Ю. А. Кузнецова. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2021. – 95 с.

Розглянуто як загальні, так і окремі питання криптографії (шифрування інформації): проблеми захисту інформації в комп'ютерних системах, види криптосистем, шифрів та інші питання, у тому числі й не пов'язані безпосередньо із засекречуванням даних. Викладено основи застосування криптографії в практичній інформатиці з урахуванням сучасних досягнень у криптології.

Для студентів усіх форм навчання за спеціальностями напряму «Комп'ютерні науки», що виконують лабораторні роботи за темою «Криптографічний захист інформації», а також програмістів, робота яких пов'язана з шифруванням інформації і які займаються розробленням систем з обмеженим доступом.

Іл. 19. Табл. 25. Бібліогр.: 12 назв

УДК 004.056.5

ВСТУП

Криптографія історично зародилася з потреби передачі секретної інформації. Тривалий час вона була пов'язана лише з розробленням спеціальних методів перетворення інформації з метою її подання у формі, недоступній для потенційного зловмисника. Згодом розширилася сфера застосування електронних способів передачі та оброблення даних, і комп'ютерні інформаційні технології почали масово використовуватися. Проблема криптографії потребує вирішення численних завдань, не пов'язаних безпосередньо із засекречуванням інформації. Сучасні проблеми криптографії полягають у складності розроблення систем електронного цифрового підпису та таємного електронного голосування, протоколів електронного жеребкування й ідентифікації віддалених користувачів, методів захисту від нав'язування помилкових повідомлень і т. ін. Багато завдань практичної інформатики можуть бути вирішені тільки на основі використання криптографічних методів. У криптографії розглядається деякий зловмисник (опонент, криптоаналітик противника, порушник, нелегальний користувач), який обізнаний про використовувані криптографічні алгоритми, протоколи, методи і намагається скомпрометувати їх. Компрометація криптосистеми може полягати, наприклад, у несанкціонованому читанні інформації, формуванні чужого підпису, зміні результатів голосування, порушенні таємниці голосування, модифікації даних, що важко виявити. Такі дії опонента називаються криптоаналітичною атакою. Специфіка криптографії полягає в тому, що вона спрямована на розроблення методів, що забезпечують стійкість до будь-яких дій зловмисника, тоді як на момент розроблення криптосистеми неможливо передбачити способи атаки, які можуть бути винайдені в майбутньому на основі нових досягнень теорії і технологічного прогресу. Питання про те, наскільки надійно вирішується та чи інша криптографічна проблема, наприклад, оцінювання трудомісткості атаки на криптосистему, є зазвичай дуже складним і самостійним предметом, що називається криптоаналізом. Криптографія і криптоаналіз становлять єдину галузь знань – криптологію, яка нині є областю сучасної математики, що має важливі додатки в сучасних інформаційних технологіях.

Лабораторна робота № 1
ШИФРИ ПЕРЕСТАНОВКИ

Мета роботи: вивчення прийомів шифрування методом перестановки.

Теоретичні відомості

Шифр, перетворення з якого змінюють тільки порядок проходження символів вхідного тексту, але не змінюють їх самих, називається шифром перестановки (ШП).

Розглянемо перетворення з ШП, призначене для шифрування повідомлення довжиною ***n*** символів. Його можна подати у вигляді таблиці перестановки (табл. 1.1).

Таблиця 1.1

1	2	...	N
i1	I2	...	In

Тут ***i1*** – номер місця шифротексту, на яке потрапляє перша літера вхідного повідомлення при вибраному перетворенні; ***i2*** – номер місця для другої літери і т. д. У верхньому рядку таблиці виписані по порядку числа від ***1*** до ***n***, а в нижньому – ті самі числа, але в довільному порядку. Така таблиця називається підстановкою ступеня ***n***.

Знаючи підстановку, що задає перетворення, текст можна як зашифрувати, так і розшифрувати. Наприклад, якщо для перетворення використовується підстановка (табл. 1.2) і відповідно до неї зашифровано слово ***МОСКВА***, то вийде ***КОСВМА***.

Таблиця 1.2

1	2	3	4	5	6
5	2	3	4	1	6

Спробуйте розшифрувати повідомлення ***НЧЕИУК***, отримане внаслідок перетворення за допомогою зазначеної підстановки.

Як завдання лабораторної роботи пропонується самостійно виписати підстановки, що задають перетворення в описаних нижче трьох прикладах шифрів перестановки. Відповіді поміщені в кінці розділу.

Ознайомившись з методом математичної індукції, можна легко переконатися в тому, що існує ***1×2×3×...×n (n!)*** варіантів заповнення нижнього рядка таблиці (6). Таким чином, кількість різних перетворень шифру перестановки, призначеного для шифрування повідомлення довжиною ***n***, менше або дорівнює ***n!*** (Зауважимо, що до цього числа входить і варіант перетворення, що залишає всі символи на своїх місцях!).

Зі збільшенням числа ***n*** значення ***n!*** збільшується дуже швидко. Наведемо таблицю значень ***n!*** для перших ***10*** натуральних чисел (табл. 1.3).

Таблиця 1.3

N	1	2	3	4	5	6	7	8	9	10
n!	1	2	6	24	120	720	5040	40320	362880	3628800

При великих ***n*** для наближеного обчислення ***n!*** можна користуватися відомою формулою Стірлінга

$$n! \approx \sqrt{2\pi n}(n/e)^n ,$$

де ***e = 2.717281828*** ...

Прикладом ШП, призначеного для шифрування повідомлень довжиною ***n***, є шифр, у якому як множина ключів взято множину усіх перестановок ступеня ***n***, а відповідні їм перетворення шифру задаються так, як було описано вище. Число ключів такого шифру – одне ***n!***

Для використання на практиці такий шифр незручний, оскільки при великих значеннях ***n*** доводиться працювати з довгими таблицями.

Широкого застосування набули шифри перестановки, які використовують певну геометричну фігуру. Перетворення з цього шифру полягають у тому, що в фігуру вхідний текст вписується по ходу одного «маршруту», а потім по ходу другого виписується з неї. Такий шифр називають маршрутною перестановкою. Наприклад, можна вписувати вхідне повідомлення в прямокутну таблицю, вибравши такий маршрут: по горизонталі, починаючи з лівого верхнього кута, по черзі зліва направо і справа наліво. Виписувати ж повідомлення будемо по іншому маршруту: по вертикалі, починаючи з верхнього правого кута і рухаючись по черзі зверху вниз і знизу вгору.

Зашифруємо, наприклад, зазначеним способом фразу «***ПРИМЕРМАРШРУТНОЙПЕРЕСТАНОВКИ***».

Використаємо прямокутник розміром 4 х 7 (рис. 1.1).

П	Р	И	М	Е	Р	М
Н	Т	У	Р	Ш	Р	А
О	Й	П	Е	Р	Е	С
И	К	В	О	Н	А	Т

Рис. 1.1

Зашифрована фраза має такий вигляд:

МАСТАЕРРЕШРНОЕРМИУПВКЙТРПНОИ

Теоретично маршрути можуть бути значно більш витонченими, однак заплутаність маршрутів ускладнює використання таких шифрів.

Далі наводяться описи трьох різновидів шифрів перестановки.

Шифр «Сцитала»

Одним з найперших шифрувальних пристроїв був жезл («Сцитала»), що застосовувався ще за часів війни Спарти проти Афін у V столітті до н. е. Це був циліндр, на який виток до витка намотувалася вузька папірусна стрічка (без просвітів і нахлистів), а потім на цій стрічці уздовж її осі записувався необхідний для передачі текст. Стрічка змотувалася з циліндра і відправлялася адресату, який, маючи циліндр такого самого діаметра, намотував стрічку на нього і прочитував повідомлення. Ясно, що такий спосіб шифрування основувався на перестановці місцями літер повідомлення.

Шифр «Сцитала» реалізує не більше n перестановок (n, як і раніше, – довжина повідомлення). Цей шифр, як неважко побачити, еквівалентний такому шифру маршрутної перестановки: в таблицю, що складається з n стовпців, через підрядник записують повідомлення, після чого виписують літери в стовпці. Кількість задіяних стовпців у таблиці не може перевищувати довжини повідомлення (рис. 1.2).

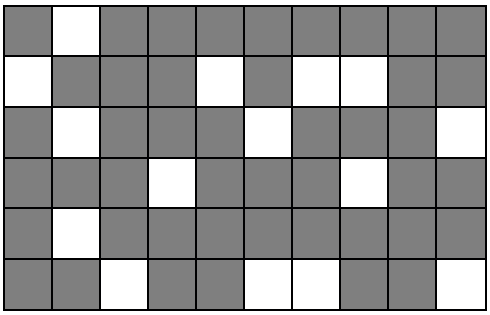


Рис. 1.2

Є ще суто фізичні обмеження, що накладаються реалізацією шифру «Сцитала». Природно припустити, що діаметр жезла не повинен перевищувати 10 см. При висоті рядка в 1 см на одному витку такого жезла вміститься максимум 32 літери ($10\pi < 32$). Таким чином, кількість перестановок, що реалізуються «Сциталой», навряд чи перевищує 32.

Шифр «Поворотна решітка»

Для використання шифру «Поворотна решітка», виготовляється трафарет з прямокутного аркуша в клітинку паперу розміром ***2m × 2k*** клітин. У трафареті вирізано ***mk*** клітин так, щоб при накладенні його на чистий аркуш паперу такого самого розміру чотирма можливими способами його вирізи повністю покривали всю площу аркуша.

Літери з повідомлення послідовно вписуються у вирізи трафарету (за рядками, в кожному рядку зліва направо) під час кожного з чотирьох його можливих положень у заздалегідь установленому порядку.

Пояснимо процес шифрування на прикладі. Нехай як ключ використовується решітка розміром 6 х 10, як показано на рис. 1.2.

Зашифруємо з її допомогою такий текст:

ШИФРРЕШЕТКАЯВЛЯЕТСЯЧАСТНЫМСЛУЧАЕМШИФРАМАРШРУТНОЙПЕРЕСТАНОВКИ

Наклавши решітку на аркуш паперу, вписуємо перші 15 (за кількістю вирізів) літер повідомлення: **ШИФРРЕШЕТКАЯВЛЯ**. Знявши решітку, побачимо текст, показаний на рис. 1.3. Повертаємо решітку на ***180°***. У віконцях з'являться нові, ще незаповнені клітини. Вписуємо в них наступні 15 літер. Отримаємо запис, зображений на рис. 1.4. Потім перевертаємо решітку в інший бік і зашифровуємо залишок тексту аналогічним чином (рис. 1.5 і 1.6).

	Ш								
И				Ф		Р	Р		
	Е				Ш				Е
			Т				К		
	А								
		Я			В	Л			Я

Рис. 1.3

Е	Ш		Т	С			Я		
И				Ф		Р	Р	Ч	
	Е	А			Ш	С			Е
Т			Т	Н			К	Ы	
	А	М	С		Л				У
		Я			В	Л		Ч	Я

Рис. 1.4

Е	Ш	А	Т	С	Е	М	Я		Ш
И	И			Ф		Р	Р	Ч	
	Е	А	Ф		Ш	С	Р		Е
Т	А		Т	Н	М		К	Ы	А
Р	А	М	С	Ш	Л	Р	У		У
	Т	Я			В	Л		Ч	Я

Рис. 1.5

Е	Ш	А	Т	С	Е	М	Я	Н	Ш
И	И	О	Й	Ф	П	Р	Р	Ч	Е
Р	Е	А	Ф	Е	Ш	С	Р	С	Е
Т	А	Т	Т	Н	М	А	К	Ы	А
Р	А	М	С	Ш	Л	Р	У	Н	У
О	Т	Я	В	К	В	Л	И	Ч	Я

Рис. 1.6

Одержувач повідомлення, який має таку саму решітку, легко прочитає вхідний текст, наклавши решітку на шифротекст по порядку чотирма способами.

Можна довести, що кількість можливих трафаретів, тобто кількість ключів шифру «решітка», становить ***T = 4mk***. Цей шифр призначений для повідомлень довжиною ***n = 4mk***. Кількість усіх перестановок у тексті такої довжини становитиме ***(4mk)!***, що набагато більше числа ***T***. Однак вже при розмірі трафарету ***8×8*** кількість можливих решіток перевищує чотири мільярди.

Шифр вертикальної перестановки

Поширений різновид шифру маршрутної перестановки – «шифр вертикальної перестановки» (ШВП). У ньому знову використовується прямокутник, в який повідомлення вписується звичайним способом (в рядки зліва направо). Виписуються літери по вертикалі, а стовпці при цьому беруться в порядку, визначеному ключем. Нехай, наприклад, цей ключ матиме вигляд (5, 4, 1, 7, 2, 6, 3) і з його допомогою треба зашифрувати повідомлення

ВОТПРИМЕРШИФРАВЕРТИКАЛЬНОЙПЕРЕСТАНОВКИ

Впишемо повідомлення в прямокутник, стовпчики якого пронумеровані відповідно до ключа (рис. 1.7).

5	**1**	**4**	**7**	**2**	**6**	**3**
В	О	Т	П	Р	И	М
Е	Р	Ш	И	Ф	Р	А
В	Е	Р	Т	И	К	А
Л	Ь	Н	О	Й	П	Е
Р	Е	С	Т	А	Н	О
В	К	И	-	-	-	-

Рис. 1.7

Вибираючи стовпчики в порядку, заданому ключем, і виписуючи послідовно літери кожного з них зверху вниз, отримуємо таку криптограму:

ОРЕЬЕКРФИЙА-МААЕО-ТШРНСИВЕВЛРВИРКПН-ПИТОТ-

Кількість ключів ШВП не більше ***m!***, де ***m*** – кількість стовпців таблиці. Зазвичай ***m*** набагато менше, ніж довжина тексту ***n*** (повідомлення вкладається в кілька рядків по ***m*** літер), а отже, і ***m!*** набагато менше ***n!***

Якщо ключ ШВП не рекомендується записувати, його можна витягати з певного слова, яке легко запам'ятовується, або пропозиції. Для цього існує

багато способів. Найбільш поширений полягає в тому, щоб приписувати літерам числа за звичайним алфавітним порядком літер. Наприклад, нехай ключовим словом буде ***ПЕРЕСТАНОВКА***. Присутня в ньому літера ***А*** отримує номер ***1***. Якщо якась літера повторюється кілька разів, то її появи нумеруються послідовно зліва направо. Тому друга поява літери ***А*** отримує номер ***2***. Оскільки літери ***Б*** в цьому слові немає, то літера ***В*** отримує номер ***3*** і так далі. Процес триває до того часу, поки всі літери не отримають номери. Отже, ми отримуємо такий ключ:

П	Е	Р	Е	С	Т	А	Н	О	В	К	А
9	4	10	5	11	12	1	7	8	3	6	2

Для забезпечення додаткової прихованості можна повторно зашифрувати вже зашифроване повідомлення. Такий метод шифрування називається подвійною перестановкою. У випадку подвійної перестановки стовпців і рядків таблиці перестановки визначаються окремо для стовпців і для рядків. Спочатку в таблицю записується текст повідомлення, потім по черзі переставляються стовпці, а потім – рядки. Під час розшифрування порядок перестановок повинен бути зворотним.

Приклад шифрування методом подвійної перестановки показано на рис. 1.8 і 1.9. Якщо зчитувати шифротекст з правої таблиці за рядками, то отримаємо таке: ***ТЮАЕООГМРЛИПОЬСВ***.

Ключем до шифру подвійної перестановки буде послідовність номерів стовпців і номерів рядків вхідної таблиці (у цьому прикладі послідовності ***4, 1, 3, 2*** і ***3, 1, 4, 2*** відповідно).

Кількість варіантів подвійної перестановки швидко зростає при збільшенні розміру таблиці:

- для таблиці 3 х 3 – 36 варіантів;
- для таблиці 4 х 4 – 576 варіантів;
- для таблиці 5 х 5 – 14 400 варіантів.

Однак подвійна перестановка не відрізняється високою стійкістю і порівняно просто зламується при будь-якому розмірі таблиці шифрування.

	4	1	3	2
3	П	Р	И	Л
1	Е	Т	А	Ю
4	В	О	С	Ь
2	М	О	Г	О

Вхідна таблиця

	1	2	3	4
3	Р	Л	И	П
1	Т	Ю	А	Е
4	О	Ь	С	В
2	О	О	Г	М

Перестановка стовпців

	1	2	3	4
1	Т	Ю	А	Е
2	О	О	Г	М
3	Р	Л	И	П
4	О	Ь	С	П

Перестановка рядків

Рис. 1.8

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

О	И	Р	М
Е	О	С	Ю
В	Т	А	Ь
Л	Г	О	П

Рис. 1.9

Застосування магічних квадратів

У середні віки для шифрування перестановкою також застосовувалися магічні квадрати.

Магічними квадратами називають квадратні таблиці з вписаними в їх клітини послідовними натуральними числами, починаючи від ***1***, які дають у сумі по кожному стовпцю, кожному рядку і кожній діагоналі те саме число.

Шифрований текст вписували в магічні квадрати відповідно до нумерації їх клітин. Якщо потім виписати вміст такої таблиці за рядками, то вийде шифротекст, сформований завдяки перестановці літер вхідного повідомлення. У ті часи вважалося, що створені за допомогою магічних квадратів шифротексти охороняє не тільки ключ, але й магічна сила.

Приклад магічного квадрата і його заповнення повідомленням ***ПРИЛЕТАЮ ВОСЬМОГО*** показано на рис. 1.8.

Шифротекст, одержуваний при зчитуванні вмісту правої таблиці за рядками, має цілком загадковий вигляд: ***ОИРМЕОСЮВТАЬЛГОП***.

Кількість магічних квадратів швидко зростає зі збільшенням розміру квадрата. Існує тільки один магічний квадрат розміром ***3 x 3*** (якщо не враховувати його повороти). Кількість магічних квадратів ***4 x 4*** становить вже 880, а кількість магічних квадратів ***5 x 5*** – близько 250 000.

Магічні квадрати середніх і великих розмірів були хорошою базою для забезпечення потреб шифрування того часу, оскільки практично нереально вручну перебрати всі варіанти для такого шифру.

Перестановка бітів

Метод *перестановки* бітів у символі зумовлений використанням комп'ютерів у процесі шифрування.

Нехай повідомлення складається з символів, кожен з яких у машинному поданні є послідовністю з восьми бітів. Переставивши деяким чином біти в кожному символі, отримаємо зашифровану послідовність (рис. 1.10).

Щоб розшифрувати її, необхідно лише провести зворотну перестановку.

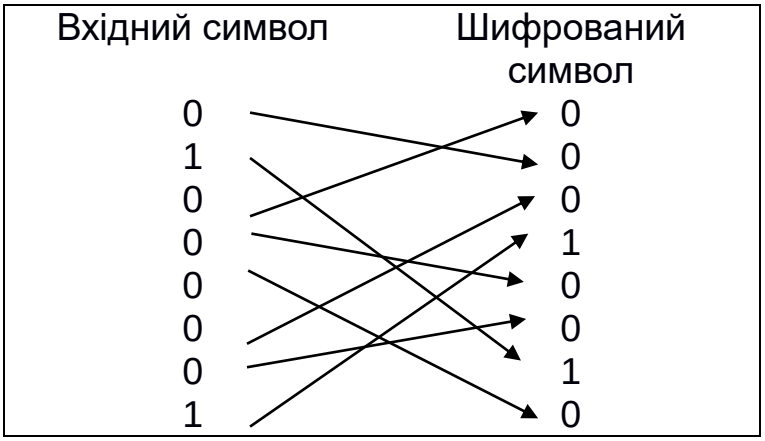


Рис. 1.10

Злом шифру перестановки

Розглянемо методи злому шифру перестановки. Проблема, що виникає при відновленні повідомлення, зашифрованого ШП, полягає не тільки в тому, що кількість можливих ключів велика навіть за невеликої довжини тексту. Якщо і вдасться перебрати всі допустимі варіанти перестановок, то буде не завжди ясно, який з цих варіантів істинний. Наприклад, нехай потрібно відновити вхідний текст за криптограмою ***АОГР***, і нам нічого не відомо, крім того, що застосовувався шифр перестановки. Який варіант «осмисленого» вхідного тексту визнати істинним: ***ГОРА*** або ***РОГА***? А може, ***АРГО***? Наведемо приклад ще більш заплутаної ситуації. Нехай потрібно відновити повідомлення за криптограмою ***ААНИНК-ТЕОМЛ,З.ЬЬЗИВТЛП-ЬЯО***, отриманою шифром перестановки.

Можливі, як мінімум, два варіанти вхідного повідомлення:

КАЗНИТЬ,- НЕЛЬЗЯ- ПОМИЛОВАТЬ.
КАЗНИТЬ- НЕЛЬЗЯ,- ПОМИЛОВАТЬ.

Ці варіанти мають прямо протилежний зміст, і за наявних умов у нас немає можливості визначити, який з варіантів істинний.

Іноді завдяки особливостям реалізації шифру вдається отримати інформацію про використане перетворення перестановки.

Розглянемо шифр «Сцитала». Вище вже згадувалося питання про кількість перестановок, що реалізуються «Сциталою». Їх виявилося не більше 32. Це число невелике, тому можна перебрати всі варіанти. При достатній довжині повідомлення отримаємо, найімовірніше, єдиний читабельний варіант тексту.

Розглянемо приклад з ШВП. За умовою пробіл між словами під час запису тексту в таблицю опускався, тому робимо висновок, що всі стовпці, які містять пробіл в останньому рядку, повинні стояти в кінці тексту. Таким чином, стовпці розбиваються на дві групи (що містять 6 і 5 літер). Для завершення відновлення початкового тексту досить знайти порядок проходження стовпців у кожній з груп окремо, що набагато простіше.

Аналогічна ситуація виникає і при неповному використанні шифру «решітка».

Нехай є решітка розміром ***m x r*** і зашифроване з її допомогою повідомлення довжиною ***mr – k***, що не містить пробілів. Незаповнені ***k*** місць у решітці за умови, що $mr/4 \geq k$, відповідають вирізам у четвертому положенні решітки. На основі такої інформації відбувається різке зменшення кількості допустимих решіток (їх буде $4^{mr/(4-k)}$).

Ще один підхід до розкриття шифрів вертикальної перестановки – лінгвістичний. Він оснований на тому, що в природних мовах деякі комбінації літер зустрічаються дуже часто, інші – набагато рідше, а деякі взагалі не зустрічаються (наприклад, «*ыьъ*»).

Будемо підбирати порядок проходження стовпців один за одним так, щоб у всіх рядках цих стовпців виходили «читабельні» відрізки тексту.

Поєднання лінгвістичного методу з урахуванням додаткової інформації досить швидко допоможе розкрити повідомлення.

Приклад виконання завдання

Зашифрувати вхідну послідовність символів методом перестановки, використовуючи підстановку довжиною чотири символи. Використовувати файли як джерело вхідної і приймач шифрованої інформації.

```
program FileCoding;
const
 Brk = 4; {Довжина підстановки}
type
 Range = 1..Brk;
const
 stend: array [Range] of Range = (3,1,4,2); {Підстановка}
```

```
 {При декодуванні можна використовувати ідентичну}
 {Процедуру декодування, але з підстановкою (2,4,1,3)}
var
 FIn: file; {Вхідний файл}
 FOut: file of byte; {Вхідний файл}
 Tmp: array [Range] of byte; {Змінна для зберігання прочитаних байтів}
 RealCnt: integer; {Кількість реально прочитаних байтів}
begin {program}
 assign (FIn, 'input.txt');
 assign (FOut, 'output.txt');
 reset (FIn, 1);
 rewrite (FOut);
 while not (eof (FIn)) do
 begin
 BlockRead (FIn, tmp, Brk, RealCnt);
 for i: = RealCnt + 1 to Brk do {Якщо прочитано менше Brk байтів,}
 Tmp [i]: = 0; {То доповнити нулями}
 for i: = 1 to Brk do
 write (FOut, Tmp [stend [i]]);
 end;
 close (FIn);
 close (FOut);
end. {Program}
```

Контрольні запитання

1. У чому полягає метод шифрування перестановкою?
2. Що таке маршрутна перестановка?
3. Який «маршрут» можна використовувати для реалізації шифру «Сцитала»?
4. Що називається «поворотною решіткою»?
5. Оцініть кількість ключів шифру вертикальної перестановки. У скільки разів ця кількість ключів збільшиться при використанні подвійної перестановки?
6. Наведіть приклад використання магічного квадрата для шифрування повідомлення «***ЯУЕЗЖАЮВНОВГОРОД***».
7. Що таке шифрування перестановкою бітів?
8. Запропонуйте шлях розкриття шифру перестановки. Які труднощі виникають при цьому і які «помилки» шифрувальників можна використовувати?

Варіанти завдань

Для непарних варіантів (1, 3, ..., 13) пропонується реалізувати процедуру шифрування, для парних (2, 4, ..., 14) – дешифрування з використанням зазначених методів. Передбачте вибір ключа шифрування.

1, 2. Вхідну послідовність розбийте на групи по чотири символи, далі в кожній групі символи переставте з використанням підстановки, яку ви виберете самостійно.

3, 4. Реалізуйте маршрутну перестановку з використанням шифрувальної таблиці 5 х 8. Маршрут виберіть самостійно.

5, 6. Реалізуйте процедуру, що моделює використання «Сцитали». Кількість стовпців шифрувальної таблиці виберіть самостійно.

7, 8. Виберіть поворотну решітку і змоделюйте її використання.

9, 10. Реалізуйте шифрування подвійною перестановкою. Розмірність шифрувальної таблиці виберіть самостійно.

11, 12. Змоделюйте використання магічних квадратів. Передбачте їх генерацію.

13, 14. Реалізуйте метод перестановки бітів.

Визначте ключ (варіанти ключів при неоднозначності тлумачення), використаний для шифрування у варіанті 1. Вхідні дані: шифротекст, довжина груп розбиття (4 символи).

Визначте ключ, використаний при шифруванні у варіанті 9.

Вхідні дані: розмірність шифрувальної таблиці – 3 х 3.

Лабораторна робота № 2
ШИФРИ ЗАМІНИ

Мета роботи: вивчення алгоритмів шифрування і розшифрування повідомлень за допомогою шифрів заміни.

Теоретичні відомості

Шифрами заміни називають такі шифри, в яких використовують заміну кожного символу відкритого повідомлення на інші символи – шифропозначення, причому порядок проходження шифропозначень збігається з порядком проходження відповідних їм символів відкритого повідомлення.

Нехай, наприклад, шифрується повідомлення російською мовою і заміні підлягає кожна літера повідомлення. Формально в цьому випадку шифр заміни можна описати таким чином. Для кожної літери α вхідного алфавіту будується деяка множина символів $M\alpha$, що називається множиною шифропозначень для літери α.

Табл. 2.1 є ключем шифру заміни. Знаючи її, можна здійснити як шифрування, так і розшифрування.

Під час шифрування кожна літера α відкритого повідомлення, починаючи з першої, замінюється будь-яким символом з безлічі $M\alpha$. Завдяки цьому можна отримати різні варіанти зашифрованого повідомлення для того самого відкритого повідомлення.

У найпростішому випадку множина шифропобозначень $M\alpha$ складається з одного елемента. Такий шифр називається шифром простої заміни.

Таблиця 2.1

А	Б	…	Я
$М_А$	$М_Б$	…	$М_Я$

Шифри простої заміни

Система шифрування Цезаря

Як ключ шифру Цезаря використовують табл. 2.2, перший рядок якої містить літери вхідного алфавіту, а другий – послідовність літер, записаних в алфавітному порядку, але починається не з літери ***А***, а з будь-якої іншої.

Таблиця 2.2

А	Б	...	Е	Ю	Я
Д	Е	...	Б	В	Г

Під час шифрування літеру вхідного повідомлення знаходять у першому рядку і замінюють її на відповідну літеру другого рядка. Запам'ятати ключ досить просто – треба лише запам'ятати першу літеру другого рядка.

Серйозний недолік цього шифру – обмежена кількість ключів, яка дорівнює кількості літер в алфавіті.

Афінна система підстановок Цезаря

Тут літери вхідного повідомлення перетворюються таким чином:

$$T_1 = AT + B \pmod m,$$

де ***T*** – порядковий номер літери вхідної послідовності;

T_1 – порядковий номер відповідної літери зашифрованої послідовності;

m – розмір алфавіту;

A, B – цілі числа, причому ***A*** та ***m*** – взаємно прості.

Приклад

Зашифруємо фразу ***КОРАБЛИ ОТПЛЫВАЮТ ВЕЧЕРОМ***, використовуючи афінну систему підстановок при ***A = 13***, ***B = 5***. Розмір алфавіту ***m = 32*** (будемо вважати, що в початковому алфавіті замість літери ***Й*** використовується ***И***, а замість ***Ё*** – ***Е***, та додамо 32-м символом пробіл). Унаслідок перетворень отримаємо:

Повідомлення ***КОРАБЛИ ОТПЛЫВАЮТ ВЕЧЕРОМ***.

Шифротекст ***ЫПИЕУЗО ЩПВЬЗШЕЯВ ЩЖГЖИПХ***.

Гасловий шифр

У цьому шифрі запам'ятовування ключа основане на гаслі – слові або фразі, що легко запам'ятовується. Наприклад, виберемо слово-гасло «***заявление***» і заповнимо другий рядок таблиці за таким правилом: спочатку вписується слово-гасло, причому повторювані літери відкидаються, потім ця таблиця доповнюється літерами алфавіту, що не ввійшли до неї. Ключ матиме такий вигляд:

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

Полібіанський квадрат

Шифр винайшов грецький письменник та історик Полібій. Прямокутна таблиця заповнюється літерами алфавіту у випадковому порядку. Кожна літера відкритого повідомлення замінюється літерою, розташованою

нижче у тому самому стовпці. Якщо літера знаходиться в останньому рядку табл. 2.3, то для її шифрування беруть верхню літеру стовпця. Наприклад, слово «***АЛФАВИТ***» після шифрування перетвориться на «***УЪУТСЗ***».

Таблиця 2.3

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

Шифрувальна таблиця Трисемуса

Для отримання такого шифру використовується таблиця для запису літер і ключове слово (або фраза). У таблицю спочатку вписується ключове слово, причому повторювані літери відкидаються. Потім ця таблиця доповнюється літерами, які не ввійшли до алфавіту. На рис. 2.1 зображено таблицю Трисемуса з ключовим словом «***БАНДЕРОЛЬ***». Застосування таблиці аналогічне застосуванню полібіанського квадрата.

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

Рис. 2.1

Перевагою цього та інших розглянутих шифрів простої заміни є простота реалізації, недоліком – легка розкриваність з огляду на те, що такі шифри зберігають інформацію про частоту літер вхідного тексту. Це дозволяє застосовувати методи підрахунку частот для розшифрування повідомлень. З метою усунення цього недоліку застосовують біграмний шифр Плейфейра і систему омофонів.

Біграмний шифр Плейфейра

Цей шифр оснований на таблиці, аналогічній таблиці Трисемуса.

Процедура шифрування складається з таких кроків:

1. Відкритий текст розбивається на пари літер (біграми). Текст повинен мати парну кількість літер, і в ньому не має бути біграм, що містять дві однакові літери.

2. Послідовність біграм відкритого тексту перетворюється на послідовність біграм за допомогою шифрувальної таблиці за такими правилами:

1) якщо обидві літери біграми відкритого повідомлення не потрапляють в один рядок або стовпець, то для заміни знаходять літери в кутах прямокутника, що визначається цією парою літер;

2) якщо обидві літери біграми відкритого повідомлення належать одному стовпцю таблиці, то їх замінюють на літери, які знаходяться під ними; якщо при цьому літера відкритого тексту знаходиться в нижньому рядку, то для шифрування береться літера з верхнього рядка того самого стовпчика;

3) якщо обидві літери біграми відкритого повідомлення належать одному рядку таблиці, то вони замінюються на літери, які знаходяться праворуч від них; якщо при цьому літера відкритого тексту знаходиться в крайньому правому стовпці, то для шифрування береться літера з крайнього лівого стовпця того самого рядка.

Система омофонів

Цей шифр характеризується тим, що літери вхідного повідомлення мають кілька замін. Кількість замін береться пропорційною ймовірності появи літери у відкритому тексті. Дані про розподіли ймовірностей літер у російському тексті наведено в табл. 2.4.

Таблиця 2.4

Літера	Ймовірність	Літера	Ймовірність	Літера	Ймовірність	Літера	Ймовірність
Пропуск	0.175	Р	0.040	Я	0.018	Х	0.009
О	0.090	В	0.038	Ы	0.016	Ж	0.007
Е	0.072	Л	0.035	З	0.016	Ю	0.006
А	0.062	К	0.028	Ъ	0.014	Ш	0.006
И	0.062	М	0.026	Б	0.014	Ц	0.004
Н	0.053	Д	0.025	Г	0.013	Щ	0.003
Т	0.053	П	0.023	Ч	0.012	Э	0.003
С	0.045	У	0.021	Й	0.010	Ф	0.002

Шифруючи літеру вхідного повідомлення, вибирають випадковим чином одну з її замін. Заміни (що часто називаються омофонами) можуть бути подані трирозрядними числами від ***000*** до ***999***. Наприклад, літері ***О*** присвоюють ***90*** випадкових номерів, літерам ***Б*** та ***Ъ*** – по 14 номерів. Якщо омофони присвоюються випадковим чином різним появам тієї самої літери, то кожен омофон з'являється в шифротексті рівноймовірно.

Система омофонів забезпечує простий захист від криптоаналітичних атак, основаних на підрахунку частот появи літер у шифротексті.

Шифри складної заміни

Шифри складної заміни називають багатоалфавітними, тому що для шифрування кожного символа вхідного повідомлення застосовують свій шифр простої заміни.

Багатоалфавітна підстановка послідовно й циклічно змінює використовувані алфавіти.

При ***r***-алфавітній підстановці символ x_0 вхідного повідомлення замінюється символом y_0 з алфавіту B_0, символ ***x1*** – символом ***y1*** з алфавіту B_1 і т. д., символ x_{r-1} – символом y_{r-1} з алфавіту B_{r-1}, символ x_r – символом y_r знову з алфавіту B_0 і т. д.

У табл. 2.5 наведено загальну схему багатоалфавітної підстановки для випадку $r = 4$.

Таблиця 2.5

Вхідний символ	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9
Алфавіт підстановки	B0	B1	B2	B3	B0	B1	B2	B3	B0	B1

Ефект використання багатоалфавітної підстановки полягає в тому, що забезпечується маскування природної статистики вхідної мови, оскільки конкретний символ з вхідного алфавіту може бути перетворений на кілька різних символів шифрувальних алфавітів B_i.

Шифр Гронсфельда

Шифрування здійснюється таким чином. Під літерами вхідного повідомлення записують цифри числового ключа. Якщо ключ – коротший за повідомлення, то його запис циклічно повторюють. Для заміни вибирають ту літеру, яка зміщена за алфавітом на відповідну цифру ключа. Наприклад, застосовуючи як ключ групу з чотирьох початкових цифр числа ***e*** (основи натуральних логарифмів), а саме 2718, отримуємо для вхідного повідомлення ***ВОСТОЧНЫЙ ЭКСПРЕСС*** такий шифротекст:

Повідомлення ***ВОСТОЧНЫЙ ЭКСПРЕСС***

Ключ ***271827182 71827182***

Шифротекст ***ДХТЬРЮОГЛ ДЛЩСЧЖЩУ***

Шифр Гронсфельда є окремим випадком системи шифрування Віженера.

Система шифрування Віженера

Цей шифр складної заміни можна описати таблицею шифрування, що називається таблицею (квадратом) Віженера. Таблиця Віженера (табл. 2.6) використовується для шифрування і розшифування.

Таблиця 2.6

Ключ	А	Б	В	Г	Д	…	Э	Ю	Я
0	А	Б	В	Г	Д	…	Э	Ю	Я
1	Б	В	Г	Д	Е	…	Ю	Я	А
2	В	Г	Д	Е	Ж	…	Я	А	Б
3	Г	Д	Е	Ж	З	…	А	Б	В
…	…	…	…	…	…	…	…	…	…
30	Ю	Я	А	Б	В	…	Ы	Ъ	Э
31	Я	А	Б	В	Г	…	Ъ	Э	Ю

Таблиця має два входи: верхній рядок підкреслених символів, що використовується для зчитування чергової літери вхідного відкритого тексту; крайній лівий стовпець ключа.

Послідовність ключів зазвичай отримують з числових значень літер ключового слова.

Під час шифрування вхідне повідомлення виписують у рядок, а під ним записують ключове слово або фразу. Якщо ключ виявився коротшим за повідомлення, то його циклічно повторюють. У процесі шифрування знаходять у верхньому рядку таблиці чергову літеру вхідного тексту і в лівому стовпчику – чергове значення ключа. Літеру шифротексту знаходять на перетинанні стовпця, обумовленого літерою, що шифрують, та рядка, обумовленого відповідною літерою ключа.

Приклад

Зашифруємо повідомлення ***ПРИЛЕТАЮ СЕДЬМОГО*** з використанням ключа ***АМБРОЗИЯ***:

Повідомлення ***ПРИЛЕТАЮ СЕДЬМОГО***

Ключ ***АМБРОЗИЯ АМБРОЗИЯ***

Шифротекст ***ПЪЙЫУЩИЭ ССЕКЬХЛН***

Шифр «подвійний квадрат» Вітстона

На відміну від полібіанського, подвійний квадрат використовує відразу дві таблиці, розміщені по горизонталі, а шифрування виконується біграмами, як і у шифрі Плейфейра.

Процедура шифрування виконується таким чином. Перед шифруванням вхідне повідомлення розбивається на біграми. Кожна

біграма шифрується окремо. Першу літеру біграми знаходять у лівій частині таблиці, а другу літеру – в правій (табл. 2.7). Потім будують уявний прямокутник так, щоб літери біграми знаходились у його протилежних вершинах. Інші дві вершини цього прямокутника дають літери біграми шифротексту.

Таблиця 2.7

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

Якщо обидві літери біграми знаходяться в одному рядку, то і літери шифротексту беруть з того самого рядка. Першу літеру біграми шифротексту беруть з лівої частини таблиці в стовпці, відповідному другій літері біграми повідомлення. Друга ж літера біграми шифротексту береться з правої частини таблиці в стовпці, відповідному першій літері біграми повідомлення.

Приклад.

Повідомлення ***ПРИЛЕТАЮ_ШЕСТОГО***

Шифротекст ***ПЕОВЩНФМЕШРФБЖДЦ***

Приклад виконання завдання

Як приклад виконання завдання розглянемо функцію Code, яка шифрує вхідний рядок st_in за допомогою системи Віженера з ключем Key і повертає зашифрований рядок. Оригінал тексту може містити великі літери російського алфавіту і пробіл:

```
{Функції shift і shift_ перетворять код ASCII символу
 на порядковий номер за алфавітом}
function shift (k: integer): integer; begin if k> 127 then shift: = k-127
 else shift: = 0;
 end;

 function shift_ (k: integer): integer;
 begin
 if k> 0 then shift _: = k + 127
 else shift _: = ord ( ");
 end;

function Code (st_in, key: string): string; var st: string; i, j: integer;
 k1, k2: integer;
```

```
begin
st: = ''; j: = 1;
for i: = 1 to length (st_in) do
begin
k1: = shift (ord (st_in [i]));
k2: = shift (ord (key [j]));
st: = st + chr (shift _ ((k1 + k2) mod Alfasize));
inc (j);
if j> length (key) then j: = 1;
end;
code: = st;
end;
```

Контрольні запитання

1. Які шифри називають шифрами заміни?
2. Що таке ключ шифру заміни?
3. Що називають безліччю шифропозначень?
4. Наведіть приклади шифрів простої заміни. Опишіть алгоритм одного з них.
5. Які основні недоліки шифрів простої заміни?
6. У чому відмінність шифрів простої і складної заміни?
7. Опишіть алгоритми шифрування, основані на гаслі.
8. Які існують шифри складної заміни?
9. Яким чином для шифрування використовують «подвійний квадрат» Вітстона?
10. У чому полягає шифрування з використанням системи Віженера?

Варіанти завдань

Реалізувати алгоритми шифрування (для непарних варіантів) і дешифрування (для парних) файлів за допомогою методу, зазначеного у варіанті:

1, 2. Система шифрування Цезаря.
3, 4. Афінна система підстановок Цезаря.
5, 6. Гасловий шифр.
7, 8. Полібіанський квадрат.
9, 10. Шифрувальна таблиця Трисемуса.
11, 12. Біграмний шифр Плейфейра.
13, 14. Система омофонів.
15, 16. Шифр Гронсфельда.
17, 18. Система шифрування Віженера.
19, 20. Шифр «подвійний квадрат» Вітстона.

Лабораторна робота № 3
ШИФРУВАННЯ МЕТОДОМ ГАММУВАННЯ

Мета роботи: вивчення застосування методу гаммування для шифрування та дешифрування тексту.

Теоретичні відомості

Опис методу

Гаммування є широко застосовуваним криптографічним перетворенням.

Під гаммуванням мають на увазі процес накладення за певним законом гамми шифру на відкриті дані. Гамма шифру – це псевдовипадкова послідовність, вироблена за заданим алгоритмом для шифровки відкритих даних і дешифрування зашифрованих даних.

Процес шифрування полягає в генерації гамми шифру за допомогою датчика псевдовипадкових чисел і накладення отриманої гамми на вхідний відкритий текст у зворотному порядку, наприклад, з використанням операції додавання за модулем 2.

Слід зазначити, що перед шифруванням відкриті дані розбивають на блоки $Т_о^{(i)}$ однакової довжини, зазвичай по 64 біти. Гамма шифру виробляється у вигляді послідовності блоків $Г_Ш^{(i)}$ аналогічної довжини.

Рівняння шифровки можна записати у вигляді

$$T_ш^{(i)} = Г_ш^{(i)} \oplus T_о^{(i)},\ i = 1 \ldots M,$$

де $T_ш^{(i)}$ – і-й блок шифротексту;

$Г_ш^{(i)}$ – і-й блок гамми шифру;

$T_о^{(i)}$ – і-й блок відкритого тексту;

M – кількість блоків відкритого тексту.

Процес дешифрування зводиться до повторної генерації гамми шифру і накладення цієї гамми на зашифровані дані.

Рівняння дешифрування має вигляд

$$T_о^{(i)} = Г_ш^{(i)} \oplus T_ш^{(i)},\ i = 1 \ldots M.$$

Переваги та недоліки методу

Одержуваний цим методом шифротекст досить важкий для розкриття, оскільки ключ тут є змінним. По суті, гамма шифру повинна змінюватися випадковим чином для кожного блока, що піддається шифруванню. Якщо період гамми перевищує довжину зашифрованого тексту і зловмисникові

невідома жодна з частин вхідного тексту, то такий шифр можна розкрити тільки прямим перебором усіх варіантів ключа. У цьому випадку криптостійкість шифру визначається довжиною ключа.

Однак метод гаммування не допоможе, якщо зловмисник дізнається фрагмент вхідного тексту і відповідну йому шифрограму. Простим вирахуванням за модулем виходить відрізок псевдоймовірної послідовності і за ним відновлюється вся послідовність.

Методи генерації псевдовипадкових послідовностей чисел

Під час шифрування методом гаммування як ключ використовується випадковий рядок бітів, який об'єднується з відкритим текстом, також поданим у двійковому вигляді за допомогою побітового складання за модулем 2, і в результаті виходить шифрований текст. Генерування непередбачуваних двійкових послідовностей великої довжини є однією з важливих проблем класичної криптографії. Для вирішення цієї проблеми широко використовуються генератори двійкових псевдовипадкових послідовностей.

Генеровані псевдовипадкові ряди чисел часто називають гаммою шифру, або просто гаммою (за назвою літери γ грецького алфавіту, яка часто використовується в математичних формулах для позначення випадкових величин).

Зазвичай для генерації послідовності псевдовипадкових чисел застосовують комп'ютерні програми, які, хоч і називаються генераторами випадкових чисел, насправді виробляють детерміновані числові послідовності, які за своїми властивостями дуже схожі на випадкові.

До криптографічно стійкого генератора ПВП чисел (гамми шифру) висуваються три основні вимоги:

- період гамми повинен бути досить великим для шифрування повідомлень різної довжини;
- гамма повинна бути практично непередбачуваною, що означає неможливість передбачити наступний біт гамми, навіть якщо відомі тип генератора і попередня ділянка гамми;
- генерування гамми не повинно викликати великих технічних складнощів.

Довжина періоду гамми є найважливішою характеристикою генератора ПВЧ. Після закінчення періоду числа почнуть повторюватися і їх можна буде передбачити.

Один з перших способів генерації ПВЧ на ЕОМ запропонував 1946 р. Джон фон Нейман. Суть цього способу полягає в тому, що кожне наступне випадкове число утворюється піднесенням до квадрата числа з відкиданням цифр молодших і старших розрядів. Однак цей спосіб виявився ненадійним і від нього незабаром відмовилися.

З відомих процедур генерації послідовності ПВЧ найчастіше застосовується так званий лінійний конгруентний генератор. Цей генератор виробляє послідовність ПВЧ $Y_1, Y_2, ..., Y_{i-1}, Y_i, ...,$ використовуючи співвідношення

$$Y_i = (a \cdot Y_{i-1} + b) \bmod m,$$

де Y_i – ***i*-е** (поточне) число послідовності;

Y_{i-1} – попереднє число послідовності;

M – модуль; a – множник; b – приріст;

Y_0 – породжувальне число (початкове значення).

Поточне псевдовипадкове число Y_i отримують з попереднього числа Y_{i-1} множенням його на коефіцієнт a, додаванням до приросту b і обчисленням залишку від ділення на m. Це рівняння генерує ПВЧ з періодом повторення, що залежить від вибраних значень a та b, і може набувати значення m. Значення m зазвичай встановлюється таким, що дорівнює 2^n, де n – довжина машинного слова в бітах, або таким, що дорівнює простому числу, наприклад, $m = 2^{31} - 1$. Як показав Д. Кнут, лінійний конгруентний датчик ПВЧ має максимальний період тоді і тільки тоді, коли b – непарне та $a \bmod 4 = 1$.

Для отримання послідовності ПВЧ також застосовуються адитивні і мультиплікативні генератори.

Мультиплікативний генератор за допомогою рекурентного співвідношення виробляє послідовності чисел:

$$Y_i = (a \cdot Y_{i-1}) \bmod m.$$

Вимоги до значень констант a та m такі самі, як і для лінійного конгруентного генератора.

Поточне випадкове число Y_i адитивного датчика виходить із суми чисел Y_{i-1} і Y_{i-2} обчисленням модуля від ділення цієї суми на m:

$$Y_i = (Y_{i-1} + Y_{i-2}) \bmod m.$$

Опис алгоритмів

Алгоритм шифрування

1. Проініціалізувати датчик випадкових чисел.
2. Виділити блок відкритого тексту.
3. Згенерувати гамму шифру.
4. Отримати блок зашифрованого тексту, склавши за модулем 2 блок відкритого тексту з гаммою шифру.
5. Якщо текст не закінчився, перейти до п. 2, інакше – до п. 6.
6. Кінець алгоритму шифрування.

Алгоритм дешифрування

1. Проініціалізувати датчик випадкових чисел.
2. Виділити блок зашифрованого тексту.
3. Згенерувати гамму шифру.
4. Отримати блок відкритого тексту, склавши за модулем 2 блок зашифрованого тексту з гаммою шифру.
5. Якщо зашифрований текст не закінчився, перейти до п. 2, інакше – до п. 6.
6. Кінець алгоритму дешифрування.

Приклад виконання завдання

Зашифрувати і розшифрувати текст, що знаходиться у файлі з ім'ям Source.txt. Закодований текст зберегти у файл з ім'ям Coded.txt, розшифрований текст записати у файл DeCoded.txt. Для генерації гамми використати мультиплікативний датчик зі значеннями **a = 5, m = 4096, Y0 = 4003**.

```
// ---------------------------------------------- -------------
// Текст програми
// ---------------------------------------------- -------------
// Підключення заголовних файлів для роботи з файловими // потоками

#include <iostream.h>
#include <fstream.h>
// Параметри датчика
const int a = 5;
const int m = 4096;
int y;
// Ініціалізація початкового значення
void RndInit (void)
 {
 y = 4003;
 }
// Генерація гамми шифру
// t – масив з 64 бітів (8 байтів)
void Rnd (char * t)
 {
 int i;
```

```
for (i = 0; i <8; i ++)
{
y = (a * y)% m;
t [i] = y;
}
}

void main (void)
{
char
TextSh [8], // Блок шифрованого тексту
* Ch, // Покажчик на символ для читання з файлу
Text1 [8], // Блок ДЕШИФРОВАНОГО тексту
Gamma [8]; // Гамма шифру
int i, j;
// КОДУВАННЯ
// Ініціалізація початкового значення
RndInit ();
ifstream Fin ( "Source.txt"); // Вхідний файл
ofstream Fout ( "Coded.txt"); // Файл із зашифрованим текстом
// Читання блоків тексту з файлу
while (! Fin.eof ())
{
Rnd (Gamma); // Генерація гамми шифру
// Читання поточного блока
for (i = 0; i <8; i ++)
{
Fin.read (ch, 1);
Text1 [i] = * ch;
if (Fin.eof ())
break;
}
// Шифрування блока
for (j = 0; j <i; j ++)
TextSh [j] = Text1 [j] ^ Gamma [j];
// Запис блока шифрованого тексту в файл
Fout.write (TextSh, i);
}
```

```
Fin.close ();
Fout.close ();
// ДЕКОДУВАННЯ / Ініціалізація початкового значення
RndInit ();
Fin.open ( "Coded.txt"); // Вхідний файл
Fout.open ( "DeCoded.txt"); // Файл з розшифрованим текстом
// Читання блоків шифру з файлу
while (! Fin.eof ())
{
Rnd (Gamma); // Генерація гамми шифру
// Читання поточного блока
for (i = 0; i <8; i ++)
{
Fin.read (ch, 1);
Text1 [i] = * ch;
if (Fin.eof ())
break;}
// Дешифрування блока
for (j = 0; j <i; j ++)
TextSh [j] = Text1 [j] ^ Gamma [j];
// Запис блока розшифрованого тексту в файл
Fout.write (TextSh, i);
}
Fin.close ();
Fout.close ();
}
```

Контрольні запитання

1. Що таке гаммування? Що розуміють під гаммою шифру?
2. Які операції можна застосовувати при накладенні гамми?
3. У чому полягають процеси шифрування та дешифрування?
4. Які переваги і недоліки має метод гаммування?
5. Які вимоги висуваються до криптографічно стійкого генератора ПВЧ?
6. Чому найбільш важливою є довжина періоду гамми?
7. Опишіть лінійний конгруентний спосіб генерації ПВЧ.
8. Опишіть адитивний і мультиплікативний генератори ПВЧ.
9. Опишіть алгоритми шифрування та дешифрування відкритого тексту методом гаммування.

Варіанти завдань

Завдання згруповані по два. Непарні варіанти пишуть програму шифрування тексту із застосуванням зазначеного методу, парні варіанти програмують дешифратор.

Зашифрувати і розшифрувати текст, що знаходиться у файлі з ім'ям Source.txt. Закодований текст зберегти у файл з ім'ям Coded.txt, розшифрований текст записати у файл DeCoded.txt.

Для генерації гамми використовувати:

1, 2. Адитивний датчик зі значеннями m = 4096, Y0 = 4003, Y1 = 59.

3, 4. Лінійний конгруентний датчик зі значеннями a = 5, b = 7, m = 4096, Y0 = 4003, Y1 = 59.

5, 6. Мультиплікативний датчик зі значеннями a = 7, m = 4096, Y0 = 502.

7–12. Датчик з найбільшим періодом; датчики і значення для них вибрати з попередніх варіантів. Гамму генерувати за допомогою вказаного датчика. Підібрати для нього такі значення параметрів, щоб період був найбільшим.

13–16. Мультиплікативний датчик.

17–20. Лінійний конгруентний датчик.

Лабораторна робота № 4
ОДНОРАЗОВА СИСТЕМА ШИФРУВАННЯ

Мета роботи: ознайомлення з традиційними симетричними криптографічними системами – шифрами складної заміни, вивчення алгоритму кодування одноразової системи шифрування.

Теоретичні відомості

Традиційні симетричні криптосистеми (шифри складної заміни)

Більшість засобів захисту інформації базується на використанні криптографічних шифрів і процедур шифрування – розшифрування. Під шифром розуміють сукупність обернених перетворень множини відкритих даних на множину закритих даних, що задаються ключем і алгоритмом криптографічного перетворення.

Ключ – це конкретний секретний стан деяких параметрів алгоритму криптографічного перетворення даних, що забезпечує вибір тільки одного варіанта з усіх можливих для цього алгоритму.

Основною характеристикою шифру є криптостійкість, тобто його стійкість до розкриття методами криптоаналізу. Зазвичай ця характеристика визначається інтервалом часу, необхідним для розкриття шифру.

До шифрів, що використовуються для криптографічного захисту інформації, висувається кілька вимог:

- достатня криптостійкість (надійність закриття даних);
- простота процедур шифрування і розшифрування;
- незначна надмірність інформації завдяки шифруванню;
- нечутливість до невеликих помилок шифрування та ін.

Тією чи іншою мірою цим вимогам відповідають:

- шифри перестановок;
- шифри заміни;
- шифри гаммування;
- шифри, основані на аналітичних перетвореннях даних.

Шифрування заміною (підстановкою) полягає в тому, що символи шифрованого тексту замінюються символами того самого або іншого алфавіту відповідно до заздалегідь обумовленої схеми заміни.

Процеси шифрування і розшифрування здійснюються в межах деякої криптосистеми. Характерною особливістю симетричних криптосистем є застосування того самого секретного ключа як під час шифрування, так і розшифрування повідомлень.

І відкритий текст, і шифротекст утворюються з літер алфавіту.

У загальному вигляді деякий алфавіт A можна подати так:

$$A=\{a_0, a_1, \ldots, a_{m-1}\}$$

Під час виконання криптографічних перетворень корисно замінити літери алфавіту цілими числами. Це дає змогу спростити виконання необхідних алгебричних маніпуляцій. Наприклад, можна встановити взаємно однозначну відповідність між російським алфавітом $A = \{АБВГ...ЮЯ\}$ і множиною цілих $Z_{32} = \{0, 1, 2, 3, ..., 31\}$.

Текст з n літерами з алфавіту Z_m можна розглядати як n-граму

$$X = (x_0 , x_1 , x_2 ,\ldots., x_{n-1}),$$

де $x_i \in Z_m$, $0 <= i<n.$ Для деякого цілого $n = 1, 2, 3, ...$.

Через Z_m, n будемо позначати множину n-грам, утворених з літер множини Z_m.

Криптографічне перетворення E являє собою сукупність перетворень:

$$E = \{E^{(n)}:1 <= n<\infty\},$$
$$E^{(n)} : Z_{m,n} \rightarrow Z_{m,n}.$$

Перетворення $E^{(n)}$ визначає, як кожна n-грама відкритого тексту $x \in Zm, n$ замінюється n-грамою шифротексту y, тобто

$Y = E^{(n)}(x)$, причому $x, y \in Z_{m,n}$.

Криптографічна система може розглядатися як ряд криптографічних перетворень

$$E=\{E_k:K \in K\},$$

помічених параметром K, який називається ключем.

Множина значень ключа утворює ключовий простір K.

Шифри складної заміни називають багатоалфавітними, тому що для шифрування кожного символу вхідного повідомлення застосовують свій шифр простої заміни. Багатоалфавітна підстановка послідовно і циклічно змінює використовувані алфавіти.

При r-алфавітній підстановці символ x_0 вхідного повідомлення замінюється символом y_0 з алфавіту B_0, символ x_1 – символом y_1 з алфавіту B_1 і т. д.

Наприклад, для $r = 4$ вхідні символи – $X_0\ X_1\ X_2\ X_3\ X_4\ X_5\ X_6\ X_7\ X_8\ X_9$, алфавіт підстановки – $B_0\ B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ B_8\ B_9$.

Ефект використання багатоалфавітної підстановки полягає в тому, що забезпечується маскування природної статистики вхідної мови, оскільки конкретний символ з вхідного алфавіту A може бути перетворений на кілька різних символів шифрувальних алфавітів B_j. Ступінь забезпечення захисту теоретично пропорційний довжині періоду r у послідовності використовуваних алфавітів B_j.

Одноразова система шифрування

Майже всі застосовувані на практиці шифри характеризуються як умовно надійні, оскільки в принципі вони можуть бути розкриті за наявності необмежених обчислювальних можливостей. Абсолютно надійні шифри не можна зруйнувати навіть при використанні необмежених обчислювальних можливостей. Існує єдиний такий шифр, що застосовується на практиці, – одноразова система шифрування. Характерною її особливістю є одноразове використання ключової послідовності.

Одноразова система шифрує вхідний відкритий текст

$$X = (X_0, X_i, ... X_{n-i})$$

у шифротекст

$$Y = (Y_0, Y_i, ... , Y_{n-i})$$

за допомогою підстановки Цезаря

$$Y_i = (X_i + K_i) \bmod m, \quad 0 <= i < n,$$

де K_i – i-й елемент випадкової ключової послідовності.

Ключовий простір K одноразової системи є набором дискретних випадкових величин з Z_m і містить m^n значень.

Процедура розшифрування описується співвідношенням

$$X_i = (Y_i - K_i) \bmod m,$$

де K_i – i-й елемент тієї самої випадкової ключової послідовності.

Одноразова система винайдена 1917 р. американцями Дж. Моборном і Г. Вернамом. Для реалізації цієї системи підстановки іноді використовують одноразовий блокнот. Він складається з відривних сторінок, на кожній з яких надрукована таблиця з випадковими числами (ключами) K_J. Блокнот виконується в двох примірниках: один використовує відправник, а інший – одержувач. Для кожного символу X_i повідомлення використовується свій ключ K_J з таблиці тільки один раз. Після того як таблиця використана, її потрібно видалити з блокнота і знищити. Шифрування нового повідомлення починається з нової сторінки.

Цей шифр абсолютно надійний, якщо набір ключів K_J дійсно випадковий і непередбачуваний. Якщо криптоаналітик спробує використати для заданого шифротексту всі можливі набори ключів і відновити всі можливі варіанти вхідного тексту, то вони виявляться рівноймовірними. Не існує способу вибрати вхідний текст, який був дійсно посланий. Теоретично доведено, що одноразові системи не розкриваються, оскільки їх шифротекст не містить достатньої інформації для відновлення відкритого тексту.

Здавалося б, завдяки цій перевазі одноразові системи слід застосовувати у всіх випадках, що потребують абсолютної інформаційної безпеки. Однак можливості застосування одноразової системи обмежені

суто практичними аспектами. Суттєвим моментом є вимога одноразового використання випадкової ключової послідовності. Ключова послідовність з довжиною, що не менше довжини повідомлення, повинна передаватися одержувачу повідомлення заздалегідь або окремо деяким секретним каналом. Ця вимога не буде занадто обтяжливою для передачі дійсно важливих одноразових повідомлень, наприклад, по гарячій лінії Вашингтон – Москва. Однак така вимога практично нездійсненна для сучасних систем оброблення інформації, де потрібно шифрувати не один мільйон символів.

У деяких варіантах одноразового блокнота вдаються до простішого керування ключовою послідовностю, але це дещо знижує надійність шифру. Наприклад, ключ визначається зазначенням місця в книзі, відомої відправнику і одержувачу повідомлення. Ключова послідовність починається з указаного місця цієї книги і використовується таким же чином, як у системі Віженера.

При шифруванні вхідного повідомлення його виписують у рядок, а під ним записують ключове слово (фразу). Якщо ключ виявився коротшим за повідомлення, то його циклічно повторюють. У процесі шифрування символ вхідного тексту замінюють на деякий інший символ, віддалений від нього на кількість літер, що відповідає символу ключа.

Іноді такий шифр називають шифром з бігаючим ключем. Керування ключовою послідовністю в такому варіанті шифру набагато простіше, оскільки довга ключова послідовність може бути подана в компактній формі. Але, з іншого боку, ці ключі не будуть випадковими. Тому у криптоаналітика з'являється можливість використовувати інформацію про частоти літер вхідної природної мови.

Опис послідовності дій для алгоритму одноразового шифрування

Для шифрування деякого повідомлення $A_0, \ldots, An-1$, що складається з літер алфавіту $A = \{a_0, \ldots, a_{m-1}\}$, необхідно:

- побудувати схему заміщення літер алфавіту цілими числами від 1 до $mA \rightarrow X$, що дає змогу здійснювати обернене перетворення $X \rightarrow A$ (наприклад, $a_0 - 1, a_1 - 2, \ldots, a_{m-1} - m$);
- перетворити вхідний текст на множину $X = (X_0, \ldots, X_{n-1})$;
- отримати послідовність $K = (X_0, \ldots, X_{n-1})$ випадкових рівномірно розподілених чисел від 0 до $m - 1$ (0 <= K <= m – 1);
- виконати підстановку Цезаря $Y_i = (X_i + K_i) \bmod m$;
- для отриманої послідовності $Y = (Y_0, \ldots, Y_{n-1})$ зробити обернене перетворення з числового подання Y в літери алфавіту A: $Y \rightarrow B$.

Отримана послідовність $B = (B_0, \ldots, B_{n-1})$ буде шифротекстом, а послідовність $K = (K_0, \ldots, K_{n-1})$ – ключем.

Для розшифрування повідомлення $B_0, \ldots, B_{n-1}$ необхідно провести дії в зворотному порядку, замінивши підстановку Цезаря на формулу $X_i = (Y_i - K_i) \bmod m$.

Приклад шифрування та дешифрування в одноразовій системі шифрування

Нехай необхідно закодувати повідомлення «***СООБЩЕНИЕ***».
1. Алфавітом для вхідного тексту будемо вважати російський алфавіт.
2. ***С*** $\rightarrow$ ***17***
О $\rightarrow$ ***14***
О $\rightarrow$ ***14***
Б $\rightarrow$ ***1***
Щ $\rightarrow$ ***25***
Е $\rightarrow$ ***5***
Н $\rightarrow$ ***13***
И $\rightarrow$ ***8***
Е $\rightarrow$ ***5.***
3. ***K = (7, 12, 1, 5, 21, 25, 15, 8, 10)***.
4. ***Y = (24, 26, 15, 6, 14, 30, 18, 16, 15)***.
5. ***B = «ШЬПЖОЮТРП»***.
Декодування матиме такий вигляд:
B $\rightarrow$ ***Y: Y = (24, 26, 15, 6, 14, 30, 18, 16, 15);***
X: $X_i = (Y_i - K_i)$, ***X = (17, 14, 14, 1, 25, 5, 13, 8, 5);***
A: X $\rightarrow$ ***A, A = «ПІДПРИЄМСТВОМ».***

Приклад програмної реалізації системи одноразового кодування

Програма дає змогу кодувати файли на основі таблиці ASCII.
Текст програми:

```
program onetimecoding;
 var
 FileNameToCode, FileNameToDecode, FileNameWithSeed: string;

{отримує нову послідовність випадкових чисел}
procedure GetNewSeed (N, m: integer; FSeed: string);
 var
 F: file of byte;
 i: integer;
 Ki: byte;
 K: string;
 begin
 assign (F, FSeed);
 rewrite (F);
 randomize;
 for i: = 1 to N do
```

```
begin
Ki: = random (m);
write (F, Ki)
end;
close (F)
end;

{Кодує текст}
procedure Code (FileName, Seed, FNameToSaveCodedText: string; m: integer);
var
Fprimery, Fcodedtext, Fcode: file of char;
X, Y, K: char;
i, j: integer;

begin
assign (Fprimery, FileName);
assign (Fcode, Seed);
assign (Fcodedtext, FNameToSaveCodedText);
reset (Fprimery);
GetNewSeed (filesize (Fprimery), m, Seed);
reset (Fcode);
rewrite (Fcodedtext);
while not EOF (Fprimery) do
begin
read (Fprimery, X);
read (Fcode, K);
Y: = chr ((ord (X) + ord (K)) mod m);
write (Fcodedtext, Y)
end;
close (Fprimery);
close (Fcode);
close (Fcodedtext);
end;

{Декодує текст}
procedure Decode (FileName, Seed, FNameToSaveDecodedText: string; m: integer);
var
Fcodedtext, Fcode, Fdecodedtext: file of char;
X, Y, K: char;
begin
assign (Fcodedtext, FileName);
assign (Fcode, Seed);
```

```
 assign (Fdecodedtext, FNameToSaveDecodedText);
 rewrite (Fdecodedtext);
 reset (Fcode);
 reset (Fcodedtext);
 while not EOF (Fcodedtext) do
 begin
 read (Fcodedtext, Y);
 read (Fcode, K);
 X: = chr ((ord (Y) -ord (K)) mod m);
 write (Fdecodedtext, X)
 end;
 close (Fdecodedtext);
 close (Fcode);
 close (Fcodedtext);
 end;
begin
 Code ( 'text', 'seed', 'ctext', 256);

 Decode ( 'ctext', 'seed', 'newtext', 256);
end.
```

Контрольні запитання

1. Що є характерною особливістю симетричних систем?
2. Що таке ключ?
3. У чому полягає суть шифрування заміною (підстановкою)?
4. Що дає змогу спростити виконання необхідних алгебричних дій у процесі шифрування?
5. Чим відрізняються шифри складної заміни від простих шифрів підстановки?
6. Чому одноразова система шифрування так називається?
7. На основі яких перетворень здійснюється кодування?
8. Які переваги й недоліки одноразової системи шифрування?
9. Чи можна розшифрувати закодоване повідомлення, не знаючи ключа? Якщо так, то в яких випадках?
10. Який з цих випадків є гіршим з огляду на дешифрування? Під час передачі було спотворено і-й символ:
    а) закодованого повідомлення;
    б) ключа.
11. Чи доцільним є використання одноразової системи шифрування в сучасних умовах? Відповідь аргументуйте.

Варіанти завдань

Скласти програму шифрування повідомлення за допомогою алгоритму одноразової системи шифрування і його розшифрування (непарний варіант – шифрування повідомлення, парний – розшифрування). Оригінал тексту, кодоване повідомлення, ключ і відновлене повідомлення зберігати у файлах. Алфавіт вхідного повідомлення: варіанти 1–10 – англійська мова; варіанти 11–20 – російська мова (табл. 4.1).

Таблиця 4.1

Номер варіанта	Повідомлення
1, 2	THE MESSAGE
3, 4	BE BACK TOMORROW
5, 6	CODE ME PLEASE
7, 8	WAIT ME ON THE SAME PLACE
9, 10	DO YOU STILL TRY TO CODE ME
11, 12	СООБЩЕНИЕ
13, 14	ЗАВТРА ТАМ ЖЕ
15, 16	ШИФР – ОСНОВА ЗАЩИТЫ
17, 18	А РАСШИФРОВАТЬ СМОЖЕШЬ
19, 20	ПОПРОБУЙ СНОВА

Лабораторна робота № 5
ШИФРУВАННЯ АЛГОРИТМОМ DES

Мета роботи: вивчення можливостей алгоритму шифрування DES.

Теоретичні відомості

Опис алгоритму

Алгоритм шифрування DES здійснює шифрування 64-бітових блоків даних за допомогою 64-бітового ключа. Процес шифрування полягає в початковій перестановці бітів 64-бітового блока, в 16 циклах шифрування, а розшифрування – в 16 циклах розшифрування і кінцевій перестановці бітів.

Слід відразу зазначити, що всі наведені таблиці є стандартними і повинні включатися в реалізацію алгоритму DES в незмінному вигляді. Всі перестановки і коди в таблицях підібрані розробниками таким чином, щоб максимально ускладнити процес розшифрування шляхом підбору ключа.

Нехай з файлу вхідного тексту зчитаний черговий 64-бітовий (8-байтовий) блок ***T***. Цей блок перетвориться за допомогою матриці початкової перестановки IP. Біти вхідного блока (64 біти) переставляються відповідно до матриці IP (табл. 5.1).

Таблиця 5.1

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

Отримана послідовність бітів розділяється на дві послідовності: L_0 – ліві, або старші біти, R_0 – праві, або молодші біти, кожна з яких містить 32 біти. Потім виконується ітеративний процес шифрування, що складається з 16 циклів. Нехай T_i – результат i-ї ітерації:

$$T_i = L_i R_i,$$

де L_i – перші 32 біти послідовності;

R_i – останні 32 біти послідовності.

Тоді результат i-ї ітерації можна описати формулами

$$L_i = R_{i-1},\ i = 1, 2, 3, \ldots, 16;$$
$$R_i = L_{i-1}\ xor\ f(R_{i-1}, K_i),\ i = 1, 2, 3, \ldots, 16.$$

Функція ***f*** називається функцією шифрування. Її аргументами є послідовність R_{i-1}, отримана на попередньому кроці ітерації, і 48-бітовий ключ K_i, який є результатом перетворень 64-бітового ключа шифру ***K***.

На останньому кроці ітерації отримують послідовності R_{16} і L_{16}, які конкатенуються в 64-бітову послідовність.

Після закінчення шифрування здійснюється відновлення позицій бітів за допомогою матриці зворотної перестановки IP^{-1} (табл. 5.2).

Таблиця 5.2

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	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

Елементи матриць ***IP*** і IP^{-1} пов'язані (табл. 5.3).

Таблиця 5.3

Елемент IP	40	8	48	16	56	24	64 ...
Елемент IP^{-1}	1	2	3	4	5	6	7 ...

Процес розшифрування даних є інверсним відносно процесу шифрування. Всі дії повинні бути виконані в зворотному порядку. Це означає, що дані, які підлягають розшифруванню, спочатку переставляються відповідно до матриці IP^{-1}, а потім над послідовністю бітів $R_{16}L_{16}$ виконуються ті самі дії, що і в процесі шифрування, але в зворотному порядку.

Ітеративний процес розшифрування може бути описаний такими формулами:

$$R_i = L_i,\ i=1,2,3,\ \ldots,\ 16;$$
$$L_{i-1} = R_i \text{ xor } f(L_i, K_i),\ i = 1,2,3,\ \ldots,\ 16.\ R_{16}L_{16}.$$

Отже, для процесу розшифрування з переставленим вхідним блоком $R_{16}L_{16}$ на першій ітерації використовується ключ K_{16}, на другій – K_{15} і т. д. На 16-й ітерації використовується ключ K_1, на останньому кроці ітерації будуть отримані послідовності L_0 і R_0, які конкатенуються в 64-бітову послідовність L_0R_0.

Потім у цій послідовності 64 біти переставляються відповідно до матриці ***IP***. Результат такого перетворення – вхідна послідовність бітів (рис. 5.1).

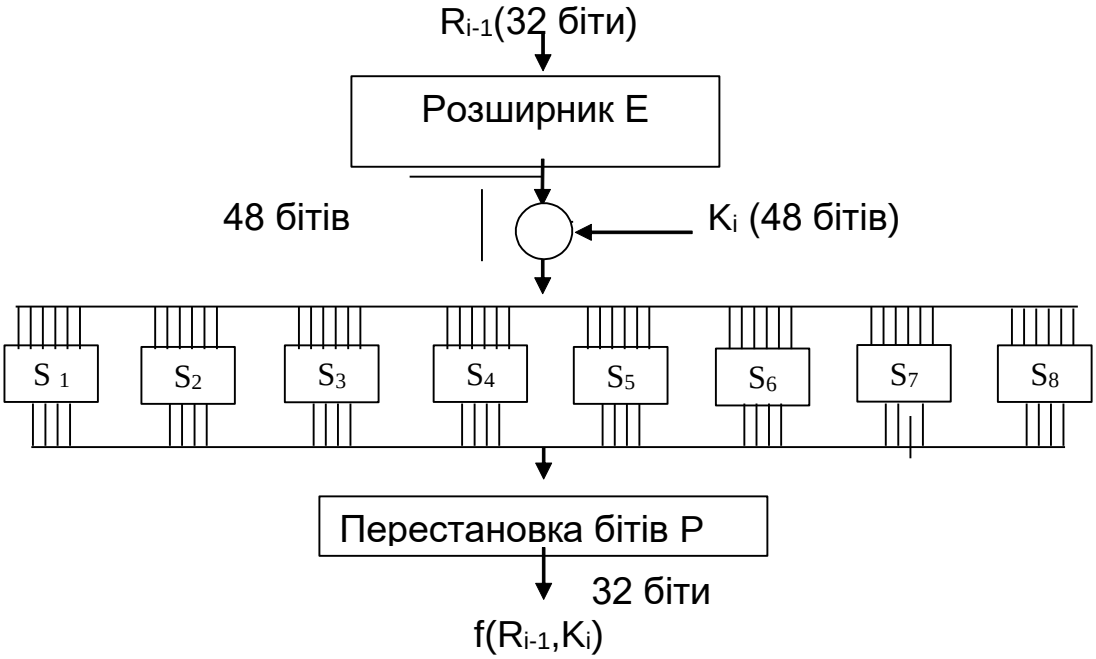


Рис. 5.1

Тепер розглянемо, що ховається під перетворенням, позначеним літерою ***f***. Для обчислення значення функції ***f*** використовуються:

- функція ***E*** (розширення від 32 до 48 бітів);
- функція ***S₁, S₂, ..., S₈*** (перетворення 6-бітового числа на 4-бітове);
- функція ***P*** (перестановка бітів у 32-бітовій послідовності).

Наведемо визначення цих функцій.

Функція розширення ***E*** приймає блок з 32 бітів і породжує блок з 48 бітів відповідно до матриці ***E*** (табл. 5.4).

Таблиця 5.4

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	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

Отриманий результат складається за модулем 2 (операція ***XOR***) з поточним значенням ключа K_i і потім розбивається на вісім 6-бітових блоків ***B₁, B₂, ..., B₈***. Далі кожен з цих блоків використовується як номер елемента в функціях ***S₁, S₂, ..., S₈***, що містять 4-бітові значення.

Нехай на вхід функції S_j надходить 6-бітовий блок $B_j = b_1b_2b_3b_4b_5b_6$, тоді 2-бітове число b_1b_6 буде вказувати номер рядка матриці, а 4-бітове число $b_2b_3b_4b_5$ – номер стовпця (табл. 5.5).

Таблиця 5.5

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

Сукупність 6-бітових блоків $B_1, B_2, \ldots, B_8$ забезпечує вибір 4-бітового елемента в кожній з функцій $S_1, S_2, \ldots, S_8$. В результаті отримуємо $S_1(B_1)\ S_2(B_2)\ \ldots\ S_8(B_8)$, тобто 32-бітовий блок (оскільки матриці S_j містять 4-бітові елементи). Цей 32-бітовий блок перетвориться за допомогою функції перестановки бітів ***P*** (табл. 5.6). Таким чином, функція шифрування $f(R_{i-1},K_i) = P(S_1(B_1)\ S_2(B_2)\ \ldots\ S_8(B_8))$.

Таблиця 5.6

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

На кожній ітерації використовується нове значення ключа K_i (довжиною 48 бітів), яке обчислюється з початкового 64-бітового ключа ***K*** з вісьмома бітами контролю парності, розташованими в позиціях 8, 16, 24, 32, 40, 48, 56, 64. Для видалення контрольних бітів і підготовки ключа до роботи використовується функція ***G*** початкової підготовки ключа (табл. 5.7).

Таблиця 5.7

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

Результат перетворення $G(K)$ розбивається на дві половини C_0 і D_0 по 28 бітів кожна. За першими чотирма рядками матриці ***G*** визначають, як вибираються біти послідовності C_0, наступні чотири рядки – послідовності D_0. Для генерації послідовностей C_0 і D_0 не використовуються біти 8, 16, 24, 32, 40, 48, 56 і 64 ключа шифру. Таким чином, в дійсності ключ шифру є 56-бітовим (рис. 5.2).

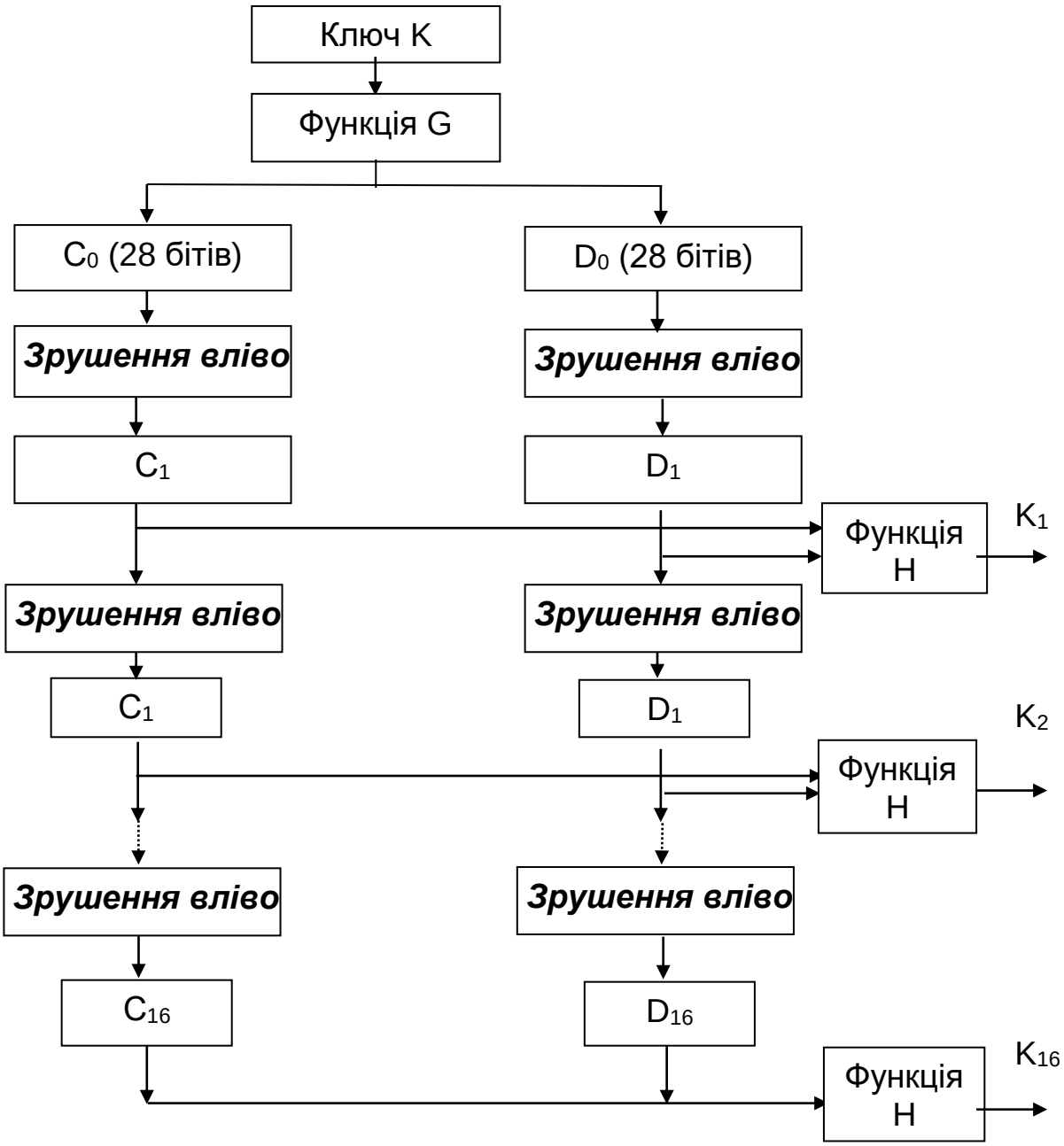


Рис. 5.2

Після визначення ***Co*** і ***Do*** рекурсивно визначаються C_i і D_i, ***i = 1, 2, ..., 16***. Для цього застосовуються операції циклічного зрушення вліво на один або два біти залежно від номера кроку ітерації, як показано в табл. 5.8.

Таблиця 5.8

Номер ітерації	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Кількість зрушень (бітів)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Ключ K_i, який визначається на кожному кроці ітерації, є результат вибору конкретних бітів з 56-бітової послідовності $C_i D_i$ та їх перестановки.

Іншими словами, ключ $K_i = H(C_i D_i)$ (табл. 5.9).

Таблиця 5.9

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	47	55	
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Основні режими роботи алгоритму DES

Алгоритм DES дає змогу безпосередньо перетворювати 64-бітовий вхідний відкритий текст на 64-бітовий вхідний шифрований текст, проте дані рідко обмежуються 64 розрядами. Для використання алгоритму DES при вирішенні різноманітних криптографічних завдань розроблено чотири робочих режими:

- електронна кодова книга ECB (Electronic Code Book);
- зчеплення блоків шифру CBC (Cipher Block Chaining);
- зворотний зв'язок за шифротекстом CFB (Cipher Feed Back);
- зворотний зв'язок за виходом OFB (Output Feed Back).

Режим «Електронна кодова книга»

Довгий файл розбивають на 64-бітові відрізки (блоки) по вісім байтів. Кожен з цих блоків шифрують незалежно з використанням того самого ключа шифрування. Основна перевага – простота реалізації, недолік – відносно слабка стійкість проти кваліфікованих криптоаналітиків. Через фіксований характер шифрування при обмеженій довжині блока (64 біти) можливе проведення криптоаналізу «зі словником». Блок такого розміру може повторитися в повідомленні внаслідок великої надлишковості в тексті природною мовою. Це приводить до того, що ідентичні блоки відкритого тексту в повідомленні будуть подані ідентичними блоками шифротексту, що дає криптоаналітику деяку інформацію про зміст повідомлення.

Режим «Зчеплення блоків шифру»

У цьому режимі вхідний файл ***M*** розбивається на 64-бітові блоки: $M = M_1M_2...M_n$. Перший блок M_1 складається за модулем 2 з 64-бітовим початковим вектором IV, який змінюється щодня і тримається в секреті. Отримана сума потім шифрується з використанням ключа DES. Отриманий 64-бітовий шифр C_1 складається за модулем 2 з другим блоком тексту, результат шифрується, і виходить другий 64-бітовий шифр C_2, і т. д. Процедура повторюється доти, доки не будуть оброблені всі блоки тексту. Таким чином, для всіх значень $i = 1 ... n$ (n – кількість блоків) результат шифрування C_i визначається таким чином: $C_i = DES (M_i \text{ xor } C_{i-1})$, де C_0 = IV – початкове значення шифру, що дорівнює початковому вектору ініціалізації. Очевидно, що останній 64-бітовий блок шифротексту є функцією секретного ключа, початкового вектора і кожного біта відкритого тексту незалежно від його довжини. Цей блок шифротексту називають кодом аутентифікації повідомлення (КАП). Код КАП може бути легко перевірено одержувачем, який володіє секретним ключем і початковим вектором, шляхом повторення процедури, виконаної відправником. Перевага цього режиму полягає в тому, що він не дає змоги накопичуватися помилкам при передачі. Блок ***M*** є функцією тільки C_{i-1} і C_i. Тому помилка при передачі приведе до втрати тільки двох блоків вхідного тексту (рис. 5.3).

Режим «Зворотний зв'язок за шифром»

У цьому режимі розмір блока може бути більше або менше 64 бітів. Файл, що підлягає шифруванню, зчитується послідовними блоками довжиною ***k*** бітів ($k = 1...64$). Вхідний блок (64 біти) спочатку містить вектор ініціалізації, вирівняний за правим краєм. Припустимо, що внаслідок розбиття на блоки було отримано ***n*** блоків довжиною ***k*** бітів кожен (залишок дописується нулями або пропусками). Тоді для будь-якого $i = 1...n$ блок шифротексту $C_i = M_i \text{ xor } P_{i-1}$, де P_{i-1} позначає ***k*** старших бітів попереднього зашифрованого блока. Оновлення вхідного блока здійснюється шляхом зсуву його старших ***k*** бітів на ***k*** і запису в нього C_i. Для відновлення зашифрованих даних P_{i-1} і C_i обчислюються аналогічним чином та $M_i = C_i \text{ xor } P_{i-1}$ (рис. 5.4).

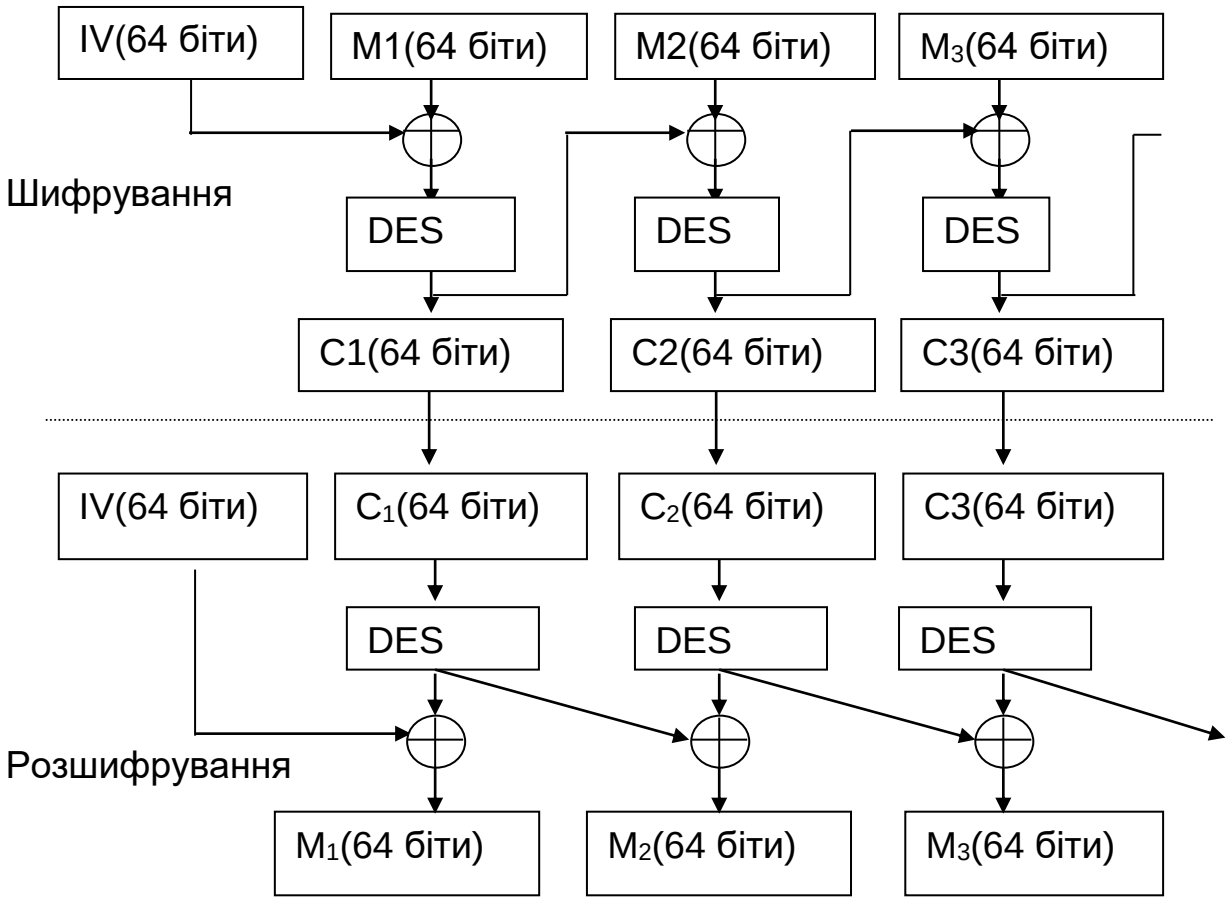

IV(64 біти)
M1(64 біти)
M2(64 біти)
M3(64 біти)
Шифрування
DES
DES
DES
C1(64 біти)
C2(64 біти)
C3(64 біти)
IV(64 біти)
C1(64 біти)
C2(64 біти)
C3(64 біти)
DES
DES
DES
Розшифрування
M1(64 біти)
M2(64 біти)
M3(64 біти)

Рис. 5.3

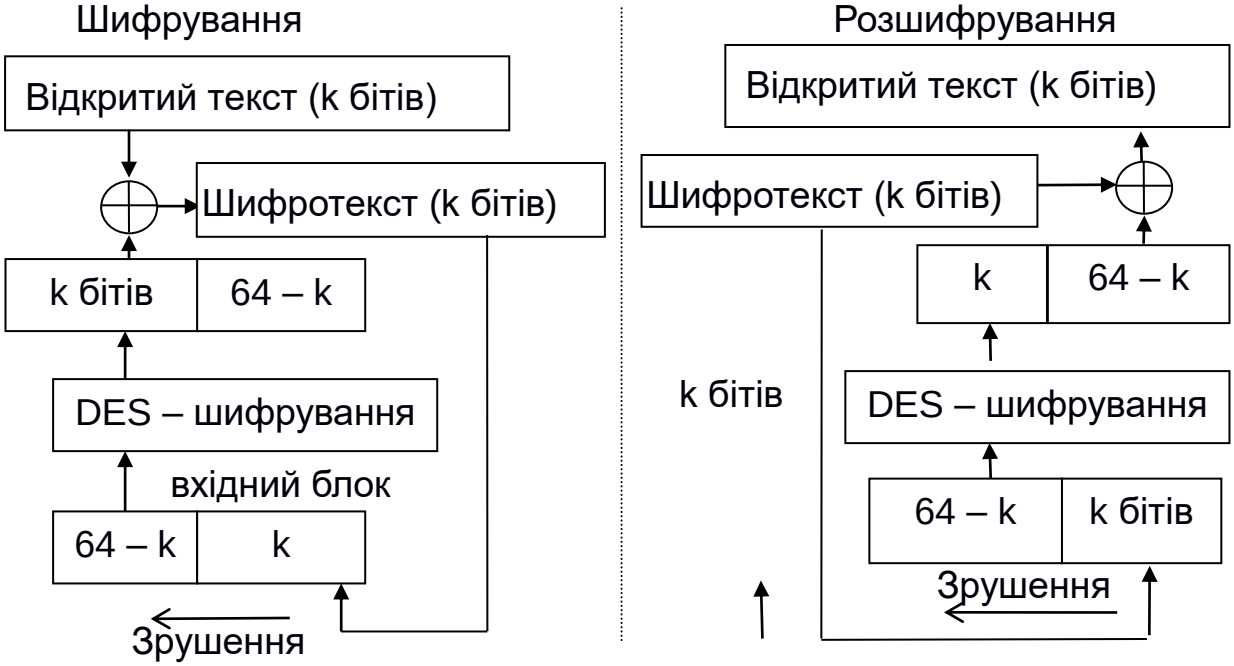

Шифрування
Відкритий текст (k бітів)
Шифротекст (k бітів)
k бітів
64 – k
DES – шифрування
вхідний блок
64 – k
k
Зрушення
Розшифрування
Відкритий текст (k бітів)
Шифротекст (k бітів)
k
64 – k
k бітів
DES – шифрування
64 – k
k бітів
Зрушення

Рис. 5.4

Режим «Зворотний зв'язок за виходом»

Цей режим також використовує змінний розмір блока і зсувний регістр, ініціалізовані так само, як у режимі СРВ: вхідний блок спочатку містить вектор ініціалізації IV, вирівняний за правим краєм. При цьому для кожного сеансу шифрування даних необхідно використовувати новий початковий стан регістра, який потрібно пересилати по каналу відкритим текстом. Покладемо $M = M_i, M_2, ..., M_n$. Для всіх $i = 1 ... n$, $C_i = M_i$ **xor** P_i, де P_i – старші k бітів операції DES (C_{i-1}). На відміну від режиму зворотного зв'язку за шифром, оновлення зсувного регістру здійснюється шляхом відкидання старших k бітів і дописування справа P_i (рис. 5.5).

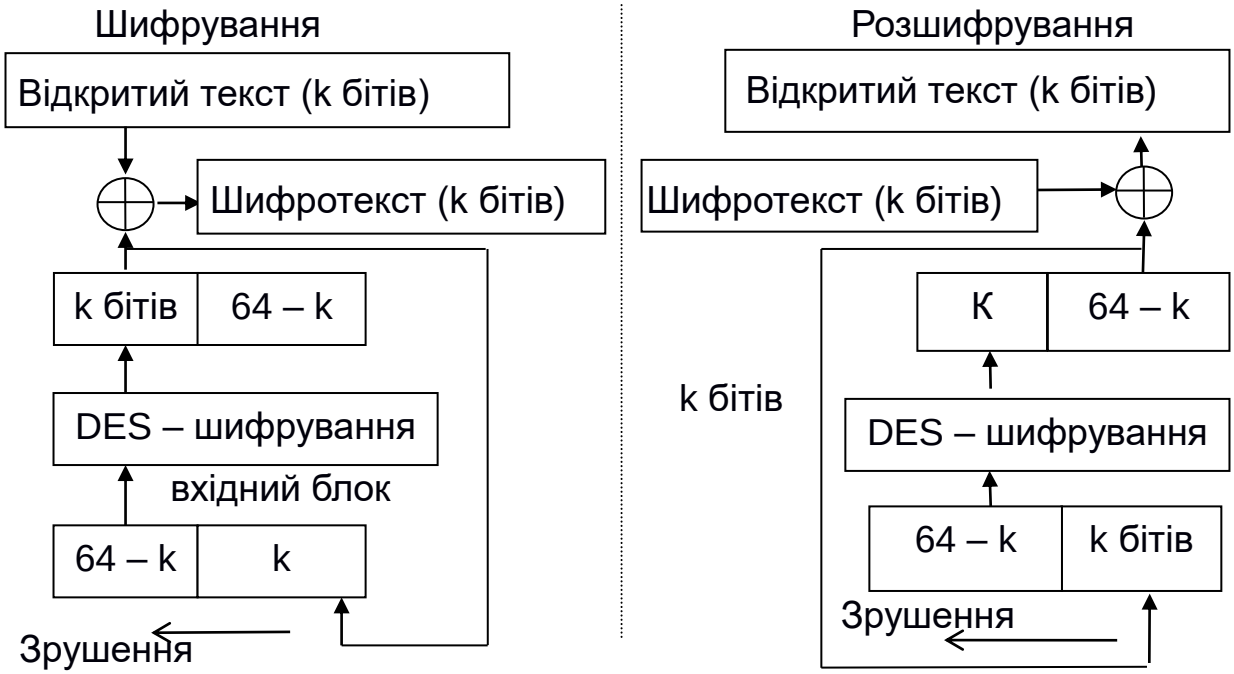


Рис. 5.5

Приклад виконання завдання

Процес шифрування:

```
perestanovkaIp (M64); {початкова перестановка масиву 64 бітів}
 For i: = 1 to 32 do {розбиття вхідної послідовності на дві 32-бітові}
 begin
 L_new [i]: = M64 [i];
 R_new [i]: = M64 [i + 32];
 end;
 For j: = 1 to 16 do {16 циклів шифрування}
 begin {for j}
```

```
L_old: = L_new;
R_old: = R_new;
F (R_old, k48 [j], R_out1);
XOR (L_old, R_out1, R_out2);
L_new: = R_old;
R_new: = R_out2;
end; {for j}
For i: = 1 to 32 do {конкатенація послідовності}
begin {for i}
M64 [i]: = R_new [i];
M64 [i + 32]: = L_new [i];
end {for i};
```

Процес розшифрування:

```
For i: = 1 to 32 do begin {розбиття кінцевої послідовності на дві
32-бітові}
R_new [i]: = M64 [i];
L_new [i]: = M64 [i + 32];
end;
For j: = 16 downto 1 do begin {for j} {16 циклів розшифрування}
L_old: = L_new;
R_old: = R_new;
F (L_old, k48 [j], R_out1);
XOR (R_old, R_out1, R_out2);
R_new: = L_old;
L_new: = R_out2;
end; {for j}
For i: = 1 to 32 do begin {for i} {конкатенація послідовності}
m64 [i]: = l_new [i];
m64 [i + 32]: = R_new [i];
end {for i};
perestanovkaIp_minus1 (M64); {кінцева перестановка масиву 64 бітів}
```

Процедура перестановки perestanovkaIp може бути реалізована таким чином:

```
procedure perestanovkaIP (var M64: Mas64);
var M64temp: mas64;
const vectorIp: array [1..64] of byte =
(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);
begin {perestanovkaIP}
for i: = 1 to 64 do M64temp [vectorIp [i]]: = M64 [i];
for i: = 1 to 64 do M64 [i]: = M64temp [i];
end{ perestanovkaIP}.
```

Аналогічно реалізуються всі перестановки: IP-1, E, P, G, H.

Функція F може бути представлена так:

```
Procedure F (R32_local: mas32; k48_local: mas48; var Rout_local: mas32);
procedure S (B6: mas6; jj: byte; var B4: mas4);
const vector_s: array [0..31,0..15] of byte =
((14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7), ... (2, 1,14, 7, 4,10, 8 , 13,15,12, 9, 0, 3, 5,));
var nstr, nstlb, z: byte;
begin {S}
nstr: = 0;
if b6 [1] = 1 then nstr: = nstr + 2;
if b6 [6] = 1 then nstr: = nstr + 1;
if b6 [2] = 1 then nstlb: = nstlb + 8;
if b6 [3] = 1 then nstlb: = nstlb + 4;
if b6 [4] = 1 then nstlb: = nstlb + 2;
if b6 [5] = 1 then nstlb: = nstlb + 1;
z: = vector_s [nstlb, jj * 4 + nstr];
if z> = 8 then begin B4 [1]: = 1; z: = z-8; end;
if z> = 4 then begin B4 [1]: = 1; z: = z-4; end;
if z> = 2 then begin B4 [1]: = 1; z: = z-2; end;
if z> = 1 then B4 [1]: = 1;
end {S};
begin {functionf}
E (R32_local, R48_local);
XOR48 (R48_local, k48_local, Rk48_local);
for j: = 1 to 8 do begin {for j}
```

```
for i: = 1 to 6 do b6 [i]: = Rk48_local [i + (j-1) * 6];
s (b6, j, b4);
for i: = 1 to 4 do Rout_local [i + (j-1) * 4]: = b4 [i];
end; {For j}
for i: = 1 to 32 do
bm3 [i]: = Rout_local [Vector_P [i]];
Rout_local: = bm3;
end{ functionf};
```

Контрольні запитання

1. Накресліть структуру алгоритму шифрування і розшифрування DES.
2. Принцип обчислення функції шифрування F (функції розширення, перетворення і перестановки).
3. Алгоритм обчислення ключів.
4. Зв'язок матриць перестановки IP і IP^{-1}. Варіанти матриць.
5. Режим роботи алгоритму DES «Електронна кодова книга».
6. Режим CBC алгоритму DES.
7. Режим CFB алгоритму DES.
8. Режим OBC алгоритму DES.

Варіанти завдань

Реалізувати алгоритм DES (шифрування і розшифрування):

- 1, 2. Робочий режим EBC;
- 3, 4. Робочий режим CBC;
- 5, 6. Робочий режим CFB;
- 7, 8. Робочий режим OBC.

Лабораторна робота № 6
ШИФРУВАННЯ МЕТОДОМ ВЕРНАМА

Мета роботи: вивчення шифрування методом Вернама; набуття навичок розроблення кодерів та декодерів для шифрування й дешифрування послідовностей символів методом Вернама.

Теоретичні відомості

Історична довідка

Система Вернама є окремим випадком системи підстановок Віженера при m = 2. Конкретна версія цього шифру запропонована 1926 р. Гілбертом Вернамом. Старовинний телетайп фірми AT&T зі зчитувальними пристроями Вернама і обладнанням для шифрування використовувався корпусом зв'язку армії США.

Кожна літера вхідного тексту в алфавіті, розширеному деякими додатковими знаками, спочатку переводилася з використанням телеграфного коду Бодо в п’ятибітовий блок (b_0, b_1, ..., b_4). До початкового тексту Бодо додавався (за модулем 2) ключ $k = (k_0, k_1, ..., k_{к-1})$, записаний на паперовій стрічці.

Термінологія

Як інформація, що підлягає шифруванню і дешифруванню, будуть розглядатися тексти, побудовані на деякому алфавіті. Під цими термінами розуміється таке.

Алфавіт – кінцева множина знаків, які використовуються для кодування інформації.

Текст – упорядкований набір з елементів алфавіту.

Можна навести такі приклади алфавітів, що використовуються в сучасних ІС:

- алфавіт Z_{33} – 32 літери російського алфавіту і пробіл;
- алфавіт Z_{256} – символи, що входять до стандартних кодів ASCII і КОИ-8;
- бінарний алфавіт – $Z_2 = \{0, 1\}$;
- вісімковий алфавіт або шістнадцятковий алфавіт.

Шифрування – перетворювальний процес: вхідний текст, який має також назву відкритого тексту, замінюється шифрованим текстом.

Дешифрування – зворотний процес шифрування. На основі ключа шифрований текст перетворюється на вхідний.

Ключ – інформація, необхідна для безперешкодного шифрування і дешифрування текстів.

Моделювання кодера і декодера

Схему передачі повідомлень з використанням шифрування методом Вернама показано на рис. 6.1.

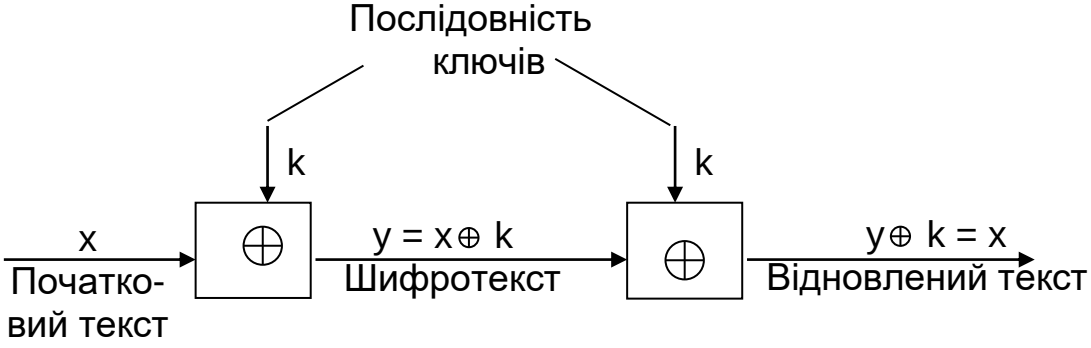


Рис. 6.1

Шифрування вхідного тексту, попередньо перетвореного на послідовність двійкових символів ***x***, здійснюється шляхом складання за модулем 2 символів ***x*** з послідовністю двійкових ключів ***k***. Отримуємо символи шифротексту:

$$y = x \oplus k.$$

Розшифрування полягає в додаванні за модулем 2 символів ***y*** шифротексту з тією самою послідовністю ключів ***k***:

$$y \oplus k = x \oplus k \oplus k = x.$$

При цьому послідовності ключів, використані при шифруванні та розшифруванні, компенсують одна одну (при додаванні за модулем 2), і в результаті відновлюються символи ***x*** вхідного тексту.

Під час моделювання перетворити вхідний текст на послідовність двійкових символів можна таким чином:

```
for(i=0;i<n*sizeof(char)*8;i++)
    m[i]=(((str[i/(sizeof(char)*8)])&(int)pow(2,i%(sizeof(char)*8)))>>i);
```

де ***n*** – кількість елементів у тексті;
m – масив двійкових символів;
8 – кількість бітів у байті;
Str – рядок, що містить текст.
Кодер може мати такий вигляд:

```
for(i=0,j=0;m[i+1] != '\0'; i++, j<M ? j++ :j=0) m[i]^=k[j],
```

де ***k*** – послідовність ключів;

M – довжина послідовності ключів.

Декодер повністю збігається з кодером.

Під час розроблення Вернам перевіряв систему за допомогою закільцьованих стрічок, установлених на передавачі й приймачі для того, щоб використовувалася та сама послідовність ключів.

Слід зазначити, що метод Вернама не залежить від довжини послідовності ключів і, крім того, дає змогу використовувати випадкову послідовність ключів. Однак під час реалізації методу Вернама виникають серйозні проблеми, пов'язані з доставкою одержувачу такої самої послідовності ключів, як і у відправника, або з необхідністю безпечного зберігання ідентичних послідовностей ключів у відправника й одержувача.

Послідовність ключів можна реалізувати як послідовність псевдовипадкових чисел. У цьому випадку слід синхронізувати тільки параметри псевдовипадкового генератора. Прикладом такого датчика може бути адитивний:

$$y_{n+1} = (y_n + y_{n-1}) \bmod m,$$

де ***mod*** – залишок від ділення.

Установки сенсора можна взяти такими: $m = 4096 \cdot 4$; $y_1 = 4091$; $y_2 = m - 5$.

Контрольні запитання

1. Що таке шифрування, дешифрування?
2. Що таке алфавіт, текст?
3. Що таке ключ?
4. У чому полягає шифрування методом Вернама?
5. Яким чином можна реалізувати нескінченну послідовність ключів?
6. У якому вигляді має бути текст перед безпосереднім шифруванням?

Варіанти завдань

1. Змоделювати кодер з послідовністю ключів з п'яти елементів.
2. Змоделювати декодер з послідовністю ключів з п'яти елементів.
3. Змоделювати кодер з послідовністю ключів з 17-ти елементів.
4. Змоделювати декодер з послідовністю ключів з 17-ти елементів.

5. Змоделювати кодер з нескінченною послідовністю ключів, що формуються за допомогою адитивного псевдовипадкового датчика випадкових чисел.

6. Змоделювати декодер з нескінченною послідовністю ключів, що формуються за допомогою адитивного псевдовипадкового датчика випадкових чисел.

Лабораторна робота № 7
КОМБІНУВАННЯ БЛОКОВИХ АЛГОРИТМІВ

Мета роботи: вивчення можливості комбінування блокових алгоритмів.

Теоретичні відомості

Спочатку стандарт DES призначався для шифрування і розшифрування даних ЕОМ. Однак його застосування було узагальнено і на аутентифікацію.

У системах автоматичного оброблення даних людина не в змозі переглянути дані, щоб установити, чи не внесені до них будь-які зміни. При величезних обсягах даних, що проходять у сучасних системах оброблення, перегляд зайняв би надто багато часу. До того ж надлишковість даних може виявитися недостатньою для виявлення помилок. Навіть у тих випадках, коли перегляд людиною є можливим, дані можуть бути змінені таким чином, що виявити ці зміни дуже важко. Наприклад, «do» може бути замінено на «do not», «$ 1900.» – на «$ 9100». Без додаткової інформації людина під час перегляду може легко сприйняти змінені дані за справжні. Такі ризики можуть існувати навіть при використанні шифрованих даних. Тому бажано мати автоматичний спосіб виявлення змін даних.

Звичайні коди, що виявляють помилки, непридатні, адже якщо алгоритм утворення коду відомий, то супротивник може виробити правильний код після внесення змін до даних. Однак за допомогою алгоритму DES дані можна захистити як від випадкових, так і від навмисних, але несанкціонованих змін.

Цей процес описує стандарт для аутентифікації ЕОМ. Суть стандарту полягає в тому, що дані зашифровуються в режимі зворотного зв'язку за шифротекстом або в режимі зчеплення блоків шифру, унаслідок чого виходить остаточний блок шифру, який є функцією всіх розрядів відкритого тексту. Після цього повідомлення, яке містить відкритий текст, може бути передано з використанням обчисленого остаточного блока шифру, що є криптографічною контрольною сумою.

Ті самі дані можна захистити, користуючись як шифруванням, так і аутентифікацією. Дані захищаються від ознайомлення шифруванням, а зміни виявляються за допомогою аутентифікації. Алгоритм аутентифікації можна застосувати як до відкритого, так і до зашифрованого текстів. При фінансових операціях, коли в більшості випадків реалізується і шифрування, і аутентифікація, остання застосовується і до відкритого тексту.

Шифрування й аутентифікацію використовують для захисту даних, що зберігаються в ЕОМ. У багатьох ЕОМ паролі зашифровують у незворотному порядку і зберігають у пам'яті машини. Коли користувач

звертається до ЕОМ і вводить пароль, цей пароль зашифровується і порівнюється зі збереженим значенням. Якщо обидві зашифровані величини однакові, користувач отримує доступ до машини, в іншому випадку – отримує відмову.

Нерідко зашифрований пароль створюється за допомогою алгоритму DES, причому ключ вважається рівним паролю, а відкритий текст – коду ідентифікації користувача.

За допомогою алгоритму DES можна також зашифрувати файли ЕОМ для їх зберігання.

Одним з найбільш важливих випадків застосування алгоритму DES є захист повідомлень електронної системи платежів під час операцій з широкою клієнтурою і між банками.

Алгоритм DES реалізується в банківських автоматах, терміналах торгових точок, на автоматизованих робочих місцях і в головних ЕОМ. Діапазон даних, які вони захищають, досить широкий – від оплат вартістю 50 доларів до переказів на багато мільйонів доларів. Гнучкість основного алгоритму DES дає змогу використовувати його в найрізноманітніших областях застосування електронної системи платежів.

Комбінування блокових алгоритмів

Зараз блоковий алгоритм DES вважається відносно безпечним алгоритмом шифрування. Він піддавався ретельному криптоаналізу протягом 20 років. Найбільш практичним способом його злому є метод перебору всіх можливих варіантів ключа. Ключ DES має довжину 56 бітів, тому існує 2^{56} можливих варіантів такого ключа. Якщо припустити, що суперкомп'ютер може випробувати мільйон варіантів ключа за секунду, то буде потрібно 2285 років для знаходження правильного ключа. Якби ключ мав довжину 128 бітів, то потрібно було б 10^{25} років (для порівняння: вік Всесвіту – близько 10^{10} років).

Неважко уявити собі, що при постійному прогресі можливостей комп'ютерної техніки недалеко той час, коли машини пошуку ключа DES методом повного перебору стануть економічними для потужних у фінансовому відношенні державних і комерційних організацій.

Виникає питання: чи не можна використовувати DES як блок для створення алгоритму з більш довгим ключем?

У принципі існує багато способів комбінування блокових алгоритмів для отримання нових алгоритмів. Одним з таких способів комбінування є багаторазове шифрування, тобто використання блокового алгоритму кілька разів з різними ключами для шифрування того самого блока відкритого тексту. Дворазове шифрування блока відкритого тексту тим самим ключем не дає позитивного результату. При використанні того

самого алгоритму таке шифрування не впливає на складність криптоаналітичної атаки повного перебору.

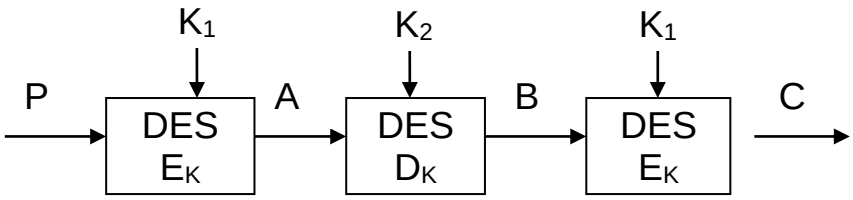
Розглянемо ефективність дворазового шифрування блока відкритого тексту за допомогою двох різних ключів. Спочатку шифрують блок ***P*** ключем K_1, а потім створений шифротекст $E_{K1}(P)$ – ключем K_2. Унаслідок дворазового шифрування отримують криптограму $C = E_{K2}(E_{K1}(P))$. Розшифрування є зворотним процесом: $P = D_{K1}(D_{K2}(C))$.

Якщо блоковий алгоритм має властивості групи, то завжди знайдеться такий ключ K_3, що $C = E_{K2}(E_{K1}(P)) = E_{K3}(P)$.

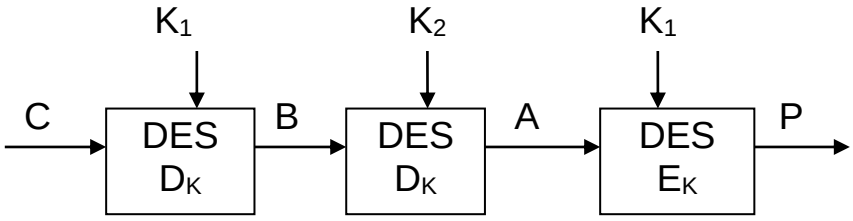
Якщо ж блоковий алгоритм не є групою, то результівний дворазово шифрований блок тексту виявиться набагато складнішим для злому методом повного перебору варіантів. Замість 2^n спроб, де n – довжина ключа в бітах, потрібно 2^{2n} спроб. Зокрема, якщо $n = 64$, то дворазово зашифрований блок тексту потребуватиме 2^{128} спроб для знаходження ключа.

Однак Р. Меркль і М. Хеллман показали на прикладі DES, що, використовуючи метод «обміну часу на пам'ять» і криптоаналітичну атаку, можна зламати таку схему дворазового шифрування за 2^{n+1} спроб, хоча ця атака потребує дуже великого обсягу пам'яті.

Більш привабливу ідею запропонував У. Тачмен. Суть цієї ідеї полягає в тому, щоб шифрувати блок відкритого тексту ***P*** три рази за допомогою двох ключів K_1 і K_2 (рис. 7.1).



Шифрування



Розшифрування

Рис. 7.1

Процедура шифрування:

$$C = E_{K1}(D_{K2}(E_{K1}(P))),$$

тобто блок відкритого тексту ***P*** спочатку шифрується ключем K_1, потім розшифровується ключем K_2 і остаточно шифрується ключем K_1.

Цей режим іноді називають режимом EDE (encrypt-decrypt-encrypt). Введення в таку схему операції розшифрування D_{K2} дає змогу забезпечити її сумісність зі схемою одноразового використання алгоритму DES. Якщо в схемі триразового використання DES вибрати всі ключі однаковими, то ця схема перетвориться на схему одноразового використання DES.

Процедура розшифрування виконується в зворотному порядку:

$$P = D_{K1}(E_{K2}(D_{K1}(C))),$$

тобто блок шифротексту ***C*** спочатку розшифровується ключем K_1, а потім шифрується ключем K_2 і остаточно розшифровується ключем K_1.

Якщо вхідний блоковий алгоритм має ***n***-бітовий ключ, то схема триразового шифрування має ***2n***-бітовий ключ. Чергування ключів K_1 і K_2 дає змогу запобігти криптоаналітичній атаці «зустріч посередині». Ця схема наводиться в стандартах X9.17 і ISO 8732 як засіб поліпшення характеристик алгоритму DES.

При триразовому шифруванні можна застосувати три різних ключі. При цьому зростає загальна довжина результівного ключа. Процедури шифрування і розшифровування описуються виразами

$$C = E_{K3}(D_{K2}(E_{K1}(P))),$$
$$P = D_{K1}(E_{K2}(D_{K3}(C))).$$

Триключевий варіант має ще більшу стійкість. Очевидно, що якщо потрібно підвищити безпеку великого парку обладнання, що використовує DES, то набагато дешевше переключитися на схеми триразових DES, ніж переходити на інший тип криптосхем.

Варіанти завдань

1. Провести дворазове шифрування блока відкритого тексту за допомогою двох різних ключів, використовуючи блоковий алгоритм DES у режимі «Електронна кодова книга».
2. Провести дворазове шифрування блока відкритого тексту за допомогою двох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зчеплення блоків шифру».
3. Провести дворазове шифрування блока відкритого тексту за допомогою двох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зворотний зв'язок за шифром».

4. Провести дворазове шифрування блока відкритого тексту за допомогою двох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зворотний зв'язок за виходом».

5. Провести триразове шифрування блока відкритого тексту за допомогою трьох різних ключів, використовуючи блоковий алгоритм DES у режимі «Електронна кодова книга».

6. Провести триразове шифрування блока відкритого тексту за допомогою трьох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зчеплення блоків шифру».

7. Провести триразове шифрування блока відкритого тексту за допомогою трьох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зворотний зв'язок за шифром».

8. Провести триразове шифрування блока відкритого тексту за допомогою трьох різних ключів, використовуючи блоковий алгоритм DES у режимі «Зворотний зв'язок за виходом».

Лабораторна робота № 8
АЛГОРИТМ ШИФРУВАННЯ ДАНИХ IDEA

Мета роботи: ознайомлення з алгоритмом шифрування даних IDEA; набуття досвіду програмної реалізації складових алгоритму IDEA.

Теоретичні відомості

На думку К. Шеннона, в практичних шифрах необхідно використовувати два спільних принципи: розсіювання та перемішування.

Розсіювання – це поширення впливу одного знака відкритого тексту на багато знаків шифротексту, що дає змогу приховати статистичні властивості відкритого тексту.

Перемішування передбачає використання таких шифрувальних перетворень, які ускладнюють відновлення взаємозв'язку статистичних властивостей відкритого і шифрованого текстів. Однак шифр повинен не тільки ускладнювати розкриття, а й забезпечувати легкість шифрування і розшифрування за умови, що користувачеві відомий секретний ключ.

Поширеним способом досягнення ефектів розсіювання та перемішування є використання складного шифру, тобто такого шифру, який може бути реалізований у вигляді деякої послідовності простих шифрів, кожен з яких посилює сумарне розсіювання і перемішування.

У складених шифрах як прості шифри найчастіше використовуються прості перестановки і підстановки. Під час перестановки просто перемішують символи відкритого тексту, причому конкретний вид перемішування визначається секретним ключем. При підстановці кожен символ відкритого тексту замінюють іншим символом з того ж алфавіту, а конкретний вид підстановки також визначається секретним ключем. Слід зауважити, що в сучасному блоковому шифрі блоки відкритого тексту і шифротексту являють собою двійкові послідовності зазвичай довжиною 64 біти. В принципі кожен блок може набувати 2^{64} значень. Тому підстановки виконуються в дуже великому алфавіті, що містить до $2^{64} \approx 10^{19}$ «символів».

При багаторазовому чергуванні простих перестановок і підстановок, керованих досить довгим секретним ключем, можна отримати дуже стійкий шифр з хорошим розсіюванням і перемішуванням. Розглянутий нижче криптоалгоритм IDEA побудований в повній відповідності до вказаної методології.

Алгоритм шифрування даних IDEA

Алгоритм IDEA (International Data Encryption Algorithm) є блоковим шифром, який оперує 64-бітовими блоками відкритого тексту. Безсумнівною перевагою цього алгоритму є те, що його ключ має довжину 128 бітів. Той самий алгоритм використовується і для шифрування, і для розшифрування.

Перша версія алгоритму IDEA була запропонована 1990 р. Г. Х. Леєм і Дж. Мессі. Перша назва алгоритму – PES (Proposed Encryption Standard). Покращений варіант цього алгоритму, розроблений 1991 р., отримав назву IPES (Improved Proposed Encryption Standard). 1992 року назву IPES було змінено на IDEA. Як і більшість інших блочних шифрів, алгоритм IDEA використовує під час шифрування процеси змішування і розсіювання, причому ці процеси легко реалізуються апаратними та програмними засобами.

В алгоритмі IDEA використовуються такі математичні операції:

- порозрядне складання за модулем 2 (операція «виключне АБО»); операція позначається як «$\oplus$»;
- складання беззнакових цілих за модулем 2 (модуль 65536); операція позначається як «+»;
- множення цілих за модулем ($2^{16}+1$) (модуль 65537), що розглядаються як беззнакові цілі, за винятком того, що блок з 16 нулів розглядається як 2^{16}; операція позначається як «*».

Усі операції виконуються над 16-бітовими субблоками. Ці три операції несумісні, оскільки:

- жодна пара з цих трьох операцій не задовольняє асоціативному закону, наприклад: $\boldsymbol{a + (b \oplus c) \neq (a + b) \oplus c}$;
- жодна пара з цих трьох операцій не задовольняє дистрибутивному закону, наприклад: $\boldsymbol{a + (b \cdot c) \neq (a + b) \cdot (a + c)}$.

Комбінування цих трьох операцій забезпечує комплексне перетворення входу, істотно ускладнюючи криптоаналіз IDEA порівняно з DES, який базується виключно на операції «виключне АБО».

Загальну схему алгоритму IDEA показано на рис. 8.1.

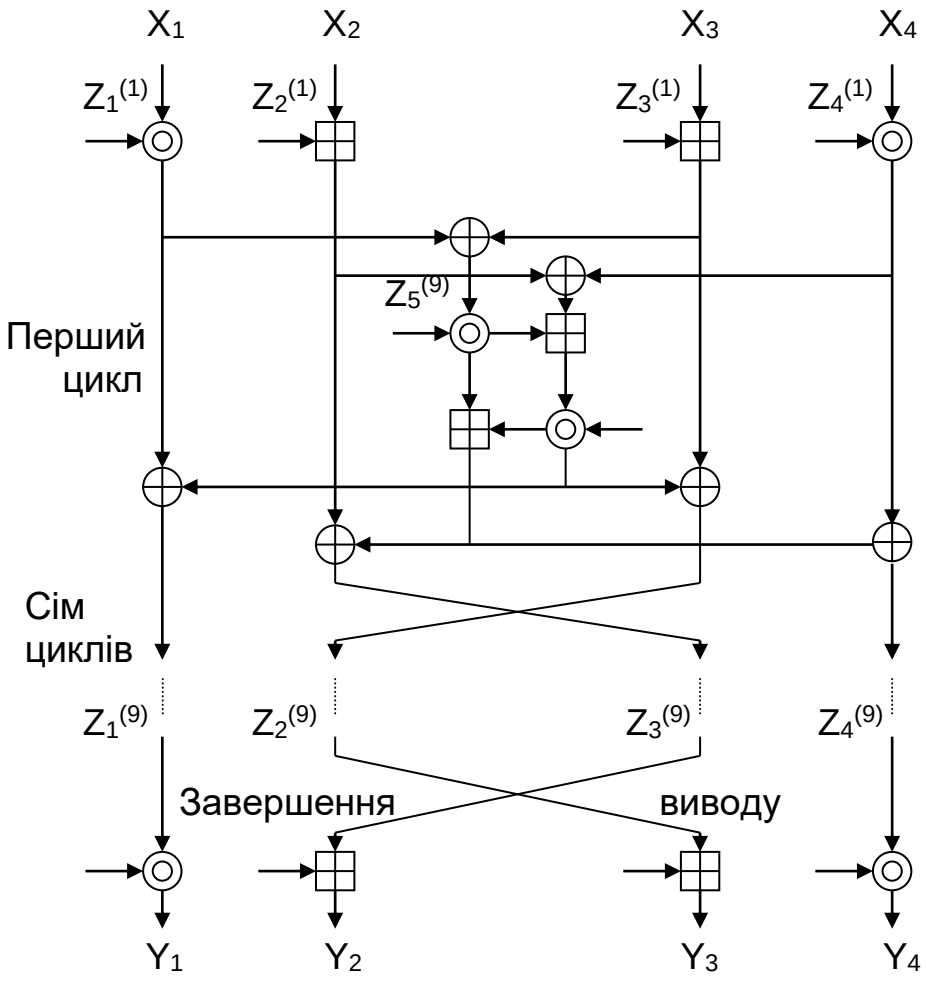


Рис. 8.1

Позначення:

Xi – 16-бітовий субблок відкритого тексту, ***i = 1 ... 4***;

Yi – 16-бітовий субблок шифротексту, ***i = 1 ... 4***;

Zj (r) – 16-бітове з'єднання (субблок ключа), ***j = 1 ... 6***, ***r = 1 ... 8***;

$\oplus$ – порозрядне підсумовування за модулем 2 16-бітових субблоків;

+ – додавання за модулем 2 16-бітових цілих;

(*) – множення за модулем 2 16-бітових цілих (з нульовим субблоком, відповідним 2^{16}).

64-бітовий блок даних ділиться на чотири 16-бітових субблоки. Ці чотири субблоки стають входом у перший цикл алгоритму. Всього виконується вісім циклів. Між циклами другий і третій субблоки міняються місцями. У кожному циклі має місце така послідовність операцій:

(1) (*) – множення субблока X_1 і першого підключа;

(2) + – додавання субблока X_2 і другого підключа;

(3) + – додавання субблока X_3 і третього підключа;

(4) (*) – множення субблока X_4 і четвертого підключа;
(5) $\oplus$ – складання результатів кроків (1) і (3);
(6) $\oplus$ – складання результатів кроків (2) і (4);
(7) (*) – множення результату кроку (5) і п'ятого підключа;
(8) + – додавання результатів кроків (6) і (7);
(9) (*) – множення результату кроку (8) з шостим підключем;
(10) + – додавання результатів кроків (7) і (9);
(11) $\oplus$ – складання результатів кроків (1) і (9);
(12) $\oplus$ – складання результатів кроків (3) і (9);
(13) $\oplus$ – складання результатів кроків (2) і (10);
(14) $\oplus$ – складання результатів кроків (4) і (10).

Виходом циклу є чотири субблоки, які отримують як результати виконання кроків (11), (12), (13) і (14). На завершення циклу переставляють місцями два внутрішніх субблоки (за винятком останнього циклу). В результаті формується вхід для наступного циклу.

Після восьмого циклу здійснюють заключне перетворення виходу:

(1) (*) – множення субблока X_1 і першого підключа;
(2) + – додавання субблока X_2 і другого підключа;
(3) + – додавання субблока X_3 і третього підключа;
(4) (*) – множення субблока X_4 і четвертого підключа.

Нарешті, ці результівні чотири субблоки $Y_1 \ldots Y_4$ знову об'єднують для одержання блока шифротексту.

Створення підключів Z_j також відносно нескладно. Алгоритм використовує всього 52 підключі (по шість для кожного з восьми циклів і ще чотири для перетворення виходу). Спочатку 128-бітовий ключ ділять на вісім 16-бітових підключів. Це – перші вісім підключів для алгоритму (шість підключів – для першого циклу і перші два підключі – для другого циклу). Потім 128-бітовий ключ циклічно зсувається вліво на 25 бітів і знову ділиться на вісім підключів. Перші чотири з них використовують у другому циклі, останні чотири – в третьому. Ключ знову циклічно зсувається вліво ще на 25 бітів для отримання наступних восьми підключів і так далі, поки виконання алгоритму не завершиться.

Розшифрування здійснюють аналогічним чином, за винятком того, що порядок використання підключів стає зворотним, причому ряд значень підключів замінюється на зворотні значення. Підключі розшифрування є в основному або адитивними, або мультиплікативними зворотними величинами підключів шифрування (табл. 8.1).

Таблиця 8.1

Цикл	Підключі шифрування	Підключі розшифрування
1	$Z_1^{(1)}$ $Z_2^{(1)}$ $Z_3^{(1)}$ $Z_4^{(1)}$	$Z_1^{(9)-1}$ $-Z_2^{(9)}$ $-Z_3^{(9)}$ $Z_4^{(9)-1}$
2	$Z_5^{(1)}$ $Z_6^{(1)}$	$Z_5^{(8)}$ $Z_6^{(8)}$
3	$Z_1^{(2)}$ $Z_2^{(2)}$ $Z_3^{(2)}$ $Z_4^{(2)}$	$Z_1^{(8)-1}$ $-Z_2^{(8)}$ $-Z_3^{(8)}$ $Z_4^{(8)-1}$
4	$Z_5^{(2)}$ $Z_6^{(2)}$	$Z_5^{(8)}$ $Z_6^{(8)}$
5	$Z_1^{(3)}$ $Z_2^{(3)}$ $Z_3^{(3)}$ $Z_4^{(3)}$	$Z_1^{(7)-1}$ $-Z_2^{(7)}$ $-Z_3^{(7)}$ $Z_4^{(7)-1}$
6	$Z_5^{(3)}$ $Z_6^{(3)}$	$Z_5^{(7)}$ $Z_6^{(7)}$
7	$Z_1^{(4)}$ $Z_2^{(4)}$ $Z_3^{(4)}$ $Z_4^{(4)}$	$Z_1^{(6)-1}$ $-Z_2^{(6)}$ $-Z_3^{(6)}$ $Z_4^{(6)-1}$
8	$Z_5^{(4)}$ $Z_6^{(4)}$	$Z_5^{(6)}$ $Z_6^{(6)}$
Перетворення виходу	$Z_1^{(5)}$ $Z_2^{(5)}$ $Z_3^{(5)}$ $Z_4^{(5)}$	$Z_1^{(5)-1}$ $-Z_2^{(5)}$ $-Z_3^{(5)}$ $Z_4^{(5)-1}$
	$Z_5^{(5)}$ $Z_6^{(5)}$	$Z_5^{(5)}$ $Z_6^{(5)}$
	$Z_1^{(6)}$ $Z_2^{(6)}$ $Z_3^{(6)}$ $Z_4^{(6)}$	$Z_1^{(4)-1}$ $-Z_2^{(4)}$ $-Z_3^{(4)}$ $Z_4^{(4)-1}$
	$Z_5^{(6)}$ $Z_6^{(6)}$	$Z_5^{(4)}$ $Z_6^{(4)}$
	$Z_1^{(7)}$ $Z_2^{(7)}$ $Z_3^{(7)}$ $Z_4^{(7)}$	$Z_1^{(3)-1}$ $-Z_2^{(3)}$ $-Z_3^{(3)}$ $Z_4^{(3)-1}$
	$Z_5^{(7)}$ $Z_6^{(7)}$	$Z_5^{(3)}$ $Z_6^{(3)}$
	$Z_1^{(8)}$ $Z_2^{(8)}$ $Z_3^{(8)}$ $Z_4^{(8)}$	$Z_1^{(2)-1}$ $-Z_2^{(2)}$ $-Z_3^{(2)}$ $Z_4^{(2)-1}$
	$Z_5^{(8)}$ $Z_6^{(8)}$	$Z_5^{(2)}$ $Z_6^{(2)}$
	$Z_1^{(9)}$ $Z_2^{(9)}$ $Z_3^{(9)}$ $Z_4^{(9)}$	$Z_1^{(1)-1}$ $-Z_2^{(1)}$ $-Z_3^{(1)}$ $Z_4^{(1)-1}$

Для реалізації алгоритму IDEA було прийнято припущення, що нульовий субблок становить $2_{16} = -1$; при цьому мультиплікативна зворотна величина від 0 дорівнює нулю. Обчислення значень мультиплікативних зворотних величин потребує деяких витрат, але це доводиться робити тільки один раз для кожного ключа розшифрування.

Алгоритм IDEA може працювати в будь-якому режимі блокового шифру, передбаченому для алгоритму DES. Алгоритм IDEA має низку переваг перед алгоритмом DES. Він значно безпечніше, оскільки 128-бітовий ключ алгоритму IDEA вдвічі більше ключа DES. Внутрішня структура алгоритму IDEA забезпечує кращу стійкість до криптоаналізу. Існуючі програмні реалізації алгоритму IDEA приблизно вдвічі швидше реалізації алгоритму DES. Алгоритм IDEA шифрує дані на IBM PC/486 зі швидкістю 2,4 Мбіт/с. Реалізація IDEA на НВІС шифрує дані зі швидкістю 177 Мбіт/с при частоті 25 МГц. Алгоритм IDEA запатентований у Європі і США.

Варіанти завдань

1. Реалізувати програмним шляхом операції 1–4 циклу алгоритму IDEA в режимі шифрування. Оформити програму у вигляді процедур і функцій.
2. Реалізувати програмним шляхом операції 5–9 циклу алгоритму IDEA в режимі шифрування. Оформити програму у вигляді процедур і функцій.

3. Реалізувати програмним шляхом операції 10–14 циклу алгоритму IDEA в режимі шифрування. Оформити програму у вигляді процедур і функцій.

4. Реалізувати програмним шляхом заключне перетворення виходу алгоритму IDEA в режимі шифрування. Оформити програму у вигляді процедур і функцій.

5. Використовуючи програми завдань 1–4, змоделювати програмним шляхом алгоритм IDEA в режимі шифрування.

6. Реалізувати програмним шляхом операції 1–4 циклу алгоритму IDEA в режимі розшифрування. Оформити програму у вигляді процедур і функцій.

7. Реалізувати програмним шляхом операції 5–9 циклу алгоритму IDEA в режимі розшифрування. Оформити програму у вигляді процедур і функцій.

8. Реалізувати програмним шляхом операції 10–14 циклу алгоритму IDEA в режимі розшифрування. Оформити програму у вигляді процедур і функцій.

9. Реалізувати програмним шляхом заключне перетворення виходу алгоритму IDEA в режимі розшифрування. Оформити програму у вигляді процедур і функцій.

10. Використовуючи програми завдань 6–9, змоделювати програмним шляхом алгоритм IDEA в режимі розшифрування.

Лабораторна робота № 9
СИСТЕМИ З ВІДКРИТИМ КЛЮЧЕМ. АЛГОРИТМ RSA

Мета роботи: ознайомлення з криптографічними системами з відкритим ключем; вивчення алгоритму кодування RSA.

Теоретичні відомості

Системи з відкритим ключем (СВК)

Якими б не були складними і надійними криптографічні системи, їх слабке місце при практичній реалізації – проблема розподілу ключів. Для того, щоб був можливий обмін конфіденційною інформацією між двома суб'єктами ІС, ключ повинен бути згенерований одним з них, а потім якимось чином знову ж у конфіденційному порядку переданий іншому, тобто в загальному випадку для передачі ключа знову потрібно використання якоїсь криптосистеми.

Для вирішення цієї проблеми на основі результатів, отриманих класичною та сучасною алгеброю, були запропоновані системи з відкритим ключем.

Суть їх полягає в тому, що кожний адресат ІС генерує два ключі, пов'язані між собою за певним правилом. Один ключ оголошується відкритим, а другий – закритим. Відкритий ключ публікується і доступний кожному, хто бажає послати повідомлення адресату. Секретний ключ зберігається в таємниці.

Вхідний текст шифрується відкритим ключем адресата і передається йому. Зашифрований текст у принципі не може бути розшифрований тим самим відкритим ключем. Дешифрування повідомлення можливо лише з використанням закритого ключа, який відомий тільки самому адресату.

Криптографічні системи з відкритим ключем використовують так звані необернені, або односторонні, функції, що мають таку властивість: при заданому значенні x відносно легко обчислити значення $f(x)$, однак якщо $y = f(x)$, то немає простого шляху для обчислення значення x.

Множина класів необернених функцій породжує все розмаїття систем з відкритим ключем, проте не кожна необоротна функція придатна для використання в реальних ІС.

У самому визначенні необоротності є невизначеність. Під необоротністю розуміється не теоретична необоротність, а практична неможливість обчислити зворотне значення, використовуючи сучасні обчислювальні засоби за доступний для огляду інтервал часу.

Тому, щоб гарантувати надійний захист інформації, до систем з відкритим ключем (СВК) висувають дві важливі та очевидні вимоги:

– перетворення вхідного тексту має бути незворотним і виключати його відновлення на основі відкритого ключа;

– визначення закритого ключа на основі відкритого також має бути неможливим на сучасному технологічному рівні, при цьому бажана точна нижня оцінка складності (кількості операцій) розкриття шифру.

Алгоритми шифрування з відкритим ключем набули значного поширення в сучасних інформаційних системах. Так, алгоритм RSA став світовим стандартом де-факто для відкритих систем і рекомендований МККТТ.

Взагалі ж усі пропоновані сьогодні криптосистеми з відкритим ключем спираються на один з таких типів необоротних перетворень:

- розкладання великих чисел на прості множники;
- обчислення логарифму в кінцевому полі;
- обчислення коренів алгебричних рівнянь.

Тут же слід зазначити, що алгоритми криптосистеми з відкритим ключем (СВК) можна використовувати за такими призначеннями:

1. Як самостійні засоби захисту переданих і збережених даних.

2. Як засоби для розподілу ключів. Алгоритми СВК більш трудомісткі, ніж традиційні криптосистеми. Тому часто на практиці раціонально за допомогою СВК розподіляти ключі, обсяг яких як інформації незначний, а потім за допомогою звичайних алгоритмів здійснювати обмін великими інформаційними потоками.

3. Як засоби аутентифікації користувачів.

Розглянемо найбільш поширену систему з відкритим ключем.

Алгоритм RSA

Незважаючи на досить велику кількість різних СВК, найбільш популярною є криптосистема RSA, що була розроблена 1977 року і отримала назву на честь її творців: Рональді Ривеста (Rivest), Аді Шаміра (Shamir) і Леонарда Ейдельмана (Adleman).

Вони скористалися тим фактом, що знаходження великих простих чисел в обчислювальному відношенні здійснюється легко, але розкладання на множники добутку двох таких чисел практично нездійсненно. Доведено (теорема Рабіна), що розкриття шифру RSA еквівалентно такому розкладанню. Тому для будь-якої довжини ключа можна дати нижню оцінку кількості операцій для розкриття шифру, а з урахуванням продуктивності сучасних комп'ютерів оцінити і необхідний на це час.

Можливість гарантовано оцінити захищеність алгоритму RSA стала однією з причин популярності цієї СВК порівняно з десятками інших схем. Тому алгоритм RSA використовується в банківських комп'ютерних мережах, особливо для роботи з віддаленими клієнтами (обслуговування кредитних карток).

Зараз алгоритм RSA використовується в багатьох стандартах, серед яких SSL, S-HHTP, S-MIME, S/WAN, STT і PCT.

Розглянемо математичні результати, покладені в основу цього алгоритму.

Теорема 1 *(мала теорема Ферма).*

Якщо ***p*** – просте число, то

$$x^{p-1} = 1 \pmod p \qquad (9.1)$$

для будь-якого значення ***x***, простого щодо ***p***, та

$$x^p = x \pmod p \qquad (9.2)$$

для будь-якого значення ***x***.

Доведення. Досить довести справедливість рівнянь (9.1) і (9.2) для $x \in Z$. Проведемо доведення методом індукції.

Очевидно, що рівняння (9.2) виконується при $x = 0$ і $x = 1$. Далі $x^p = (x - 1 + 1)^p = \sum C(p, j)(x - 1)^j = (x - 1)^p + 1 \pmod p,\ 0 \le j \le p,$ оскільки $C(p, j) = 0 \pmod p$ при $0 < j < p$. З урахуванням цієї нерівності і пропозицій методу індукції теорему доведено.

Визначення. Функцією Ейлера $\varphi(N)$ називається кількість позитивних цілих, менших ***n*** і простих відносно ***n***.

n	2	3	4	5	6	7	8	9	10	11	12
φ(n)	1	2	2	3	2	6	4	6	4	10	4

Теорема 2. Якщо $n = pq$ (***p*** і ***q*** – відмінні одне від одного прості числа), то

$$\varphi(n) = (p - 1)(q - 1).$$

Теорема 3. Якщо $n = pq$ (***p*** і ***q*** – відмінні одне від одного прості числа) і ***x*** – просте число щодо ***p*** і ***q***, то

$$x^{\varphi(n)} = 1 \pmod n.$$

Наслідок. Якщо $n = pq$ (***p*** і ***q*** – відмінні одне від одного прості числа) і ***e*** – просте число щодо $\varphi(n)$, то відображення

$$E_{e,n}: x \to x^e \pmod n$$

є взаємно однозначним на Z_n.

Очевидним є і той факт, що якщо ***e*** – просте число щодо $\varphi(n)$, то існує ціле число ***d*** таке, що

$$ed = 1 \pmod{\varphi(n)}. \qquad (9.3)$$

На цих математичних фактах і базується популярний алгоритм RSA.

У криптосистемі RSA відкритий ключ ***e***, секретний ключ ***d***, повідомлення ***M*** і криптограма ***C*** належать множині цілих чисел

$$Z_N = \{0, 1, 2, ..., N - 1\},$$

де ***N*** – модуль:

$$N = P \cdot Q.$$

Тут ***P*** та ***Q*** – випадкові великі прості числа.

Як відкритий ключ ***e*** вибирають число, взаємно просте з $\varphi(N)$, так, щоб виконувалася умова

$$1 < e \leq \varphi(N). \qquad (9.4)$$

Далі за допомогою формули (9.3) обчислюється закритий (секретний) ключ ***d***.

Відкритий ключ ***e*** використовують для шифрування даних, закритий ключ ***d*** – для розшифрування.

Шифрування визначає криптограму ***C*** відповідно до виразу

$$C = M^e \pmod N. \qquad (9.5)$$

Обернення цієї функції, тобто визначення значення ***M*** за відомими значеннями ***C***, ***e*** та ***N***, практично нездійсненно при $N \approx 2^{512}$.

Однак зворотне завдання, тобто завдання розшифрування криптограми ***C***, можна вирішити за допомогою формули

$$M = C^d \pmod N. \qquad (9.6)$$

Такий результат випливає з наведених вище теорем і співвідношення (9.3).

{e, N} утворює відкритий ключ, а ***{d, N}*** – закритий (хоча можна взяти і навпаки).

Таким чином, одержувач, який створює криптосистему, захищає два параметри: закритий ключ ***d***, пару чисел ***(P, Q)***, добуток яких дає значення модуля ***N***.

З іншого боку, одержувач відкриває значення модуля ***N*** і відкритий ключ ***e***.

Опис послідовності дій алгоритму RSA

Припустимо, що користувач ***A*** хоче передати користувачеві ***B*** повідомлення в зашифрованому вигляді, використовуючи криптосистему RSA. У цьому випадку користувач ***A*** виступає в ролі відправника повідомлення, а користувач ***B*** – в ролі одержувача. Розглянемо послідовність дій користувача ***B*** і користувача ***A***.

Користувач ***B*** вибирає два довільних великих простих числа ***P*** і ***Q***.

Користувач ***B*** обчислює значення модуля $N = P \cdot Q$.

Користувач ***B*** обчислює функцію Ейлера $\varphi(N)$ і вибирає випадковим чином значення відкритого ключа ***e***, взаємно простого з $\varphi(N)$, з урахуванням умови (9.4).

Користувач ***B*** обчислює значення секретного ключа ***d***, використовуючи формулу (9.3).

Користувач ***B*** пересилає користувачеві ***A*** пару чисел ***(e, N)*** по незахищеному каналу.

Якщо користувач ***A*** хоче передати користувачеві ***B*** повідомлення ***M***, він виконує такі кроки.

1. Користувач ***A*** розбиває вхідний відкритий текст ***M*** на блоки, кожен з яких може бути поданий у вигляді числа

$$M_i = 0, 1, ..., N - 1.$$

2. Користувач ***A*** шифрує текст, поданий у вигляді послідовності чисел M_i, за формулою

$$C_i = M_i^e \pmod N$$

і відправляє криптограму ***{Ci}*** користувачеві ***B***.

3. Користувач ***B*** розшифровує прийняту криптограму ***{Ci}***, використовуючи закритий ключ ***d***, за формулою

$$M_i = C_i^d \pmod N.$$

У результаті отримує послідовність чисел $\{M_i\}$, що являють собою вхідне повідомлення ***M***. Щоб алгоритм RSA мав практичну цінність, необхідно мати можливість без істотних витрат генерувати великі прості числа, вміти оперативно обчислювати значення ключів ***e*** і ***d***.

Приклад виконання шифрування та дешифрування в системі RSA

Зашифруємо повідомлення «***CAB***». Для простоти будемо використовувати маленькі числа (на практиці застосовуються набагато більші).

Виберемо $P = 3$ і $Q = 11$.

Визначимо $N = P \cdot Q = 3 \cdot 11 = 33$.

Знайдемо $\varphi(N) = \varphi(33) = (P - 1)(Q - 1) = 2 \cdot 10 = 20$. Отже, як e виберемо число, взаємно просте з 20, наприклад $e = 7$.

Виберемо число d. Як таке число можна взяти будь-яке число (наприклад, 3), для якого задовольняється співвідношення

$$(7d) \pmod{20} = 1.$$

Уявімо шифроване повідомлення як послідовність цілих чисел за допомогою відображень $A \to 1$, $B \to 2$, $C \to 3$. Тоді повідомлення набуде вигляду $(3, 1, 2)$. Зашифруємо повідомлення за допомогою ключа $\{7, 33\}$:

$$C_1 = (3^7) \pmod{33} = 2187 \pmod{33} = 9;$$
$$C_2 = (1^7) \pmod{33} = 1 \pmod{33} = 1;$$
$$C_3 = (2^7) \pmod{33} = 128 \pmod{33} = 29.$$

Розшифруємо отримане зашифроване повідомлення $(9, 1, 29)$ на основі закритого ключа $\{3, 33\}$:

$$M_1 = (9^3) \pmod{33} = 729 \pmod{33} = 3;$$
$$M_2 = (1^3) \pmod{33} = 1 \pmod{33} = 1;$$
$$M_3 = (29^3) \pmod{33} = 24389 \pmod{33} = 2.$$

Таким чином, відновлено початкове повідомлення: CAB.

Відкритий ключ публікується і доступний кожному, хто бажає послати власнику ключа повідомлення, яке зашифровується зазначеним алгоритмом. Після шифрування повідомлення неможливо розкрити за допомогою відкритого ключа. Власник же закритого ключа легко може розшифрувати прийняте повідомлення.

Практична реалізація RSA

Зараз алгоритм RSA активно реалізується як самостійні криптографічні продукти, а також як вбудовані засоби в популярних додатках.

Важлива проблема практичної реалізації – генерація великих простих чисел. Рішення завдання «у лоб» – генерація випадкового великого числа n (непарного) і перевірка його подільності на множники від 3 аж до $n^{0.5}$. У разі неуспіху слід узяти значення $n + 2$ і т. д.

У принципі як p і q можна використовувати «майже» прості числа, тобто числа, для яких ймовірність того, що вони прості, наближається до одиниці. Але в разі, якщо використано складене число, а не просте, криптостойкість RSA зменшується. Є непогані алгоритми, які дають змогу генерувати «майже» прості числа з рівнем довіри 2^{-100}.

Інша проблема – визначення необхідної довжини ключа.

Для практичної реалізації алгоритмів RSA корисно знати оцінки трудомісткості розкладання простих чисел різної довжини, зроблені Шроппелем (табл. 9.1).

Таблиця 9.1

log10 n	Кількість операцій	Примітки
50	1.4 · 1010	Розкриваємо на суперкомп'ютерах
100	2.3 · 1015	На межі сучасних технологій
200	1.2 · 1023	За межами сучасних технологій
400	2.7 · 1034	Потребує істотних змін у технології
800	1.3 · 1 051	Не розкриваємо

Наприкінці 1995 року вдалося практично реалізувати розкриття шифру RSA для 500-значного ключа. Для цього за допомогою мережі Internet було задіяно 1600 комп'ютерів.

Самі автори RSA рекомендують використовувати такі розміри модуля ***n***:

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

Третій важливий аспект реалізації RSA – обчислювальний. Адже доводиться використовувати апарат довгої арифметики. Якщо використовується ключ довжиною ***k*** бітів, то для відкритого ключа потрібно $O(k^2)$ операцій, для закритого ключа – $O(k^3)$ операцій, а для генерації нових ключів – $O(k^4)$ операцій.

Криптографічний пакет BSAFE 3.0 (RSA DS) на комп'ютері Pentium-90 здійснює шифрування зі швидкістю 21.6 Кбіт/с для 512-бітового ключа і зі швидкістю 7.4 Кбіт/с – для 1024-бітового. Найшвидша апаратна реалізація забезпечує швидкості в 60 разів більше.

Порівняно з тим же алгоритмом DES алгоритм RSA потребує часу в тисячі і десятки тисяч разів більшого.

Шифрування за алгоритмом RSA

Розглянемо ділянку програми, яка шифрує символи з вхідного файлу F_in:

```
Writeln ( 'Криптограма:');
While (not eof (F_in)) do
 Begin
 Read (F_in, m);
 c: = St (byte (c) -48, e) mod n;
```

```
Write (F_out, char (c));
Write ( '-', c);
End;
```

Тут висновок криптограми виводиться і в файл F_out, і на екран. Функція St (a, b) підносить число **a** до степеня **b**. Щоб перейти від цифрового символу до символу з таким кодом (наприклад, замість символу "1" використовувати символ з кодом 1, тобто # 1), віднімаємо з коду цього символу (byte (c)) код символу "0" (byte (0) = 48). Потім для запису в файл знову відновлюємо символ за його кодом (char (c)), а для виведення на екран залишаємо число-код отриманого символу.

Контрольні запитання

1. У чому полягає суть систем з відкритим ключем?
2. За допомогою яких ключів шифрується і розшифровується повідомлення в СВК?
3. Що таке необернені функції? Які типи необоротних перетворень використовуються в RSA?
4. Які головні вимоги ставлять до RSA?
5. У яких випадках можна використовувати алгоритми криптосистем з відкритим ключем?
6. На яких математичних фактах базується алгоритм RSA?
7. Як вибираються числа ***P*** і ***Q*** алгоритму RSA?
8. Які значення творець криптосистеми RSA повідомляє користувачам, а які зберігає в таємниці?
9. Чи можна розшифрувати повідомлення за допомогою відкритого ключа?
10. Як обчислюється значення функції Ейлера? Для чого його використовують в алгоритмі RSA?
11. За допомогою яких формул проводять шифрування і дешифрування повідомлення?
12. Чи зміниться криптограма, якщо числа ***P*** і ***Q*** поміняти місцями?
13. Нехай ***P*** = 3, ***Q*** = 13. Які з чисел 2, 3, 5, 9, 29 можна використовувати як відкритий ключ ***e***? Чому?
14. Для ***P*** = 3, ***Q*** = 5, ***e*** = 3, ***M*** = {3, 2} (***M*** – вхідне повідомлення) покажіть за кроками процес шифрування і дешифрування. Обчисліть закритий ключ ***d*** самостійно.

Варіанти завдань

Скласти програму шифрування повідомлення за допомогою алгоритму RSA і його розшифрування (непарний варіант – шифрування повідомлення, парний – розшифрування). Для подання даних результату піднесення до степеня використовувати тип LongInt. Щоб не виникало помилки переповнення, вхідне повідомлення розглядати як послідовність символів з кодами 0, 1, ..., 9. Оригінал тексту, криптограму і відновлене повідомлення зберігати у файлах. Завдання за варіантами наведено в табл. 9.2.

Таблиця 9.2

Номер варіанта	***P***	***Q***	***e***	***d***	Вхідний алфавіт
1, 2	2	11	3	7	1234567890
3, 4	2	11	7	3	1234567890
5, 6	2	17	3	11	1780
7, 8	3	11	3	7	1267890
9, 10	3	11	7	3	1234567890
11, 12	3	13	5	5	1234567890
13, 14	3	17	11	3	12345670
15, 16	5	7	5	5	1234567890
17, 18	2	5	3	7	1234567890
19, 20	2	7	5	5	1234567890

Лабораторна робота № 10
СХЕМА ШИФРУВАННЯ ПОЛІГА – ХЕЛЛМАНА

Мета роботи: ознайомлення з асиметричними криптографічними системами; вивчення алгоритму кодування Поліга – Хеллмана.

Теоретичні відомості

Асиметричні системи

Ефективними системами криптографічного захисту даних є асиметричні криптосистеми, що називаються також криптосистемами з відкритим ключем. У таких системах для шифрування даних використовується один ключ, а для розшифрування – другий (тому їх і називають асиметричними). Перший ключ є відкритим і може бути опублікований для використання всіма користувачами системи, які зашифровують дані. Розшифрувати дані за допомогою відкритого ключа неможливо.

Для розшифрування даних одержувач зашифрованої інформації використовує другий ключ, який є секретним.

Якими б не були складними і надійними криптографічні системи, їх слабке місце при практичній реалізації – проблема розподілу ключів. Для того, щоб був можливий обмін конфіденційною інформацією між двома суб'єктами ІС, ключ повинен бути згенерований одним з них, а потім якимось чином знову ж у конфіденційному порядку переданий другому, тобто в загальному випадку для передачі ключа знову потрібно використання якоїсь криптосистеми.

Для вирішення цієї проблеми на основі результатів запропоновані системи з відкритим ключем, описані в лабораторній роботі № 9.

Криптографічні системи з відкритим ключем використовують так звані необернені, або односторонні, функції, що мають таку властивість: при заданому значенні x відносно просто обчислити значення $f(x)$, однак якщо $y = f(x)$, то немає простого шляху для обчислення значення x.

Множина класів необернених функцій породжує все розмаїття систем з відкритим ключем, проте не кожну необернену функцію можна використовувати в реальних ІС.

У визначенні необоротності вже є невизначеність. Під необоротністю розуміється не теоретична необоротність, а практична неможливість обчислити обернене значення, використовуючи сучасні обчислювальні засоби за доступний для огляду інтервал часу.

Тому, щоб гарантувати надійний захист інформації, до систем з відкритим ключем (СВК) висувають дві важливі й очевидні вимоги:

1. Перетворення вхідного тексту має бути незворотним і виключати його відновлення на основі відкритого ключа.

2. Визначення закритого ключа на основі відкритого також має бути неможливим на сучасному технологічному рівні. При цьому бажана точна нижня оцінка складності (кількості операцій) розкриття шифру.

Алгоритми шифрування з відкритим ключем набули широкого застосування в сучасних інформаційних системах. Так, алгоритм RSA став світовим стандартом де-факто для відкритих систем і рекомендований МККТТ.

Взагалі ж усі пропоновані сьогодні криптосистеми з відкритим ключем спираються на один з таких типів необоротних перетворень:

1. Розкладання великих чисел на прості множники.
2. Обчислення логарифму в кінцевому полі.
3. Обчислення коренів алгебричних рівнянь.

Тут же слід зазначити, що алгоритми криптосистеми з відкритим ключем (СВК) можна використовувати в трьох призначеннях:

1. Як самостійні засоби захисту переданих і збережених даних.

2. Як засоби для розподілу ключів. Алгоритми СВК більш трудомісткі, ніж традиційні криптосистеми. Тому часто на практиці раціонально за допомогою СВК розподіляти ключі, обсяг яких як інформації незначний, а потім за допомогою звичайних алгоритмів здійснювати обмін великими інформаційними потоками.

3. Як засоби аутентифікації користувачів.

Схема шифрування Поліга – Хеллмана

Схема шифрування Поліга – Хеллмана подібна до схеми шифрування RSA і являє собою несиметричний алгоритм, оскільки використовуються різні ключі для шифрування і розшифрування. Водночас цю схему не можна віднести до класу криптосистем з відкритим ключем, тому що ключі шифрування та розшифрування легко виводяться один з одного. Обидва ключі потрібно тримати в секреті.

Аналогічно за схемою RSA криптограма C і відкритий текст P визначаються зі співвідношень

$$C = P^e \bmod n;$$
$$P = C^d \bmod n,$$

де $ed \equiv 1$ (за модулем деякого складеного числа).

На відміну від алгоритму RSA в цій схемі число n не визначається через два великих простих числа; число n має залишатися частиною секретного ключа. Якщо хто-небудь дізнається значення e та n, то зможе обчислити значення d.

Не знаючи значення ***e*** або ***d***, супротивник буде змушений обчислювати значення

$$E = \log_p C \pmod n.$$

Відомо, що це є важким завданням.

Схема шифрування Поліга – Хеллмана запатентована в США і Канаді.

Опис послідовності дій алгоритму шифрування Поліга – Хеллмана

Припустимо, що користувач ***A*** хоче передати користувачеві ***B*** повідомлення в зашифрованому вигляді, використовуючи криптосистему Поліга – Хеллмана. У цьому випадку користувач ***A*** виступає в ролі відправника повідомлення, а користувач ***B*** – в ролі одержувача. Розглянемо послідовність дій користувачів ***B*** і ***A***.

Користувач ***B*** вибирає два довільних великих числа ***e*** і ***d***, щоб вони мали тільки один спільний дільник – одиницю.

Користувач ***B*** обчислює просте число ***n***.

Користувач ***B*** пересилає користувачеві ***A*** пару чисел ***(e, n)*** по захищеному каналу.

Користувач ***A*** хоче передати користувачеві ***B*** повідомлення ***P*** і виконує такі кроки:

1. Розбиває вхідний відкритий текст ***P*** на блоки, кожен з яких може бути поданий у вигляді числа

$$P_i = 0, 1, \ldots, n - 1.$$

2. Шифрує текст, поданий у вигляді послідовності чисел P_i, за формулою

$$C_i = P_i^e \pmod n$$

і відправляє криптограму ***{Ci}*** користувачеві ***B***.

Користувач ***B*** розшифровує прийняту криптограму ***{Ci}***, використовуючи закритий ключ ***d***, за формулою

$$P_i = C_i^d \pmod n.$$

У результаті отримує послідовність чисел ***{Pi}***, що являють собою вхідне повідомлення ***P***. Щоб алгоритм Поліга – Хеллмана мав практичну цінність, необхідно мати можливість оперативно обчислювати значення ключів ***e*** і ***d***.

Приклад виконання шифрування та дешифрування у криптосистемі Поліга – Хеллмана

Зашифруємо повідомлення «***АДЖ***». Для простоти будемо використовувати маленькі числа (на практиці застосовуються набагато більші).

Виберемо ***e = 25*** і ***d = 9***, ***n = 29***.

Уявімо шифроване повідомлення як послідовність цілих чисел за допомогою відображення: ***А → 1, Д → 5, Ж → 7***. Тоді повідомлення набуде вигляду ***(1, 5, 7)***. Зашифруємо повідомлення за допомогою ключа ***{25, 29}***:

$$C_1 = (1^{25}) \pmod{29} = 1;$$
$$C_2 = (5^{25}) \pmod{29} = 13;$$
$$C_3 = (7^{25}) \pmod{29} = 23.$$

Розшифруємо отримане зашифроване повідомлення ***(1, 13, 23)*** на основі ключа ***{9, 29}***:

$$P_1 = (1^{9}) \pmod{29} = 1;$$
$$P_2 = (13^{9}) \pmod{29} = 5;$$
$$P_3 = (23^{9}) \pmod{29} = 7.$$

Таким чином, відновлено початкове повідомлення: ***АДЖ***.

Порівняння схеми шифрування Поліга – Хеллмана з системою RSA

Нині алгоритм RSA активно реалізується у вигляді самостійних криптографічних продуктів, а також як вбудовані засоби в популярних додатках.

У схемі Поліга – Хеллмана обидва ключі необхідно тримати закритими, тому основних переваг асиметричні системи не мають. Якщо немає можливості передати ключ шифрування по захищеному каналу, то ця схема не підходить. Необхідно використовувати RSA. Однак схема Поліга – Хеллмана простіша за систему RSA.

Шифрування за алгоритмом Поліга – Хеллмана

Розглянемо ділянку програми, що шифрує символи з вхідного файлу text:

```
printf ( "cryptogram: ');
while (fgetc (text, p))
{
 c = pow (p-48, e)% n;
 putc (crypt, c);
}
```

Тут висновок криптограми виводиться і в файл crypt, і на екран.

Контрольні запитання

1. У чому полягає суть асиметричних систем?
2. Що таке відкритий ключ (public key)?
3. Що таке необернені функції? Які типи необоротних перетворень використовуються в RSA?
4. Які головні вимоги висувають до RSA?
5. Для яких призначень можна використовувати алгоритми криптосистем з відкритим ключем?
6. Які переваги системи RSA перед алгоритмом Поліга – Хеллмана?
7. Як вибираються числа ***e***, ***d*** і ***n*** у схемі Поліга – Хеллмана?
8. Які значення творець криптосистеми за алгоритмом Поліга – Хеллмана повідомляє користувачам, а які зберігає в таємниці?
9. Чи можна розшифрувати повідомлення, знаючи тільки один ключ?
10. За допомогою яких формул проводяться шифрування і дешифрування повідомлення?
11. Чи зміниться криптограма, якщо числа ***e*** і ***d*** поміняти місцями?

Варіанти завдань

Скласти програму шифрування і розшифрування повідомлення за допомогою алгоритму Поліга – Хеллмана (непарний варіант – шифрування повідомлення, парний – розшифрування). Для подання даних результату зведення в ступінь використовувати тип long int. Щоб не виникало помилки переповнення, вхідне повідомлення слід розглядати як послідовність символів з кодами 0, 1, ..., 9. Оригінал тексту, криптограму і відновлене повідомлення зберегти у файлах. Завдання за варіантами наведено в табл. 10.1.

Таблиця 10.1

Номер варіанта	Повідомлення
1, 2	794341
3, 4	033439
5, 6	656339
7, 8	123064
9, 10	344865
11, 12	097432
13, 14	456783
15, 16	486168
17, 18	864532
19, 20	052371

Лабораторна робота № 11
СХЕМА ШИФРУВАННЯ ЕЛЬ ГАМАЛЯ

Мета роботи: вивчення методу шифрування Ель Гамаля.

Теоретичні відомості

Схема Ель Гамаля, запропонована 1985 року, може бути використана як для шифрування, так і для цифрових підписів. Безпека схеми Ель Гамаля обумовлена складністю обчислення дискретних алгоритмів у кінцевому полі.

Щоб генерувати пару ключів (відкритий ключ – закритий ключ), спочатку вибирають деяке велике просте число P і велике ціле число G, причому $G < P$. Числа P та G можуть бути поширені серед групи користувачів. Потім вибирають випадкове ціле число A, причому $A < P$. Число A є секретним ключем і має зберігатися в секреті. Далі обчислюють $Y = G^A \bmod P$. Число Y є відкритим ключем.

Для того, щоб зашифрувати повідомлення M, вибирають випадкове ціле число K, $1 < K < P - 1$ таке, що числа K та $P - 1$ є взаємно простими.

Потім обчислюють такі числа:

$$Y_1 = G^k \bmod P;$$
$$Y_2 = (Y^k \bmod P) \oplus M.$$

Пара чисел (A, B) є шифротекстом. Зауважимо, що довжина шифротексту вдвічі більше довжини вхідного відкритого тексту M.

Щоб розшифрувати шифротекст (A, B), обчислюють

$$M = Y_2 \oplus (Y_1^A \bmod P).$$

Приклад

Виберемо: $P = 11$, $G = 8$, секретний ключ $A = 3$.

Рахуємо: $Y = G^A \bmod P = 8^3 \bmod 11 = 512 \bmod 11 = 6$.

Отже, відкритий ключ $Y = 6$.

Нехай повідомлення $M = 5$. Виберемо деяке випадкове число $K = 9$. Переконаємося, що $НОД(9, 10) = 1$. Обчислюємо пару чисел $(Y1, Y2)$:

$$Y_1 = G^k \bmod P = 8^9 \bmod 11 = 134217728 \bmod 11 = 7;$$
$$Y_2 = (Y^k \bmod P) \oplus M = (6^9 \bmod 11) \oplus 5 =$$
$$= (10077696 \bmod 11) \oplus 5 = 2 \oplus 5 = 7.$$

Отримаємо шифротекст $(7, 7)$.

Виконаємо розшифрування цього шифротексту. Обчислюємо повідомлення M, використовуючи секретний ключ A:

$$M = Y_2 \oplus (Y_1^A \bmod P) = 7 \oplus (7^3 \bmod 11) = 7 \oplus (343 \bmod 11) = 7 \oplus 2 = 5.$$

У реальних схемах шифрування слід використовувати як модуль ***P*** велике ціле просте число, що має у двійковому поданні довжину 512...1024 біти.

До переваг цієї системи можна віднести те, що для обміну повідомленнями немає необхідності передавати особистий ключ каналами зв'язку. Цю перевагу мають усі системи з відкритим ключем.

До недоліків цієї системи можна віднести великі витрати обчислювальних ресурсів на реалізацію операцій з великими числами при тому ж рівні захищеності, що й у інших криптосистем. Другим недоліком є подвоєння довжини повідомлення.

Алгоритм

Розподіл ключів:

– вибрати просте число ***P*** та ціле ***G (G < P)***;

– вибрати секретний ключ ***A < P***;

– обчислити відкритий ключ $Y = G^A \bmod P$;

– поширити серед групи користувачів числа ***P***, ***G***, ***Y***.

Шифрування повідомлення ***M***:

– вибрати випадкове ***1 < K <P – 1*** таке, що ***K*** і ***P – 1*** – взаємно прості числа;

– обчислити пару чисел Y_1 і Y_2 за формулами;

– передати пару (Y_1, Y_2).

Розшифрування повідомлення (Y_1, Y_2): обчислити ***M*** за формулою $M = Y_2 \oplus (Y_1^A \bmod P)$.

Контрольні запитання

1. Які властивості повинні мати числа, що використовуються в криптосистемі Ель Гамаля?
2. Якщо вхідне повідомлення складається з двох чисел, то якою буде довжина зашифрованого повідомлення?
3. Напишіть формулу для обчислення відкритого ключа.
4. Напишіть формули для обчислення шифротексту.
5. Напишіть формулу для розшифрування отриманого повідомлення.

Приклад виконання завдання

```
program ElGamal;
Var p, g, a, k, k1, y, key: LongInt;
 y1, y2: Char;
 i: Word;
 F_in: Text;
 F_out: Text;
 c: Char;
 o: char;
Function NOD (M, N: LongInt): LongInt;
Begin
 while (M <> 0) and (N <> 0) do
 Begin
 if (M> N)
 Then M: = M mod N
 Else N: = N mod M;
 End;
 if (N <> 0) then NOD: = N
 else NOD: = M;
End;
Function IsSimple (p: LongInt): Boolean;
Var k, m: LongInt;
 l: Extended;
Begin
 l: = sqrt (p);
 k: = 1;
 m: = 1;
 while ((k <l) and (m <> 0)) do
 begin
 inc (k);
 m: = p mod k;
 end;
 if (m <> 0) Then IsSimple: = true
 Else IsSimple: = False;
End;

Function SimpleNum (var p: longint): LongInt;
Begin
 while (not IsSimple (p)) do inc (p);
 SimpleNum: = p;
End;
```

```
Function st (g, k: LongInt): LongInt;
Var Pr: LongInt;
Begin
 Pr: = 1;
 for i: = 1 to k do
 Pr: = Pr * g;
 st: = Pr;
End;
Begin
{Writeln ( 'Введіть просте число p');
 readln (P);
 Writeln ( 'Введіть ціле число g');
 Readln (g);
 Writeln ( 'Введіть секретний ключ:');
 Readln (a);}
 p: = 11;
 g: = 9;
 a: = 3;
 Writeln ( 'p =', SimpleNum (p));
 y: = st (g, a) mod p;
 writeln ( 'Відкритий ключ - y =', y);
 Assign (F_in, 'in.dat');
 Reset (F_in);
 Assign (F_out, 'out.dat');
 Rewrite (F_out);
 randomize;
 While (not eof (F_in)) do
 Begin
 k1: = random (p-2) +1;
 {Пошук взаємно простого числа з числом p – 1 у всіх числах, більших k}
 k: = k1;
 While ((NOD (k, p-1) <> 1) and (k <p-1) or (k = 31)) do inc (k);
 if (k = 1)
 Then
 Begin
 {Пошук взаємно простого числа з числом p – 1 у всіх числах, менших k}
 k: = k1;
 While (NOD (k, p-1) <> 1) do dec (k);
 if (k = p-1)
 Then
 Begin
 Writeln ('Помилка !!! Не можна знайти відповідне k');
```

```
halt;
End;
End;
y1: = char (st (g, k) mod p);
Write (F_out, y1);
key: = st (y, k) mod p;
Read (F_in, c);
y2: = char (byte (c) xor key);
Write (F_out, y2);
End;
Close (F_in);
Close (F_out);
{Дешифровка}
Assign (F_in, 'in2.dat');
Rewrite (F_in);
Assign (F_out, 'out.dat');
Reset (F_out);
While (not eof (F_out)) do
Begin
Read (F_out, y1);
Read (F_out, y2);
c: = char (((st (LongInt (y1), a) mod p) xor byte (y2)));
Write (F_in, c);
End;
Close (F_out);
Close (F_in);
End.
```

Варіанти завдань

Завдання за варіантами наведено в табл. 11.1.

Таблиця 11.1

Номер варіанта	p	g	a
1, 2	11	9	3
3, 4	11	9	8
5, 6	11	9	5
7, 8	7	6	5
9, 10	11	3	6

Лабораторна робота № 12
ПЕРЕВІРКА ПРАВИЛЬНОСТІ КЛЮЧА

Мета роботи: вивчення способів перевірки правильності ключа шифрування в криптосистемах.

Теоретичні відомості

Як вводити ключ?

Вводити ключ до програми шифрування найпростіше з клавіатури. Саме цей варіант реалізований у більшості готових програм, і ознайомлення з ними починається з ввічливого запрошення: «Enter your password: ...». Тільки майте на увазі: пароль, що вводиться з клавіатури, можна підглянути. Якщо програма шифрування при введенні пароля відображає його на екрані, такою програмою краще не користуватися. Пароль, що вводиться при шифруванні файлу, не повинен відображатися на екрані.

Якщо при введенні пароля користувач випадково натиснув не ту кнопку і думає, що ввів один пароль, а насправді – інший, то програма говорить: «Пароль неправильний». Щоб такого не було, зазвичай програми шифрування при введенні пароля для шифрування тексту просять ввести пароль двічі. Якщо користувач вперше ввів один пароль, а вдруге – інший, значить, принаймні, один раз він помилився, а якщо обидва рази ввів те саме слово, значить, все нормально. Навряд чи користувач двічі помилиться однаково.

Як перевіряти правильність ключа?

До речі, а як програма при розшифруванні файлу визначає, що пароль правильний? По-різному. Деякі програми взагалі не перевіряють правильність пароля. В цьому випадку, якщо був введений неправильний пароль, то файл начебто розшифровується, але буде видно зовсім не те, що було зашифровано. Це незручно, тому краще, коли перед розшифруванням програма перевіряє правильність пароля.

Залишається питання: як перевіряти правильність пароля? Можна просто вписати пароль на початок зашифрованого файлу перед шифротекстом. При розшифруванні вводиться пароль, програма читає початок зашифрованого файлу і порівнює те, що було введено, і те, що знаходиться у файлі. Якщо співпало, значить, пароль правильний. Просто і зручно. Що ж станеться, якщо хто-небудь інший випадково перегляне зашифрований файл, наприклад, за допомогою Norton Commander? Весь файл – суцільна абракадабра, а на початку файлу – осмислене слово. Неважко здогадатися, що це і є пароль.

Очевидно, еталон пароля, який зберігається в зашифрованому файлі, теж треба зашифрувати. Тільки як? Найпростіше зашифрувати пароль за тією ж схемою, що і текст вхідного файлу. Якщо шифр стійкий (а інакше його і не варто використовувати), то пароль буде закритий надійно. Як ключ можна взяти константу, тоді в кожному зашифрованому файлі еталон пароля буде зашифрований на одному і тому ж ключі. Так, наприклад, робить вбудована система шифрування файлів електронної пошти Sprint Mail. Тільки там зашифрований пароль зберігається не на початку зашифрованого файлу, а в кінці. Якщо хтось дізнається ключ, на якому шифруються всі паролі, то зможе розшифрувати всі зашифровані файли. І неважливо, що паролі різні – зловмисник візьме потрібний пароль прямо з файлу, який хоче прочитати.

Звідки ж зловмисник дізнається ключ, яким зашифровані еталони ключів? Цей ключ вбудований у програму шифрування, його ніхто не знає. Проте, якщо зловмисник достатньо кваліфікований, якщо він вміє користуватися дизасемблером і налагоджувачем, то цей ключ він легко дізнається. Зазвичай вирішити таке завдання можна всього за кілька годин. Як це можна зробити – окрема тема. Але маючи лише exe-файл, певні навички і багато вільного часу, можна розібратися в тому, що робить програма, до найдрібніших подробиць.

Краще шифрувати еталон пароля сам на себе: взяти пароль як відкритий текст і взяти той же пароль як ключ. Під час розшифрування файлу, коли вводиться пароль, програма намагається розшифрувати тільки початок файлу. Якщо в результаті вийшов рядок, що збігається з паролем, значить, пароль правильний, і можна розшифровувати файл далі, а якщо вийшло щось інше, – неправильний.

Деякі програми шифрування шифрують паролі інакше. Береться певний рядок, завжди один і той же, шифрується на паролі і записується в зашифрований файл. Diskreet, наприклад, шифрується на паролі в рядок «ABCDEFGHENRIXYZ» (цей рядок завершується нульовим байтом, як прийнято в мові Сі). Коли Diskreet перевіряє пароль, він бере початок зашифрованого файлу (точніше, байти з 16-го по 31-й) і розшифровує їх на паролі, який ввів користувач. Якщо після розшифрування вийшло «ABCDEFGHENRIXYZ», то пароль правильний, якщо не вийшло, – неправильний.

На перший погляд здається, що ця схема гірше, ніж попередня. Дійсно, в попередньому випадку зловмисникові невідомі ні відкритий текст, ні ключ, а в останньому випадку ніхто не знає тільки ключ, а відкритий текст відомий. Але остання схема нітрохи не краща за попередню. Якщо шифр стійкий, то ключ шифрування неможливо отримати за прийнятний час, навіть якщо відомі і відкритий текст, і шифротекст.

Приклад виконання завдання

Виконати перевірку пароля методом шифрування його самого на себе. Додати результат на початок зашифрованого файлу. Дешифрування здійснювати тільки при правильності введеного пароля дешифрування.

```
Program Verify_Password;
Uses Strings;

{Додайте текст функцій кодування і декодування
Згідно з Вашим методом}
Function Coding (C: Char; Key: String): Char;
Begin
End;
Function Decoding (C: Char; Key: String): Char;
Begin
End;

{Процедура додавання зашифрованого пароля на початок файлу}
Procedure AddPassword (Key: String; var F: Text);
Var i, n: byte;
Begin
n: = Length (Key);
For i: = 1 to n do
Write (F, Coding (Key [i], Key));
End;

{Функція перевірки правильності пароля}
Function CheckPassword (Key: String; var F: Text): Boolean;
Var i, n: byte;
S: String;
c: Char;
Begin
n: = Length (Key);
For i: = 1 to n do
Begin
Read (F, c);
S [i]: = Decoding (c, Key);
End;
For i: = 1 to n do
if S [i] = Key [i] then CheckPassword: = true
else CheckPassword: = false;
```

```
 End;

Var
 F_in, F_out: Text;
 C: Char;
 Key: String [10];

Begin
 Assign (F_in, 'text.dat');
 Reset (F_in);
 Assign (F_out, 'coding.dat');
 Rewrite (F_out);
 Write ( 'Введіть пароль для шифрування:');
 Readln (Key);
 AddPassword (Key, F_out);
 While not eof (F_in) do
 Begin
 Read (F_in, C);
 Write (F_out, Coding (C, Key));
 End;
 Close (F_in);
 Close (F_out);
 Assign (F_in, 'coding.dat');
 Reset (F_in);
 Write ( 'Введіть пароль для дешифрування:');
 Readln (Key);
 If CheckPassword (Key, F_in) Then
 Begin
 Assign (F_out, 'decoding.dat');
 Rewrite (F_out);
 While not eof (F_in) do
 Begin
 Read (F_in, C);
 Write (F_out, Decoding (C, Key));
 End;
 Close (F_out);
 Writeln ( 'OK');
 End
 Else
 Writeln ( 'Паролі не збігаються. Немає запису дешифрований');
 Close (F_in);
End.
```

Контрольні запитання

1. Як можна вводити ключ у програму шифрування?
2. Як можна застрахуватися від випадкової помилки при введенні пароля?
3. Опишіть метод додавання пароля на початок зашифрованого файлу.
4. Опишіть метод шифрування пароля за допомогою константного ключа.
5. Опишіть метод шифрування пароля сам на себе.
6. Опишіть метод шифрування константного рядка.
7. У чому полягають переваги та недоліки описаних схем перевірки ключа?

Варіанти завдань

1. Виконати одну з попередніх лабораторних робіт з криптографії, додавши перевірку правильності ключа одним з описаних методів.
2. Додати пароль на початок зашифрованого файлу.
3. Зашифрувати пароль за тією ж схемою, що і текст. Як ключ взяти константу.
4. Зашифрувати пароль сам на себе.
5. Зашифрувати константний рядок.

Додаток

ЗАВДАННЯ ПОШУКУ ПРОСТИХ ЧИСЕЛ

Криптографія тісно пов'язана з різними проблемами теорії чисел, такими як пошук найменшого спільного кратного і пошук простих чисел. Зокрема, прості числа використовуються в системі шифрування RSA.

У теорії простих чисел основними є два завдання. Першим завданням вирішується питання про те, чи буде задане число ***N*** простим, другим, що має велике практичне значення в криптографії, – завдання побудови досить великих простих чисел.

Д.1. Визначення простих і складених чисел

Існує ефективний спосіб переконатися, що задане число є складеним, без розкладання його на множники. Відповідно до малої теореми Ферма, якщо число ***N*** просте, то для будь-якого цілого ***a***, що не ділиться на ***N***, виконується умова

$$a^{N-1} \equiv 1 (\mathrm{mod}\, N) \cdot \qquad (Д.1)$$

Якщо ж при якомусь ***a*** це порівняння порушується, то можна стверджувати, що ***N*** – складене. Ця перевірка не потребує великих обчислень.

Завдання полягає в пошуку числа ***a***, що не задовольняє умові (Д.1). Можна намагатися знайти необхідне число, перебираючи поспіль усі числа, починаючи з ***2***, або вибирати їх випадковим чином на відрізку ***1 < a < N***.

Однак є складені числа ***N***, що мають властивість (Д.1) для будь-якого цілого ***a*** з умовою ***(a, N) = 1***. Такі числа називаються числами Кармайкла. Можна довести, що будь-яке з чисел Кармайкла має вигляд ***N = p1, ..., pr, r ≥ 3***, де всі прості p_i – різні, причому ***N – 1*** ділиться на кожну різницю p_i ***– 1***. Лише недавно була вирішена проблема про нескінченність множини таких чисел.

1976 року Міллер запропонував замінити перевірку (Д.1) перевіркою іншої умови: якщо ***N*** – просте число, ***N – 1*** = $2^S t$, де ***t*** непарне, то відповідно до малої теореми Ферма для кожного ***a*** з умовою ***(a, N) = 1*** хоча б одна з дужок у добутку

$$\left(a^t - 1\right)\left(a^t + 1\right)\left(a^{2t} + 1\right) \cdot \ldots \cdot \left(a^{2^{S-1}t} + 1\right) = a^{N-1} - 1$$

ділиться на ***N***. Перетворення цієї властивості можна використовувати, щоб відрізняти складені числа від простих.

Нехай ***N*** – непарне складене число, ***N – 1 = $2^s t$***, де ***t*** – непарне. Назвемо ціле число ***a***, ***1 < a <N***, «хорошим» для ***N***, якщо порушується одна з двох умов:

а) ***N*** не ділиться на ***a***;

б) ***$a^t \equiv 1 \pmod N$*** або існує ціле ***k***, ***0 ≤ k <s***, таке, що

$$a^{2^k t} \equiv -1 (\mathrm{mod}\ N).$$

Зі сказаного випливає, що для простого числа ***N*** не існує «хороших» чисел ***a***. Якщо ж ***N*** складене, то їх існує не менше $\frac{3}{4}(N-1)$.

Д.2. Алгоритм, який доводить непростоту числа

Вибрати випадковим чином число ***a***, ***1 < a < N***, і перевірити зазначені вище властивості «а» і «б».

Якщо хоча б одна з властивостей «а» або «б» порушується, то число ***N*** є складеним.

Інакше (виконано обидві умови) слід повернутися до кроку 1.

Складене число не буде правильно визначено після однократного виконання алгоритму з ймовірністю не більше ***4^{-1}***. Вірогідність не визначити його після ***k*** повторень менше або дорівнює ***4^{-k}***.

Міллер запропонував детермінований алгоритм визначення складених чисел, що має складність ***$O(\ln^3 N)$***. Згідно з ним досить перевірити умови «а» і «б» для всіх цілих чисел ***a***, ***$2 \le a \le 70 \ln^2 N$***. Якщо для всіх них ці умови виконуються, то число ***N*** є простим або ступенем простого числа. Остання можливість легко перевіряється.

Однак справедливість результату цього алгоритму залежить від недоведеної нині розширеної гіпотези Рімана.

Д.3. Побудова простих чисел

Найбільш ефективним засобом побудови простих чисел є модифікована мала теорема Ферма.

Нехай ***N***, ***S*** – непарні натуральні числа, ***N – 1 = SR***, причому для кожного простого дільника ***q*** числа ***S*** існує ціле число ***a***, таке, що

$$a^{N-1} \equiv 1 (\mathrm{mod}\ N), \qquad \left(a^{\frac{N-1}{q}} - 1,\ N \right) = 1.$$

Тоді кожен простий дільник ***p*** числа ***N*** задовольняє рівнянню

$$p \equiv 1 \pmod{2S}.$$

Наслідок. Якщо виконані умови теореми і $R \leq 4S + 2$, то ***N*** – просте число.

Покажемо тепер, як за допомогою останнього твердження, маючи велике просте число ***S***, можна побудувати істотно більше просте число ***N***. Виберемо для цього випадковим чином парне число ***R*** на проміжку $S \leq R \leq 4S + 2$ і покладемо $N = SR + 1$. Потім перевіримо число ***N*** на відсутність малих простих дільників, розділивши його на малі прості числа, і випробуємо за допомогою вказаного вище алгоритму. Якщо з'ясується, що число складене, то слід вибрати нове значення ***R*** і знову повторити обчислення.

Побудоване таким чином просте число ***N*** буде задовольняти нерівності $N > S^2$, тобто буде записуватися вдвічі більшою кількістю цифр, ніж вхідне ***S***.

БІБЛІОГРАФІЧНИЙ СПИСОК

Аршинов, М. Н. Коды и математика / М. Н. Аршинов, Л. Е. Садовский. – М. : Наука, 1983. – 144 с.

Винокуров, А. ГОСТ не прост..., а очень прост / А. Винокуров // Монитор. – 1995. – № 1. – С. 60–73.

Герасименко, В. А. Защита информации в автоматизированных системах обработки данных. В 2 кн. / В. А. Герасименко. – М. : Высшая шк., 1994. – 576 с.

ГОСТ 28147–89. Система обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. Введ. 01.01.88. – М. : Госкомстат, 1989. – 28 с.

ГОСТ Р 34.10–94. Государственный стандарт Российской Федерации. Криптографическая защита информации. Процедуры выработки и проверки электронной цифровой подписи на основе асимметричного криптографического алгоритма. – Введ. 01.01.94. – М. : Изд-во стандартов, 1993. – 18 с.

Диффи, У. Защищенность и имитостойкость: введение в криптографию / У. Диффи, М. Хеллман // ТИИЭР. – 1979. – Т. 67, № 3. – С. 71–109.

Диффи, У. Новые направления в криптографии / У. Диффи, М. Хеллман // ТИИЭР. – 1976. – Т. 22, № 6. – С. 644–654.

Домашев, А. В. Программирование алгоритмов защиты информации: учеб. пособ. / А. В. Домашев. – М. : Нолидж, 2000. – 288 с.

Жельников, В. Криптография от папируса до компьютера / В. Жельников. – М. : ABF, 1966. – 335 с.

Кнут, Д. Искусство программирования на ЭВМ. В 5 т. Т. 2. Получисленные алгоритмы / Д. Кнут. – М., 2001. – 832 с.

Саломан, А. Криптография с открытым ключом / А. Саломан. – М. : Мир, 1996. – 318 с.

Шеннон, К. Э. Теория связи в секретных системах / К. Э. Шеннон. – М. : Изд-во иностр. лит-ры, 1963. – 830 с.

ЗМІСТ

Навчальне видання

Дем'яненко Владислав Анатолійович
Кузнецова Юлія Анатоліївна

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

Редактор Н. В. Мазепа

Зв. план, 2021
Підписано до видання 19.04.2021
Ум. друк. арк. 5,3. Обл.-вид. арк. 5,94. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001