МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ЦИФРОВИХ ТЕХНОЛОГІЙ, ДИЗАЙНУ ТА ТРАНСПОРТУ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ ТА ДИЗАЙНУ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни «Технології захисту інформації»
для студентів усіх форм навчання

Перший (бакалаврський) рівень вищої освіти
Спеціальність – 122 КОМП'ЮТЕРНІ НАУКИ
Освітня програма – КОМП'ЮТЕРНИЙ ДИЗАЙН

Одеса – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ЦИФРОВИХ ТЕХНОЛОГІЙ, ДИЗАЙНУ ТА ТРАНСПОРТУ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ ТА ДИЗАЙНУ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни «Технології захисту інформації»
для студентів усіх форм навчання

Перший (бакалаврський) рівень вищої освіти
Спеціальність – 122 КОМП'ЮТЕРНІ НАУКИ
Освітня програма – КОМП'ЮТЕРНИЙ ДИЗАЙН

Затверджено на засіданні
кафедри інформаційних технологій
проєктування та дизайну
Протокол № 1 від 27.08.2025

Одеса – 2025

Методичні вказівки до виконання лабораторних робіт з дисципліни «Технології захисту інформації» для студентів першого (бакалаврського) рівня за спеціальністю 122 Комп’ютерні науки, за освітньо-професійною програмою «Комп’ютерний дизайн» / Укл.: А. С. Коляда, О. С. Лопаков, В. В. Космачевський – Одеса : Національний університет «Одеська політехніка», 2025. – 48 с.

Укладачі:

Коляда А. С., к.т.н., доц. кафедри
Лопаков О. С., ст. викладач
Космачевський В. В., ст. викладач

Дані методичні вказівки розглядають основні аспекти криптографічного захисту інформаційних систем, які необхідно опрацювати під час виконання лабораторних робіт. Оскільки інформація є цінним ресурсом, виникають ризики несанкціонованого доступу та викривлення даних, що вимагає застосування ефективних методів захисту. Виконання лабораторних робіт спрямоване на практичне освоєння криптографічних методів захисту, зокрема забезпечення конфіденційності та цілісності даних у цифрових системах. Особливу увагу приділено впливу сучасних технологій на розвиток криптографії, методам контролю справжності інформації, а також загрозам, пов’язаним із безпаперовим документообігом, електронними платежами та цифровими архівами. Завдання лабораторних робіт передбачають вивчення та реалізацію криптографічних алгоритмів, оцінку їхньої ефективності, а також аналіз можливих атак на захищені системи. Це дозволить студентам отримати практичні навички розробки та використання сучасних засобів інформаційної безпеки.

ЗМІСТ

Лабораторна робота №1 Прості шифри

Мета роботи: Ознайомлення з простими симетричними криптографічними шифрами на основі методів підстановок, перестановок і гамування.

Короткі теоретичні відомості

Класифікація шифрів за ключовою інформацією

Першим важливим показником, що дозволяє зробити поділ шифрів, є обсяг інформації, невідомої третій стороні. У тому випадку, коли зловмисникові повністю невідомий алгоритм виконаного над повідомленням перетворення, шифр називається **тайнописом.**

На відміну від тайнопису ***криптографією з ключем*** називають сьогодні алгоритми шифрування, в яких сам алгоритм перетворень широко відомий і доступний для досліджень кожному бажаючому, але шифрування проводиться на основі невеликого обсягу інформації - ключа, відомого тільки відправнику і одержувачу ключа.

У деяких (найбільш простих) випадках ключ формує людина, яка відправляє повідомлення, в окремих випадках ключ створюється автоматично за допомогою програмного забезпечення або навіть запитується у віддаленій базі даних ключів. В сучасної криптографії в залежності від методик розмір ключа становить від 56 до 4096 біт. Запам'ятовувати ключ, який є насправді просто великою послідовністю чисел, звичайній людині досить складно хоч в двійковій, хоч в десятковій формі запису. Тому всі сучасні системи пропонують користувачеві вводити не ключ - набір цифр, а пароль - довільну текстову фразу. Пароль може складатися з одного або декількох слів, бути осмисленим чи ні, головне - щоб він легко запам'ятовувався користувачем.

Симетричне / асиметричне шифрування

Усі криптоалгоритми з ключем діляться на симетричні і асиметричні. У симетричних криптоалгоритмах ключі, які використовуються на передавальної і приймальні сторонах, повністю ідентичні. Такий ключ несе в собі всю інформацію про засекречування повідомлення і тому не повинен бути відомий нікому, крім двох що беруть участь у розмові сторін. Тому щодо ключа симетричних систем часто називаються ***шифрами на секретному ключі***.

Загальна схема процесу передачі повідомлення приведена на рис. 1. Симетричне шифрування можна застосовувати як при відправці повідомлень між двома користувачами, розділеними великою відстанню, так і при відправці «посилань» одним і тим же користувачем самому собі. Прикладом подібних відправлень є шифрування файлів на жорстких дисках і змінних носіях, щоб інші користувачі тих же ЕОМ не могли зчитати інформацію за відсутності власника.

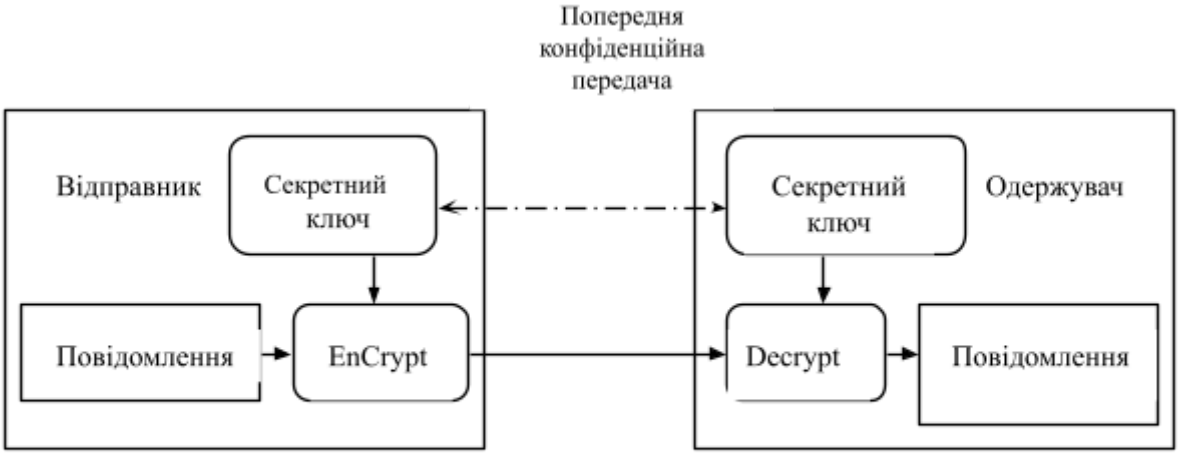


Рис. 1. Шифрування на секретному ключі

У асиметричному шифруванні для шифрування застосовується один ключ, а для дешифрування - інший (рис.2). Чому це необхідно? Справа в тому, що процедура шифрування в асиметричних системах влаштована таким чином, що жодна стороння особа не може, знаючи зашифрований таким способом текст і ключ шифрування, відновити вихідний текст. Прочитати зашифрований текст можна, тільки знаючи ключ дешифрування. А раз так, то ключ шифрування може бути відомий всім користувачам мережі - його розкриття не заподіє ніякої шкоди. Тому ключ шифрування в асиметричних системах називається відкритим ключем. Ключ дешифрування необхідно тримати в строгому секреті, як і секретний ключ симетричних систем. Тому він носить назву закритого ключа, а самі асиметричні системи отримали ще одну назву - ***шифри на відкритому ключі***.

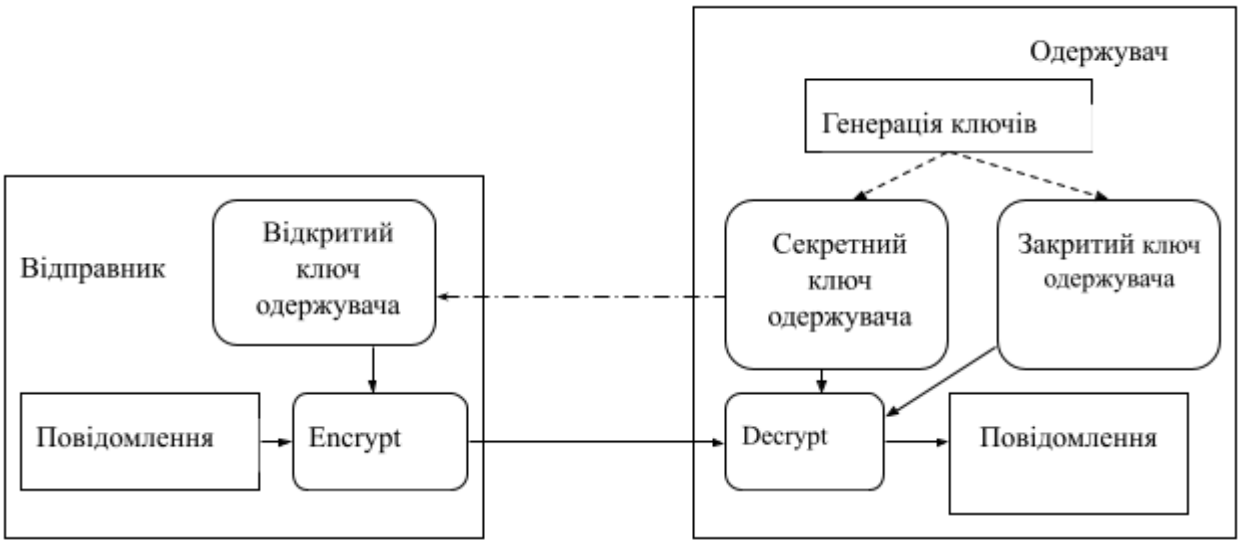


Рис. 2. Шифрування на відкритому ключі

Потокове / блочне шифрування

Наступним критерієм класифікації шифрів є схема обробки ними потоку інформації. Згідно з ним, симетричні криптоалгоритми діляться на потокові та блокові шифри. Поточний шифр здатний обробляти інформацію побітово, тобто подібна схема може, отримавши порцію з довільної кількості біт (може бути, навіть одного), зашифрувати / дешифрувати її і передати для подальшої обробки інших модулів. Подібна схема дуже зручна в каналах послідовного зв'язку, де сам процес передачі інформації може обриватися в довільний момент і потім через деякий проміжок часу тривати далі.

Однак побітова обробка інформації є дуже повільної в тих випадках, коли обчислювальна техніка має можливості для паралельної обробки (тобто програмної реалізації). Розрядність основної маси сучасних процесорів дорівнює 32 бітам, існують 64- і 80-розрядні апаратні платформи. У цих умовах, особливо тоді, коли інформація все одно переноситься в буфер в тому чи іншому обсязі (пакети в комп'ютерних мережах, файли на носіях), набагато вигідніше застосовувати абсолютно інші принципи криптографічних перетворень, звані ***блоковими шифрами***.

У блокових шифрах перетворення можуть застосовуватися тільки над інформацією строго певного обсягу. Розмір блоку на сьогоднішній день дорівнює 64, 126 або 256 бітів. Часткове шифрування (наприклад, спроба обробити 177 біт) неможливо. Блочне шифрування отримало набагато більш широке поширення через розвиток сучасної обчислювальної техніки, і, якщо потокові шифри однаково часто реалізуються як програмно, так і апаратно, то блокові шифри в переважній більшості реалізуються програмно.

Загальна схема класифікації шифрів приведена на рис. 3.

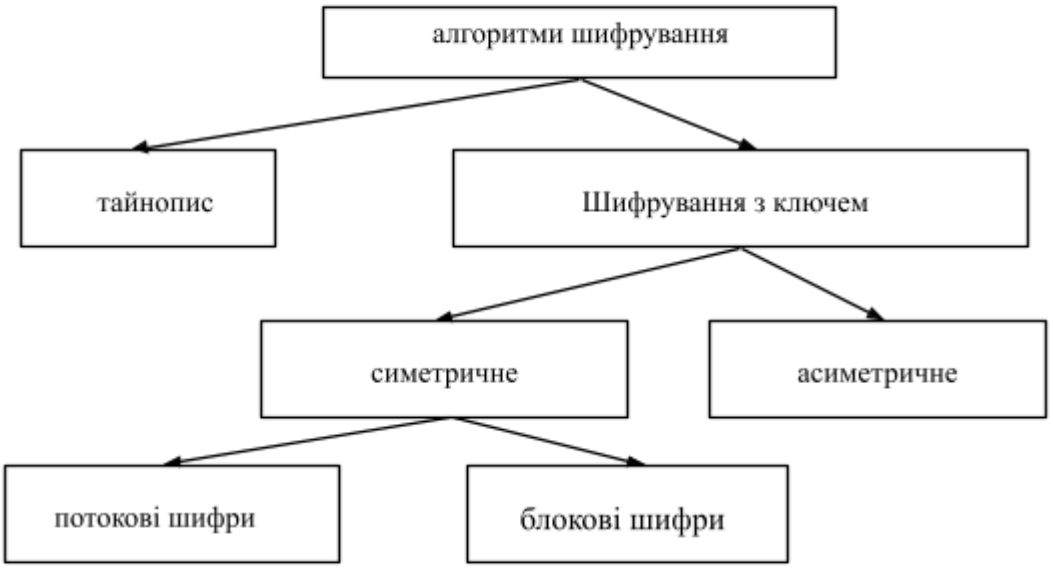


Рис. 3. Загальна схема класифікації шифрів

Класифікація за способом

Підстановлювальні шифри

Підстановлювальним шифром називається шифр, який кожен символ відкритого тексту в шифротексті замінює іншим символом. Одержувач інвертує підстановку шифротексту, відновлюючи відкритий текст. У класичній криптографії існує чотири типи підстановлювальних шифрів:

- простий підстановлювальний шифр**,** або ***моноалфавітний шифр,***- це шифр, який кожен символ відкритого тексту замінює відповідним символом шифротексту. Простими підстановлювальнимми шифрами є криптограми в газетах;

- однозвучний підстановлювальний шифр схожий на просту підстановлювальний криптосистему за винятком того, що один символ відкритого тексту відображається на кілька символів шифротексту. Наприклад, "A" може відповідати 5, 13, 25 або 56, "B" - 7, 19, 31 або 42 і так далі;

- поліграмний підстановлювальний шифр - це шифр, який блоки символів шифрує по групах. Наприклад, "ABA" може відповідати "RTQ", "ABB" може відповідати "SLL" і так далі;

- поліалфавітний підстановлювальний шифр складається з декількох простих підстановлювальних шифрів. Наприклад, можуть бути використані п'ять різних простих підстановлювальних фільтрів; кожен символ відкритого тексту замінюється з

використанням одного конкретного шифру.
Відомий шифр Цезаря, в якому кожен символ відкритого тексту замінюється символом, що знаходиться трьома символами правіше по модулю 26 ("A" замінюється на "D," "B" - на "E", ... "W" - на "Z" , "X" - на "A", "Y" - на "B", "Z" - на "C"), являє собою простий підстановлювальний фільтр. Він дійсно дуже простий, так як алфавіт шифротекста є зміщений, а не випадково розподілений алфавіт відкритого тексту.
Перестановлювальні шифри
Шифр, перетворення з якого змінюють тільки порядок проходження символів вихідного тексту, але не змінюють їх самих, називається шифром перестановки (ШП). Розглянемо перетворення з ШП, призначене для шифрування повідомлення довжиною n символів. Його можна уявити за допомогою таблиці

$$\begin{pmatrix} 1\ 2 \dots n \\ i_1\ i_2 \dots i_n \end{pmatrix}$$

де i1 - номер місця шифротекста, на яке потрапляє перша буква вихідного повідомлення при обраному перетворенні, i2 - номер місця для другої літери і т. д.
У верхньому рядку таблиці виписані по порядку числа від 1 до n, а в нижній ті ж числа, але в довільному порядку. Така таблиця називається підстановкою ступеня n.

Знаючи підстановку, що задає перетворення, можна здійснити як шифрування, так і розшифрування тексту. Наприклад, якщо для перетворення використовується підстановка

$$\begin{pmatrix} 1\,2\,3\,4\,5\,6 \\ 5\,2\,3\,1\,4\,6 \end{pmatrix}$$

і відповідно до неї зашифрована слово МОРКВА, то вийде ВОРМКА.

Знаючи метод математичної індукції, легко переконатися в тому, що існує n!(N факторіал) варіантів заповнення нижнього рядка таблиці (1). Таким чином, число різних перетворень шифру перестановки, призначеного для шифрування повідомлень довжини n, менше або дорівнює n!. (Зауважимо, що в це число входить і варіант перетворення, що залишає всі символи на своїх місцях!).

Зі збільшенням числа n значення n! факторіал зростає дуже швидко:

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

Прикладом ШП, призначеного для шифрування повідомлень довжини n, є шифр, в якому в якості безлічі ключів взято безліч всіх підстановок ступеня n, а відповідні їм перетворення шифру задаються, як було описано вище. Число ключів такого шифру одно n!
Таблиця 1.1 - Значення N! для перших 10 натуральних чисел

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

Широке поширення отримали фігури перестановки, які використовують деяку геометричну фігуру. Перетворення з цього шифру полягають у тому, що в фігуру вихідний текст вписується по ходу одного "маршруту", а потім по ходу другого виписується з неї. Такий шифр називається ***маршрутною перестановкою***. Наприклад, можна вписувати вихідне повідомлення в прямокутну таблицю, вибравши такий маршрут: по горизонталі, починаючи з лівого верхнього 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:
ПРІМЕРМАРШРУТНОЇПЕРЕСТАНОВКИ
використовуючи прямокутник розміру 4×7

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

Зашифрована фраза виглядає так:
МАСТАЕРРЕШРНОЕРМІУПВКЇТРПНОИ

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

Для використання шифру, званого ***поворотними гратами***, виготовляється трафарет з прямокутного аркуша паперу в клітинку розміру 2m×2k клітин. У трафареті вирізане m×k клітин так, що при накладенні його на аркуш чистого паперу того ж розміру чотирма можливими способами його вирізи повністю покривають всю площу аркуша.

Букви повідомлення послідовно вписуються в вирізи трафарету (по рядках, в кожному рядку зліва направо) під час кожного з чотирьох його можливих положень в заздалегідь встановленому порядку.

Приклад шифрування. Нехай в якості ключа використовуються ґрати 6×10, наведена на рис. 4.1.

Зашифруємо з її допомогою текст:

ШИФРРЕШІТКАЯВЛЯЕТСЯЧАСТНИМСЛУЧАЕМШИФРАМАРШРУТНОЇ ПЕРЕСТАНОВКИ

Наклавши грати на аркуш паперу, вписуємо перші 15 (по числу вирізів) букв повідомлення ШИФРРЕШІТКАЯВЛЯ ... .. Знявши грати, ми побачимо текст, представлений на рис. 4. 2. Повертаємо грати на 180°. У віконцях з'являться нові, ще не заповнені клітини. Вписуємо в них наступні 15 букв. Вийде запис, наведена на рис. 4. 3. Потім повертаємо грати на іншу сторону і зашифровувати залишок тексту аналогічним чином (рис. 4.4, 4.5).

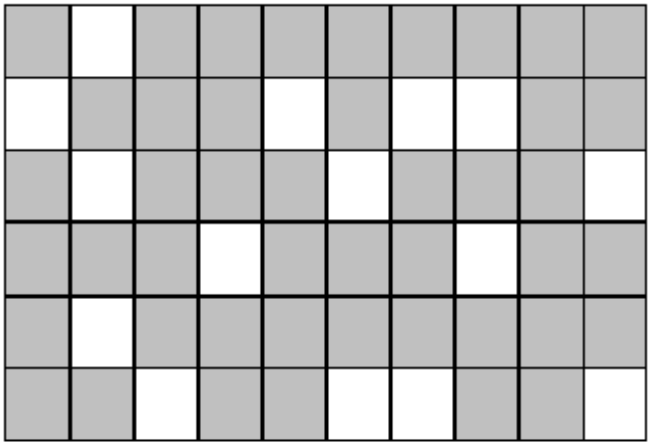


Рис. 4.1

	Ш								
І				Ф		Р	Р		
	Е				Ш				Е
			Т				До		
	А								
		Я			В	Л			Я

Рис. 4. 2

Е	Ш		Т	З			Я		
І				Ф		Р	Р	Ч	
	Е	А			Ш	З			Е
			Т				До		
	А								
		Я			В	Л			Я

Рис. 4.3

Е	Ш	А	Т	З	Е	М	Я		Ш
І	І			Ф		Р	Р	Ч	
	Е	А	Ф		Ш	З	Р		Е
Т	А		Т	Н	М		До	И	А
Р	А	М	З	Ш	Л	Р	У		У
	Т	Я			В	Л		Ч	Я

Рис. 4. 4

Одержувач повідомлення, що має точно таку ж грати, без праці прочитає вихідний текст, наклавши грати на шифротекст по порядку чотирма способами. Можна довести, що число можливих трафаретів, тобто кількість ключів шифру "решітка", становить $T = 4^{mk}$. Цей шифр призначений для повідомлень довжини n = 4mk. Число всіх перестановок в тексті такої довжини складе (4mk)!, що у багато разів більше числа Т. Однак, вже при розмірі трафарету 8×8 число можливих решіток перевершує 4 мільярди

Шифром вертикальної перестановки(ШВП) називається широко розповсюджений різновид шифру маршрутної перестановки. У ньому використовується прямокутник, в якому повідомлення вписується звичайним способом (по рядках зліва направо). Виписуються букви по вертикалі, а стовпці при цьому беруться в порядку, визначеному

ключем. Нехай, наприклад, цей ключ такий: (5,1,4,7,2,6,3), і з його допомогою треба зашифрувати повідомлення:
ОСЬПРІМЕРШИФРАВЕРТИКАЛЬНОЇПЕРЕСТАНОВКИ

Впишемо повідомлення в прямокутник, стовпчики якого пронумеровані відповідно до ключа:

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

Тепер, вибираючи стовпці в порядку, заданому ключем і виписуючи послідовно букви кожного з них зверху вниз, отримуємо таку криптограму:

СРЕЬЕКРФИЇА-МААЕО-ЬШРНСИОЕВЛРВІРКПН-ПИТОТ-

Число ключів ШВП не більше m!, де m - число стовпців таблиці. Як правило, m набагато менше, ніж довжина тексту n (повідомлення вкладається в кілька рядків по m букв), а значить, і m! багато менше n !.

У разі, коли ключ ШВП не рекомендується записувати, його можна витягати з будь-якого слова або речення. Для цього існує багато способів. Найбільш поширений полягає в тому, щоб приписувати буквах числа відповідно до звичайного алфавітним порядком букв. Наприклад, нехай ключовим словом буде ПЕРЕСТАНОВКА. Присутня в ньому буква А отримує номер 1. Якщо якась буква входить кілька разів, то її поява нумерується послідовно зліва направо. Тому друге входження літери А отримує номер 2. Оскільки літери Б в цьому слові немає, то буква В отримує номер 3 і так далі. Процес триває до тих пір, поки всі букви не отримають номери. Таким чином, ми отримуємо наступний 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

Гамування

Гамування також є широко застосовуваним криптографічним перетворенням. Принцип шифрування гамуванням полягає в генерації гами шифру за допомогою датчика псевдовипадкових чисел і накладення отриманої гами на відкриті дані оборотним чином (наприклад, використовуючи додавання по модулю 2).

Процес дешифрування даних зводиться до повторної генерації гами шифру при відомому ключі і зверненні процесу накладення такої гами на зашифровані дані.

Для програмної генерації гами шифру необхідно скористатися датчиком псевдовипадкових чисел. Найбільш простим для реалізації є генератор лінійної конгруентної послідовності:

$$T_{i+1} = (A.* T_i + C) \bmod m$$

Такий датчик ПСЧ генерує псевдовипадкові числа з певним періодом повторення, що залежать від вибраних значень A і C. Значення *m* зазвичай встановлюється рівним 2n, де n - довжина машинного слова в бітах. Датчик має максимальний період M до того, як генерується послідовність почне повторюватися. З причини, зазначеної раніше, необхідно вибирати числа A і C такі, щоб період M був максимальним. Як показано Д. Кнутом, лінійний конгруентний датчик ПСЧ має максимальну довжину M тоді і тільки тоді, коли C - непарне, і A mod 4 = 1.

Завдання на лабораторну роботу

Зашифрувати шифром маршрутної перестановки своє прізвище і ім'я, вибравши при цьому алгоритм заповнення матриці.

Скласти програму та блок схему алгоритму генератора псевдовипадкових чисел, використовуючи конгруентний метод формування псевдовипадкових чисел, на інтервалі [0; 1).

Опис способу обробки результатів

Суть методу полягає в обчисленні послідовності випадкових чисел X_n, вважаючи:

$$X_{n+1} = (aX_n + c) \mod m,$$

де m — модуль (натуральне число, відносно якого обчислюють остачу від ділення; $m \geqslant 2$), a — множник ($0 \leqslant a < m$), c — приріст ($0 \leqslant c < m$), X_0 — початкове значення ($0 \leqslant X_0 < m$).

Ця послідовність називається *лінійною конгруентною послідовністю*. Наприклад, для $m = 10, X_0 = a = c = 7$ отримаємо послідовність $7, 6, 9, 0, 7, 6, 9, 0, \ldots$

При виборі числа m необхідно враховувати наступні умови:

1) число m має бути досить великим, так як період не може мати більше m елементів;

2) значення числа m має бути таким, щоб $(aX_n + c) \mod m$ обчислювалося швидко.

На практиці при реалізації методу виходячи із зазначених умов найчастіше вибирають $m = 2^e$, де e - число бітів в машинному слові. При цьому варто враховувати, що молодші виконавчі розряди згенерованих таким чином випадкових чисел демонструють поведінку, далеке від випадкового, тому рекомендується використовувати тільки старші розряди. Подібна ситуація не виникає, коли $m = w \pm 1$, де w - довжина машинного слова. В такому випадку молодші розряди X_n поводяться так само випадково, як і старші. Вибір множника a і збільшення c в основному обумовлений необхідністю виконання умови досягнення періоду максимальної довжини.

Для того щоб отримати результат на інтервалі [0; 1), потрібно отримане псевдо випадкове число розділити на m:

Yn+1 = Xn+1 / m

Опис ходу експерименту

Псевдокод

```
Random {
a = 1103515245
c = 12345
m = 4294967296
next = 1;
void seed(initial_number) {
        next = initial_number;
}
double next() {
        next = (a * next + c) mod m;
        return next / m;
}
};

Random rand;
rand.seed(42);
```

```
for n in 0..5 {
println("{1}: {2}", n, rand.next());
}
```

Варіанти завдань

№	m	a	c
1	2^{32}	1664525	1013904223
2	2^{32}	22695477	1
3	2^{31}	1103515245	12345
4	2^{32}	1103515245	12345
5	2^{32}	134775813	1
6	2^{32}	214013	2531011
7	2^{24}	1140671485	12820163
8	$2^{31} - 1$	2147483629	2147483587
9	$2^{31} - 1$	16807	0
10	$2^{31} - 1$	48271	0
11	2^{64}	6364136223846793005	1442695040888963407
12	2^{64}	6364136223846793005	1
13	2^{32}	69069	1
14	2^{48}	25214903917	11
15	2^{31}	65539	0

Домашнє завдання

1. Реалізувати один із простих шифрів (підстановки або перестановки) для довільного тексту.
2. Проаналізувати криптостійкість вибраного шифру та зробити короткий висновок.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. Які основні види шифрів виділяють за типом використання ключової інформації?
2. У чому полягає різниця між симетричними та асиметричними шифрами? Які переваги та недоліки кожного методу?
3. Що таке метод гамування в криптографії? Як він використовується для забезпечення безпеки інформації?

4. Які види шифрів виділяють залежно від способу обробки даних (блокові та потокові шифри)? Які особливості їх використання?
5. Як створюється гамма для шифрування?

Література

1. Борщенко І. О. Основи криптографії: навчальний посібник. – Київ: КНТ, 2018. – 256 с.
2. Шеннон К. Теорія комунікації в секретних системах / К. Шеннон // Bell System Technical Journal. – 1949. – Т. 28, №4. – С. 656–715.
3. Станойлович Д. Сучасні методи шифрування: від симетричних до асиметричних алгоритмів. – Харків: Фоліо, 2020. – 320 с.
4. Столін В. О. Потокові шифри та їх застосування у захисті інформації: монографія. – Львів: Вид-во ЛНУ, 2019. – 198 с.
5. Брус М. Гамування в криптографії: теоретичні основи та практичне використання / М. Брус, О. Коваль // Журнал сучасних інформаційних технологій. – 2021. – Т. 15, №2. – С. 112–119.
6. Євдокимов М. В. Генератори псевдовипадкових чисел для криптографічних застосувань. – Дніпро: УДХТУ, 2022. – 164 с.

Лабораторна робота №2 Застосування режиму однократного гамування

Мета роботи: Освоїти на практиці застосування режиму однократного гамування. Дослідити побітно безперервне шифрування даних.

Короткі теоретичні відомості

Найпростішою і в той же час найбільш надійною з усіх схем шифрування є так звана схема одноразового використання (рис. 1), винахід, яке найчастіше пов'язують з ім'ям Г.С. Вернама.

Гамування - це накладення (зняття) на відкриті (зашифровані) дані криптографічного гами, тобто послідовності елементів даних, що виробляються за допомогою деякого криптографічного алгоритму, для отримання зашифрованих (відкритих) даних.

З точки зору теорії криптоанализу, метод шифрування випадкової одноразової рівноймовірної гамою тієї ж довжини, що і відкритий текст , є абсолютно стійким (далі для стислості будемо вживати термін "одноразове гамування"). Крім того, навіть розкривши частину повідомлення, дешифрувальник не поліпшить загальне становище - інформація про розкриту ділянку гами не дає інформації по інших її частинах.

Припустимо, в таємному діловому листуванні використовується метод одноразового накладення гами на відкритий текст. Нагадаємо, що "накладення" гами не що інше, як виконання операції додавання по модулю 2 (xor) її елементів з елементами відкритого тексту, яка в мові програмування С позначається знаком ^, а в математиці - знаком $\oplus$.

Стандартні операції над бітами: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$.

Цей алгоритм шифрування є симетричним. Оскільки подвійне поповнення однієї і тієї ж величини по модулю 2 відновлює початкове значення, шифрування і розшифрування виконується однією і тією ж програмою.

Режим шифрування одноразового гамування реалізується в такий спосіб:

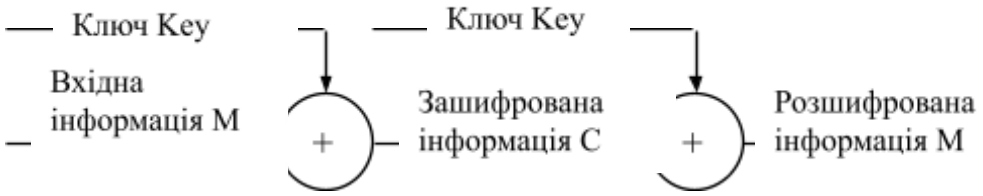


Рис. 1 - Схема одноразового гамування Вернама

Завдання знаходження шифротексту при відомому ключі і відкритому тексті полягає в застосування наступного правила до кожного символу відкритого тексту:

$$C_i = M_i \oplus Key_i, \quad (1)$$

де C_i - i-тий символ отриманого зашифрованого послання, M_i - i-тий символ відкритого тексту, Keyi - i-тий символ ключа, де i = 1, 2, ..., m. Розмірності відкритого тексту і ключа повинні збігатися, і отриманий шифртекст буде ідентичною довжини.

Завдання знаходження ключа за відомим шифротекстом і відкритого тексту може бути вирішена, виходячи з (1). Обидві частини рівності складемо по модулю 2 з M_i.

$$C_i \oplus M_i = M_i \oplus Key_i \oplus M_i = Key_i, \quad (2)$$

$$Key_i = C_i \oplus M_i, \quad (3)$$

Таким чином, отримали формули для вирішення обох поставлених завдань. Так як відкритий текст представлений в символьному вигляді, а ключ - у своєму шістнадцятковому представленні, то у відповідність з таблицею ASCII-кодів можна уявити ключ в символьному вигляді. Тоді вже будуть можливі операції (1), (3), необхідні для вирішення поставлених завдань.

Завдання на лабораторну роботу

Вибрати з таблиці метод гамування та параметр b (для розрахунку $M = 2^b$), інші параметри (a, c, T(0) вибрати довільно).
Розробити програму шифрування і дешифрування тексту.
Виконати шифрування вихідного тексту, отримати шифрограму, здійснити її дешифрування і порівняння з вихідним текстом.
Зміна один або декілька параметрів генератора випадкових чисел, виконати шифрування того ж тексту ще раз і порівняння з попереднім варіантом.

Опис способу обробки результатів

К. Шенноном доведено, що, якщо ключ є фрагментом істинно випадкової двійковій послідовністю з рівномірним законом розподілу, причому його довжина дорівнює довжині вихідного повідомлення, і використовується цей ключ тільки один раз, після чого знищується, такий шифр є абсолютно стійким, навіть якщо криптоаналітик має необмежений ресурс часу і необмежений набір обчислювальних ресурсів. Дійсно, зловмиснику відомо тільки зашифроване повідомлення C, при цьому всі різні ключові послідовності Key можливі і різновірогідні, а значить, можливі і будь-які повідомлення M, тобто криптоалгоритм не дає ніякої інформації про відкрите тексті.
Необхідні і достатні умови абсолютної стійкості шифру:
повна випадковість ключа;
рівність довжин ключа і відкритого тексту;
одноразове використання ключа.
Розглянемо приклад.
Ключ Центру:
05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54
Повідомлення Центру:
Штірліц - Ви Герой !!
D8 F2 E8 F0 EB E8 F6 20 2D 20 C2 FB 20 C3 E5 F0 EE E9 21 21
Зашифрований текст, що знаходиться у Мюллера:
DD FE FF 8F E5 A6 C1 F2 B9 30 CB D5 02 94 1A 38 E5 5B 51 75
Дешифрувальники спробували ключ:
05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 55 F4 D3 07 BB BC 54
і отримали текст:
D8 F2 E8 F0 EB E8 F6 20 2D 20 C2 FB 20 C1 EE EB E2 E0 ED 21
Штірліц - Ви Бовдур!
Пробуючи нові ключі, вони будуть бачити все нові і нові фрази, прислів'я, віршовані строфи, словом, всілякі тексти заданої довжини.
Режим шифрування одноразового гамування одним ключем двох видів відкритого тексту реалізується в такий спосіб:

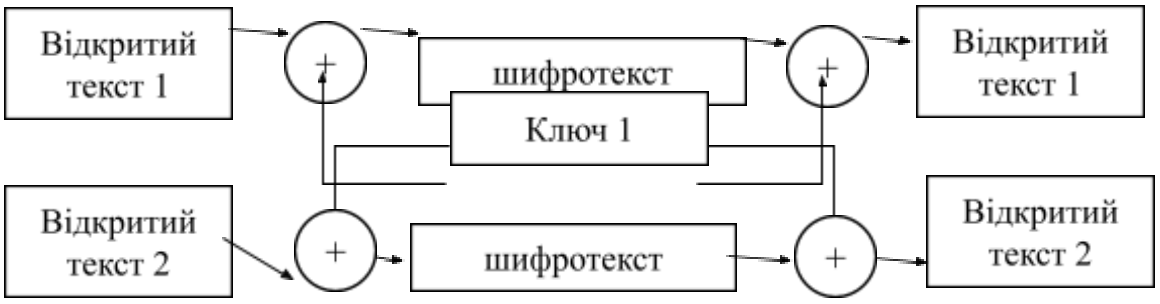


Рис. 2 - Загальна схема шифрування двох різних текстів одним ключем.
За допомогою формул режиму однократного гамування отримаємо шифротексти обох телеграм:

$$Y_1 = T_1 \oplus K,$$
$$Y_2 = T_2 \oplus K \qquad (4)$$

Завдання знаходження відкритого тексту за відомим шифротекстом двох телеграм, зашифрованих одним ключем, може бути вирішена, виходячи з (4). Складемо по модулю 2 обидва рівності (4). З огляду на таку властивість операції xor (⊕), Що

$$1 \oplus 1 = 0,\ 1 \oplus 0 = 1 \qquad (5)$$

отримуємо:

$$Y_1 \oplus Y_2 = T_1 \oplus K \oplus T_2 \oplus K = T_1 \oplus T_2 \qquad (6)$$

Припустимо, що одна з телеграм є "рибою" - тобто має фіксований формат, в який вписуються значення полів, і зловмисникові цей формат відомий. Тоді він отримує досить багато пар Y1⊕Y2 (відомий вид обох кодувань) і, припустимо, Т1. Тоді, враховуючи (5), маємо:

$$Y_1 \oplus Y_2 \oplus T_1 = T_1 \oplus T_2 \oplus T_1 = T_2 \qquad (7)$$

Таким чином, зловмисник отримує можливість визначити ті символи повідомлення T_2, які знаходяться на позиціях відомої "риби" повідомлення T_1. Здогадуючись за логікою повідомлення T_2, зловмисник має реальний шанс дізнатися ще кілька символів повідомлення T_2. Потім використовує (7), замість T_1 підставляючи розпізнані символи повідомлення T_2. І так далі. Діючи таким чином, зловмисник якщо навіть не прочитає обидва повідомлення, то значно зменшить простір їх пошуку.

Опис ходу експерименту

Шифрування даних за допомогою датчика псевдовипадкових чисел (ПВЧ). Лінійні конгруентні датчики ПВЧ.

Щоб отримати лінійні послідовності елементів гами, довжина яких не перевищує розмір шифрованих даних, використовують датчики ПВЧ. Одним з хороших конгруентних генераторів є лінійний конгруентний датчик ПВЧ. Він виробляє послідовності псевдовипадкових чисел Т (i), що описуються співвідношенням:

T (i + 1) = (A * T (i) + C) mod M,

де А і С - константи, Т (0) - почакова величина, М - модуль (натуральне число, відносно якого обчислюють остачу від ділення).

Очевидно, що ці три величини і утворюють ключ. Такий датчик ПВЧ генерує псевдовипадкові числа з певним періодом повторення, що залежать від вибраних значень А і С. Значення М зазвичай встановлюється рівним 2^b, де b-довжина машинного слова в бітах. Необхідно вибирати числа А і С так, щоб період М був максимальним. Як показано Д.Кнуттом, лінійний конгруентний датчик має максимальну довжину М тоді, коли С непарне і А mod 4 = 1.

Як приклад використання лінійного конгруентного датчика ПВЧ розглянемо процес шифрування вихідного тексту "абв". Нехай b = 5, тоді відповідно до номером алфавіті: буква "а" має двійковий код 00001; буква "б" має двійковий код 00010; буква "в" має двійковий код 00011. Вхідний текст буде представлений у вигляді послідовності 00001 00010 00011. Для формування гами шифру виберемо параметри датчика ПВЧ: А = 5; С = 3; Т (0) = 7; $M = 2^b$; b = 5; $M = 2^5 = 32$. Сформуємо три псевдовипадкових числа:

Т (1) = (5 * 7 + 3) mod 32 = 6 (00110);
Т (2) = (5 * 6 + 3) mod 32 = 1 (00001);
Т (3) = (5 * 1 + 3) mod 32 = 8 (01000).

Отримана гамма шифру 00110 00001 01000. Зашифрований текст виходить шляхом накладення гами шифру на вихідний текст (шляхом додавання за модулем 2):

00001 00010 00011
00110 00001 01000

00111 00011 01011

що відповідає шифрограмі "жвк", "ж" (сьома буква в алфавіті) має код 00111, "в" (третя буква в алфавіті) має код 00011, "к" (одинадцята буква в алфавіті) має код 01011. Дешифрування проводиться шляхом накладення тієї ж гами на зашифрований текст:

00001 00010 00011
00110 00001 01000

_____
00111 00011 01011

Метод гамування зі зворотним зв'язком

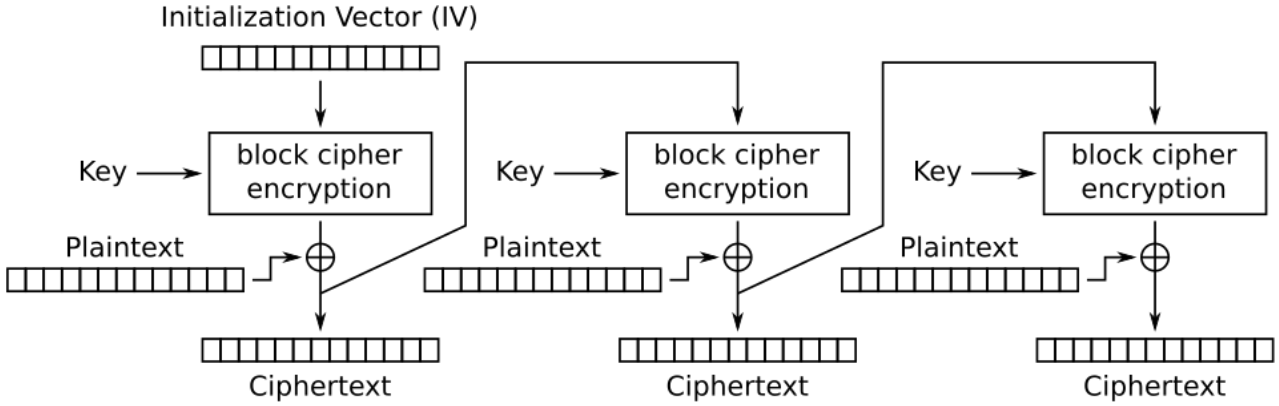
Метод гамування зі зворотним зв'язком (англ. Cipher Feedback Mode, CFB) — один із варіантів використання симметричного блочного шифру, за якого для шифрування наступного блоку відкритого тексту він складається за модулем 2 (XOR) із перешифрованим (блоковим шифром) результатом шифрування попереднього блоку.
Шифрування може бути описано таким чином:

$$C_0 = IV$$

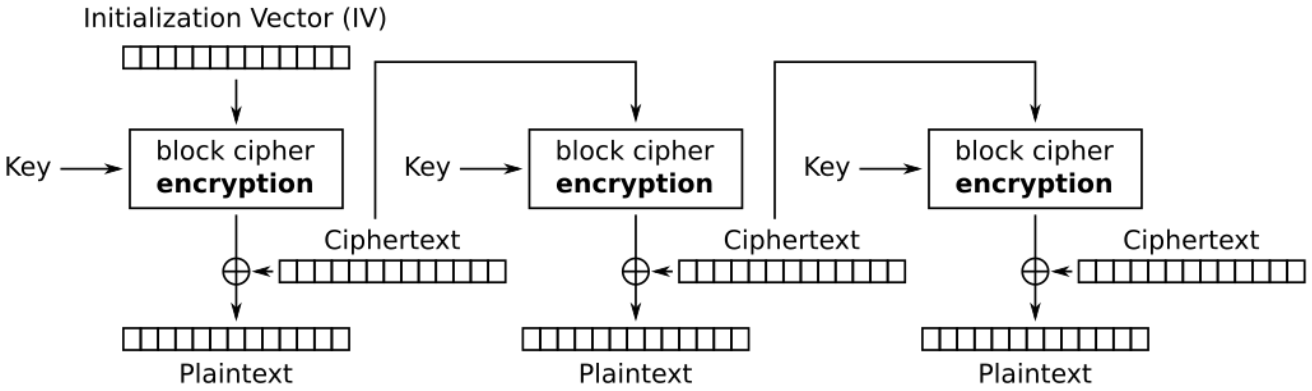
$$C_i = E_k\left(C_{i-1}\right) \oplus P_i$$

$$P_i = E_k\left(C_{i-1}\right) \oplus C_i$$

де ***i*** - номери блоків, IV - вектор ініціалізації (початкове значення шифротексту), C_i і P_i - блоки зашифрованого і відкритого текстів відповідно, а E_k - функція блочного шифрування.



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

Зашифруємо вхідний текст "абв", представлений у вигляді послідовності 00001 00010 00011.
Нехай A = 5; C = 3; b = 5; M = 32; T (0) = 7, IV = 4.
Тоді:
T (1) = (5 * 7 + 3) mod 32 = 6 (00110).
C(0) = IV = 4 (00100)
C(1) = (T(1) ⊕ C(0)) ⊕ P(1) = (6(00110) ⊕ 4(00100)) ⊕1(00001) = **3(00011)**
T (2) = (5 * 6 + 3) mod 32 = 1 (00001).
C(2) = (T(2) ⊕ C(1)) ⊕ P(2) = (1(00001) ⊕ 3(00011)) ⊕2(00010) = **0(00000)**
T (3) = (5 * 1 + 3) mod 32 = 8 (01000).
C(3) = (T(3) ⊕ C(2)) ⊕ P(3) = (8(01000) ⊕ 0(00000)) ⊕3(00011) = **11(01011)**
Отримана шифрограма **00011 00000 01011**

Варіанти завдань

№	Вид генератора ПВЧ	Кількість розрядів b
1	Лінійні конгруентні датчики ПВЧ	6
2	Гамування зі зворотним зв'язком	7
3	Лінійні конгруентні датчики ПВЧ	8
4	Гамування зі зворотним зв'язком	6
5	Лінійні конгруентні датчики ПВЧ	7
6	Гамування зі зворотним зв'язком	8
7	Лінійні конгруентні датчики ПВЧ	6
8	Гамування зі зворотним зв'язком	7
9	Лінійні конгруентні датчики ПВЧ	8
10	Гамування зі зворотним зв'язком	6
11	Лінійні конгруентні датчики ПВЧ	7
12	Гамування зі зворотним зв'язком	8
13	Лінійні конгруентні датчики ПВЧ	6
14	Гамування зі зворотним зв'язком	7
15	Лінійні конгруентні датчики ПВЧ	8
16	Гамування зі зворотним зв'язком	6
17	Лінійні конгруентні датчики ПВЧ	7
18	Гамування зі зворотним зв'язком	8
19	Лінійні конгруентні датчики ПВЧ	6
20	Гамування зі зворотним зв'язком	7

21	Лінійні конгруентні датчики ПВЧ	8
22	Гамування зі зворотним зв'язком	6
23	Лінійні конгруентні датчики ПВЧ	7
24	Гамування зі зворотним зв'язком	8

Домашнє завдання

1. Реалізувати шифрування та дешифрування тексту в режимі однократного гамування з іншими параметрами генератора.
2. Дослідити вплив повторного використання ключа на безпеку шифрування.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. Яка математична формула визначає роботу лінійного конгруентного датчика псевдовипадкових чисел (ПВЧ)? Які параметри впливають на його якість?
2. Чому лінійні конгруентні датчики ПВЧ вважаються недостатньо надійними для криптографічних цілей? Які існують методи покращення їх стійкості?
3. Що таке метод гамування зі зворотним зв'язком (Feedback Ciphering)? У чому полягає його відмінність від звичайного гамування?
4. Як формується гамма у методі гамування зі зворотним зв'язком? Яка роль шифрувального блоку у цьому процесі?

Література

1. Борщенко І. О. Основи криптографії: навчальний посібник. – Київ: КНТ, 2018. – 256 с.
2. Шеннон К. Теорія комунікації в секретних системах / К. Шеннон // Bell System Technical Journal. – 1949. – Т. 28, №4. – С. 656–715.
3. Станойлович Д. Сучасні методи шифрування: від симетричних до асиметричних алгоритмів. – Харків: Фоліо, 2020. – 320 с.
4. Столін В. О. Потокові шифри та їх застосування у захисті інформації: монографія. – Львів: Вид-во ЛНУ, 2019. – 198 с.
5. Брус М. Гамування в криптографії: теоретичні основи та практичне використання / М. Брус, О. Коваль // Журнал сучасних інформаційних технологій. – 2021. – Т. 15, №2. – С. 112–119.
6. Євдокимов М. В. Генератори псевдовипадкових чисел для криптографічних застосувань. – Дніпро: УДХТУ, 2022. – 164 с.

Лабораторна робота №3 потоковий шифр RC4

Мета роботи: Вивчення алгоритму потокового шифру RC4. Вивчення програмної реалізації шифру RC4.

Короткі теоретичні відомості

У криптографії RC4 (Rivest Cipher 4, також відомий як ARC4 або ARCFOUR, що означає Alleged RC4) є потоковим шифром. RC4 є найбільш широко розповсюдженим комерційним потоковим шифром, що має застосування в мережевих протоколах таких як SSL, WEP, WPA, а також в Microsoft Windows, Apple OCE, Secure SQL тощо. Він був розроблений в 1987 році Роном Рівестом для компанії RSA Data Security (зараз RSA Security). З тих пір дизайн був комерційною таємницею і був анонімно розміщений в Інтернеті в 1994 році.

RC4 став частиною деяких загальновживаних протоколів і стандартів шифрування, таких як WEP у 1997 році та WPA у 2003/2004 роках для бездротових карт; і SSL у 1995 році та його наступник TLS у 1999 році, до того як його заборонили для всіх версій TLS у 2015 році згідно з RFC 7465 через атаки на RC4, які послаблювали або ламали RC4, що використовувався в SSL/TLS. Основними факторами успіху RC4 у такому широкому спектрі застосувань були його швидкість і простота: ефективні реалізації як у програмному, так і в апаратному забезпеченні було дуже легко розробити.

Завдання на лабораторну роботу

Використовуючи потоковий алгоритм RC4 зашифрувати та дешифрувати текст за допомогою ключа довжину якого взяти із варіанту по списку в журналі.

Побудувати блок-схему алгоритму RC4.

Опис способу обробки результатів

Опис алгоритму RC4

RC4 генерує псевдовипадковий потік бітів (ключовий потік). Як і з будь-яким потоковим шифром, його можна використовувати для шифрування шляхом поєднання з відкритим текстом за допомогою побітового виключного або (XOR); розшифрування виконується таким же чином (оскільки виключне або з певними даними є інволюцією). Це схоже на шифр одноразового блокнота, за винятком того, що використовуються згенеровані псевдовипадкові біти, а не підготовлений потік.

Для генерації ключового потоку шифр використовує секретний внутрішній стан, який складається з двох частин:

Перестановка всіх 256 можливих байтів (позначається як "S" нижче).

Два 8-бітних індексні вказівники (позначаються як "i" та "j").

Перестановка ініціалізується за допомогою ключа змінної довжини, зазвичай від 40 до 2048 біт, використовуючи алгоритм налаштування ключа (Key-scheduling algorithm, KSA). Після

завершення цього процесу потік бітів генерується за допомогою алгоритму псевдовипадкової генерації (Pseudo-random generation algorithm, PRGA).

Опис ходу експерименту

Алгоритм налаштування ключа (KSA)

Алгоритм налаштування ключа використовується для ініціалізації перестановки в масиві "S". "Довжина ключа" визначається як кількість байтів у ключі й може бути в діапазоні $1 \leq$ довжина ключа ≤ 256, зазвичай від 5 до 16 байтів, що відповідає довжині ключа 40–128 біт. Спочатку масив "S" ініціалізується до перестановки за зразком ідентичності. Потім S обробляється протягом 256 ітерацій у подібний спосіб до основного алгоритму PRGA, але також одночасно змішує байти ключа.

Псевдокод:

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Алгоритм псевдовипадкової генерації (PRGA)

Для стільки ітерацій, скільки потрібно, PRGA модифікує стан і виводить байт із ключового потоку. В кожній ітерації PRGA:

інкрементує "i";

знаходить i-й елемент масиву S, S[i], і додає його до "j";

обмінює значення S[i] та S[j], потім використовує суму S[i] + S[j] (за модулем 256) як індекс для отримання третього елемента з S (ключового потоку K нижче);

цей байт XOR-иться з наступним байтом повідомлення для отримання наступного байта шифротексту або відкритого тексту.

Кожен елемент масиву S змінюється з іншим елементом принаймні один раз за кожні 256 ітерацій.

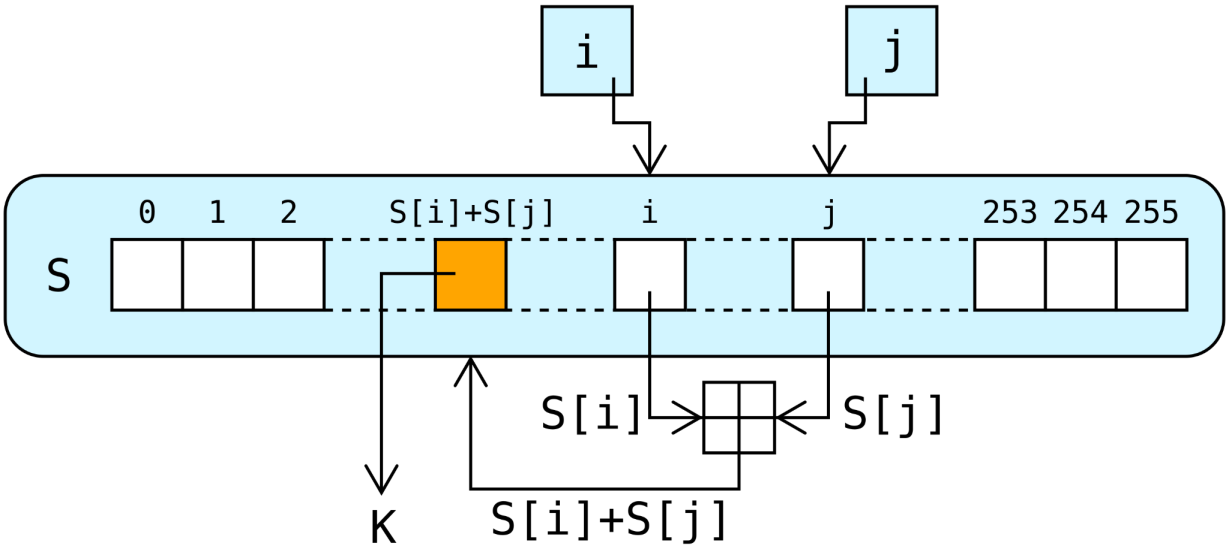


Рис 1. Етап пошуку RC4. Вихідний байт вибирається шляхом пошуку значень S[i] і S[j], додавання їх по модулю 256, а потім використання суми як індексу в S; S(S[i] + S[j]) використовується як байт ключового потоку K.

Псевдокод:

```
i := 0
j := 0
while GeneratingOutput:
```

```
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    t := (S[i] + S[j]) mod 256
    K := S[t]
    output K
endwhile
```

Таким чином, утворюється потік K[0], K[1], ..., які XOR-яться з відкритим текстом для отримання шифротексту. Тому шифротекст[l] = відкритий текст[l] ⊕ K[l].

Варіанти завдань

№	Довжина ключа в байтах
1	5
2	6
3	7
4	8
5	9
6	10
7	11
8	12
9	13
10	14
11	15
12	16
13	17
14	18
15	19
16	20
17	21
18	22
19	23
20	24
21	25
22	26
23	27
24	28

Домашнє завдання

1. Реалізувати алгоритм RC4 для іншої довжини ключа та порівняти результати шифрування.
2. Описати основні відомі вразливості RC4 та причини його обмеженого використання.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. Який принцип лежить в основі роботи потокового шифру RC4? Як формуються ключі для шифрування?
2. Як у RC4 генерується ключовий потік? Яку роль відіграє ініціалізація стану (KSA)?
3. Які основні вразливості RC4 були виявлені в сучасних дослідженнях? Як вони впливають на безпеку алгоритму?
4. У яких криптографічних протоколах використовувався RC4? Чому його поступово витіснили інші алгоритми?
5. Чому вважається небезпечним повторне використання одного ключа у RC4? Як це впливає на криптостійкість?

Література

1. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
2. Fluhrer S., Mantin I., Shamir A. *Weaknesses in the Key Scheduling Algorithm of RC4* // Proceedings of the 8th Annual Workshop on Selected Areas in Cryptography. – 2001. – P. 1–24.
3. Karthikeyan B. *RC4 Stream Cipher and Its Variants* // Cryptography and Security Series. – Springer, 2011. – 256 p.
4. D’Agapeyeff A. *Historical and Modern Uses of RC4: From Success to Vulnerabilities* // Cryptographic Review. – 2015. – Vol. 12, No. 3. – P. 44–59.
5. RFC 7465. *Prohibiting RC4 Cipher Suites* // Internet Engineering Task Force (IETF), 2015.

Лабораторна робота №4 Алгоритм шифрування Магма (ДСТУ 28147:2009)

Мета роботи: ознайомитися з шифруванням і дешифруванням інформації за допомогою алгоритму ДСТУ 28147:2009. Провести порівняння криптостійкості алгоритмів ДСТУ 28147:2009 і DES.

Короткі теоретичні відомості

У країнах СНД в якості стандарту на блокові алгоритми шифрування із закритим ключем був прийнятий алгоритм шифрування Магма. В Україні відомий під назвою ДСТУ ГОСТ 28147:2009 Він рекомендується до використання для криптографічного захисту даних. Шифр побудований за тими ж принципами, що і американський DES, проте в порівнянні є більш зручний для програмної реалізації.

На відміну від американського DES застосовується довший ключ - 256 біт, а також 32 раунду шифрування. Таким чином, основні параметри алгоритму криптографічного перетворення даних Магма наступні:

розмір блоку становить 64 біта,

розмір ключа - 256 біт,

кількість раундів - 32.

Алгоритм являє собою класичну мережу Фейстеля.

В алгоритмі шифрування використовуються наступні операції:

складання слів по модулю 2^{32};

циклічний зсув слова вліво на вказане число біт;

побітове додавання по модулю 2;

заміна по таблиці.

Завдання на лабораторну роботу

Розробити програму для шифрування вхідного тексту за алгоритмом ДСТУ 28147:2009;

Програма має виводити вхідне та зашифроване повідомлення у символьному, байтовому (+padding) і масив з 64 бітних чисел виглядах;

Також вивести значення ключа і ключових елементів k_i;

Скласти блок-схему роботи програми.

Опис способу обробки результатів

Структура одного раунду Магма приведена на Рис. 1.

Вхідний блок даних розбивається на дві частини, які потім обробляються як окремі 32-бітові цілі числа. Спочатку права половина блоку і ключ раунду складаються по модулю 2^{32}. Потім проводиться поблочна підстановка. 32-бітове значення, отримане на

попередньому кроці (позначимо його S), інтерпретується як масив з восьми 4-бітових блоків: $S=(S_0,S_1,S_2,S_3,S_4,S_5,S_6,S_7)$. Далі значення кожного з восьми блоків замінюється на нове, яке вибирається по таблиці замін (S-блок) наступним чином: значення блоку S_i замінюється на S_i-тий по порядку елемент (нумерація з нуля) i-го вузла замін (тобто i-того рядка таблиці замін , нумерація також з нуля). Іншими словами, в якості заміни для значення блоку вибирається елемент з номером рядка, рівним номеру замінного блоку, і номером стовпця, рівним значенню якого замінюють блоку як 4-бітового цілого невід'ємного числа. У кожному рядку таблиці замін записані числа від 0 до 15 в довільному порядку без повторень. Значення елементів таблиці замін взяті від 0 до 15, так як в чотирьох бітах, які піддаються підстановці, може бути записано ціле число без знака в діапазоні від 0 до 15. Наприклад, перший рядок S-блоку може містити такі значення: 5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11. У цьому випадку значення блоку S_0 (чотири молодших біта 32-розрядного числа S) заміниться на число, що стоїть на позиції, номер якої дорівнює значенню замінного блоку. Якщо $S_0 = 0$, то воно буде замінене на 5, якщо $S_0 = 1$, то воно буде замінене на 8 і т.д.

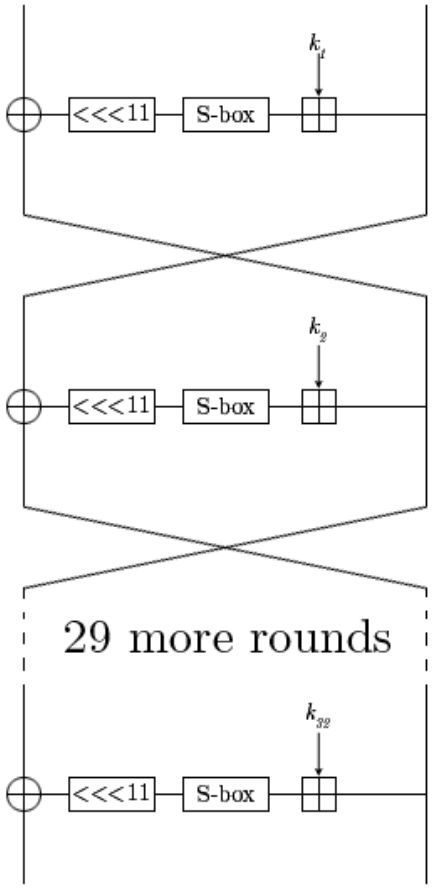


Рис. 1 Структура одного раунду Магма

Номер S-блоку	Значення															
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Рис. 2 Приклад S-блоку

Після виконання підстановки все 4-бітові блоки знову об'єднуються в єдине 32-бітове слово, яке потім циклічно зсувається на 11 бітів вліво. Нарешті, за допомогою побітової операції "сума по модулю 2" результат об'єднується з лівою половиною, внаслідок чого виходить нова права половина R_i. Нова ліва частина L_i береться рівній молодшої частини перетворюється блоку: $L_i = R_{i-1}$.

Отримане значення перетворюється блоку розглядається як результат виконання одного раунду алгоритму шифрування.

Магма є блоковим шифром, тому перетворення даних здійснюється блоками в так званих базових циклах. Базові цикли полягають в багаторазовому виконанні для блоку даних основного раунду, розглянутого нами раніше, з використанням різних елементів ключа і відрізняються один від одного порядком використання ключових елементів. У кожному раунді використовується один з восьми можливих 32-розрядних підключей.

Розглянемо процес створення підключей раундів. У Магма ця процедура дуже проста, особливо в порівнянні з DES. 256-бітний ключ K розбивається на вісім 32-бітових підключей, що позначаються K_0, K_1, K_2,K_3, K_4, K_5, K_6, K_7. Алгоритм включає 32 раунди, тому кожен підключ при шифруванні використовується в чотирьох раундах в послідовності, представленої на таблиці 1.

Таблиця 1. Послідовність використання підключей при шифруванні

Раунд	1	2	3	4	5	6	7	8
Підключ	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
Раунд	9	10	11	12	13	14	15	16
Підключ	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
Раунд	17	18	19	20	21	22	23	24
Підключ	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
Раунд	25	26	27	28	29	30	31	32
Підключ	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

Процес розшифрування проводиться за тим же алгоритмом, що і шифрування. Єдина відмінність полягає в порядку використання підключей K_i. При розшифрування підключи повинні бути використані в зворотному порядку, а саме, як вказано на таблиці 2.

Таблиця 2. Послідовність використання підключей при розшифрування

Раунд	1	2	3	4	5	6	7	8

Підключ	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
Раунд	9	10	11	12	13	14	15	16
Підключ	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0
Раунд	17	18	19	20	21	22	23	24
Підключ	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0
Раунд	25	26	27	28	29	30	31	32
Підключ	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

Опис ходу експерименту

Основні режими шифрування

Магма передбачає такі режими шифрування даних: проста заміна, гамування, гамування зі зворотним зв'язком і один додатковий режим вироблення імітовставки.

У будь-якому з цих режимів дані обробляються блоками по 64 біта, на які розбивається шифруємий масив, саме тому Магма відноситься до блокових шифрів. У режимах гамування є можливість обробки неповного блоку даних розміром менше 8 байт, що істотно при шифруванні масивів даних з довільним розміром, який може бути не кратним 8 байтам.

Режим простої заміни. Цей режим використання блочного шифру аналогічний розглянутому в попередній лабораторній роботі режиму простої поблочної заміни (ECB). У цьому режимі кожен блок вихідних даних шифрується незалежно від інших блоків, із застосуванням одного і того ж ключа шифрування. Особливістю цього режиму є те, що однакові блоки вихідного тексту перетворюються в однаковий шифротекст. Тому Магма рекомендує використовувати режим простої заміни тільки для шифрування ключів. Режими гамування та гамування зі зворотним зв'язком можуть використовуватися для шифрування даних довільного розміру.

У режимі гамування біти вихідного тексту складаються по модулю 2 з гамою, яка виробляється за допомогою алгоритму шифрування по Магма. Тобто алгоритм шифрування по Магма в даному режимі використовується в якості генераторів 64-розрядних блоків гами. При шифруванні кожного нового блоку даних гамма, використана на попередньому кроці, зашифрована і використовується вже як "нова" гамма. В якості початкового масиву даних, для отримання найпершої гами, використовується 64-розрядний початковий блок даних, який повинен бути однаковим на стороні шифрування та розшифрування. Завдяки тому, що накладання та зняття гами здійснюється за допомогою однієї і тієї ж операції додавання по модулю 2, алгоритми шифрування і розшифрування в режимі гамування збігаються.

Так як всі елементи гами різні для реальних шифруємих масивів, то результат шифрування навіть двох однакових блоків в одному масиві даних буде різним. Крім того, хоча елементи гами і виробляються однаковими порціями в 64 біта, використовуватися може і частина такого блоку з розміром, рівним розміром шифруємого блоку. Саме це дає можливість шифрування неповних блоків даних.

Режим гамування зі зворотним зв'язком схожий на режим гамування і відрізняється від нього тільки способом вироблення елементів гами. При гамування зі зворотним зв'язком черговий 64-бітний елемент гами виробляється як результат перетворення за базовим циклом алгоритму Магма попереднього блоку зашифрованих даних. Цим досягається зачеплення блоків - кожен блок шифротексту в цьому режимі залежить від відповідного і всіх попередніх блоків відкритого тексту. Тому даний режим іноді називається гамуванням із зчепленням блоків. На стійкість шифру факт зачеплення блоків не робить ніякого впливу.

Для вирішення завдання виявлення спотворень в зашифрованому масиві даних в Магма передбачений додатковий режим криптографічного перетворення - вироблення

імітовставки.

Імітовставка - це контрольна комбінація, що залежить від відкритих даних і секретної ключової інформації. Метою використання імітовставки є виявлення всіх випадкових або навмисних змін в масиві інформації. У фазі приготування імітовставки вхідний текст обробляється блоками наступним чином:

$$Y = f((X_{ii-1})), K) \text{ для всех } i \text{ от } 1 \text{ до } n$$

де f - базовий цикл по Магма; X_i - 64-розрядний блок вихідного тексту; K - ключ.

У якості імітовставки береться частина блоку Y_n, отриманого на виході, зазвичай 32 його молодших біта. Таким чином, зловмисник, не володіючи ключем шифрування, не може обчислити імітовставку для заданого відкритого масиву інформації, а також підібрати відкриті дані під задану імітовставка.

Відмінності алгоритмів шифрування по Магма і DES

Незважаючи на те, що алгоритм, викладений в Магма, проектувався досить давно, в нього закладено достатній запас по надійності. Це пов'язано, перш за все, з великою довжиною ключа шифрування.

Як відомо, розробники сучасних криптосистем дотримуються принципу, що секретність зашифрованих повідомлень повинна визначатися секретністю ключа. Це означає, що навіть якщо сам алгоритм шифрування відомий криптоаналітику, той, проте, не повинен мати можливості розшифрувати повідомлення, якщо не має в своєму розпорядженні відповідним ключем. Всі класичні блочні шифри, в тому числі DES і Магма, відповідають цим принципом і спроектовані таким чином, щоб не було шляху розкрити їх більш ефективним способом, ніж повним перебором по всьому ключовому простору, тобто по всіх можливих значеннях ключа. Ясно, що стійкість таких шифрів визначається розміром використовуваного в них ключа.

У шифрі, що реалізується в Магма, використовується 256-бітовий ключ, і обсяг ключового простору становить 2^{256}. Навіть якщо, як і в "Алгоритми шифрування DES і AES", Припустити, що на злом шифру кинуті всі сили обчислювального комплексу з можливістю перебору 10^{12} (це приблизно дорівнює 2^{40}) ключів в одну секунду, то на повний перебір всіх 2^{256} ключів потрібно 2^{216} секунд (цей час становить понад мільярд років).

До вже зазначених відмінностей між алгоритмами DES і Магма можна додати також наступне. В основному раунді DES застосовуються нерегулярні перестановки вихідного повідомлення, в Магма використовується 11-бітний циклічний зсув вліво. Остання операція набагато зручніше для програмної реалізації. Однак перестановка DES збільшує лавинний ефект. У Магма зміна одного вхідного біта впливає на один 4-бітовий блок при заміні в одному раунді, який потім впливає на два 4-бітових блоку наступного раунду, три блоки наступного і т.д. У Магма потрібно 8 раундів перш, ніж зміна одного вхідного біта вплине на кожен біт результату; DES для цього потрібно тільки 5 раундів.

Також слід зазначити, що на відміну від DES, у Магма таблицю замін для виконання операції підстановки можна довільно змінювати, тобто таблиця замін є додатковим 512-бітовим ключем.

Домашнє завдання

1. Змінити значення S-блоків або ключа та проаналізувати вплив на результат шифрування.
2. Порівняти основні характеристики алгоритмів Магма та DES.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів

7. висновки.

Контрольні запитання

1. Що таке алгоритм Магма, і яку роль він виконує в забезпеченні інформаційної безпеки?
2. Як у структурі алгоритму Магма реалізована мережа Фейстеля? У чому її переваги для даного алгоритму?
3. Яку функцію виконують таблиці підстановок (S-блоки) в алгоритмі Магма? Як їх правильний вибір впливає на криптостійкість?
4. Який розмір ключа та блоку даних використовується в алгоритмі Магма? Як це впливає на безпеку та ефективність алгоритму?
5. У яких випадках рекомендовано використовувати алгоритм Магма? Як він забезпечує відповідність сучасним вимогам інформаційної безпеки?

Література

1. ДСТУ 28147:2009. *Система криптографічного захисту інформації. Процеси перетворення інформації.* – Київ: Держспоживстандарт України, 2009.
2. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
3. Кучерявенко В. А. *Криптографія: основи і застосування*. – Харків: ХНУРЕ, 2017. – 356 с.
4. Брус М. Г., Коваль О. І. *Розробка та аналіз стійкості таблиць підстановок для ДСТУ 28147:2009* // Журнал сучасних інформаційних технологій. – 2020. – Т. 15, №3. – С. 34–49.

Лабораторна робота №5 Стандарт шифрування Advanced Encryption Standard (AES)

Мета роботи: Вивчити алгоритм роботи симетричного блокового шифрування AES. Вивчити алгоритми обробки ключа

Короткі теоретичні відомості

Rijndael, AES це федеральний стандарт шифрування США. Був розроблений в 1997 році, в Бельгії Йоаном Дамен (Joan Daemen) Вінсентом Райменом (Vincent Rijnmen). Представляє собою архітектуру «Квадрат». Параметри:

розмір блоку, біт	128, 192, 256(1)
розмір ключа, біт	128, 192, 256
число раундів	10, 12, 14(2)
розмір ключового елемента, біт	128, 192, 256 (дорівнює розміру блоку)
число ключових елементів	11, 13, 15 (на 1 більше числа раундів)

Так як алгоритм може бути сформульований в термінах всього лише двох операцій, - побітового підсумовування по модулю 2 і індексованого вилучення з пам'яті, виконуваних над байтами, - він може бути ефективно реалізований на будь-яких комп'ютерних платформах від молодших мікроконтролерів до супер процесорів. В силу тих же причин, а також тому що архітектура алгоритму допускає високий ступінь паралелізму, він може бути також дуже ефективно реалізований в апаратурі. В кінцевому підсумку саме наведені вище фактори вкупі з високою стійкістю алгоритму визначили його перемогу в конкурсі на місце нового стандарту.

2. Пряме і зворотне перетворення в шифрі мають однакову алгоритмічну структуру і різняться константами зсуву, ключовими елементами, вузлами замін і константами множення. При апаратної реалізації вони можуть бути суміщені на 60%, при програмної оптимальне швидкодію може бути досягнуто лише при повністю роздільних реалізаціях обох функцій.

(1) Як стандарт прийнятий варіант шифру тільки з розміром блоку 128 біт.

(2) Число раундів шифрування визначається в залежності від розміру блоку і ключа по наступній таблиці:

розмір ключа / розмір блоку	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

Іншими словами, з двох розмірів вибирається максимальний, і якщо він дорівнює 128 біт, то використовується 10 раундів, якщо 192 біта, то 12, і якщо 256 - то 14 раундів шифрування.

Завдання на лабораторну роботу

Реалізувати алгоритм AES без використання сторонніх бібліотек, програма має приймати текст довільної довжини і шифрувати його за алгоритмом AES. Розмір блоку по стандарту AES = 128 біт. Розмір ключа вибрати з таблиці варіантів.

Опис способу обробки результатів

Шифр Rijndael виконаний в архітектурі "Квадрат" (Square), що отримала свою назву від першого побудованого відповідно до її принципами криптоалгоритма. У Rijndael Блоки відкритих і зашифрованих даних, відповідно T і T ', представляються у вигляді масивів з 16, 24 або 32 байтів:

$T = (t_1, t_2,...,t_N)$

$T' = (t'_1, t'_2,...,t'_N)$

$| t | = | t' | = 8$, $N \in \{16, 24, 32\}$.

Відповідно до використаними архітектурними принципами в ході криптографічних перетворень вихідний і зашифрований блоки даних, а також всі проміжні результати процесу шифрування інтерпретуються як матриці байтів розміром $4 \times n$, Звідки отримуємо $n = N/4$, $n \in \{4, 6, 8\}$. Матриці заповнюються байтами вхідного блоку (відкритих даних при зашифрованими і зашифрованих даних при розшифрування відповідно) по стовпцях зверху вниз і зліва направо, і в точно такому ж порядку беруться байти з матриці-результату:

$$T = \begin{bmatrix} t_1 & t_5 & \dots & t_{N-3} \\ t_2 & t_6 & \dots & t_{N-2} \\ t_3 & t_7 & \dots & t_{N-1} \\ t_4 & t_8 & \dots & t_N \end{bmatrix}, \quad T' = \begin{bmatrix} t'_1 & t'_5 & \dots & t'_{N-3} \\ t'_2 & t'_6 & \dots & t'_{N-2} \\ t'_3 & t'_7 & \dots & t'_{N-1} \\ t'_4 & t'_8 & \dots & t'_N \end{bmatrix}.$$

Схема перетворення даних при шифруванні показана на рис. 1, схема відповідного алгоритму - на рис. 2.

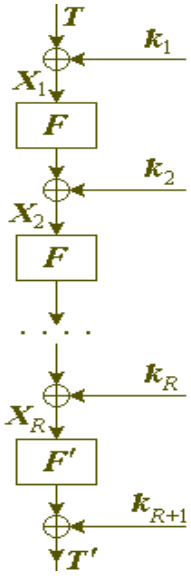


Рис. 1. Цикл шифрування Rijndael - схема перетворення даних.

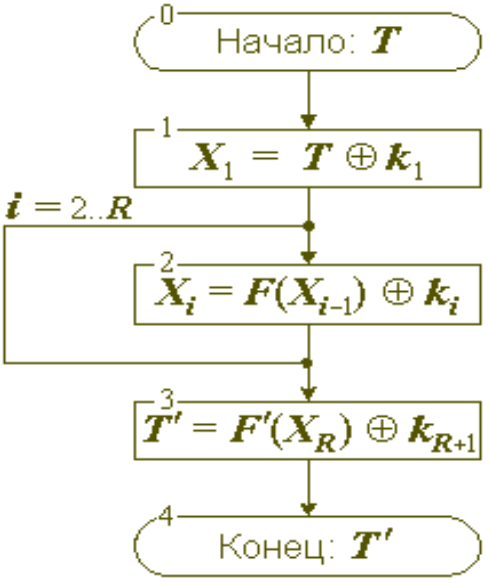


Рис. 2. Цикл шифрування Rijndael - схема алгоритму.

На рисунках використані наступні позначення:

T, T'- відкритий і зашифрований блоки даних відповідно;

k_i- і-тий ключовий елемент;

F, F'- регулярне нелінійне перетворення і перетворення останнього раунду відповідно;

X_i- проміжний стан шифруємого блоку після додавання і-того ключового елемента.

Як видно з рисунків 1 і 2, процес шифрування складається з чергуючих додатків ключових елементів до блоку даних і нелінійного перетворення цього блоку:

$T' = E_K(T) = k_{R+1} \oplus F'(k_R \oplus F(k_{R-1} \oplus \ldots F(k_2 \oplus F(k_1 \oplus T))\ldots))$.

Число R раундів шифрування змінне і залежить від розміру блоку даних і ключа. Додаток ключових елементів, яким починається і закінчується процес шифрування, а також деякі інші операції раундового перетворення виконується побайтно в кінцевому поле Галуа GF (2^8), операцією додавання в ньому є побітовое сумування по модулю 2. Відповідно, кожен ключовий елемент є байтовою матрицею того ж самого розміру, що й блок даних. За один раунд шифрування перетвориться повний блок даних, а не його частина, як в мережах Файстеля. На останньому раунді функція нелінійного перетворення відрізняється від аналогічної функції, використовуваної в інших раундах - це зроблено для забезпечення алгоритмічної еквівалентності прямого і зворотного перетворень шифрування.

Процес розшифрування блоку даних алгоритмічно ідентичний процесу його шифрування і, отже, малюнки 1 і 2 також справедливі і для нього, якщо через T позначити блок зашифрованих даних, а через T '- відкритих. Однак відмінності між цими двома процедурами в архітектурі "Квадрат" кілька більш істотні, ніж в мережах Файстеля - вони розрізняються не тільки порядком використання ключових елементів в раундах шифрування, але і самими цими елементами, і деякими іншими константами, використовуваними в алгоритмі.

Нелінійне перетворення F матриці даних складається з трьох кроків: заміни байтів матриці на нові значення (S []), циклічного зсуву рядків матриці вліво ($R_{\leftarrow}$), Множення матриці даних зліва на постійну матрицю-циркулянт M:

$X' = F(X) = M \times R_{\leftarrow}(S(X))$.

Схема перетворення даних показана на рис. 3, схема відповідного алгоритму - на рис. 4.

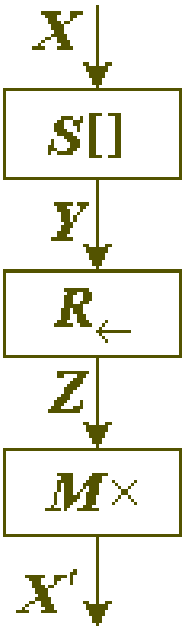


Рис. 3. Схема нелінійного перетворення блоку даних.

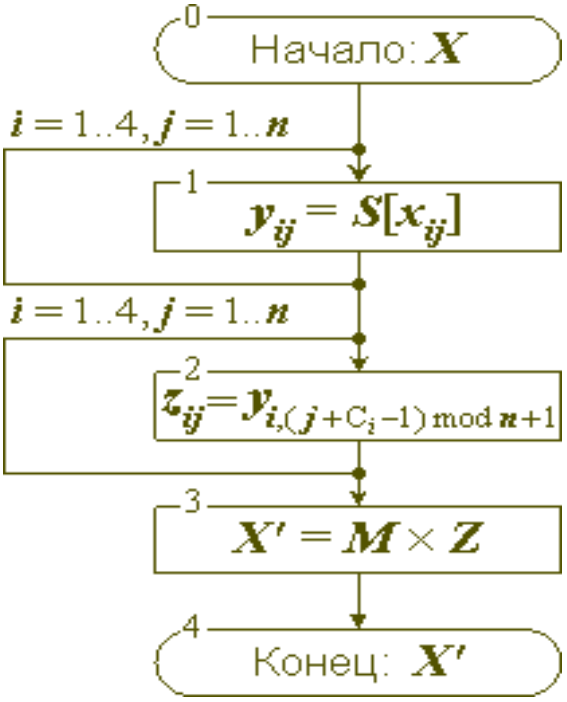


Рис. 4. Схема алгоритму нелінійного перетворення.

Всі вхідні (X), вихідні (X ') і проміжні (Y, Z) значення перетворення є матрицями байтів однакового розміру $4\times n$. Функція перетворення останнього раунду F' відрізняється від регулярної функції перетворення F відсутністю кроку множення матриці даних зліва на постійну матрицю, схема перетворення блоку даних на останньому раунді і схема відповідного алгоритму наведені на рисунках 5 і 6 відповідно:

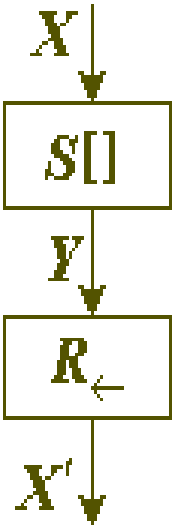


Рис. 5. Схема нелінійного перетворення останнього раунду.

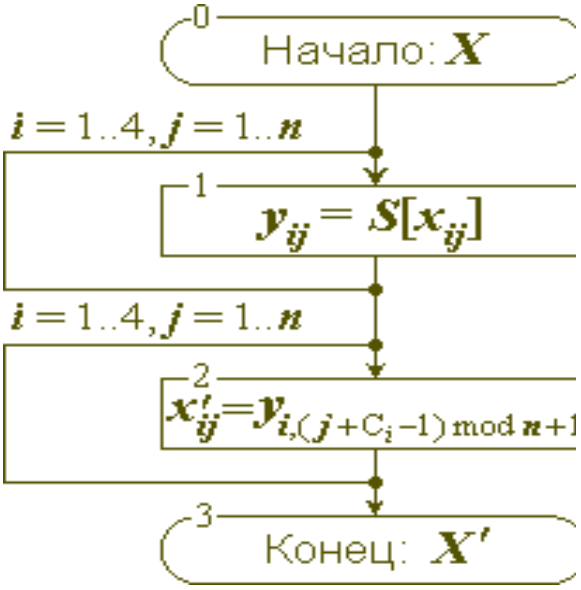


Рис. 6. Схема алгоритму нелінійного перетворення останнього раунду.

Вся нелінійність перетворення зосереджена в його першому кроці - заміні, другий і третій кроки є лінійними. Перший крок служить для перемішування інформації всередині байтів, другий забезпечує "експорт" змін в інші стовпці, третій здійснює дифузію змін в одному елементі матриці на весь відповідний стовпець. Таким чином, за два раунди досягається дифузія змін в одному єдиному біті на весь блок даних. Нижче кожен з цих дій розглянуто детально, при цьому деякі перетворення байтів визначені в термінах операцій в кінцевому полі $GF(2^8)$, породженому неприводним поліномом $m(x)$ над полем $GF(2)$: $m(x) = x^8+x^4+x^3+x+1$. Операція складання в цьому полі є ні чим іншим, як побітовим підсумовуванням за модулем 2, множення у відповідність з визначенням поля виконується як звичайне множення поліномів над GF(2) по модулю полінома $m(x)$. При маніпулюванні з

байтами даних як з елементами поля GF(2^8) кожен біт відповідає доданку виду x^i відповідно до старшинством біта в байті. Можна сказати, що якщо байт з цілочисельним значенням b представлений у вигляді полінома *B(x)*, то справедливо b = *B(2)*.

Побайтовая заміна.

В ході побайтової заміни кожен байт матриці даних замінюється на нове значення того ж розміру, індексуючи загальний для всіх байтів вектор замін S 8-в-8 біт:

y_{ij}= S [x_{ij}], $1 \leqslant i \leqslant 4$, $1 \leqslant j \leqslant n$,

де n - число стовпців матриці даних - 4,6 або 8. Єдиний вузол замін в шифрі Rijndael конструюється за допомогою наступного алгебраїчного співвідношення:

$S[y] = (x^4+x^3+x^2+x+1) + y^{-1} \cdot (x^7+x^6+x^5+x^4+1) \bmod (x^8+1)$.

При цьому звернення ненульових байтів здійснюється в описаному вище кінцевому полі GF (2^8), для нульового байта вважають 0-1 = 0. Таким чином, байтова заміна визначається як звернення елемента-байта в кінцевому полі GF (2^8), до визначення для нульового елемента поля, з подальшим аффінним перетворенням результату. Коефіцієнти цього перетворення обрані таким чином, щоб у отриманого вузла замін були відсутні точки нерухомості ($S[y] = y$), і "антинерухомості" ($S[y] = \sim y$). Тільда (знаком "~") позначена операція побітового доповнення свого аргументу.

Природно, зазначена вище формула для побудови вузла замін не призначена для використання безпосередньо під час шифрування - набагато ефективніше використовувати вже готовий вузол замін, наведений в наступній таблиці:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
Ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
Bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
Cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
Dx	70	3e	b5	66	48	03	f6	0E	61	35	57	b9	86	c1	1d	9e
Ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
Fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Значення, яке замінюється вибирається на перетині рядка, яка визначається старшої 16-річної цифрою замінного значення, і стовпчика, що визначається його молодшої цифрою.

Порядкове обертання матриці.

В ході даної операції кожен рядок матриці даних, крім першої, обертається (циклічно зсувається) вліво на певне число позицій, залежне від номера рядка і від розміру блоку даних:

$$z_{ij} = y_{i,(j+C_i-1) \bmod n+1}, \quad 1 \leqslant i \leqslant 4, \quad 1 \leqslant j \leqslant n.$$

Перший рядок завжди залишається на місці: $C_1 = 0$, для неї наведена вище формула істотно спрощується: $z_{1j} = y_{1j}$. Нижче в таблиці наведено величини зсуву для рядків матриці з другого по четвертий в залежності від числа стовпців n в матриці:

n	4	6	8
C_2	1	1	1
C_3	2	2	3
C_4	3	3	4

Множення на постійну матрицю.

На цьому кроці матриця даних зліва множиться на постійну матрицю-циркулянт M:

$X = M \times X$,

$$\boldsymbol{M} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}.$$

При виконанні матричного множення операції додавання і множення елементів обох матриць виконуються в кінцевому полі GF (2^8). Матриця M є циркулянт: кожна її рядок виходить циклічним зрушенням попереднього рядка вправо на один елемент. Елементи матриці обрані таким чином, щоб звести до мінімуму трудомісткість операції множення: в ній присутні лише невеликі за значенням числа 01, 02 і 03, половина елементів - поодинокі, тобто реального множення виконувати для них не потрібно. Цим самим забезпечується висока ефективність можливих реалізацій цієї операції.

Слід додати, що операція множення в кінцевому полі GF (2^8) є досить трудомісткою в програмній реалізації і ніяким чином не зводиться до звичайного арифметичному множенню. Якщо множення двійкових чисел реалізується зрушеннями і звичайним арифметичним підсумовуванням, то множення поліномів над полем GF(2) - тими ж зрушеннями і побітовим підсумовуванням за модулем 2. Однак в шифрі Rijndael одним з множників завжди є константа і розмір операндів невеликий - один байт. Це дозволяє реалізувати множення на константу в поле GF (2^8) як заміну, що істотно підвищує ефективність програмної реалізації. Для кожного множника-константи при цьому потрібно свій окремий вузол замін. навпаки,

Розшифрування в Rijndael алгоритмічно еквівалентно зашифрування, проте між цими двома процедурами є певні відмінності, набагато більш суттєві, ніж в мережах Файстеля, де все зводиться до порядку використання ключових елементів. Розшифрування відрізняється від шифрування за такими чотирма пунктами:

1. Ключові елементи використовуються в порядку, зворотному тому, в якому вони використовуються при зашифрованими. Крім того, всі ключові елементи, крім першого і останнього, повинні бути помножені зліва на матрицю, зворотну матриці M. Таким чином, якщо при шифруванні використовується наступна послідовність ключових елементів

$k_1, k_2, k_3, \ldots, k_R, k_{R+1}$,

то при розшифруванні повинна бути використана наступна послідовність елементів:

$k_{R+1}, M^{-1} \times k_R, \ldots, M^{-1} \times k_3, M^{-1} \times k_2, k_1$.

2. На етапі побайтової заміни використовується вузол замін S^{-1} обернений тому, який використовується в процедурі зашифровування S. Це значить, що яке б не було байтове значення b, завжди справедливо наступне співвідношення:

$S^{-1}[S[b]] = b$.

Вказаний вузол замін S^{-1} надано в наступній таблиці, значення наведені в 16-річному форматі:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
Ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
Bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
Cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
Dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
Ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
Fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

3. На кроці порядкового обертання матриці даних здійснюється циклічний зсув строк на ту ж саму кількість елементів, що і при зашифрованими, але в зворотну сторону - вправо. Або, в силу властивостей операції циклічного зсуву, можна здійснити обертання рядків матриці в ту ж сторону, що і при шифруванні, тобто вліво, але на іншу кількість елементів, що обчислюється за такою формулою:

$C_i' = n - C_i$, $2 \leqslant i \leqslant 4$.

Константи циклічного зсуву вліво рядків матриці в процедурі розшифрування в залежності від числа стовпців матриці даних наведені в наступній нижче таблиці:

n	4	6	8
C_2'	3	5	7
C_3'	2	4	5
C_4'	1	3	4

4.На кроці множення зліва на постійну матрицю використовується матриця M^{-1}, зворотна використовуваної при зашифрованими матриці M:

$$\boldsymbol{M}^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}.$$

Множення в кінцевому полі GF (2^8) на елементи матриці M^{-1} з точки зору обчислювальних витрат є більш трудомісткою операцією, ніж множення на елементи матриці M. Крім того, в зворотному матриці присутні чотири різних елемента, тоді як у вихідній - тільки три, що дозволяло "заощадити" одне множення з чотирьох. При безпосередньої реалізації множення в полі GF(2^8) модулі розшифрування виходять помітно менше швидкодіючими, ніж модулі шифрування. Однак, ця особливість не є настільки суттєвою, як може здатися на перший погляд. По-перше, в більшості практичних режимів використання шифру застосовується тільки пряме перетворення (зашифрування) - подібна ситуація має місце при шифруванні з використанням потокових режимів (в тому числі і при розшифрування), при виробленні імітовставки (коду автентифікації), при виробленні хеш-функції і при виробленні масивів псевдовипадкових даних. По-друге, якщо множення на константу в поле GF (2^8) реалізувати як заміну, відмінності в трудомісткості нівелюються.

Опис ходу експерименту

Огляд процесу розширення ключа

Вхід: початковий ключ (128, 192 або 256 біт)

Вихід: серія кругових ключів, один для кожного раунду, включаючи початковий круговий ключ.

AES-128: Генерує 11 кругових ключів (10 раундів + 1 початковий круговий ключ) з 4x4 матриці (16 байтів).

AES-192: Генерує 13 кругових ключів з 24-байтового ключа.

AES-256: Генерує 15 кругових ключів з 32-байтового ключа.

Кроки розширення ключа для AES-128

Зважаючи на 128-бітний ключ (16 байтів), ключ ділиться на 4 32-бітові слова. Алгоритм генерує 44 слова загалом для AES-128, при цьому кожен круговий ключ складається з 4 слів (16 байтів). Перші 4 слова — це початковий ключ, а решта 40 слів генеруються шляхом трансформацій.

$W[i]=W[i-4] \oplus \text{Transform}(W[i-1])$

1. Поділ ключ на слова

AES-128 використовує 4x4 байтову матрицю, де кожен ряд представляє частину ключа.

Key=[K0,K1,K2,K3]
Де кожне Ki є 32-бітним словом (4 байти).
2. Генерація додаткових слів за допомогою рекурентної формули
Для AES-128 процес розширення ключа слідує цим правилам:
Якщо i mod 4=0:
$W[i]=W[i-4]\oplus SubWord(RotWord(W[i-1]))\oplus Rcon[i/4]$
В іншому випадку: $W[i]=W[i-4]\oplus W[i-1]$
3. Операції SubWord, RotWord та Rcon
RotWord:
Обертає 32-бітне слово вліво на один байт (8 біт).
Приклад:
RotWord(0x09CF4F3C)=0xCF4F3C09
SubWord:
Застосовує S-Box заміну до кожного байта слова.
Приклад:
SubWord(0xCF4F3C09)→S-Box(0xCF),S-Box(0x4F),S-Box(0x3C),S-Box(0x09)
Rcon (Константа раунду):
Попередньо визначена константа, яка використовується для забезпечення унікальності кожного кругового ключа. Вона базується на степенях 2 у $GF(2^8)$.
Приклад: Rcon[1]=0x01,Rcon[2]=0x02,Rcon[3]=0x04,Rcon[4]=0x08
Значення Rcon використовуються тільки для слів, де i mod 4=0.
4. Приклад розширення ключа для AES-128
Припустимо, початковий 128-бітний ключ такий:
Key=[2B,7E,15,16,28,AE,D2,A6,AB,F7,CF,15,88,09,CF,4F]
Поділ ключ на 4 слова:
W[0]=2B7E1516,W[1]=28AED2A6,W[2]=ABF7CF15,W[3]=8809CF4F
Генерація W[4]:
$W[4]=W[0]\oplus SubWord(RotWord(W[3]))\oplus Rcon[1]$
W[3]=8809CF4F→RotWord(W[3])=09CF4F88
Підстановка кожного байта за допомогою S-Box: SubWord(09CF4F88)=54D9902C
Rcon[1]=0x01000000
Тепер: $W[4]=2B7E1516\oplus 54D9902C\oplus 01000000=7E94751A$
Генерація W[5]:
$W[5]=W[4]\oplus W[1]=7E94751A\oplus 28AED2A6=56B327BC$
Повторюємо цей процес для решти слів, поки не буде згенеровано всі 44 слова.
5. Формування кругових ключів
Кожен круговий ключ складається з 4 слів (16 байтів). Для AES-128 11 кругових ключів отримуються з 44 слів:
Ключ раунду 0=[W0,W1,W2,W3]
Ключ раунду 1=[W4,W5,W6,W7]
…
Ключ раунду 10=[W40,W41,W42,W43]

Принцип роботи MixColumns
На етапі MixColumns кожен стовпець матриці стану обробляється окремо. Матриця стану в AES є 4x4 матрицею байтів, де кожен стовпець складається з 4 байтів (32 біти).
Алгоритм MixColumns
Структура:
Стан (state) представляється у вигляді 4 стовпців (C0, C1, C2, C3).
Кожен стовпець обробляється незалежно.
Лінійне перетворення:
MixColumns виконує множення кожного стовпця на поліном у полі Галуа GF(2^8). Основний поліном, який використовується, це: x^4+1
Для AES множення виконується за допомогою специфічної матриці:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Векторне множення:
Кожен стовпець матриці стану (C) множиться на цю матрицю, що забезпечує змішування байтів. Для стовпця C:
$C' = \text{MixColumns}(C) = M \cdot C$
де M — це матриця перетворення.
Приклад обробки стовпця
Припустимо, стовпець виглядає так:

$$C = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Тоді результуючий стовпець C′ буде обчислюватися наступним чином:

$$C' = \begin{bmatrix} (2 \cdot b_0) \oplus (3 \cdot b_1) \oplus (1 \cdot b_2) \oplus (1 \cdot b_3) \\ (1 \cdot b_0) \oplus (2 \cdot b_1) \oplus (3 \cdot b_2) \oplus (1 \cdot b_3) \\ (1 \cdot b_0) \oplus (1 \cdot b_1) \oplus (2 \cdot b_2) \oplus (3 \cdot b_3) \\ (3 \cdot b_0) \oplus (1 \cdot b_1) \oplus (1 \cdot b_2) \oplus (2 \cdot b_3) \end{bmatrix}$$

де множення і додавання є операціями поля Галуа GF(2^8).
Пояснення Галуа-полів (Galois Fields)
Галуа-поля (або полі Галуа) — це алгебраїчні структури, які використовуються в багатьох областях математики і комп'ютерних наук, зокрема в криптографії, теорії кодування та обробці сигналів. Вони названі на честь французького математика Евариста Галуа, який зробив значний внесок у розвиток теорії полів.
Галуа-поле — це поле, яке складається з кінцевої кількості елементів. Основні властивості полів:
Кожен елемент має обернений: для кожного ненульового елемента a існує елемент b, такий що $a \cdot b = 1$.
Виконуються властивості асоціативності, комутативності, дистрибутивності для додавання і множення.
Існує нульовий елемент (позначається 0) і одиничний елемент (позначається 1), які відповідають основним арифметичним операціям.
В Галуа-полях виконуються дві основні операції: додавання і множення.
Додавання:
Додавання в Галуа-полях виконується за модулем 2. Наприклад:
1+1=0 (аналогічно логічному XOR).
0+1=1, 1+0=1, 0+0=0.
Множення:
Множення в Галуа-полях може бути складнішим, оскільки воно включає використання поліномів. Наприклад, у полі GF(2^8) використовується модульний поліном x^8 + x^4 + x^3 + x + 1.
При множенні елементів результати можуть перевищувати максимальний ступінь, тому проводиться модульне зменшення.
Приклад реалізації функції множення в Галуа-полі GF(2^8) на C++:
```
class GaloisField {
```

```
public:
    // Поліном для обчислення модулю
    static const uint8_t POLY = 0x1B; // x^8 + x^4 + x^3 + x + 1

    // Множення в GF(2^8)
    static uint8_t galoisMultiply(uint8_t a, uint8_t b) {
    uint8_t result = 0;
    uint8_t temp = a;

    for (int i = 0; i < 8; i++) {
    // Перевірка, чи i-й біт b є 1
    if (b & 0x01) {
        result ^= temp; // XOR, якщо біт b[i] == 1
    }

    // Множення в полі Галуа (переміщення вліво)
    bool overflow = temp & 0x80; // Перевірка на переповнення
    temp <<= 1; // Зсув вліво
    if (overflow) {
        temp ^= POLY; // XOR з поліном
    }

    // Зсув b вправо для обробки наступного біта
    b >>= 1;
    }
    return result;
    }
};
```

Варіанти завдань

№	Розмір ключа
1	128
2	192
3	256
4	128
5	192
6	256
7	128
8	192
9	256
10	128
11	192
12	256
13	128
14	192
15	256
16	128
17	192
18	256
19	128
20	192
21	256
22	128
23	192
24	256

Домашнє завдання

1. Реалізувати шифрування AES з іншою довжиною ключа (128/192/256 біт).
2. Пояснити переваги AES порівняно з попередніми симетричними алгоритмами.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. Що таке алгоритм AES, і які його основні цілі та функції в криптографії?
2. Який розмір блоку та ключа використовується в AES? Як змінюється кількість раундів залежно від довжини ключа?
3. Які основні етапи шифрування в алгоритмі AES? Опишіть послідовність дій у раунді.
4. Що таке SubBytes, ShiftRows, MixColumns та AddRoundKey? Яку роль вони відіграють у забезпеченні безпеки AES?
5. Які особливості алгоритму AES забезпечують його стійкість до криптоаналітичних атак?

Література

1. National Institute of Standards and Technology (NIST). *Announcing the Advanced Encryption Standard (AES)* // FIPS PUB 197. – Gaithersburg, MD: NIST, 2001.
2. Daemen J., Rijmen V. *The Design of Rijndael: AES – The Advanced Encryption Standard*. – Springer, 2002. – 238 p.
3. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
4. Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
5. Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.

Лабораторна робота №6 Шифрування з відкритим ключем. Алгоритм RSA

Мета роботи: Вивчити принцип роботи асиметричних алгоритмів шифрування на прикладі алгоритму RSA

Короткі теоретичні відомості

Головна проблема використання одноключових (симетричних) криптосистем полягає в розподілі ключів. Для того, щоб був можливий обмін інформацією між двома сторонами, ключ повинен бути згенерований однією з них, а потім в конфіденційному порядку переданий іншій. Особливої гостроти ця проблема набула в наші дні, коли криптографія стала загальнодоступною, внаслідок чого кількість користувачів великих криптосистем може обчислюватися сотнями і тисячами.

Початок асиметричним шифрів було покладено в роботі «Нові напрямки в сучасній криптографії» Уитфілда Діффі і Мартіна Хеллмана, опублікованій в 1976 році. Перебуваючи під впливом роботи Ральфа Меркле (Ralph Merkle) про поширення відкритого ключа, вони запропонували метод отримання секретних ключів для симетричного шифрування, використовуючи відкритий канал. У 2002 році Хеллман запропонував називати даний алгоритм «Діффі - Хеллмана - Меркле», Визнаючи внесок Меркле в винахід

криптографії з відкритим ключем. Проте робота Діффі-Хеллмана створила великий теоретичний доробок для відкритої криптографії, першою реальною криптосистемою з відкритим ключем вважають алгоритм RSA (названий по імені авторів - Рон Ривест (Ronald Linn Rivest), Аді Шамір (Adi Shamir) і Леонард Адлеман (Leonard Adleman) з Массачусетського Технологічного Інституту (MIT)).
Суть шифрування з відкритим ключем полягає в тому, що для шифрування даних використовується один ключ, а для розшифрування інший (тому такі системи часто називають асиметричними).
Основна передумова, яка привела до появи шифрування з відкритим ключем, полягало в тому, що відправник повідомлення (той, хто зашифровує повідомлення), не обов'язково повинен бути здатний його розшифровувати. Тобто навіть маючи вихідне повідомлення, ключ, за допомогою якого воно шифрувати, і знаючи алгоритм шифрування, він не може розшифрувати закриті повідомлення без знання ключа розшифрування.
Перший ключ, яким шифрується вихідне повідомлення, називається відкритим і може бути опублікований для використання всіма користувачами системи. Розшифрування за допомогою цього ключа неможливо. Другий ключ, за допомогою якого дешифрується повідомлення, називається секретним (закритим) і повинен бути відомий тільки законному одержувачу закритого повідомлення.

Завдання на лабораторну роботу

Реалізувати шифрування та дешифрування тексту довільної довжини за допомогою асиметричного алгоритму RSA без використання зовнішніх бібліотек. Програма має включати:
Генератор простих чисел (p і q);
Функцію знаходження оберненого елементу по модулю за допомогою розширеного алгоритму Евкліда.

Опис способу обробки результатів

Алгоритми шифрування з відкритим ключем використовують так звані незворотні або односторонні функції. Ці функції мають наступну властивість: при заданому значенні аргументу x відносно просто обчислити значення функції (x), однак, якщо відомо значення функції y = f (x), то немає простого шляху для обчислення значення аргументу x. Наприклад, функція SIN. Знаючи x, легко знайти значення SIN (x) (наприклад, x =π, Тоді SIN (π) = 0). Однак, якщо SIN (x) = 0, однозначно визначити x не можна, тому що в цьому випадку x може бути будь-яким числом, що визначається за формулою і *π, Де і - ціле число.
Однак не всяка необоротна функція годиться для використання в реальних криптосистемах. У їх числі і функція SIN. Слід також зазначити, що в самому визначенні необоротності функції присутня 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 криптосистем з відкритим ключем пред'являються два важливих і очевидних вимоги.
1. Перетворення вихідного тексту повинно бути умовно необоротним і виключати її відновлення на основі відкритого ключа.
2. Визначення закритого ключа на основі відкритого також повинно бути неможливим на сучасному технологічному рівні.
Усі пропоновані сьогодні криптосистеми з відкритим ключем спираються на один з наступних типів односторонніх перетворень.
Розкладання великих чисел на прості множники (алгоритм RSA).
Обчислення дискретного логарифма або дискретне зведення в ступінь (алгоритм Діффі-Хелмана-Меркле, схема Ель-Гамаля).
Питання про укладанні рюкзака (ранця) (автори Хелман і Меркле).
Обчислення коренів алгебраїчних рівнянь.

Використання кінцевих автоматів (автор Тао Ренжи).
Використання кодових конструкцій.
Використання властивостей еліптичних кривих.

Опис ходу експерименту

Алгоритм RSA. Стійкість RSA ґрунтується на великій обчислювальної складності відомих алгоритмів розкладання добутку простих чисел на множники. Наприклад, легко знайти добуток двох простих чисел 7 і 13 навіть подумки - 91. Спробуйте в розумі знайти два простих числа, добуток яких дорівнює 323 (числа 17 і 19). Звичайно, для сучасної обчислювальної техніки знайти два простих числа, добуток яких одно 323, не проблема. Тому для надійного шифрування алгоритмом RSA, як правило, вибираються прості числа, кількість двійкових розрядів яких дорівнює кільком сотням.
Алгоритм RSA складається з 4 етапів: генерації ключів, шифрування, розшифрування та розповсюдження ключів.
Безпека алгоритму RSA побудована на принципі складності факторизації цілих чисел. Алгоритм використовує два ключі — відкритий (public) і секретний (private), разом відкритий і відповідний йому секретний ключі утворюють пари ключів (keypair). Відкритий ключ не потрібно зберігати в таємниці, він використовується для шифрування даних. Якщо повідомлення було зашифровано відкритим ключем, то розшифрувати його можна тільки відповідним секретним ключем.
Генерація ключів
Для того, щоб згенерувати пари ключів виконуються такі дії:

1. Вибираються два великі прості числа p і q
2. Обчислюється їх добуток $n = pq$
3. Обчислюється функція Ейлера $\varphi(n) = (p-1)(q-1)$
4. Вибирається ціле число e таке, що $1 < e < \varphi(n)$ та e взаємно просте з $\varphi(n)$
5. За допомогою розширеного алгоритму Евкліда знаходиться число d таке, що $ed \equiv 1 \pmod{\varphi(n)}$

Число n називається модулем, а числа e і d — відкритою й секретною експонентами (англ. encryption and decryption exponents), відповідно. Пари чисел (n, e) є відкритою частиною ключа, а (n, d) — секретною. Числа p і q після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті.

Шифрування
Припустимо, що Боб хотів би відправити повідомлення M Алісі. Спочатку він перетворює M в ціле число m так, щоб $0 \le m < n$ за допомогою узгодженого оборотного протоколу, відомого як схеми доповнення. Потім він обчислює зашифрований текст c, використовуючи відкритий ключ Аліси e, за допомогою рівняння:

$$c = m^e \bmod n$$

Це може бути зроблено досить швидко, навіть для 500-бітних чисел, з використанням модульного зведення в ступінь. Потім Боб передає c Алісі.
Розшифрування
Для розшифрування повідомлення Боба m Алісі потрібно обчислити таку рівність:

$$m = c^d \bmod n$$

№	Опис операції	Приклад
1	Вибирається два простих числа **p** і **q**	p=7, q=13
2	Розраховується добуток **n = p * q**	n=91
3	Розраховується функція Ейлера **ϕ(n)=(p-1)(q-1)** Результат дорівнює кількості позитивних чисел, які не перевищують **n** і взаємно прості з **n**	ϕ(n) = (7-1)(13-1) = 91-7-13+1
4	Вибирається довільне число **e** (0<e<n), взаємно просте з результатом функції Ейлера. (Відкритий ключ)	e=5
5	Розраховується секретний ключ **d** із відношення **(d*e) mod ϕ(n) = 1**	(d*5) mod 72 = 1, d = 29
6	Публікуються відкриті ключі **e** та **n** в спеціальному сховищі, де виключається можливість підміни	

Примітки. Просте число - натуральне число, більше одиниці і не має інших дільників, крім самого себе і одиниці. Взаємно прості числа - числа, які не мають спільних дільників, крім 1 (наприклад, 2 і 3 — взаємно прості, а 2 і 4 — ні, тому що діляться на 2).
Процедури шифрування і дешифрування виконуються за такими формулами
$C = T^e \bmod n$,
$T = C^d \bmod n$.
де T, C - числові еквіваленти символів відкритого і шифрованого повідомлення.
Слід зазначити, що p і q вибираються таким чином, щоб n було більше коду будь-якого символу відкритого повідомлення. В автоматизованих системах вихідне повідомлення переводитися в двійкове подання, після чого шифрування виконується над блоками біт, рівної довжини. При цьому довжина блоку повинна бути менше, ніж довжина двійкового представлення n.
У висновку слід зазначити стійкість даного алгоритму. У 2003 р Аді Шамір і Еран Тромер розробили схему пристрою TWIRL, яке при вартості $ 10 000 може дешифрувати 512-бітний ключ за 10 хвилин, а при вартості $ 10 000 000 - 1024-бітний ключ менше, ніж за рік. В даний час Лабораторія RSA рекомендує використовувати ключі розміром 2048 бітів.

Домашнє завдання

1. Реалізувати генерацію пари ключів RSA з іншими параметрами.
2. Проаналізувати вплив розміру ключа RSA на криптостійкість і швидкодію.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. Що таке шифрування з відкритим ключем, і які основні його переваги в порівнянні з

симетричним шифруванням?
2. Які математичні принципи лежать в основі алгоритму RSA? Як використовується проблема факторизації великих чисел для забезпечення його безпеки?
3. Як у RSA відбувається генерація пари ключів? Які етапи включає цей процес?
4. Як працює алгоритм RSA у процесі шифрування та дешифрування даних? У чому полягає роль відкритого та закритого ключів?
5. Які основні вразливості алгоритму RSA? Як можна забезпечити додаткову стійкість до атак?

Література

1. Rivest R. L., Shamir A., Adleman L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* // Communications of the ACM. – 1978. – Vol. 21, No. 2. – P. 120–126.
2. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
3. Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
4. Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.
5. Boneh D. *Twenty Years of Attacks on the RSA Cryptosystem* // Notices of the AMS. – 1999. – Vol. 46, No. 2. – P. 203–213.

Лабораторна робота №7 Обмін ключами за схемою Діффі-Хеллмана

Мета роботи: Освоїти обмін ключами за схемою Діффі-Хеллмана, вивчаючи проблему первісних коренів. Вивчити методи знаходження простих чисел.

Короткі теоретичні відомості

Протокол Діффі-Хеллмана (Diffie-Hellman Key Exchange) — це криптографічний протокол, який дозволяє двом сторонам спільно генерувати секретний ключ через небезпечний канал зв'язку. Цей секретний ключ використовується для подальшого шифрування повідомлень. Протокол заснований на складності обчислення дискретного логарифму в великих простих полях. У 1976 році була опублікована робота американських математиків У. Діффі і М.Е. Хеллмана «Нові напрямки в криптографії», в ній вони запропонували конкретну конструкцію так званого "відкритого розподілу ключів". Мета алгоритму полягає в тому, щоб два учасники могли безпечно обмінятися ключем, який в подальшому може використовуватися в будь-якому алгоритмі симетричного шифрування. Сам алгоритм Діффі-Хеллмана може застосовуватися тільки для обміну ключами. Алгоритм заснований на труднощі обчислень дискретних логарифмів. Безпека обміну ключа в алгоритмі Діффі-Хеллмана випливає з того факту, що, хоча відносно легко обчислити експоненти по модулю простого числа, дуже важко обчислити дискретні логарифми. Для

великих простих чисел завдання вважається нерозв'язною.

Завдання на лабораторну роботу

Реалізувати програму, яка демонструвала б, як проводиться обмін ключами за схемою Діффі-Хеллмана:

Прості числа X_A, X_B і q отримувати випадковим чином за допомогою генератора простих чисел на основі тесту Рабіна-Міллера;

Реалізувати та використати функцію знаходження числа Ейлера;

Реалізувати та використати функцію знаходження первісного кореня

Опис способу обробки результатів

Прості числа

Простих чисел не так мало, як здається, наприклад, існує приблизно 10^{151} простих чисел довжиною від 1 біта до 512 включно. Для чисел близьких n, ймовірність того, що вибране число виявиться простим, дорівнює 1 / ln n. Тому повне число простих чисел, менших n одно n / ln n. Вважається, що ймовірність вибору двома людьми одного і того ж великого простого числа нехтує мала.

Існують різні імовірнісні перевірки на простоту чисел, що визначають чи є число простим з заданим ступенем достовірності. За умови, що ця ступінь достовірності достатньо велика, такі способи досить гарні. Такі прості числа часто називають «промисловими простими», тобто вони прості з контрольованою можливістю помилки.

Повсюдно використовуються є алгоритм, розроблений Майклом Рабіном по ідеям Гарі Міллера.

Тест Rabin-Miller

Виберіть для перевірки випадкове число p. Обчисліть b як найбільше число поділок p-1 на 2 тобто 2^b - найбільша ступінь числа 2, на яке ділиться p-1. Потім обчисліть m, таке, що $p = 1 + 2^b m$

1. Виберіть випадкове число a, менше p.
2. Встановіть j = 0 і $z = a^m \bmod p$
3. Якщо z = 1 або якщо z = p-1, то p проходить перевірку і може бути простим числом.
4. Якщо j> 0 і z = 1, то p не є простим числом.
5. Встановіть j = j + 1.

Якщо j <b і z <p-1, встановіть z = z2 mod p і поверніться на ④.

Якщо z = p-1, то p проходить перевірку і може бути простим числом.

6. Якщо j = b і z≠*p*-1, то p не є простим числом.

Повторити цю перевірку потрібно t раз.

Доведено, що в цьому тесті ймовірність проходження перевірки складовим числом убуває швидше, ніж в інших. Гарантується, що 75% можливих значень, а виявляться показниками того, що вибране число p складене. Це означає, що ймовірність прийняти складене число p за просте не перевищує величини (¼) ^{t}.

Псевдокод:

МІЛЛЕР-РАБІН(n, k)

1. **якщо** n парне **тоді**
2. **повернути** ХИБА
3. $m \leftarrow (n-1)$ div $2; t \leftarrow 1$
4. **поки** m парне
5. $m \leftarrow m$ div $2; t \leftarrow t+1$
6. **для** $i \leftarrow 1$ **до** k
7. $a \leftarrow Random()$ mod n
8. $u \leftarrow a^m$ mod n
9. **якщо** $u \neq 1$ **тоді**
10. $j \leftarrow 1$
11. **поки** $u \neq n-1$ і $j < t$
12. $u \leftarrow u^2$ mod $n; j \leftarrow j+1$
13. **якщо** $u \neq n-1$ **тоді**
14. **повернути** ХИБА
15. **повернути** ІСТИНА

Генерація простого числа

1. Згенеруйте випадкове n-бітове число p.
2. Встановіть його старший і молодший біти рівними 1. Старший біт гарантуватиме необхідну довжину шуканого числа, а молодший біт забезпечує його непарність.
3. Переконайтеся, що p не ділиться на невеликі прості числа: 3, 5, 7, 11 і т.д. Найбільш ефективною є перевірка на подільність для всіх простих чисел, менших 2000.
4. Виконайте тест Rabin-Miller мінімум 5 разів.

Якщо p не пройшло хоча б одну перевірку з 3 або 4, Воно не є простим.

Перевірка, що випадкове непарне p не ділиться на 3, 5 і 7 відсікає 54% непарних чисел. Перевірка подільності на всі прості числа, менші 256 відсікає 80% складових непарних чисел.

Навіть, якщо складене число «просочилося» через цей алгоритм, це буде відразу ж помічено, тому що шифрування і дешифрування не працюватимуть.

Первісна (примітивний) корінь по модулю n

В силу теореми Ейлера, для будь-яких взаємно простих, а й n виконується співвідношення

$a^{\varphi(N)} \equiv 1 \bmod n,$ (9.1)

де $\varphi(n)$позначає функцію Ейлера, значення якої дорівнює числу позитивних цілих значень, менших n і взаємно простих з n. Для простого числа p

$\varphi(p) = p-1,$

Якщо припустити, що два числа p і q прості, тоді для $n = p^{\alpha} * q^{\beta}$ функція Ейлера матиме вигляд

$\varphi(n) = \varphi(p^{\alpha} * q^{\beta}) = \varphi(p^{\alpha}) * \varphi(q^{\beta}) = [(p-1)\, p^{\alpha-1}] * [(q-1)\, q^{\beta-1}].$ (9.2)

Розглянемо більш загальне співвідношення, ніж (9.1). Кажуть, що число a, взаємно просте з модулем n, належить показнику m, якщо m - таке найменше натуральне число, що виконується порівняння

$a^m \equiv 1 \bmod n.$ (9.3)

Якщо a і n є взаємно простими, то існує, принаймні, одне число $m = \varphi(n)$, Що задовольняє (9.3). Найменше з позитивних чисел m, для яких виконується (9.3), є довжиною періоду послідовності, що генерується ступенями a.

Справедливі наступні властивості.

Властивість 1. Числа $a^0, a^1, ..., a^{m-1}$ попарно непорівнянні по модулю n.

Дійсно, $a^l \equiv a^k \bmod n$, L> k $\Rightarrow a^{l-k} \equiv 1 \bmod n$, Де $l - k \in$N, $l - k < m$.

Властивість 2. $a^{\gamma} \equiv a^{\gamma'} \bmod n \Leftrightarrow \gamma \equiv \gamma' \bmod m$. розділимо$\gamma$, γ' на m із залишками

$\gamma = Mq + r$, $\gamma' = Mq' + r'$. тоді $a^{\gamma} \equiv a^{\gamma'} \Leftrightarrow$ amq + r $\equiv amq' + r' \Leftrightarrow$ ar $\equiv ar' \Leftrightarrow$r '= r. Звідси випливає наступна властивість.

Властивість 3. m | *φ(n)*. Число, що належить показнику*φ(n)*, Називається первісним коренем за модулем *n*.
Властивість 4. За будь-якій простій модулю p існує первісний корінь. Первісні корені існують по модулях 2, 4, p^{α}, $2p^{\alpha}$, Де p - непарне просте, $\alpha \in N$.
Властивість 5. Нехай c = *φ(n)*і q1, q2, ..., q_k - різні прості дільники числа c. Число a, взаємно просте з модулем n, буде первісним коренем тоді і тільки тоді, коли не виконана жодна з наступних порівнянь:

$$a^{c/q_1} \equiv 1 \bmod n, a^{c/q_2} \equiv 1 \bmod n, \ldots, a^{c/q_k} \equiv 1 \bmod n$$

Необхідність випливає з того, що $a^{\varphi(n)} \equiv 1 \bmod n$ і порівняння не має місця при менших показниках ступеня. Назад, припустимо, що a не задовольняє жодному з порівнянь, і нехай a належить показнику m <c. Тоді m | c⇒*c = mn*. Позначимо через q простий дільник u. Тоді легко отримати протиріччя:
$a^{c/q} = a^{mu/q} = (am)^{u/q} \equiv 1 \bmod n$.
Якщо деяка послідовність має довжину *φ(n),*тоді ціле число a генерує своїми ступенями безліч всіх ненульових лишків за модулем *n*. Таке ціле число називають первісним коренем числа *a* по модулю *n*. Кількість їх одно для числа *n*
φ(N-1), (9.4)
де *φ* - функція Ейлера.

Приклад (Перевірка Властивості 5.)Нехай n = 41. Маємо c =φ(41) = 40 = $23 \cdot 5$. Отже, первісний корінь не повинен задовольняти двом порівнянь
$a^{8} \equiv 1 \pmod{41}$, $a^{20} \equiv 1 \pmod{41}$.
Спробуємо числа 2, 3, 4, …: $2^8 \equiv 10$, $2^{20} \equiv 1$, $3^8 \equiv 1$, $4^8 \equiv 18$, $4^{20} \equiv 1$, $5^8 \equiv 18$, $5^{20} \equiv 1$, $6^8 \equiv 10$, $6^{20} \equiv 40$. Звідси бачимо, що 6 є найменшим первісним коренем за модулем 41.

Опис ходу експерименту
Припустимо, що двом абонентам необхідно провести конфіденційне листування, а в їхньому розпорядженні немає спочатку обумовленого секретного ключа. Однак між ними існує канал, захищений від модифікації, тобто дані, що передаються по ньому, можуть бути прослухані, але не змінені (такі умови мають місце досить часто). У цьому випадку дві сторони можуть створити однаковий секретний ключ, ні разу не передавши його по мережі, за наступним алгоритмом.

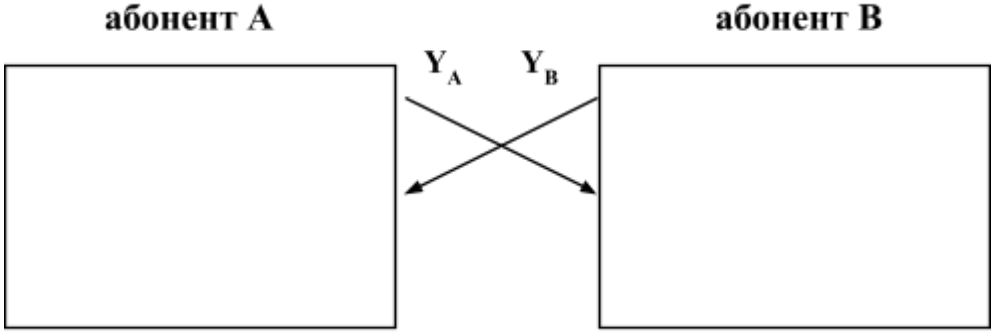


Рис. 1.Обмін ключами за схемою Діффі - Хеллмана.

Алгоритм полягає в наступному:
Глобальні відкриті елементи:
q - просте число
a - первісний корінь q
Обчислення ключа абонентом A:
вибирається секретне число X_A (X_A <q)
обчислення відкритого значення Y_A: $Y_A = a^{X_A} \pmod{q}$
Обчислення ключа абонентом B:

вибирається секретне число X_B (X_B <q)

обчислення відкритого значення Y_B: $Y_B = a^{X_B} (\mathrm{mod}\, q)$

Обчислення секретного ключа абонентом А: $K = (Y_B)^{X_A} (\mathrm{mod}\, q)$

Обчислення секретного ключа абонентом В: $K = (Y_A)^{X_B} (\mathrm{mod}\, q)$

Необхідно ще раз відзначити, що алгоритм Діффі-Хеллмана працює тільки на лініях зв'язку, надійно захищених від модифікації. Якби він був застосуємо на будь-яких відкритих каналах, то давно зняв би проблему поширення ключів і, можливо, замінив собою всю асиметричну криптографію.

Приклад:

Нехай $q = 97$ і $a = 5$

Абонент А: згенерувати випадкове число $X_A = 36$

Абонент В: згенерувати випадкове число $X_B = 58$

Ці елементи вони тримають в секреті. Далі кожен з них обчислює новий елемент: $Y_A = 5^{36} (\mathrm{mod}\, 97) = 50$, $Y_B = 5^{58} (\mathrm{mod}\, 97) = 44$

Потім вони обмінюються цими елементами по каналу зв'язку. Тепер абонент А, отримавши Y_B і знаючи свій секретний елемент X_A, обчислює загальний ключ: $K_A = 44^{36} (\mathrm{mod}\, 97) = 75$.

Аналогічно надходить абонент В: $K_B = 50^{58} (\mathrm{mod}\, 97) = 75$

Домашнє завдання

1. Реалізувати алгоритм обміну ключами Діффі–Хеллмана з іншими параметрами.
2. Описати можливі атаки на протокол Діффі–Хеллмана та способи їх запобігання.

Рекомендації щодо оформлення звіту з роботи

1. опис використовуваного методу,
2. опис вихідних даних,
3. алгоритм роботи програми,
4. текст програми,
5. результати роботи програми,
6. аналіз результатів
7. висновки.

Контрольні запитання

1. У чому полягає основна ідея обміну ключами за схемою Діффі-Хеллмана?
2. Які загальні параметри використовуються в схемі Діффі-Хеллмана, і як вони обираються?
3. Як обчислюється спільний секретний ключ в учасників протоколу?
4. Які основні вразливості має схема Діффі-Хеллмана? Як можна захиститися від атак "людина посередині"?
5. У яких сучасних протоколах застосовується обмін ключами за схемою Діффі-Хеллмана?

Література

1. Diffie W., Hellman M. *New Directions in Cryptography* // IEEE Transactions on Information Theory. – 1976. – Vol. 22, No. 6. – P. 644–654.
2. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 3rd ed. – New York: Wiley, 2020. – 784 p.
3. Menezes A., van Oorschot P., Vanstone S. *Handbook of Applied Cryptography*. – Boca Raton: CRC Press, 1996. – 816 p.

4. Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. – Boston: Pearson, 2020. – 800 p.
5. Koblitz N. *A Course in Number Theory and Cryptography*. – Springer, 1994. – 250 p.