Міністерство освіти і науки України
Державний вищій навчальний заклад
“Криворізький національний університет”

МЕТОДИЧНІ ВКАЗІВКИ

для самостійної роботи
з дисципліни
„ Захист інформації у комп’ютерних системах ”

Для студентів денної та заочної форм навчання

Спеціальність 123 «Комп’ютерна інженерія»
Факультет інформаційних технологій
Кафедра комп’ютерні системи і мережі

Кривий Ріг
2018

Методичні вказівки до самостійної роботи для студентів денної та заочної форм навчання з дисципліни „ Захист інформації у комп'ютерних системах ".

Укладач: доц.,к.т.н. Вдовиченко І.Н.
Комп'ютерний набір : доц., к.т.н. Вдовиченко І.Н.

Схвалено на засіданні кафедри комп'ютерні системи і мережі
(Протокол № 7 від 20.02. 2018 р.)

завкафедрою _____ д.т.н. А.І. Купін
підпис

Методичні вказівки розглянуто та схвалено вченою радою факультету інформаційних технологій
(Протокол № 7 від 21.02. 2018 р.)

Голова вченої ради ФІТ _____ В.А. Чубаров
підпис

Начальник навчально-методичного відділу _____ Г.Х.Отверченко
підпис

Методичні вказівки до самостійної роботи з дисципліни «Захист інформації в комп'ютерних системах» містять короткі теоретичні відомості, завдання для самостійної роботи, докладний опис ходу їх виконання, завдання за варіантами, а також перелік рекомендованої літератури.

Призначені для студентів денної та заочної форм навчання за спеціальністю 123 «Комп'ютерна інженерія»

Наклад примірників: за вимогою

ЗМІСТ

ВСТУП

Швидкий розвиток комп’ютерних інформаційних технологій вносить помітні зміни в життя суспільства. Все частіше поняття «інформація» використовується як позначення спеціального товару, що можна купити, продати чи обміняти. При цьому вартість інформації часто в сотні і тисячі разів перевищує вартість комп’ютерної системи, в якій вона знаходиться. Тому цілком природно виникає необхідність в захисті інформації від несанкціонованого доступу, зміни, викрадення, знищення і інших злочинних дій.

Проблеми захисту інформації викликають все більшу увагу як спеціалістів, так і численних користувачів сучасних комп’ютерних засобів. Для розв’язання цих проблем не існує якогось одного технічного засобу чи прийому. Однак спільним в вирішенні багатьох проблем є використання криптографії і криптоподібних перетворень інформації.

Лабораторна робота №1.

Тема: Шифрування методами перестановки.

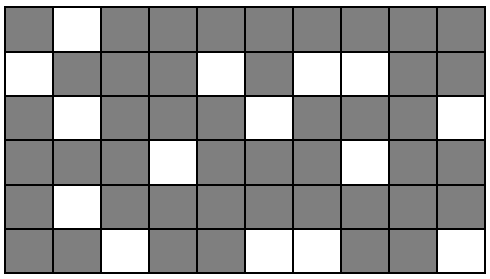
Мета: навчитись розроблювати програми для шифрування методами перестановок.

Теоретичний матеріал

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

Шифр «Сцитала»

Шифр «Сцитала» реалізує не більш n перестановок (n, як і раніше, - довжина повідомлення). Дійсно, цей шифр, еквівалентний наступному шифру маршрутної перестановки: у таблицю, що складається з n стовпців, упорядковано записують повідомлення, після чого виписують букви по стовпцях. Число задіяних стовпців у таблиці не може перевершувати довжини повідомлення.



Є ще чисто фізичні обмеження, що накладаються реалізацією шифру «Сцитала». Природно припустити, що діаметр жезла не повинний перевершувати 10см. При висоті рядка в 1см на одному витку такого жезла уміститься не більш 32 букв ($10\pi<32$). Таким чином, число перестановок, реалізованих «Сциталой» навряд чи перевершує 32.

Шифр вертикальної перестановки

Широко поширений різновид шифру маршрутної перестановки, називана ***«шифром вертикальної перестановки»*** (ШВП). У ньому знову використовується прямокутник, у який повідомлення вписується звичайним способом (по рядках ліворуч праворуч). Виписуються букви по вертикалі, а стовпці при цьому беруться в порядку, обумовленому ключем.

Число ключів ШВП не більш m!, де m – число стовпців таблиці. Як правило, m набагато менше, ніж довжина тексту n (повідомлення укладається в кілька рядків по m букв), а, виходить, і m! багато менше n!.

У випадку, коли ключ ШВП не рекомендується записувати, його можна витягати з якогось легко запам'ятовується чи слова пропозиції. Для цього існує багато способів. Найбільш розповсюджений полягає в тому, щоб приписувати буквам числа відповідно до звичайного алфавітного порядку букв.

Для забезпечення додаткової скритності можна повторно зашифрувати повідомлення. Такий метод шифрування називається подвійною перестановкою. У випадку подвійної перестановки стовпців і рядків таблиці, перестановки

визначаються окремо для стовпців і окремо для рядків. Спочатку в таблицю записується текст повідомлення, а потом по черзі переставляються стовпці, а потім рядки. При розшифруванні порядок перестановок повинний бути зворотним.

Число варіантів подвійної перестановки швидко зростаємо при збільшенні розміру таблиці:

для таблиці 3 х 3 36 варіантів;

для таблиці 4 х 4 576 варіантів;

для таблиці 5 х 5 14400 варіантів.

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

Застосування магічних квадратів

У середні століття для шифрування перестановкою застосовувалися і магічні квадрати.

Магічними квадратами називають квадратні таблиці з уписаними в їхні клітки послідовними натуральними числами, починаючи від 1, що дають у сумі по кожнім стовпці, кожному рядку, і кожної діагоналі те саме число.

Текст, що шифрувався, вписували в магічні квадрати відповідно до нумерації їх кліток. Якщо потім виписати вміст такої таблиці по рядках, то вийде шифртекст, сформований завдяки перестановці букв вихідного повідомлення.

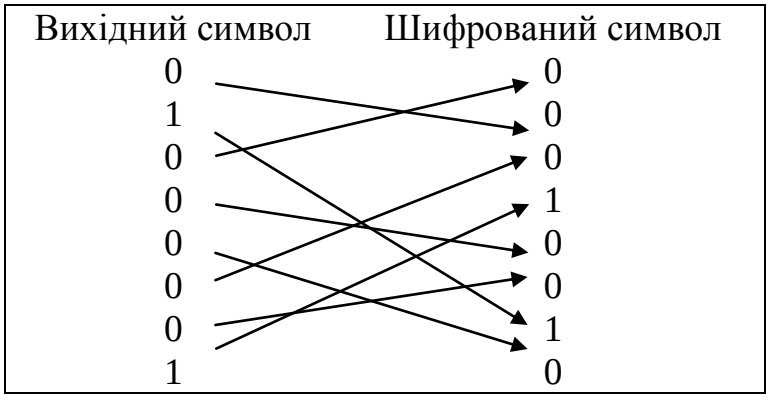
Число магічних квадратів швидко зростає зі збільшенням розміру квадрата. Існує тільки один магічний квадрат розміром 3х3 (якщо не враховувати його повороти). Кількість магічних квадратів 4х4 складає вже 880, а кількість магічних квадратів 5х5 – близько 250000.

Перестановка біт

Використанням комп'ютерів у процесі шифрування продиктований наступний метод – ***перестановки біт*** у символі.

Нехай повідомлення складається із символів, кожної з який у машинному представленні являє собою послідовність з восьми біт. Переставивши деяким чином біти в кожнім символі, ми одержимо зашифровану послідовність.

Щоб розшифрувати її, нам буде необхідно лише зробити зворотну перестановку.



Розкриття шифру перестановки

Перейдемо до питання про методи розкриття шифру перестановки. Проблема, що виникає при відновленні повідомлення, зашифрованого ШП, складається не тільки в тім, що кількість можливих ключів велика навіть при невеликих довжинах тексту. Якщо й удається перебрати всі припустимі варіанти перестановок, не завжди ясно, який з цих варіантів вірний. Наприклад, нехай потрібно відновити вихідний текст по криптограмі АОГР, і нам нічого не відомо, крім того, що застосовувався шифр перестановки. Який варіант «осмисленого» вихідного тексту визнати щирим: чи ГОРА РОГУ? А може бути АРГО?

Іноді, за рахунок особливостей реалізації шифру, вдається одержати інформацію про використовуване перетворення перестановки.

Розглянемо шифр «Сцитала». Вище вже розглядалося питання про кількість перестановок, реалізованих «Сциталой». Їх виявилося не більш 32. Це число невелике, тому можна здійснити перебір усіх варіантів. При достатній довжині повідомлення ми, швидше за все, одержимо єдиний варіант тексту, що шифрується.

Розглянемо приклад зі ШВП. За умовою пробіли між словами при записі тексту в таблицю опускалися, тому укладаємося, що всі стовпці, що містять пробіл в останньому рядку, повинні стояти наприкінці тексту. Таким чином, виникає розбивка стовпців на двох груп (що містять 6 букв, і містять 5 букв). Для завершення відновлення вихідного тексту досить знайти порядок проходження стовпців у кожній із груп окремо, що набагато простіше.

Приклад виконання завдання

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

```
program FileCoding;
const
 Brk=4;                        { Довжина підстановки }
type
 Range=1..Brk;
const
 stend:array[Range] of Range=(3,1,4,2); { Підстановка  }
 { При декодуванні можна використовувати едентичну    }
 { процедуру декодування, але з підстановкою (2,4,1,3)   }
var
 FIn:file;                     { Вхідний файл }
 FOut:file of byte;            { Вихідний файл }
 Tmp:array[Range] of byte;   { Перемінна для збереження прочитаних
байт }
 RealCnt:integer;              { Кількість реальна прочитаних байт }
begin { program }
 assign(FIn,'input.txt');
 assign(FOut,'output.txt');
 reset(FIn,1);
 rewrite(FOut);
 while not(eof(FIn)) do
```

```
  begin
   BlockRead(FIn,tmp,Brk,RealCnt);
   for i:=RealCnt+1 to Brk do  { Якщо прочитано менше Brk байт, }
    Tmp[i]:=0;                 { те доповнити нулями }
   for i:=1 to Brk do
    write(FOut,Tmp[stend[i]]);
  end;
 close(FIn);
 close(FOut);
end.  { program }
```

Контрольні питання

1. У чому полягає метод шифрування перестановкою?
2. Що таке маршрутна перестановка?
3. Який «маршрут» можна використовувати для реалізації шифру «Сцитала»?
4. Оцініте кількість ключів шифру вертикальної перестановки. В скількох разів ця кількість ключів зростає при використанні подвійної перестановки?
5. Приведіть приклад використання магічного квадрата для шифрування повідомлення **‘ЯУЕЗЖАЮВКИЕВ’**.
6. Що таке шифрування перестановкою біт?
7. Запропонуєте шлях розкриття шифру перестановки. Які складності виникають при цьому і які «помилки» шифрувальників можна використовувати?

Варіанти завдань

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

Написати програму шифрування (дешифрування), в якості тестового прикладу використати варіант із практичного завдання №1.

1,2. Вхідну послідовність розбийте на групи по чотирьох символу, далі в кожній групі символи переставте з використанням підстановки, що виберіть самостійно.

3,4. Реалізуйте маршрутну перестановку з використанням таблиці, що шифрує, 5x8. Маршрут виберіть самостійно.

5,6. Реалізуйте процедуру, що моделює використання «Сцитала». Число стовпців таблиці, що шифрує, виберіть самостійно.

7,8. Реалізуйте шифрування подвійною перестановкою. Розмірність таблиці, що шифрує, вибрати самостійно.

11,12. Змоделюйте використання магічних квадратів. Передбачите їхню генерацію.

13,14. Реалізуйте метод перестановки біт.

Лабораторна робота №2.

Тема: Шифрування методами заміни

Мета: навчитись писати програми для шифрування (дешифрування) методи заміни

Теоретичний матеріал

Шифрами заміни називають такі шифри, що здійснюються шляхом заміни кожного символу відкритого повідомлення на інші символи – шифропозначення, причому порядок проходження шифропозначений збігається з порядком проходження відповідних їм символів відкритого повідомлення.

Нехай, наприклад, зашифровується повідомлення російською мовою і при цьому заміні підлягає кожна буква повідомлення. Формально в цьому випадку шифр заміни можна описати в такий спосіб. Для кожної букви α вихідного алфавіту будується деяка множина символів $M\alpha$, що називається безліччю шифро-зображень для букви α.

Таблиця є ключем шифру заміни. Знаючи її можна здійснити як зашифрування, так і розшифрування.

При зашифруванні кожна буква α відкритого повідомлення, починаючи з першої, заміняється будь-яким символом з множини M_α. За рахунок цього можна одержати різні варіанти зашифрованого повідомлення для того самого відкритого повідомлення.

У найпростішому випадку безліч шифро-зображень M_α складається з одного елемента. Такий шифр називається шифром простої заміни.

А	Б	…	Я
$M_А$	$M_Б$	…	$M_Я$

Шифри простої заміни

Система шифрування Цезаря

Як ключ шифру Цезаря використовують таблицю, перший рядок якої містить букви вихідного алфавіту, а другий рядок – послідовність букв, записаних за абеткою, але починається не з букви А, а з який-небудь іншої:

А	Б	…	Э	Ю	Я
Д	Е	…	Б	В	Г

При шифруванні букву вихідного повідомлення знаходять у першому рядку і заміняють її на відповідну букву другого рядка. Запам'ятати ключ досить просто – треба лише запам'ятати першу букву другого рядка.

Серйозний недолік даного шифру – обмежене число ключів, що відповідає числу букв в алфавіті.

Афінна система підстановок Цезаря

Тут букви вихідного повідомлення перетворяться в такий спосіб:

$T_1=AT+B \pmod m$,

де T – порядковий номер букви вихідної послідовності,

T_1 - порядковий номер відповідної букви зашифрованої послідовності,
m – розмір алфавіту,
A,У – цілі числа, причому A и m взаємно прості.

Лозунговий шифр

У даному шифрі запам'ятовування ключа засноване на гаслі – легко запам'ятовується чи слові фразі. Наприклад, виберемо слово – гасло “заява” і заповнимо другий рядок таблиці за наступним правилом: *спочатку вписується слово - гасло, причому повторювані букви відкидаються, потім ця таблиця доповнюється не ввійшли в неї буквами алфавіту*. Ключ буде мати вид:

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

Полібіанський квадрат

Шифр винайшов грецький письменник і історик Полібій. Прямокутна таблиця заповнюється буквами алфавіту у випадковому порядку. Кожна буква відкритого повідомлення заміняється буквою, розташованої нижче в тім же стовпці. Якщо буква знаходиться на останньому рядку таблиці, то для її шифрування беруть саму верхню букву стовпця.

Таблиця шифрування Трисемуса

Для одержання такого шифру використовується таблиця для запису букв і ключове слово (чи фраза). У таблицю спочатку вписується ключове слово, причому повторювані букви відкидаються. Потім ця таблиця доповнюється буквами алфавіту що не ввійшли в неї. На малюнку зображена таблиця Трисемуса з ключовим словом “БАНДЕРОЛЬ”. Застосування таблиці аналогічно застосуванню полібіанського квадрата.

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

Достоїнством цього й інших розглянутих шифрів простої заміни є простота реалізації, недоліком – легке розкривання, зважаючи на те, що дані шифри зберігають інформацію про частоту зустрічальності букв вихідного тексту. Це дозволяє застосовувати методи підрахунку частот для розшифровки повідомлень. З метою усунення даного недоліку застосовують наступні методи:

Біграмний шифр Плейфейра.

Цей шифр заснований на таблиці, аналогічній таблиці Трисемуса.

Процедура шифрування включає наступні кроки:

1) Відкритий текст розбивається на пари букв. Текст повинний мати парне число букв, і в ньому не повинно бути пар букв (біграми), що містять дві однакові букви.

2) Послідовність біграм відкритого тексту перетвориться в послідовність біграм

за допомогою таблиці, що шифрує, за наступними правилами:

Якщо обидві букви біграми відкритого повідомлення не попадають в один чи рядок стовпець, тоді для заміни знаходять букви в кутах прямокутника, обумовленого даною парою букв.

Якщо обидві букви біграми відкритого повідомлення належать одному стовпцю таблиці, то їх заміняють на букви, що лежать під ними. Якщо при цьому буква відкритого тексту знаходиться в нижньому рядку, то для шифрування береться буква з верхнього рядка того ж стовпця.

Якщо обидві букви біграми відкритого повідомлення належать одному рядку таблиці, то вони заміняються на букви, що лежать праворуч від них. Якщо при цьому буква відкритого тексту знаходиться в крайньому правому стовпці, то для шифрування береться буква з крайнього лівого стовпця того ж рядка.

Система омофонів

Даний шифр характеризується тим, що букви вихідного повідомлення мають кілька замін. Число замін береться пропорційним імовірності появи букви у відкритому тексті. Дані про розподіли імовірностей букв у російському тексті приведені в таблиці.

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

Шифруючи букву вихідного повідомлення, вибирають випадковим образом одну з її замін. Заміни (часто називані омофонами) можуть бути представлені трьохрозрядними числами від 000 до 999. Наприклад, букві ПРО привласнюються 90 випадкових номерів, буквам Б и Ъ – по 14 номерів. Якщо омофони привласнюються випадковим образом різним появам однієї і тієї ж букви, тоді кожен омофон з'являється в шифртексті рівно значно.

Система омофонів забезпечує найпростіший захист від криптоаналітичних атак, заснованих на підрахунку частот появи букв у шифртексті.

Шифри складної заміни

Шифри складної заміни називають багато алфавітними, тому що для шифрування кожного символу вихідного повідомлення застосовують свій шифр простої заміни. Багато алфавітна підстановка послідовно циклічно змінює використовувані алфавіти.

При r-алфавітній підстановці символ x_0 вихідного повідомлення заміняється символом y_0 з алфавіту B_0, символ x_1 - символом y_1 з алфавіту B_1, і так далі,

символ x_{r-1} заміняється символом y_{r-1} з алфавіту B_{r-1}, символ x_r заміняється символом y_r знову з алфавіту B_0, і т.д.

Загальна схема багато алфавітної підстановки для випадку r = 4:

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

Ефект використання багато алфавітної підстановки полягає в тім, що забезпечується маскування природної статистики вихідної мови, тому що конкретний символ з вихідного алфавіту може бути перетворений у декілька різних символів шифрувальних алфавітів B_i.

Шифр Гронсфельда

Шифрування здійснюється в такий спосіб. Під буквами вихідного повідомлення записують цифри числового ключа. Якщо ключ коротше повідомлення, то його запис циклічно повторюють. Для заміни вибирають ту букву, що зміщена за алфавітом на відповідну цифру ключа. Наприклад, застосовуючи як ключ групу з чотирьох початкових цифр числа *e* (підстави натуральних логарифмів), а саме 2718, одержуємо для вихідного повідомлення СХІДНИЙ ЕКСПРЕС наступний шифртекст:

Шифр Гронсфельда являє собою окремий випадок системи шифрування Вижинера.

Система шифрування Вижинера

Цей шифр складної заміни можна описати таблицею шифрування, називаною таблицею (квадратом) Вижинера. Таблиця Вижинера використовується для зашифрування і розшифрування.

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

Таблиця має два входи: верхній рядок підкреслених символів, використовуваний для зчитування чергової букви вихідного відкритого тексту; крайній лівий стовпець ключа.

Послідовність ключів звичайно одержують з числових значень букв ключового слова.

При шифруванні вихідного повідомлення його виписують у рядок, а під ним записують ключове чи слово фразу. Якщо ключ виявився коротше повідомлення, то його циклічно повторюють. У процесі шифрування знаходять у верхньому

рядку таблиці чергову букву вихідного тексту й у лівому стовпці чергове значення ключа. Буква шифртекста знаходиться на перетинанні стовпця, обумовленого буквою, що шифрується, і рядка, обумовленої числовим значенням ключа.

Шифр “подвійний квадрат” Уітстона

На відміну від полібіанського шифр “подвійний квадрат” використовує відразу двох таблиць з випадково розташованими буквами, розміщені по одній горизонталі, а шифрування йде біграмами як і в шифрі Плейфейра.

Процедура шифрування виконується в такий спосіб. Перед шифруванням вихідне повідомлення розбивається на біграми. Кожна біграма шифрується окремо. Першу букву біграми знаходять у лівій таблиці, а другу букву – у правій таблиці. Потім думкою будують прямокутник так, щоб букви біграми лежали в його протилежних вершинах. Інші дві вершини цього прямокутника дають букви біграми шифртекста.

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

И	Ч	Г	Я	Т
,	Ж	Ь	М	О
З	Ю	Р	В	Щ
Ц	:	П	Е	Л
Ъ	А	Н	.	Х
Э	К	С	Ш	Д

Якщо обидві букви біграми лежать в одному рядку, то і букви шифртекста беруть з того ж рядка. Першу букву біграми шифртекста беруть з лівої таблиці в стовпці, що відповідає другій букві біграми повідомлення. Друга ж буква біграми шифртекста береться з правої таблиці в стовпці, що відповідає першій букві біграми повідомлення.

Приклад виконання завдання

Як приклад виконання завдання розглянемо функцію Code, що шифрує вихідний рядок st_in за допомогою системи Вижинера з ключем Key і повертає зашифрований рядок. Вихідне повідомлення може містити прописні букви російського алфавіту і пробіл.

```
{ Функції shift і shift_  перетворять код ASCII символу
   у порядковий номер за алфавітом }
function shift(k:integer):integer;
```

```
  begin   if k>127  then shift:=k-127
             else shift:=0;
  end;

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

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

Контрольні питання

1. Які шифри називають шифрами заміни?
2. Що таке ключ шифру заміни?
3. Що називають множиною шифро-зображень?
4. Приведіть приклади шифрів простої заміни. Опишіть алгоритм одного з них.
5. Які основні недоліки шифрів простої заміни?
6. У чому відмінність шифрів простій і складна заміна?
7. Опишіть алгоритми шифрування, засновані на гаслі.
8. Які шифри складної заміни вам відомі?
9. Яким образом для шифрування використовують “подвійний квадрат” Уітстона?
10. У чому полягає шифрування з використанням системи Вижинера?

Варіанти завдань

Написати програми шифрування (для непарних варіантів) і дешифрування (для парних) файлів за допомогою методу, зазначеного у варіанті.

1-2. Система шифрування Цезаря.
3-4. Афінна система підстановок Цезаря.
5-6. Лозунговий шифр.
7-8. Полібіанський квадрат.
9-10. Таблиця, що шифрує, Трисемуса.
11-12. Біграмний шифр Плейфейра.
13-14. Система омофонів.
15-16. Шифр Гронсфельда.

17-18. Система шифрування Вижинера.
19-20. Шифр “подвійний квадрат” Уитстона.

Лабораторна робота №3.

Тема: Шифрування методом стримування.

Мета: навчитися писати програми для шифрування методом гамування.

Опис методу.

Гамування є широко застосовуваним криптографічним перетворенням.

Під *гамуванням* розуміють процес накладення по визначеному законі гами шифру на відкриті дані. *Гама шифру* - це псевдо випадкова послідовність, вироблена по заданому алгоритмі для шифровки відкритих даних і дешифрування зашифрованих даних.

Процес *шифрування* полягає в генерації гами шифру за допомогою датчика псевдо випадкових чисел і накладенні отриманої гами на вихідний відкритий текст оборотним образом, наприклад з використанням операції додавання по модулі 2.

Слід зазначити, що перед шифровкою відкриті дані розбивають на блоки $T_o^{(i)}$ однакової довжини, звичайно по 64 бита. Гама шифру виробляється у виді послідовності блоків $Г_ш^{(i)}$ аналогічної довжини.

Рівняння шифровки можна записати у вигляді

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

де $T_ш^{(i)}$ *i-й* блок шифртекста;
$Г_ш^{(i)}$ *i-й* блок гами шифру;
$Te^{(i)}$ *i-й* блок відкритого тексту;
M кількість блоків відкритого тексту.

Процес *дешифрування* зводиться до повторної генерації гами шифру і накладенню цієї гами на зашифровані дані. Рівняння дешифрування має вигляд

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

Переваги і недоліки методу.

Одержуваний цим методом шифртекст досить важкий для розкриття, оскільки ключ тут є перемінним. По суті справи гама шифру повинна змінюватися випадковим образом для кожного блоку, що шифрується. Якщо період гами перевищує довжину усього тексту, що шифрується, і зловмиснику невідома ніяка частина вихідного тексту, то такий шифр можна розкрити тільки прямим перебором усіх варіантів ключа. У цьому випадку криптостійкість шифру визначається довжиною ключа.

Однак, метод гаммирування стає неспроможним, якщо зловмисник довідається фрагмент вихідного тексту і відповідну йому шифрограму. Простим вирахуванням по модулі виходить відрізок псевдо випадкової послідовності і по ньому відновлюється вся послідовність.

Методи генерації псевдо випадкових послідовностей чисел

При шифруванні методом гаммирування як ключ використовується випадковий рядок бітів, що поєднується з відкритим текстом, також представленим у двійковому виді, за допомогою побітового додавання по модулі 2, і в результаті виходить шифрований текст. Генерування непередбачених двійкових послідовностей великої довжини є однієї з важливих проблем класичної криптографії. Для рішення цієї проблеми широко використовуються генератори двійкових псевдовипадкових послідовностей.

Генеровані псевдовипадкові ряди чисел часто називають гамою чи шифру просто гамою (за назвою букви γ грецького алфавіту, часто використовуваної в математичних формулах для позначення випадкових величин).

Звичайно для генерації послідовності псевдовипадкових чисел застосовують комп'ютерні програми, що, хоча і називаються генераторами випадкових чисел, насправді виробляють детерміновані числові послідовності, що по своїх властивостях дуже схожі на випадкові.

До криптографічно стійкого генератора ПСП чисел (гами шифру) пред'являються три основних вимоги:

– період гами повинний бути досить великим досить великим для шифрування повідомлень різної довжини;

– гама повинна бути практично непередбаченої, що означає неможливість пророчити наступний біт гами, навіть якщо відомі тип генератора і попередній шматок гами;

– генерування гами не повинне викликати великих технічних складностей;

Довжина періоду гами є самою важливою характеристикою генератора ПСЧ. По закінченні періоду числа почнуть повторюватися і їхній можна буде пророчити.

Один з перших способів генерації ПСЧ на ЕОМ запропонував у 1946 р. Джон фон Нейман. Суть цього способу полягає в тому, що кожне наступне випадкове число утвориться зведенням у квадрат попереднього числа з відкиданням цифр молодших і старших розрядів. Однак цей спосіб виявився ненадійним і від нього незабаром відмовилися.

З відомих процедур генерації послідовності ПСЧ найбільше часто застосовується так називаний *лінійний конгруентний генератор*. Цей генератор виробляє послідовність ПСЧ Y_1, Y_2, ..., Y_{i-1}, Y_i, ..., використовуючи співвідношення

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

де Y_i i-і (поточне) число послідовності;

Y_{i-1} попереднє число послідовності;

m – модуль;

a – множник;

b – збільшення;

a Y_0 - число, що породжує, (вихідне значення).

Поточне псевдовипадкове число Y_i одержують з попереднього числа Y_{i-1} множенням його на коефіцієнт a, додаванням зі збільшенням b і обчисленням залишку від розподілу на m. Дане рівняння генерує ПСЧ із періодом повторення,

що залежить від обраних значень a і b і може досягати значення m. Значення m звичайно встановлюється рівним 2^n, де n - довжина машинного слова в бітах, або рівним простому числу, наприклад $m=2^{31}-1$. Як показано Д. Батогом, лінійний конгруентний датчик ПСЧ має максимальний період тоді і тільки тоді, коли b - непарне, і a mod 4 = 1.

Також для одержання послідовності ПСЧ застосовуються адитивні і мультиплікативні генератори.

Мультиплікативний генератор виробляє послідовності чисел за допомогою рекурентного співвідношення:

$$Yi = (a \bullet Yi\text{-}1) \bmod m.$$

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

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

$$Yi = (Yi\text{-}1 + Yi\text{-}2) \bmod m.$$

Опис алгоритмів.

Алгоритм шифровки.

Проініціалізувати датчик випадкових чисел.

Виділити блок відкритого тексту.

Згенерувати гаму шифру.

Одержати блок зашифрованого тексту, склавши по модулі 2 блок відкритого тексту з гамою шифру.

Якщо текст не закінчився, перейти до пункту 2, інакше до пункту 6.

Кінець алгоритму шифровки.

Алгоритм дешифрування.

Проініціалізувати датчик випадкових чисел.

Виділити блок зашифрованого тексту.

Згенерувати гаму шифру.

Одержати блок відкритого тексту, склавши по модулі 2 блок зашифрованого тексту з гамою шифру.

Якщо зашифрований текст не закінчився, перейти до пункту 2, інакше до пункту 6.

Кінець алгоритму дешифрування.

Приклад виконання завдання.

Зашифрувати і розшифрувати текст, що знаходиться у файлі з ім'ям *Source.txt*. Закодований текст зберегти у файлі з ім'ям *Coded.txt*, розшифрований текст записати у файл *DeCoded.txt*. Для генерації гами використовувати мультиплікативний датчик зі значеннями a = 5, m = 4096, Y_0 = 4003.

```
// ------------------------------
// Текст програми
// ------------------------------
// Підключення заголовних файлів для роботи з файловими потоками
#include <iostream.h>
#include <fstream.h>
```

```
// Параметри датчика
const int a=5;
const int m=4096;
int y;
// Ініціалізація вихідного значення
void RndInit(void)
 {
  y=4003;
 }
// Генерація гами шифру
// t - масив з 64-х бітів (8-ми байт)
void Rnd(char *t)
 {
  int i;
  for(i=0; i<8; i++)
   {
    y=(a*y) % m;
    t[i]=y;
   }
 }

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

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

Контрольні питання.

1. Що таке гамування? Що розуміють під гамою шифру?
2. Які операції можна застосовувати при накладенні гами? У чому полягає процес шифровки і дешифрування?
4. Які достоїнства і недоліки в методу гамування?
5. Які вимоги пред'являються до криптографічно стійкого генератора ПСП? Чому найбільш важлива довжина періоду гами?
6. Опишіть лінійний конгруентний спосіб генерації ПСП.
7. Опишіть адитивний і мультиплікативний генератори ПСП.
8. Опишіть алгоритми шифровки і дешифрування відкритого тексту методом гамування.

Варіанти завдань.

Непарні варіанти пишуть програму шифровки тексту з застосуванням зазначеного методу, парні варіанти програмують дешифратор.

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

Для генерації гами використовувати:

1-2 адитивний датчик зі значеннями m = 4096, Y_0 = 4003, Y_1 = 59;

3-4 лінійний конгруентний датчик зі значеннями a = 5, b = 7, m = 4096, Y_0 = 4003, Y_1 = 59;

5-6 мультиплікативний датчик зі значеннями a = 7, m = 4096, Y_0 = 502;

7-12 датчик, період у який найбільший; датчики і значення для них вибрати з попередніх варіантів. Гаму генерувати за допомогою зазначеного датчика. Підібрати для нього такі значення параметрів, щоб період був найбільшим.

13-16 мультиплікативний датчик;

17-20 лінійний конгруентний датчик.

Лабораторна робота №4.

Тема: Симетричні криптосистеми: шифри складної заміни.

Мета: навчитися розроблювати програми для шифрування (дешифрування) методиками симетричних криптосистем.

Традиційні симетричні криптосистеми: шифри складної заміни

Одноразова система шифрування

Одноразова система винайдена в 1917 р. американцями Дж. Моборном і Г.Вернамом. Для реалізації цієї системи підстановки іноді використовують одноразовий блокнот. Цей блокнот складений з відривних сторінок, на кожній з який надрукована таблиця з випадковими числами (ключами) K_J. Блокнот виконується в двох екземплярах: один використовується відправником, а іншої - одержувачем. Для кожного символу X, повідомлення використовується свій ключ K_J з таблиці тільки один раз. Після того як таблиця використана, вона повинна бути вилучена з блокнота і знищена. Шифрування нового повідомлення починається з нової сторінки.

Цей шифр абсолютно надійний, якщо набір ключів K_J, дійсно випадковий і непередбачений. Якщо криптоаналітик спробує використовувати для заданого шифртекста всі можливі набори ключів і відновити всі можливі варіанти вихідного тексту, то вони усі виявляться рівноймовірними. Не існує способу вибрати вихідний текст, що був дійсно посланий. Теоретично доведено, що одноразові системи є системами, що не розкриваються, оскільки їхній шифртекст не містить достатньої інформації для відновлення відкритого тексту.

Опис послідовності дій для алгоритму одноразового шифрування.

Для шифрування деякого повідомлення A_0,..,An-1 складеного з букв алфавіту $A=\{a_0,...,a_{m-1}\}$ необхідно:

Побудувати схему заміщення букв алфавіту цілими числами від 1 до m $A \rightarrow X$, що дозволяє здійснювати зворотне перетворення $X \rightarrow A$. (Наприклад, a_0–1, a_1–2,…,a_{m-1}–m)

Перетворити вихідний текст у безліч X=(X_0,…...,Xn-1)

Одержати послідовність K=(X_0,…...,Xn-1) випадкових рівномірно розподілених чисел від 0 до m-1 (0<=K<=m-1).

Виконати підстановку Цезаря $Y_i=(X_i+K_i) \bmod m$

Для отриманої послідовності Y=(Y_0,...,Yn-1) зробити зворотне перетворення з числового представлення Y у букви алфавіту A : $Y \rightarrow B$.

Отримана послідовність B=(B_0,...,Bn-1) буде шифротекстом, а послідовність K=(K_0,…...,Kn-1) – ключем.

Для розшифрування повідомлення B_0,...,Bn-1 необхідно зробити дії в

зворотному порядку, замінивши підстановку Цезаря на формулу $X_i=(Y_i-K_i)\text{mod}m$.

Приклад програмної реалізації системи одноразового кодування

Програма дозволяє кодувати файли на основі ASCII таблиці

Текст програми:

```
program onetimecoding;
 var
  FileNameToCode,FileNameToDecode,FileNameWithSeed:string;
{одержує нову послідовність випадкових чисел}
procedure GetNewSeed(N,m:integer;FSeed:string);
 var
  F:file of byte;
  i:integer;
  Ki:byte;
  K:string;
 begin
  assign(F,FSeed);
  rewrite(F);
  randomize;
  for i:=1 to N do
   begin
    Ki:=random(m);
    write(F,Ki)
   end;
  close(F)
 end;
{кодує текст}
procedure
Code(FileName,Seed,FNameToSaveCodedText:string;m:integer);
 var
  Fprimery,Fcodedtext,Fcode:file of char;
  X,Y,K:char;
  i,j:integer;
 begin
  assign(Fprimery,FileName);
  assign(Fcode,Seed);
  assign(Fcodedtext,FNameToSaveCodedText);
  reset(Fprimery);
  GetNewSeed(filesize(Fprimery),m,Seed);
  reset(Fcode);
  rewrite(Fcodedtext);
  while not EOF(Fprimery) do
   begin
    read(Fprimery,X);
    read(Fcode,K);
    Y:=chr( (ord(X)+ord(K)) mod m );
    write(Fcodedtext,Y)
   end;
  close(Fprimery);
  close(Fcode);
  close(Fcodedtext);
 end;
```

```
{декодирует текст}
procedure
Decode(FileName,Seed,FNameToSaveDecodedText:string;m:integer);
 var
  Fcodedtext,Fcode,Fdecodedtext:file of char;
  X,Y,K:char;
 begin
  assign(Fcodedtext,FileName);
  assign(Fcode,Seed);
  assign(Fdecodedtext,FNameToSaveDecodedText);
  rewrite(Fdecodedtext);
  reset(Fcode);
  reset(Fcodedtext);
  while not EOF(Fcodedtext) do
   begin
    read(Fcodedtext,Y);
    read(Fcode,K);
    X:=chr( (ord(Y)-ord(K)) mod m );
    write(Fdecodedtext,X)
   end;
  close(Fdecodedtext);
  close(Fcode);
  close(Fcodedtext);
 end;
begin
 Code('text','seed','ctext',256);

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

Контрольні питання:

1. Що є характерною рисою симетричних систем?
2. Що таке ключ?
3. У чому полягає суть шифрування заміною(підстановкою)?
4. Що дозволяє спростити виконання необхідних алгебраїчних дій у процесі шифрування?
5. Чим відрізняються шифри складної заміни від простих шифрів підстановки?
6. Чому дана система шифрування називається одноразової?
7. На основі яких перетворень здійснюється кодування?
8. Які достоїнства і недоліки одноразової системи шифрування?
9. Чи можна розшифрувати закодоване повідомлення, не знаючи ключа ? Якщо так, то в яких випадках?
10. Який з випадків гірше з погляду дешифрування: при передачі був перекручений і-ий символ
    а) закодованого повідомлення
    б) ключа ?

Завдання

Скласти програму шифровки повідомлення за допомогою алгоритму одноразової системи шифрування і його розшифровок (непарний варіант –

шифровка повідомлення, парний – розшифровка)... Вихідне повідомлення, кодоване повідомлення, ключ і відновлене повідомлення зберігати у файлах. Алфавіт вихідного повідомлення:

Варіанти 1-10 - Англійський

Варіанти 11-20 – Російський

N варіанта	**Повідомлення**
1-2	THE MESSAGE
3-4	BE BACK TOMORROW
5-6	CODE ME PLEASE
7-8	WAIT ME ON THE SAME PLACE
9-10	DO YOU STILL TRY TO CODE ME
11-12	ПОВІДОМЛЕННЯ
13-14	ЗАВТРА ТАМ ЖЕ
15-16	ШИФР – ОСНОВА ЗАХИСТУ
17-18	А РОЗШИФРУВАТИ ЗМОЖЕШ
19-20	СПРОБУЙ СНОВУ

Лабораторна робота №5.

Тема: Шифрування алгоритмом DES.

Мета: навчитися писати програми шифрування (дешифрування) алгоритмом DES.

Опис алгоритму.

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

Слід відразу зазначити, що всі таблиці, що приводяться, є стандартними і повинні включаться в реалізацію алгоритму DES у незмінному виді. Усі перестановки і коди в таблицях підібрані розроблювачами таким чином, щоб максимально утруднити процес розшифровки шляхом підбора ключа.

Нехай з файлу вихідного тексту лічений черговий 64-бітовий (8-байтовый) блок T. Цей блок T перетвориться за допомогою матриці початкової перестановки IP. Біти вхідного блоку (64 бита) переставляються відповідно до матриці IP.

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

Матриця початкової перестановки IP.

Отримана послідовність бітів розділяється на 2 послідовності: $L_{0\text{-ліві}}$ чи старші біти, $R_{0\text{-праві}}$ чи молодші біти, кожна з який містить 32 бита. Потім виконується ітеративний процес шифрування, що складає з 16 циклів. Нехай T_i- результат і-ої ітерації:

$$T_i=L_iR_i,$$

де Li-перші 32 бита послідовності, Ri-останнього 32 бита послідовності. Тоді результат і-ої ітерації описується наступними формулами:

$$Li=Ri-1,\ i=1,2,3,\ \ldots,\ 16,$$

$$Ri=Li-1\ \text{xor}\ f\ (Ri-1,\ Ki),\ i=1,2,3,\ \ldots,\ 16...$$

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

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

По закінченні шифрування здійснюється відновлення позицій бітів за допомогою матриці зворотної перестановки IP^{-1}.

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

Матриця зворотної перестановки IP^{-1}.

Елементи матриць IP і IP^{-1} зв'язані.

Зв'язок елементів матриць

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

Процес розшифровування даних є інверсним стосовно процесу шифрування. Усі дії повинні бути виконані в зворотному порядку. Це означає, що дані, що розшифровуються, спочатку переставляються відповідно до матриці IP^{-1} ,а потім над послідовністю бітів $R_{16}L_{16}$ виконуються ті ж дії, що й у процесі шифрування, але в зворотному порядку.

Ітеративний процес розшифрування може бути описаний наступними формулами:

$$Ri=Li,\ i=1,2,3,\ \ldots,\ 16,$$

$$Li-1=Ri\ \text{xor}\ f\ (Li,\ Ki),\ i=1,2,3,\ \ldots,\ 16.\ R16L\ 16$$

Таким чином, для процесу розшифрування з переставленим вхідним блоком $R_{16}L_{16}$ на першій ітерації використовується ключ K_{16}, на другій ітерації – K_{15} і т.д. На 16-ий ітерації використовується ключ K_1. на останньому кроці ітерації будуть отримані послідовності L_0 і R_0, що конкатенуються в 64-бітову послідовність L_0R_0. Потім у цій послідовності 64 біта переставляються відповідно до матриці IP. Результат такого перетворення – вихідна послідовність бітів.

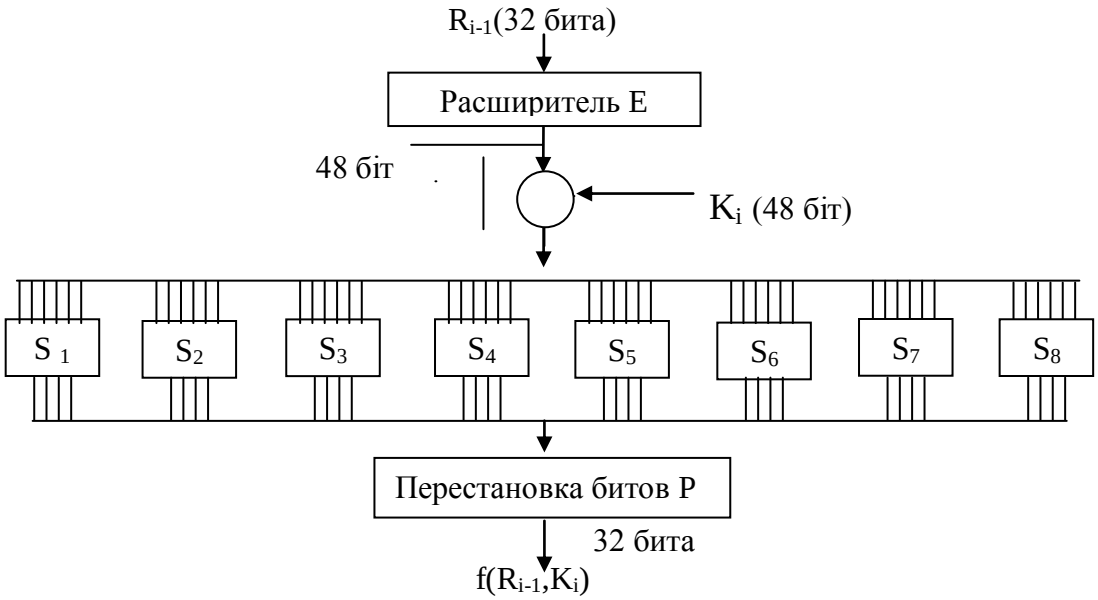


Рис 1. Схема обчислення функції шифрування f

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

функція E (розширення 32 біт до 48);

функція S_1,S_2, …,S_8 (перетворення 6-бітового числа в 4-бітове);

функція P (перестановка бітів у 32-бітовій послідовності).

Приведемо визначення цих функцій.

Функція розширення E приймає блок з 32 біт і породжує блок з 48 біт, відповідно до матриці E.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Матриця розширення E

Отриманий результат складається по модулі 2(операція XOR) з поточним значенням ключа K_i і потім розбивається на вісьмох 6-бітових блоків B_1,B_2, …,B_8... Далі кожний з цих блоків використовується як номер елемента у функціях S_1,S_2, …,S_8, утримуючих 4-бітові значення.

Нехай на вхід функції S_j надходить 6-бітовий блок $B_j=b_1b_2b_3b_4b_5b_6$, тоді двохбітове число b_1b_6 указує номер рядка матриці, а четирьохбітове число $b_2b_3b_4b_{5\text{-номер}}$ стовпця в наступній таблиці.

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

Функції перетворення S

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

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

Функція P перестановки бітів

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

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

Функція G первісної підготовки ключа

Результат перетворення G(K) розбивається на двох половин C_0 і D_0 по 28 біт кожна. Перші чотири рядки матриці G визначають, як вибираються біти послідовності $C_{0,\ що}$ випливають чотири рядки - послідовності D_0. Для генерації послідовностей З і Do не використовуються біти 8,16,24,32,40,48,56 і 64 ключа шифру. Таким чином, у дійсності ключ шифру є 56-бітовим.

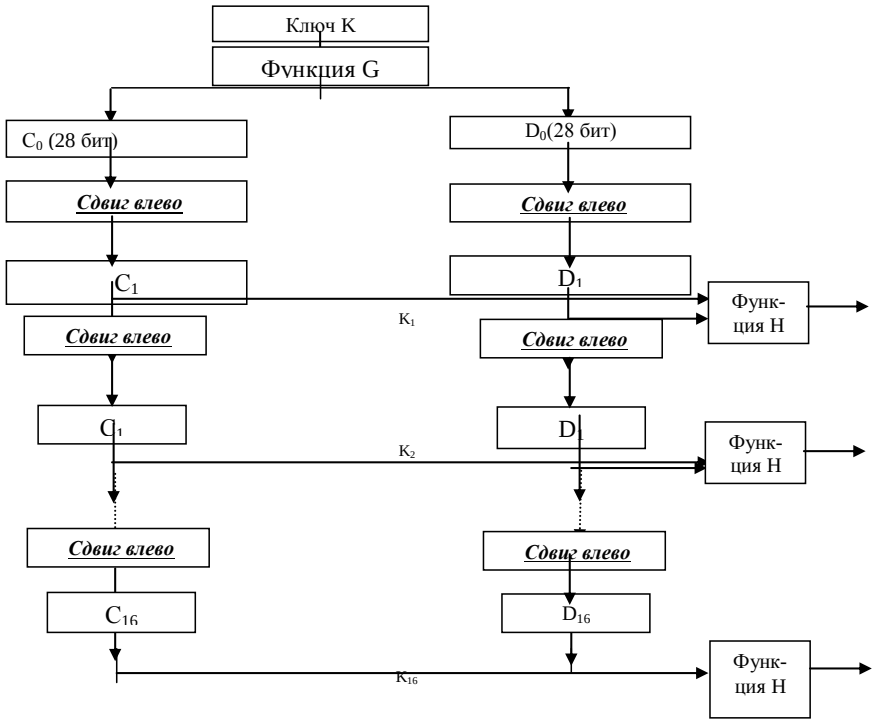


Рис 2.Схема обчислення ключів K_i

Після визначення З і Do рекурсивно визначаються C_i і D_i, i = 1, 2,..., 16. Для цього застосовуються операції циклічного зрушення вліво на один чи два бита в залежності від номера кроку ітерації, як показано в наступній таблиці.

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

Таблиця зрушень для обчислення ключа

Ключ Ki, обумовлений на кожнім кроці ітерації, є результат вибору конкретних бітів з 56-бітової послідовності C_i D_i, і їх перестановки. Іншими словами, ключ $Ki=H(C_i D_i)$.

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

Функція H завершальної обробки ключа

Основні режими роботи алгоритму DES

Алгоритм DES дозволяє безпосередньо перетворювати 64-бітовий вхідний відкритий текст у 64-бітовий вихідний шифрований текст, однак дані рідко обмежуються 64 розрядами. Щоб скористатися алгоритмом DES для рішення різноманітних криптографічних задач, розроблені чотири робітники режиму:

- електронна кодова книга ECB (Electronic Code Book);
- зчеплення блоків шифру CBC (Cipher Block Chaining);
- зворотний зв'язок по шифротексту CPB (Cipher Feed Back);
- зворотний зв'язок по виходу OFB (Output Feed Back).

Опис режимів див. У методичних вказівках для виконання практичних робіт.

Приклад виконання завдання

Процес шифрування:

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

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

Процес розшифрування:

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

Процедура перестановки perestanovkaIp може бути реалізована в такий спосіб:

```
procedure perestanovkaIP(var M64:Mas64);
 var M64temp:mas64;
 const vectorIp:array[1..64]of byte=
                (58,50,42,34,26,18,10,2,
                 60,52,44,36,28,20,12,4,
                 62,54,46,38,30,22,14,6,
                 64,56,48,40,32,24,16,8,
                 57,49,41,33,25,17, 9,1,
                 59,51,43,35,27,19,11,3,
                 61,53,45,37,29,21,13,5,
                 63,55,47,39,31,23,15,7);
  begin{ perestanovkaIP}
   for i:=1 to 64 do  M64temp[vectorIp[i]]:=M64[i];
   for i:=1 to 64 do  M64[i]:=M64temp[i];
 end{ perestanovkaIP}.
```

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

Функція F може бути представлена так:

```
 procedure F(R32_local:mas32;k48_local:mas48;
 var Rout_local:mas32);

       procedure S(B6:mas6;jj:byte;var B4:mas4 );
       const vector_s:array[0..31,0..15]of byte=
       ((14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7),…(2,1,14,7,4,10,8,
13,15,12,9,0,3,5,));
                  var nstr,nstlb,z:byte;
                  begin{S}
                  nstr:=0;
                  if b6[1]=1 then nstr:=nstr+2;
                  if b6[6]=1 then nstr:=nstr+1;
                  if b6[2]=1 then nstlb:=nstlb+8;
                  if b6[3]=1 then nstlb:=nstlb+4;
                  if b6[4]=1 then nstlb:=nstlb+2;
                  if b6[5]=1 then nstlb:=nstlb+1;
                  z:=vector_s[nstlb,jj*4+nstr];
                  if z>=8 then begin B4[1]:=1; z:=z-8; end;
                  if z>=4 then begin B4[1]:=1; z:=z-4; end;
                  if z>=2 then begin B4[1]:=1; z:=z-2; end;
                  if z>=1 then B4[1]:=1;
                  end{S};

  begin{ functionf }
    E(R32_local,R48_local);
    XOR48(R48_local,k48_local,Rk48_local);
    for j:=1 to 8 do
      begin { for j }
       for i:=1 to 6 do b6[i]:=Rk48_local[i+(j-1)*6];
        s(b6,j,b4);
       for i:=1 to 4 do Rout_local[i+(j-1)*4]:=b4[i];
      end; { for j }
       for i:=1 to 32 do
        bm3[i]:=Rout_local[Vector_P[i]];
       Rout_local:=bm3;
  end{ functionf };
```

Контрольні питання

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

Варіанти завдань

Реалізувати алгоритм DES (шифрування і дешифрування)

1-2. Робочий режим EBC.

3-4. Робочий режим CBC.

5-6. Робочий режим CFB.

Робочий режим OBC.

Лабораторна робота №6.

Тема: Шифрування методом Вернама.

Мета: навчитися писати програми для шифрування (дешифрування) методом Вернама.

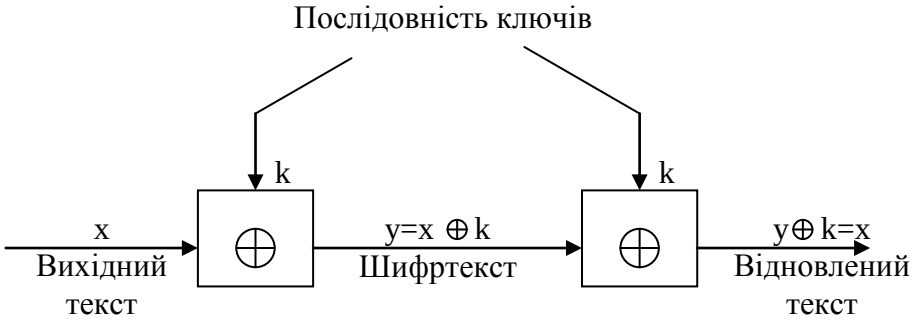
Теоретичний матеріал

Система Вернама є частиною системи підстановок Вижинера при *m*=2. Конкретна версія цього шифру, запропонована в 1926*р.* Гильбертом Вернамом. Стародавній телетайп фірми AT&T із пристроєм, що зчитує, Вернама й устаткуванням для шифрування, використовувався корпусом зв'язку армії США.

Кожна буква вихідного тексту в алфавіті, розширеному деякими додатковими знаками, спочатку переводилася з використанням *телеграфного коду Бодо* в п'ятбітовий блок($b_0, b_1 .. b_4$). До вихідного тексту Бодо додавався (по модулі 2) ключ $k=(k_0, k_1, ..., k_{к-1})$ записаний на паперовій стрічці.

Моделювання кодера і декодера

Схема передачі повідомлень з використанням шифрування методом Вернама показана на малюнку:



Шифрування вихідного тексту, попередньо перетвореного в послідовність двійкових символів **x**, здійснюється шляхом додавання по модулі 2 символів **x** з послідовністю двійкових ключів **k**. Одержимо символи шифртекста :

$y = x \oplus k$

Розшифровка складається в додаванні по модулі 2 символів у шифр тексту з тією же послідовністю ключів k:

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

При цьому послідовності ключів, використані при шифруванні і розшифруванні, компенсують один одного (при додаванні по модулі 2), і в

результаті відновлюються символи x вихідного тексту.

При моделюванні, перетворення вихідного тексту, у послідовність двійкових символів, можна здійснити в такий спосіб:

```
for(i=0;i<n*sizeof(char)*8;i++)

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

тут:

n- кількість елементів у тексті;

m- масив двійкових символів;

8- кількість біт у байті;

str- рядок утримуюча текст.

Кодер може мати наступний вид:

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

тут:

k- послідовність ключів

M- довга послідовності ключів

Декодер цілком збігається з кодером.

При розробці системи Вернам перевіряв за допомогою закільцьованих стрічок, установлених на передавачі і приймачі для того, щоб використовувалася одна і та ж послідовність ключів.

Слід зазначити, що метод Вернама не залежить від довгі послідовності ключів і, крім того, він дозволяє використовувати випадкову послідовність ключів. Однак при реалізації методу Вернама виникають серйозні проблеми, зв'язані з необхідністю доставки одержувачу такої ж послідовності ключів, як і у відправника, або з необхідністю безпечного збереження ідентичних послідовностей ключів у відправника й одержувача.

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

$$yn+1 = (yn + yn-1) \bmod m$$

тут mod - залишок від розподілу;

Параметри датчика можна взяти наступними: m=4096*4; y_1=4091; y_2=m-5.

Контрольні питання

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

Варіанти завдань

Змоделювати кодер з послідовністю ключів з 5 елементів.

Змоделювати декодер з послідовністю ключів з 5 елементів.

Змоделювати кодер з послідовністю ключів з 17 елементів.

Змоделювати кодер з послідовністю ключів з 17 елементів.

Змоделювати кодер з нескінченною послідовністю ключів формованих за допомогою адитивного псевдо-випадкового датчика випадкових чисел.

Змоделювати декодер з нескінченною послідовністю ключів формованих за допомогою адитивного псевдо-випадкового датчика випадкових чисел.

Лабораторна робота №7.

Тема: Алгоритм шифрування даних IDEA.

Мета: навчитися розроблювати програми для шифрування (дешифрування) алгоритмом IDEA.

Теоретичний матеріал.

Алгоритм шифрування даних IDEA

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

Перша версія алгоритму IDEA була запропонована в 1990 р., її автори - Х.Лий і Дж.Мэсси. Первісна назва алгоритму PES (Proposed Encryption Standard). Поліпшений варіант цього алгоритму, розроблений у 1991 р., одержав назву IPES (Improved Proposed Encryption Standard). У 1992 р. IPES змінив своє ім'я на IDEA. Як і більшість інших блокових шифрів, алгоритм IDEA використовує при шифруванні процеси змішування і розсіювання, причому всі процеси легко реалізуються апаратними і програмними засобами.

В алгоритмі IDEA використовуються наступні математичні операції:

– порозрядне додавання по модулі 2 (операція " що виключає ЧИ"); операція позначається як ⊕;

– додавання беззнакових цілих по модулі 2 (модуль 65536); операція позначається як +;

– множення цілих по модулі (2^{16}+1) (модуль 65537), розглянутих як беззнакові цілі, за винятком того, що блок з 16 нулів розглядається як 2^{16}; операція позначається як *.

Всі операції виконуються над 16-бітовими субблоками. Ці три операції несумісні в тім змісті, що:

– ніяка пара з цих трьох операцій не задовольняє асоціативному закону, наприклад a + (b □ c) □ (a + b) □ c;

– ніяка пара з цих трьох операцій не задовольняє дистрибутивному закону, наприклад a + (b * c) □ (a + b) * (a + c).

Комбінування цих трьох операцій забезпечує комплексне перетворення входу, істотно утрудняючи криптоаналз IDEA у порівнянні з DES, що базується винятково на операції " що виключає ЧИ".

Розшифрування здійснюють аналогічним образом, за винятком того, що порядок використання підключів стає зворотним, причому ряд значених підключів заміняється на зворотні значення. Підключі розшифрування являються в основному або адитивними, або мультиплікативними зворотнім величинам

підключів шифрування (табл.1).

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

Таблиця 1. Підключи шифрування і розшифрування алгоритму IDEA.

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

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

Завдання до самостійної роботи.

Реалізувати програмним шляхом 1-4 операції циклу алгоритму IDEA у режимі шифрування. Оформити програму у виді процедур і функцій.

Реалізувати програмним шляхом 5-9 операції циклу алгоритму IDEA у режимі шифрування. Оформити програму у виді процедур і функцій.

Реалізувати програмним шляхом 10-14 операції циклу алгоритму IDEA у режимі шифрування. Оформити програму у виді процедур і функцій.

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

Використовуючи програми завдань 1-4, змоделювати програмним шляхом алгоритм IDEA у режимі шифрування.

Реалізувати програмним шляхом 1-4 операції циклу алгоритму IDEA у режимі розшифрування. Оформити програму у виді процедур і функцій.

Реалізувати програмним шляхом 5-9 операції циклу алгоритму IDEA у режимі розшифрування. Оформити програму у виді процедур і функцій.

Реалізувати програмним шляхом 10-14 операції циклу алгоритму IDEA у режимі розшифрування. Оформити програму у виді процедур і функцій.

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

Використовуючи програми завдань 6-9, змоделювати програмним шляхом алгоритм IDEA у режимі розшифрування.

Лабораторна робота №8.

Тема: Системи з відкритим ключем. Алгоритм RSA.

Мета: навчитися писати програми шифрування (дешифрування) алгоритмом RSA.

Системи з відкритим ключем (СІК)

Суть цих систем полягає в тому, що кожним адресатом ІС генеруються два ключі, зв'язані між собою за визначеним правилом. Один ключ з'являється *відкритим*, а іншої *закритим*. Відкритий ключ публікується і доступний кожному, хто бажає послати повідомлення адресату. Секретний ключ зберігається в таємниці.

Вихідний текст шифрується відкритим ключем адресата і передається йому. Зашифрований текст у принципі не може бути розшифрований тим же відкритим ключем. Дешифрування повідомлення можливе тільки з використанням закритого ключа, що відомий тільки самому адресату.

Криптографічні системи з відкритим ключем використовують так називані *необоротні чи однобічні функції*, що мають наступну властивість: при заданому значенні x відносно просто обчислити значення $f(x)$, однак якщо $y=f(x)$, то немає простого шляху для обчислення значення x.

Безліч класів необоротних функцій і породжує вся розмаїтість систем з відкритим ключем. Однак не всяка необоротна функція годить для використання в реальних ІС.

У самім визначенні необоротності 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 систем з відкритим ключем (СІК) пред'являються дві важливих і очевидних вимоги:

1. Перетворення вихідного тексту повинне бути необоротним і виключати його відновлення на основі відкритого ключа.
2. Визначення закритого ключа на основі відкритого також повинне бути неможливим на сучасному технологічному рівні. При цьому бажана точна нижня оцінка складності (кількості операцій) розкриття шифру.

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

Узагалі ж усі пропоновані на сьогодні криптосистеми з відкритим ключем спираються на один з наступних типів необоротних перетворень:

Розкладання великих чисел на прості множники.

Обчислення логарифма в кінцевому полі.

Обчислення коренів алгебраїчних рівнянь.

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

1. Як *самостійні засоби захисту* переданих і збережених даних.

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

Засобу аутентифікації користувачів.

Розглянемо найбільше розповсюджену систему з відкритим ключем.

Алгоритм RSA. Послідовність дій алгоритму RSA.

Припустимо, що користувач А хоче передати користувачу В повідомлення в зашифрованому виді, використовуючи криптосистему RSA. У такому випадку користувач А виступає в ролі відправника повідомлення, а користувач В – у ролі одержувача. Розглянемо послідовність дій користувача В и користувача А.

Користувач В вибирає два довільних великих простих числа P і Q.

Користувач В обчислює значення модуля N=P*Q.

Користувач В обчислює функцію Эйлера φ(N) і вибирає випадковим образом значення відкритого ключа e, взаємно простого з φ(N), , з урахуванням умови (4).

Користувач В обчислює значення секретного ключа d, використовуючи формулу (3).

Користувач В пересилає користувачу А парі чисел (e,N) по незахищеному каналі.

Якщо користувач А хоче передати користувачу В повідомлення M, він виконує наступні кроки.

Користувач А розбиває вихідний відкритий текст M на блоки, кожний з який може бути представлений у виді числа

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

Користувач А шифрує текст, представлений у виді послідовності чисел M_i по формулі

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

і відправляє криптограму $\{C_i\}$ користувачу В.

Користувач В розшифровує прийняту криптограму $\{C_i\}$, використовуючи закритий ключ d по формулі

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

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

Приклад виконання шифровки і дешифрування в системі RSA розглядається в методичних вказівках до лабораторних робіт.

Приклад програми шифрування по алгоритму RSA

Розглянемо ділянку програми, що робить шифрування символів із вхідного файлу F_in:

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

Тут висновок криптограми виробляється й у файл F_out і на екран. Функція St(a,b) зводить число a у ступінь b. Щоб перейти від цифрового символу до символу з таким кодом (наприклад, замість символу '1' використовувати символ з кодом 1, тобто #1) віднімаємо з коду цього символу (byte(c)) код символу '0' (byte(0)=48). Потім для запису у файл знову відновлюємо символ по його коду (char(c)), а для висновку на екран залишаємо код^-число-код отриманого символу.

Контрольні питання:

1. У чому складається суть систем з відкритим ключем?
2. За допомогою яких ключів шифрується і розшифровується повідомлення в СІК?
3. Що таке необоротної функції? Які типи необоротних перетворень використовуються в СІК?
4. Які головні вимоги пред'являються до СІК?
5. Для яких призначень можна використовувати алгоритми криптосистем з відкритим ключем?
6. На яких математичних фактах заснований алгоритм RSA?
7. Як вибираються числа P і Q алгоритму RSA?
8. Чи можна розшифрувати повідомлення за допомогою відкритого ключа?
9. За допомогою яких формул виробляється шифровка і дешифрування повідомлення?
10. Чи зміниться криптограма, якщо числа P і Q поміняти місцями?
11. Нехай P=3, Q=13. Які з чисел: 2, 3, 5, 9, 29 можна використовувати як відкритий ключ e? Чому?

Завдання

Скласти програму шифровки повідомлення за допомогою алгоритму RSA і його розшифровки (непарний варіант – шифровка повідомлення, парний – розшифровка). Для представлення даних результату зведення в ступінь використовувати тип LongInt. Щоб не виникало помилки переповнення, вихідне повідомлення розглядати як послідовність символів з кодами 0, 1, ..., 9. Вихідне повідомлення, криптограму і відновлене повідомлення зберігати у файлах.

N варіанта	P	Q	E	d	Вихідний алфавіт
1-2	2	11	3	7	1234567890
3-4	2	11	7	3	1234567890

5-6	2	17	3	11	1780
7-8	3	11	3	7	1267890
9-10	3	11	7	3	1234567890
11-12	3	13	5	5	1234567890
13-14	3	17	11	3	12345670
15-16	5	7	5	5	1234567890
17-18	2	5	3	7	1234567890
19-20	2	7	5	5	1234567890

Лабораторна робота №9.

Тема: Схема шифрування Поліга-Хеллмана.

Мета: навчитися писати програми для шифрування (дешифрування) методом Поліга-Хеллмана.

Схема шифрування Полига-Хеллмана

Схема шифрування Полига-Хеллмана подібна зі схемою шифрування RSA. Вона являє собою несиметричний алгоритм, оскільки використовуються різні ключі для шифрування і розшифрування. У той же час цю схему не можна віднести до класу криптосистем з відкритим ключем, тому що ключі шифрування і розшифрування легко виводяться один з іншого. Обидва ключі потрібно тримати в секреті.

Аналогічно схемі RSA криптограма C і відкритий текст P визначаються зі співвідношень:

$C = P^e \bmod n$,

$P = C^d \bmod n$,

де $e*d \equiv 1$ (по модулі деякого складеного числа).

На відміну від алгоритму RSA у цій схемі число n не визначається через два великих простих числа; число n повинне залишатися частиною секретного ключа. Якщо хто-небудь довідається значення e і n, він зможе обчислити значення d.

Не знаючи значень e чи d, супротивник буде змушений обчислять значення $E=\log_p C \pmod n$.

Відомо, що це є важкою задачею.

Схема шифрування Полига-Хеллмана запатентована в США і Канаді.

Опис послідовності дій алгоритму шифрування Полига-Хеллмана.

Припустимо, що користувач А хоче передати користувачу В повідомлення в зашифрованому виді, використовуючи криптосистему Полига-Хеллмана. У такому випадку користувач А виступає в ролі відправника повідомлення, а користувач У – у ролі одержувача. Розглянемо послідовність дій користувача В и користувача А.

1. Користувач У вибирає два довільних великих числа e і d, щоб вони мали тільки один загальний дільник - одиниця.
2. Користувач В обчислює просте число n.
3. Користувач У пересилає користувачу А парі чисел (e,n) по **захищеному** каналі.

4. Користувач А хоче передати користувачу В повідомлення Р, він виконує наступні кроки.
5. Користувач А розбиває вихідний відкритий текст Р на блоки, кожний з який може бути представлений у вигляді числа

$$P_i = 0, 1, \ldots, n\text{-}1...$$

6. Користувач А шифрує текст, представлений у виді послідовності чисел P_i по формулі

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

7. відправлення криптограму $\{C_i\}$ користувачу В.
8. Користувач У розшифровує прийняту криптограму $\{C_i\}$, використовуючи закритий ключ d по формулі

$P_i=C_i^d \pmod n$.

У результаті буде отримана послідовність чисел $\{P_i\}$, що являють собою вихідне повідомлення Р. Щоб алгоритм Полига-Хеллмана мав практичну цінність, необхідно мати можливість оперативно обчислювати значення ключів e і d.

Приклад виконання шифровки і дешифрування в криптосистемі Полига-Хеллмана.

Зашифруємо повідомлення “АДЖ”. Для простоти будемо використовувати маленькі числа (на практиці застосовуються набагато більші).

Виберемо e=25 і d=9, n=29.

Представимо повідомлення, що зашифровується, як послідовність цілих чисел за допомогою відображення: А→1, Д→5, Ж→7. Тоді повідомлення приймає вид (1,5,7). Зашифруємо повідомлення за допомогою ключа {25,29}.

$C_1 = (1^{25}) \pmod{29} = 1$,

$C_2 = (5^{25}) \pmod{29} = 13$,

$C_3 = (7^{25}) \pmod{29} = 23$.

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

$P_1 = (1^9) \pmod{29} = 1$,

$P_2 = (13^9) \pmod{29} = 5$,

$P_3 = (23^9) \pmod{29} = 7$.

Таким чином, відновлене вихідне повідомлення: АДЖ.

Шифрування по алгоритму Полига-Хеллмана

Розглянемо ділянку програми, що робить шифрування символів із вхідного файлу text:

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

Тут висновок криптограми виробляється й у файл crypt і на екран.

Контрольні питання:

1. Які переваги системи RSA перед алгоритмом Полига-Хеллмана?
2. Як вибираються числа e, d, n у схемі Полига-Хеллмана?
3. Які значення творець криптосистеми по алгоритму Полига-Хеллмана повідомляє користувачам, а які зберігає в таємниці?
4. Чи можна розшифрувати повідомлення, знаючи тільки один ключ?
5. За допомогою яких формул виробляється шифровка і дешифрування повідомлення?

Завдання

Скласти програму шифровки повідомлення за допомогою алгоритму Полига-Хеллмана і його розшифровки (непарний варіант – шифровка повідомлення, парний – розшифровка). Для представлення даних результату зведення в ступінь використовувати тип long int. Щоб не виникало помилки переповнення, вихідне повідомлення розглядати як послідовність символів з кодами 0, 1, ..., 9. Вихідне повідомлення, криптограму і відновлене повідомлення зберігати у файлах.

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

Лабораторна робота №10.

Тема: Схема шифрування Ель Гамаля.

Мета: навчитися розроблювати програми для метода шифрування Ель Гамаля.

Теоретичний матеріал.

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

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

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

Потім обчислюють числа

$$Y_1=G^k \bmod P$$

$$Y_2=(Y^k \bmod P) \oplus M$$

Пари чисел (A,B) є шифртекстом. Помітимо, що довжина шифртексту вдвічі більше довжини вихідного відкритого тексту M.

Для того, щоб розшифрувати шифртекст (A,B), обчислюють

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

Приклад.

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

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

Отже, відкритий ключ Y = 6.

Нехай повідомлення M = 5. Виберемо деяке випадкове число K = 9. Переконаємося, що НОД (9, 10) = 1. Вираховуємо пари чисел (Y_1,Y_2):

$Y_1 = G^k \bmod P = 8^9 \bmod 11 = 134217728 \bmod 11 = 7$,

$Y_2 = (Y^k \bmod P) \oplus M = (6^9 \bmod 11) \oplus 5 = (10077696 \bmod 11) \oplus 5 = 2 \oplus 5 = 7$

Одержимо шифртекст (7,7).

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

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

У реальних схемах шифрування необхідно використовувати як модуль P велике ціле простої число, що має в двійковому представленні довжину 512 ... 1024 біт.

До переваг даної системи можна віднести те, що для обміну повідомленнями немає необхідності передавати секретний ключ по каналах зв'язку. Цим достоїнством володіють усі системи з відкритим ключем.

До недоліків даної системи можна віднести великі витрати обчислювальних ресурсів на реалізацію операцій з великими числами, при тім же рівні захищеності, що й інші криптосистеми. Другим недоліком є подвоєння довжини повідомлення.

Алгоритм

Розподіл ключів:

Вибрати просте число P і ціле G (G<P)

Вибрати секретний ключ A < P

Обчислити відкритий ключ $Y=G^A \bmod P$

Поширити серед групи користувачів числа P, G, Y

Шифрування повідомлення M:

Вибрати випадкове 1 < K < P-1, при якому K і P-1 – взаємно прості числа

Обчислити пари чисел Y_1 і Y_2 по формулах.

Передати пари (Y_1,Y_2)

Розшифровка повідомлення (Y_1,Y_2):

Обчислити M по формулі $M = Y_2 \oplus (Y_1^A \bmod P)$

Приклад виконання завдання.

```
program ElGamal;
Var p,g,a,k,k1,y,key:LongInt;
   y1,y2:Char;
   i:Word;
```

```
   F_in :Text;
   F_out :Text;
   c:Char;
   o:char;
Function NOD( M,N:LongInt ):LongInt;
Begin
 while( M<>0) and (N<>0 ) do
  Begin
    if( M>N )
    Then M:=M mod N
    Else N:=N mod M;
  End;
 if(N<>0) then NOD:=N
 else NOD:=M;
End;
Function IsSimple( p:LongInt ): Boolean;
Var k,m:LongInt;
   l:Extended;
Begin
 l:=sqrt( p );
 k:=1;
 m:=1;
 while( (k<l)and(m<>0) ) do
  begin
   inc(k);
   m:=p mod k;
  end;
 if( m<>0 ) Then IsSimple:=true
 Else IsSimple:=False;
End;
Function SimpleNum( var p:longint ): LongInt;
Begin
  while( not IsSimple( p ) ) do inc(p);
  SimpleNum:=p;
End;
Function st( g,k:LongInt ):LongInt;
Var Pr:LongInt;
Begin
 Pr:=1;
 for i:=1 to k do
   Pr:=Pr*g;
 st:=Pr;
End;
Begin
{ Writeln( 'Введіть просте число p' );
 readln( P );
 Writeln( 'Введіть ціле число g' );
 Readln(g);
 Writeln( 'Введіть секретний ключ:' );
 Readln(a);}
 p:=11;
 g:=9;
 a:=3;
```

```
 Writeln( 'p=', SimpleNum(p) );
 y:=st(g,a) mod p;
 writeln('Відкритий ключ - y=',y);
 Assign( F_in, 'in.dat' );
 Reset( F_in );
 Assign( F_out, 'out.dat' );
 Rewrite( F_out );
 randomize;
 While( not eof( F_in )) do
   Begin
     k1:=random(p-2)+1;
     {Пошук взаємно простого числа з числом p-1 у всіх числах,
великих k }
     k:=k1;
     While( (NOD( k, p-1 )<>1) and (k<p-1) or (k=31)) do inc(k);
     if( k=1 )
     Then
      Begin
     {Пошук взаємно простого числа з числом p-1 у всіх числах,
менших k }
       k:=k1;
       While( NOD( k, p-1 )<>1  ) do dec( k );
       if( k=p-1 )
       Then
         Begin
           Writeln( 'Помилка!!! Не можна знайти придатне к' );
           halt;
         End;
       End;
     y1:=char(st(g,k) mod p);
     Write( F_out, y1 );
     key:=st(y,k) mod p;
     Read( F_in, c );
     y2:=char(byte(c) xor key);
     Write( F_out, y2 );
   End;
 Close( F_in );
 Close( F_out );
 { Дешифрування }
 Assign( F_in, 'in2.dat' );
 Rewrite( F_in );
 Assign( F_out, 'out.dat' );
 Reset( F_out );
 While( not eof( F_out )) do
   Begin
     Read( F_out, y1 );
     Read( F_out, y2 );
     c:=char(((st( LongInt(y1),a ) mod p) xor byte(y2)));
     Write( F_in, c );
   End;
 Close( F_out );
 Close( F_in );
End.
```

Контрольні питання:

1. Якими властивостями повинні володіти числа, що використовуються в системі Ель Гамаля?
2. Якщо вихідне повідомлення складається з двох чисел, яка буде довжина зашифрованого повідомлення?
3. Напишіть формулу для обчислення відкритого ключа.
4. Напишіть формули для обчислення шифротексту, та для розшифровки отриманого повідомлення.

Варіанти завдань.

№ варіанта	P	g	a
1-2	11	9	3
3-4	11	9	8
5-6	11	9	5
7-8	7	6	5
9-10	11	3	6

Лабораторна робота №11.

Тема: Перевірка вірогідності ключа.

Мета: вивчити методи перевірки вірогідності ключа.

Елементи теорії

Як вводити ключ?

Як ви збираєтеся вводити ключ у програму шифрування? Простіше всього – із клавіатури. Саме цей варіант реалізований у більшості готових програм, і знайомство з ними починається з увічливого запрошення: «Enter your password: ... ». Тільки майте через, що пароль, що вводиться з клавіатури, можна підглянути. Якщо програма шифрування при введенні пароля відображає його на екрані, такою програмою краще не користатися. Коли при зашифруванні файлу ви вводите пароль, він не повинний відображатися на екрані.

А що трапиться, якщо при введенні пароля ви випадково натиснули не ту клавішу? Ви думаєте, що ввели один пароль, а насправді ввели іншої. Пробуєте розшифрувати файл, а програма говорить: « Пароль неправильний ». Щоб такого не було, звичайно програми шифрування при введенні пароля для зашифрування тексту просять увести пароль двічі. Якщо користувач у перший раз ввів один пароль, а в друг раз – іншої, виходить, принаймні один раз він помилився. А якщо обидва рази користувач увів те саме, виходить, усі нормально. Навряд чи користувач двічі помилиться однаково.

Як перевіряти правильність ключа?

До речі, а як програма при розшифруванні файлу визначає, що пароль правильний? По-різному. Деякі програми взагалі не перевіряють правильність пароля. У цьому випадку, якщо ви ввели неправильний пароль, файл як би розшифрується, але ви побачите зовсім не те, що зашифрували. Це незручно. Так що краще, коли перед розшифровкою програма перевіряє правильність пароля.

Залишається питання: як перевіряти правильність пароля? Можна просто

вписати пароль у початок зашифрованого файлу перед шифртекстом. При розшифруванні ви вводите пароль, програма читає початок зашифрованого файлу і порівнює те, що ви ввели, і те, що у файлі. Якщо збіглося, значить пароль правильний. А якщо не збіглося – неправильний. Просто і зручно. Тільки що відбудеться, якщо хто-небудь інший випадково перегляне зашифрований файл, наприклад, за допомогою Norton Commander? Весь файл – суцільна абракадабра, а на початку файлу – осмислене слово. Не потрібно бути семи п'ядей у чолі, щоб догадатися, що це і є пароль.

Очевидно, еталон пароля, що зберігається в зашифрованому файлі, теж треба зашифрувати. Тільки як? Найпростіше зашифрувати пароль по тій же схемі, що і текст вихідного файлу. Якщо шифр стійкий (а інакше його і не варто використовувати), пароль буде закритий надійно. А що брати як ключ? Можна взяти константу. Тоді в кожнім зашифрованому файлі еталон пароля буде зашифрований на тому самому ключі. Так, наприклад, робить убудована система шифрування файлів електронної пошти Sprint Mail. Тільки там зашифрований пароль зберігається не на початку зашифрованого файлу, а наприкінці. Але що буде, якщо хтось довідається ключ, на якому шифруються всі паролі? Він зможе розшифрувати усі файли, що ви зашифруєте. І не важливо, що пароль різні – зловмисник візьме потрібний пароль прямо з файлу, що хоче прочитати.

Ви, швидше за все, подумали: а відкіля зловмисник довідається ключ, на якому шифруються еталони ключів? Цей ключ убудований у програму шифрування, його ніхто не знає, навіть ви його не знаєте. Проте, якщо зловмисник досить кваліфікований, якщо він уміє користатися дизасемблером і відладчиком, цей ключ він довідається без праці. Звичайно на вирішення такої задачі іде усього кілька годин. Як це можна зробити – тема окремої статті. Поки повірте на слово – маючи тільки exe-файл, визначені навички і багато вільного часу, можна розібратися в тім, що робить програма, до дрібних подробиць.

Краще шифрувати еталон пароля сам на собі. Узяти пароль як відкритий текст і взяти той же пароль як ключ. При розшифруванні файлу, коли ви вводите пароль, програма намагається розшифрувати тільки початок файлу. Якщо в результаті вийшов рядок, що збігається з паролем, виходить, пароль правильний, і можна розшифровувати файл далі. А якщо вийшло щось інше, значить пароль неправильний.

Деякі програми шифрування шифрують паролі інакше. Береться якийсь рядок, завжди та сама, шифрується на паролі і записується в зашифрований файл. Diskreet, наприклад, шифрується на паролі в рядок **«ABCDEFGHENRIXYZ»** (цей рядок завершується нульовим байтом, як прийнято в мові C). Коли Diskreet перевіряє пароль, він берет начало зашифрованого файлу (точніше байти з 16-го по 31-й) і розшифровує їх на паролі, що ввів користувач. Якщо після розшифрування вийшов **«ABCDEFGHENRIXYZ»** – пароль правильний, якщо не вийшло – неправильний.

На перший погляд здається, що ця схема гірше, ніж попередня. Дійсно, у попередньому випадку зловмиснику не відомі ні відкритий текст, ні ключ, а в останньому випадку невідомий тільки ключ, а відкритий текст відомий. Але остання схема нітрохи не краще попередньої. Якщо шифр стійкий, то ключ шифрування неможливо одержати за прийнятний час, навіть якщо відомі і відкритий текст і шифртекст.

Приклад виконання завдання

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

```
Program Verify_Password;
Uses Strings;
  {Додайте текст функцій кодування і декодування в
   відповідності з вашим методом}
  Function Coding( C : Char; Key : String ):Char;
  Begin
  End;
  Function Decoding( C : Char; Key : String ):Char;
  Begin
  End;

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

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

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

Begin
  Assign( F_in, 'text.dat' );
  Reset( F_in );
  Assign( F_out, 'coding.dat' );
  Rewrite( F_out );
  Write('Уведіть пароль для шифрування:');
  Readln( Key );
  AddPassword( Key, F_out);
```

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

Контрольні питання

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

Завдання

Виконати одну з попередніх лабораторних робіт із криптографії, додавши перевірку правильності ключа одним з описаних методів.

Додавання пароля в початок зашифрованого файлу.

Шифрування пароля по тій же схемі, що і тексту. Як ключ узяти константу.

Шифрування пароля сам на себе.

Шифрування константного рядка.

Перелік рекомендованої літератури

Ю.В. Романец, П.А. Тимофеев, В.Ф. Шаньгин “Защита информации в компьютерных системах и сетях” 2015