МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КІРОВОГРАДСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ОСНОВИ ЗАХИСТУ ІНФОРМАЦІЇ

Методичні вказівки до виконання лабораторних робіт
з дисципліни «Основи захисту інформації»
для студентів денної та заочної форми навчання
напряму 8.050102 Комп’ютерна інженерія

Кіровоград 2013

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КІРОВОГРАДСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Мелешко Є.В., Смірнов О.А.

ОСНОВИ ЗАХИСТУ ІНФОРМАЦІЇ

Методичні вказівки до виконання лабораторних робіт
з дисципліни «Основи захисту інформації»
для студентів денної та заочної форми навчання
напряму 8.050102 Комп'ютерна інженерія

Затверджено
на засіданні кафедри
програмного забезпечення
Протокол №4
від 9.10.2013

Кіровоград 2013

Основи захисту інформації. Методичні вказівки до виконання лабораторних робіт для студентів денної та заочної форми навчання напряму 8.050102 «Комп'ютерна інженерія» / Укл.: Є.В. Мелешко, О.А. Смірнов – Кіровоград: КНТУ, 2013. – 46 с.

Дані методичні вказівки присвячені методології проектування й розробки систем захисту інформації. Ця методологія ввібрала в себе останні досягнення в області захисту інформації. Методичні вказівки адресовані майбутнім керівникам відповідних відділів та розроблювачам програмного забезпечення. Вони містять основні теоретичні положення та рекомендації, необхідні для засвоєння навчального матеріалу.

Укладачі: Мелешко Є.В., канд. техн. наук, доцент
Смірнов О.А., канд. техн. наук, професор

Рецензенти: Сидоренко В.В., доктор техн. наук, професор
Якименко Н.М., канд. фіз-мат. наук, доцент

ЗМІСТ

Лабораторна робота № 1

Тема: Шифри Цезаря та Відженера. Частотний криптоаналіз

Мета: набути навичок роботи з підстановочними та поліалфавітними шифрами, навчитися реалізовувати статистичний криптоаналіз підстановочних шифрів

Шифр Цезаря

Шифр Цезаря - симетричний алгоритм шифрування підстановками. Використовувався римським імператором Юлієм Цезарем для таємного листування.

Принцип дії даного шифру полягає в тому, що кожна літера x_i в відкритому тексті замінюється на відповідну літеру y_i, таку яка розміщена від літери x в алфавіті через k позицій.

Ключ шифрування k – це кількість літер, на які робиться зсув.

Якщо зіставити кожному символу алфавіту його порядковий номер (нумеруючи з 0), то шифрування і дешифрування можна виразити формулами:

Формула для шифрування:

$$y_i = (x_i + k) \bmod n,$$

формула для дешифрування:

$$x_i = (y_i - k + n) \bmod n,$$

де x_i – номер i-ї букви відкритого тексту, y_i – номер i-ої букви зашифрованого тексту, n – кількість літер в алфавіті, k – ключ шифрування (число, на яке буде зсунута кожна буква).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	І	Ї	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я

Рис. 1.1 – Нумерація алфавіту для шифрування методом Цезаря

Приклад 1.1. Завдання: Необхідно зашифрувати слово ЗНАННЯ, ключем шифрування k=3. Так як алфавіт український, то n=32.

Шифрування методом Цезаря:

	з	**н**	**а**	**н**	**н**	**я**
x_i	9	16	0	16	16	31
$+k$	+3	+3	+3	+3	+3	+3
	=12	=19	=3	=19	=19	=34
mod n	mod 32	mod 32	mod 32	mod 32	mod 32	mod 32
	=12	=19	=3	=19	=19	=2
y_i	**й**	**р**	**г**	**р**	**р**	**в**

Зашифроване слово: ЙРГРРВ

Дешифрування методом Цезаря:

	й	**р**	**г**	**р**	**р**	**в**
y_i	12	19	3	19	19	2
$-k$	–3	–3	–3	–3	–3	–3
	=9	=16	=0	=16	=16	= –1
$+n$	+32	+32	+32	+32	+32	+32
	=41	=48	=32	=48	=48	=31
mod n	mod 32	mod 32	mod 32	mod 32	mod 32	mod 32
	=9	=16	=0	=16	=16	=31
x_i	**з**	**н**	**а**	**н**	**н**	**я**

Результат дешифрування: ЗНАННЯ

Шифр Відженера

Шифр Віженера – поліалфавітний шифр, який у якості ключа використовує слово.

Формула для шифрування:

$$y_i = (x_i + k_j) \bmod n,$$

формула для дешифрування:

$$x_i = (y_i - k_j + n) \bmod,$$

де k_j – номер в алфавіті j-тої літери ключа шифрування.

Ключове слово повторюється поки не отримано гаму, рівну довжині повідомлення.

Приклад 1.2. Завдання: Необхідно зашифрувати слово ЗНАННЯ, ключем шифрування k=ЕОМ (тобто {6, 17, 15}). Так як алфавіт український, то n=32.

Шифрування методом Відженера:

	з	**н**	**а**	**н**	**н**	**я**
x_i	9	16	0	16	16	31
$+k_j$	+6	+17	+15	+6	+17	+15
	=15	=33	=15	=22	=33	=46
mod n	mod 32	mod 32	mod 32	mod 32	mod 32	mod 32
	=15	=1	=15	=22	=1	=14
y_i	**о**	**б**	**м**	**у**	**б**	**л**

Зашифроване слово: ОБМУБЛ

Дешифрування методом Відженера:

	й	**р**	**г**	**р**	**р**	**в**
y_i	15	1	15	22	1	14
$-k_j$	–6	–17	–15	–6	–17	–15
	=9	= –16	=0	=16	= –16	= –1
$+n$	+32	+32	+32	+32	+32	+32
	=41	=16	=32	=48	=16	=31
mod n	mod 32	mod 32	mod 32	mod 32	mod 32	mod 32
	=9	=16	=0	=16	=16	=31
x_i	**з**	**н**	**а**	**н**	**н**	**я**

Результат дешифрування: ЗНАННЯ

Частотний криптоаналіз

Для здійснення частотного криптоаналізу зашифрованого тексту спочатку треба визначити частоту поява літер (або, за необхідності, всіх символів) у відкритих текстах на потрібній мові. Для цього використаємо наступну формулу:

$$V = \frac{n_i}{N},$$

Де V – частота появи букви в тексті.

n_i – кількість *i*-ої букви в тексті.

N – кількість всіх букв у тексті.

В різних текстах та в різних ділянках одного тексту частота появи літер буде дещо відрізнятися. Так, наприклад, в пригодницькому романі буква «ф» буде зустрічатися значно рідше, ніж в підручнику з математики, де часто будуть вживатися слова **ф**ормула, гра**ф**ік, **ф**ункція тощо.

Для отримання гарних статистичних даних, що добре будуть ілюструвати частоту появи літер для заданої мови взагалі, а не для окремих випадків, для побудови таблиці частоти появи літер треба взяти один або декілька довгих текстів на загальну тематику.

Оскільки невідомо якої довжини повинен бути текст, щоб одержати за допомогою нього гарні статистичні дані, то можна або просто взяти книгу розміром, напр., не менше 50 Мб, або ж здійснювати підрахунок все більших та більших об'ємів текстових даних поки частоти появи літер при збільшенні кількості тексту не перестануть змінюватися.

Згідно цього другого варіанту слід розділити досліджуваний текст на фрагменти, в кожному з яких підраховують n_{ij} — кількість літери *i* у фрагменті *j*. Накопичена частота V обчислюється за такою формулою:

$$V_n = V_{n-1} + \frac{n_{ij}}{N_j},$$

де N_j – розмір фрагменту тексту (загальна кількість літер у ньому).

Отримаємо послідовність $\{V_1, V_2, V_3, \dots, V_n\}$, яку дослідимо на предмет наявності властивості стабільності. Ця властивість полягає в тому, що починаючи з номера m для всіх n>m буде виконуватися:

$$|V_n - V_{n-1}| < \varepsilon,$$

де ε – нескінченно мала величина. В будемо брати значення ε = 0,0001.

Чим менше значення ε ми візьмемо, тим більш точні значення частот появи літер отримаємо і тим більше ці частоти будуть наближатися до ймовірності появи літер в досліджуваній мові.

Формула $|V_n - V_{n-1}| < \varepsilon$ є умовою завершення обробки фрагментів тексту.

Наведемо приклад таблиці ймовірностей появи літер в російській мові в таблиці 1.1.

Таблиця 1.1 – Безумовні ймовірності появи літер в російській мові

Літера	Ймовірність	Літера	Ймовірність	Літера	Ймовірність
Пробіл	0,175	к	0.028	ч	0,012
о	0.090	м	0,026	й	0.010
е	0.072	д	0.025	х	0.009
а	0.062	п	0.023	ж	0.007
и	0.062	у	0.021	ю	0.006
т	0.053	я	0.018	ш	0.006
н	0.053	ы	0.016	ц	0.004
с	0.045	з	0.016	щ	0.003
р	0.040	ь,ъ	0.014	э	0.003
в	0.038	б	0.014	ф	0.002
л	0.035	г	0.013		

Після того як таблиця частот появи літер побудована для взлому шифру Цезаря необхідно побудувати таку ж таблицю для зашифрованого тексту та співставити частоти появ літер, напр., якщо в зашифрованому тексті літера О зустрічається з ймовірністю літери У, то це і є літера У, для визначення ключа шифрування слід відняти від номера літери У номер літери О: k = (22–17) mod 32 = 5. Для виключення помилки (так як є літери з однаковими частотами) слід співставити декілька частот літер, або для повної надійності – всі.

Частотний криптоаналіз шифру Відженера виконується так:

1. Знаходиться довжина ключа. Шифротекст розбивається на групи по номеру літери ключа шифрування.

2. За допомогою частотного аналізу груп знаходяться літери ключа.

Завдання:

1) Реалізувати кодер та декодер на основі шифру Цезаря. В якості ключа шифрування взяти другу літеру свого прізвища. Реалізувати частотний криптоаналіз даного шифру.

2) Реалізувати кодер та декодер на основі шифру Відженера. В якості ключа шифрування взяти своє прізвище.

Контрольні питання:

1) Як реалізується шифр Цезаря?

2) Як реалізується шифр Відженера?

3) Що таке частотний криптоаналіз?

4) Як будуються таблиця частот появи літер?

5) Чим відрізняється алгоритм криптоаналізу шифру Відженера від шифру Цезаря?

Лабораторна робота № 2

Тема: Симетричні алгоритми шифрування. DES

Мета: набути навичок роботи з симетричними алгоритмами шифрування

Розглянемо загальну схему симетричної, або традиційної, криптографії.

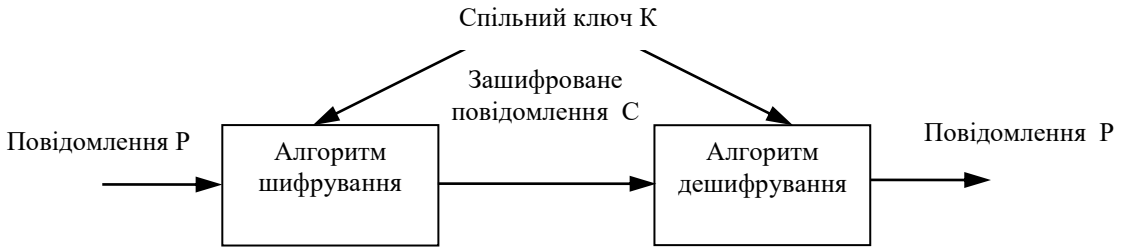

Рис. 2.1. Загальна схема симетричного шифрування

В симетричному шифруванні використовується один секретний ключ як для шифрування так і для дешифрування (або по одному ключі можна легко обчислити інший).

До симетричного шифрування відносяться такі алгоритми як DES, Triple-DES, Blowfish, IDEA, RC4 тощо. Багато розповсюджених шифрів (наприклад IDEA, DES, BLOWFISH) є блоковими шифрами. Це означає, що вони беруть блоки даних фіксованого розміру (звичайно 64 біта), і перетворять їх в інший 64-бітний блок за допомогою функції, що задається ключем.

Найпоширенішим і найбільш відомим алгоритмом симетричного шифрування є **DES** (Data Encryption Standard). Алгоритм був розроблений в 1977 році, в 1980 році був прийнятий NIST (National Institute of Standards and Technolody США) як стандарт (FIPS PUB 46).

Реалізації DES можуть бути знайдені, наприклад, у бібліотеках libdes, alodes, SSLeay, Crypto++, descore, chalmers-des і в destoo.

Існують розробки, що дозволяють виконувати шифрування в рамках стандарту DES апаратним способом, що забезпечує досить високу швидкодію.

Алгоритм шифрування DES

Дані шифруються 64-бітовими блоками, використовуючи 56-бітовий ключ. Ключ шифрування повинен бути відомий як відправнику так і одержувачу.

Процес шифрування складається із таких етапів:

1) початкова перестановка 64-бітного вихідного тексту, під час якої біти переставляються у відповідності зі стандартною таблицею.
2) 16 раундів однієї й тієї ж функції, що використовує операції зсуву і підстановки.

3) ліва і права половини виходу останньої (16-ї) ітерації міняються місцями.
4) перестановка результату, отриманого на третьому етапі, ця перестановка інверсна початковій перестановці.

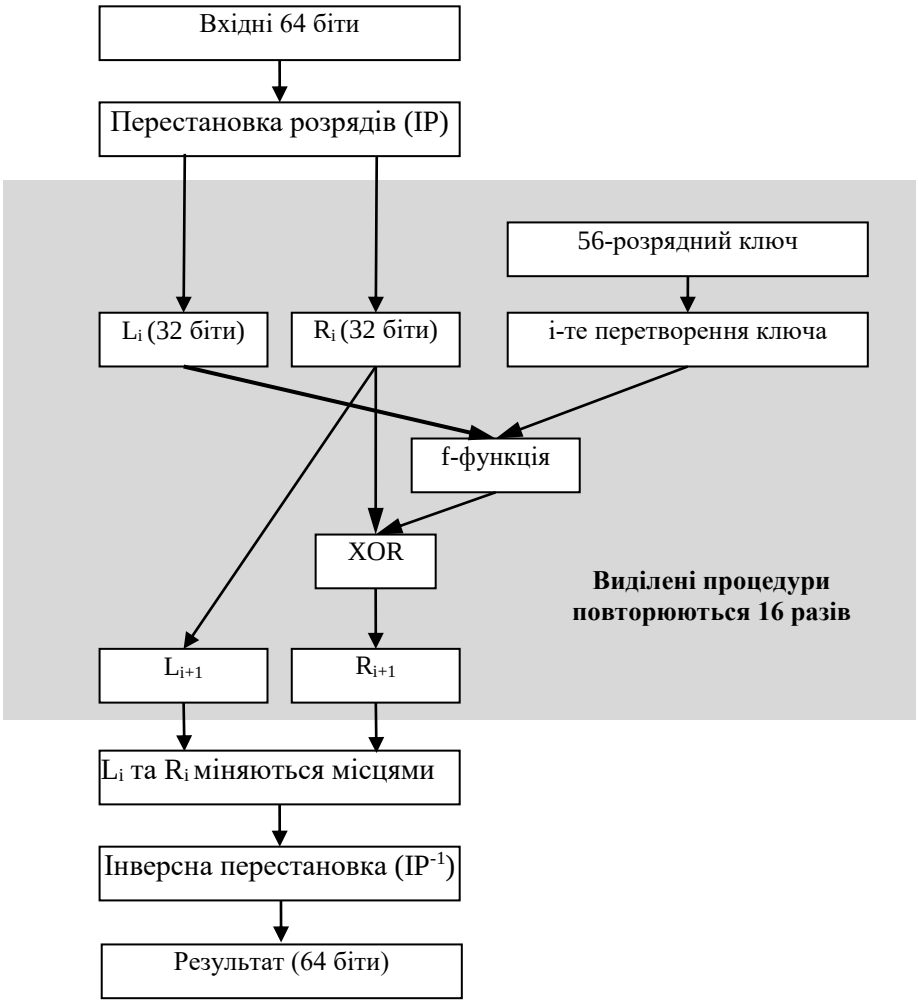

Рис. 2.2. Структурна схема реалізації алгоритму DES

Тепер більш детально розглянемо наведені вище етапи алгоритму.

1 Етап

Шифрування починається з перестановки бітів (**IP - Initial Permutation)** в 64-розрядному блоці вихідних даних. 58-ой біт стає першим, 50-ий - другим і т.д.

Схема початкової перестановки бітів (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
```

2 Етап

Отриманий блок ділиться на дві 32-розрядні частини L_0 і R_0. Далі 16 разів повторюються наступні 4 процедури:

1) Перетворення ключа з урахуванням номера ітерації і (перестановки розрядів з видаленням 8 біт, у результаті виходить 48-розрядний ключ)

2) За допомогою функції $f(L_i,K_i)$ генерується 32-розрядний вихідний код.
3) Виконується операція XOR: $R_i \otimes f(L_i,K_i)$, результат позначається R_{i+1}.
4) Виконується операція присвоєння $L_{i+1}=R_i$.

Перетворення ключів K_n (n=1,…,16; K_n=KS(n, key), де n - номер ітерації) здійснюються відповідно до алгоритму, показаному на рис. 3.

Для описання алгоритму обчислення ключів K_n (функція KS) досить визначити структури “Вибір 1” та “Вибір 2”, а також задати схему зсувів вліво (табл. 3). “Вибір 1” та “Вибір 2” являють собою перестановки бітів ключа (PC-1 і PC-2; табл. 1, 2). При необхідності біти 8, 16,..., 64 можуть використовуватися для контролю парності.

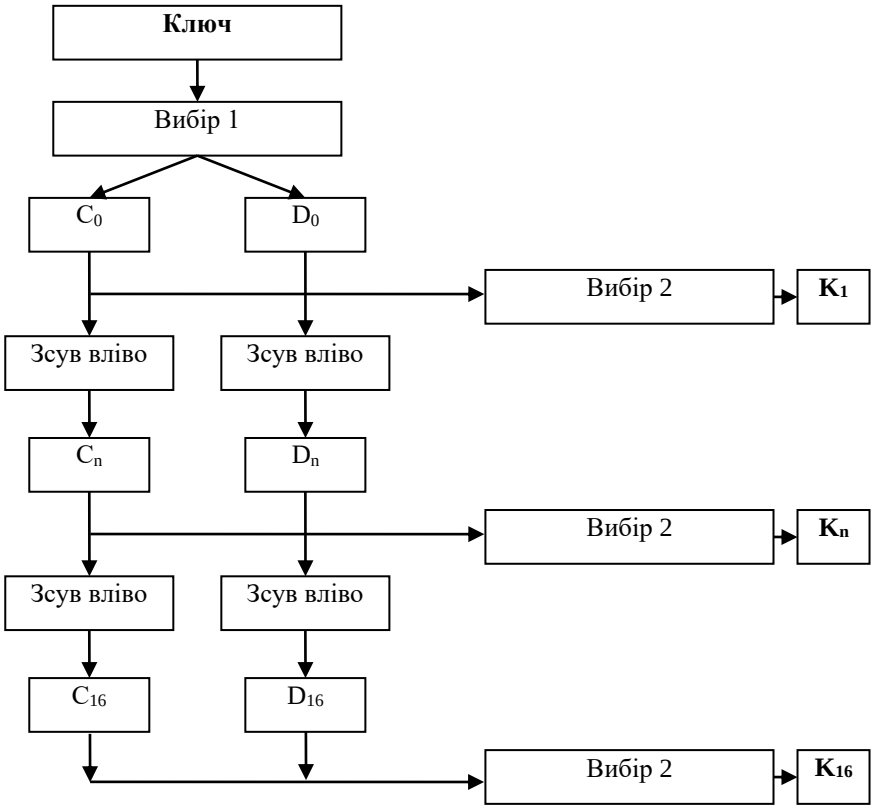

Рис. 2.3. Алгоритм обчислення послідовності ключів K_n

Таблиця 1. PC-1 (Вибір 1)

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

Таблиця 2. PC-2 (Вибір 2)

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

Вибір бітів по таблицях з відповідних блоків здійснюється в такий спосіб. Таблиця розглядається як послідовність її рядків, записаних один за одним, починаючи з першого рядка. Біти блоку даних відповідної довжини нумеруються зліва на право, починаючи з одиниці. Кожний елемент **m** таблиці розглядається, як номер біта $\mathbf{b_m}$ у блоці даних. Перетворення полягає в заміні всіх елементів **m** на біти $\mathbf{b_s}$.

У C_0 увійдуть біти 57, 49, 41,..., 44 і 36, а в D0 - 63, 55, 47,..., 12 і 4. Тому що схема зсувів задана (табл. 2) C_1,D_1; C_n, D_n і так далі можуть бути легко отримані з C_0 і D_0. Так, наприклад, C_3 і D_3 виходять із C_2 і D_2 циклічним зсувом вліво на 2 розряди.

Таблиця 3 Схема зсувів ключа

Номер циклу	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Зсув вліво (шифрування)	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**1**
Зсув вправо (розшифрування)	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**1**

Схема роботи функції f показана на рис. 4. Спочатку 32 вхідні розряди розширюються за допомогою перетворення EP до 48, при цьому деякі розряди повторюються.

Функція розширення EP**:**

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

Для отриманого 48-розрядного коду й ключа виконується операція XOR. Результуючий 48-розрядний код перетвориться в 32-розрядний за допомогою S-Матриць. Вихідний 48-розрядний код ділиться на 8 груп по 6 розрядів. Перший і останній розряд у групі використовується як адреса рядка, а середні 4 розряди - як адреса стовпця. У результаті кожні 6 біт коду перетворяться в 4 біти, а весь 48-розрядний код в 32-розрядний, для цього потрібно 8 S-Матриць (S-*boxes*).На виході S-Матриць здійснюється перестановка за схемою P-блока.

P-блок:

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

S-Матриці являють собою таблиці, що містять 4-рядки і 16 стовпців.

S1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	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
S2	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
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
S3	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
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
S4	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
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
S5	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
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
S6	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
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	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	4	11	2	14	15	0	8	13	3	32	9	7	5	10	6	1
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
S8	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
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

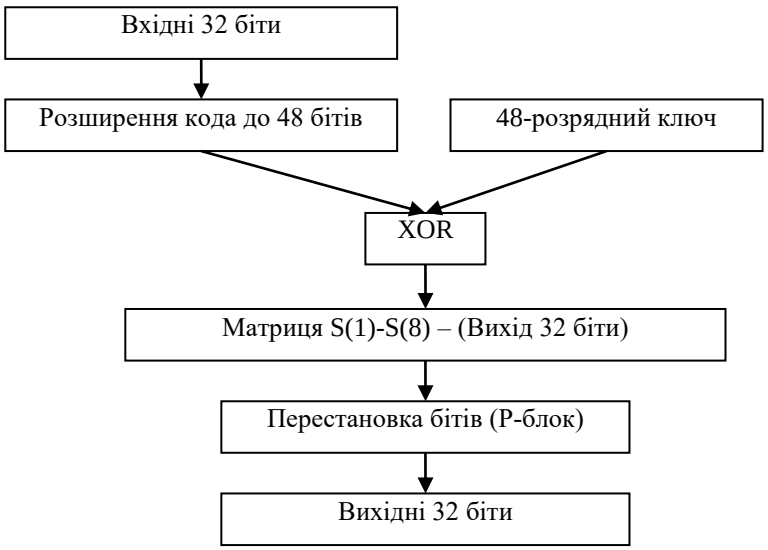

Рис. 2.4. Алгоритм роботи функції f

3 Етап

L_{16} та R_{16} міняються місцями.

4 Етап

Інверсна перестановка розрядів здійснюється наступним розміщенням 64 бітів зашифрованих даних (першим бітом стає 40-вий, другий 8-мим і т.д.).

Схема інверсної перестановки бітів (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
```

Алгоритм дешифрування DES

DES використовує для шифрування блока одну і ту ж функцію, різниця полягає лише в тому, що к**лючі повинні використовуватися в зворотному порядку** (якщо для шифрування використовувалися ключі K_1, K_2, K_3, K_4, K_5,…, K_{16}, то ключами дешифровки будуть K_{16}, K_{15}, K_{14}, K_{13}, K_{12},…, K_1). Це досягається тим, що зсув здійснюється не вліво, а вправо (дивись табл. 3).

Проблеми DES

Так як довжина ключа дорівнює 56 бітів, існує 2^{56} можливих ключів. На сьогодні така довжина ключа недостатня, оскільки допускає успішне застосування лобових атак. Альтернативою DES можна вважати *потрійний DES*, IDEA, а також алгоритм Rijndael, прийнятий як новий стандарт на алгоритми симетричного шифрування.

Також без відповіді поки залишається питання, чи можливий криптоаналіз із використанням існуючих характеристик алгоритму *DES*. Основою алгоритму є вісім таблиць підстановки, або S-*boxes*, які застосовуються в кожній ітерації. Існує небезпека, що ці S-*boxes* конструювалися таким чином, що криптоаналіз можливий для зломщика, що знає слабкі місця S-*boxes*. Протягом багатьох років обговорювалося як стандартне, так і несподіване поводження S-*boxes*, але все-таки нікому не вдалося виявити їх фатально слабкі місця.

Завдання: Створити програму, що реалізує алгоритм DES. Зашифрувати та розшифрувати файл об’ємом не менший 50 кб.

Контрольні питання:

1) Що таке симетричні алгоритми шифрування?
2) В чому полягає принцип роботи алгоритму DES?
3) Що таке S-блоки?
4) Яка довжина ключа в алгоритмі DES?
5) Як здійснюється генерація підключів шифрування в алгоритму DES?

Лабораторна робота №3

Тема: Асиметричні алгоритми шифрування. RSA

Мета: набути навичок роботи з асиметричними алгоритмами шифрування та протоколами обміну відкритими ключами

Концепція криптосистеми з відкритим ключем

Ефективними системами криптографічного захисту даних є асиметричні криптосистеми (криптосистемами з відкритим ключем). У таких системах для шифрування даних використовується один ключ, а для розшифрування - інший (звідси й назва - асиметричні). Перший ключ є *відкритим* і може бути опублікований для використання всіма користувачами системи, які зашифровують дані. Розшифрування даних за допомогою відкритого ключа неможливо. Для розшифрування даних одержувач зашифрованої інформації використовує другий ключ, що є *закритим (секретним)*. Зрозуміло, ключ розшифрування не може бути визначений із ключа шифрування. Узагальнена схема асиметричної криптосистеми з відкритим ключем показана на рис.1. Генератор ключів доцільно розташовувати на стороні одержувача В (щоб не пересилати секретний ключ $К_В$ по незахищеному каналу).

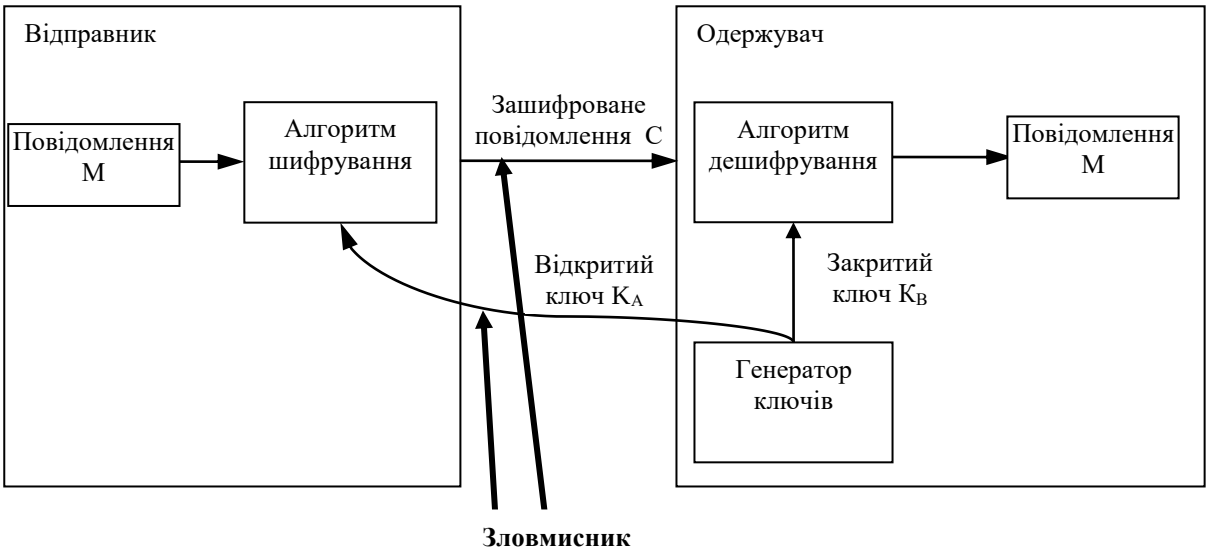

Рис. 3.1. Узагальнена схема асиметричної криптосистеми з відкритим ключем

Криптосистема шифрування даних RSA

Алгоритм RSA запропонували в 1978 р. три автори: Р.Райвест (Rivest), А.Шамир (Shamir) і А.Адлеман (Adleman). Алгоритм одержав свою назву по

перших літерах їх прізвищ. Алгоритм RSA став першим повноцінним алгоритмом з відкритим ключем, що може працювати як у режимі шифрування даних, так і в режимі електронного цифрового підпису.

Надійність алгоритму ґрунтується на важкості факторизації великих чисел та важкості обчислення дискретних логарифмів.

Алгоритм RSA складається з наступних пунктів:

1. Вибрати прості числа **p** і **q**.
2. Обчислити **n = p * q**.
3. Обчислити **m = (p - 1) * (q - 1)**.
4. Вибрати число **d** взаємно просте з **m**.
5. Вибрати число **e** таке, щоб **e * d = 1 (mod m)**.

Число **d** – закритий ключ, **e** –відкритий.

Повідомлення перед шифруванням необхідно розбити на блоки - числа від 0 до n - 1. Шифрування та дешифрування даних здійснюються в такий спосіб:

- Шифрування: $b = a^e \pmod n$
- Дешифрування: $a = b^d \pmod n$

Слід також зазначити, що ключі e і d рівноправні, тобто повідомлення можна шифрувати як ключем e, так і ключем d, при цьому розшифровка повинна бути зроблена за допомогою іншого ключа.

Знаходження взаємно простих чисел

На кроці 4 алгоритми RSA необхідно знайти число d взаємно просте з m. Число d повинне бути менше m, таким чином розрядність числа d дорівнює сумі бітів у числах p і q. Для знаходження взаємно простих чисел використовується алгоритм Евкліда, що знаходить найбільший спільний дільник двох чисел. Якщо знайдений дільник більше одиниці, то необхідно вибрати інше число d і повторити перевірку.

Алгоритм Евкліда

1. Беруться числа a і b
2. Обчислюється r - залишок від ділення a на b: a = b * q + r
3. Якщо r = 0, то b - шукане число (найбільший спільний дільник), кінець
4. Замінити пари чисел <a, b> парою <b, r>, перейти до пункту 2

При обчисленні найбільшого спільного дільника за допомогою алгоритму Евкліда буде виконано не більше 5 * p операцій ділення із залишком, де p є кількість цифр у десятковому записі меншого із чисел a і b. На практиці алгоритм працює дуже швидко.

В 5-му пункті алгоритму RSA передбачається знаходження такого числа e, щоб e * d = 1 (mod m). Обчислення числа e зводиться до рішення рівняння m * x + d * e = 1 в натуральних числах. Число x не важливе. Для цього потрібно

використати модифікований алгоритм Евкліда, який працює тільки якщо числа d та m взаємно прості.

Завдання: Реалізувати алгоритм RSA. Зашифрувати та розшифрувати файл об’ємом не менший 50 кб. Порівняти зі швидкістю виконання алгоритму DES (лаболаторна робота №2).

Контрольні питання:

1) Що таке асиметричні алгоритми шифрування?

2) В чому полягає принцип роботи алгоритму RSA?

3) В чому полягає суть алгоритма Евкліда? Для чого він використовується в алгоритмі RSA?

4) Яка довжина ключів в алгоритмі RSA?

5) Поясніть схему розподілу ключів шифрування. Для чого потрібні закриті та відкриті ключі?

Лабораторна робота № 4

Тема: Генератори псевдовипадкових чисел

Мета: навчитися реалізовувати генератори псевдовипадкових чисел та перевіряти числа на простоту для створення ключів шифрування

Криптографічні системи мають потребу в надійному генераторі випадкових чисел, значення якого не зможе передбачити атакуючий. Випадкові числа звичайно використовуються для генерації ключів сесії і їхня якість критичним образом впливає на якість системи в цілому. Генератор випадкових чисел легко випустити із уваги, і тоді він стає найслабшою ланкою системи.

Деякі машини можуть мати спеціальні апаратні генератори шуму. Шум від струму витоку діода або транзистора, молодші біти аудіосигнала, інтервали між перериваннями тощо. Усе це хороші джерела випадкових даних якщо їх пропустити через підходящу хеш-функцію. Гарною ідеєю є одержання в міру можливості справжнього шуму з оточення.

Менш надійними, але простішими в реалізації є генератори псевдовипадкових чисел на основі математичних функцій.

Найпростіша послідовність, яку можна запропонувати для реалізації генератора рівномірного розподілу:

$$I_{j+1}=(a*I_j) \bmod m$$

При відповідному виборі констант. Константи запропоновані Park та Miller:

$$a=7^5=16807,\ m=2^{31}-1=2147483647$$

і протестовані в дослідженнях Lewis, Goodman, Miller (1969).

Пряма реалізація цього методу можлива на мовах асемблера, але мови високого рівня можуть при цьому зафіксувати переповнення. Для обходу цього Scharge запропонував метод часткової факторизації модуля. Модуль розкладається таким чином:

$$m=a*q+r$$

Якщо $r<q$ і $0<z< m-1$, то при цьому величини $a*(z \bmod q)$ і $r*[z/q]$ завжди лежать в інтервалі 0,..., m-1. Для множення $(a*z)(\bmod m)$ при цьому використовується алгоритм:

$t = a(z \bmod q)-r[z/q]$

якщо $t<0$, то $t += m$.

$(a*z)(\bmod m)=t$.

У випадку констант Парка-Міллера можна використати q=12773 і r=2836.

Для асиметричних систем необхідно генерувати прості випадкові числа. Простий спосіб перевірки на простоту - ділення числа на всі числа менші за

нього не працездатний уже з 32-бітними числами (потрібно дуже багато часу на виконання).

У цьому випадку, для вибору простих чисел використовують інші методи, один з яких представлений нижче.

Для перевірки числа на простоту розглянемо алгоритм Рабіна-Міллера, що робить наступне:

1) генерується випадкове число p, та встановлюється в одиницю його молодший біт;
2) обчислюється b - число дільників (p - 1) на 2 (тобкто 2^b – це найбільша степінь числа 2, на яку ділиться(p - 1));
3) обчислюється m, таке що $p = 1 + 2^b * m$;
4) вибирається випадкове число a, менше ніж p;
5) встановлюється j = 0 та z =am mod p;
6) якщо z = 1 або якщо z = p - 1, то p **проходить перевірку** і може бути простим числом;
7) якщо j > 0 і z = 1, то p не є простим числом;
8) встановлюється j = j + 1. Якщо j < b та $z \neq p - 1$, встановити $z = z^2$ та повернутися на етап (7). Якщо z = p – 1 то p проходить перевірку і може бути простим числом;
9) якщо j = b та $z \neq p - 1$, то p не є простим числом.

Для прискорення швидкодії даного алгоритму, доцільно перед його використанням робити перевірку на те, чи не ділиться випадкове число p на 3, 5 та 7, це одразу дозволяє відсікати 54% непростих чисел, ще до етапу (7).

Завдання: Створити програму, що генерує псевдовипадкові числа та перевіряє їх на простоту.

Контрольні питання:

1) Чим відрізняються випадкові та псевдовипадкові числа? Для чого вони потрібні в криптографічних алгоритмах?
2) Як можна здійснити генерацію псевдовипадкових чисел?
3) Що здійснює алгоритм Рабіна-Міллера? Для чого це необхідно?
4) Які існують джерела отримання випадкових чисел?
5) Які числа можна використовувати як ключі в асиметричних алгоритмах шифрування?

Лабораторна робота № 5

Тема: Хеш-функції. MD5 та SHA-1

Мета: навчитися реалізовувати криптографічні хеш-функції

Хеш-функція – це така функція, що перетворює вхідний масив даних довільної довжини у вихідний бітовий рядок фіксованої довжини так, щоб зміна вхідних даних приводила до непередбачуваної зміни вихідних даних. Такі перетворення також називаються хешуванням або функціями згортки, а їх результати називають хешем, хеш-кодом або дайджестом повідомлення.

В криптографії хеш-функції використовуються у створенні електронного цифрового підпису та для аутентифікації користувачів (коли в базі банних замість паролів містяться їх хеш-значення).

Хеш-функції також використовуються в деяких структурах даних - хеш таблицях і декартових деревах.

Криптографічна хеш-функція повинна забезпечувати:

- стійкість до колізій: два різні набори даних повинні мати різні результати перетворення, тобто для заданого повідомлення M повинно бути практично неможливо підібрати інше повідомлення M', що має такий же хеш.
- безповоротність (неможливість обчислити початкові дані по результату перетворення)

Згідно парадоксу про «дні народження», знаходження колізії для хеш-функції з довжиною хеш-коду n біт вимагає O(sqrt(n)) операцій. Таким чином, цим числом оцінюється стійкість хеш-функції до колізій.

Більшість хеш-функцій будується на основі односпрямованої функції `f()`, яка створює вихідне значення довжиною `n` при наявнсті двох вхідних довжиною `n`. Ціми вхідними значеннями являються блок текста `M`, та хеш-значення H_{i-1} попереднього блока текста (рис.1).

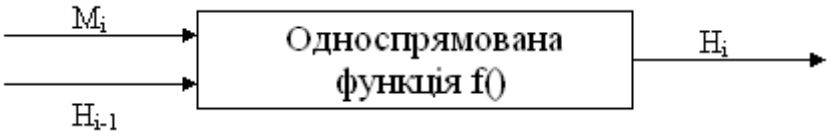

Рис. 5.1. Побудова односпрямованої хеш-функції

$H_i = f(M_i, H_{i-1})$.

Хеш-значення, що обчислюється при введенні останнього блоку тексту, стає хеш-значенням усього повідомлення `M`.

У результаті односпрямована хеш-функція завжди формує вихід фіксованої довжини `n` (незалежно від довжини вхідного тексту).

MD5 *(Message Digest 5)* - 128-бітний алгоритм хешування, розроблений професором Рональдом Л. Рівестом у 1991 році.

Алгоритм MD5

Константи, що використовуються в даному алгоритмі:

1. Початкові значення для ініціалізації змінних:

H[0]=67452301_{16}

H[1]=$EFCDAB89_{16}$

H[2]=$98BADCFE_{16}$

H[3]=10325476_{16};

2. Константи додавання в раундах:

y[j]=HIGHEST_32_BITS(ABS(SIN(j+1))) j=0...63,

де функція HIGHEST_32_BITS(X) визначає 32 самих старших біта з двійкового запису дробового числа X, а операнд SIN(j+1) вважається взятим в радіанах.

3. масив порядку вибору комірок у раундах:

z[0...63] = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12,
5, 8, 11, 4, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2,
0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9);

4. масив величини бітових циклічних зсувів вліво:

s[0...63] = (7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21).

Етапи виконання алгоритму:

1) Вхідні дані вирівнюються так, щоб їх розмір був рівний 448 по модулю 512. Спочатку дописується одиничний біт (навіть якщо довжина вже дорівнює потрібній величині), потім необхідне число нульових бітів.
2) Дописуються 64-біти, що містять довжину даних до вирівнювання. Якщо довжина перевершує $2^{64} - 1$, то дописуються молодші біти.
3) Ініціалізуються чотири 32-бітні змінні:
   `A = H[0]; B = H[1]; C = H[2]; D = H[3].`

4) Кожний 512-бітний блок, представлений у вигляді 16-ти 32-розрядних значень X[0]...X[15], проходить через стискаючу функцію, яка перемішує його з допоміжним блоком (H[0],H[1],H[2],H[3]):

цикл по j **від** 0 **до** 15
T = (A + f(B,C,D) + x[z[j]] + y[j]) ROL s[j]
(A,B,C,D) = (D,B+T,B,C)
кінец_циклу
цикл по j **від** 16 **до** 31

T = (A + g(B,C,D) + x[z[j]] + y[j]) ROL s[j]
(A,B,C,D) = (D,B+T,B,C)
кінец_циклу
цикл по j **від** 32 **до** 47
T = (A + h(B,C,D) + x[z[j]] + y[j]) ROL s[j]
(A,B,C,D) = (D,B+T,B,C)
кінец_циклу
цикл по j **від** 48 **до** 63
T = (A + k(B,C,D) + x[z[j]] + y[j]) ROL s[j]
(A,B,C,D) = (D,B+T,B,C)
кінец_циклу
(H[0],H[1],H[2],H[3]) = (H[0]+A,H[1]+B,H[2]+C,H[3]+D)

Функції, які використовуються в раундах:

- У першому раунді f(U,V,W)=(U **and** V) **or** ((**not** U) **and** W).
- У другому раунді g(U,V,W)=(U **and** W) **or** (V **and** (not W)).
- У третьому раунді h(U,V,W)=U **xor** V **xor** W.
- У четвертому раунді k(U,V,W)=V **xor** (U **or** (**not** W)).

5) Підсумковий хеш = ABCD (128 бітів)

Алгоритм SHA-1

Алгоритм SHA-1 (Secure Hash Algorithm) - криптографічна хеш-функція призначена для обчислення з повідомлення довжиною до 2^{64} бітів значення довжиною 160 бітів. **SHA-1** розроблений у США в рамках стандарту безпечного хешування SHS (Secure Hash Standard) в 1992 р. Алгоритм SHA призначений для використання разом з алгоритмом цифрового підпису DSA.

При введенні повідомлення M довільної довжини меншої ніж 2^{64} бітів алгоритм SHA-1 створює 160-бітове вихідне хеш-значення. Потім цей дайджест повідомлення використовується як вхід алгоритму DSA, що обчислює цифровий підпис повідомлення M. Формування цифрового підпису для дайджесту повідомлення, а не для самого повідомлення підвищує ефективність процесу підписання, оскільки дайджест повідомлення звичайно набагато коротше самого повідомлення. Такий же дайджест повідомлення повинен обчислюватися користувачем, що перевіряє отриманий підпис, при цьому як вхід в алгоритм використовується отримане повідомлення M.

Розглянемо докладніше роботу алгоритму SHA-1. Насамперед вихідне повідомлення M вирівнюють так як і при md5, щоб воно стало кратним 512 бітам, спочатку додаються одиниця, потім стільки нулів, скільки необхідно для

одержання повідомлення, що на 64 біта коротше, ніж кратне 512, і нарешті додають 64-бітове подання довжини вихідного повідомлення.

Ініціалізуються п'ять 32-бітових змінних у вигляді:

```
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
E = 0xC3D2E1F0
```

Потім починається головний цикл алгоритму. У ньому обробляється по 512 бітів повідомлення по черзі для всіх 512-бітових блоків, наявних у повідомленні. Перші п'ять змінних `A, B, C, D, E` копіюються в інші змінні `a, b, c, d, e`:

```
a = A, b = B, c = C, d = D, e = E
```

Головний цикл містить чотири цикли по 20 операцій кожний. Кожна операція реалізує нелінійну функцію від трьох з п'яти змінних `a, b, c, d, e`, а потім робить зсув і додавання.

Алгоритм SHA має наступний набір нелінійних функцій:

$f_t(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ для t = 0...19,

$f_t(X, Y, Z) = X \oplus Y \oplus Z$ для t = 20...39,

$f_t(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$ для t = 40...59,

$f_t(X, Y, Z) = X \oplus Y \oplus Z$ для t = 60...79,

В алгоритмі використовуються також чотири константи:

K_t = 0x5A827999 для t = 0...19,
K_t = 0x6ED9EBA1 для t = 20...39,
K_t = 0x8F1BBCDC для t = 40...59,
K_t = 0xCA62C1D6 для t = 60...79.

Блок повідомлення перетвориться із шістнадцяти 32-бітових слів ($M_0 ... M_{15}$) у вісімдесят 32-бітових слів ($W_0 ... W_{79}$) за допомогою наступного алгоритму:

$$W_t = M_t \text{ для } t = 0...15,$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1 \text{ для } t = 16...79,$$

де t - номер операції, W_t - t-й субблок розширеного повідомлення, <<< S - циклічне зрушення вліво на S біт.

З урахуванням уведених позначень головний цикл із вісімдесятьох операцій можна описати так:

```
цикл по t від 0 до 79
TEMP = (a <<< 5) + ft (b, c, d) + e + Wt + Kt
e = d
```

```
 d = c
 c = (b <<< 30)
 b = a
 a = TEMP
кінець_циклу
```

Схема виконання однієї операції показана на рис.2.

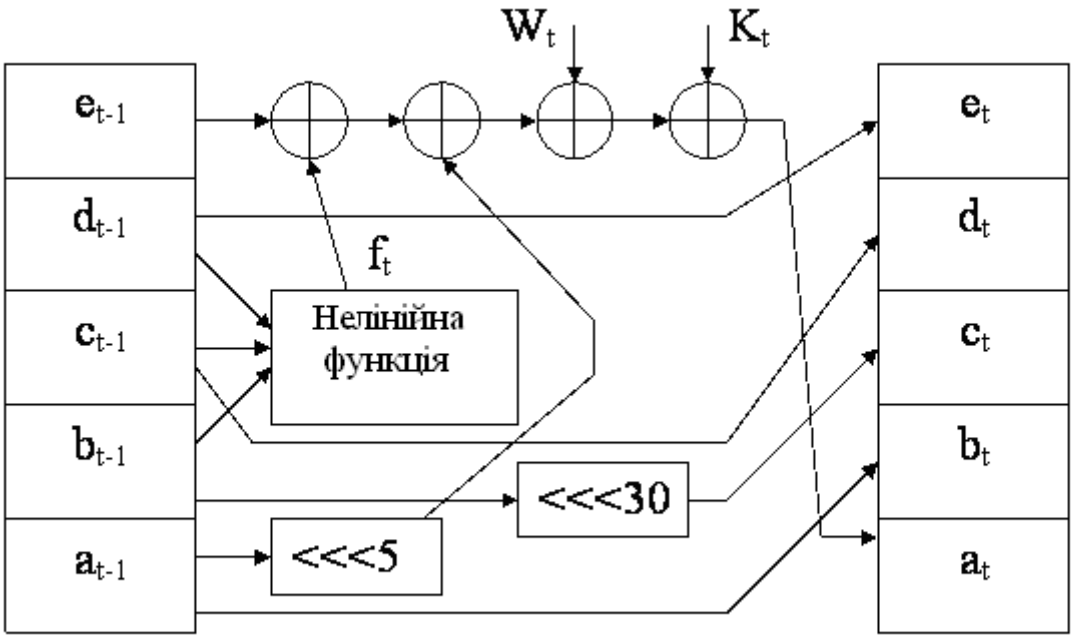

Рис. 5.2. Схема виконання однієї операції алгоритму SHA-1

Після закінчення головного циклу значення `a, b, c, d, e` складаються з `A, B, C, D, E` відповідно, і алгоритм приступає до обробки наступного 512-бітового блоку даних. Остаточний вихід формується у вигляді конкатенації значень `A, B, C, D, E.`

Відмінності SHA-1 від MD5 полягають у наступному:

– SHA-1 видає 160-бітове хеш-значення, тому він більш стійкий до атак повного перебору й атак "дня народження", ніж MD5, що формує 128-бітові хеш-значения.

– Стискаюча функція SHA-1 складається з 80 кроків, а не з 64 як в MD5.

– Розширення вхідних даних здійснюється не простим їхнім повторенням в іншому порядку, а рекуррентною формулою.

– Ускладнено процес перемішування.

Завдання: Розробити програму, яка реалізує алгоритми хешування MD5 та SHA-1. Порівняти швидкість виконання.

Контрольні питання:

1) Що таке хеш-функція?
2) Для чого хеш-функції застосовуються в криптографії?
3) В чому полягає принцип роботи алгоритму SHA-1?
4) В чому полягає принцип роботи алгоритму MD5?
5) Який алгоритм більш надійний SHA-1 чи MD5?

Лабораторна робота №6

Тема: Аутентифікація даних і цифровий підпис

Мета: набути навичок роботи з електронним цифровим підписом

Метою аутентифікації електронних документів є їхній захист від можливих видів злочинних дій, до яких відносяться:

1. *активне перехоплення* - порушник, що підключився до мережі, перехоплює документи (файли) і змінює їх;
2. *маскарад* - абонент С посилає документ абоненту В від імені абонента А;
3. *ренегатство* - абонент А заявляє, що не посилав повідомлення абоненту В, хоча насправді послав;
4. *підміна* - абонент В змінює або формує новий документ і заявляє, що одержав його від абонента А;
5. *повтор* - абонент С повторює раніше переданий документ, що абонент А надсилав абоненту В.

Для аутентифікації текстів використовується електронний цифровий підпис (ЕЦП). Функціонально він аналогічний звичайному рукописному підпису й володіє його основними достоїнствами:

1. засвідчує, що підписаний текст надходить від імені, того хто поставив підпис;
2. не дає цій особі можливості відмовитися від зобов'язань, пов'язаних з підписаним текстом;
3. гарантує цілісність підписаного тексту.

Загальний алгоритм створення ЕЦП.

При формуванні ЕЦП відправник насамперед обчислює хеш-функцію h(M) тексту М. Обчислене значення хеш-функції h(M) являє собою один короткий блок інформації m, що характеризує весь текст М у цілому. Потім число m шифрується секретним (закритим) ключем відправника. Одержане при цьому значення і є цифровим підписом тексту М.

При перевірці ЕЦП одержувач повідомлення знову обчислює хеш-функцію m = h(M) прийнятого по мережі тексту М, після чого за допомогою відкритого ключа відправника перевіряє, чи відповідає отриманий підпис обчисленому значенню m хеш-функції.

Принциповим моментом у системі ЕЦП є неможливість підробки підпису без знання секретного ключа.

На відміну від шифрування під час підписання документа відкритий і закритий ключі міняються місцями. Підписання здійснюється закритим ключем, а перевірка підпису - відкритим.

Цифровий підпис на основі RSA

Першою й найбільш відомою в усьому світі конкретною системою ЕЦП стала система на основі RSA.

Алгоритм підписання документа.

Спочатку необхідно обчислити пару ключів (секретний ключ і відкритий ключ). Для цього відправник (автор) електронних документів обчислює два великих простих числа P и Q, потім знаходить їх добуток N=P*Q і значення функції φ(N)=(P-1)(Q-1).

Далі відправник обчислює число E з умов:

$E \le \varphi(N)$, НСД $(E, \varphi(N)) = 1$

і число D з умов:

$D < N$, $E * D \equiv 1 \pmod{\varphi(N)}$.

Пари чисел (E, N) є відкритим ключем. Цю пару чисел автор передає партнерам по переписці для перевірки його цифрових підписів. Число D зберігається автором як секретний ключ для підписання документів.

Допустимо, що відправник хоче підписати повідомлення M перед його відправленням. Спочатку повідомлення M (блок інформації, файл, таблиця) стискають за допомогою хеш-функції h() у ціле число m:

$m = h(M)$.

Потім обчислюють цифровий підпис S під електронним документом M, використовуючи хеш-значення m і секретний ключ D:

$S = m \pmod{N}$.

Пара (M, S) передається партнерові-одержувачеві як електронний документ M, підписаний цифровим підписом S, причому підпис S сформований власником секретного ключа D.

Алгоритм перевірки підпису.

Після прийому пари (M, S) одержувач обчислює хеш-значення тексту M двома різними способами. Насамперед він відновлює хеш-значення m', застосовуючи криптографічне перетворення підпису S з використанням відкритого ключа E:

$m' = S^E \pmod{N}$.

Крім того, він знаходить результат хешування прийнятого повідомлення M за допомогою такої ж хеш-функції h():

$m = h(M)$.

Якщо дотримується рівність обчислених значень m' та m:

$S^E \pmod{N} = h(M)$,

то одержувач визнає пару (M,S) справжньою. Доведено, що тільки власник секретного ключа D може сформувати цифровий підпис S під документом M, а визначити секретне число D по відкритому числу E не легше,

ніж розкласти модуль N на множники.

Крім того, можна суворо математично довести, що результат перевірки цифрового підпису S буде позитивним тільки в тому випадку, якщо при обчисленні S був використаний секретний ключ D відповідний відкритому ключу E. Тому відкритий ключ E іноді називають "ідентифікатором" підписувача.

Верифікація відкритого ключа

Безпосереднє використання відкритих ключів вимагає додаткового їх захисту й ідентифікації для визначення зв'язку із секретним ключем. Без такого додаткового захисту зловмисник може представити себе як відправником підписаних даних, так і одержувачем зашифрованих даних, замінивши значення відкритого ключа або порушивши його ідентифікацію. У цьому випадку кожний може видати себе за англійську королеву. Все це приводить до необхідності верифікації відкритого ключа. Для цих цілей використовується електронний сертифікат.

Електронний сертифікат - цифровий документ, що зв'язує відкритий ключ із певним користувачем або додатком. Завіряє сертифікат Центр Сертифікації (ЦС). Виходячи з функцій, які виконує ЦС, він є основним компонентом всієї Інфраструктури Відкритих Ключів. За допомогою ЦС, кожний користувач може перевірити достовірність відкритого ключа. Лише сертифіковані ключі мають юридичну силу.

Завдання: Створити програму, яка повинна здійснювати підписання документа та перевірку підпису. Цифровий підпис реалізувати на основі алгоритму RSA (Лаболаторна робота №3, №5). **Розмір файлу не мнеш 100 кБ.**

Контрольні питання:

1) Що таке аутентифікація цифрових документів? Від чого вона захищає?

2) Що таке електронний цифровий підпис?

3) Чим подібні та чим відрізняються алгоритми накладання цифрового підпису та алгоритми асиметричного фифрування?

4) Як відбувається накладання ЕЦП?

5) Як відбувається перевірка ЕЦП?

Лабораторна робота №7

Тема: Криптоаналіз. Взлом RSA

Мета: навчитися реалізовувати криптоаналіз алгоритму RSA для коротких ключів шифрування

Розглянемо різні способи атак на одну з найбільш популярних схем відкритого шифрування та цифрового підпису - RSA. Розглянуті методи припускають наявність певних математичних слабостей схеми, внаслідок недосконалої програмної реалізації. Також приводяться засоби протидії цим атакам. Їх необхідно брати до уваги при реалізації схеми RSA або протоколів, заснованих на ній.

Насамперед згадаємо саму **схему підпису алгоритмом RSA:**

1. Вибираються p, q - великі прості числа. Обчислюється добуток n = pq.
2. Обчислюється φ(n) = (p-1)(q-1). Вибирається e - взаємно просте з φ(n).
3. З рівняння (e·d) mod φ(n) = 1 знаходиться число d.

Отримані числа (e, n) - відкритий ключ користувача, а d - секретний ключ.

Процедура шифрування: $C = M^e \pmod n$, де C - зашифрований текст, M - відкритий текст, що задовольняє наступній умові: $M^{\varphi(n)} \pmod n = 1$.

Процедура дешифрування: $M = C^d \pmod n$.

Генерація цифрового підпису: цифровий підпис $Q = M^d \pmod n$.

Перевірка цифрового підпису: $Q^e \pmod n \stackrel{?}{=} M$.

1. Метод безключового читання RSA.

Початкові умови: Супротивникові відомий відкритий ключ (e, n) і шифротекст C.

Завдання: Знайти вихідний текст M.

Алгоритм:

1. Супротивник підбирає число j, для якого виконується наступне співвідношення:

$$C^{e^J} \pmod n = C,$$

2. Знайшовши таке j, супротивник може обчислити відкритий текст M за формулою:

$$M = C^{e^{J-1}} \pmod n,$$

Це слідує з того, що $(C^e)^e)^e..)^e \pmod n = C^{e^J} \pmod n = (C^{e^{J-1}} \pmod n)^e = C$. Тобто деяке число $C^{e^{J-1}} \pmod n$ у ступені e дає шифротекст C. А це число якраз і є текстом M.

Приклад 7.1.

Початкові умови: відкритий ключ e = 5, n = 119, шифр C = 95.

Завдання: дізнатися значення тексту M.

$C^e 2 (\text{mod } n) = 95^5 2 (\text{mod } 119) = 109$

$C^e 3 (\text{mod } n) = 95^5 3 (\text{mod } 119) = 79 = M$

$C^e 4 (\text{mod } n) = 95^5 4 (\text{mod } 119) = 95 = C$

Відповідь: M = 79.

2. Атака на підпис RSA у схемі з нотаріусом.

Початкові умови. Є електронний нотаріус, що підписує документи, що проходять через нього. N - деякий відкритий текст, що нотаріус не бажає підписувати. Супротивникові відомий відкритий ключ (e,n) нотаріуса.

Завдання. Підписати цей текст N.

Супротивник виробляє якесь випадкове число x, взаємопросте з N і обчислює $y=x^e (\text{mod } n)$. Потім одержує значення M=y і передає його на підпис нотаріусові. Той підписує (адже це вже не текст N!) $M^d (\text{mod } n)=S$. Одержуємо, що

$S=M^d (\text{mod } n)=y^d N^d=(x^e)^d N^d = xN^d$,

а значить $N^d=Sx^{-1} (\text{mod } n)$. Залишається просто поділити отримане S на x.

Захист. При підписі додавати деяке випадкове число в повідомлення (наприклад, час). У такий спосіб вийде викривлення числа M при підписанні.

3. Атака на підпис RSA по обраному шифротексту.

Початкові умови. Є шифротекст C. Супротивникові відомий відкритий ключ (e,n) відправника повідомлення.

Завдання. Знайти текст M

Супротивник вибирає якесь r: r<n, (r,n)=1 і обчислює $x=r^e (\text{mod } n)$. Потім обчислює

$t= r^{-1} (\text{mod } n)$ і $y=x (\text{mod } n)$ і посилає y на підпис відправникові.

Відправник, нічого не підозрюючи, підписує текст y: $w=y^d (\text{mod } n)$ і відправляє w назад.

Супротивник обчислює $tw (\text{mod } n)= r^{-1}y^d (\text{mod } n)=$ (оскільки $r=x^d \text{ mod } n)= x^{-d}x^d C^d (\text{mod } n)=C^d=M$.

Супротивник не може відразу послати C на підпис, тому що відправник переглядає отримані в результаті підпису повідомлення й може помітити провокацію.

Атака носить трохи гіпотетичний характер, але проте дозволяє зробити кілька важливих висновків: а) підписувати й шифрувати треба різними ключами, або б) додавати випадковий вектор при підписанні чи використовувати хеш-функцію.

Завдання: Створити програму, що реалізує один з розглянутих алгоритмів криптоаналізу RSA.

Контрольні питання:

1) В чому полягає метод безключового читання RSA?

2) В чому полягає атака на підпис RSA у схемі з нотаріусом?

3) В чому полягає атака на підпис RSA по обраному шифротексту?

4) Наскільки ефективні дані методи та як їх робота залежить від довжини ключа шифрування?

5) Як можна протистояти вказаним методам криптоаналізу?

Лабораторна робота №8

Тема: Програмні закладки. Клавіатурний шпигун

Мета: ознайомитися з поняттям програмних закладок, способами їх реалізації та протидії

Програмні закладки - своєрідні програми, що використовують вірусну технологію прихованого встановлення, поширення та активізації. Однак, на відміну від вірусів, які просто знищують інформацію, програмні закладки, насамперед, призначені для її несанкціонованого одержання.

По методу впровадження в комп'ютерну систему програмні закладки можна розділити на:

- програмно-апаратні закладки, асоційовані з апаратними засобами комп'ютера (їхнє середовище перебування, як правило - BIOS);
- завантажувальні закладки, асоційовані з програмами початкового завантаження, які розташовуються в завантажувальних секторах;
- драйверні закладки, асоційовані із драйверами;
- прикладні закладки, асоційовані із прикладним програмним забезпеченням загального призначення;
- виконуючі закладки, асоційовані із виконуючими програмними модулями, що містять код цієї закладки;
- закладки-імітатори, інтерфейс яких збігається з інтерфейсом деяких службових програм, що вимагають введення конфіденційної інформації (паролів, криптографічних ключів, номерів кредитних карток);
- замасковані закладки, які маскуються під програмні засоби оптимізації роботи комп'ютера або під програми ігрового та розважального призначення.

Найпоширенішим типом закладок є клавіатурні шпигуни. Метою таких закладок є одержання інформації, що вводиться із клавіатури, (у тому числі й паролів), збереження її у зарезервованих для цього секторах, а потім пересилання накопичених даних по мережі на комп'ютер зловмисника.

Клавіатурні шпигуни діляться на *імітатори*, *фільтри* та *замінювачі* залежно від способу перехоплення паролів.

Імітатори працюють за наступним алгоритмом. Зловмисник впроваджує в операційну систему програмний модуль, що імітує запрошення до авторизації, щоб увійти в систему. Потім імітатор переходить у режим очікування введення ідентифікатора та пароля. Після того як потрібна інформація буде введена, програма відправляє отримані дані в місце, доступне зловмисникові й закривається. Після чого з'являється оригінал входу, у якому користувач, забивши потрібні дані, зможе потрапити в систему. Найбільш просунуті програми видають повідомлення про помилку перед своїм закриттям,

щоб не викликати підозр.

Фільтри «полюють» за всіма даними, які користувач операційної системи вводить із клавіатури комп'ютера. Фільтри одержують сигнали від клавіатури, які далі аналізуються на предмет виявлення даних про різні паролі. Найпростіший фільтр збирає в домовленому місці всі введені із клавіатури дані, більш витончені здатні відфільтровувати тільки ті дані, які необхідні зловмисникові.

Замінювачі повністю або частково підмінюють собою програмні модулі операційної системи, відповідальні за аутентифікацію користувачів. Перед виконанням замінювачам необхідно: 1) подібно комп'ютерному вірусу впровадитися в один або кілька системних файлів; 2) використовувати інтерфейсні зв'язку між програмними модулями системи аутентифікації для вбудовування себе в ланцюжок обробки введеного користувачем пароля.

Крім того, багато клавіатурних шпигунів відслідковують список запущених додатків, уміють робити "знімки" екрана за заданим розкладом, шпигувати за вмістом буфера обміну й вирішувати ряд завдань, націлених на приховане спостереження за користувачем. Записувана інформація зберігається на диску й більшість сучасних клавіатурних шпигунів можуть формувати різні звіти, можуть передавати їх по електронній пошті або http/ftp протоколу. Крім того, ряд сучасних клавіатурних шпигунів користуються RootKit технологіями для маскування слідів своєї присутності в системі.

Перед описанням основних принципів роботи клавіатурного шпигуна необхідно розглянути модель апаратного введення системи Windows.

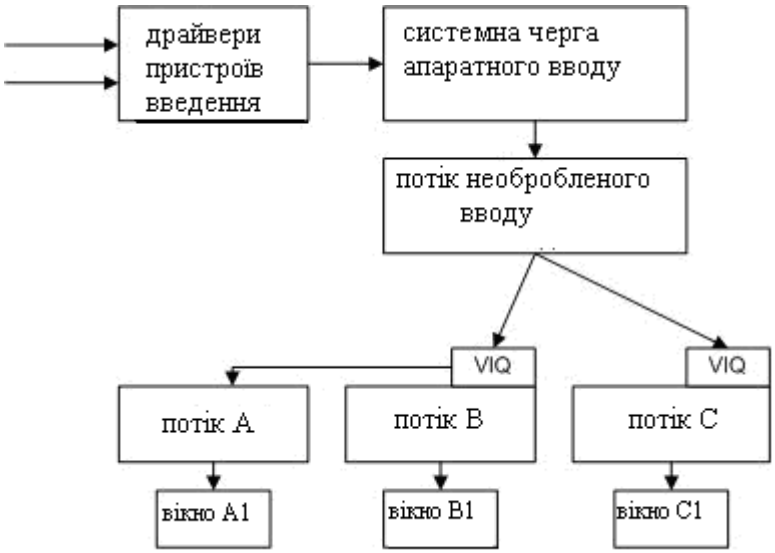

Рис. 8.1. Модель апаратного введення системи Windows

При виникненні будь-яких подій введення (натисканні клавіш,

переміщенні миші) події обробляються відповідним драйвером і поміщаються в системну чергу апаратного введення У системі є особливий потік неопрацьованого введення, який називається RIT (Raw Input Thread), що витягає події із системної черги та перетворює у повідомлення. Отримані повідомлення поміщаються в кінець черги віртуального введення одного з потоків (віртуальна черга потоку називається VIQ - Virtualized Input Queue). При цьому RIT сам з'ясовує, у чергу якого конкретно потоку необхідно помістити подію. Для подій миші потік визначається пошуком вікна, над яким розташований курсор миші. Клавіатурні події відправляються тільки одному потоку - так званому активному потоку (тобто потоку, якому належить вікно, з яким працює користувач). Насправді це не зовсім так - зокрема, на рисунку показаний потік А, що не має черги віртуального введення У цьому випадку виходить, що потоки А і В спільно використовують одну чергу віртуального введення. Це досягається за допомогою виклику API функції AttachThreadInput, що дозволяє одному потоку підключитися до черги віртуального введення іншого потоку.

Слід зазначити, що потік неопрацьованого введення відповідає за обробку спеціальних сполучень клавіш, зокрема Alt+Tab і Ctrl+Alt+Del.

Спостереження за клавіатурним вводом за допомогою пасток

Дана методика є класичною для клавіатурних шпигунів. Суть методу складається в застосуванні механізму пасток (hook) операційної системи. Пастки дозволяють спостерігати за повідомленнями, які обробляються вікнами інших програм. Установка та видалення пасток виконується за допомогою добре документованих функцій API бібліотеки user32.dll (функція SetWindowsHookEx дозволяє встановити пастку, UnhookWindowsHookEx - зняти її). При установці пастки вказується тип повідомлень, для яких повинен викликатися оброблювач пастки. Зокрема, є два спеціальних типи пастки WH_KEYBOARD і WH_MOUSE - для реєстрації подій клавіатури й миші відповідно. Пастка може бути встановлена для заданого потоку й для всіх потоків системи. Пастка для всіх потоків системи дуже зручна для побудови клавіатурного шпигуна.

Код оброблювача подій пастки повинен бути розташований в DLL. Ця вимога пов'язане з тим, що DLL з оброблювачем пастки проектується системою в адресний простір всіх GUI процесів. Цікавою особливістю є те, що проектування DLL відбувається не в момент установки пастки, а при одержанні GUI процесом першого повідомлення, що задовольняє параметрам пастки.

Методика пасток досить проста й ефективна, але в ній є ряд недоліків. Першим недоліком можна вважати те, що DLL з пасткою проектується в адресний простір всіх GUI процесів, що може застосовуватися для виявлення

клавіатурного шпигуна. Крім того, реєстрація подій клавіатури можлива тільки для GUI додатків.

Спостереження за клавіатурним вводом за допомогою опитування клавіатури

Дана методика заснована на періодичному опитуванні стану клавіатури. Для опитування стану клавіш у системі передбачена спеціальна функція GetKeyboardState, що повертає масив з 255 байт, у якому кожний байт містить стан певної клавіші на клавіатурі. Даний метод уже не вимагає впровадження DLL в GUI процеси й у результаті шпигун менш помітний.

Однак зміна статусу клавіш відбувається в момент зчитування потоком клавіатурних повідомлень із його черги, і в результаті подібна методика працює тільки для спостереження за GUI додатками. Від цього недоліку звільнена функція GetAsyncKeyState, що повертає стан клавіші на момент виклику функції.

Недоліком клавіатурних шпигунів такого типу є необхідність періодичного опитування стану клавіатури з досить високою швидкістю, не менш 10-20 опитувань у секунду.

Спостереження за клавіатурним вводом за допомогою перехоплення API функцій

Дана методика не одержала широкого поширення, але проте вона може з успіхом застосовуватися для побудови клавіатурних шпигунів. Найпростішим способом може бути перехоплення функцій GetMessage, PeekMessage і TranslateMessage бібліотеки User32, що дозволить вести моніторинг всіх повідомлень, прийнятих GUI додатками.

Клавіатурний шпигун на базі драйверу

Даний метод ще більш ефективний, ніж описані вище методи. Можливі як мінімум два варіанти реалізації цього методу - написання та установка в систему свого драйвера клавіатури замість штатного або встановлення драйвера - фільтра.

При створенні свого драйверу треба перехоплювати переривання від клавіатури (IRQ 1) та напряму звертатися до портів вводу-виводу (60h, 64h).
Приклади драйверів-фільтрів: драйвер-фільтр драйвера класу клавіатури Kbdclass, драйвер-фільтр функціонального драйвера i8042prt. *Драйвер-фільтр драйвера класу клавіатури Kbdclass* перехоплює запити до клавіатури за допомогою встановлення фільтра зверху пристрою "\Device\KeyboardClass0", створеного драйвером Kbdclass. Фільтруються тільки запити типу IRP_MJ_READ, оскільки саме вони дозволяють одержати коди натиснутих і відпущених клавіш. *Драйвер-фільтр функціонального драйвера i8042prt* перехоплює запити до клавіатури за допомогою встановлення фільтра зверху

безіменного пристрою, що створюється драйвером i8042prt під пристроєм Device\KeyboardClass0. Драйвер i8042prt представляє собою програмний інтерфейс для додавання додаткової функції обробки переривання IRQ1 (IsrRoutine), у якій можна зробити аналіз отриманих від клавіатури даних.

Апаратні клавіатурні шпигуни

До них належать:

– Установка пристрою спостереження в розрив кабелю клавіатури (наприклад, пристрій може бути виконаний у вигляді перехідника PS/2);

– Вбудовування пристрою спостереження в клавіатуру;

– Зчитування даних шляхом реєстрації побічних електромагнітних випромінювань і наведень;

– Візуальне спостереження за клавіатурою.

Апаратні клавіатурні шпигуни зустрічаються набагато рідше, ніж програмні. Однак при перевірці особливо відповідальних комп'ютерів (наприклад, тих, що використовуються для здійснення банківських операцій) про можливість апаратного спостереження за клавіатурним введенням не слід забувати.

Методики пошуку клавіатурних шпигунів

Пошук по сигнатурах. Даний метод не відрізняється від типових методик пошуку вірусів. Сигнатурний пошук дозволяє однозначно ідентифікувати клавіатурні шпигуни, при правильному виборі сигнатур ймовірність помилки практично дорівнює нулю. Однак сигнатурний сканер зможе виявляти заздалегідь відомі й описані в його базі дані об'єкти;

Евристичні алгоритми. Як очевидно з назви, це методики пошуку клавіатурного шпигуна по його характерних рисах. Евристичний пошук носить ймовірнісний характер. Як показала практика, цей метод найбільш ефективний для пошуку клавіатурних шпигунів найпоширенішого типу - заснованих на пастках. Однак подібні методики дають багато помилкових спрацьовувань. Існують сотні безпечних програм, що не є клавіатурними шпигунами, але використовують пастки для спостереження за клавіатурним вводом і мишею. Найпоширеніші приклади - програми Punto Switcher, словник Lingvo, програмне забезпечення від мультимедійних клавіатур і мишей;

Моніторинг API функцій. Дана методика заснована на перехопленні ряду функцій, що застосовуються клавіатурним шпигуном - зокрема, функцій SetWindowsHookEx, UnhookWindowsHookEx, GetAsyncKeyState, GetKeyboardState. Виклик даних функцій будь-яким додатком дозволяє вчасно підняти тривогу, однак проблеми численних помилкових спрацьовувань будуть аналогічні методу 2;

Відстеження драйверів, процесів і сервісів, що використовуються

системою. Це універсальна методика, що застосовується не тільки проти клавіатурних шпигунів а й проти вірусів. Цей метод застосовують Kaspersky Inspector або Adinf, які відслідковують появу в системі нових файлів.

Завдання: Створити клавіатурний шпигун, що при кожному запуску створює файл у який записує дату, ім'я користувача, програму, до якої йде звертання і текст набраний на клавіатурі. А також створити програму, що знаходить та знищує цю програмну закладку. Реалізувати будь-яким з вище перерахованих методів.

Контрольні питання:

1) Що таке програмна закладка?
2) Які існують програмні закладки?
3) Як реалізувати програмну закладку?
4) Які існують методи протидії програмним закладкам?
5) Перелічіть методи пошуку клавіатурних шпигунів.

Лабораторна робота №9

Тема: Стеганографія

Мета: навчитися реалізовувати LSB-метод стеганографії для вбудовування прихованих даних у графічні зображення

Стеганографія - це наука про приховану передачу інформації шляхом збереження в таємниці самого факту передачі. На відміну від криптографії, що приховує вміст секретного повідомлення, стеганографія приховує саме його існування.

Наприкінці 90-х років виділилося кілька напрямків стеганографії: к**ласична, комп'ютерна та цифрова.**

Кілька прикладів класичної стеганографії: використання «невидимих» чорнил; запис на бічній стороні колоди карт, розташованих у домовленому порядку; «жаргонні шифри», де слова мають інше обумовлене значення; трафарети, які будучи покладеними на текст, залишають видимими тільки значущі букви; вузлики на нитках і т.д.

Комп'ютерна стеганографія — напрямок класичної стеганографії, заснований на особливостях комп'ютерної платформи. Приклади: стеганографічна файлова система StegFS для Linux, приховання даних у невикористовуваних областях форматів файлів, підміна символів у назвах файлів, текстова стеганографія й т.д.

Цифрова стеганографія — напрямок класичної стеганографії, заснований на прихованні або вбудовуванні додаткової інформації в цифрові об'єкти. Але, як правило, дані об'єкти є мультимедіа-об'єктами (зображення, відео, аудіо, текстури 3D-об'єктів). При цьому файли звичайно спотворюються, але такі спотворення перебувають нижче порога чутливості середньостатистичної людини і не приводять до помітних змін цих об'єктів. Крім того в оцифрованих об'єктах, що споконвічно мають аналогову природу, і так завжди є присутнім шум.

Застосування цифрової стеганографії

Цифрова стеганографія як наука народилася буквально в останні роки. Вона містить у собі наступні напрямки:

1) вбудовування інформації з метою її прихованої передачі;

2) вбудовування цифрових водяних знаків (ЦВЗ) (watermarking);

3) вбудовування ідентифікаційних номерів (fingerprinting);

4) вбудовування заголовків (captioning).

Найбільш поширений легальний напрямок стеганографії - вбудовування цифрових водяних знаків, що є основою для систем захисту авторських прав. Методи цього напрямку спрямовані на вбудовування прихованих маркерів,

стійких до різних спотворень контейнера (архівації, атак зловмисників тощо). Вбудовуватися може як текст, так і малюнок невеликого розміру.

Алгоритми

По способу вбудовування інформації стегоалгоритми можна розділити на лінійні (адитивні), нелінійні та інші.

Найбільш відомими представниками **прямих методів** є:

- LSB-методи (Least Significant Bits);
- Методи модифікації палітри.

Нелінійні:

- дискретне косинус-перетворення (ДКП);
- вейвлет-перетворення;
- дискретне перетворення Фур'є;
- перетворення Карунена-Лоєва;
- сингулярне розкладання.

До **інших** відноситься наприклад метод на основі фрактального перетворення.

Метод LSB

LSB (Least Significant Bit, найменший значущий біт) — суть цього методу полягає в заміні останніх значущих бітів у контейнері (зображенні, аудіо- чи відеозаписі) на біти приховуваного повідомлення. Різниця між порожнім і заповненим контейнерами повинна бути не відчутна для органів сприйняття людини.

Методи LSB є нестійкими до всіх видів атак і можуть бути використані тільки при відсутності шуму в каналі передачі даних. Також наявність прихованого повідомлення легко виявити. Виявлення такого стегоповідомлення здійснюється по аномальних характеристиках розподілу значень діапазону молодших бітів цифрового сигналу (рис. 1).

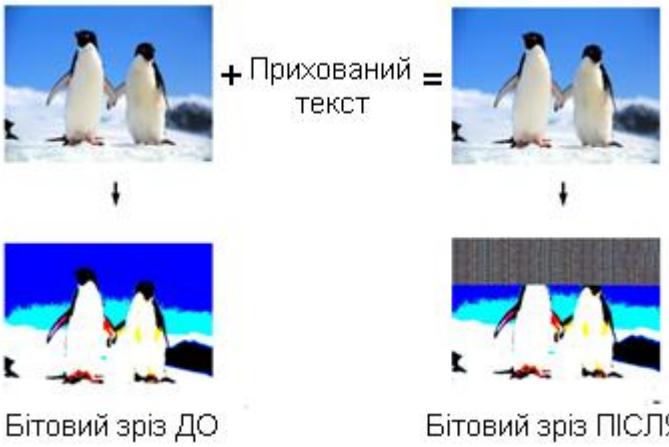

Рис. 9.1. Вигляд бітового зрізу файлу до та після вдубовування текстового повідомлення методом LSB

Всі методи LSB являються як правило адитивними.Найлегше даний метод застосовувати для .bmp файлів.

Формат bmp (від слів BitMap - бітова карта, бітовий масив) – один з растрових форматів операційної системи *Windows*, який запам'ятовує одно- і багатокольорові *(RGB)* зображення у формі *Pixel*.

Структура Bmp-файлу

Ім'я	**Довжина**	**Зсув**	**Опис**
Заголовок файлу растрової графіки (BitMapFileHeader) (14 байт)			
Type	2	0	Сигнатура (Код 4D42h - Букви 'BM')
Size	4	2	Розмір файлу
Reserved 1	2	6	Зарезервовано
Reserved 2	2	8	Зарезервовано
OffsetBits	4	10	Місцезнаходження даних растрового масива
Інформаційний заголовок растрового масива (BitMapInfoHeader) (40 байт)			
Size	4	14	Довжина цього заголовка
Width	4	18	Ширина зображення (в пікселях)
Height	4	22	Висота зображення (в пікселях)
Planes	2	26	Число площин
BitCount	2	28	Глибина кольору, біт/піксель
Compression	4	30	Тип компресії (0 - нестиснуте зображення)
SizeImage	4	34	Розмір зображення, байт
XpelsPerMeter	4	38	Горизонтальне розрішення, піксель/метр
YpelsPerMeter	4	42	Вертикальне розрішення, піксель/метр
ColorsUsed	4	46	Число кольорів, що використовуються
ColorsImportant	4	50	Число основних кольрів
Таблиця кольрів (ColorTable) (довжина змінюється від 8 до 1024 байт)			
ColorTable	4*N	x	палітра (256 елементів по 4 байтив для 256-кольрового зображення)
Дані зображення (BitMap Array)			
Image	Size	x	Зображення, записане по рядках з ліва на право і знизу вгору

ВАЖЛИВО!

1. Зображення зберігається построчно знизу-вгору. Для збереження кожного рядка виділяється кратна 4 кількість байт. У незначащих байтах зберігається сміття. Старшому біту або тетраді відповідає самий лівий піксель.

2. Не всі файли BMP мають таку структуру. Наприклад, файли BMP із глибиною 16 і 24 біт/піксель не мають таблиць кольрів; у цих файлах значення пікселів растрового масиву безпосередньо характеризують значення кольрів RGB.

При зберіганні зображення True Color кожному пікселю відповідають три послідовні байти, що зберігають складові кольори B, G, R; (не R, G, B).

Елемент палітри являє собою чотирьохбайтовий запис (структуру). У цій структурі зберігаються складові R-червоного, G-зеленого та B-синього кольорів. Один байт зарезервований. Палітра може й не використовуватися, наприклад в True Color.

Структура елемента палітри:

```
typedef struct tagRGBQUAD
 {
        char   rgbBlue;
        char   rgbGreen;
        char   rgbRed;
        char   rgbReserved;
 } RGBQUAD;
```

У полі **Тип стиснення** може стояти:

0 - стиснення не використовується

1 - RLE8 стиснення (для 256-ти кольорового зображення)

2 - RLE4 стиснення (для 16-ти кольорового зображення)

У полі **Глибина кольору**:

1 - чорно-біле зображення

4 - 16-ти кольорове

8 - 256-ти кольорове

24 - True Color

```
typedef unsigned long DWORD;     // Подвійне слово - 32 біти (розряди)
 typedef unsigned int WORD;          //  Слово - 16 бітів (розрядів)
 typedef signed long LONG;
 typedef unsigned int UINT;
 // Заголовок файлу
 typedef struct tagBITMAPFILEHEADER
 {
        UINT bfType;                 // 'BM' = 4D42h
        DWORD bfSize;
        UINT bfReserved1;
        UINT bfReserved2;
        DWORD bfOffBits;         // Зсув до растра
 } BITMAPFILEHEADER;

// Заголовок Bitmap
 typedef struct tagBITMAPINFOHEADER
 {
        DWORD biSize;
```

```
        LONG biWidth;
        LONG biHeight;
        WORD biPlanes;
        WORD biBitCount;
        DWORD biCompression;
        DWORD biSizeImage;
        LONG biXPelsPerMeter;
        LONG biYPelsPerMeter;
        DWORD biClrUsed;
        DWORD biClrImportant;
} BITMAPINFOHEADER;
```

Завдання: Розробити програму, яка реалізує метод стеганографії LSB для зображень формату .bmp. Програма повинна вбудовувати та вилучати приховане повідомлення.

Контрольні питання:

1) Що таке стеганографія? Чим вона відрізняється від криптографії?

2) Що собою представляє структура BMP-файлу? Що таке найменщ значущий біт?

3) В чому полягає метод LSB?

4) Як можна виявити дані приховані методом LSB?

5) Які удосконалення можна внести в метод LSB, щоб ускладнити можливість виявлення прихованих даних?

Список літератури

1. Смірнов О.А., Віхрова Л.Г., Осадчий С.І., Ковтун В.Ю., Мелешко Є.В. Основи захисту інформації: Навчальний посібник. – Кіровоград: РВЛ КНТУ, 2011. – 322 с.
2. Закон Украины "О защите информации в автоматизированных системах" от 5 июня 1994 г. // Безопасность информации, 1995, N1. С. 56-62.
3. http://www.intuit.ru/department/security/secbasics/
4. Галатенко В.А. Основы информационной безопасности. Интернет-университет информационных технологий - ИНТУИТ.ру, 2008
5. Лапонина О.Р. Основы сетевой безопасности: криптографические алгоритмы и протоколы взаимодействия. Интернет-университет информационных технологий - ИНТУИТ.ру, 2005
6. Галатенко В.А. Стандарты информационной безопасности. Интернет-университет информационных технологий - ИНТУИТ.ру, 2005
7. Э. Мэйволд Безопасность сетей Эком, 2006
8. Хаулет Т. Защитные средства с открытыми исходными текстами БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007
9. Department of Defense Trusted Computer System Evaluation Criteria DoD 5200.28-STD, 1993.
10. Information Technology Security Evaluation Criteria (ITSEC). Harmonized Criteria of France - Germany - the Netherlands - the United Kingdom Department of Trade and Industry, London, 1991.
11. Security Architecture for Open Systems Interconnection for CCITT Applications. Recommendation X.800 CCITT, Geneva, 1991.
12. Site Security Handbook. Holbrook P., Reynolds J., Request for Comments: 1244, 1991.
13. James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback Report on the Development of the Advanced Encryption Standard (AES) Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Technology Administration U.S. Department of Commerce. 2000г. 116с.
14. Государственный Стандарт Российской Федерации ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ. Процедуры выработки и проверки электронной цифровой подписи на базе асимметричного криптографического алгоритма 1994г.

15. Государственный Стандарт Российской Федерации ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ. Функция хэширования 1994г.
16. RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile 2002г. 129с.
17. RFC 3281 An Internet Attribute Certificate Profile for Authorization 2002г. 40с.
18. RFC 2510 Internet X.509 Public Key Infrastructure Certificate Management Protocols 1999г. 72с.
19. RFC 2511 Internet X.509 Certificate Request Message Format 1999г. 25с.
20. RFC 3369 Cryptographic Message Syntax 2002г. 60с.
21. RFC 2560 X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP 1999г. 23с.
22. RFC 2797 Certificate Management Messages over CMS 2000г. 47с.
23. RFC 3379 Delegated Path Validation and Delegated Path Discovery Protocol Requirements 2002г. 15с.
24. RFC 2633 S/MIME Version 3 Message Specification 1999г. 32с.
25. RFC 2632 S/MIME Version 3 Certificate Handling 1999г. 13с.
26. Security Architecture for the Internet Protocol RFC 2401 1998г. 66с.
27. Internet Security Association and Key Management Protocol (ISAKMP) RFC 2408 1998г. 86с.
28. The Internet Key Exchange (IKE) RFC 2409 1998г. 41с.
29. ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевірка // Держстандарт України, К.:2003.