МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

І. В. УСТЕНКО

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «БЕЗПЕКА ПРОГРАМ ТА ДАНИХ»

Миколаїв
2021

УДК 004.056(075)
У 79

Автор:
І. В. Устенко, кандидат технічних наук, доцент, доцент кафедри програмного забезпечення автоматизованих систем Національного університету кораблебудування імені адмірала Макарова.

Рецензент:
М.А.Годовиченко, кандидат технічних наук, доцент кафедри проектного навчання в інформаційних технологіях Одеського національного політехнічного університету

Затверджено та рекомендовано до друку рішенням методичної ради Національного університету кораблебудування імені адмірала Макарова.

Устенко І. В.

У79 Методичні вказівки до виконання лабораторних робіт з дисципліни «Безпека програм та даних» / І. В. Устенко. – Миколаїв : НУК імені адмірала Макарова, 2021. – 44 с.

Методичні вказівки з дисципліни «Безпека програм та даних» мають направленість на закріплення теоретичних та отримання практичних знань в галузі теорії інформаційної безпеки. Методичні вказівки укладені у відповідності з робочою навчальною програмою дисципліни «Безпека програм та даних» для студентів третього курсу спеціальності 121 «Інженерія програмного забезпечення» та містять основні теоретичні положення, завдання на лабораторні роботи та контрольні запитання для перевірки знань, набутих студентами при вивченні дисципліни.

УДК 004.056(075)

ВСТУП

При виконанні лабораторних робіт кожен студент отримує індивідуальне завдання. Номер варіанта відповідає порядковому номеру студента у списку групи. По кожній лабораторній роботі оформлюється звіт, що повинен задовольняти зазначеним далі вимогам, частина з яких обов'язкова для виконання.

З дисципліни «Безпека програм та даних» передбачена бальна система оцінювання успішності студентів. Відповідно до неї за кожну виконану лабораторну роботу студент отримує бали за факт, своєчасність і якість виконання та захисту лабораторної роботи. Якість виконання роботи оцінюється за кількома критеріями, частина з яких загальна для всіх лабораторних робіт (ці критерії наводяться далі), а частина – специфічна для конкретної лабораторної роботи (ці критерії наводяться в описі кожної роботи).

Бали за своєчасність виконання та захисту лабораторної роботи нараховують за принципом «чим раніше здана робота, тим більше балів набрано». Прийом викладачем кожної лабораторної роботи включає три етапи (в зазначеному порядку):

1) демонстрація роботи програми на комп'ютері (перевіряється правильність роботи програми);

2) пояснення вмісту звіту (перевіряється розуміння логіки роботи програми, знання теоретичного матеріалу, на основі якого написана програма, відповідність звіту встановленим вимогам);

3) захист лабораторної роботи (відповіді на питання).

Робота №1

ТИПОВІ СХЕМИ ІДЕНТИФІКАЦІЇ ТА АУТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ КОМП'ЮТЕРНОЇ СИСТЕМИ

Мета роботи: закріплення знань про сервісні служби та механізми захисту інформації, про порушення захисту.

Стислі теоретичні відомості

Існує декілька типових схем для ідентифікації та аутентифікації користувачів в комп'ютерній системі.

Схема 1. У комп'ютерній системі виділяється об'єкт-еталон для ідентифікації та аутентифікації користувачів. Структура об'єкта-еталона показана в табл. 1.1. Тут $E_i = F(ID_i, K_i)$, де F – функція, яка має властивість «невідновності» значення K_i по E_i і ID_i. «Невідновність» K_i оцінюється деякою пороговою трудомісткістю T_0 рішення задачі відновлення інформації, що аутентифікує K_i по E_i і ID_i.

Таблиця 1.1 – Структура об'єкту-еталону для схеми 1

N користувача	Інформація для ідентифікації	Інформація для аутентифікації
1	ID_1	E_1
2	ID_2	E_2
…	…	…
N	ID_n	E_n

Протокол ідентифікації та аутентификации (для схеми 1).

1. Користувач надає свій ідентифікатор ID.
2. Якщо ID не збігається ні з одним ID_i, зареєстрованим у когось в комп'ютерній системі, то ідентифікація відкидається – користувач не допускається до роботи, інакше (існує $ID_i = ID$) встановлюється, що користувач, який назвався користувачем *i*, пройшов ідентифікацію.
3. Суб'єкт аутентифікації вимагає у користувача його аутентифікатор K.
4. Суб'єкт аутентифікації обчислює значення $E = F(ID_i, K)$.
5. Суб'єкт аутентифікації порівнює значення E і E_i. При збіганні цих значень встановлюється, що даний користувач успішно аутентифікований в си-

стемі. Інформація про цього користувача передається в програмні модулі, що використовують ключі користувачів (тобто в систему шифрування, розмежування доступу тощо). В іншому випадку аутентифікація відкидається – користувач не допускається до роботи.

Дана схема ідентифікації та аутентифікації користувача може бути модифікована. Модифікована схема 2 володіє кращими характеристиками в порівнянні зі схемою 1.

Схема 2. В комп'ютерній системі виділяється модифікований об'єкт-еталон, структура якого показана в табл. 1.2.

Таблиця 1.2 – Структура модифікованого об'єкту-еталону

N користувача	Інформація для ідентифікації	Інформація для аутентифікації
1	ID_1, S_1	E_1
2	ID_2, S_2	E_2
…	…	…
N	ID_n, S_n	E_n

На відміну від схеми 1, у схемі 2 значення E_i дорівнює $F(S_i, K_i)$, де S_i – випадковий вектор, що задається при створенні ідентифікатора користувача, тобто при створенні рядка, необхідного для ідентифікації та аутентифікації користувача; F – функція, яка має властивість «невідновності» значення K_i по E_i та S_i.

Для збереження паролю користувача від підлягає шифруванню. В реальних комп’ютерних системах використовуються шифри з високою криптостійкістю, але в начальних цілях достатньо скористатись шифром Плейфеєра.

Шифр Плейфеєра

Одним з найбільш відомих шифрів, що базуються на заміщенні комбінацій букв, є *шифр Плейфеєра* (Playfair), у якому біграми відкритого тексту розглядаються як самостійні одиниці, перетворені в задані біграми шифрованого тексту.

Алгоритм Плейфеєра заснований на використанні матриці букв розмірності 5×5 (для англійського алфавіту, де I та J – одна буква), створеної на основі

деякого ключового слова. На рис. 1.1 наведена матриця отримана на основі ключа PLAYFAIR.

P	L	A	Y	F
I/J	R	B	C	D
E	G	H	K	M
N	O	Q	S	T
U	V	W	X	Z

Рисунок 1.1 – Матриця для шифрування

Матриця створюється шляхом розміщення букв, використаних у ключовому слові, зліва направо і зверху вниз (букви, що повторюються, відкидаються). Букви алфавіту, що залишились, розміщаються в природному порядку. Відкритий текст шифрується порціями по дві букви у відповідності з наступними правилами:

1. Якщо виявляється, що букви відкритого тексту, які повторюються, утворюють одну пару для шифрування, то між цими буквами вставляється спеціальна буква-заповнювач, наприклад X. Зокрема, таке слово як BALLOON буде перетворено до виду BA LX LO ON.

2. Якщо букви відкритого тексту попадають в один рядок матриці, то кожна з них заміняється наступною буквою (для заміни останнього елементу рядка матриці використовується перший елемент того ж рядка). Наприклад, RD шифрується як BI або BJ.

3. Якщо букви відкритого тексту попадають у той самий стовпець матриці, то кожна з них заміняється буквою розташованою нижче (для заміни останнього елементу стовпця використовується верхній елемент того ж стовпця). Наприклад, AW шифрується як BA.

4. Якщо не виконується жодне з приведених вище умов, кожна буква з пари букв відкритого тексту заміняється буквою, що знаходиться на перетині рядка матриці, що містить цю букву, і стовпця, в якому знаходиться друга буква відкритого тексту. Наприклад, RX шифрується як CV.

Завдання

1. Написати програму для створення об'єкта-еталона ідентифікації та аутентифікації користувачів комп’ютерної системи (схема 1):

– для шифрування пароля використовувати метод Плейфеєра;

– розмір матриці для шифрування 7×5;

– в якості ключового слова використовувати прізвище користувача з табл. 1.3 (номер варіанта відповідає номеру користувача);

– поточний для шифрування алфавіт:

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

– в якості букви-заповнювача використовувати **'**.

2. В об'єкті-еталоні зберігається наступна інформація, що представлена в табл. 1.3.

Таблиця 1.3 – Об’єкт-еталон

№ користувача	Інформація для ідентифікації **(ID_i)**	Інформація для аутентифікації (зашифрований пароль *i*-го користувача, **E_i)**
1	волков	дєкиалмфєп
2	корсун	гіаукїфп
3	пащенко	спгдсщ
4	коваленко	авко
5	пєтухова	авілгм
6	войтюк	їтлфо.
7	щербаков	фщбщцжй'
8	шишкун	хепбтфдїй'
9	демченко	йвткяфбї
10	лисенко	обкртбхліїю
11	смірнова	іпслчю
12	таран	уфрічю
13	волошин	йланвс
14	киришко	икшкїд
15	станько	тапєїь

Контрольні запитання

1. Дайте визначення ідентифікації.
2. Дайте визначення аутентифікації.
3. Що таке об'єкт-еталон для ідентифікації і аутентифікації користувачів?
4. Що таке суб'єкт аутентифікації?
5. Які основні властивості інформації необхідно підтримувати при забезпеченні інформаційної безпеки?
6. Дайте визначення парольного захисту інформації.
7. Дайте визначення терміна інформаційна безпека.
8. Які особливості парольного захисту інформації?
9. Наведіть алгоритм шифрування відкритого тексту за допомогою шифру Плейфеєра.
10. Що робити, якщо при шифруванні за допомогою шифру Плейфеєра символи з однієї біграми знаходяться в одному рядку?

Робота №2

БІОМЕТРИЧНА АУТЕНТИФІКАЦІЯ ЗА ДОПОМОГОЮ КЛАВІАТУРНОГО ПОЧЕРКУ

Мета роботи: закріплення знань про біометричну аутентифікацію за допомогою клавіатурного почерку.

Стислі теоретичні відомості

Таблиця класифікації формальних особливостей почерку людини містить наступні основні характеристики: сила руху, динамічність, напруженість, змістовність, витягнутість букв, нахил, ступінь зв'язності букв в слові, напрямок рядка, швидкість, спосіб тримання ручки, ритм, виразність тощо. Зрозуміло, що в разі аналізу клавіатурного почерку багато з перерахованих характеристик стають абсолютно безглуздими, завдяки наявності стандартних клавіатур і драйверів до них. Однак, скасовуючи ряд характеристик, комп'ютер дозволяє отримати інші характеристики, раніше недоступні, зокрема: швидкість набору різних слів відносно один одного, відносні інтервали між натисканнями клавіш, що належать різним полям клавіатури та ін.

При цьому нові характеристики навіть більш інформативні по відношенню до старих. Наприклад, характеристика «швидкість набору різних слів відносно один одного» при грамотно поставленому тесті дозволяє визначити практично однозначно сферу інтересів тестованого.

Подібний підхід можна використовувати при створенні програми «Детектор брехні». М'язова пам'ять руки не контролюється свідомістю і обов'язково буде проявляти себе.

Клавіатурний почерк можна описати наступними параметрами:

– швидкість введення – відношення кількості введених символів до часу набору;

– динаміка введення – відрізки часу між натисканнями клавіш і їх утриманням;

– частота виникнення помилок при введенні;

– характерне використання клавіш – наприклад, які функціональні клавіші натискає оператор при введенні великих літер.

Таким чином, особливості клавіатурного почерку представлено в табл. 2.1.

Таблиця 2.1 – Особливості клавіатурного почерку

Характеристика	Формула
1. Швидкість введення	$Tm = T/n$ де T – час набору, n – число символів
2. Швидкість введення кожного слова тестового тексту	$Ts[i] = T[i]/k[i]$ де $T[i]$ – час набору i-го слова тексту, $k[i]$ – кількість символів i-го слова тексту
3. Середній тимчасовий проміжок між словами	$T\text{п} = (T - \sum_{i=1}^{i=L} T[i])/L$ де L – кількість слів тексту
4. Ступінь зв'язності набору (обчислюється після виключення грубих помилок)	$S = SQRT\dfrac{(t[j] - M)^2}{n-1}$ де $t[j]$ – час між набором j та $j+1$ символів в слові тестового тексту (пробіли виключаються) $M = \sum_{i=1}^{i=L} T[i])/n$

Ефективність аутентифікації по клавіатурному почерку, як і будь-якого іншого біометричного методу, визначається за допомогою трьох основних критеріїв:

– FAR (False Acceptance Rate) – процентний поріг, який визначає ймовірність того, що одна людина може бути прийнятий за іншу (коефіцієнт помилкового доступу). FAR також називають «помилкою 2 роду»;

– FRR (False Rejection Rate) – ймовірність того, що людина може бути визначено залежно від системи (коефіцієнт помилкового відмови в доступі). FRR також називають «помилкою 1 роду»;

– CER (Crossover Error Rate) – точка, в якій калібрування системи призводить до рівності FAR = FRR.

Завдання

1. Зібрати статистику за своїми характеристиками введення тексту (відповідно до варіанту в табл. 2.2).

2. Написати програму для аутентифікації користувача за характеристиками клавіатурного почерку. Програма повинна розраховувати параметри введення тексту згідно варіанту і порівнювати отриманні значення з вашими параметрами (допустиме відхилення становить 10%).

Таблиця 2.2 – Варіанти завдань

Варіант	Завдання
1, 5, 10, 15	Швидкість введення тестового тексту, кількість похибок при введенні тексту
2, 6, 9, 11	Швидкість введення кожного слова тестового тексту, кількість похибок при введенні тексту
3, 7, 12, 14	Середній часовий проміжок між словами, кількість похибок при введенні тексту
4, 8, 13	Ступінь зв'язності набору, кількість похибок при введенні тексту

Контрольні запитання

1. Дайте визначення біометричної аутентифікації.

2. Поясніть суть біометричної аутентифікації за допомогою клавіатурного почерку.

3. Які ще існують засоби біометричної аутентифікації?

4. У чому полягають відмінності клавіатурного почерку від звичайного?

5. Якими параметрами можна описати клавіатурний почерк?

6. Як визначається ефективність аутентифікації за клавіатурним почерком?

Робота №3

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ. ТАБЛИЧНІ ПЕРЕСТАНОВКИ

Мета роботи: закріплення знань про криптографічні методи захисту інформації, про вимоги до криптографічних систем, про їх класифікацію, про різні типи криптографічних методів; придбання навичок практичної реалізації методів криптографічного перетворення, зокрема методів табличних перестановок.

Стислі теоретичні відомості

Перестановкою символів вихідного тексту називається його перевпорядкування таким чином, що символ з позиції i переміщається в позицію $k_{(i)}$. Наприклад, текст «НАДІШЛІТЬ_ДОПОМОГУ» на підставі ключа 3, 17, 9, 1, 4, 11, 6, 8, 18, 5, 14, 12, 10, 15, 13, 7, 16, 2 буде зашифрований у повідомлення «ДГЬНІДЛТУШОО_МПІОА».

Вектор індексів перестановок k і є ключем шифрування. Вектор k^{-1}, за допомогою якого відновлюється вихідний текст із шифровки є ключем дешифрування. Ключі k і k^{-1} взаємно однозначно перетворюються. Загальна можлива кількість перестановок вихідного тексту з n елементів дорівнює $n!$. Звідси випливає, що потенційно існує велика кількість відображень вихідного тексту в шифрований. Практична реалізація криптографічних систем вимагає, щоб перестановки були визначені алгоритмами, що залежать від відносно невеликої кількості параметрів (ключів). Перестановки дуже часто є основою для комбінованих криптографічних систем, а в чистому виді практично не застосовуються.

Найпростіший з таких шифрів використовує перетворення «*сходинок*», що полягає в тому, що відкритий текст записується уздовж похилих рядків визначеної довжини, а потім зчитується горизонтальними рядками. Наприклад, для шифрування тексту «НАДІШЛІТЬ_ДОПОМОГУ» за методом сходинок зі сходинками довжиною 3, запишемо текст у такому вигляді

Н І І _ П О
А Ш Т Д О Г
Д Л Ь О М У

Шифроване повідомлення буде «НІІ_ПОАШТДОГДЛЬОМУ». Такий «шифр» особливої складності для криптоаналізу не представляє.

Табличні перестановки – цілий клас досить простих у застосуванні і недостатньо надійних методів шифрування, що є різновидом методу перестановки. Різні варіанти шифрування даного класу досить часто застосовувались на практиці в середні століття.

Таблична перестановка за рядками і стовпцям – таблична перестановка, при якій вихідний текст записується в таблицю визначеного розміру рядками, а зашифрований текст зчитується стовпцями. Наприклад, текст «НАДІШЛІТЬ_ДОПОМОГУ» за допомогою табл. 3.1 розміром 6×3 буде зашифрований у «НІПАТОДЬМІ_ОШДГЛОУ».

Таблиця 3.1 – Рядки і стовпці

Н	А	Д	І	Ш	Л
І	Т	Ь	_	Д	О
П	О	М	О	Г	У

Таблична перестановка за рядками і стовпцям на основі ключа – небагато більш надійний спосіб шифрування з класу табличних перестановок. При такому способі шифрування узгоджується не тільки розмір таблиці, але і ключове слово, що у залежності від природного порядку його букв в алфавіті дає індексний вектор перестановок. Результатом шифрування вихідного тексту «НАДІШЛІТЬ_ДОПОМОГУ» із ключем «ТЕРМІН» (табл. 3.2, ліворуч) буде шифрований текст «АШІЛДНТД_ОЬЮГОУМП» (табл. 3.2, праворуч). Часто для підвищення криптостійкості таких методів вихідний текст послідовно перетасовують, використовуючи дві таблиці і два ключі, причому таблиці вибирають різного розміру.

Таблиця 3.2 – Рядки і стовпці на основі ключа

Т	Е	Р	М	І	Н
6	1	5	3	2	4
Н	А	Д	І	Ш	Л
І	Т	Ь	_	Д	О
П	О	М	О	Г	У

Е	І	М	Н	Р	Т
1	2	3	4	5	6
А	Ш	І	Л	Д	Н
Т	Д	_	О	Ь	І
О	Г	О	У	М	П

Таблична перестановка за нелінійним законом. Повідомлення записується в таблицю розмірності $m \times n$ послідовно рядками, а зчитування коду відбувається або «змійкою», або «за спіраллю», або за іншим зигзагоподібним шляхом.

Таблична перестановка за рядками і стовпцям із двома ключами припускає задіяти в перестановках не тільки стовпці, але і рядки. При цьому передаються два ключі (для рядків і стовпців).

Наприклад, текст «НАДІШЛІТЬ_ДОПОМОГУ» (табл. 3.3, ліворуч) із ключем «ТЕРМІН» по стовпцях і ключем «НАМ» по рядках буде зашифрований у «ТД_ОЫОГОУМПАШЛДН» (табл. 3.3, праворуч).

Таблиця 3.3 – Рядки і стовпці із двома ключами

		Т	Е	Р	М	І	Н
		6	1	5	3	2	4
Н	3	Н	А	Д	І	Ш	Л
А	1	І	Т	Ь	_	Д	О
М	2	П	О	М	О	Г	У

		Е	І	М	Н	Р	Т
		1	2	3	4	5	6
А	1	Т	Д	_	О	Ь	І
М	2	О	Г	О	У	М	П
Н	3	А	Ш	І	Л	Д	Н

При такому і подібному шифруванні не можна домогтися достатньої криптостійкості не від ручного, не від комп'ютерного дешифрування навіть при досить великих розмірах таблиць і ключів. Але такий шифр можна зробити істотно більш захищеним, виконавши шифрування з використанням перестановок кілька разів. Виявляється, що в цьому випадку застосовану для шифрування перестановку відтворити вже не так просто.

Завдання

Написати програму шифрування та розшифрування тексту криптографічним методом:

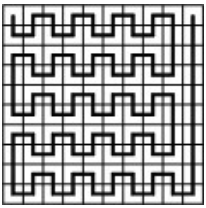
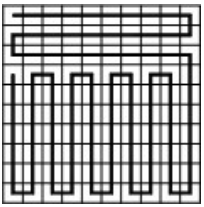
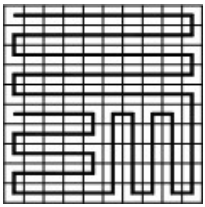
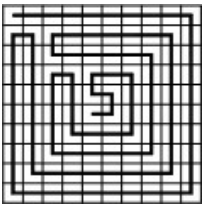
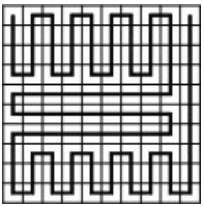
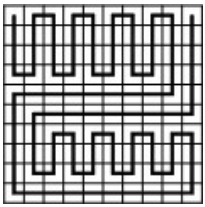
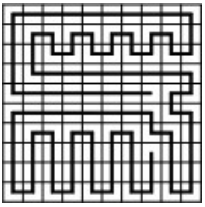
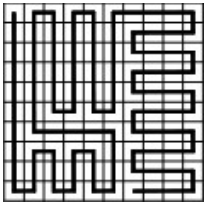
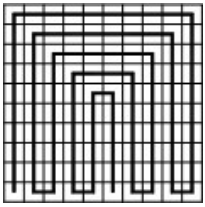
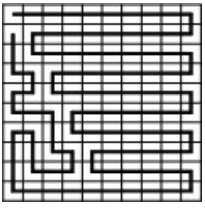
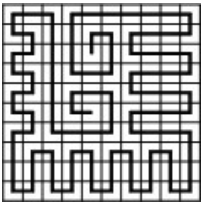
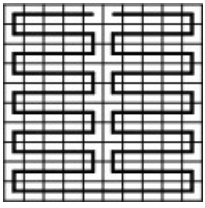
– для варіантів 1-5 – методом табличних перестановок згідно варіанту (табл. 3.4). Кількість стовпців повинна дорівнювати кількості букв прізвища, а рядків – імені;

– для варіантів 6-17 – методом шифрування за нелінійним законом, згідно табл. 3.5.

Таблиця 3.4 – Завдання для варіантів 1-5

№	Криптографічний метод	Примітки
1	метод «сходинок»	кількість сходинок дорівнює кількості букв прізвища
2	таблична перестановка за рядками і стовпцями	
3	таблична перестановка за рядками і стовпцями на основі ключа	в якості ключа потрібно взяти прізвище
4	таблична перестановка «змійкою»	
5	таблична перестановка «за спіраллю»	

Таблиця 3.5 – Завдання для варіантів 6-17

№	Схема	№	Схема	№	Схема
6		10		14	
7		11		15	
8		12		16	
9		13		17	

Контрольні запитання

1. Чим займається криптологія?
2. Дайте визначення поняттю шифрування і дешифрування.
3. У чому полягає відмінність симетричних криптографічних систем від систем з відкритим ключем?
4. Яка характеристика шифру визначає його стійкість?
5. Назвіть основні вимоги до сучасних криптографічних систем.
6. Назвіть типи криптографічних методів.
7. На які групи за стійкістю, діляться криптографічні алгоритми?
8. Що таке перестановка і таблична перестановка?
9. Які криптографічні методи табличних перестановок Ви знаєте?
10. В чому полягає метод «сходинок»?
11. В чому полягає таблична перестановка за рядками і стовпцями?
12. В чому полягає таблична перестановка за рядками і стовпцями на основі ключа?
13. В чому полягає таблична перестановка «змійкою», «за спіраллю» або за нелінійним законом?
14. В чому полягає таблична перестановка за рядками і стовпцями з двома ключами?

Робота №4

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ. ПІДСТАНОВКИ

Мета роботи: закріплення знань про криптографічні методи захисту інформації, про вимоги до криптографічних систем, про їх класифікацію, про різні типи криптографічних методів; придбання навичок практичної реалізації методів криптографічного перетворення, зокрема методів підстановок.

Стислі теоретичні відомості

Самим древнім і найпростішим з відомих шифрів підстановки є шифр, що використовувався Юлієм Цезарем для передачі шифрованих послань для управління військами. Він відноситься до групи моноалфавітних підстановок і називається *шифром Цезаря*. У шифрі кожна буква алфавіту заміняється буквою, що знаходиться на три позиції далі в цьому ж алфавіті. Слід звернути увагу, що алфавіт вважається «циклічним», потому після останньої букви йде перша.

Для української мови використовують звичайний алфавіт, але розширений двома знаками «’» (апостроф) і «_» (пробіл):

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

Наприклад, повідомлення «НАДІШЛІТЬ_ДОПОМОГУ» буде мати такий вигляд «РГЖКЮОКХ’ВЖСТСПСЕЦ».

Якщо вихідний текст позначити $(x_1, x_2, \ldots, x_m)$, а шифрований – $(y_1, y_2, \ldots, y_m)$, то шифр Цезаря можна представити правилом $y_i = x_{(i+3)} \bmod m$. У загальному випадку зміщення можна зробити на k позицій і правило в цьому випадку буде мати такий вигляд:

$$y_i = x_{(i+k)} \bmod m.$$

Якщо відомо, що певний текст був шифрований за допомогою шифру Цезаря, то за допомогою простого перебору усіх варіантів розкрити шифр дуже просто – для цього досить перевірити n (кількість букв в алфавіті) можливих варіантів тексту.

Моноалфавітні шифри. При наявності всього n можливих варіантів ключів шифр Цезаря не вважається надійним. Суттєвого розширення простору

ключів можна досягти використовуючи довільні підстановки. В цьому випадку кількість варіантів буде дорівнювати $n!$. Наприклад, для алфавіту української мови, кількість варіантів буде дорівнювати

$$35! = 10333147966386144929666651337523200000000.$$

Однак для криптоаналітика існує й інша лінія атаки. Якщо відома мова відкритого тексту, можна використовувати відому інформацію про характерні ознаки, властиві текстам відповідною мовою.

Моноалфавітні шифри легко розкриваються, тому що вони успадковують частотність уживання букв оригінального алфавіту. Контрзаходом у даному випадку є застосування для однієї букви не одного, а декількох замінників (називаних омофонами). Якщо число символів-замінників, призначених букві, вибрати пропорційним частоті появи цієї букви, то підрахунок частотності вживання букв у шифрованому тексті стає безглуздим. Але навіть при вживанні омофонів кожному елементу відкритого тексту відповідає тільки один елемент шифрованого тексту, тому в останньому, як і раніше, повинні спостерігатися характерні показники частоти повторення комбінацій декількох букв (наприклад, біграм), і в результаті задача криптоаналізу залишається досить елементарною.

Одним з найбільш відомих шифрів, що базуються на заміщенні комбінацій букв, є шифр Плейфеєра, у якому біграми відкритого тексту розглядаються як самостійні одиниці, перетворені в задані біграми шифрованого тексту. Цей шифр було розглянуто в лабораторної роботі №1.

Ще одним цікавим шифром, що базується на заміщенні комбінацій букв, є шифр Хілла. За алгоритмом шифру кожні m послідовні букви відкритого тексту замінюються m буквами шифрованого тексту. Підстановка визначається m лінійними рівняннями. Наприклад, при $m = 3$, одержуємо наступну систему рівнянь:

$$\begin{aligned} y_1 &= (k_{11}x_1 + k_{12}x_2 + k_{13}x_3) \bmod n, \\ y_2 &= (k_{21}x_1 + k_{22}x_2 + k_{23}x_3) \bmod n, \\ y_3 &= (k_{31}x_1 + k_{32}x_2 + k_{33}x_3) \bmod n. \end{aligned}$$

де x_i і y_i – букви відповідно вихідного та шифрованого тексту, матриця K – це матриця розмірності 3×3, що представляє ключ шифрування.

Зашифруємо текст «ПОВІДОМЛЕННЯ» за допомогою ключа «КЛЮЧ» ($m = 2$). Запишемо текст та ключ у числовому виді (починаючи з 0):

вихідний текст «19 18 2 11 5 18 16 15 6 17 17 32»,

ключ $\begin{pmatrix} 14 & 15 \\ 31 & 27 \end{pmatrix}$.

Перші дві букви зашифрованого тексту будуть:

$$(14 \cdot 19 + 15 \cdot 18) \bmod 35 = 11,$$
$$(31 \cdot 19 + 27 \cdot 18) \bmod 35 = 25.$$

Аналогічно знайдемо інші пари букв і отримаємо зашифроване повідомлення «ІХОЗХІЩЦФЛОЦ».

Для розшифровки потрібно скористатися матрицею, зворотної до K. Зворотна існує не для всякої матриці, тому в такому випадку слід взяти інший ключ.

Для визначення зворотної по модулю n матриці K^{-1} ключа потрібно знайти його визначник Δ, а потім залишок від ділення на n. У розглянутому прикладі це буде:

$$(14 \cdot 27 - 15 \cdot 31) \bmod 35 = -87 \bmod 35 = 18.$$

Якщо визначник матриці дорівнює 0, то необхідно взяти інший ключ. Інший ключ треба взяти і в тому випадку, коли $n \bmod \Delta = 0$.

Кожний елемент зворотної матриці $\overline{\mathbf{K}}^{-1}$ буде знаходитись за формулою:

$$\bar{k}_{ij}^{-1} = \frac{(-1)^{i+j}}{\Delta} \cdot \begin{cases} \Delta_{ij}, & i = j; \\ \Delta_{ji}, & i \neq j, \end{cases}$$

де Δ_{ij} – визначник матриці, отриманої видаленням i-го рядка та j-го стовпця.

Для розглянутого приклада $\overline{\mathbf{K}}^{-1}$ матриця буде:

$$\begin{pmatrix} 27/18 & -15/18 \\ -31/18 & 14/18 \end{pmatrix}.$$

У зв'язку з тим, що нам потрібна зворотна по модулю n матриця K^{-1}, то потрібно перетворити кожен елемент матриці $\overline{\mathbf{K}}^{-1}$ таким чином, щоб він був цілим та меншим за n.

Для цього можна скористатись наступним алгоритмом:

```
p = 0
поки ((n*p + k̄_ij^-1 Δ) mod Δ ≠ 0) і (p ≤ nmax)
     збільшити p на 1
якщо p ≤ nmax
     то      k_ij^-1 = ((n*p + k̄_ij^-1 Δ) div Δ) mod n
     інакше 'візьміть інший ключ'
```

Зворотна по модулю 35 матриця K^{-1} ключа для розглянутого прикладу буде дорівнювати: $\begin{pmatrix} 19 & 5 \\ 8 & 28 \end{pmatrix}$

Як і у випадку шифру Плейфеєра, перевага шифру Хілла полягає в тому, що він цілком маскує частоту входження окремих букв. А для шифру Хілла чим більше розмір ключа, тим більше в шифрованому тексті ховається інформації про розходження в значеннях частоти появи інших комбінацій символів. Так, шифр Хілла, з матрицею 3×3 ховає частоту появи не тільки окремих букв, але і двобуквених комбінацій.

Недоліком шифру Хілла є те, що цей шифр легко розкрити при наявності відомого відкритого тексту.

Шифрування за допомогою біграм. Маються дві таблиці з випадково розміщеними на них алфавітами. Для шифрування, вихідний текст розбивається на пари букв (біграми), наприклад «ПО ВІ ДО МЛ ЕН НЯ». Перша буква біграми шукається в першій таблиці, друга в другій. Умовно між цими протилежними вершинами будується прямокутник. Одна діагональ такого прямокутника з'єднує пари букв вихідної біграми, а друга діагональ дає пару букв шифрованої біграми. Наприклад, букви вихідного тексту «ДО» будуть зашифровані як «ТЯ» на основі табл. 4.1. Треба відмітити, що шифрування і дешифрування відбувається за одним алгоритмом, що спрощує як програмну, так і апаратну реалізацію такого методу.

Многоалфавітні підстановки. Слабку криптостійкість моноалфавітних підстановок усувають многоалфавітними підстановками, що перетворюють вихідний текст на підставі ключа у шифрований за допомогою узагальненої підстановки Цезаря.

Таблиця 4.1 – Шифрування за допомогою біграм

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

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

Пояснимо на прикладі даний метод шифрування. За ключ візьмемо текст «ТЕРМІНПОВІДОМЛЕННЯ» такої ж довжини, як і вихідний текст. Зашифруємо за його допомогою текст «НАДІШЛІТЬ_ДОПОМОГУ» (табл. 4.2-4.3).

Таблиця 4.2 – Підготовка до шифрування

А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М	Н
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я	’	_	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	

Таблиця 4.3 – Шифрування многоалфавітною підстановкою

17	0	5	11	28	15	11	22	30	34	5	18	19	18	16	18	3	23
22	6	20	16	11	17	19	18	2	11	5	18	16	15	6	16	16	32
5	6	25	27	5	32	30	6	32	11	10	2	1	33	22	0	19	21

Зашифроване повідомлення буде таким: «ДЕХЧДЯЬЕЯІИВБ'ТАПС».

Для такої системи підстановки використовують також термін «одноразова стрічка» та «одноразовий блокнот». Кількість можливих криптографічних перетворень вихідного тексту многоалфавітної підстановки дорівнює m^n, де m – довжина тексту, n – кількість символів в алфавіті.

Шифрування накладенням непередбаченого ключа на вихідний текст повідомлення робить останній таким, яке не розкривається за умови, що довжина ключа не менше довжини самого тексту. Таке накладення нескінченного ключа на вихідний текст радикально змінює статистичні характеристики вихід-

ного тексту. Системи одноразового використання теоретично неможливо розшифрувати, тому що вони не містять достатньої інформації для відновлення вихідного тексту.

Але ці системи не застосовуються для забезпечення таємності при обробці інформації в сучасних автоматизованих системах, оскільки вони вимагають незалежного вибору значень ключа для кожної букви вихідного тексту, а значить і передачі величезної кількості ключової інформації каналами зв'язку.

Система шифрування Віжинера. На відміну від многоалфавітної підстановки, в алгоритмі шифрування Віжинера ключ має кінцеву довжину. Таким чином, відсутня вимога шифрувати кожну букву вихідного тексту окремим значенням ключа. Візьмемо ключ кінцевої довжини і продовжимо його до нескінченної послідовності, повторюючи ланцюжок.

Зашифруємо попереднє повідомлення за допомогою ключа «ТЕРМІН» (табл. 4.4).

Таблиця 4.4 – Шифрування повідомлення

17	0	5	11	28	15	11	22	30	34	5	18	19	18	16	18	3	23
22	6	20	16	11	17	22	6	20	16	11	17	22	6	20	16	11	17
5	6	25	27	5	32	33	28	16	16	16	1	7	24	2	0	14	6

Шифрований текст: «ДЕХЧДЯ’ШМММБЄФВАКЕ».

Для попереднього випадку при ключі кінцевої довжини «ТЕРМІН», ключ для многоалфавітної підстановки буде «ТЕРМІННАДІШЛІТЬ_ДО» (табл. 4.5).

Таблиця 4.5 – Шифрування з ключем «ТЕРМІННАДІШЛІТЬ_ДО»

17	0	5	11	28	15	11	22	30	34	5	18	19	18	16	18	3	23
22	6	20	16	11	17	22	6	20	16	11	17	22	6	20	16	11	17
5	6	25	27	5	32	33	28	16	16	16	1	7	24	2	0	14	6

Шифрований текст: «ДЕХЧДЯ’ШМММБЄФВАКЕ».

Віжинер запропонував для многоалфавітної підстановки використовувати ключ, який є результатом конкатенації ключа кінцевої довжини з відкритим текстом, який отримав назву системи з автоматичним вибором ключа.

Варто визнати, що і многоалфавітні підстановки доступні криптоаналітичному розкриттю. Їх криптостійкість різко убуває зі зменшенням довжини ключа. Проте шифрування підстановкою Віжинера допускає нескладну апаратну чи програмну реалізацію, а при досить великій довжині ключа може бути використане в сучасних автоматизованих системах.

Завдання

Написати програму шифрування та розшифрування тексту криптографічним методом підстановок згідно варіанту (табл. 4.6). В якості вихідного тексту взяти реферат з першої роботи. У варіантах, де не накладається обмеження на розмір ключа, в якості ключа взяти прізвище.

Таблиця 4.6 – Варіанти методів підстановок

№	Криптографічний метод	№	Криптографічний метод
1	координатна моноалфавітна підстановка	7	многоалфавітна підстановка
2	шифрування за допомогою біграм	8	система шифрування Віжинера
3	многоалфавітна підстановка	9	координатна моноалфавітна підстановка
4	система шифрування Віжинера	10	шифрування за допомогою біграм
5	координатна моноалфавітна підстановка	11	многоалфавітна підстановка
6	шифрування за допомогою біграм	12	система шифрування Віжинера

Контрольні запитання

1. Чим відрізняються криптографія від криптоаналізу?
2. Що таке ключ і простір ключів?
3. Назвіть типи криптографічних методів.
4. На які групи, з точки зору стійкості, діляться криптографічні алгоритми?

5. Чим відрізняються криптографічні методи перестановок від підстановок?

6. Чому моноалфавітні шифри вважаються ненадійними?

7. У чому полягає особливість шифру Хілла?

8. Навіщо при реалізації алгоритму Хілла потрібна зворотна матриця ключа?

9. Чим відрізняється зворотна матриця ключа, отримана при розшифруванні методом Хілла, від звичайної зворотної матриці?

10. Який недолік шифру Хілла Ви знаєте і яким чином ним можна скористатись?

11. Які є різновиди координатних моноалфавітних підстановок?

12. У чому полягає відмінність багатосимвольних підстановок від односимвольних?

Робота №5

ВИВЧЕННЯ РОБОТИ АСИМЕТРИЧНИХ ШИФРІВ. ГЕНЕРАЦІЯ ПРОСТОГО ВЕЛИКОГО ЧИСЛА. ЗВЕДЕННЯ ПРОСТОГО ВЕЛИКОГО ЧИСЛА В ЦІЛУ СТЕПІНЬ ПО МОДУЛЮ N

Мета роботи: закріплення знань про системи шифрування з відкритим ключем, аутентифікацію, умови існування алгоритмів з двома ключами, односторонні функції та функції-пастки; ознайомлення з основними визначеннями теорії чисел; придбання навичок асиметричного шифрування, знаходження великих простих чисел за допомогою ймовірного алгоритму, знаходження найбільшого загального дільника.

Стислі теоретичні відомості

Будь-яка криптосистема заснована на використанні ключів. Найпоширенішим алгоритмом асиметричного шифрування є RSA (автори Rivest, Shamir та Alderman). RSA – це система з відкритим ключем призначена як для шифрування, так і для аутентифікації, яка була розроблена в 1977 році. Вона основана на складності розкладу дуже великих цілих чисел на прості співмножники.

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

Існують різні імовірнісні перевірки на простоту чисел, що визначають. чи є число простим із заданим степенем достовірності.

Одним з найбільш розповсюджених є алгоритм, розроблений Рабіном за ідеями Міллера. Тест Міллера-Рабіна визначення простого числа є комбінацією тестів Ферма і квадратного кореня. Він знаходить сильне псевдопросте число (просте число з дуже високою ймовірністю).

Теорема Ферма стверджує, що якщо n – просте число та a – будь-яке ціле число, $1 \le a \le n-1$, тоді $a^{n-1} \equiv 1 (\mathrm{mod}\, n)$.

Отже, якщо n – ціле число, чия простота є під питанням, знаходження a в інтервалі, де ця рівність не виконується, докаже, що n – складене.

Розглянемо, у чому полягає тест Міллера-Рабіна.

Нехай $n > 0$ – непарне ціле. Виберемо ціле число a, яке задовольняє нерівності $1 \le a \le n-1$, і назвемо його основою. Оскільки n – непарне, $n - 1$ повинно бути парним числом. Перший крок тесту полягає у визначенні такого показника $m \ge 1$, для якого $n - 1 = 2^k m$, де m не ділиться на 2. Таким чином, можна записати $n - 1$ як добуток непарного числа m і степеня числа 2.

Далі тест обчислює відрахування по модулю n у такій послідовності степенів: $a^m, a^{m^2}, a^{m^4}, ..., a^{m^{2^{k-1}}} \pmod n$.

Розберемося, якими властивостями володіє ця послідовність в разі простого n. Отже, поки не буде обумовлено, n – просте число. Теорема Ферма говорить, що $a^{n-1} = 1 (\text{mod } n)$. Значить, якщо n – просте, то останнє відрахування в послідовності завжди дорівнює 1.

У тесті Міллера-Рабина використовується критерій, заснований на факті, що для простого модуля квадратними коренями з одиниці є лише числа ± 1, а для складеного непарного модуля кількість таких коренів більше двох. Якщо n – просте, повинно бути отримане $x \equiv 1 (\text{mod } n)$ або $x \equiv -1 (\text{mod } n)$, що випливає із малої теореми Ферма:

$$x^2 = x \cdot x = (a^{(n-1)/2})(a^{(n-1)/2}) = a^{n-1} \equiv 1 (\text{mod } n)$$

та з твердження 1.

Твердження 1. Нехай n – просте та $x^2 \equiv 1 (\text{mod } n)$. Тоді $x \equiv 1 (\text{mod } n)$ або $x \equiv -1 (\text{mod } n)$.

Це твердження говорить про те, що квадратні корні з 1 за модулем простого числа дорівнюють тільки 1 або –1.

Під час обчислення квадратних коренів з одиниці по простому модулю завжди буде або одиниця, або корінь, що дорівнює $-1 (\text{mod } n)$.

Це означає, що в ряду відрахувань чисел $a^m, a^{m^2}, a^{m^4}, ..., a^{m^{2^{k-1}}} \pmod n$ по простому модулю n або з'явиться $-1 (\text{mod } n)$ (зверніть увагу, що в модульній

арифметиці –1 означає $n - 1$), або всіх їх можна порівняти з одиницею, що можливо, коли $a^m = 1 (\mathrm{mod}\ n)$.

Якщо n – складене, то можливі і інші випадки.

Розглянемо алгоритм тесту Міллера-Рабина:

Ініціалізація

Вибирають основу і обчислюють $T = a^m \bmod n$, де $m = (n - 1)/2^k$.

1. Якщо T дорівнює +1 або –1, оголошують, що n – сильне псевдопросте число і процес зупиняється. Кажуть, що n пройшов два випробування: тест Ферма і випробування квадратним коренем. Оскільки, якщо T дорівнює ± 1, то T стане рівним 1 на наступному кроці і залишається 1 до проходження тесту Ферма. Крім того, T пройшов випробування тестом квадратного кореня, тому що T і квадратний корінь були б рівними 1 на наступному кроці і T дорівнював би ± 1 на цьому кроці.

2. Якщо T дорівнює іншому значенню, немає впевненості чи є n простим числом, чи складеним об'єктом, а значить процес буде продовжуватися на наступному кроці.

Крок 1

Обчислюють T^2.

1. Якщо результат дорівнює +1, відомо, що тест Ферма пройдено, бо T залишається 1 для подальших випробувань. Випробування квадратним коренем, проте, не пройдено. Оскільки T дорівнює 1 на цьому кроці і на попередньому кроці мало інше значення, ніж ± 1 (причина, з якої алгоритм не зупинився на попередньому кроці), n оголошують складеним об'єктом, і процес зупиняється.

2. Якщо результат дорівнює –1, відомо, що для n в кінцевому рахунку відбудеться тест Ферма. Також він пройде випробування квадратним коренем, оскільки T дорівнює –1 в цьому кроці і стане 1 на наступному кроці. Число n оголошують сильним псевдовипадковим простим числом і зупиняють процес.

3. Якщо T має ще якесь значення, то не зрозуміло чи є воно простим числом, і процес триває на наступному кроці.

Крок 2

Цей крок виконується у циклі до $k - 1$. В ньому виконуються ті ж самі дії, що в кроці 1.

Цей крок не є необхідним. Якщо його досягнуто і не прийнято рішення, в ньому немає сенсу. Якщо результат цього кроку –1, значить, тест Ферма пройдено, але оскільки результат попереднього кроку не ± 1, то випробування квадратного кореня не пройдене. Якщо після кроку $k - 1$ процес не зупинено, оголошується, що n – складене.

Знаходження значення виразу $a^b \bmod n$

Один з кроків алгоритм RSA є обчислення виразу $a^b \bmod n$, при цьому значення a і b мають великі значення. Обчислення такого виразу «в лоб» викликає складності навіть у сучасних комп'ютерах. Тому для обчислення можна застосувати наступний алгоритм.

В загальному випадку припустимо, що треба знайти значення a^b, де a та b – додатні цілі числа. Якщо представити b у вигляді бінарного числа $b_k b_{k-1} \ldots b_0$, то

$$m = \sum_{b_i \neq 0} 2^i$$

і тому

$$a^m = a^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} a^{2^i},$$

$$a^m \bmod n = \left[\prod_{b_i \neq 0} a^{2^i} \right] \bmod n = \prod_{b_i \neq 0} \left[a^{2^i} \bmod n \right].$$

Таким чином, для знаходження $a^k \bmod n$ необхідно застосувати алгоритм, що приведено нижче:

```
d = 1
для i від k до 0 крок –1 робити
пц
      d = d^2 mod n
      якщо b_i = 1 то d = (d*a) mod n
кц
степінь за модулем дорівнює d
```

В табл. 5.1 приведено приклад послідовності значень, що одержані в результаті виконання цього алгоритму. Такий алгоритм називається швидким зведенням до степеню для $a^b \bmod n$.

Дані в таблиці наведені для $a = 7$, $k = 560 = 1000110000$, $n = 561$.

Таблиця 5.1 – Результат роботи алгоритму

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
d	7	49	157	526	160	241	298	166	67	1

Генерація простого великого числа

1. Згенерувати випадкове 16-бітове число a.

2. Встановити його старший і молодший біти рівними 1. Старший біт гарантуватиме необхідну довжину шуканого числа, а молодший біт забезпечує його непарність.

3. Переконатися, що a не ділиться на невеликі прості числа: 3, 5, 7, 11 тощо. Найбільш ефективною є перевірка на подільність для всіх простих чисел, менших 2000.

4. Виконати тест Міллера-Рабіна мінімум 3 рази.

5. Якщо a не пройшло хоча б одну перевірку з п. 3 або 4, воно не є простим. В такому випадку треба змінити вихідне число на –2 або на +2. Повторити п. 3, 4. Виконати п. 5, поки не буде знайдено просте число.

Перевірка, що випадкове непарне n не ділиться на 3, 5 і 7 відсікає 54% непарних чисел. Перевірка подільності на всі прості числа, менші 256 відсікає 80% складених непарних чисел. Навіть, якщо складене число «просочилося» через цей алгоритм, це буде відразу ж помічено, оскільки шифрування і дешифрування не працюватимуть.

6. Повторити п. 1-5 для випадкового 8-бітового числа b.

Завдання

Створити додаток, який:

– генерує прості числа з перевіркою тестом Міллера-Рабіна;

– реалізує алгоритм зведення простого великого числа a в цілу ступінь b по модулю n.

Контрольні запитання

1. Наведіть загальну схему процесу шифрування з відкритим ключем.
2. Які мають назви ключі в симетричному та асиметричному шифруванні?
3. Який недолік у систем шифрування з відкритим ключем?
4. Чим відрізняється шифрування з відкритим ключем від аутентифікації?
5. Яка функція називається функцією-пасткою?
6. Що таке взяття числа за модулем?
7. Чим відрізняється шифрування з відкритим ключем від симетричного шифрування?
8. У чому полягає тест Міллера-Рабіна?
9. Що таке сильне псевдопросте число?

Робота №6

СТВОРЕННЯ ЦИФРОВОГО ПІДПИСУ

Мета роботи: освоїти процес створення цифрового підпису.

Стислі теоретичні відомості

Для вирішення завдання аутентифікації інформації сьогодні використовується концепція електронної цифрового підпису (ЕЦП). ЕЦП – це набір методів, які дозволяють перенести властивості рукописного підпису під документом в область електронного документообігу.

Реальний підпис має такі властивості:

1. Достовірність, яка стверджує, що користувач свідомо підписав документ.

2. Непідробність, що доводить, що конкретний користувач, а ніхто інший за нього підписав документ.

Такими ж властивостями повинен володіти електронний цифровий підпис, для чого в його основу покладені криптографічні методи.

Для створення ЕЦП можна застосовувати асиметричний шифр RSA.

Першим етапом цього алгоритму є створення пари ключів: відкритого і особистого, а також розповсюдження відкритого ключа. Для алгоритму RSA етап створення ключів складається з наступних операцій:

1. Вибираються два простих числа p і q.

2. Обчислюється їх добуток $n = p \cdot q$ та $f = (p - 1)(q - 1)$.

3. Вибирається довільне число e менше n, таке, що НЗД$(e, f) = 1$, тобто e повинно бути взаємно простим з числом f.

4. Знаходиться таке *ціле* число d, що $(e \cdot d) \bmod f = 1$.

5. Два числа (e, n) – публікуються як відкритий ключ.

6. Число d зберігається в найсуворішій таємниці – це і є особистий ключ, що дозволить читати всі послання, зашифровані за допомогою пари чисел (e, n).

Для знаходження чисел *e* і *d* (п. 3-4) можна скористатись розширеним алгоритмом Евкліда з деякими змінами (див. нижче). Для цього використовується циклічна схема:

```
вибрати довільне e менше n
поки РАлгЕвкліда(e,f,mcd,d) або (mcd ≠ 1) робити
     зменшити або збільшити e на 1
```

Логічна функція РАлгЕвкліда повертає істину, якщо для заданого *e* існує *d*. Крім того, функція повертає значення *d* та найбільшого загального дільника mcd. Схема функції наведена нижче:

```
РАлгЕвкліда = Істина
X1 = 1, X2 = 0, X3 = f
Y1 = 0, Y2 = 1, Y3 = e
поки Y3 ≠ 1 робити
пц
     якщо Y3 = 0 то
          mcd = X3, РАлгЕвкліда = Хибність
          повернути результат
     все
     Q = [X3/Y3]         // ціла частина від ділення
     T1 = X1 - Q*Y1, T2 = X2 - Q*Y2, T3 = X3 - Q*Y3
     X1 = Y1, X2 = Y2, X3 = Y3
     Y1 = T1, Y2 = T2, Y3 = T3
кц
mcd = Y3, d = Y2
якщо Y2 < 0 то збільшити d на f
повернути результат
```

На прийомній стороні процес дешифрування можливий за допомогою особистого числа *d*. Для того щоб прочитати повідомлення $c_i = m_i^e \bmod n$ досить звести його до степеня *d* за модулем *n*: $c_i^d \bmod n = m_i^{e \cdot d} \bmod n = m_i$.

Операції зведення до степеню великих чисел досить трудомісткі для сучасних процесорів, навіть за алгоритмами, що оптимізовані. Тому зазвичай весь текст повідомлення кодується звичайним блоковим шифром, але з використанням ключа сеансу, а от сам ключ сеансу шифрується асиметричним алгоритмом за допомогою відкритого ключа одержувача і розташовується в початку шифрованого повідомлення.

Два числа p і q, добутком яких є модуль n, повинні мати приблизно однакову довжину оскільки в цьому випадку знайти співмножники складніше, ніж у випадку коли довжина чисел значно розрізняється.

Оптимальний розмір модуля n визначається вимогами безпеки: модуль більшого розміру забезпечує велику безпеку, але й сповільнює роботу алгоритму RSA.

Процедура формування електронного підпису *sign* під повідомленням (рис. 6.1) схожа з шифруванням документу, але в степінь закритого ключа d по відрахуванню n зводиться не саме повідомлення або його частини, а дайджест повідомлення h. Отже, невід'ємною частиною алгоритмів ЕЦП є хешування інформації, на рисунку воно позначено через H.

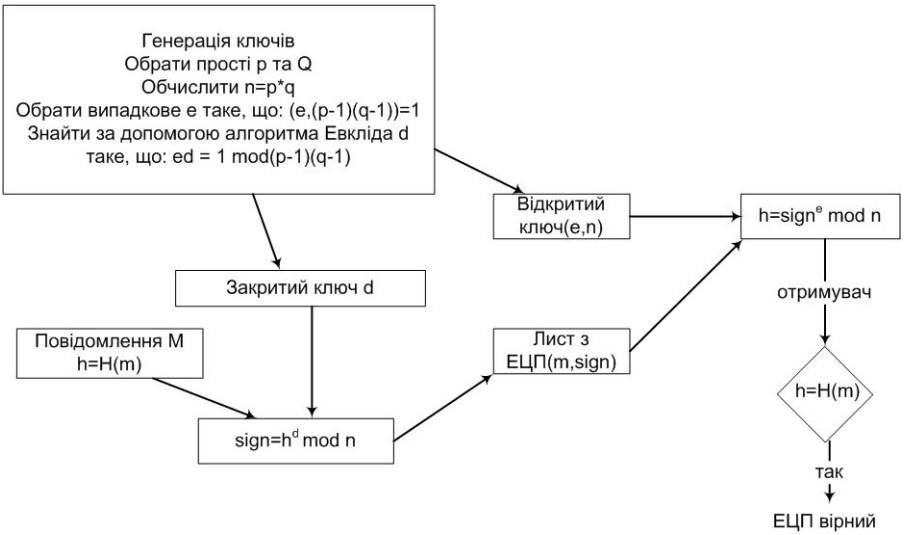


Рисунок 6.1 – Процедура формування електронного підпису sign

Для реалізації криптографічних методів застосовують односторонні функції шифрування, які мають назву хеш-функції. Основне призначення подібних функцій – отримання згортання довільного розміру його дайджесту – значення фіксованого розміру.

Завдання

Написати програму для створення цифрового підпису на основі алгоритму RSA.

Для виконання завдання потрібно скористуватись наступною послідовністю дій:

1. Знайти значення хеш-функції від свого прізвища та імені. Значенням хеш-функції буде результат виконання операції `xor` між літерами прізвища і імені в кодуванні ASCII.
2. Сформувати цифровий підпис.
3. Надіслати лабораторну роботу з прикріпленими цифровим підписом (без інших розпізнавальних знаків - прізвища, імені, фотографії) і своїм відкритим ключем на електронну адресу викладача з поштової скриньки групи (якщо такого ящика не існує, то створити).

Контрольні запитання

1. Дайте визначення електронного цифрового підпису.
2. Дайте визначення хеш-функції.
3. У чому полягає відмінність ЕЦП від хеш-функції?
4. Які з систем з відкритим ключем використовується для генерації цифрового підпису?
5. Які складні для вирішення завдання використовуються для підвищення стійкості алгоритму RSA
6. Що таке одностороння хеш-функція?
7. Яке складне для вирішення завдання лежить в основі знаходження дискретного логарифма?

Робота №7

ОБФУСКАЦІЯ

Мета роботи: освоїти процес ускладнення аналізу програм за рахунок обфускації.

Стислі теоретичні відомості

Обфускація – приведення виконуваного коду до виду, що зберігає функціональність програми, але ускладнює аналіз і розуміння алгоритмів роботи.

Методи обфускації:

1. *Лексична обфускація*. Даний вид обфускації полягає у форматуванні коду програми, змінюванні його структури, таким чином, щоб він став нечитабельним, менш інформативним і важким для вивчення.

Лексична обфускація включає в себе:

– видалення всіх коментарів в коді програми, або зміна їх на дезінформуючі;

– видалення різних пробілів, відступів, які зазвичай використовують для кращого візуального сприйняття коду програми;

– заміну імен ідентифікаторів (імен змінних, функцій тощо), на довільні довгі набори символів;

– додавання різних зайвих (сміттєвих) операцій;

– зміна розташування блоків (функцій, процедур).

2. *Обчислювальна обфускація*. Зміни стосуються головної структури потоку управління. До них можна віднести:

– розширення умов циклів;

– додавання недосяжного коду;

– додавання надлишкових операцій.

Завдання

1. Вибрати код програми за варіантом (варіанти в кінці лабораторної роботи в табл. 7.1).

2. Скомпілювати програму, перевірити її працездатність.

2.1. Запустити MS Visual Studio.

2.2. Створити проект Visual C++/Win32/Консольний додаток Win32 з ім'ям «test».

2.3. Вставити код програми, отриманий в п. 1, в «test.cpp».

2.4. Натиснути F5 для компіляції і запуску програми.

3. Здійснити лексичну обфускацію.

3.1. Замінити назви змінних на такі, що не несуть сенсу.

До

```
#include <stdafx.h>
#include <stdio.h>
// головна функція
int main()
{
  // номер варіанту
  int q = 1;
  switch( q )
  {
    // пусто (варіант 0)
    case 0:
      printf("null\n");
      break;
    // перший варіант
    case 1:
      for (int i=0; i<=1; i++)
        printf("variant 1\n");
      break;
    // другий варіант
    case 2:
      printf("variant 2\n");
      break;
    // інакші варіанти
    default:
      printf("other variant\n");
  }
  return 0;
}
```

Після

```
#include <stdafx.h>
#include <stdio.h>
// головна функція
int main()
{
  // номер варіанту
  int xX03123012 = 1;
  switch( xX03123012 )
  {
    // пусто (варіант 0)
    case 0:
      printf("null\n");
      break;
    // перший варіант
    case 1:
      for (int weq83efas_23=0;
           weq83efas_23<=1; weq83efas_23++)
        printf("variant 1\n");
      break;
    // другий варіант
    case 2:
      printf("variant 2\n");
      break;
    // інакші варіанти
    default:
      printf("other variant\n");
  }
  return 0;
}
```

Коментарі до коду:

– змінено назву змінної `q` на `xX03123012`;
– змінено назву змінної `i` на `weq83efas_23`.

3.2. Використовувати шістнадцяткове представлення для деяких чисел.

До

```
#include <stdafx.h>
#include <stdio.h>
// головна функція
int main()
{
  // номер варіанту
  int xX03123012 = 1;
  switch( xX03123012 )
  {
    // пусто (варіант 0)
    case 0:
```

Після

```
#include <stdafx.h>
#include <stdio.h>
// головна функція
int main()
{
  // номер варіанту
  int xX03123012 = 0x26d2+60-0x270d;
  switch( xX03123012 )
  {
    // пусто (варіант 0)
    case 0x36d-877:
```

```
    printf("null\n");
    break;
  // перший варіант
  case 1:
    for (int weq83efas_23=0;
         weq83efas_23<=1; weq83efas_23++)
      printf("variant 1\n");
    break;
  // другий варіант
  case 2:
    printf("variant 2\n");
    break;
  // інакші варіанти
  default:
    printf("other variant\n");
 }
 return 0;
}
```

```
    printf("null\n");
    break;
  // перший варіант
  case -3550+0xddf:
    for (int weq83efas_23=0;
         weq83efas_23<=1; weq83efas_23++)
      printf("variant 1\n");
    break;
  // другий варіант
  case 0xca4-3233+1:
    printf("variant 2\n");
    break;
  // інакші варіанти
  default:
    printf("other variant\n");
 }
 return xX03123012-1;
}
```

Коментарі до коду:

– застосовано шістнадцяткове представлення чисел, замість десяткового (наприклад, `0x26d2+60-0x270d=9938+60-9997=1`, `2=0xca4-3233+1`).

3.3. Видалити пробіли і відступи та замінити коментарі на такі, що дезінформують.

До

```
#include <stdafx.h>
#include <stdio.h>
// головна функція
int main()
{
  // номер варіанту
  int xX03123012 = 0x26d2+60-0x270d;
  switch( xX03123012 )
  {
    // пусто (варіант 0)
    case 0x36d-877:
      printf("null\n");
      break;
    // перший варіант
    case -3550+0xddf:
      for (int weq83efas_23=0;
           weq83efas_23<=1; weq83efas_23++)
        printf("variant 1\n");
      break;
    // другий варіант
    case 0xca4-3233+1:
      printf("variant 2\n");
      break;
    // інакші варіанти
    default:
      printf("other variant\n");
  }
  return xX03123012-1;
}
```

Після

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0;weq83efas_23<=1;weq8
3efas_23++)printf("variant 1\n");break;/*м
ій голос*/case 0xca4-3233+1:printf("varian
t 2\n");break;/*інші голоса*/default:print
f("other variant\n");}return xX03123012-1;
}
```

3.4. Скомпілювати програму клавішею F5 і перевірити її функціональність. При появі помилок необхідно відновити код згідно варіанту і повторити лексичну обфускацію.

3.5. Зробити резервну копію коду програми (наприклад, в текстовому файлі).

4. Здійснити обчислювальну обфускацію.

4.1. Розширити умови циклів.

До

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0;weq83efas_23<=1;weq8
3efas_23++)printf("variant 1\n");break;/*м
ій голос*/case 0xca4-3233+1:printf("varian
t 2\n");break;/*інші голоса*/default:print
f("other variant\n");}return xX03123012-1;
}
```

Після

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0,j33floor1==0x2;weq83
efas_23<=1 && j33floor1==0x2;weq83efas_23+
+)printf("variant 1\n");break;/*мій голос*
/case 0xca4-3233+1:printf("variant 2\n");b
reak;/*інші голоса*/default:printf("other
variant\n");}return xX03123012-1;}
```

Коментарі до коду:

– до циклу `for` була включена змінна `j33floor1`, значення якої завжди дорівнює двом, тобто, додана умова `j33floor1==0x2` буде завжди виконуватися.

4.2. Додати надлишкові операції.

До

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0,j33floor1==0x2;weq83
efas_23<=1 && j33floor1==0x2;weq83efas_23+
+)printf("variant 1\n");break;/*мій голос*
/case 0xca4-3233+1:printf("variant 2\n");b
reak;/*інші голоса*/default:printf("other
variant\n");}return xX03123012-1;}
```

Після

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0,j33floor1==0x2;weq83
efas_23<=1 && j33floor1==0x2;weq83efas_23+
+){j33floor1=j33floor1*(21+0x4)/(15+0xa);p
rintf("variant 1\n");}break;/*мій голос*/c
ase 0xca4-3233+1:printf("variant 2\n");bre
ak;/*інші голоса*/default:printf("other va
riant\n");}return xX03123012-1;}
```

Коментарі до коду:

– додана надлишкова операція `j33floor1=j33floor1*(21+0x4)/(15+0xa)` (ця операція не змінює значення `j33floor1`, тому, що множник завжди дорівнює одиниці).

4.3. Додати недосяжний код.

До

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0,j33floor1==0x2;weq83
efas_23<=1 && j33floor1==0x2;weq83efas_23+
+){j33floor1=j33floor1*(21+0x4)/(15+0xa);p
rintf("variant 1\n");}break;/*мій голос*/c
ase 0xca4-3233+1:printf("variant 2\n");bre
```

Після

```
// модуль розпізнавання мови
#include <stdafx.h>
#include <stdio.h>
int main(){/*номер запису*/int xX03123012=
0x26d2+60-0x270d;switch(xX03123012){/*голо
с шпигуна*/case 0x36d-877:printf("null\n")
;break;/*голос офіцера*/case -3550+0xddf:f
or(int weq83efas_23=0,j33floor1==0x2;weq83
efas_23<=1 && j33floor1==0x2;weq83efas_23+
+){j33floor1=j33floor1*(21+0x4)/(15+0xa);i
f(j33floor1==0x13-2)weq83efas_23++;printf(
"variant 1\n");}break;/*мій голос*/case 0x
```

```
ak;/*інші голоса*/default:printf("other va
riant\n");}return xX03123012-1;}
```

```
ca4-3233+1:printf("variant 2\n");break;/*і
нші голоса*/default:printf("other variant\
n");}return xX03123012-1;}
```

Коментарі до коду:

– додано недосяжний код до циклу

```
if (j33floor1==0x13-2) weq83efas_23++
```

(умова ніколи не виконається, оскільки в даній програмі `j33floor1` завжди дорівнює 2).

4.4. Скомпілювати програму клавішею F5 і перевірити її функціональність. При появі помилок необхідно відновити код з резервної копії і повторити обчислювальну обфускацію.

5. Зберегти проект.

6. Помінятися кодом після обсфускації з одногрупником.

7. Спробувати розібратися в коді програми, отриманому від одногрупника, і привести код до виду, близького до початкового (під вихідним видом розуміється читабельний код з пробілами і відступами, без сміттєвих операцій, з вашими коментарями того, що робить програма).

8. Скомпілювати програму. Переконатися, що програма виконує необхідні функції. У разі помилок при компіляції, повторити п. 7.

Таблиця 7.1 – Варіанти кодів програм

№	Код програми
1	2

1	`#include "stdafx.h"`
	`#include <iostream>`
	`using namespace std;`
	`void main()`
	`{`
	`  int X;`
	`  cout << "Введіть цифру: ";`
	`  cin >> X;`
	`  // Покрокові обчислення`
	`  for (int i = 1; i < 10 ; i++)`
	`  {`
	`  X *= i;`
	`  cout << "\nРезультат: " << X;`
	`  }`
	`  cin >> X;`
	`}`

2	`#include <stdafx.h>`
	`#include <iostream>`
	`#include <conio.h>`
	`using std::cout;`
	`using std::endl;`
	`using std::cin;`
	`const int n = 5;`
	`void main()`
	`{`
	`  int mas[n];`
	`  cout << "Введіть масив:\n";`

1	2

	`for (int i = 0; i < n; i++)`
	`{`
	`cin >> mas[i];`
	`}`
	`int i = 0;`
	`do`
	`{`
	`mas[i] += 5;`
	`i++;`
	`} while (i < 5);`
	`cout << "\nМасив після додавання:\n";`
	`for (i = 0; i < n; i++)`
	`{`
	`cout << mas[i] << "\t";`
	`}`
	`_getch();`
	`}`

3	`#include <stdafx.h>`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`int i; // лічильник циклу`
	`int sum = 0; // сума чисел від 1 до 1000`
	`setlocale(0, "");`
	`// задаємо початкове значення 1, кінцеве 1000 і задаємо крок циклу - 1`
	`for (i = 1; i <= 1000; i++)`
	`{`
	`sum += i;`
	`}`
	`cout << "Сума чисел " << sum << endl;`
	`cin >> i;`
	`return 0;`
	`}`

4	`#include "stdafx.h"`
	`#include <iostream>`
	`using namespace std;`
	`int main(int argc, char* argv[])`
	`{`
	`int speed = 5, count = 1;`
	`while ( speed < 60 )`
	`{`
	`speed += 10; // збільшення швидкості`
	`cout << count <<"-швидкість = " << speed << endl;`
	`count++; // підрахунок повторень циклу`
	`}`
	`system("pause");`
	`return 0;`
	`}`

5	`#include "stdafx.h"`
	`#include <iostream>`
	`#include <ctime>`
	`using namespace std;`
	`int main(int argc, char* argv[])`
	`{`
	`srand( time( 0 ) );`
	`int unknown_number = 1 + rand() % 10; // число, що загадується`
	`int enter_number; // змінна для збереження числа, що вводиться`
	`cout << "Введіть число [1:10]: "; // починаємо відгадувати`
	`cin >> enter_number;`
	`while ( enter_number != unknown_number )`
	`{`
	`cout << " Введіть число [1:10]: ";`
	`cin >> enter_number; // продовжуємо відгадувати`
	`}`

Продовження табл. 7.1

1	2

	`cout << "Ви виграли!!!\n";`
	`system("pause");`
	`return 0;`
	`}`

6	`#include "stdafx.h"`
	`#include <iostream>`
	`using namespace std;`
	`int main(int argc, char* argv[])`
	`{`
	`int counter_even = 0;`
	`for (int count = 2; count <= 50; count += 2) // заголовок циклу`
	`counter_even++; // підрахунок парних чисел`
	`cout << "Кількість парних чисел = " << counter_even << endl;`
	`system("pause");`
	`return 0;`
	`}`

7	`#include "stdafx.h"`
	`#include <iostream>`
	`#include <ctime>`
	`using namespace std;`
	`int main(int argc, char* argv[])`
	`{`
	`srand(time(0));`
	`int balance = 8; // баланс`
	`do // початок циклу do while`
	`{`
	`cout << "balance = " << balance << endl; // показати баланс`
	`// змінна, для зберігання значення, що віднімається`
	`int removal = rand() % 3;`
	`// показати значення, що віднімається`
	`cout << "removal = " << removal << endl;`
	`balance -= removal; // управління умовою`
	`}`
	`while ( balance > 0 ); // кінець циклу do while`
	`system("pause");`
	`return 0;`
	`}`

8	`#include "stdafx.h"`
	`#include <iostream>`
	`using namespace std;`
	`int main(int argc, char* argv[])`
	`{`
	`cout << "Введіть початкове значення: "; // початкове значення із інтервалу`
	`int first_limit;`
	`cin >> first_limit;`
	`cout << "Введіть кінцеве значення: "; // кінцеве значення із інтервалу`
	`int second_limit;`
	`cin >> second_limit;`
	`int sum = 0, count = first_limit;`
	`do`
	`{`
	`sum += count; // збільшення суми`
	`count++; // інкремент початкового значення із інтервалу, що задано`
	`} while (count <= second_limit); // кінець циклу do while`
	`cout << "Сума = " << sum << endl; // друк суми`
	`system("pause");`
	`return 0;`
	`}`

Контрольні запитання

1. Що таке обфускація?
2. Навіщо застосовують обфускацію?
3. Які методи обфускації розглянуті в даній роботі?
4. В чому полягає лексична обфускація?
5. У чому полягає обчислювальна обфускація?
6. Є звичайний код програми і обфусцірованний код цієї ж програми. Кожен код скомпілювали і створили виконувані файли програм. Яка програма буде вимагати більше обчислювальних ресурсів при виконанні? Обґрунтуйте свою відповідь.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. *Баричев С.Г.*, *Гончаров В.В.*, *Серов Р.Е.* Основы современной криптографии. – М.: Горячая линия-Телеком, 2001. – 120 с.

2. *Безруков Н.Н.* Классификация компьютерных вирусов MS-DOS и методы защиты от них. – М.: Информэйшн Компьютер Энтерпрайз, 1990. – 48 с.

3. *Безруков Н.Н.* Компьютерная вирусология: Справ. руководство. – К.: УРЕ, 1991. – 416 с.

4. *Гундарь К.Ю.*, *Гундарь А.Ю.*, *Янишевский Д.А.* Защита информации в компьютерных системах. – К.: "Корнейчук", 2000. – 152 с.

5. *Малюк А.А.*, *Пазизин С.В.*, *Погожин Н.С.* Введение в защиту информации в автоматизированных системах. – М.: Горячая линия-Телеком, 2001. – 148 с.

6. *Нечаев В.И.* Элементы криптографии (Основы теории защиты информации): Учеб. пособие для ун-тов и пед. вузов / Под ред. В.А. Садовничего – М.: Высш. шк., 1999. – 109 с.

7. *Новиков Ф.А.* Дискретная математика для программистов. – СПб.: Питер, 2001. – 304 с.

8. *Столлингс В.* Криптография и защита сетей: принципы и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2001. – 672 с.

9. *Устенко І.В.* Системи захисту інформації: Навчальний посібник. – Миколаїв: НУК, 2005. – 67 с.

10. *Хижняк П.Л.* Пишем вирус и… антивирус. / Под общей редакцией И.М. Овсянниковой. – М.: ИНТО, 1991. – 90 с.

ЗМІСТ

Начальне видання

УСТЕНКО Ірина Валеріївна

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ

«БЕЗПЕКА ПРОГРАМ ТА ДАНИХ»

Надруковано з оригінал-макету,
підготовленого автором видання

Підписано до друку 08.06.2021 р. Формат 60×94⁄16. Папір офсетний.
Друк цифровий. Ум. друк. арк. 2,0. Тираж 100.
Зам. №___ від 2021 р.

Надруковано ТОВ "ГРУПА КОМПАНІЙ ПЛАНЕТА"
тел. (048) 775-16-30; (093) 170-33-99