Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

«До захисту допущено»
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ДИПЛОМНИЙ ПРОЕКТ
освітньо-кваліфікаційного рівня "Спеціаліст"
зі спеціальності 7.05010201 "Комп'ютерні системи та мережі"
на тему:

ПРОГРАМНИЙ ЗАСІБ ЗАХИСТУ ІНФОРМАЦІЇ В ОПЕРАЦІЙНІЙ СИСТЕМІ WINDOWS XP

Студент групи КСМзс-51_____Кульчицька-Губ'як Г.Є.
(підпис)

Керівник:
викладач _____ Дубчак Л.О.
(підпис)

Нормоконтроль:
к.т.н., доцент _____Василькiв Н.М.
(підпис)

Консультант
з охорони праці:
доцент _____Сапожник Г.В.
(підпис)

2012

Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії
Спеціальність 7.05010201 – "Комп'ютерні системи та мережі"

"Затверджую"
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА
Кульчицької-Губ'як Галини Євгенівної

1. **Тема проекту:** "Програмний засіб захисту інформації в операційній системі WINDOWS XP" затверджена наказом університету № ___від "___"_____20__ р.
2. **Термін здачі студентом закінченого проекту** "___"_____20__ р.
3. **Вихідні дані для проекту:**Технічне завдання.
4. **Перелік задач, які мають бути вирішені:**
   - провести аналіз існуючих рішень для програмної реалізації;
   - визначити оптимальну структурну перевірки цілісності даних;
   - розробити структуру алгоритму модифікованої хеш-функції;
   - розробити архітектуру програмного комплексу;
   - виконати програмну реалізацію модифікованого алгоритму MD5;
   - виконати тестування програмної частини системи;
   - проаналізувати результати роботи програмного засобу.

5. **Перелік графічного матеріалу** (з точним вказанням обов'язкових креслень)
   - Цикл модифікації хеш-функції MD5. Схема структурна
   - Модифікація хеш-функції MD5. Схема функціональна
   - Стиснення даних. Схема алгоритму
   - Процес вилучення документів. Схема структурна

6. **Консультанти по проекту** (із зазначенням розділів):

Розділ	Консультант	Підпис
Охорона праці	Сапожник Г.В.	

КАЛЕНДАРНИЙ ПЛАН

№	Назва розділів дипломного проекту	Термін виконання	Позначки керівника про виконання завдань
1	Аналіз інформаційної безпеки в комп'ютерних системах	15.09.2011 – 5.11.2011	
2	Алгоритмічне забезпечення модифікованої хеш-функції MD5	6.11.2011 – 31.01.2012	
3	Розробка програмного комплексу	1.02.2012 – 14.04.2012	
4	Охорона праці	15.04.2012 – 23.04.2012	

Завдання прийняв до виконання _____
(підпис)

Керівник дипломного проекту _____
(підпис)

АНОТАЦІЯ

Робота виконана на 97 с, містить 8 рисунків, 9 таблиць, 6 додатків, з них 4 графічного матеріалу.

Метою даного дипломного проекту є розробка програмного комплексу, що дозволяє обчислити значення хеш-функції модифікованого алгоритму MD5 для перевірки цілісності даних в операційній системі Windows XP.

Розроблений програмний комплекс використовує методи інтерфейсу CryptoAPI, що дозволяє побудувати багаторівневу систему захисту, що залежить від рівнів загроз.

Розроблена система апробована на практиці і може бути використана для встановлення цілісності і автентичності інформації.

ANNOTATION

Work is executed on 97 p., contains 8 figures, 9 tables, 6 additions, from them 4 graphic material.

The purpose of this work is development of programmatic complex, which allows calculating the value of one way modification algorithm of hash function MD5 for verification of integrity the data in operation system Windows XP.

Developed a programmatic complex uses the methods of interface of CRYPTOAPI, which allows to build the multilevel system of defence which depends on the levels of threats.

The system is developed approved in practice and can be used for establishment of integrity and authenticity of information.

ТЕХНІЧНЕ ЗАВДАННЯ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

1.1 Найменування — "Програмний засіб захисту інформації в операційній системі Windows XP"

1.2 Область застосування даного комплексу — для керування потоками інформації на мережевому і системному рівнях в операційній системі Windows XP.

2 ОСНОВА ДЛЯ РОЗРОБКИ

Основою для розробки є індивідуальне завдання на дипломний проект, затвердженою кафедрою комп'ютерної інженерії (КІ) факультету комп'ютерних інформаційних технологій Тернопільського національного економічного університету.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

3.1 Метою даної розробки є розробка програмного комплексу для знаходження хеш-функції.

3.2 Призначення - призначена для встановлення автентичності і достовірності інформації, що використовується в сучасних комп'ютерних системах.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами даної розробки є матеріали навчальної і реферативної літератури, технічна документація, науково-дослідні роботи, журнали.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до структури і функцій

Програмний комплекс для обчислення хеш-функції повинен мати модульну структуру.

5.2 Вимоги до апаратної сумісності

5.2.1 Розроблена система перевірки достовірності інформації має забезпечити: обчислення хеш-функції довжиною від 128 до 256 біт, резервне копіювання проміжних даних.

5.3 Вимоги до надійності

5.3.1 В операційній системі Windows на основі розроблюваного програмного засобу повинна забезпечуватися надійна безперебійна робота стандартних та прикладних програм.

5.4 Вимоги до безпеки

5.4.1 Реалізація програмного комплексу модифікованої хеш-функції MD5 повинна відповідати вимогам ГОСТ 25.861-85

5.5 Умови експлуатації

5.5.1. Мікроклімат в приміщеннях, де використовуватиметься розроблений програний засіб, повинен відповідати нормам виробничого мікроклімату для обчислювальних центрів (ГОСТ 12.1.005-88).

6 ВИМОГИ ОХОРОНИ ПРАЦІ

В розділі “Охорона праці” дипломного проекту повинен бути даний аналіз умов праці розробника програмних засобів.

7. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙОМУ

7.1 Представлення дипломного проекту на попередній захист.

7.2 Представлення дипломного проекту на захист.

ЗМІСТ

ВСТУП

Важливою задачею захисту інформації залишається перевірка її цілісності. Контроль цілісності здійснюється шляхом розрахунку деякої контрольної суми даних. Проблема простих алгоритмів обчислення контрольної суми полягає в тому, що досить легко підібрати кілька масивів даних, що мають однакову контрольну суму. Криптоалгоритми широко застосовуються не лише для задач шифрування даних, але й для автентифікації та перевірки цілісності. На сьогодні існують добре відомі і апробовані криптоалгоритми (як симетричні, так і несиметричні), стійкість яких доведена математично або базується на необхідності рішення математично складної задачі (факторизації, дискретного логарифму і т.д.). Криптографічно стійкими вважаються контрольні суми, які обчислюються за допомогою знаходження хеш-функції, яка приєднується до вихідного тексту.

Якщо потрібно перевірити ідентичність файлу, необхідно послати значення хеш-функції. Якщо її значення збігаються, то файли вважаються ідентичними. Одностороння функція визначена на множині натуральних чисел і не потребує великих ресурсів для обчислення власного значення. Проте, обчислити зворотнє значення функції є надзвичайно складною задачею.

В односторонньої хеш-функції може бути безліч імен: функція стиснення, функція скорочення (contraction function), короткий виклад, криптографічна контрольна сума, код цілісності повідомлення (message integrity check (MIC)) і код виявлення маніпуляції (manipulation detectioncode (MDC)). Як би вона не називалася, ця функція посідає важливе місце в сучасній криптографії. Однонапрямлена хеш-функція — це одна із частин багатьох протоколів.

Актуальність даної роботи полягає в тому, що розроблений програмний комплекс дозволить застосувати модифікований алгоритм MD5 для обчислення хеш-функції, що дозволяє встановити автентичність інформації.

Мета роботи полягає у розробці програмного комплексу для операційної

системи Windows XP на основі модифікованого алгоритму хешування MD5, що дозволить здійснити перевірку достовірності інформації.

1 АНАЛІЗ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ В КОМП'ЮТЕРНИХ СИСТЕМАХ

1.1 Сучасний стан захисту інформації в комп'ютерних системах

Надійний захист інформації, який забезпечував би попередження, перекручення або знищення інформації, а також унеможливлював би її зловмисне отримання, використання або несанкціоновану модифікацію є однією із важливих задач. Особливої гостроти набуває ця проблема у зв'язку із масовою комп'ютеризацією інформаційних процесів і, насамперед, у зв'язку з об'єднанням комп'ютерів в інформаційно-обчислювальні мережі, що забезпечує масовий доступ будь-яких користувачів до їх ресурсів.

Впровадження популярних дешевих комп'ютерних систем масового попиту і застосування робить їх надзвичайно вразливими щодо деструктивних впливів. Безліч прикладів, що підтверджують поширеність цього явища, можна знайти на сторінках численних видань, а ще більше в Internet. Причому понад 80% комп'ютерних злочинів відбувається за допомогою Internet. Оцінки втрат коливаються від десятків мільйонів до мільярдів. Точні оцінки неможливо отримати в принципі з багатьох причин.

Протягом останніх років постійно винаходять нові й нові види та форми обробки інформації і паралельно винаходять все нові види і форми її захисту. Однак цілком її ніяк не вдається захистити і, напевно, не вдасться взагалі. Інакше кажучи, можна говорити про деяке кризове становище в забезпеченні безпеки в інформаційних технологіях [1].

Спостерігається збільшення обсягів інформації, що накопичується, зберігається та обробляється за допомогою засобів обчислювальної техніки. При цьому мова йде не тільки про різке і буквальне збільшення самих обсягів, а про розширення методів, способів і можливостей її зосередження і збереження. Наприклад, проникнувши в базу даних, можна отримати інформацію практично про все на світі. Особливо розширилися можливості подібного роду з виникненням

глобальної мережі Internet.

Комп’ютерна техніка за останні роки отримали гігантську потужніть, але стали набагато простішими в експлуатації. Це означає, що все більша кількість користувачів одержує доступ до комп’ютера, а середня кваліфікація їх знижується. Користувачі самі здійснюють їх адміністрування, але більшість з них не в змозі постійно підтримувати безпеку своїх систем на високому рівні, оскільки це вимагає відповідних знань, навичок, часу і коштів. Поширення мережевих технологій об’єднало їх в мережі, тепер безпека мережі починає залежати від безпеки її компонентів і зловмиснику досить порушити роботу однієї з них, щоб скомпрометувати всю мережу.

Телекомунікаційні технології об’єднали локальні мережі в глобальні. І саме розвиток Internet, окрім всіх плюсів користування, забезпечує широкі можливості для здійснення порушень безпеки систем обробки інформації всього світу. Якщо комп’ютер підключений до Internet, то для зловмисника немає ніякого значення, де він знаходиться — у сусідній кімнаті чи на іншому кінці світу.

Розвиток локальних мереж і Internet диктує необхідність здійснення ефективного захисту при віддаленому доступі до інформації, а також взаємодії користувачів через загальнодоступні мережі.

Варто зазначити, що захисту підлягає не будь-яка інформація, а тільки цінна. Цінною ж стає та інформація, володіння якою дасть змогу її дійсному чи потенційному власнику одержати який-небудь виграш: моральний, матеріальний, політичний і ін. Оскільки, в суспільстві завжди існують люди, які бажають мати якісь переваги над іншими, то в них може виникнути бажання незаконним шляхом одержати цінну інформацію, а в її власника виникає необхідність її захищати. Цінність інформації є критерієм при прийнятті будь-якого рішення про її захист. Хоча було багато спроб формалізувати процес оцінки інформації з використанням методів теорії інформації та аналізу рішень, цей процес залишається дуже суб’єктивним.

Для оцінки інформації потрібен її розподіл на категорії не тільки відповідно

до її цінності, а й за важливістю. За рівнем важливості можна розділити: життєво важлива незамінна інформація, наявність якої необхідна для функціонування системи; важлива інформація, яку можна замінити чи відновити, але цей процес важкий і пов'язаний з великими витратами; корисна інформація, яку важко відновити, однак система може досить ефективно функціонувати і без неї; несуттєва інформація, без якої система продовжує існувати.

Для власника джерела інформації та зловмисника (організація чи особа, що прагне незаконно одержати інформацію) значущість однієї і тієї ж інформації може бути різна. Практика показала, що захищати потрібно не тільки секретну інформацію, оскільки і несекретна, піддана несанкціонованим ознайомленням чи модифікації, може привести до витоку чи втрати пов'язаної з нею секретної інформації, а також невиконання функцій обробки секретної інформації.

Перехід до інформаційного суспільства у велетенській сфері інформаційної діяльності людей виокремив інформаційний сектор економіки, основну частину якого становить суто інформаційна індустрія, а в ній найперспективніша структура — Internet-економіка. Глобальна комп'ютерна мережа Internet вважається „четвертим каналом", що зв'язує людей між собою [1]. В інформаційному світі існує дві відомі технологічні тенденції: потужність комп'ютерів зростає вдвічі кожен рік і корисність мережі для суспільства пропорційна квадрату числа користувачів. Надзвичайно високі темпи зростання глобальної комп'ютерної мережі Internet зумовлені тим, що вона базується на обох цих закономірностях. Сьогодні світ бурхливо переживає ще один бум – зміщення акцентів з комунікаційної та інформаційно-пошукової функцій Internet на реалізацію за її допомогою сучасного бізнесу. Електронна пошта, програми електронних пейджерів, чати та інші способи спілкування у мережі забезпечують обмін між діловими партнерами пересічною і навіть стратегічною комерційною інформацією за лічені хвилини. Завдяки цьому долаються географічні та національні кордони географічного простору. Весь світ стає клієнтом фірми, що визначає стратегію маркетингу, оскільки ареною конкуренції стає весь світовий економічний простір.

Це до небачених меж розширює можливості підприємства, хоч і підвищує його ризики.

Мережа стала неперевершеним засобом для проведення маркетингу і здійснення прямих онлайнових продажів, підвищення рівня обслуговування клієнтів, найпотужнішим інструментом управління фірмою і джерелом інформації для наукових і практичних розробок. Складне управлінське завдання з налагодження спільної роботи багатьох структурних підрозділів установи під час обробки інформації вирішується шляхом створення і реалізації Intranet локальної комп'ютерної мережі підприємства, яка працює на основі Internet-технологій. Протягом найближчого десятиріччя глобальна тенденція використання можливостей Internet для бізнесу вплине на десятки секторів світової економіки. Перетворення основних бізнес-процесів з допомогою Internet-технологій становить сутність електронного бізнесу та електронної комерції. І тут особливо важливими є засоби захисту передавання електронних повідомлень, надійний та ефективний зв'язок. Безпека продавців і покупців у мережі має підтримуватися не тільки законодавчими заходами, а й програмно-технічними засобами. Виробники апаратних пристроїв і розробники програмних продуктів, усвідомлюючи важливість ринку електронної комерції як найперспективнішого в сучасній інформаційній економіці, повинні приділяти велику увагу розробці надійних правил і засобів захищеної передачі інформації мережею.

Однією з найпоширеніших крадіжок є ідентифікаційна інформація: злодії збирають персональну інформацію — імена, адреси та інші важливі дані, а після цього замовляють картки під цими іменами. Хакери і кракери „зламують" сайти, які зберігають цю інформацію, а в деяких випадках злочинці вдають із себе легітимних онлайнових торговців і збирають інформацію в покупців, які нічого не підозрюють. Дійсні номери карток також можуть бути автоматично згенеровані [2]. Отже, головними вимогами до здійснення комерційних операцій в Internet є конфіденційність, цілісність, автентифікація, гарантії і збереження таємниці. Крім відповідальності осіб і установ та дотримання законів, що захищають споживача

від можливого шахрайства, наведені вимоги можна забезпечити технічними і програмними засобами захисту інформації.

Таким чином, інформація — це одна з найцінніших речей в сучасному житті, і поява глобальних комп'ютерних мереж зробила простим отримання доступу до інформації як для окремих людей, так і для великих організацій. Але легкість і швидкість доступу до даних за допомогою комп'ютерних мереж, таких як Internet, також зробили значними наступні загрози безпеки даних за відсутності заходів їх захисту: неавторизований доступ до інформації; неавторизована зміна інформації; неавторизований доступ до мереж та інших сервісів; інші мережеві атаки, такі як повтор перехоплених раніше транзакцій і атаки типу "відмова в обслуговуванні" [2].

Коли кінцевий користувач посилає електронне повідомлення по мережі, він широко відкриває двері перед потенційним агресором. Як стверджують знавці комп'ютерної безпеки, люди навіть не уявляють собі, наскільки легко досвідчений хакер може скористатися недосконалістю комунікаційних протоколів і ознайомитися із змістом електронних Internet-листів, послати фальшиве повідомлення і навіть отримати доступ до інших підключених до мережі систем. Все, що для цього треба знати — це ім'я домена або одна IP-адреса; якщо дана інформація стає відома зловмиснику, то перед ним відкривається шлях до здійснення різноманітних негативних дій.

Саме з цих причин на сьогодні велика частина уваги приділяється пошуку і аналізу засобів і методів підвищення захисту інформації в глобальній комп'ютерній мережі. Всі рішення грунтуються на попередньому шифруванні даних при передачі по каналах зв'язку. Такий підхід істотно зменшує вірогідність злому. Існує багато технологій, які використовують різні алгоритми і стандарти шифрування. Але слід врахувати, що при роботі в мережі на криптографічну підсистему накладається ряд обмежень. Оскільки час шифрування обмежений, в ній повинен використовуватися дуже простий, легко реалізований і в той же час криптостійкий алгоритм шифрування. Для підвищення виробництва в цілому, виробники таких систем

пішли по шляху реалізації деяких блоків на апаратному рівні. Тобто було створено спеціальне обладнання, що виконує ряд функцій, пов’язаних з обробкою і шифруванням даних.

1.2 Структурний підхід до проведення атак в операційній системі сімейства Windows

Під час дослідження ядра операційної системи Windows NT провідний спеціаліст з інформаційних систем Нім Вочман виявив, що при звичайній зміні одного біта програмного коду змінюються всі налаштування безпеки для всієї мережі комп’ютерів під керуванням Windows.

Варто звернути увагу і на вектор вторгнення. Це є структурний підхід, що дозволяє виявити недолік в програмі, здійснює переміщення коду з одного місця зберігання в інше; характеризує структуру даних або середовище, яке містить і передає код з одної області зберігання в іншу. Що стосується переповнення буфера, то вектори вторгнення — це добре підготовлені вхідні повідомлення, які заставляють програму порушника переходити в стан переповнення буфера. Тобто це фрагмент атаки, під час якої відбувається проникнення і запуск коду в програмі.

Розрізняють вектор вторгнення і корисне навантаження (payload) [3]. Корисне навантаження — це програмний код, який реалізує наміри атакуючого. Поєднання вектора вторгнення і корисного навантаження використовується для проведення повної (цілісної) атаки, а сам вектор вторгнення використовується для отримання контролю над вказівниками команд. Після отримання контролю над вказівником команди зловмисник може встановити його на будь-який буфер, що контролюється, або іншу область пам’яті, в якій зберігається корисне навантаження. Таким чином, коли атакуючий може контролювати вказівник команд, тоді в нього з’являється можливість передати керування (змінити хід

виконання програми) програмою коду корисного навантаження. Зловмисник заставляє вказівник команд вказувати на небезпечний код, що призводить до його виконання.

Вектори вторгнення завжди пов’язані з конкретною помилкою або недоліком в програмному забезпеченні, що підлягає атаці. Для кожної версії програмного забезпечення існують унікальні вектори вторгнення. При розробці засобів атаки зловмисник повинен спроектувати і створити конкретні вектори вторгнення для конкретної задачі.

При створенні вектора вторгнення повинні враховуватись деякі фактори: розмір буфера, впорядкування даних в пам’яті і обмеження набору символів. За звичай, вектори вторгнення відповідають правилам визначеного протоколу. Наприклад, переповнення буфера в маршрутизаторі може бути організовано за допомогою вектора вторгнення для обробки пакетів протокола BGP (Border Gateway Protocol), як зображено на рисунку 1.1.

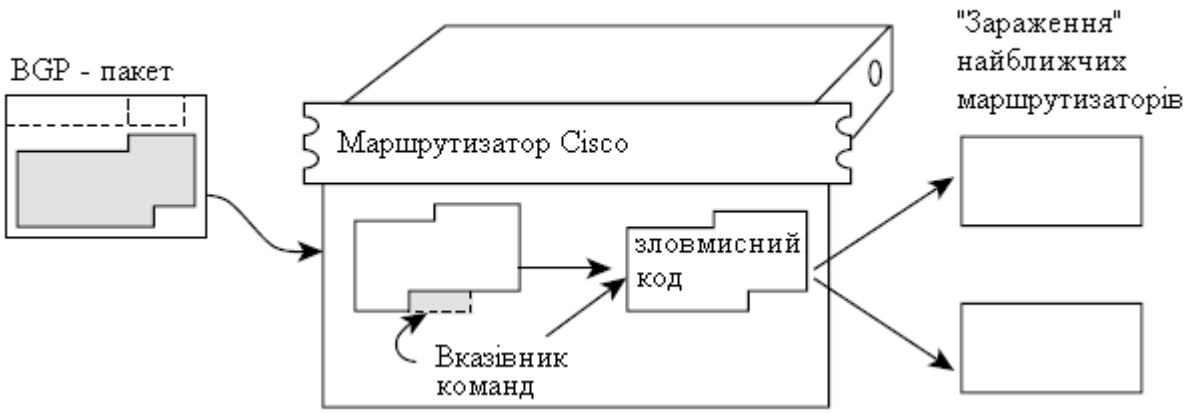

Рисунок 1.1 — Використання зловмисного BGP – пакету для взлому маршрутизатора Cisco

Цей вектор вторгнення реалізовано як спеціально підготовлений BGP – пакет. Оскільки підтримка протоколу BGP є необхідною для нормальної роботи в Internet, то атака такого типу здатна знищити системи, які обслуговують мільйони

користувачів. Більш яскравим прикладом є протокол OSPF (Open Shotest Path First). В маршрутизаторах Cisco реалізація цього протокола може бути використана для видалення інформації про цілу локальну мережу масштабного мережевого вузла.

Для атак на переповнення буфера характерна наявність чіткої межі між вектором вторгнення і корисного навантаження. Ця межа називається адресою повернення (return address). Адресу повернення можна схематично визначити в той момент, коли корисне навантаження або отримує керування над центральним процесором, або не спрацьовує [3]. На рисунку 1.2 зображено вектор вторгнення, який містить вказівник команд, який в результаті завантажується в центральний процесор (CPU) комп'ютера, що атакується.

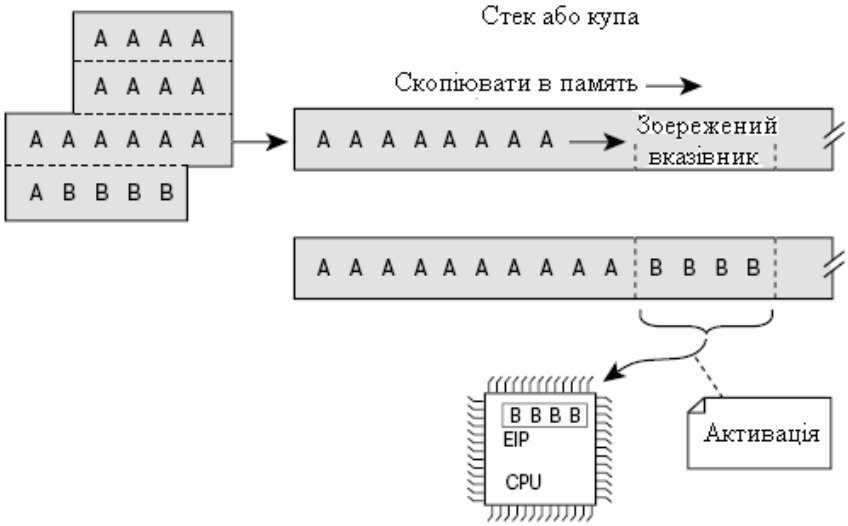

Рисунок 1.2 — Розміщення вказівника в центральному процесорі комп'ютера

Варто зазначити, що однією із важливих задач вектора вторгнення є вибір області пам'яті, в якій повинно зберігатися корисне навантаження. Його можна зберігати безпосередньо у буфері або в окремій області пам'яті. Зловмисник повинен знати адресу комірки пам'яті (в якій зберігається корисне навантаження), і цю адресу має враховувати вектор вторгнення (рисунок 1.3). Обмеження для набору символів призводять до обмежень в значеннях, які доступні для того, щоб

вказати адреси пам’яті. Наприклад, при обмеженні на введення чисел більше ніж 0xB0000001, вибраний вказівник повинен знаходитися в пам’яті вище цієї адреси [4]. Такі обмеження відповідають проблемам, які виникають при перетворенні аналізаторами байтів, які використовуються для символів коду атаки, на інші значення, або при роботі фільтрів, які блокують недопустимі символи у потоці даних. На практиці в багатьох атаках використовуються літерно-цифрові символи.

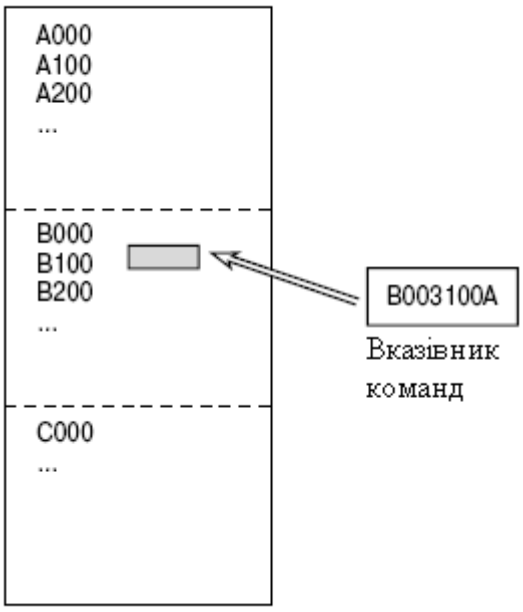

Рисунок 1.3 — Вказівник на корисне навантаження в пам’яті

Атаки на переповнення буфера застосовуються і до вбудованих систем (мережеве обладнання, принтери, мобільні телефони та ін. невеликі пристрої). Програмний код, який здійснює керування цими вбудованими системами вразливий до вище згаданих атак. Цікаво, що серверне програмне забезпечення стає більш стійким до атак на переповнення буфера, а основний акцент зміщається на програмне забезпечення вбудованих систем.

Вбудовані системи запускаються на різних апаратних платформах. У більшості таких систем для зберігання даних використовується технологія NVRAM. Вбудовані системи широко застосовуються у воєнній апаратурі,

наприклад, стандартній воєнній радарній системі AN/SPS-73 [4]. Ця радарна система працює під керуванням захищеної системи VxWorks. Як і в більшості комерційних системах із закритим кодом, в операційній системі VxWorks і програмному забезпеченні для неї є багато помилок, що дозволяють проводити атаки на переповнення буфера. Більшість вразливих місць можуть бути без аутентифікації, наприклад, через RPC-пакети [4]. Таким чином, апаратне забезпечення із вбудованими системами є ціллю для проведення атак.

Проте існує думка, що вбудовані системи не вразливі до віддалених атак, оскільки через відсутність в пристрої інтерактивного командного інтерпретатора доступ або використання "коду командного інтерпретатора" є неможливим. Помилковим є твердження, що зловмисник зможе лише вивести із ладу пристрій керування. Насправді внесений код здатний виконати будь-який набір команд, в тому числі і програму командного інтерпретатора, яка підтримує функції рівня операційної системи. Немає значення, що код не постачається разом із пристроєм. Цей код додається під час атаки. Тобто при атаках на переповнення буфера не потрібно наявності повнофункціонального інтерактивного інтерпретатора TCP/IP. Натомість, при атаці може повністю стиратися конфігураційний файл або замінюватися пароль [5].

Багато складних програм можуть бути встановлені в ході віддалених атак на вбудовану систему. Код командного інтерпретатора є лише одним із варіантів. Не відіграє важливої ролі те, який процесор використовується або яка схема адресації, оскільки зловмиснику потрібно створити лише діючий код для апаратних засобів, що підлягають атаці. Апаратні засоби із вбудованим програмним забезпеченням є достатньо документовані і загальнодоступні.

Група дослідників з проблем захисту Phenoelit створила програму з кодом командного інтерпретатора для проведення віддаленої атаки на маршрутизатор Cisco 1600 на основі процесора Motorola 68360 QUICC (програма демонструвалася на конференції Blackhat в 2002 році) [5]. Для цієї атаки вектор вторгнення використовує переповнення буфера в операційній системі IOS від Cisco і декілька

нових методів використання структур керування “купами” в IOS. Змінюючи структури “купи”, можна впровадити і виконати зловмисний код. В опублікованому варіанті атаки код командного інтерпретатора — це створений код у вигляді машинних команд Motorola, який відкриває потайний хід до маршрутизатора. Цим кодом можна скористатися при наявності будь-якого переповнення буфера в пристроях Cisco.

Ціллю атак на переповнення буфера стають і СУБД (системи управління базами даних). В будь-якій системі керування базами даних існує декілька точок для проведення атак. Програма для роботи з базою даних складається із певного числа взаємодіючих компонентів. В їх число входять сценарії (об’єднують різні модулі між собою), програми для роботи через інтерфейс командної стрічки, процедури і клієнтські програми, що безпосередньо пов’язані з базою даних. Кожен із цих компонентів є потенційним об’єктом для проведення атаки на переповнення буфера. В якості прикладу можна навести вразливу платформу SQL, в якій є нестійка функція OpenDataSource().

Атака на функцію OpenDataSource() була виконана за допомогою протокола трансакції SQL (T- SQL), для якого встановлюється прив’язка до TCP – порту 1433 [6]. Тобто цей протокол дозволяє багатократно передавати аналізатору оператори SQL.

Процедури, що зберігаються, використовуються для передачі даних сценаріям або бібліотекам DLL. Якщо в сценарії або бібліотеці DLL є помилки стрічки форматування, або якщо в сценарії використовуються вразливі виклики функцій бібліотеки (наприклад, strcpy() або system()), то з’являється можливість їх використати в своїх цілях для системи управління базою даних. Майже кожна процедура передає частину запиту. В даному випадку зловмисник може використати частину запиту, що передається, для переповнення буфера. Наявним прикладом є помилка в Microsoft SQL Server [6]. Зловмисник міг викликати переповнення буфера в коді, який обробляє розширені процедури, що зберігаються.

Деколи сценарії або процедури, що зберігаються, викликають програму з інтерфейсом командної стрічки і подають її на введення даних із запиту. У багатьох випадках це призводить до переповнення буфера або вразливості з можливістю передачі команд. Окрім того, якщо в сценарії не використовується бібліотека API для роботи з базою даних, то звичайні оператори SQL можна передавати для обробки програмі з інтерфейсом командної стрічки. Це ще одна можливість для проведення атаки на переповнення буфера.

Програмні атаки на системи, що написані мовою Java, використовують особливості мови і при перевірці цифрового підпису аплетів [6]. Переповнення буфера частіше виникає в програмах підтримки, які є зовнішніми по відношенню до JVM. Сама JVM пишеться на C для конкретної платформи, тобто без ретельної уваги до деталей реалізації, тому віртуальна машина JVM може виявитися нестійкою до атак на переповнення буфера. Проте стандартна реалізація JVM від компанії Sun Microsystems добре перевірена, і статистичні перевірки, які виконуються для викликів вразливих функцій, не дають позитивних результатів.

Окрім JVM, багаточисельні проблеми переповнення буфера характерні для систем, в яких використовується Java. В якості прикладу розглянемо систему керування реляційними базами даних Progress, в якій вбудована програма jvmStart, що призначена для запуску віртуальної Java - машини. В jvmStart є помилка перевірки на правильність вхідних даних, що представляються в командній стрічці (помилка стрічки форматування). Дані вводяться в стрічку через функцію printf () як аргумент стрічки. Звідси можна зробити висновок, що розробники програмного забезпечення повинні розглядати систему цілому, а не лише створювати окремі компоненти.

Багато атак базуються на використанні змінних середовища, які є вразливим місцем, де переповнення буфера може застосуватися для передачі в програму зловмисного коду. При цьому програма може приймати неперевірені вхідні дані.

Одним із методів пошуку можливостей переповнення буфера є подання на вхід програми довгих аргументів, що дозволяє досліджувати результати виконання.

Цей підхід використовується засобами безпеки програмного забезпечення. Тому, можна створювати довгі запити для Web або FTP-сервера.

1.3 Формування вимог та постановка задачі

У сучасному програмному забезпеченні (ПЗ) криптоалгоритми широко застосовуються не лише для задач шифрування даних, але й для автентифікації та перевірки цілісності. На сьогодні існують добре відомі і апробовані криптоалгоритми (як симетричні, так і несиметричні), стійкість яких доведена математично, або базується на необхідності рішення математично складної задачі (факторизації, дискретного логарифму і т.д.). Таким чином, атака проводиться шляхом повного перебору або за допомогою рішення математично складної задачі.

Постійно з'являється інформація про помилки або «дірки» в тій чи іншій програмі (в т.ч. які використовують криптоалгоритми). Тому знання про атаки і «дірки» в криптосистемах, а також розуміння причин їхнього виникнення є однією із необхідних умов розробки захищених систем. Перспективним напрямом досліджень в даній області є аналіз успішно проведених атак або виявлених недоліків в криптосистемах з метою їх узагальнення, класифікації і виявлення причин і закономірностей їх появи і існування.

Група нестійких криптоалгоритмів є найбільш поширеною причиною через чинники, які розглянемо нижче.

Мала швидкість стійких криптоалгоритмів — це основний чинник, що ускладнює застосування хороших алгоритмів, наприклад, в системах «тотального» шифрування або поточного шифрування. Зокрема, програма Norton DiskReet, що містить в собі реалізацію алгоритму DES, при зміні користувачем ключа може не перешифрувати весь диск, оскільки це займе багато час. Аналогічно, програма компресії Stacker фірми Stac Electronics має опцію закриття паролем компресуючих

даних. Проте вона не має фізичної можливості шифрувати цим паролем свій файл розміром в декілька сот мегабайт, тому вона обмежується дуже слабким алгоритмом і зберігає хеш-функцію, що залежить від пароля і захищених даних. Досліджено величину криптостійкості цієї функції, яка дорівнює 2^8, тобто пароль може бути розкритий тривіально.

Експортні обмеження — це причина, що пов'язана з експортом криптоалгоритмів або з необхідністю здобувати патент чи права на них. Зокрема, в США заборонений експорт криптоалгоритмів з довжиною ключа більше 40 біт. Очевидно, що така криптостійкість не може вважатися надійною при сучасних обчислювальних потужностях і навіть на персональному комп'ютері, якщо припустити, що швидкість перебору становитиме 50 000 паролів/сек, то в середньому одержимо час перебору близько 4 місяців. Відомі приклади програм, що підлягають експортним обмеженням - це останні версії браузерів Інтернету, зокрема, Netscape Navigator фірми Netscape Communications і Internet Explorer фірми Microsoft. Вони шифрують дані зі 128-бітним ключем для користувачів всередині США і зі 40-бітним ключем для всіх інших [2].

Незнання або небажання використовувати відомі алгоритми — це ситуація, яка особливо спостерігається в програмах типу Freeware і Shareware, наприклад, в архіваторах. Архіватор ARJ (до версії 2.60 включно) використовує (по замовчуванню) дуже слабкий алгоритм шифрування – просте гамування. Припустимо, що в даному випадку його використання допустимо, оскільки текст, що стискається, повинен бути абсолютно не надмірним і статистичні методи криптоаналізу тут не підходять. Проте після детальнішого дослідження виявилось, що в тексті, що архівується, присутня деяка невипадкова інформація, наприклад, таблиця Хаффмана і деяка інша службова інформація. Тому, коли відомо або з деякою ймовірністю можна передбачити значення цих службових змінних, то можна із тією ж ймовірністю визначити і відповідні символи пароля.

Використання слабких криптоалгоритмів часто призводить до успішного проведення атаки за відкритим текстом. У випадку з архіватором ARJ, якщо

зловмиснику відомий хоч би один файл із зашифрованого архіву, то він з легкістю визначить пароль архіву і витягне звідти решту файлів. Навіть, якщо немає жодного файлу в не зашифрованому вигляді, то в будь-якому випадку просте гамування дозволяє досягти швидкості перебору в 350000 паролей/сек. на машині класу Pentium.

Не дивлячись на те, що в цьому випадку застосовуються криптостійкі або сертифіковані алгоритми, ця група причин призводить до порушень безпеки криптосистем із-за їх неправильної реалізації.

Існує багато прикладів, коли криптосистема або обрізає пароль користувача, або генерує з нього дані, що мають меншу кількість біт, ніж сам пароль. Наприклад: у багатьох (попередніх) версіях UNIX пароль користувача обрізається до 8 байт перед хешуванням.

Перевірка на слабкі ключі має бути важливою вимогою до створення захищеної системи. Деякі криптоалгоритми (зокрема, DES, IDEA) при шифруванні із специфічними ключами не можуть забезпечити належний рівень криптостійкості. Такі ключі називають слабкими (weak). Для DES відомо 4 слабких і 12 напівслабких (semi-weak) ключів. Приблизна кількість слабких ключів IDEA складає близько 251.

Недостатня захищеність від руйнуючих програмних засобів (РПЗ) — це комп'ютерні віруси, програмні закладки і тому подібні програми, що здатні перехопити секретний ключ або відкриті дані, а також замінити алгоритм на не криптостійкий. У тому випадку, якщо програміст не передбачив достатніх способів захисту від РПС, вони легко здатні порушити безпеку криптосистеми. Особливо це актуально для операційних систем, які не мають вбудованих засобів захисту або засобів розмежування доступу. Перехоплення пароля — це давно відомий спосіб викрадення пароля, коли программа-«фантом» емулює запрошення ОС, пропонуючи ввести ім'я користувача і пароль, запам'ятовує його в деякому файлі і зупиняє роботу з повідомленням "Invalid password". Для MS DOS і Windows існує безліч закладок для читання і збереження паролів, що набирають на клавіатурі

(через перехоплення відповідного переривання), наприклад, при роботі утиліти Diskreet v. 6.0.

Наявність залежності в часі обробки ключів — це порівняльно новий аспект щодо недостатньо коректної реалізації криптоалгоритмів. Багато криптосистем з різною швидкістю обробляють різні вхідні дані.

Основним методом виявлення атак, що використовує більшість сучасних комерційних продуктів, є сигнатурний аналіз. Відносна простота даного методу дозволяє з успіхом використовувати його на практиці. Система виявлення атак (IDS), що застосовують сигнатурний аналіз, звичайно нічого не знають про правила політики безпеки (тому в даному випадку мова йде не про злочинну активність, а тільки про атаки). Основний принцип їхнього функціонування — порівняння у системі/мережі подій із сигнатурами відомих атак — той же принцип, що використовується й в антивірусному програмному забезпеченні.

Існує два, які не виключають один одного, підходи до виявлення мережевих атак: аналіз мережевого трафіка й аналіз контента. У першому випадку аналізуються тільки заголовки мережевих пакетів, у другому — їхній вміст.

Звичайно, найбільш повний контроль інформаційних взаємодій може бути забезпечений тільки шляхом аналізу всього вмісту мережевих пакетів, включаючи їхні заголовки й області даних.

2 АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ МОДИФІКОВАНОЇ ХЕШ-ФУНКЦІЇ MD5

2.1 Структура модифікованої хеш-функції MD5

Криптографічні методи забезпечують не лише конфіденційність, але і контроль цілісності даних, що передаються або зберігаються. Контроль цілісності в основному здійснюється шляхом розрахунку деякої "контрольної суми" даних. Математиками та інженерами, що працюють в області передачі даних і теорії кодування розроблено безліч алгоритмів, що розраховують контрольні суми даних, що передаються. Для багатьох випадків простої контрольної суми (наприклад, відомого алгоритму crc32 чи послідовного побайтно або послівного додавання вихідного тексту з відомою константою) її вистачає для роботи, особливо тоді, коли хороша швидкість обробки даних і не відомий наперед обсяг даних (типовий випадок — передача даних по каналах зв'язку) [5].

Проблема простих алгоритмів обчислення контрольної суми полягає в тому, що досить легко підібрати кілька масивів даних, що мають однакову контрольну суму. Криптографічно стійкі контрольні суми обчислюються як результат застосування до вихідного тексту хеш-функції.

Однією із гіпотез теорії складності і теорії функцій є гіпотеза про існування однобічних функцій. Під однобічною функцією розуміють функцію, яка визначена (наприклад) на безлічі натуральних чисел і не потребує для обчислення свого власного значення великих обчислювальних ресурсів. Проте, обчислення зворотної функції (тобто за відомим значенням функції відновити значення аргументу) виявляється неможливо чи теоретично (у крайньому випадку) неможливо обрахувати [6].

Загальне існування односторонніх функцій поки не доведено. Тому, всі хеш-функції, що використовуються на даний час, є лише "кандидатами" в односторонні

функції, хоча і мають досить хороші властивості. Основними властивостями криптографічно "стійкої" хеш-функції є властивість розсіювання, властивість стійкості до колізій і властивість необоротності.

Колізією хеш-функції H називається ситуація, при який існують два різних тексти T1 і T2, але H(T1) = H(T2). Значення хеш-функції завжди має фіксовану довжину, а на довжину вихідного тексту не накладається ніяких обмежень. Звідси можна зробити висновок, що колізії існують. Вимога стійкості до колізій означає, що для криптографічно "стійкої" хеш-функції для заданого тексту T1 неможливо знайти текст T2, що викликає колізію.

Властивість розсіювання вимагає, щоб мінімальні зміни тексту, що підлягає хешуванню, викликали максимальні зміни в значенні хеш-функції.

Основними алгоритмами, що застосовуються на сьогоднішній день алгоритмами, які реалізують хеш-функції є MD2, MD4, MD5, HAVAL, SHA-1 і його варіант SHA-2, російський алгоритм, що описується в стандарті ДСТ Р 34.11-94. Найбільш часто використовуються MD5, SHA-1. Довжина значення хеш-функції різна. Типовою довжиною є 16—32 байта. Проте, багато досліджень у застосуванні хеш-функції, стверджують, що прийдеться відмовитися від MD5, тому що, як було виявлено, "його стійкість до колізій знизилась і, напевно, підійшла близько до тієї оцінки, після якої про стійкість взагалі говорити не приходиться" [6].

В односторонньої хеш-функції може бути безліч імен: функція стиснення, функція скорочення (contraction function), короткий виклад, криптографічна контрольна сума, код цілісності повідомлення (message integrity check (MIC)) і код виявлення маніпуляції (manipulation detectioncode (MDC)). Як би вона не називалася, ця функція посідає важливе місце в сучасній криптографії. Однонапрямлена хеш-функція — це одна із частин багатьох протоколів.

Хеш-функції — це функції, математичні чи інші, які одержують на вхід змінну довжину і перетворюють її у фіксовану довжину, звичайно, меншої довжини (значення хеш-функції). Просту хеш-функції можна розглядати як функцію, що

одержує змінну довжину і повертає байт, який є результатом операцій XOR всіх вхідних байтів.

Зміст хеш-функції представляє собою одержання характерної ознаки змінної довжини — значення, за яким аналізуються різні вхідні повідомлення при рішенні зворотньої задачі. Оскільки, хеш-функція є співвідношенням "багато до одного", неможливо з усією визначеністю сказати, що два рядки збігаються, але їх можна використовувати, одержуючи прийнятну оцінку точності.

Однонапрямлена хеш-функція — це хеш-функція, що працює тільки в одному напрямку: легко обчислити значення хеш-функції за вхідним повідомленням, але важко відновити вхідне повідомлення, значення хеш-функції якого дорівнює заданій величині. Згадувані раніше хеш-функції не є однонапрямленими: задавши конкретний байт, не представляє великих затрат створити рядок байтів, XOR яких дає задане значення.

Стійкою однонапрямленою хеш-функцією вважається хеш-функція без зіткнень — важко створити два вхідних повідомлення з однаковим значенням хеш-функції.

Хеш-функція є відкритою. Безпека однонапрямленої хеш-функції полягає саме в її однонапрямленості. На виході немає видимої залежності від входу. Зміна одного біту вхідного повідомлення призводить до зміни, в середньому, половини бітів значення хеш-функції. Неможливо знайти таке вхідне повідомлення, що відповідає заданому значенню хеш-функції.

Якщо потрібно перевірити ідентичність файлу, необхідно послати значення хеш-функції. Якщо значення хеш-функції збігається, то файли вважаються ідентичними. Це особливо корисно при фінансових транзакціях, коли захищають електронну комерцію, щоб замість зняття з рахунку $100 не зняли $1000. В звичайних умовах можна використовувати однонапрямлену хеш-функцію без ключа, для того, щоб будь-хто міг перевірити значення хеш-функції. Якщо потрібно, щоб перевірку значення хеш-функції міг здійснювати тільки один одержувач, то для цього використовуються методи наведені нижче.

Код перевірки дійсності повідомлення (message authentication code (MAC)), відомий також як код перевірки дійсності даних (data authentication code (DAG)), це є однонапрямлена хеш-функцію з додаванням секретного ключа. Значення хеш-функції є функцією від вхідного повідомлення і ключа. Особливості застосування ті, що і для хеш-функцій, але тільки той, хто знає ключ, може перевірити значення хеш-функції. MAC можна створити за допомогою хеш-функції чи блокового алгоритму шифрування, існують також і спеціалізовані MAC [6].

Однонапрямлена функція $H(M)$ застосовується до повідомлення довільної довжини M і повертає значення фіксованої довжини h, $h = H(M)$, де h має довжину m. Багато функцій дозволяють обчислювати значення фіксованої довжини по вхідних даних довільної довжини, але в однонапрямлених хеш-функцій є додаткові властивості, які роблять їх однонапрямленими.

Знаючи M, можна легко обчислити h. Знаючи H, важко визначити M, для якого $H(M)=h$. Знаючи M, важко визначити інше повідомлення, M', для якого $H(M)= H(M')$. Довжина 64-бітових хеш-функцій є занадто малою, щоб протистояти розкриттю методом дешифрування. Більш практичні однонапрямлені хеш-функції, що видають 128-бітові хеш-значення. При цьому, щоб знайти два документи з однаковими хеш-значеннями, для розкриття методу дешифрування прийдеться хэшувати 2^{64} випадкових документів, що не характеризує її стійкою до атак. NIST у своєму стандарті безпечного хешування (Secure Hash Standard (SHS)), використовує 160-бітове хеш-значення. Це ще більше ускладнює розкриття методу дешифрування, для якого знадобиться 2^{80} хешувань.

Після деякої первісної обробки MD5 обробляє вхідний текст 512-бітовими блоками, розбитими на 16 32-бітових підблоків. На виході алгоритму буде набір з чотирьох 32-бітових блоків, що з’єднується в єдине 128-бітове хэш-значення. По-перше, повідомлення доповнюється так, щоб його довжина була на 64 біта коротшою за числа, кратні 512. Цим доповненням є 1, за якою до кінця повідомлення ставляться нулі (скільки потрібно). Потім, до результату додається 64-бітове представлення довжини повідомлення. Ці дві дії служать для того, щоб

довжина повідомлення була кратна 512 бітам (що потрібно для частини алгоритму, що залишилася) і щоб гарантувати, що різні повідомлення не будуть виглядати однаково після доповнення. Ініціалізуються чотири змінних:

A = 0x01234567

B = 0x89abcdef

C = 0xfedcba98

D = 0x76543210

Вони називаються змінними зчеплення.

Розглянемо основного циклу алгоритму. Цей цикл продовжується виконуватися, поки не вичерпаються 512-бітові блоки повідомлення. Чотири змінних копіюються в інші змінні: A в a, B в b, C в c і D в d.

Головний цикл складається з чотирьох дуже схожих етапів (додаток А). На кожному етапі 16 разів використовуються різні операції. Кожна операція є нелінійною функцією над трьома змінними з a, b, c і d. Потім вона додає цей результат до четвертої змінної, підблоку тексту і константи. Далі результат циклічно зсувається вправо на змінне число бітів і додає результат до однієї із змінних a, b, c і d. Нарешті результат заміняє одну із змінних a, b, c і d (додаток Б). Існують чотири нелінійних функції, що використовуються на кожному етапі для обчислення алгоритму (для кожного етапу - інша функція).

$$F(X,Y,Z) = (X \cup Y) \cap ((\overline{X}) \cap Z),$$

$$G(X,Y,Z) = (X \cap Z) \cup (Y \cap (\overline{Z})),$$

$$H(X,Y,Z) = X \oplus Y \oplus Z,$$

$$I(X,Y,Z) = Y \oplus (X \cup (\overline{Z})),$$

де $\oplus$ — це XOR;

$\cup$ — AND;

$\cap$ — OR;

$\overline{X}$ — NOT.

Ці функції розроблені таким чином щоб відповідні біти X, Y і Z були

незалежні і незміщені, а кожен біт результату також був би незалежним і незміщеним. Функція F — це побітова умова: якщо *X*, то *Y*, інакше *Z*. Функція H —побітова операція парності.

Якщо M_j означає j-ий підблок повідомлення (від 0 до 15), а <<<*s* означає циклічний зсув вліво на *s*-бітів, то використовуються наступні чотири операції:

FF(a,b,c,d,M_j,s,t_i) означає $a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$

GG(a,b,c,d,M_j,s,t_i) означає $a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$

HH(a,b,c,d,M_j,s,t_i) означає $a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$

II(a,b,c,d,M_j,s,t_i) означає $a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$.

Вище наведені константи t_i вибиралися наступним чином:

На *i-ому* етапі t_i є цілою частиною 2 32 *abs(sin(*i*)), де *i* виміряється в радіанах. Після всього цього *a*, *b*, *c* і *d* додаються до *A*, *B*, *C* і *D*, відповідно, і алгоритм продовжується для наступного блоку даних. Остаточним результатом служить об'єднання *A*, *B*, *C* і *D*:

$$aa = a + aa,$$
$$b = b + bb,$$
$$c = c + cc$$
$$d = d + dd$$

$$H(M) = aa \| bb \| cc \| dd .$$

Модифікація алгоритму MD5 — одностороння хеш-функція змінної довжини [7]. Повідомлення опрацьовується блоками по 1024 біт. Це вдвічі більше, ніж в MD5. Використовується вісім 32-бітні змінні зчеплення і змінна кількість етапів від трьох до п’яти, в кожному етапі є 16 дій. Функція може видавати хеш-значення довжиною 126, 160, 192, 224, 256 бітів. Розглянемо модифікацію алгоритму MD5, що використовує три етапи стиснення (256 біт). Для цього на вхід подається 1024-бітне повідомлення $M = (m_0, m_2, \ldots, m_{31})$ і на виході отримуємо 256 бітне значення (додаток В). Процес стиснення відбувається наступним чином:

1. Нехай $(aa, bb, cc, dd, ee, ff, gg, hh)$ буде 256-бітним значенням. Ініціалізуємо змінні (a, b, c, d, e, f, g, h) як $(aa, bb, cc, dd, ee, ff, gg, hh)$.

2. Формуємо наступні 96 кроків (оскільки кожний етап має по 32 кроки):

$для\ \ i = 0,1,2$

$для\ \ j = 0 \ldots 31$

$p = f_{i+1}(g, f, e, d, c, b, a)$

$r = (p >> 7) + (h >> 11) + m_{i,j} + k_{i,j}$

$h = g$

$g = f$

$f = e$

$e = d$

$d = c$

$c = b$

$b = a$

$a = r$

В кожному етапі використовується константа $k_{j,i}$. Символ “>>” означає циклічний зсув вправо. Перестановка повідомлення на кожному етапі описується в [5]. Функції f_1, f_2, f_3 визначаються наступним чином:

$$f_1(g, f, e, d, c, b, a) = cd \oplus ag \oplus bf \oplus ce \oplus e,$$
$$f_2(g, f, e, d, c, b, a) = adf \oplus bcf \oplus ef \oplus ef \oplus ac \oplus df \oplus bd \oplus bc \oplus fg \oplus g,$$
$$f_3(g, f, e, d, c, b, a) = def \oplus cf \oplus be \oplus dg \oplus ad \oplus a$$

3. Змінні a, b, c, d, e, f, g, h додаємо до вхідних значень:

$aa = a + aa,\ bb = b + bb, \ldots, hh = h + hh.$

4. $H(M) = hh \| gg \| ff \| ee \| dd \| cc \| bb \| aa.$

Основні властивості функції f_1 описуються в [7].

2.2 Особливості застосування хеш-функції в комп’ютерних системах

Роль інформації на сьогодні настільки велика, що інформаційна індустрія стала однією із провідних галузей дослідження, а пристрої, що одержали

поширення для обробки цифрових даних – комп'ютери – є одним із символів нашої цивілізації. Інформація, представлена у різних формах, подібно іншим товарам виробляється, зберігається, транспортується до споживача, продається, купується, нарешті споживається, застаріває, псується, і т.д. Протягом життєвого циклу інформаційні масиви можуть піддаватися різним небажаної для їхнього споживача впливам, проблемам боротьби з якими стає непередбаченою.

Через те, що інформація має нематеріальний характер, масиви даних не несуть на собі ніяких відбитків, за якими можна було б судити про їхнє минуле — про те, хто є автором, про час створення, про факти, відомості про тих, хто вносить зміни. Модифікація інформаційного масиву не залишає відчутних слідів на ньому і не може бути виявлена звичайними методами. "Сліди модифікації" у тій чи іншій формі можуть бути присутніми тільки на матеріальних носіях інформації – так, як спеціальна експертиза цілком здатна встановити, що сектор *X* на деякій дискеті був записаний пізніше всіх інших секторів з даними на цій же доріжці дискети, і цей запис вироблявся на іншому дисководі. Зазначений факт, будучи встановленим, може, наприклад, означати, що в дані, збережені на дискеті, були внесені зміни. Але після того, як ці дані будуть переписані на інший носій, їхньої копії вже не будуть містити ніяких слідів модифікації. Реальні комп'ютерні дані за час свого життя багаторазово змінюють фізичну основу представлення і постійно кочують із носія на носій. Оскільки створення й використання інформаційних масивів практично завжди розділені в часі чи в просторі, у споживача завжди можуть виникнути обґрунтовані сумніви в тому, що отриманий ним масив даних створений потрібним джерелом і при тій точності такий, яким він дійшов до нього.

Таким чином, у системах обробки інформації, окрім забезпечення її таємності, важливим є гарантувати наступні властивості для кожного масиву даних, що обробляється:

- дійсність – він прийшов до споживача саме таким, яким був створений джерелом і не піддавався на своєму життєвому шляху несанкціонованим змінам;
- авторство – він був створений саме тим джерелом, яким є споживач.

Забезпечення системою обробки цих двох якостей масивів інформації і складає задачу їх аутентифікації, а відповідна здатність системи забезпечити надійну аутентифікацію даних називається її автентичністю.

На перший погляд може здатися, що дана задача визначається простим шифруванням. Дійсно, якщо масив даних шифрується із використанням стійкого шифру, такого, наприклад, як ДСТУ 28147–89, то для нього практично завжди буде справедливо наступне:

- у нього важко вносити зміни зрозумілим чином, особливо зі ступенем імовірності, що відрізняється від одиниці, факти модифікації зашифрованих масивів даних стають очевидними після їхнього дешифрування – ця очевидність виражається в тому, що такі дані перестають бути коректними для їхнього інтерпретатора: замість тексту російською мовою з'являється погана копія, архіватори повідомляють, що цілісність архіву порушена і т.д.;
- тільки ті користувачі системи, які володіють секретним ключем шифрування, можуть виготовити зашифроване повідомлення, таким чином, якщо до одержувача приходить повідомлення, зашифроване на його секретному ключі, він може бути впевненим у його авторстві, тому що крім нього самого тільки законний відправник міг створити це повідомлення.

Використання шифрування в системах обробки даних саме по собі незручне, забезпечується його автентичність по наступних причинах:

1. Зміни, які були внесені в зашифровані дані, стають очевидними після дешифрування тільки у випадку великої надмірності вихідних даних. Ця надмірність має місце, наприклад, якщо масив інформації є текстом на будь-якої людської мови. Проте, у загальному випадку ця вимога може не виконуватися – якщо випадкова модифікація даних не робить їх неприпустимими для інтерпретації. Якщо говорити мовою криптології, то автентичність і таємність несуть різні властивості криптосистем. Якщо більш просто: властивості систем обробки інформації забезпечують таємність і дійсність даних, що обробляються, у загальному випадку можуть не збігатися.

2. Факт успішного дешифрування зашифрованих даних за допомогою секретного ключа може підтвердити їхнє авторство тільки для самого одержувача. Третя сторона не зможе зробити на підставі цього однозначного висновку про авторство масиву інформації, тому що його автором може бути кожний із власників секретного ключа, а їх як мінімум два – відправники й одержувач. Тому в даному випадку суперечки про авторство повідомлення не можуть бути дозволені незалежним арбітражем. Це важливо для тих систем, де між учасниками немає взаємної довіри, що дуже характерно для банківських систем, що зв'язані із керуванням над значними цінностями.

Таким чином, існування проблеми підтвердження дійсності й авторства масивів даних, окремої від задачі забезпечення їхньої таємності, не викликає сумніву.

2.3 Оцінка стійкості хеш-функції

Є багато методів, що дозволяють підвищити крипостійкість системи, серед них блок хешування паролів – метод, що дозволяє користувачам запам'ятовувати не 128 байт, тобто 256 цифр ключа, а деяке осмислене вираження певної величини, чи послідовності символів, що називається паролем. При розробці будь-якого криптоалгоритму потрібно враховувати, що в більшості випадків кінцевим користувачем системи є людина, а не автоматична система. Тоді виникає питання про те чи це є зручно, і взагалі чи реально людині запам'ятати 128-бітний ключ (32 цифри). Проте, межа запам'ятовування лежить в межах від 8-12 подібних символів. Тому, якщо змушувати користувача оперувати самим ключем, тим самим він буде змушений записати його на якому-небудь листку і електронному носії, наприклад, у текстовому файлі. Це різко знижує захищеність системи [8].

Для вирішення цієї проблеми були розроблені методи, що перетворюють

вимовний, осмислений рядок довільної довжини — пароль, у зазначений ключ заздалегідь заданої довжини. У переважній більшості випадків для цієї операції використовують хеш-функцію (від англ. hashing – дрібна нарізка і перемішування). Хеш-функцією називається таке математичне чи алгоритмічне перетворення заданого блоку даних, що має наступні властивості:

1) хеш-функція має нескінченну область визначення,

2) хеш-функція має кінцеву область значень,

3) вона необоротна,

4) зміна вхідного потоку інформації на один біт змінює близько половини всіх бітів вихідного потоку, тобто результату хеш-функції.

Ці властивості дозволяють подавати на вхід хеш-функції паролі, тобто текстові рядки довільної довжини на будь-якій мові і, обмеживши область значень функції діапазоном $0..2^N-1$, де N – довжина ключа в бітах, що дозволяє одержувати на виході достатньо рівномірно розподілені по області значення блоків інформації – ключі.

Вище зазначені вимоги до хеш-функції виконують блокові шифри. Це вказує на один із можливих шляхів реалізації стійких хеш-функцій – проведення блокових криптоперетворень над рядком-пароля. Цей метод і використовується в різних варіаціях практично у всіх сучасних криптосистемах. Матеріал “рядка-пароля” багаторазово послідовно використовується як ключ для шифрування деякого заздалегідь відомого блоку даних – на виході отримуємо зашифрований блок інформації, який однозначно залежний тільки від пароля і, при цьому має досить позитивні статистичні характеристики. Такий блок і використовується як ключ для подальших криптоперетворень.

Характер застосування блокового шифру для хешування визначається відношенням розміру блоку, що використовується для даного криптоалгоритму і розрядністю необхідного хеш-результату.

Якщо зазначені вище величини збігаються, то використовується схема блокового шифрування. Первісне значення хеш-результата H_0 встановлюється

рівним 0, весь рядок-пароль розбивається на побайтові блоки, які рівні по довжині ключу, що використовується для хешування блокового шифру, потім здійснюються перетворення за рекурентною формулою:

$H_j=H_{j-1}$ XOR EnCrypt(H_{j-1},PSW_j),

де EnCrypt(X,Key) — блочний шифр, що використовується.

Останнє значення H_k використовується як шуканий результат.

На рисунку 2.1 зображено розробку хешування, де f – функція хешування, X_i – текст, IV –ініціалізуючий вектор (ключ), D – шифротекст.

У тому випадку, коли довжина ключа, приблизно в два рази перевищує довжину блоку, а подібна залежність досить часто зустрічається в блокових шифрах, використовується схема, що нагадує мережу Файстеля [6]. Характерним недоліком хеш-функції, що базується на мережі Файстеля, є велика потреба у збільшенні розміру ключа.

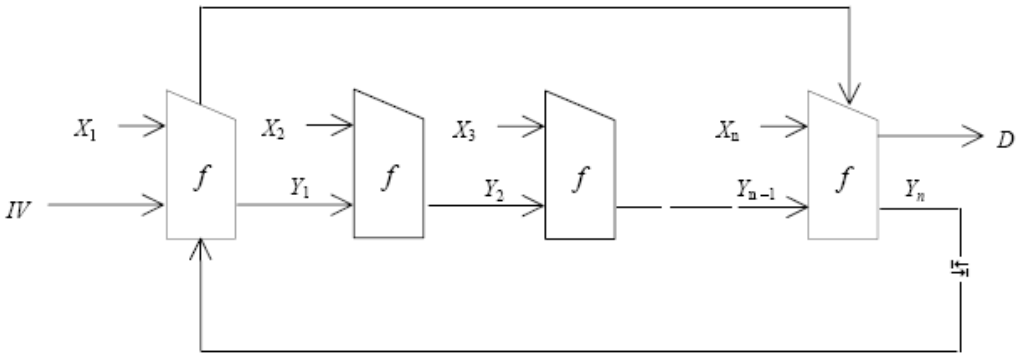

Рисунок 2.1 — Хешування повідомлення з використанням ключа IV

Для проведення тільки одного перетворення, наприклад, блоковим шифром із ключем довжиною 128 біт використовується 16 - байтний рядок-пароля, а сама довжина пароля рідко перевищує 32 символів. Отже, при обчисленні хеш-функції, над паролем будуть здійснюватися максимум 2 "повноцінних" криптоперетворення.

Цю проблему можна вирішити двома шляхами:

1) попередньо "розмножити" рядок-пароль, наприклад, записавши його

багаторазово послідовно до тих пір, поки не буде досягнено необхідної довжини, наприклад, у 256 символів;

2) модифікувати схему використання криптоалгоритму так, щоб матеріал рядок-пароля "повільніше" витрачався при обчисленні ключа.

Оскільки, традиційне шифрування забезпечує аутентифікацію і таке шифрування можна здійснити за допомогою вже доступних засобів, що знаходяться у широкому використанні. Є ряд програм, в яких одне і те ж повідомлення може передаватися кільком адресатам. Прикладом може бути повідомлення користувача про те, шо мережа в даний момент є недоступною, або сигнал тривоги, що надходить із центрального військового штабу. Дешевше і надійніше мати одного адресата, що відповідає за перевірку автентичності. При такому підході повідомлення передається у відкритому вигляді разом із відповідним кодом автентичності. Система, що відповідає за перевірку автентичності повинна мати секретний ключ і виконувати таку перевірку. Якщо буде знайдено порушення, інші системи-адресати попереджаються про небезпеку сигналом загальної небезпеки.

Дослідники Девіс і Майєр цю проблему розглянули по іншому. Вони запропонували алгоритм також на основі блокового шифру, але матеріал в якості рядка-пароля, що використовується є невеликим [9]. У ньому проглядаються елементи, які характерні і для вище описаних (рисунок 2.2), але крипостійкість цього алгоритму підтверджена численними реалізаціями в різних криптосистемах. Алгоритм одержав назву "Tandem DM":

$G_0=0$; $H_0=0$;

FOR J = 1 TO N DO

BEGIN

TMP=EnCrypt(H,[G,PSW_j]); H'=H XOR TMP;

TMP=EnCrypt(G,[PSW_j,TMP]); G'=G XOR TMP;

END;

Key=[G_k,H_k]

Захист цілісності повідомлень, побітове шифрування потоку даних — це системи шифрування, які найбільш вразливі до атак на зміну вихідного тексту. Якщо вихідний текст частково відомий зловмиснику, то модифікація бітів тексту реалізується дуже просто, шляхом інвертування бітів шифрованого тексту в тих місцях, де потрібна інверсія бітів вихідного. Така модифікація цілком реальна над фінансовим повідомленням, що включає, наприклад, номінали, число і номер купюр.

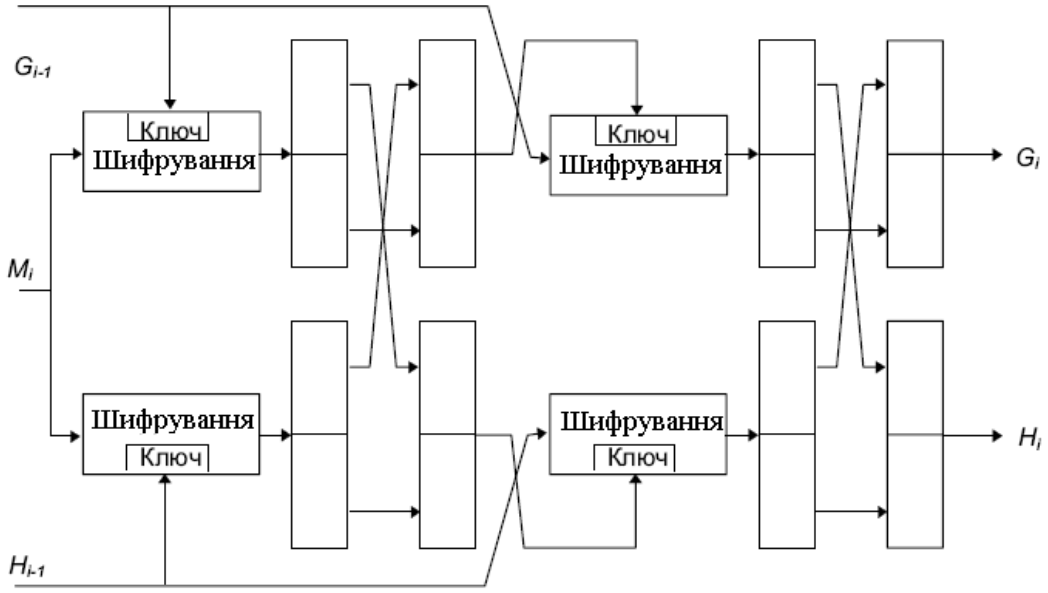

Рисунок 2.2 — Алгоритм хешування на основі блочних шифрів

Якщо повідомлення містить тільки кілька контрольних знаків, вони можуть бути також змінені, оскільки всі біти, що беруть участь в обчисленні цих знаків, відомі. Це виявляється можливим незалежно від виду функції перевірки надмірності (лінійна чи нелінійна). Звичайно, зловмиснику необхідно знати цю функцію, і він може її знати, оскільки функція перевірки не є секретною. Це означає, що зашифровані контрольні знаки, що використовуються в звичайних протоколах передачі по лініях зв'язку, — символ контролю блоку BCC (Block Check Character) і контроль за допомогою циклічного надлишкового коду CRC (Cyclic Redundancy Check), не можуть допомогти одержувачу виявити маніпуляції з

повідомленням, оскільки зловмисник може легко обчислити їхнє значення. Варто зазначити, що підтвердження дійсності повідомлення в таких системах вимагає спеціального механізму з секретними елементами, наприклад, кодом дійсності MAC (Message Authentication Code). Побітове шифрування потоку зі зворотнім зв'язком шифрування. Для багатьох реалізацій таких систем характерне явище безупинного поширення помилки, яке означає, що зловмисник не в змозі контролювати вихідний текст, що буде відновлюватися після навмисної зміни хоча б одного біта. Виключення складає ситуація, коли змінюваним є останній біт повідомлення.

Усі види контролю на надмірність будуть працювати як засіб виявлення помилкових повідомлень. В інших випадках можливо обмежене поширення помилки, наприклад, у межах 64 біт у випадку DES-алгоритму. Поки виконується контроль по надмірності, що може виявити зміни принаймні в кожному 64-му біті, наприклад, з використанням символу контролю блоку BCC, то цього є достатньо, оскільки зловмиснику нічого не залишається, окрім варіанту випадкового вгадування.

Існують реалізації, коли поширення помилки не виходить за межі одного біта. Не знаючи, як біт зворотнього зв'язку впливає на дешифрування наступного біта, зловмисник вважає, що події "біт змінений" і "біт не змінений" мають однакову імовірність (50%) [10]. Якщо наприкінці повідомлення використовується символ контролю блоку BCC, то зловмисник може модифікувати його. Вплив зворотнього зв'язку і модифікації символу контролю блоку BCC на наступний блок однаковий і складає як і раніше 50%. Кожна зміна біта в одному чи декількох символах BCC зменшує ефективність успішного контролю на 50%. Контроль за допомогою циклічного надлишкового коду CRC або іншого нелінійного методу, при умові, що зміна одного біта впливає на кілька контрольних символів, створює для зловмисника несприятливу ситуацію.

У цьому випадку найкращою стратегією для зловмисника було б виявити ті зміни, що зачіпають лише обмежене число бітів надлишкового коду CRC. DES-

алгоритм у режимі 8-бітового шифрування зі зворотним зв'язком має ті ж властивості, що й у режимі побітового шифрування. Можливо, 64-бітове шифрування з ОС має істотно інші властивості. Оскільки в цьому випадку побітове шифрування поширюється на кожен 64-бітовий блок, можливі маніпуляції на рівні блоку чи вставки або видалення блоку можуть бути виявлені. При відомому вихідному тексті можна виконати зміну останнього блоку, що містить значення контрольних функцій (символу контролю блоку BCC чи циклічного надлишкового коду CRC), щоб змінити значення контрольної функції. Додавання блоків за модулем 2 або додавання символу контролю блоку BCC не змінюють значення контрольної функції, якщо зловмисник тільки переставляє блоки повідомлення місцями.

У випадку побітового шифрування зі зворотнім зв'язком за вихідним текстом поширення внесених помилок залежить від того, як біти повідомлення впливають на роботу генератора випадкових чисел. Якщо цьому впливу піддається тільки наступний біт, то імовірність правильного дешифрування зменшується на 50% посилання кожного неправильного дешифрованого біта. Таким чином, дешифрування буде правильним тільки за умови, коли не вводяться помилки. Це означає, що зловмисник не в змозі контролювати повною мірою всі зміни, які він вводить, і це, звичайно, справедливо стосовно модифікації контрольних символів. Навіть при використанні найпростішої контрольної функції — символу контролю блоку BCC - він буде мати 50% -ний шанс на успіх. Використання більш складних функцій контролю істотно знижує ефективність вторгнення.

Проте, існує можливість, що такі системи мають більш широку область поширення помилки (і навіть необмежену). Якщо існують функції перевірки для вихідного тексту, то введення в нього нерозпізнаних змін неможливо. Нелінійні властивості процедур поблочного шифрування не дозволяють зловмиснику модифікувати блок вихідного тексту (чи байт-символ), навіть якщо йому відомо саме вихідне повідомлення. Оскільки, зміни вихідного тексту в результаті поблочного шифрування прочитати неможливо, то зловмисник не знає, як

зміняться контрольні цифри, але навіть якщо він знає, то не може здійснити бажаних змін. В результаті такі системи забезпечують високий степінь захисту від модифікацій.

У випадку поблочного шифрування потоку зі зворотнім зв'язком (ЗЗ) область поширення помилки буде складати наступний блок, а в багатьох системах і значно більше. Степінь захисту від можливих модифікацій буде вища, ніж у попередньому випадку. Область поширення помилки обмежена розмірами блоку шифрованого тексту, проте передбачати ефекти змін всередині блоку неможливо. Проте незалежність блоків дозволяє проводити маніпуляції на рівні блоку.

Будь-який криптографічний алгоритм може бути реалізований у вигляді відповідної програми. Програмними засобами захисту інформації називаються спеціально розроблені програми, що реалізують функції безпеки обчислювальної системи, здійснюють функцію обмеження доступу користувачів по паролях, ключах, багаторівневому доступу і т.д. Як правило, ці програмні засоби забезпечують досить високий степінь захисту системи і мають помірні ціни. При підключенні такої системи в глобальну мережу імовірність злому захисту збільшується.

Отже, цей спосіб захисту був прийнятий для локальних замкнутих мереж операційних систем сімейства Windows, що не мають зовнішнього виходу. Переваги такої реалізації очевидні: програмні засоби шифрування легко копіюються, вони прості у використанні, їх неважко модифікувати відповідно до конкретних потреб. Швидкодія в них достатня для забезпечення потреб локальної мережі.

3 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Вибір мови програмування

Для розробки програмного комплексу було обрано мову програмування Borland C++, яка відрізняється якістю генерації програмного коду, що забезпечує швидке виконання створених з її допомогою програм, хорошу переносимість їх та, при всіх вищезгаданих перевагах, невеликі розміри. Також Borland C++ дозволяє створювати програми з використанням об’єктно-орієнтованого програмування, яке є одним з найновітніших досягнень у розвитку мов програмування високого рівня.

Об’єктно-орієнтоване програмування (ООП) — це програмування, що сфокусоване на даних, при чому дані і поведінка тісно пов’язані між собою. Разом дані і поведінка являють собою клас, а об’єкти є екземплярами класу. Наприклад, многочлен має область значень, і вона може змінюватися такими операціями, як додавання і множення многочленів.

ООП розглядає обчислення, як моделювання поведінки. Те, що моделюється, є об’єктами, що представлені обчислювальною абстракцією. В ООП об’єкти відповідають за свою поведінку. Наприклад, многочлени, комплексні числа, цілі числа, числа з плаваючою комою — все це об’єкти, що “розуміють” операцію додавання. Кожен з цих типів включає в себе код для виконання операції додавання.

До поняття ООП має відношення ціла група концепцій, яка включає наступні:

- моделювання дій з реального світу;
- наявність типів даних, які описує користувач;
- приховування деталей реалізації;
- можливість багатократного використання коду завдяки наслідуванню;
- інтерпретація викликів функцій на етапі виконання.

Деякі з цих понять досить розмиті, деякі — абстрактні, інші мають

загальний характер.

Об’єктно-орієнтована парадигма пропонує новий підхід до розробки програмного забезпечення, яке призначене для розв’язку задач різних класів. Фундаментальна концепція об’єктно-орієнтованої парадигми полягає в передачі повідомлень об’єктам. Для цього необхідно, щоб об’єкти описувалися разом з повідомленнями, на які вони будуть реагувати, на відміну від процедурного програмування, де спочатку визначається структура даних, котрі будуть передаватися в процедури як параметри.

Існує п’ять компонентів об’єктно-орієнтованої парадигми: об’єкт, повідомлення, клас, наслідування і метод. Ці компоненти дуже сильно залежать одне від одного, причому кожен з них визначається в термінах інших. Для того, щоб зрозуміти один компонент, необхідно зрозуміти їх всі. Об’єктно-орієнтована мова програмування повинна володіти властивостями абстракції, інкапсуляції, наслідування і поліморфізму. Окрім того, об’єктно-орієнтованій мові бажано мати можливості розширення, повторного використання програмних компонент, а також параметризації [9].

Об’єктно-орієнтоване програмування дозволяє вирішувати проблеми будь-якого рівня складності. Це забезпечується шляхом розбиття однієї складної задачі на велику кількість простих, які тісно пов’язані між собою. Якщо програміст буде впевнений у правильності вирішення всіх простих задач (а цю правильність для простої задачі легше перевірити, ніж для складної), а також у коректності зв’язків між простими об’єктами, він може бути впевненим у коректності розв’язку глобальної (основної) задачі.

Компілятор Borland C++ надає потужну базу для розробки об’єктно-орієнтованих програм. Надзвичайно потужний синтаксис та оптимальний код, створюваний компілятором роблять його використання ефективним для вирішення більшості задач, що постають перед програмістами сьогодні.

Надійність і швидкість обробки інформації програмою багато в чому залежать від якості проектування методів доступу до даних системи і зв’язків між

окремими інформаційними одиницями.

C++ — це тісний союз програмування на низькому і високому рівні. С був розроблений як системна мова, близька до машинної. C++ доповнена об'єктно-орієнтованими властивостями, які дозволяють програмісту створювати чи імпортувати бібліотеки, які притаманні для конкретної задачі. Користувач може написати код на проблемному рівні, в той же час підтримуючи контакт з машинним рівнем реалізації деталей.

Вибрана мова програмування спрощує розв'язання задачі, дозволяючи зосередитися на даних, проте в будь-який момент можна повернутися до машинного коду. Це дуже важливий аспект, враховуючи складність поставлених задач.

3.2 Опис основних модулів програмного комплексу

Для реалізації модифікованої хеш-функції MD5 були підключені методи інтерфейсу CryptoAPI. Криптографічний інтерфейс прикладного програмування (CryptoAPI) — це набір функцій, які можна викликати з програм на мові С. Він дозволяє використовувати криптографічні функції для побудови власної програми. CryptoAPI передбачає три набори функцій:

— сертифікаційні функції;

— спрощені криптографічні функції;

— базові криптографічні функції.

Сертифікаційні функції — це засоби вилучення, збереження і перевірки сертифікатів цифрових підписів, які можуть супроводжувати ряд документів і сертифікати, що зберігаються на комп'ютері [10].

Спрощені криптографічні функції включають в себе функції високого рівня для створення ключів, шифрування та дешифрування інформації.

Базові криптографічні функції розташовані на нижчому рівні, їх бажано уникати, оскільки їхній виклик може призвести до конфліктів, що пов’язані з видаленням із системи деяких компонентів.

Програмний пакет CryptoAPI складається з двох рівнів:

1) рівень зв’язку з програмами, що призначений для виклику функції API;

2) рівень доповнень (розташовується після першого рівня) є відкритим і призначений для поновлення і включення нових функцій у вигляді вбудованих компонентів (plug in), в тому числі і тих, що розробляються програмістами.

Структура CryptoAPI зображена на рисунку 3.1.

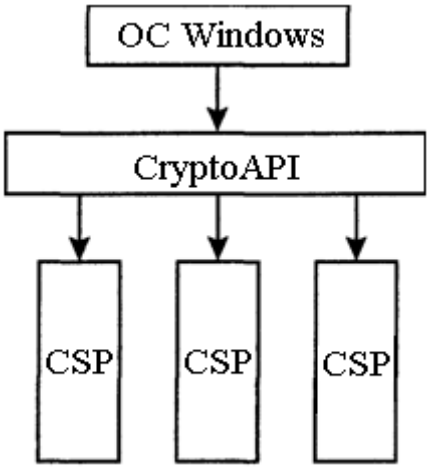

Рисунок 3.1 — Структура інтерфейсу CryptoAPI

CryptoAPI забезпечує передачу повідомлень між програмами і провайдерами служби шифрування. Вбудований компонент такого типу, що реалізує алгоритму шифрування називається провайдером служби шифрування — Cryptographic Service Provider (CSP) або криптографічним сервером. В зображеній структурі (див. рисунок 3.1) використовуються функції передачі повідомлення між програмами і вибраними для операційної системи Windows програми криптосервера CSP.

Криптографічний сервер (CSP) — це сервер, що може виконувати стандартний набір задач, що ініціюються викликом функцій CryptoAPI. Типовий

CSP (Cryptographic Service Provider) складається з DLL і файлу підписів, що використовує CryptoAPI для перевірки цілісності та ідентифікації користувачів, що звертаються до сервера. Таким чином, CSP — це змінний модуль програмного забезпечення. Він нагадує драйвери ОС Windows. Можливість використання CSP виникає одразу після його реєстрації, при цьому не потрібно вносити зміни на рівні ОС.

Для того, щоб забезпечити конфіденційність, всі дані, якими маніпулює CSP (особливо ключі), повертаються у викликану програму у формі дескрипторів і залишаються недоступними на рівні ОС. Окрім того, програмне забезпечення не впливає на метод криптоперетворення даних. Роль програми обмежується передачею даних серверу і вказанням необхідного типу перетворення. Різні сервери можуть використовувати однакові алгоритми, але при цьому приймають різну логіку заповнення і різні розміри ключів [11].

CryptoAPI використовує базу даних ключів. Зберігання ключів (key store) — це база даних для зберігання ключів ініціалізуючого вектора, що обслуговується компонетном CSP. Ключі, що зберігаються в ній, використовуються в процесі виклику програми. Створення бази даних ключів гарантує функціональність інтерфейсу CryptoAPI.

Кожний криптосервер (CSP) має асоціативну базу даних для ключових контейнерів, яка містить всі приватні і загальні ключі користувачів, що мають доступ до даного комп'ютера. Модель бази даних ключів забражено на рисунку 3.2.

Кожний ключовий контейнер має унікальне ім'я. Без бази даних CryptoAPI будь-які виклики CryptoAPI будуть відхилені. База даних, по замовчуванню, містить контейнер з реєстраційними іменами всіх користувачів. Проте, конкретна програма під час встановлення створює спеціальний ключовий контейнер і пари ключів, присвоюючи їм власне ім'я. Оскільки, тип сервера впливає на алгоритми застосування криптографічних функцій, пов'язані між собою програми повинні користуватися одним CSP.

Рисунок 3.2 — Модель бази даних CryptoAPI

Фрагмент опису функції, що використовує контейнери ключів виглядає наступним чином:

```
CryptAcquireContext(phProv      :PHCRYPTPROV;
        pszContainer :LPAWSTR;
        pszProvider  :LPAWSTR;
        dwProvType   :DWORD;
        dwFlags      :DWORD) :BOOL; stdcall;
```

Робота з CryptoAPI починається з виклику цієї функції, яка виконує підключення до криптопровайдера і повертає його дескриптор в параметрі phProv. Криптопровайдер — це незалежний програмний модуль (у вигляді dll), що виконує криптографічні алгоритми. В параметрі pszContainer необхідно вказувати ім’я контейнера ключів, який буде використовуватися. Варто зазначити, що кожний криптопровайдер містить базу даних, в якій зберігаються ключі користувачів. Ці ключі групуються в контейнерах. Ключові пари зберігаються для асиметричних алгоритмів, сеансові ключі не зберігаються. Тому, кожний контейнер містить ім’я і ключ, що використовується при хешуванні.

Параметр dwFlags може приймати значення 0 або одне з насптупних:

— CRYPT_VERIFYCONTEXT — прапорець, що призначений для програм,

які не повинні мати доступ до закритих ключів контейнера. Такі програми можуть звертатися тільки до хеш-функції. В цьому параметр pszContainer має дорівнювати nil;

— CRYPT_NEWKEYSET — створює новий контейнер ключів, проте самі ключі не створюються;

— CRYPT_DELETEKEYSET — видаляє контейнер разом із ключами, що в ньому зберігаються. Якщо задано цей прапорець, то підключення до криптопровайдера не відбувається і параметр phProv є невизначеним;

— CRYPT_MACHINE_KEYSET — по замовчуванню контейнери ключів зберігаються як користувацькі. Цей прапорець можна встановлювати в комбінації з іншими, що дозволяє вказати чи є контейнер машинним.

Функція повертає значення true, якщо не було помилки, в іншому випадку — false. GetLastError повертає код помилки.

Функція для створення хеш-функції виглядає наступним чином:

```
CryptCreateHash(hProv   :HCRYPTPROV;
                Algid   :ALG_ID;
                hKey    :HCRYPTKEY;
                dwFlags :DWORD;
                phHash  :PHCRYPTHASH) :BOOL; stdcall;
```

Функція створює в системі хеш-об’єкт і повертає його дескриптор в параметрі phHash. Дані, що надходять на вхід хеш-об’єкта, перетворюються і зберігаються всередині хеш-об’єкта. В параметрі hProv потрібно вказати дескриптор провайдера, що отриманий за допомогою функції CryptAcquireContext.

Параметр Algid вказує на те, що використовується модифікація алгоритму MD5. Параметр dwFlags зарезервований і дорівнює 0.

У випадку успішного виконання функція повертає значення true, в іншому випадку — false. GetLastError повертає код помилки.

Функція знищення хеш-об’єкта, створеного за допомогою CryptCreateHash, виглядає наступним чином:

CryptDestroyHash(hHash :HCRYPTHASH) :BOOL; stdcall;

В параметрі hHash вказується дескриптор хеш-об’єкта.

Наступна функція дозволяє додавати дані до об’єкта хеш-функції:

```
CryptHashData(hHash       :HCRYPTHASH;
              const pbData      :PBYTE;
                  dwDataLen   :DWORD;
                  dwFlags     :DWORD) :BOOL; stdcall;
```

Функція може викликатися кілька разів. Дані, з яких обчислюється хеш-функція, розбиті на блоки. В параметрі hHash вказується дескриптор хеш-об’єкта, створеного за допомогою CryptCreateHash. Параметр pbData містить вказівники на дані, а dwDataLen — містить розмір цих даних в байтах.

У випадку успішного виконання функція повертає значення true, в іншому випадку — false. GetLastError повертає код помилки.

Функція CryptSignHash обчислює значення підпису від значення хеша:

```
CryptSignHash(hHash       :HCRYPTHASH;
                  dwKeySpec   :DWORD;
                  sDescription :LPAWSTR;
                  dwFlags     :DWORD;
                  pbSignature  :PBYTE;
                  pdwSigLen   :PDWORD) :BOOL; stdcall;
```

В параметрі hHash вказується дескриптор хеш-об’єкта, створеного за допомогою CryptCreateHash. Параметр dwKeySpec вказує який ключ буде використовуватися для підпису. Цей параметр може приймати значення AT_SIGNATURE або AT_KEYEXCHANGE. Ключі повинні знаходитися в контейнері. Параметр sDescription містить довільну стрічку опису. Ця стрічка додається до хеша. Параметр dwFlags дорівнює 0. Параметр pbSignature вказує на буфер, куди розташовується цифровий підпис, а pdwSigLen — розмір цього буфера. Якщо розмір наперед невідомий, то можна встановити pbSignature рівним nil, і тоді в параметрі pdwSigLen отримаємо необхідний розмір буфера.

У випадку успішного виконання функція повертає значення true, в іншому випадку — false. GetLastError повертає код помилки.

Функція CryptVerifySignature здійснює перевірку підпису:

```
CryptVerifySignature(hHash        :HCRYPTHASH;
                     const pbSignature  :PBYTE;
                         dwSigLen     :DWORD;
                         hPubKey      :HCRYPTKEY;
                         sDescription :LPAWSTR;
                         dwFlags      :DWORD) :BOOL; stdcall;
```

Параметр hHash — дескриптор хеш-об'єкта, значення якого перевіряються. Параметр pbSignature — вказує на буфер, що містить підпис, dwSigLen — розмір цього буфера. Параметр hPubKey — дескриптор відкритого ключа, за допомогою якого здійснюється перевірка підпису. Відкритий ключ має відповідати закритому, який використовувався для створення підпису. Параметр sDescription і dwFlags повинні відповідати параметрам функції CryptSignHash при здійсненні підпису.

У випадку успішного виконання функція повертає значення true, в іншому випадку — false. GetLastError повертає код помилки.

Визначається також і загальний клас HashTable. Цей клас утворюється від базового класу списків List і забезпечує механізм зберігання з дуже ефективними методами доступу. Допускаються дані будь-якого типу з обмеженням, що для цього типу даних має бути визначений оператор «==». Для того, щоб порівняти ключові поля двох елементів даних, програма має перегрузити оператор «==».

Об'єкт типу HashTable — це список елементів типу T (число). В ньому реалізовані всі методи, які потребують абстрактного класу List. Програма задає розмір таблиці і хеш-функцію, перетворюючи елемент T в довге ціле беззнакове число. Опис параметру функції за допомогою типу, що заданий в шаблоні, дозволяє використовувати даний клас для хешування даних будь-якого типу:

```
struct NameRecord
{
```

```
  String name;
  int count;
};
HashTable<NameRecord> HF(101,hash);
NameRecord rec;
rec.name = "OPT_MD5";
rec.count = 1;
HF.Insert(rec);
cout << HF.ListSize();
rec.name = "Betsy";
if (HF.Find(rec)
{
  rec.cout += 1;
  HF.Update(rec);
}
else
  cerr << "Помилка: \"Ключ OPT_MD5 має бути в таблиці.\"\n;
```

Методи Insert, Find, Delete і ClearList є базовими методами обробки списків. Наприклад, стрічка HF.Insert(rec) дозволяє вставити текст OPT_MD5 в таблицю. Метод HF.Update(rec) служить для обновлення елемента, що вже знаходиться в таблиці.

Методи ListSize і ListEmpty реалізовані в базовому класі. Елемент даних current завжди вказує на останнє доступне значення даних. Він використовується методом Update і довільними класами, які повинні повертати зсилки.

Метод Insert обчислює значення хеш-функції і шукає об’єкт типу LinkedList для того, щоб перевірити чи є такий елемент даних в таблиці. Якщо є, то Insert обновляє цей елемент даних, встановлює на нього вказівник current і повертає керування. Якщо такого елемента немає в таблиці, то Insert додає його в кінець списку, встановлює на нього вказівник current і збільшує розмір списку.

Метод Find використовує хеш-функцію і переглядає список на предмет співпадіння з вхідним параметром:

```
template <class T>
int HashTable<T>::Find(T& key) {
  int hashval = int(hf(key) % NumBuckets);
  LinkedList<T>& lst = buckets[hashval];
  for (lst.Reset(); !lst.EndOfList(); lst.Next())
    if (lst.Data() == key)
    {
      key = lst.Data();
      current = &lst.Data();
      return 1;
    }
  return 0;
}
```

Якщо співпадіння знайдено, метод копіює дані в змінну key, встановлює вказівник current на відповідний вузол і повертає значення true. В іншому випадку метод повертає значення false.

Клас HashTableIterator дозволяє полегшити обробку даних в хеш-таблиці. Він здійснює перегляд даних в таблиці. Обхід елементів таблиці починається з пошуку непустого блоку в масиві списків. Знайшовши непустий блок, функція переглядає всі вузли цього списку, а потім розглядає наступний блок. Ітератор повинен бути прив’язаний до списку. В даному випадку змінній hashTable присвоюється адреса конкретного екземпляра класу HashTable. Оскільки, клас HashTableIterator є дружнім по відношенню HashTable, то він має доступ до всіх прихованих даних, включаючи масив buckets і його розмір numBuckets.

Змінна currentBucket є індексом зв’язаного списку, який переглядається в даний момент, а currBucketPtr — вказівник цього списку. Обхід кожного блоку здійснюється інтерпретатором, що вбудований в клас LinkedList.

Метод SearchNextNode викликається для виявлення наступного списку, що підлягає обходу. Переглядаються всі блоки, починаючи з cb до тих пір, поки не зустрінеться непустий блок. Змінній currentBucket присвоюється індекс цього списку, а змінній currBucketPtr — його адреса. Якщо немає пустих списків, то функція повертає значення currentBucket = -1.

Конструктор ініціалізує базовий клас Iterator і присвоює прихованому вказівнику hashTable адресу хеш-таблиці:

```
template <class T>
HashTableIterator<T>::HashTableIterator(HashTable<T>& hf):
      Iterator<T>(hf), HashTable(&hf)
{
 SearchNextNode(0);
}
```

Непустий список виявляється за допомогою виклику SearchNextNode з нульовим параметром. За допомогою метода Next здійснюється просування вперед по поточному списку на один елемент. По досягненні кінця списку функція SearchNextNode настроює ітератор на наступний непустий блок.

Розмір хеш-таблиці має бути достатньо великим, щоб в ній залишалося велике число пустих місць. Чим менша таблиця, тим більший середній час пошуку ключів в ній. Хеш-таблицю можна розглядати як сукупність зв’язаних списків. В міру того, як збільшується таблиця, збільшується кількість списків і, відповідно, середнє число вузлів в кожному списку зменшується.

3.3 Результати роботи програмного комплексу

На вхід програми подається файл, в якому міститься повідомлення різної довжини. Програма обчислює хеш функцію від вхідного повідомлення і записує у

файл. Для перевірки ідентичності хеш-функції використовуються саморозпаковуючі файли, що мають однакову хеш-функцію (додаток Г). Нижче наведені результати виконання програмного комплексу:

— використовується три етапи (PASS=3) і файл без і з повідомленям (додаток Д). На виході отримується хеш-функція довжиною 128 і 160 біт (FPTLEN):

PASS=3, FPTLEN=128:

OPT_MD5 ("") = 1BDC556B29AD02EC09AF8C66477F2A87

PASS=3, FPTLEN=160:

OPT_MD5 ("a") = 5E1610FCED1D3ADB0BB18E92AC2B11F0BD99D8ED

— використовується чотири етапи (PASS=4):

PASS=4, FPTLEN=192:

OPT_MD5 ("OPT_MD5") =
74AA31182FF09BCCE453A7F71B5A7C5E80872FA90CD93AE4

PASS=4, FPTLEN=224:

OPT_MD5 ("0123456789") =
144CB2DE11F05DF7C356282A3B485796DA653F6B702868C7DCF4AE76

— використовується п’ять етапів:

PASS=5, FPTLEN=256:

OPT_MD5 ("abcdefghijklmnopqrstuvwxyz") =
1A1DC8099BDAA7F35B4DA4E805F1A28FEE909D8DEE920198185CBCAED
8A10A8D

OPT_MD5 ("ABCDEFGHIJKLMWXYZabcdefghijklmnopqrstuvwxyz0123456
789") = C5647FC6C1877FFF96742F27E9266B6874894F41A08F5913033D9D5
32AEDDB39

Розглянемо як буде змінюватися для одного і того ж повідомлення значення хеш-функції взалежності від кількості етапів і довжини хеш-значення:

PASS=3, FPTLEN=128;

OPT_MD5

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMOPQRSTUVWXYZ01234567
89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMOPQRSTUVWXYZ0123456
789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMOPQRSTUVWXYZ012345
6789") =
ddf4304cc5ffa3db8aab60d4f8fc2a00
PASS=3, FPTLEN=160;
OPT_MD5 (“…”) = e709559359b15917623050e41d27a306c6c3a9db;
PASS=3, FPTLEN=192;
OPT_MD5 (“…”) = 51e25280ad356c06f4b913b3cdb3abaaac5879dda0a4fea4;
PASS=3, FPTLEN=224;
OPT_MD5 (“…”) =
28a2c164e10bb3076574cc8aa8584fd6d04f6d82c37ea5c21e451b3;
PASS=3, FPTLEN=256;
OPT_MD5(“…”) =
5537364e3d75174b846d21adf9b113f9d8f97e4750df64d428c01e782f9ade4d;
PASS=4, FPTLEN=128;
OPT_MD5(“…”) = c7d981e8270e39888ba96cafe8745636;
PASS=4, FPTLEN=162;
OPT_MD5 (“…”) = 3444e38cc2a132b818b554ced8f7d9592df28f57;
PASS=4, FPTLEN=192;
OPT_MD5(“…”) = 0ca58f140ed92828a27913ce5636611abcada220fccf3af7;
PASS=4, FPTLEN=224;
OPT_MD5(“…”) =
a9d0571d0857773e71363e4e9dfcca4696dba3e5019e7225e65e0cb1
PASS=4, FPTLEN=256;
OPT_MD5 (“…”) =
1858d106bdc2fc787445364a163cfc6027597a45a58a2490d14203c8b9bdd268
PASS=5, FPTLEN=256;

OPT_MD5 (“…”) =
f93421623f852ac877584d1e4bba5d9345a95f81bfd277fe36dfeed1815f83d5

З вище наведених результатів видно, що від кількості етапів залежить і хеш-функція. При одному і тому ж вхідному повідомленні значення хеш-функції від 3 до 5 етапів буде різним.

Було здійснено перевірку хеш-функції. В результаті чого файли мали ідентичні імена і однаковий вміст.

В подальшому можна досліджувати модифіковану хеш-функцію MD5 на стійкість до колізій.

4 ОХОРОНА ПРАЦІ

Поняття “охорона праці” визначено статтею 1 Закону України “Про охорону праці”. Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів і засобів, спрямованих на збереження здоров’я і працездатності людини в процесі праці [13].

Основною ціллю охорони праці є створення на кожному робочому місці безпечних умов праці, експлуатації обладнання, зменшення або повна нейтралізація дії шкідливих і небезпечних виробничих факторів на організм людини і зниження виробничого травматизму та професійних захворювань.

Мета охорони праці – надання знань щодо загальних питань законодавства з охорони праці, виробничої санітарії, пожежної безпеки, електробезпеки, гігієни праці, способів надання першої допомоги потерпілим при нещасних випадках, аваріях.

4.1 Аналіз санітарно-гігієнічних умов праці

У даному розділі розглядаються питання по охороні праці при розробці програмного комплексу перевірки достовірності інформації для операційної системи Windows XP.

Даний програмний комплекс розроблявся на базі інженерно-технічного відділу фірми “Polyservice”. У відділі постійно працює до восьми чоловік. Види виконуючих робіт – вивчення матеріалів у відповідності з розвитком техніки, розробка, налагодження і запуск програм на комп’ютерах відділу. Для цих цілей у відділі використовується 9 персональних ЕОМ класу Celeron 1700, а також

лазерний принтер формату А4 марки Canon LBP-810. На персональних комп'ютерах (ПК) встановлені монітори марки SAMSUNG SyncMaster 755 DFX (розширення 1024x768).

План інженерно-технічного відділу подано на рисунку 4.1.

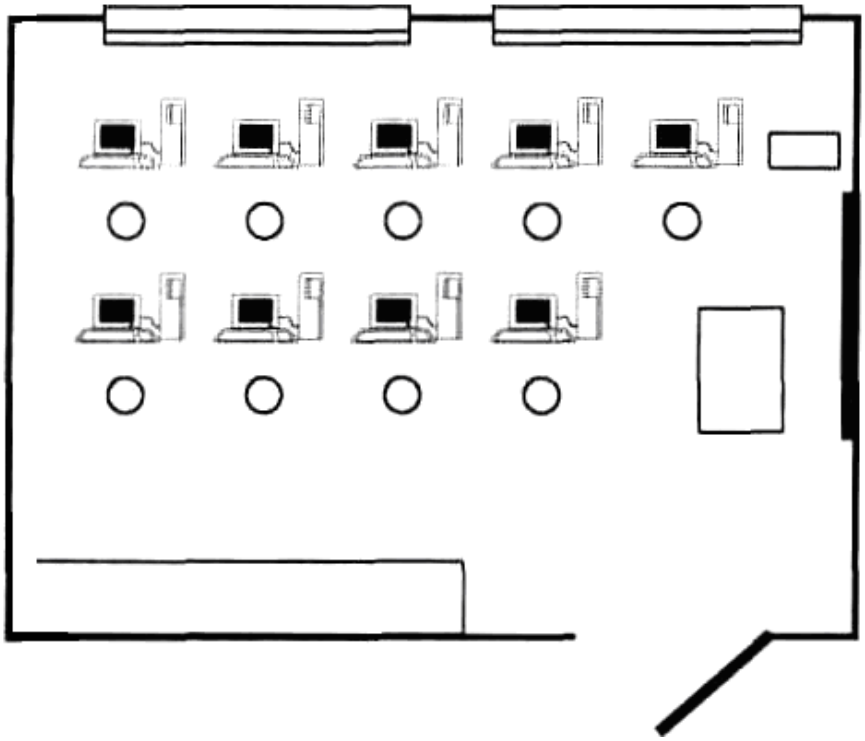

Рисунок 4.1 – План інженерно-технічного відділу

Розміри відділу описуються в таблиці 4.1.

Таблиця 4.1 − Розміри відділу

Позначення	Визначення	Значення
I	Довжина	9 м
d	Ширина	7 м
h	Висота	4 м
S_0	Площа	63 м^2
V_0	Об'єм	252 м^3

Згідно СН-245-71, на одного працюючого об’єм приміщення повинний складати не менше 19,5 м^2, площа – не менше 6 м^2. Число працюючих у приміщенні Np=9. Таким чином, на одне робоче місце приходиться площа S=63/9=7 (м^2) і об’єм V = 252/9=28 м3. Ці значення відповідають вимогам.

Крім того, повинні дотримуватися норми приміщення, подані у таблиці 4.2.

Таблиця 4.2 – Норми приміщення

Параметр	Значення
ширина основних проходів	≥1200 мм
ширина допоміжних проходів	≥700 мм
відстань між двома столами, якщо між ними є стілець	≥1300 мм

У розглянутому приміщенні відділу: відстань між двома столами становить 1500мм, відстань між двома столами в ряді − 1500мм, а між рядами − 2000 мм.

Отже, норми виконуються.

У технічних умовах роботи ЕОМ вказуються робочі діапазони параметрів мікроклімату:

− температура повітря 5 − 45 °С,

− відносна вологість повітря 40 − 90 %.

Однак, вимоги точного регулювання параметрів повітряного середовища приміщення значно звужують ці діапазони.

З метою забезпечення комфортних умов для персоналу, а також максимальної безвідмовності функціонування техніки, встановлюють вимоги до повітряного середовища приміщень [14]. Так, у відділі повинні бути:

− температура повітря 18 – 22 °С,

− відносна вологість повітря 50 – 60 %,

− атмосферний тиск 1013 – 1013,5 гПа.

При зниженні тиску погіршується відвід тепла від елементів ЕОМ,

знижуються ізоляційні властивості повітря.

Як було показано вище, показники об'єму і площі приміщення на одного працюючого відповідають нормативним значенням.

Роботи, що проводяться в інженерно-технічному відділі відносяться до легких фізичних робіт групи 1а, відповідно до ГОСТ 12.1.005-88, оскільки вони проходять сидячи і не вимагають фізичного навантаження, здійснюються при нормальних метеорологічних умовах і не викликають забруднення одягу і рук. Витрати енергії не перевищують 172 Дж/с (155 ккал/год).У таблиці 4.3 і таблиці 4.4 наведені норми температури, відносної вологості і швидкості руху повітря на робочих місцях відповідно до ГОСТ 12.1.005-88, що встановлює норми виробничого мікроклімату. Дані приведені для приміщень з незначним надлишком явного тепла (до 20 ккал/год м^3) для виконання легких робіт.

Таблиця 4.3 - Норми температури, відносної вологості і швидкості руху повітря на постійних робочих місцях

Період року	Норми	Температура повітря *t*, ^{0}C	Відносна вологість, %	Швидкість руху повітря, м/с
Холодний	оптим.	22-24	40-60	менше, ніж 0,1
	доп.	21-25	менше 75	менше, ніж 0,1
Теплий	оптим.	23-25	40-60	0,1
	доп.	22-28	менше, ніж 55	0,1-0,2

Таблиця 4.4 − Відносна вологість повітря в теплий період року

Температура повітря, ^{0}C	28	27	26	25	24	≤23
Відносна вологість, %	≥55	60	65	70	75	75

Основними джерелами тепла у відділі є:

− сонячна радіація,

− система опалення,

− люди, що працюють у приміщенні,

− устаткування.

У таблиці 4.5 приведені дані, виміряні в інженерно-технічному відділі у лютому місяці.

Таблиця 4.5 − Результати виміру параметрів мікроклімату у відділі

Параметр	Значення
Температура повітря t, °C	17 – 20
Відносна вологість, %	50 – 60
Швидкість руху повітря, м/с	0,2

Як видно з таблиці 4.5, у розглянутому приміщенні відділу значення параметрів мікроклімату відповідають нормативним. Постійність цих параметрів підтримується загальною системою утеплення і кондиціонування повітря. При цьому використовується кондиціонер SAMSUNG AQT-24A5RE, а при необхідності здійснюється провітрювання приміщення. У таблиці 4.6 зображено параметри кондиціонера SAMSUNG AQT-24A5RE. Він забезпечує встановлені норми мікроклімату у відділі.

Таблиця 4.6 − Параметри кондиціонера SAMSUNG AQT-24A5RE

Параметр	Значення
Потужність охолодження	6.8 кВт
Продуктивність охолодження	24 000 БТЕ/год
Потужність обігріву	6,9 кВт
Продуктивність обігріву	24 000 БТЕ/год
Видалення вологи з повітря	3 л/год
Циркуляція повітря	14 $м^3$/хв

Джерелами пилу в інженерно-технічному відділі є: книги, документація, роздруківки, а також одяг, взуття працівників і зовнішнє повітря.

Встановлений у відділі кондиціонер SAMSUNG AQT-24A5RE забезпечує встановлені норми чистоти поступаючого зі сторони приміщення повітря, що надходить ззовні. У відділі періодично проводиться вологе прибирання. Зазначені умови забезпечують підтримку в нормі параметрів чистоти повітряного середовища.

У відділі використовується природне і штучне освітлення. Природне освітлення здійснюється з допомогою трьох вікон загальною площею S= 42 м^2 , що забезпечує коефіцієнт природної освітленості Е=1,5%. Це відповідає СНіП І-4-79 [12].

Штучне освітлення у відділі здійснюється системою загального рівномірного освітлення, що реалізована на основі люмінесцентних ламп типу ЛДЦ-40-1, що мають наступні параметри:

− висока світловіддача;

− тривалий термін служби;

− мала яскравість освітлювальної поверхні;

− близькість спеціального складу до природного освітлення.

Робота за монітором ПЕОМ по розряду зорових робіт відноситься до III типу (роботи високої точності з розміром об’єкта 0,2-0,4 мм). При загальному освітленні, освітленість робочого місця повинна складати від 200 до 400 лк.

При штучному освітленні нормуються наступні параметри:

− Е (лк) – найменша припустима освітленість;

− М – показник дискомфорту;

− $К_п$ (%) – коефіцієнт пульсації освітлення;

Номінальний світловий потік лампи білого свічення ЛДЦ-40-1: $Ф_л$=3120 лм.

У відділі застосовуються світильники, у яких встановлені дві лампи. Висота підвіски світильника визначається за формулою:

$$h = H - h_C - h_P - h_П, \quad (4.1)$$

де H – висота приміщення (м),

h_C – висота світильника (м),

$h_П$ – відстань від стелі до підвіски (м),

h_P – висота робочої поверхні (м).

Для розглянутого відділу:

H = 4 м,

h_C = 0,15 м,

$h_П$ = 0 м, (підвісу немає)

h_P = 0,8 м.

Звідси h = 4 – 0,15 – 0,8 = 3,05(м).

Світильники розташовані в 3 ряди. Висота підвіски світильників складає 3,05 м відносно підлоги, відстань між рядами 1 м, відстань від ряду до стіни 1,5 м. Приміщення має наступні габарити:

− довжина A = 9 м,

− ширина B = 7 м.

Визначимо освітленість у робочій точці. Для розрахунку загальної рівномірної освітленості при горизонтальній робочій поверхні використовуємо метод коефіцієнта використання світлового потоку.

Розрахункова формула для світлового потоку світильника має такий вигляд:

$$Фл = \frac{E \cdot Кз \cdot S \cdot Z}{N \cdot n}, \quad (4.2)$$

де N − кількість світильників у відділі (N = 6 3=18);

n − коефіцієнт використання світлового потоку;

$Фл$ − світловий потік ламп;

$Кз$ − коефіцієнт запасу ($Кз$ = 1,5);

Z − коефіцієнт нерівномірності;

S − площа приміщення;

E − освітленість, створювана усіма світильниками.

Звідси одержуємо формулу для розрахунку освітленості на робочому місці:

$$E = \frac{Фл \cdot N \cdot n}{Кз \cdot S \cdot Z} \tag{4.3}$$

Коефіцієнт використання світлового потоку залежить від:

− КПД кривої розподілу сили світла світильника;

− коефіцієнта відбивання стелі $R_П$ і стін $R_С$;

− висоти підвісу світильників $h_П$;

— показника приміщення i:

$$i = \frac{A \cdot B}{h \cdot (A + B)} \tag{4.4}$$

Тобто i = (9•6)/(3,05•(9+6)) = 1,18.

Стеля і стіни пофарбовані в білий колір.

Приймаємо $R_П$ = 50%, $R_С$ = 30%.

Звідси n = 31%,

$$E = \frac{(3120 \cdot 2) \cdot 18 \cdot 0{,}31}{63 \cdot 1{,}1 \cdot 1{,}5} = 335 лк.$$

Оскільки по розряду зорової роботи робота за дисплеєм ПЕОМ відноситься до III типу (високої точності, розмір об'єкта 0.2-14 мм), то при загальному висвітленні освітленість робочого місця повинна складати від 200 до 400 лк, рекомендована освітленість при роботі з дисплеєм ПЕОМ складає 200 лк, а при сполученні роботи з документами 400 лк. Фактична освітленість на робочому місці складає 335 лк. Таким чином для роботи з дисплеєм цілком достатньо

існуючих джерел світла, однак робота з документами повинна вестися при природному освітленні або за допомогою додаткових місцевих джерел освітлення.

У відділі основними джерелами шумів є: вентилятори системи охолодження ПЕОМ, кондиціонер і друкуючий пристрій. Згідно ГОСТ 12.1.003-83, нормованою шумовою характеристикою робочих місць при постійному шумі являються рівні звукових тисків у децибелах в октавних смугах. Сукупність таких рівнів називається граничним спектром (ГС), номер якого дорівнює рівню звукового тиску в октавній смузі із середньо-геометричною частотою 1000 Гц. В таблиці 4.7 приведені значення звукового тиску у відділі при роботі принтера.

Таблиця 4.7 − Рівні звукового тиску в дБ на робочих місцях

	Середньо-геометричні смуги октавних частот							
	63	125	250	500	1000	2000	4000	8000
норма	71	61	54	49	50	42	40	38
лазерний принтер	56	56	58	61	57	60	60	60

Для розрахунку еквівалентного рівня шуму, створеного у відділі принтером і працюючими ПЕОМ, використаємо формулу:

$$L_{EKB} = \frac{1}{T} \cdot 10 \cdot \lg\left(\sum 10^{0,1 \cdot 1,1}\right). \qquad (4.5)$$

Враховуючи, що $L_{принтера}$ = 60 дБ і принтер працює в середньому 1 год в день, а $L_{ПЕОМ}$ = 35 дБ і ПЕОМ працює кожен робочий день, еквівалентний шум складає:

$$L_{EKB} = \frac{1}{8} \cdot 10 \cdot \lg\left(0^{0,1 \cdot 60 \cdot 1} + 10^{0,1 \cdot 35 \cdot 8}\right) = 35,78 дБ.$$

По ГОСТ 12.1.003-83 $L_{допустиме}$ = 50 дБА, відповідно рівень шуму у відділі не перевищує норму.

Потенційну небезпеку для людини представляють електричні прилади і устаткування. Ураження людини електричним струмом може відбутися в результаті дотику до відкритих струмопровідних частин при ушкодженні ізоляції мережевих шнурів, при пробої, при короткому замиканні або в результаті необережних дій самої людини.

Даний відділ по ступені небезпеки ураження електричним струмом відноситься до приміщень без підвищеної небезпеки. Споживачами електроенергії являються ПЕОМ, принтери і джерела освітлення.

Корпуси сучасних ПЕОМ виготовляються із пластмас (передня панель) і металу (верхня кришка і задня панель). При дотику до металевих частин корпуса ПЕОМ у випадку пробою на корпус людина може потрапити під небезпечну для життя напругу. Для цього в конструкції ПЕОМ передбачене спеціальне електричне з'єднання з нульовим захисним провідником металевих частин корпусу, що можуть виявитися під напругою. Для цього в ПЕОМ застосовується спеціальна мережева вилка з трьома контактами (два контакти призначені для підключення живлення, а третій – для підключення до зануленого проводу).

Корпуса моніторів виготовляються з непровідних матеріалів, а живлення здійснюється спеціальним кабелем, що підключається до системного блоку ПЕОМ так, щоб виключити ураження людини електричним струмом.

Корпуса сучасних принтерів також виготовляються з пластмас, а конструкція кабелю живлення аналогічна кабелю ПЕОМ. Тому небезпека ураження струмом при дотику людини до корпусів принтера чи дисплея незначна.

Електропроводка у відділі розміщена в спеціальних захисних коробах.

У відділі застосовуються наступні засоби захисту:

− малі напруги;

− занулення неструмопровідних частин, що можуть виявитися під напругою;

− ізоляція струмопровідних частин;

− попереджувальні написи.

На випадок аварії передбачені запобіжники в приладах. До роботи в відділі не допускаються особи, що не пройшли інструктаж по електробезпеці.

При вході у відділ висить плакат з інструктажем по техніці безпеки при роботі та ремонті електроустаткування. Силовий щиток має такі запобіжні властивості, як система блокування відкривання кришки щитка та ізолюючий килимок.

При розробці документації до дипломного проекту використовувалась ПЕОМ типу Celeron 1700 і лазерний принтер Canon LBP – 810.

Найбільш значимі фактори при роботі з ПЕОМ наступні:

− іонізоване випромінювання;

− низькочастотні електромагнітні випромінювання;

− зорова напруженість.

Шум у приміщенні, де виконуються роботи, потребуючі концентрації уваги, не повинні перевищувати 55 дБА за ГОСТ 12,1.003-83.

Джерелом шуму у відділі є принтер, рівні звуку якого по технічному паспорті не більш 63 дБА.

Джерелами рентгенівського і ультрафіолетового випромінювання у відділі є електронно-променеві трубки (екрани моніторів). Такі екрани генерують м’яке рентгенівське випромінювання з енергією фотонів від 1 КЕВ до 1 МЕВ і відносяться до категорії іонізованих випромінювань.

Захист від впливу цих випромінювань може бути досягнутий такими способами:

− шляхом віддалення на можливо максимальну відстань оператора від екрана (в основному 0,5...0,7 м);

− скорочення часу безперервної роботи (захист по часу);

− розміщенням оператора під деяким кутом до діагональної осі екрана.

Відповідно ДО ГОСТ 27016-86 для відеотерміналів на основі ЕПТ

нормовані значення наступні:

− потужність дози рентгенівського випромінювання в точці простору на відстані 5 см від поверхні екрану монітора не повинна перевищувати 0.03 мкР/с при 41 годинному робочому тижню;

− щільність потоку ультрафіолетового випромінювання не повинна перевищувати 10 Вт/м^2.

Фактичні значення для моніторів, які використовуються, приведені в таблиці 4.8.

Таблиця 4.8 − Фактичні і нормативні значення характеристики дисплея

Найменування параметрів	Нормовані значення	Фактичні значення
Розміри символів по висоті h, мм	≥ 3	4
Ширина лінії, мм	≥ 0.4	0.4
Яскравість зображення, лм	100	100
Потужність дози рентгенівського випромінювання на відстані 5 см, мкР/с	≤0.03	0.01
Щільність потоку ультрафіолетового випромінювання, Вт/м2	≤10	8
Шум, дБА	≤40	10

Електромагнітні випромінювання низької частоти (від 12 до 150 Гц) роблять найбільш шкідливий вплив на організм людини. Тривалий вплив низькочастотних полів сприяє порушенню репродуктивної функції і виникненню раку.

Для зниження рівня перемінного електромагнітного поля в сучасних моніторах, що відповідають специфікаціям Low Radiation (LR), MPRII і TCQ92, застосовуються котушки компенсації, встановлені на електронно-променевій трубці (ЕПТ), а також спеціальні матеріали в її конструкції. Застосовувані при роботі у відділі монітори Celeron 1700, 2001 року виготовлення, задовольняють встановлені норми.

В найбільшій мірі негативний вплив на зір при роботі з ПЕОМ зв’язано з нерівномірно спроектованим освітленням, прямими і відбитими від екранів відблисками, несприятливим розподілом яскравості в полі зору, пульсацією екрана, неправильним розміщенням робочого місця відносно світлових променів.

Оптимальною для робочих приміщень, призначених для роботи з відеотерміналами, вважається освітленість 200 – 400 лк. Стрибок яскравості при зміні полів зору повинен бути мінімальним, тобто інтенсивність освітлення поверхні, де знаходяться рукописи і документи, не повинні перевищувати яскравості екрана дисплея. Співвідношення яскравості екрана і безпосередньо найближчого оточення не повинне перевищувати три до одного. Фактично дані вимоги на робочому місці виконуються згідно вимог ГОСТ 27016-86.

При тривалій роботі з друкувальним пристроєм вимоги по охороні праці в області тривалих шумових впливів на оператора виконуються і відповідають встановленим нормам.

4.2 Пожежна безпека

Розглянутий інженерно-технічний відділ згідно ОНТП 24 і відноситься до категорії В, класу П-IIа ПУЕ 76/87 по пожежній небезпеці. У відділ є горючі речовини: волокнисті (папір) та тверді (дерево).

Пожежа у відділі представляє особливу небезпеку, оскільки пов’язана зі значними матеріальними втратами. Як відомо, пожежа може виникнути при взаємодії горючих речовин, окислювача і джерела запалювання. Горючими речовинами являються будівельні матеріали для акустичної обробки приміщення, перегородки, двері, підлога, папір для принтеру, корпуса ПЕОМ і принтерів, ізоляція кабелів. Особливістю сучасних ПЕОМ являється дуже висока щільність розміщення елементів електронних схем. При проходженні електричного струму

по провідниках і деталях виділяється тепло, що в умовах їх високої щільності може привести до перегріву. Надійна робота окремих елементів і електричних схем в цілому забезпечується тільки у визначених інтервалах температури, вологості і при заданих електричних параметрах. При відхиленні реальних умов експлуатації від розрахункових може виникнути пожежонебезпечна ситуація.

Кабельні лінії зв'язку являються найбільш пожежонебезпечним місцем. Для зниження загоряння і здатності розповсюдження вогню кабелі покривають вогнетривким покриттям.

Для гасіння пожежі на початковій стадії її виникнення у відділі встановлені 3 вуглекислотних вогнегасники ОУ-2.

Для передбачення пожежі в відділі прийняті такі міри:

− передбачений вільний доступ до мережевих рубильників і вимикачів;

− на випадок короткого замикання передбачені запобіжники і автоматичне відключення мережі;

− в наявності є вогнегасники ОУ-2 для гасіння електрообладнання і ОХП-10 для гасіння об'єктів, що не знаходяться під напругою;

− вхідні двері відділу відкриваються на зовні;

− ширина дверей не менше 0,8 м, а висота проходу більше 1 м;

− у відділі є план евакуації людей;

− у спільному коридорі, поруч з відділом, знаходиться пожежний кран;

− ширина загального коридору, ширина дверей, висота дверей відповідають нормативним значенням (таблиця 4.9).

Таблиця 4.9 − Характеристики евакуаційних виходів

	Нормативні значення, м	Існуючі значення, м
Ширина коридору	> 2,0	2,5
Ширина дверей	> 0,8	1,2
Висота дверей	> 2,0	2,5

ВИСНОВКИ

В результаті розробки даного дипломного проекту:

— розглянуто ймовірні загрози в сучасних комп’ютерних системах;

— проведено аналіз засобів захисту інформації, що дозволяє зробити наступний висновок: захищати необхідно всі компоненти системи: устаткування, програми, дані та персонал. Система захисту повинна бути багаторівневою і будуватися по рівнях секретності. Для надійного захисту необхідно розподілити функції між цими рівнями так, щоб здійснювалося необхідне дублювання функцій захисту, і виконувалась компенсація недоліків одного рівня іншим. Криптографічний захист повинен бути застосований на всіх верхніх рівнях, проте його застосування повинно відповідати передбачуваним загрозам;

— розглянуто особливості застосування хеш-функції в сучасних комп’ютерних системах;

— наведено структуру основного циклу і однієї операції алгоритму обчислення значення модифікованої хеш-функції MD5;

— проведено загальний огляд стійкості хеш-функції;

— визначено наступні властивості хеш-функції: хеш-функція має нескінченну область визначення; хеш-функція має скінченну область значень; вона необоротна; зміна вхідного потоку інформації на один біт змінює близько половини всіх бітів вихідного потоку, тобто результату хеш-функції;

— для реалізації модифікованої хеш-функції MD5 були підключені методи інтерфейсу CryptoAPI. Криптографічний інтерфейс прикладного програмування (CryptoAPI) — це набір функцій, які можна викликати з програм на мові С. Він дозволяє використовувати криптографічні функції для побудови власної програми. CryptoAPI передбачає три набори функцій: сертифікаційні функції; спрощені криптографічні функції; базові криптографічні функції;

— описано модель бази даних ключів. Кожний криптосервер (CSP) має

асоціативну базу даних для ключових контейнерів, яка містить всі приватні і загальні ключі користувачів, що мають доступ до даного комп’ютера;

— розроблено основні модулі обчислення значення хеш-функції і пошуку наявності об’єкта у списку елементів даних в таблиці. Якщо цей елемент знайдено в таблиці, то функція Insert обновляє цей елемент даних, встановлює на нього вказівник current і повертає керування. Якщо такого елемента немає в таблиці, то Insert додає його в кінець списку, встановлює на нього вказівник current і збільшує розмір списку.

— наведено результати виконання програмного комплексу.

Таким чином, від кількості етапів залежить значення хеш-функції. При одному і тому ж вхідному повідомленні значення хеш-функції від 3 до 5 етапів буде різним. Було здійснено перевірку хеш-функції. В результаті чого файли мали ідентичні імена і однаковий вміст.

Результати розробки мають практичне використання (додаток Е).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ємець В., Мельник А., Попович Р. Сучасна криптографія. Основні поняття. – Львів: БаК, 2003. –144 с.
2. Молдовян А.А., Молдовян В.А. Криптография. – Серия “Учебники для вузов. Специальная литература”. – Спб.: Издательство “Лань”, 2000. – 224 с.
3. Чмора А.Л. Современная прикладная криптографія. – М.: Гелиос АРВ, 2002. – 256 с.
4. Столлингс В. Криптография и защита сетей: принципы и практика.: Пер. с англ. – М.: Изд. Дом “Вильямс”, 2001 – 672 с.
5. Горпенюк А.Я., Дудикевич В.Б., Ломницький І.Б. Підвищення швидкодії при обчисленні важкооборотних функцій в асиметричних алгоритмах шифрування. // Захист інформації. – 2003. – №1(18) – С.36-43.
6. Ященко В. В. Основные понятия криптографии // Математическое просвещение. Сер. 3. — 1998.- №2.- С. 53-70.
7. Cramer, R., Ed. Advances in Cryptology // Proc. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2005, Aarhus (Denmark). – 2005. – Vol. 3494.
8. Digital Signature Standard. - NIST, U.S. Department of Commerce, Federal Information Processing Standards Publication (FIPS PUB) 186. - 2004.
9. Кнут Д. Исскуство программирования на ЭВМ. Т2: Получисленные алгоритмы. — М.: Мир, 1977. — 324 с.
10. B. Dixon, A.K. Lenstra. Factoring Integers Using SIMD Sieves // Lecture Notes in Computer Science 765:Advances in Cryptology - Eurocrypt ’93, Springer-Verlag. - P. 28-39.
11. Bowman M.C., Danzig P.B., Hardy D.R., Manber U., Schwartz M.F. The Harvest Information Discovery and Access System. [Electronic resource]. Mode of access: http://citeseer.nj.nec.com/bowman95harvest.html.

12. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Спеціаліст”. Спеціальність – “Комп’ютерні системи та мережі” /О.М.Березький, Н.М.Васильків, І.В.Васильцов, Р.Б.Трембач /Під ред. М.П.Карпінського.–Тернопіль: ТНЕУ, 2008. – 38с.

13. Батлук В.А., Гогіташвіллі Г.Г., Уваров Р.А., Смердова Т.А. Охорона праці в галузі телекомунікацій. Навчальний посібник. – Львів: Афіша, 2003 – 320 с.

14. Методичні вказівки до написання розділу „Охорона праці” в дипломних проектах з освітньо кваліфікаційного рівня „Спеціаліст” для спеціальності 7.091501 – Комп‘ютерні системи та мережі / Укл. Г.В. Сапожник, Н. М. Васильків. – Тернопіль: ТАНГ, 2004 – 24с.

Додаток Д

Фрагмент програмного коду

```
/*   PASS     define the number of passes        (3, 4, or 5)
 *   FPTLEN   define the length of a fingerprint (128, 160, 192, 224 or 256)
 */

#undef LITTLE_ENDIAN

#ifndef PASS
#define PASS       3        /* 3, 4, or 5 */
#endif

#ifndef FPTLEN
#define FPTLEN     256      /* 128, 160, 192, 224 or 256 */
#endif


/*
 *  opt_MD5.h:  specifies the interface to the OPT_MD5 (V.1) hashing library.
 *

typedef unsigned long int opt_MD5_word; /* a OPT_MD5 word = 32 bits */

typedef struct {
  opt_MD5_word    count[2];               /* number of bits in a message */
  opt_MD5_word    fingerprint[8];          /* current state of fingerprint */
  opt_MD5_word    block[32];              /* buffer for a 32-word block */
  unsigned char remainder[32*4];          /* unhashed chars (No.<128) */
} opt_MD5_state;

void opt_MD5_string (char *, unsigned char *); /* hash a string */
int  opt_MD5_file (char *, unsigned char *);   /* hash a file */
void opt_MD5_stdin (void);                     /* filter -- hash input from stdin */
void opt_MD5_start (opt_MD5_state *);          /* initialization */
void opt_MD5_hash (opt_MD5_state *, unsigned char *,
                unsigned int);             /* updating routine */
void opt_MD5_end (opt_MD5_state *, unsigned char *); /* finalization */
void opt_MD5_hash_block (opt_MD5_state *);       /* hash a 32-word block */



*  opt_MD5.c
 *
 *  Descriptions:
 *      -  opt_MD5_string:      hash a string
 *      -  opt_MD5_file:        hash a file
 *      -  opt_MD5_stdin:       filter -- hash input from the stdin device
 *      -  opt_MD5_hash:        hash a string of specified length
 *                          (Opt_MD5_hash is used in conjunction with
 *                           opt_MD5_start & opt_MD5_end.)
 *      -  opt_MD5_hash_block:  hash a 32-word block
 *      -  opt_MD5_start:       initialization
 *      -  opt_MD5_end:         finalization
```

```
#include <stdio.h>
#include <memory.h>
#include "opt_MD5app.h"
#include "opt_MD5.h"

#define VERSION    1                    /* current version number */

void opt_MD5_string (char *, unsigned char *); /* hash a string */
int  opt_MD5_file (char *, unsigned char *);   /* hash a file */
void opt_MD5_stdin (void);                     /* hash input from stdin */
void opt_MD5_start (opt_MD5_state *);          /* initialization */
void opt_MD5_hash (opt_MD5_state *,
       unsigned char *, unsigned int);      /* updating routine */
void opt_MD5_end (opt_MD5_state *, unsigned char *); /* finalization */
void opt_MD5_hash_block (opt_MD5_state *);       /* hash a 32-word block */
static void opt_MD5_tailor (opt_MD5_state *);    /* folding the last output */

static unsigned char padding[128] = {        /* constants for padding */
0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

#define f_1(x6, x5, x4, x3, x2, x1, x0)          \
          ((x1) & ((x0) ^ (x4)) ^ (x2) & (x5) ^ \
           (x3) & (x6) ^ (x0))

#define f_2(x6, x5, x4, x3, x2, x1, x0)                        \
          ((x2) & ((x1) & ~(x3) ^ (x4) & (x5) ^ (x6) ^ (x0)) ^ \
           (x4) & ((x1) ^ (x5)) ^ (x3) & (x5) ^ (x0))

#define f_3(x6, x5, x4, x3, x2, x1, x0)          \
          ((x3) & ((x1) & (x2) ^ (x6) ^ (x0)) ^ \
           (x1) & (x4) ^ (x2) & (x5) ^ (x0))

#define f_4(x6, x5, x4, x3, x2, x1, x0)                                \
          ((x4) & ((x5) & ~(x2) ^ (x3) & ~(x6) ^ (x1) ^ (x6) ^ (x0)) ^ \
           (x3) & ((x1) & (x2) ^ (x5) ^ (x6)) ^                        \
           (x2) & (x6) ^ (x0))

#define f_5(x6, x5, x4, x3, x2, x1, x0)             \
          ((x0) & ((x1) & (x2) & (x3) ^ ~(x5)) ^   \
           (x1) & (x4) ^ (x2) & (x5) ^ (x3) & (x6))

/*
 * Permutations phi_{i,j}, i=3,4,5, j=1,...,i.
 *
 * PASS = 3:
 *              6 5 4 3 2 1 0
 *              | | | | | | | (replaced by)
 *  phi_{3,1}:   1 0 3 5 6 2 4
 *  phi_{3,2}:   4 2 1 0 5 3 6
 *  phi_{3,3}:   6 1 2 3 4 5 0
 *
 * PASS = 4:
 *              6 5 4 3 2 1 0
 *              | | | | | | | (replaced by)
```

```
 *  phi_{4,1}:   2 6 1 4 5 3 0
 *  phi_{4,2}:   3 5 2 0 1 6 4
 *  phi_{4,3}:   1 4 3 6 0 2 5
 *  phi_{4,4}:   6 4 0 5 2 1 3
 *
 * PASS = 5:
 *              6 5 4 3 2 1 0
 *              | | | | | | | (replaced by)
 *  phi_{5,1}:   3 4 1 0 5 2 6
 *  phi_{5,2}:   6 2 1 0 3 4 5
 *  phi_{5,3}:   2 6 0 4 3 1 5
 *  phi_{5,4}:   1 5 3 2 0 4 6
 *  phi_{5,5}:   2 5 0 6 4 3 1
 */

#if   PASS == 3
#define Fphi_1(x6, x5, x4, x3, x2, x1, x0) \
          f_1(x1, x0, x3, x5, x6, x2, x4)
#elif PASS == 4
#define Fphi_1(x6, x5, x4, x3, x2, x1, x0) \
          f_1(x2, x6, x1, x4, x5, x3, x0)
#else
#define Fphi_1(x6, x5, x4, x3, x2, x1, x0) \
          f_1(x3, x4, x1, x0, x5, x2, x6)
#endif

#if   PASS == 3
#define Fphi_2(x6, x5, x4, x3, x2, x1, x0) \
          f_2(x4, x2, x1, x0, x5, x3, x6)
#elif PASS == 4
#define Fphi_2(x6, x5, x4, x3, x2, x1, x0) \
          f_2(x3, x5, x2, x0, x1, x6, x4)
#else
#define Fphi_2(x6, x5, x4, x3, x2, x1, x0) \
          f_2(x6, x2, x1, x0, x3, x4, x5)
#endif

#if   PASS == 3
#define Fphi_3(x6, x5, x4, x3, x2, x1, x0) \
          f_3(x6, x1, x2, x3, x4, x5, x0)
#elif PASS == 4
#define Fphi_3(x6, x5, x4, x3, x2, x1, x0) \
          f_3(x1, x4, x3, x6, x0, x2, x5)
#else
#define Fphi_3(x6, x5, x4, x3, x2, x1, x0) \
          f_3(x2, x6, x0, x4, x3, x1, x5)
#endif

#if   PASS == 4
#define Fphi_4(x6, x5, x4, x3, x2, x1, x0) \
          f_4(x6, x4, x0, x5, x2, x1, x3)
#else
#define Fphi_4(x6, x5, x4, x3, x2, x1, x0) \
           f_4(x1, x5, x3, x2, x0, x4, x6)
#endif

#define Fphi_5(x6, x5, x4, x3, x2, x1, x0) \
          f_5(x2, x5, x0, x6, x4, x3, x1)

#define rotate_right(x, n) (((x) >> (n)) | ((x) << (32-(n))))

#define FF_1(x7, x6, x5, x4, x3, x2, x1, x0, w) {                       \
```

```
    register opt_MD5_word temp = Fphi_1(x6, x5, x4, x3, x2, x1, x0);    \
    (x7) = rotate_right(temp, 7) + rotate_right((x7), 11) + (w);       \
    }

#define FF_2(x7, x6, x5, x4, x3, x2, x1, x0, w, c) {                    \
    register opt_MD5_word temp = Fphi_2(x6, x5, x4, x3, x2, x1, x0);     \
    (x7) = rotate_right(temp, 7) + rotate_right((x7), 11) + (w) + (c);  \
    }

#define FF_3(x7, x6, x5, x4, x3, x2, x1, x0, w, c) {                    \
    register opt_MD5_word temp = Fphi_3(x6, x5, x4, x3, x2, x1, x0);     \
    (x7) = rotate_right(temp, 7) + rotate_right((x7), 11) + (w) + (c);  \
    }

#define FF_4(x7, x6, x5, x4, x3, x2, x1, x0, w, c) {                    \
    register opt_MD5_word temp = Fphi_4(x6, x5, x4, x3, x2, x1, x0);     \
    (x7) = rotate_right(temp, 7) + rotate_right((x7), 11) + (w) + (c);  \
    }

#define FF_5(x7, x6, x5, x4, x3, x2, x1, x0, w, c) {                    \
    register opt_MD5_word temp = Fphi_5(x6, x5, x4, x3, x2, x1, x0);     \
    (x7) = rotate_right(temp, 7) + rotate_right((x7), 11) + (w) + (c);  \
    }

#define ch2uint(string, word, slen) {     \
  unsigned char *sp = string;             \
  opt_MD5_word    *wp = word;             \
  while (sp < (string) + (slen)) {        \
    *wp++ =  (opt_MD5_word)*sp           | \
          ((opt_MD5_word)*(sp+1) <<  8) | \
          ((opt_MD5_word)*(sp+2) << 16) | \
          ((opt_MD5_word)*(sp+3) << 24);  \
    sp += 4;                              \
  }                                       \
}
#define uint2ch(word, string, wlen) {           \
  opt_MD5_word    *wp = word;                   \
  unsigned char *sp = string;                   \
  while (wp < (word) + (wlen)) {                \
    *(sp++) = (unsigned char)( *wp       & 0xFF); \
    *(sp++) = (unsigned char)((*wp >>  8) & 0xFF); \
    *(sp++) = (unsigned char)((*wp >> 16) & 0xFF); \
    *(sp++) = (unsigned char)((*wp >> 24) & 0xFF); \
    wp++;                                       \
  }                                             \
}


/* hash a string */
void opt_MD5_string (char *string, unsigned char fingerprint[FPTLEN >> 3])
{
  opt_MD5_state   state;
  unsigned int  len = strlen (string);

  opt_MD5_start (&state);
  opt_MD5_hash (&state, (unsigned char *)string, len);
  opt_MD5_end (&state, fingerprint);
}

/* hash a file */
int opt_MD5_file (char *file_name, unsigned char fingerprint[FPTLEN >> 3])
{
```

```
  FILE         *file;
  opt_MD5_state   state;
  int          len;
  unsigned char buffer[1024];

  if ((file = fopen (file_name, "rb")) == NULL){
    return (1);                          /* fail */
  } else {
    opt_MD5_start (&state);
    while (len = fread (buffer, 1, 1024, file)) {
      opt_MD5_hash (&state, buffer, len);
    }
    fclose (file);
    opt_MD5_end (&state, fingerprint);
    return (0);                          /* success */
 }
}

/* hash input from stdin */
void opt_MD5_stdin (void)
{
  opt_MD5_state   state;
  int          i, len;
  unsigned char buffer[32],
               fingerprint[FPTLEN >> 3];

  opt_MD5_start (&state);
  while (len = fread (buffer, 1, 32, stdin)) {
    opt_MD5_hash (&state, buffer, len);
  }
  opt_MD5_end (&state, fingerprint);

  for (i = 0; i < FPTLEN >> 3; i++) {
    putchar(fingerprint[i]);
  }
}

/* initialization */
void opt_MD5_start (opt_MD5_state *state)
{
    state->count[0]       = state->count[1] = 0;   /* clear count */
    state->fingerprint[0] = 0x243F6A88;            /* initial fingerprint */
    state->fingerprint[1] = 0x85A308D3;
    state->fingerprint[2] = 0x13198A2E;
    state->fingerprint[3] = 0x03707344;
    state->fingerprint[4] = 0xA4093822;
    state->fingerprint[5] = 0x299F31D0;
    state->fingerprint[6] = 0x082EFA98;
    state->fingerprint[7] = 0xEC4E6C89;
}

/*
 * hash a string of specified length.
 * to be used in conjunction with opt_MD5_start and opt_MD5_end.
 */
void opt_MD5_hash (opt_MD5_state *state,
                   unsigned char *str, unsigned int str_len)
{
  unsigned int i,
           rmd_len,
           fill_len;
```

```
  /* calculate the number of bytes in the remainder */
  rmd_len  = (unsigned int)((state->count[0] >> 3) & 0x7F);
  fill_len = 128 - rmd_len;

  /* update the number of bits */
  if ((state->count[0] +=  (opt_MD5_word)str_len << 3)
                    < ((opt_MD5_word)str_len << 3)) {
    state->count[1]++;
  }
  state->count[1] += (opt_MD5_word)str_len >> 29;

#ifdef LITTLE_ENDIAN

  /* hash as many blocks as possible */
  if (rmd_len + str_len >= 128) {
    memcpy (((unsigned char *)state->block)+rmd_len, str, fill_len);
    opt_MD5_hash_block (state);
    for (i = fill_len; i + 127 < str_len; i += 128){
      memcpy ((unsigned char *)state->block, str+i, 128);
      opt_MD5_hash_block (state);
    }
    rmd_len = 0;
  } else {
    i = 0;
  }
  memcpy (((unsigned char *)state->block)+rmd_len, str+i, str_len-i);

#else

  /* hash as many blocks as possible */
  if (rmd_len + str_len >= 128) {
    memcpy (&state->remainder[rmd_len], str, fill_len);
    ch2uint(state->remainder, state->block, 128);
    opt_MD5_hash_block (state);
    for (i = fill_len; i + 127 < str_len; i += 128){
      memcpy (state->remainder, str+i, 128);
      ch2uint(state->remainder, state->block, 128);
      opt_MD5_hash_block (state);
    }
    rmd_len = 0;
  } else {
    i = 0;
  }
  /* save the remaining input chars */
  memcpy (&state->remainder[rmd_len], str+i, str_len-i);

#endif
}

/* finalization */
void opt_MD5_end (opt_MD5_state *state, unsigned char final_fpt[FPTLEN >> 3])
{
  unsigned char tail[10];
  unsigned int  rmd_len, pad_len;

  /*
  tail[0] = (unsigned char)(((FPTLEN  & 0x3) << 6) |
                            ((PASS    & 0x7) << 3) |
                             (VERSION & 0x7));
  tail[1] = (unsigned char)((FPTLEN >> 2) & 0xFF);
  uint2ch (state->count, &tail[2], 2);
```

```
  /* pad out to 118 mod 128 */
  rmd_len = (unsigned int)((state->count[0] >> 3) & 0x7f);
  pad_len = (rmd_len < 118) ? (118 - rmd_len) : (246 - rmd_len);
  opt_MD5_hash (state, padding, pad_len);

  /*
   * append the version number, the number of passes,
   * the fingerprint length and the number of bits
   */
  opt_MD5_hash (state, tail, 10);

  /* tailor the last output */
  opt_MD5_tailor(state);

  /* translate and save the final fingerprint */
  uint2ch (state->fingerprint, final_fpt, FPTLEN >> 5);

  /* clear the state information */
  memset ((unsigned char *)state, 0, sizeof (*state));
}

/* hash a 32-word block */
void opt_MD5_hash_block (opt_MD5_state *state)
{
  register opt_MD5_word t0 = state->fingerprint[0],    /* make use of */
                        t1 = state->fingerprint[1],    /* internal registers */
                        t2 = state->fingerprint[2],
                        t3 = state->fingerprint[3],
                        t4 = state->fingerprint[4],
                        t5 = state->fingerprint[5],
                        t6 = state->fingerprint[6],
                        t7 = state->fingerprint[7],
                        *w = state->block;

  /* Pass 1 */
  FF_1(t7, t6, t5, t4, t3, t2, t1, t0, *(w   ));
  FF_1(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 1));
  FF_1(t5, t4, t3, t2, t1, t0, t7, t6, *(w+ 2));
  FF_1(t4, t3, t2, t1, t0, t7, t6, t5, *(w+ 3));
  FF_1(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 4));
  FF_1(t2, t1, t0, t7, t6, t5, t4, t3, *(w+ 5));
  FF_1(t1, t0, t7, t6, t5, t4, t3, t2, *(w+ 6));
  FF_1(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 7));

  FF_1(t7, t6, t5, t4, t3, t2, t1, t0, *(w+ 8));
  FF_1(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 9));
  FF_1(t5, t4, t3, t2, t1, t0, t7, t6, *(w+10));
  FF_1(t4, t3, t2, t1, t0, t7, t6, t5, *(w+11));
  FF_1(t3, t2, t1, t0, t7, t6, t5, t4, *(w+12));
  FF_1(t2, t1, t0, t7, t6, t5, t4, t3, *(w+13));
  FF_1(t1, t0, t7, t6, t5, t4, t3, t2, *(w+14));
  FF_1(t0, t7, t6, t5, t4, t3, t2, t1, *(w+15));

  FF_1(t7, t6, t5, t4, t3, t2, t1, t0, *(w+16));
  FF_1(t6, t5, t4, t3, t2, t1, t0, t7, *(w+17));
  FF_1(t5, t4, t3, t2, t1, t0, t7, t6, *(w+18));
  FF_1(t4, t3, t2, t1, t0, t7, t6, t5, *(w+19));
  FF_1(t3, t2, t1, t0, t7, t6, t5, t4, *(w+20));
  FF_1(t2, t1, t0, t7, t6, t5, t4, t3, *(w+21));
  FF_1(t1, t0, t7, t6, t5, t4, t3, t2, *(w+22));
  FF_1(t0, t7, t6, t5, t4, t3, t2, t1, *(w+23));
```

```
FF_1(t7, t6, t5, t4, t3, t2, t1, t0, *(w+24));
FF_1(t6, t5, t4, t3, t2, t1, t0, t7, *(w+25));
FF_1(t5, t4, t3, t2, t1, t0, t7, t6, *(w+26));
FF_1(t4, t3, t2, t1, t0, t7, t6, t5, *(w+27));
FF_1(t3, t2, t1, t0, t7, t6, t5, t4, *(w+28));
FF_1(t2, t1, t0, t7, t6, t5, t4, t3, *(w+29));
FF_1(t1, t0, t7, t6, t5, t4, t3, t2, *(w+30));
FF_1(t0, t7, t6, t5, t4, t3, t2, t1, *(w+31));

/* Pass 2 */
FF_2(t7, t6, t5, t4, t3, t2, t1, t0, *(w+ 5), 0x452821E6);
FF_2(t6, t5, t4, t3, t2, t1, t0, t7, *(w+14), 0x38D01377);
FF_2(t5, t4, t3, t2, t1, t0, t7, t6, *(w+26), 0xBE5466CF);
FF_2(t4, t3, t2, t1, t0, t7, t6, t5, *(w+18), 0x34E90C6C);
FF_2(t3, t2, t1, t0, t7, t6, t5, t4, *(w+11), 0xC0AC29B7);
FF_2(t2, t1, t0, t7, t6, t5, t4, t3, *(w+28), 0xC97C50DD);
FF_2(t1, t0, t7, t6, t5, t4, t3, t2, *(w+ 7), 0x3F84D5B5);
FF_2(t0, t7, t6, t5, t4, t3, t2, t1, *(w+16), 0xB5470917);

FF_2(t7, t6, t5, t4, t3, t2, t1, t0, *(w   ), 0x9216D5D9);
FF_2(t6, t5, t4, t3, t2, t1, t0, t7, *(w+23), 0x8979FB1B);
FF_2(t5, t4, t3, t2, t1, t0, t7, t6, *(w+20), 0xD1310BA6);
FF_2(t4, t3, t2, t1, t0, t7, t6, t5, *(w+22), 0x98DFB5AC);
FF_2(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 1), 0x2FFD72DB);
FF_2(t2, t1, t0, t7, t6, t5, t4, t3, *(w+10), 0xD01ADFB7);
FF_2(t1, t0, t7, t6, t5, t4, t3, t2, *(w+ 4), 0xB8E1AFED);
FF_2(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 8), 0x6A267E96);

FF_2(t7, t6, t5, t4, t3, t2, t1, t0, *(w+30), 0xBA7C9045);
FF_2(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 3), 0xF12C7F99);
FF_2(t5, t4, t3, t2, t1, t0, t7, t6, *(w+21), 0x24A19947);
FF_2(t4, t3, t2, t1, t0, t7, t6, t5, *(w+ 9), 0xB3916CF7);
FF_2(t3, t2, t1, t0, t7, t6, t5, t4, *(w+17), 0x0801F2E2);
FF_2(t2, t1, t0, t7, t6, t5, t4, t3, *(w+24), 0x858EFC16);
FF_2(t1, t0, t7, t6, t5, t4, t3, t2, *(w+29), 0x636920D8);
FF_2(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 6), 0x71574E69);

FF_2(t7, t6, t5, t4, t3, t2, t1, t0, *(w+19), 0xA458FEA3);
FF_2(t6, t5, t4, t3, t2, t1, t0, t7, *(w+12), 0xF4933D7E);
FF_2(t5, t4, t3, t2, t1, t0, t7, t6, *(w+15), 0x0D95748F);
FF_2(t4, t3, t2, t1, t0, t7, t6, t5, *(w+13), 0x728EB658);
FF_2(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 2), 0x718BCD58);
FF_2(t2, t1, t0, t7, t6, t5, t4, t3, *(w+25), 0x82154AEE);
FF_2(t1, t0, t7, t6, t5, t4, t3, t2, *(w+31), 0x7B54A41D);
FF_2(t0, t7, t6, t5, t4, t3, t2, t1, *(w+27), 0xC25A59B5);

/* Pass 3 */
FF_3(t7, t6, t5, t4, t3, t2, t1, t0, *(w+19), 0x9C30D539);
FF_3(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 9), 0x2AF26013);
FF_3(t5, t4, t3, t2, t1, t0, t7, t6, *(w+ 4), 0xC5D1B023);
FF_3(t4, t3, t2, t1, t0, t7, t6, t5, *(w+20), 0x286085F0);
FF_3(t3, t2, t1, t0, t7, t6, t5, t4, *(w+28), 0xCA417918);
FF_3(t2, t1, t0, t7, t6, t5, t4, t3, *(w+17), 0xB8DB38EF);
FF_3(t1, t0, t7, t6, t5, t4, t3, t2, *(w+ 8), 0x8E79DCB0);
FF_3(t0, t7, t6, t5, t4, t3, t2, t1, *(w+22), 0x603A180E);

FF_3(t7, t6, t5, t4, t3, t2, t1, t0, *(w+29), 0x6C9E0E8B);
FF_3(t6, t5, t4, t3, t2, t1, t0, t7, *(w+14), 0xB01E8A3E);
FF_3(t5, t4, t3, t2, t1, t0, t7, t6, *(w+25), 0xD71577C1);
FF_3(t4, t3, t2, t1, t0, t7, t6, t5, *(w+12), 0xBD314B27);
FF_3(t3, t2, t1, t0, t7, t6, t5, t4, *(w+24), 0x78AF2FDA);
FF_3(t2, t1, t0, t7, t6, t5, t4, t3, *(w+30), 0x55605C60);
```

```
  FF_3(t1, t0, t7, t6, t5, t4, t3, t2, *(w+16), 0xE65525F3);
  FF_3(t0, t7, t6, t5, t4, t3, t2, t1, *(w+26), 0xAA55AB94);

  FF_3(t7, t6, t5, t4, t3, t2, t1, t0, *(w+31), 0x57489862);
  FF_3(t6, t5, t4, t3, t2, t1, t0, t7, *(w+15), 0x63E81440);
  FF_3(t5, t4, t3, t2, t1, t0, t7, t6, *(w+ 7), 0x55CA396A);
  FF_3(t4, t3, t2, t1, t0, t7, t6, t5, *(w+ 3), 0x2AAB10B6);
  FF_3(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 1), 0xB4CC5C34);
  FF_3(t2, t1, t0, t7, t6, t5, t4, t3, *(w   ), 0x1141E8CE);
  FF_3(t1, t0, t7, t6, t5, t4, t3, t2, *(w+18), 0xA15486AF);
  FF_3(t0, t7, t6, t5, t4, t3, t2, t1, *(w+27), 0x7C72E993);

  FF_3(t7, t6, t5, t4, t3, t2, t1, t0, *(w+13), 0xB3EE1411);
  FF_3(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 6), 0x636FBC2A);
  FF_3(t5, t4, t3, t2, t1, t0, t7, t6, *(w+21), 0x2BA9C55D);
  FF_3(t4, t3, t2, t1, t0, t7, t6, t5, *(w+10), 0x741831F6);
  FF_3(t3, t2, t1, t0, t7, t6, t5, t4, *(w+23), 0xCE5C3E16);
  FF_3(t2, t1, t0, t7, t6, t5, t4, t3, *(w+11), 0x9B87931E);
  FF_3(t1, t0, t7, t6, t5, t4, t3, t2, *(w+ 5), 0xAFD6BA33);
  FF_3(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 2), 0x6C24CF5C);

#if PASS >= 4
  /* Pass 4. executed only when PASS =4 or 5 */
  FF_4(t7, t6, t5, t4, t3, t2, t1, t0, *(w+24), 0x7A325381);
  FF_4(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 4), 0x28958677);
  FF_4(t5, t4, t3, t2, t1, t0, t7, t6, *(w   ), 0x3B8F4898);
  FF_4(t4, t3, t2, t1, t0, t7, t6, t5, *(w+14), 0x6B4BB9AF);
  FF_4(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 2), 0xC4BFE81B);
  FF_4(t2, t1, t0, t7, t6, t5, t4, t3, *(w+ 7), 0x66282193);
  FF_4(t1, t0, t7, t6, t5, t4, t3, t2, *(w+28), 0x61D809CC);
  FF_4(t0, t7, t6, t5, t4, t3, t2, t1, *(w+23), 0xFB21A991);

  FF_4(t7, t6, t5, t4, t3, t2, t1, t0, *(w+26), 0x487CAC60);
  FF_4(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 6), 0x5DEC8032);
  FF_4(t5, t4, t3, t2, t1, t0, t7, t6, *(w+30), 0xEF845D5D);
  FF_4(t4, t3, t2, t1, t0, t7, t6, t5, *(w+20), 0xE98575B1);
  FF_4(t3, t2, t1, t0, t7, t6, t5, t4, *(w+18), 0xDC262302);
  FF_4(t2, t1, t0, t7, t6, t5, t4, t3, *(w+25), 0xEB651B88);
  FF_4(t1, t0, t7, t6, t5, t4, t3, t2, *(w+19), 0x23893E81);
  FF_4(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 3), 0xD396ACC5);

  FF_4(t7, t6, t5, t4, t3, t2, t1, t0, *(w+22), 0x0F6D6FF3);
  FF_4(t6, t5, t4, t3, t2, t1, t0, t7, *(w+11), 0x83F44239);
  FF_4(t5, t4, t3, t2, t1, t0, t7, t6, *(w+31), 0x2E0B4482);
  FF_4(t4, t3, t2, t1, t0, t7, t6, t5, *(w+21), 0xA4842004);
  FF_4(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 8), 0x69C8F04A);
  FF_4(t2, t1, t0, t7, t6, t5, t4, t3, *(w+27), 0x9E1F9B5E);
  FF_4(t1, t0, t7, t6, t5, t4, t3, t2, *(w+12), 0x21C66842);
  FF_4(t0, t7, t6, t5, t4, t3, t2, t1, *(w+ 9), 0xF6E96C9A);

  FF_4(t7, t6, t5, t4, t3, t2, t1, t0, *(w+ 1), 0x670C9C61);
  FF_4(t6, t5, t4, t3, t2, t1, t0, t7, *(w+29), 0xABD388F0);
  FF_4(t5, t4, t3, t2, t1, t0, t7, t6, *(w+ 5), 0x6A51A0D2);
  FF_4(t4, t3, t2, t1, t0, t7, t6, t5, *(w+15), 0xD8542F68);
  FF_4(t3, t2, t1, t0, t7, t6, t5, t4, *(w+17), 0x960FA728);
  FF_4(t2, t1, t0, t7, t6, t5, t4, t3, *(w+10), 0xAB5133A3);
  FF_4(t1, t0, t7, t6, t5, t4, t3, t2, *(w+16), 0x6EEF0B6C);
  FF_4(t0, t7, t6, t5, t4, t3, t2, t1, *(w+13), 0x137A3BE4);
#endif

#if PASS == 5
  /* Pass 5. executed only when PASS = 5 */
```

```
  FF_5(t7, t6, t5, t4, t3, t2, t1, t0, *(w+27), 0xBA3BF050);
  FF_5(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 3), 0x7EFB2A98);
  FF_5(t5, t4, t3, t2, t1, t0, t7, t6, *(w+21), 0xA1F1651D);
  FF_5(t4, t3, t2, t1, t0, t7, t6, t5, *(w+26), 0x39AF0176);
  FF_5(t3, t2, t1, t0, t7, t6, t5, t4, *(w+17), 0x66CA593E);
  FF_5(t2, t1, t0, t7, t6, t5, t4, t3, *(w+11), 0x82430E88);
  FF_5(t1, t0, t7, t6, t5, t4, t3, t2, *(w+20), 0x8CEE8619);
  FF_5(t0, t7, t6, t5, t4, t3, t2, t1, *(w+29), 0x456F9FB4);

  FF_5(t7, t6, t5, t4, t3, t2, t1, t0, *(w+19), 0x7D84A5C3);
  FF_5(t6, t5, t4, t3, t2, t1, t0, t7, *(w   ), 0x3B8B5EBE);
  FF_5(t5, t4, t3, t2, t1, t0, t7, t6, *(w+12), 0xE06F75D8);
  FF_5(t4, t3, t2, t1, t0, t7, t6, t5, *(w+ 7), 0x85C12073);
  FF_5(t3, t2, t1, t0, t7, t6, t5, t4, *(w+13), 0x401A449F);
  FF_5(t2, t1, t0, t7, t6, t5, t4, t3, *(w+ 8), 0x56C16AA6);
  FF_5(t1, t0, t7, t6, t5, t4, t3, t2, *(w+31), 0x4ED3AA62);
  FF_5(t0, t7, t6, t5, t4, t3, t2, t1, *(w+10), 0x363F7706);

  FF_5(t7, t6, t5, t4, t3, t2, t1, t0, *(w+ 5), 0x1BFEDF72);
  FF_5(t6, t5, t4, t3, t2, t1, t0, t7, *(w+ 9), 0x429B023D);
  FF_5(t5, t4, t3, t2, t1, t0, t7, t6, *(w+14), 0x37D0D724);
  FF_5(t4, t3, t2, t1, t0, t7, t6, t5, *(w+30), 0xD00A1248);
  FF_5(t3, t2, t1, t0, t7, t6, t5, t4, *(w+18), 0xDB0FEAD3);
  FF_5(t2, t1, t0, t7, t6, t5, t4, t3, *(w+ 6), 0x49F1C09B);
  FF_5(t1, t0, t7, t6, t5, t4, t3, t2, *(w+28), 0x075372C9);
  FF_5(t0, t7, t6, t5, t4, t3, t2, t1, *(w+24), 0x80991B7B);

  FF_5(t7, t6, t5, t4, t3, t2, t1, t0, *(w+ 2), 0x25D479D8);
  FF_5(t6, t5, t4, t3, t2, t1, t0, t7, *(w+23), 0xF6E8DEF7);
  FF_5(t5, t4, t3, t2, t1, t0, t7, t6, *(w+16), 0xE3FE501A);
  FF_5(t4, t3, t2, t1, t0, t7, t6, t5, *(w+22), 0xB6794C3B);
  FF_5(t3, t2, t1, t0, t7, t6, t5, t4, *(w+ 4), 0x976CE0BD);
  FF_5(t2, t1, t0, t7, t6, t5, t4, t3, *(w+ 1), 0x04C006BA);
  FF_5(t1, t0, t7, t6, t5, t4, t3, t2, *(w+25), 0xC1A94FB6);
  FF_5(t0, t7, t6, t5, t4, t3, t2, t1, *(w+15), 0x409F60C4);
#endif

  state->fingerprint[0] += t0;
  state->fingerprint[1] += t1;
  state->fingerprint[2] += t2;
  state->fingerprint[3] += t3;
  state->fingerprint[4] += t4;
  state->fingerprint[5] += t5;
  state->fingerprint[6] += t6;
  state->fingerprint[7] += t7;
}

/* tailor the last output */
static void opt_MD5_tailor (opt_MD5_state *state)
{
  opt_MD5_word temp;

#if FPTLEN == 128
  temp = (state->fingerprint[7] & 0x000000FF) |
         (state->fingerprint[6] & 0xFF000000) |
         (state->fingerprint[5] & 0x00FF0000) |
         (state->fingerprint[4] & 0x0000FF00);
  state->fingerprint[0] += rotate_right(temp,  8);

  temp = (state->fingerprint[7] & 0x0000FF00) |
         (state->fingerprint[6] & 0x000000FF) |
         (state->fingerprint[5] & 0xFF000000) |
```

```
        (state->fingerprint[4] & 0x00FF0000);
    state->fingerprint[1] += rotate_right(temp, 16);

    temp  = (state->fingerprint[7] & 0x00FF0000) |
            (state->fingerprint[6] & 0x0000FF00) |
            (state->fingerprint[5] & 0x000000FF) |
            (state->fingerprint[4] & 0xFF000000);
    state->fingerprint[2] += rotate_right(temp, 24);

    temp = (state->fingerprint[7] & 0xFF000000) |
           (state->fingerprint[6] & 0x00FF0000) |
           (state->fingerprint[5] & 0x0000FF00) |
           (state->fingerprint[4] & 0x000000FF);
    state->fingerprint[3] += temp;

  #elif FPTLEN == 160
    temp = (state->fingerprint[7] &  (opt_MD5_word)0x3F) |
           (state->fingerprint[6] & ((opt_MD5_word)0x7F << 25)) |
           (state->fingerprint[5] & ((opt_MD5_word)0x3F << 19));
    state->fingerprint[0] += rotate_right(temp, 19);

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x3F <<  6)) |
           (state->fingerprint[6] &  (opt_MD5_word)0x3F) |
           (state->fingerprint[5] & ((opt_MD5_word)0x7F << 25));
    state->fingerprint[1] += rotate_right(temp, 25);

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x7F << 12)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x3F <<  6)) |
           (state->fingerprint[5] &  (opt_MD5_word)0x3F);
    state->fingerprint[2] += temp;

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x3F << 19)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x7F << 12)) |
           (state->fingerprint[5] & ((opt_MD5_word)0x3F <<  6));
    state->fingerprint[3] += temp >> 6;

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x7F << 25)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x3F << 19)) |
           (state->fingerprint[5] & ((opt_MD5_word)0x7F << 12));
    state->fingerprint[4] += temp >> 12;

  #elif FPTLEN == 192
    temp = (state->fingerprint[7] &  (opt_MD5_word)0x1F) |
           (state->fingerprint[6] & ((opt_MD5_word)0x3F << 26));
    state->fingerprint[0] += rotate_right(temp, 26);

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x1F <<  5)) |
           (state->fingerprint[6] &  (opt_MD5_word)0x1F);
    state->fingerprint[1] += temp;

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x3F << 10)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x1F <<  5));
    state->fingerprint[2] += temp >> 5;

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x1F << 16)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x3F << 10));
    state->fingerprint[3] += temp >> 10;

    temp = (state->fingerprint[7] & ((opt_MD5_word)0x1F << 21)) |
           (state->fingerprint[6] & ((opt_MD5_word)0x1F << 16));
    state->fingerprint[4] += temp >> 16;
```

```
  temp = (state->fingerprint[7] & ((opt_MD5_word)0x3F << 26)) |
        (state->fingerprint[6] & ((opt_MD5_word)0x1F << 21));
  state->fingerprint[5] += temp >> 21;

#elif FPTLEN == 224
  state->fingerprint[0] += (state->fingerprint[7] >> 27) & 0x1F;
  state->fingerprint[1] += (state->fingerprint[7] >> 22) & 0x1F;
  state->fingerprint[2] += (state->fingerprint[7] >> 18) & 0x0F;
  state->fingerprint[3] += (state->fingerprint[7] >> 13) & 0x1F;
  state->fingerprint[4] += (state->fingerprint[7] >>  9) & 0x0F;
  state->fingerprint[5] += (state->fingerprint[7] >>  4) & 0x1F;
  state->fingerprint[6] +=  state->fingerprint[7]        & 0x0F;
#endif
}
/*
 *  opt_MD5test.c:  specifies a test program for the OPT_MD5 hashing library.
*  Makefile for the testing program:
 *
 *        CC=acc
 *        CFLAGS=-fast
 *
 *        opt_MD5: opt_MD5.o opt_MD5test.o
 *              ${CC} ${CFLAGS} opt_MD5.o opt_MD5test.o -o $@
 *        opt_MD5.o opt_MD5test.o: opt_MD5app.h
 *
 *        clean:
 *              /usr/bin/rm -f *.o opt_MD5
 *

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "opt_MD5app.h"
#include "opt_MD5.h"

#define NUMBER_OF_BLOCKS 5000              /* number of test blocks */
#define BLOCK_SIZE       1000              /* number of bytes in a block */

static void opt_MD5_speed (void);            /* test the speed of OPT_MD5 */
static void opt_MD5_cert (void);             /* hash test data set */
static void opt_MD5_print (unsigned char *);  /* print a fingerprint */
static int  little_endian (void);          /* test endianity */

int main (argc, argv)
int  argc;
char *argv[];
{
  int          i;
  unsigned char fingerprint[FPTLEN >> 3];

  if (argc <= 1) {
    opt_MD5_stdin ();                          /* filter */
  }
  for (i = 1; i < argc; i++) {
    if ((argv[i][0] == '?') ||                 /* show help info */
        (argv[i][0] == '-' && argv[i][1] == '?')) {
      printf (" (none)     hash input from stdin\n");
      printf (" ? or -?    show help menu\n");
      printf (" -c         hash certification data\n");
      printf (" -e         test endianity\n");
      printf (" -mstring   hash message\n");
      printf (" -s         test speed\n");
```

```
      printf (" file_name  hash file\n");
    } else if (argv[i][0] == '-' && argv[i][1] == 'm') {  /* hash string */
      opt_MD5_string (argv[i]+2, fingerprint);
      printf ("OPT_MD5(\"%s\") = ", argv[i]+2);
      opt_MD5_print (fingerprint);
      printf ("\n");
    } else if (strcmp (argv[i], "-c") == 0) {      /* hash test set */
      opt_MD5_cert ();
    } else if (strcmp (argv[i], "-s") == 0) {      /* test speed */
      opt_MD5_speed ();
    } else if (strcmp (argv[i], "-e") == 0) {      /* test endianity */
      if (little_endian()) {
        printf ("Your machine is little-endian.\n");
        printf ("You may define LITTLE_ENDIAN to speed up processing.\n");
      } else {
        printf ("Your machine is NOT little-endian.\n");
        printf ("You must NOT define LITTLE_ENDIAN.\n");
      }
    } else {                                       /* hash file */
      if (opt_MD5_file (argv[i], fingerprint)) {
        printf ("%s can not be opened !\n= ", argv[i]);
      } else {
        printf ("OPT_MD5(File %s) = ", argv[i]);
        opt_MD5_print (fingerprint);
        printf ("\n");
      }
    }
  }
  return (0);
}

/* test the speed of OPT_MD5 */
static void opt_MD5_speed (void)
{
  opt_MD5_state   state;
  unsigned char buff[BLOCK_SIZE];
  unsigned char fingerprint[FPTLEN >> 3];
  time_t        start_time, end_time;
  double        elapsed_time;
  unsigned int  i;

  printf ("Test the speed of OPT_MD5 (PASS = %d, FPTLEN = %d bits).\n", PASS, FPTLEN);
  printf ("Hashing %d %d-byte blocks ...\n", NUMBER_OF_BLOCKS, BLOCK_SIZE);

  /* initialize test block */
  for (i = 0; i < BLOCK_SIZE; i++) {
    buff[i] = ~0;
  }

  /* get start time */
  time (&start_time);

  /* hash */
  opt_MD5_start (&state);
  for (i = 0; i < NUMBER_OF_BLOCKS; i++) {
    opt_MD5_hash (&state, buff, BLOCK_SIZE);
  }
  opt_MD5_end (&state, fingerprint);

  /* get end time */
  time (&end_time);
```

```
  /* get elapsed time */
  elapsed_time = difftime(end_time, start_time);

  if (elapsed_time > 0.0) {
    printf ("Elapsed Time = %3.1f seconds\n", elapsed_time);
    printf ("       Speed = %4.2f MBPS (megabits/second)\n",
    (NUMBER_OF_BLOCKS * BLOCK_SIZE * 8)/(1.0E6 * elapsed_time));
  } else {
    printf ("not enough blocks !\n");
  }
}

/* hash a set of certification data and print the results.  */
static void opt_MD5_cert (void)
{
  unsigned int  i;
  char          *str;
  unsigned char fingerprint[FPTLEN >> 3];

  printf ("\n");
  printf ("OPT_MD5 certification data (PASS=%d, FPTLEN=%d):", PASS, FPTLEN);
  printf ("\n");

  str = "";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\") = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "a";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\") = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "OPT_MD5";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\") = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "0123456789";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\") = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "abcdefghijklmnopqrstuvwxyz";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\") = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  opt_MD5_string (str, fingerprint);
  printf ("OPT_MD5(\"%s\")\n      = ", str);
  opt_MD5_print (fingerprint);
  printf ("\n");

  str = "pi.frac";
  if (opt_MD5_file (str, fingerprint)) {
    printf ("%s can not be opened !\n", str);
```

```
  } else {
    printf ("OPT_MD5(File %s) = ", str);
    opt_MD5_print (fingerprint);
    printf ("\n");
  }
}

/* test endianity */
static int little_endian(void)
{
  unsigned long *wp;
  unsigned char str[4] = {'A', 'B', 'C', 'D'};

  wp = (unsigned long *)str;
  if (str[0] == (unsigned char)( *wp & 0xFF)) {
    return (1);                    /* little endian */
  } else {
    return (0);                    /* big endian */
  }
}

/* print a fingerprint in hexadecimal */
static void opt_MD5_print (unsigned char fingerprint[FPTLEN >> 3])
{
  int i;

  for (i = 0; i < FPTLEN >> 3; i++) {
    printf ("%02X", fingerprint[i]);
  }
}
```

Додаток Е

Завідувачу кафедри
комп'ютерної інженерії
к.т.н., доц. О.М. Березькому

ДОВІДКА ПРО ВИКОРИСТАННЯ

Виконаний студенткою групи КСМзс-51 факультету комп'ютерних інформаційних технологій Тернопільського національного економічного університету Кульчицька-Губ'як Г.Є. дипломний проект та тему „Програмний засіб захисту інформації в операційній системі Windows XP" відповідає профілю підприємства, має практичну значимість і планується для використання на фірмі "Polyservice" ПП Булата М.М. А саме, перевірка достовірності важливих документів і виявлення змін.

Директор підприємства (організації) _____

Прізвище, ініціали підпис

М.П.