Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2022 р.

**ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА**
**за другим (магістерським) рівнем вищої освіти**
на тему
# “Дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-21М-1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Лаврусенко О.І.
«____» _____ 2022 р.

Керівник проекту
доктор технічних наук, доцент
_____ Коваленко О.В.
«____» _____ 2022 р.
Рецензент _____
_____



м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 123 *"Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2022 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Лаврусенку Олександру Івановичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI*

2. Керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, доцент*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 19-13 від 17.08.2022 року

3. Строк подання студентом роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

| | |
|---|---|
| *1. Призначення та область використання.* | *6. Наукова новизна.* |
| *2.Перегляд аналогічних існуючих систем.* | *7. Економічна ефективність розробленої програми.* |
| *3. Опис і обґрунтування проектних рішень.* | *8. Заходи з охорони праці та техніки безпеки.* |
| *4. Етапи програмування системи.* | *9. Висновки.* |
| *5. Впровадження системи в промислову експлуатацію* | |

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

| | |
|---|---|
| *Наукова новизна* | *1 аркуш* |
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |
| *Показники економічної ефективності* | *1 аркуш* |

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Савеленко Г.В. | 05.10.2022 | 14.11.2022 |
| Охорона праці | Оришака О.В. | 06.10.2022 | 16.11.2022 |
| | | | |

7. Дата видачі завдання « 6 » вересня 2022 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.10.2022 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.10.2022 р. | |
| 3. | Розробка моделі компонента | 20.10.2022 р. | |
| 4. | Розробка структур даних | 25.10.2022 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.10.2022 р. | |
| 6. | Програмування алгоритмів | 10.11.2022 р. | |
| 7. | Розрахунок економічної ефективності | 13.11.2022 р. | |
| 8. | Розрахунки з охорони праці та техніки безпеки | 15.11.2022 р. | |
| 9. | Оформлення ПЗ | 17.11.2022 р. | |
| 10. | Попередній захист роботи | 10.12.2022 р. | |
| | | | |

Дата видачі завдання
« 6 » вересня 2022 р.

Підпис керівника
_____ Коваленко О.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2022 р.

Підпис здобувача
_____ Лаврусенко О.І.
(прізвище та ініціали)

# АНОТАЦІЯ

**Лаврусенко О.І. Дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.**

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи прозорого шифрування даних з застосуванням засобів PKI.

Метою розробки є дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

Об'єктом дослідження є процес прозорого шифрування даних з застосуванням засобів PKI.

Предметом дослідження є методи прозорого шифрування даних з застосуванням засобів PKI.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.4.

**Ключові слова**: комп'ютерна інженерія, шифрування даних, PKI

## ABSTRACT

**Lavrusenko O.I. Research and software implementation of a transparent data encryption system using PKI tools. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.**

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for a system of transparent data encryption using PKI tools.

The purpose of the development is research and software implementation of a transparent data encryption system using PKI tools.

The object of the study is the process of transparent data encryption using PKI tools.

The subject of research is methods of transparent data encryption using PKI tools.

Research methods are based on information protection methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of a transparent data encryption system using PKI tools.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10.4 environment.



**Keywords:** computer engineering, data encryption, PKI

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

| | | |
|---|---|---|
| НЖМД | – | накопичувач на жорсткому магнітному диску |
| ПЗП | – | постійний запам’ятовуючий пристрій |
| ЦП | – | центральний процесор |
| DES | – | симетричний алгоритм шифрування |
| EFS | – | система шифрування даних на диску |
| FSRTL | – | бібліотека часу виконання файлової системи |
| IPSec | – | протокол захисту даних |
| LPC | – | шина |
| NTFS | – | файлова система |
| PKI | – | інфраструктура відкритих ключів |
| RSA | – | асиметричний алгоритм шифрування |
| WebDAV | – | протокол захисту даних |

# ВСТУП

**Актуальність теми**. На сьогоднішній день уже кожний чув повідомлення про те, як особисті або конфіденційні дані були втрачені через крадіжку або втрату портативного комп'ютера. Портативні комп'ютери пропадають постійно. З ростом числа розкрадань особистих даних і при більш ніж будь-коли високою важливістю дотримання нормативних вимог ретельний захист даних на мобільних комп'ютерних системах украй важливий.

Одним з рішень є використання файлової системи EFS (Encrypting File System – шифрована файлова система), що забезпечує убудоване високоефективне шифрування диска. Система EFS працює однаково добре із власними технологіями перевірки дійсності й контролю доступу системи Windows, так що користувачам не потрібно запам'ятовувати для доступу до своїх даних окремі паролі. І, нарешті, система EFS забезпечує зручні варіанти відновлення даних у випадку втрати користувачем доступу до своїх ключів шифрування (наприклад у випадку видалення або ушкодження профілю користувача або у випадку втрати смарт-карти).

Для генерування, зберігання й розгортання ключів для захисту даних у системі EFS використовується технологія відкритих ключів шифрування (PKI). У даному магістерському проекті для шифрування даних на диску у файловій системі EFS використовується алгоритм стандарту DES. Ці симетричні ключі потім захищаються асиметричною парою ключів (RSA). У системі EFS кожний файл шифрується своїм власним ключем DES, потім цей ключ шифрується користувальницьким ключем RSA і результат зберігається у файл.

**Мета й завдання дослідження.** Метою роботи є дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем прозорого шифрування даних з застосуванням засобів PKI.

– Дослідження системи прозорого шифрування даних з застосуванням засобів PKI.

– Програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

*Об'єктом дослідження* є процес прозорого шифрування даних з застосуванням засобів PKI.

*Предметом дослідження* є методи прозорого шифрування даних з застосуванням засобів PKI.

*Методи дослідження* базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод прозорого шифрування даних з застосуванням засобів PKI.

– Розроблено вітчизняний продукт прозорого шифрування даних з застосуванням засобів PKI, який має більш широкі можливості, на відміну від існуючих аналогів.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі прозорого шифрування даних з застосуванням засобів PKI.

**Достовірність наукових результатів** підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVI Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2022, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №13.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Файлова система, що шифрує (EFS) – це тісно інтегрована з NTFS служба, що розташовується в ядрі Windows 10/11. Її призначення: захист даних, що зберігаються на диску, від несанкціонованого доступу шляхом їхнього шифрування. Поява цієї служби не випадкова, і очікувалося давно. Справа в тому, що існуючі на сьогоднішній день файлові системи не забезпечують необхідний захист даних від несанкціонованого доступу. А як же Windows NT з її NTFS? Адже NTFS забезпечує розмежування доступу й захист даних від несанкціонованого доступу. Але як бути в тому випадку, коли доступ до розділу NTFS здійснюється не за допомогою засобів операційної системи Windows NT, а прямо, на фізичному рівні? Адже це порівняно легко реалізувати, наприклад, завантажившись із дискети й запустивши спеціальну програму: наприклад, досить розповсюджену NTFSdos. Звичайно, можна передбачити таку можливість, і задати пароль на запуск системи, однак практика показує, що такий захист малоефективний, особливо в тому випадку, коли за одним комп'ютером працюють відразу кілька користувачів. А якщо зловмисник може витягти жорсткий диск із комп'ютера, те тут уже не допоможуть ніякі паролі. Підключивши диск до іншого комп'ютера, його вміст можна буде прочитати легко. Таким чином, зловмисник вільно може опанувати конфіденційною інформацією, що зберігається на жорсткому диску. Для того, щоб зловмисник не міг виконати перераховані вище дії, у магістерському проекті пропонується реалізувати систему захисту файлів (EFS) на основі технології відкритих ключів (PKI). Ця технологія дозволяє розмежувати доступ до можливості управління захистом файлів на диску, за рахнок введення сертифікатів. Задачею PKI є визначення політики випуску цифрових сертифікатів, видача їх і анулювання,

зберігання інформації, необхідної для наступної перевірки правильності сертифікатів. Сертифікат – цифровий або паперовий документ, що підтверджує відповідність між відкритим ключем і інформацією, що ідентифікує власника ключа. Містить інформацію про власника ключа, відомості про відкритий ключ, його призначенні й області застосування, назва центра сертифікації й т.д.

Відкритий ключ може бути використаний для організації захищеного каналу зв'язку із власником двома способами:

– для перевірки підпису власника (автентифікація)
– для шифрування посилаємих йому даних (конфіденційність)

### 1.2 Область застосування

Областю застосування EFS є захисту від фізичного читання даних. Єдиний спосіб – це шифрування файлів. Найпростіший випадок такого шифрування – архівування файлу з паролем. Однак тут є ряд серйозних недоліків. По-перше, користувачеві потрібно щораз вручну шифрувати й дешифрувати (тобто, у нашім випадку архівувати й розархівувати) дані перед початком і після закінчення роботи, що вже саме по собі зменшує захищеність даних. Користувач може забути зашифрувати (заархівувати) файл після закінчення роботи, або (ще більш банально) просто залишити на диску копію файлу. По-друге, паролі, придумані користувачем, як правило, легко вгадуються. У кожному разі, існує достатня кількість утиліт, що дозволяють розпаковувати архіви, захищені паролем. Як правило, такі утиліти здійснюють підбор пароля шляхом перебору слів, записаних у словнику. Система EFS була розроблена з метою подолання цих недоліків.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Проведемо аналіз існуючих програмних продуктів призначених для захисту диска.

– Advanced EFS data recovery.

– Microsoft так само має у своєму арсеналу програму відновлення reccerts.exe, яку можна одержати через службу платної підтримки.

– Passware пропонує EFSkey.

– Handy Recovery.

– ARAX Disk Doctor – Data Recovery.

**Advanced EFS Data Recovery**

Програмний продукт призначений для доступу до файлів і папок, зашифрованих засобами Encrypting File System (EFS). Advanced EFS Data Recovery (AEFSDR) відновить доступ до даного, захищеним EFS, в Windows 10/11, XP, 2003, Server 2008 і Vista. У випадку, якщо диск із даними встановлений на іншому комп'ютері, відформатований або якщо деякі ключі шифрування ушкоджені, за допомогою AEFSDR ви однаково зможете відновити дані.

Microsoft Encrypting File System (EFS) є складовою частиною NTFS і доступна в багатьох сучасних версіях операційної системи Windows. EFS здійснює повне й прозоре для користувача шифрування файлів, забезпечуючи тим самим надійний захист від несанкціонованого доступу до файлів навіть у тому випадку, коли зловмисник заволодів комп'ютером або дисками, на яких зберігаються зашифровані дані.

Втратити можливість доступу до даним, захищеним EFS, досить просто. Втрата можлива у випадку установки Windows поверх старої системи, переформатування або зміни розділів диска, а також при установці диска із захищеними даними в інший комп'ютер.

Advanced EFS Data Recovery ефективно відновлює захищені засобами EFS файли навіть у тих випадках, коли всі інші методи розшифровки й відновлення не діють. Сканування жорсткого диска на низькому рівні й проглядання збережених даних сектор за сектором дозволяє Advanced EFS Data Recovery відновлювати зашифровані файли й папки навіть у тому випадку, коли деякі ключі шифрування ушкоджені або безповоротно загублені.

AEFSDR допомагає вирішувати проблеми, що виникли внаслідок помилок адміністрування EFS, таких як видалення облікових записів користувачів, відсутність агентів відновлення (Data Recovery Agents) або їхнє неправильне конфігурування, некоректний перенос облікових записів в інший домен, а також перенос дисків із зашифрованими даними між комп'ютерами.

**Handy Recovery**

Handy Recovery є програмним забезпеченням відновлення даних, розробленим для того, щоб відновлювати випадково загублені файли. Програма підтримує FAT 12/16/32, NTFS і NTFS 5 + EFS файлові системи. Дана програма показує ймовірність успішного відновлення для кожного файлу. Програмне забезпечення також пропонує всебічний перегляд диска для певних типів файлу. Доступна фільтрація файлу по імені, масці, даті й розміру.

**ARAX Disk Doctor – Data Recovery**

ARAX Disk Doctor – Data Recovery – це все в один набір сервісних програм для професійного відновлення даних.

Дана програма відновлює файли, відновлює файли з ушкоджених і форматованих розділів, і відновлює загублені або вилучені розділи, заново становлячи лейбли розділів.

ARAX Disk Doctor також включає кілька просунутих функцій відновлення даних, такі як: відновлення з RAID-1 – RAID-5 дисків, звичайних і динамічних дисків і з EFS захищених папок.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

**Delphi 10.4 Sydney**

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

− Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за

допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

**Істотне поліпшення Delphi Code Insight**

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

**Delphi Custom Managed Records**

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

**Єдине керування пам'яттю**

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

**Розширена підтримка бібліотек C++**

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

**Win 64-відладник і збирач для C++**

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL,

включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

**Підвищення якості й швидкодії інструментів**

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

**Змінені стилі VCL для High DPI**

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

**Нові High DPI стилі й стилізація окремих VCL компонент**

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній

формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

**Поліпшена кроссплатформеність**

- Додана підтримка Metal Driver GPU для macOS і iOS.
- Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.
- Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.
- Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.
- Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.
- Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

**Оновлений менеджер пакетів Getit**

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

**Універсальний інсталятор для установки Online і Offline**

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

## 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи прозорого шифрування даних з застосуванням засобів PKI.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

**Опис EFS**

**Encrypting File System** (EFS) – система шифрування даних, що реалізує шифрування на рівні файлів в операційних системах Microsoft Windows NT(починаючи з Windows 10/11 і вище), за винятком "домашніх" версій. Дана система надає можливість "прозорого шифрування" даних, які зберігаються на розділах з файловою системою NTFS для захисту потенційно конфіденційних від несанкціонованого доступу при фізичному доступі до комп'ютера й дисків. Шифруванна файлова система дозволяє користувачам зберігати дані на диску в зашифрованному форматі. Файл зашифрований одним користувачем не може бути відкритий іншим користувачем, якщо йому не призначені відповідні дозволи. Після того як файл був зашифрований, він автоматично залишається зашифрованним у будь-якому місці зберігання на диску. Користувач, що має права на відкриття файлу. працює з ним як з будь-яким іншої, а інші користувачі при спробі відкрити такий файл, одержують повідомлення "Відмовлено в доступі". Шифруванню можуть піддаватися будь-які файли, у тому числі що виконуються.

Перед використанням можливостей шифрування EFS варто визначитися чи буде використовуватися **Агент відновлення даних**. Агентом відновлення називається користувач, уповноважений розшифровувати дані, зашифровані іншим користувачем, якщо користувач втратив закриті ключі сертифіката шифрування або обліковий запис користувача віддалений і потрібно відновити зашифровані дані. Як правило, Агентом відновлення вказується Адміністратор, але може бути призначений і інший користувач. Може бути створене трохи

Агентів відновлення. Щоб призначити користувача Агентом відновлення, необхідно спочатку створити сертифікати Агента відновлення.

Автентифікація користувача й права доступу до ресурсів, що мають місце в NT працюють, коли операційна система завантажена, але при фізичному доступі до системи можливо завантажити іншу ОС щоб обійти ці обмеження. EFS використовує симетричне шифрування для захисту файлів, а також шифрування засноване на парі відкритий/закритий ключ для захисту випадково-згенерованого ключа шифрування для кожного файлу. За замовчуванням закритий ключ користувача захищений за допомогою шифрування користувальницьким паролем і захищеність даних залежить від стійкості пароля користувача.

**Опис роботи**

EFS працює, шифруючи кожний файл за допомогою алгоритму симетричного шифрування, що залежить від версії операційної системи й настроювань (починаючи з Windows XP доступна теоретична можливість використання сторонніх бібліотек для шифрування даних). При цьому використовується випадково-згенерований ключ для кожного файлу, називаний **File Encryption Key** (FEK), вибір симетричного шифрування на даному етапі пояснюється його швидкістю й більшою надійністю стосовно асиметричного шифрування. У данному магістерському проекті у якості симетричного алгоритма шифрування вибраний DES, у звязу з тим, що він є достатньо стійким та швидким.

FEK (випадковий для кожного файлу ключ симетричного шифрування) захищається шляхом асиметричного шифрування, що використовує відкритий ключ користувача файл, який шифрує, і алгоритм RSA (теоретично можливе використання інших алгоритмів асиметричного шифрування). RSA обраний тому, що він достаньо стійкий , для цих потреб, та виконується більш швидко, ніж інші алгоритми симметричного шифрування. Зашифрований у такий спосіб ключ FEK зберігається в альтернативному потоці $EFS файлової системи NTFS. Для розшифрування даних, драйвер шифрованної файлової системи, прозоро для

користувача, розшифровує FEK використовуючи закритий ключ користувача, а потім і необхідний файл за допомогою розшифруванного файлового ключа.

Оскільки шифрування/дешифрування файлів відбувається за допомогою драйвера файлової системи (по суті надбудови над NTFS), воно відбувається прозоро для користувача й додатків. Варто помітити, що EFS не шифрує файли, передані по мережі, тому для захисту переданих даних необхідно використовувати інші протоколи захисту даних (IPSec або WebDAV).

При першому використанні можливостей EFS система створює цифрове посвідчення, що використовує для роботи із зашифруванними файлами. За замовчуванням (якщо інше не встановлено адміністратором) таке посвідчення має ім'я користувача й міститься в сховище "Особисті сертифікати".

З метою відновлення доступу до зашифруванних файлів після переустановки системи або внаслідок втрати закритого ключа варто зберегти в надійному місці закриті ключі Агентів відновлення або (якщо вони не призначені), закриті ключі всіх користувачів, що використовують EFS, експортувавши їх зі сховища "Особисті" оснащення "Сертифікати". Варто помітити, що якщо для локального користувача адміністратор спробує видалити пароль облікового запису, то користувач втратить всі особисті сертифікати, а відповідно й доступ до зашифруванних EFS файлам (при такій спробі буде видане відповідне попередження)

Шифрувати можна як окремі файли, так і цілі папки, при цьому якщо шифрується папка вже утримуюча файли, тобто можливість вибору, шифрувати тільки папку або папку й вкладені файли. Шифрування папки не означає що інші користувачі не зможуть переглядати вміст папки – вони лише не зможуть відкривати зашифруванні файли. Всі нові файли, збережені в зашифруванній папці або скопійовані в неї будуть автоматично зашифруванні. Шифрувати папки більш зручно, і крім того безпечно, оскільки EFS має схему відновлення після аварійного збою (наприклад якщо під час операції шифрування відбулася критична помилка) яка передбачає створення незашифруванної архівної копії

вихідного файлу – при успішному завершенні операції шифрування архівна копія віддаляється, але може бути відновлена спеціальними програмами відновлення віддалених даних, що створює потенційну погрозу інформаційної безпеки. А при збереженні файлу в зашифруванній папці шифрування відбувається без створення такої резервної копії. Якщо все-таки шифрувалися одиночні файли, тобто можливість перезаписати кластери, що залишилися після зміни або видалення файлів на томах NTFS випадковими значеннями – для цього може бути використана команда cipher /w:шлях запущена з командного рядка (докладніше про використання цієї команди дивитеся в довіднику по командному рядку) або програми сторонніх розроблювачів.

Із усього перерахованого вище можна зробити наступні висновки:

– Система EFS надає користувачам можливість зашифровувати каталоги NTFS, використовуючи стійку, засновану на загальних ключах криптографічну схему, при цьому всі файли в закритих каталогах будуть зашифруванні. Шифрування окремих файлів підтримується, але не рекомендується через непередбачене поводження додатків.

– Система EFS також підтримує шифрування віддалених файлів, доступ до яких здійснюється як до спільно використовуваних ресурсів. Якщо мають місце користувальницькі профілі для підключення, використовуються ключі й сертифікати віддалених профілів. В інших випадках генеруються локальні профілі й використовуються локальні ключі.

– Система EFS надає можливість встановити політику відновлення даних таким чином, що зашифруванні дані можуть бути відновлені за допомогою EFS, якщо це буде потрібно.

– Політика відновлення даних убудована в загальну політику безпеки Windows. Контроль за дотриманням політики відновлення може бути делегований уповноваженим на це особам. Для кожного підрозділу організації може бути зконфігурована своя політика відновлення даних.

– Відновлення даних в EFS – закрита операція. У процесі відновлення розшифровуються дані, але не ключ користувача, за допомогою якого ці дані були зашифруванні.

– Робота із зашифруванними файлами в EFS не жадає від користувача яких-небудь спеціальних дій по шифруванню й дешифруванню даних. Дешифрування й шифрування відбуваються непомітно для користувача в процесі зчитування й запису даних на диск.

– Система EFS підтримує резервне копіювання й відновлення зашифруванних файлів без їхньої розшифровки. Програма NtBackup підтримує резервне копіювання зашифруванних файлів.

– Система EFS убудована в операційну систему таким чином, що витік інформації через файли підкачування неможливий, при цьому гарантується, що всі створювані копії будуть зашифруванні

– Передбачено численні запобіжні заходи для забезпечення безпеки відновлення даних, а також захист від витоку й втрати даних у випадку фатальних збоїв системи.

**Опис PKI**

Розглянемо технологію PKI. Задачею PKI є визначення політики випуску цифрових сертифікатів, видача їх і анулювання, зберігання інформації, необхідної для наступної перевірки правильності сертифікатів. У число додатків, що підтримують PKI, входять: EFS, захищена електронна пошта, протоколи платежів, електронні чеки, електронний обмін інформацією, захист даних у мережах із протоколом IP, електронні форми й документи з електронним цифровим підписом (ЕЦП). Діяльність інфраструктури керування відкритими ключами здійснюється на основі регламенту системи. Інфраструктура відкритих ключів ґрунтується на використанні принципів криптографічної системи з відкритим ключем. Інфраструктура керування відкритими ключами складається із центра сертифікації, кінцевих користувачів, і опціональних компонентів: центра реєстрації й мережного довідника.

Зрозуміло, що PKI оперує в роботі сертифікатами. Але у зв'язку з тим, що в PKI використовуються сертифікати, виникає множина нюансів, без яких будь-яка PKI не буде працювати коректно.

По суті, сертифікат – це ключова пара, що складається із двох ключів – зазвичай перший ключ називається закритим ключем (private key), а другий ключ – відкритим ключем (public key). Ці ключі створюються тільки в парі й мають однаковий електронний відбиток. По електронному відбитку можна визначити, чи відповідає даний відкритий ключ своєму закритому ключу. Створює ці ключі якийсь центр, що зазвичай називається центром видачі сертифікатів або центром, що засвідчує, по запиті користувача. Користувач робить запит на сертифікат, після чого, після деяких процедур ідентифікації користувача, центр видає йому сертифікат зі своїм підписом (цей підпис свідчить про те, що даний сертифікат виданий саме цим центром видачі сертифікатів і ніким іншим), після чого користувач має в наявності свій закритий ключ і відповідний йому відкритий ключ. Закритий ключ використовується для підпису даних, відкритий ключ у свою чергу використовується для шифрування даних. Відкритий ключ відомий усім, а закритий ключ зберігається в таємниці. Власник закритого ключа завжди зберігає його в захищеному місці й ні за яких умов не повинен допустити того, щоб цей ключ став відомим зловмисникам або іншим користувачам. Якщо ж закритий ключ усе таки стане відомий зловмисникам, необхідно терміново сповістити про це колег, щоб запобігти витоку важливої інформації. Тільки власник закритого ключа може підписати дані, а також розшифрувати дані, які були зашифровані відкритим ключем, що відповідає закритому ключу власника. Тому що закритий ключ використовується для підпису даних – його можна назвати своєрідним ідентифікатором передплатника. Підпис на даних або листі гарантує цілісність отриманої інформації.

**Термінологія PKI**

Із усього вище сказаного можна виділити деякі пункти, а також додати нові, для того щоб визначити основні терміни, використовувані в PKI.

Отже, в PKI використовуються терміни:

– Сертифікат – це ключова пара (складається з відкритого й закритого ключів), до якої приписаний її унікальний номер, ім'я власника сертифіката, а також ім'я центра видачі, що видав цей сертифікат.

– Закритий ключ – ключ, що зберігається в таємниці, створений з використанням PKI алгоритмів, що має свій унікальний електронний помилок і, що використовується для одержання зашифрованих даних і підпису даних.

– Відкритий ключ – ключ, створений у парі із закритим ключем, що має такий же електронний відбиток, як і закритий ключ, якому він відповідає, використовується для шифрування даних і перевірки підпису

– Підписані дані – дані, підписані за допомогою закритого ключа користувача.

– Зашифровані дані – дані, зашифровані за допомогою відкритого ключа користувача.

Терміни, які необхідні для загального розуміння:

– Мережа довіри – або ланцюжок сертифікацій, необхідна й потрібна для тих випадків, коли є множина різних центрів, що засвідчують, і виникають ситуації, коли один УЦ (удостовірюючий центр), не довіряє якомусь іншому, але при цьому може покластися на те, що загальний дружній УЦ довіряє обом

– Особисті сертифікати – сертифікати які зберігаються в користувача в особистому сховищі сертифікатів.

– Кореневі центри сертифікації – центри сертифікації, яким довіряють споконвічно всі, наприклад після установки ОС. У ці центри сертифікації можна в будь-який час додавати нові центри, яким Ви хочете довіряти

– Довірені центри сертифікації – список центрів сертифікації, яким довіряєте особисто. Щоб зробити любий УЦ довіреним, досить одержати від нього сертифікат і внести його в довірені центри Також важливо знати про поняття центра сертифікації й про кінцевих користувачів.

– Центр сертифікації (удостоверяючий центр) – є основною структурою, що формує цифрові сертифікати підлеглих центрів сертифікації й кінцевих користувачів. Центр сертифікації сам формує власний секретний ключ і сертифікат, що містить відкритий ключ даного центра. Засвідчує автентичність відкритого ключа користувача своїм електронно-цифровим підписом. Формує список відкликаних сертифікатів. Веде бази всіх виготовлених сертифікатів і списків відкликаних сертифікатів. Відкритий ключ, підписаний центром сертифікації, називається сертифікатом відкритого ключа.

– Кінцеві користувачі – є користувачі, додатки або системи, що є власниками сертифіката й використовують інфраструктуру керування відкритими ключами

– Центр Реєстрації – опціональна компонента інфраструктури, призначена для реєстрації кінцевих користувачів і забезпечення їхньої взаємодії із центром сертифікації.

– Мережний довідник – опціональна компонента інфраструктури, що містить сертифікати й списки відкликаних сертифікатів і, що служить для мети поширення цих об'єктів серед користувачів.

Основні задачі системи інформаційної безпеки, які вирішує інфраструктура керування відкритими ключами:

– забезпечення конфіденційності інформації;

– забезпечення цілісності інформації;

– забезпечення автентифікації користувачів і ресурсів, до яких звертаються користувачі;

– забезпечення можливості підтвердження зроблених користувачами дій з інформацією (невідвергаємість).

Впровадження інфраструктури керування відкритими ключами з урахуванням зниження витрат і строків впровадження здійснюється протягом семи етапів.

– Етап 1. Аналіз вимог до системи.

– Етап 2. Визначення архітектури.

– Етап 3. Визначення регламенту.

– Етап 4. Огляд системи безпеки. Аналіз і мінімізація ризиків.

– Етап 5. Інтеграція.

– Етап 6. Розгортання.

– Етап 7. Експлуатація.

**Деякі основні моменти**

Двома словами вже було сказано, для чого потрібні закритий і відкритий ключі, що таке сертифікат і центри, що засвідчують. Але тому що це основні компоненти PKI, розберемо докладніше наступні моменти:

– у чому полягає робота УЦ

– як відбувається видача сертифіката, обмін відкритими ключами і як зрозуміти, що відкритий ключ, що перебувати перед моїми очами, не фальшивий

– а також те, які бувають PKI.

**УЦ і його робота**

Основна робота центра, що засвідчує, полягає в ідентифікації користувачів і їхніх запитів на сертифікати, у видачі користувачам сертифікатів, у перевірках автентичності сертифікатів, у перевірці за сертифікатом, чи не видає користувач сертифіката себе за інший, в анулюванні або відкликанні сертифікатів, у веденні списку відкликаних сертифікатів.

**Процес роботи із сертифікатами**

Для того щоб одержати сертифікат, потрібно знайти який-небудь УЦ в Інтернеті (альтернативним рішенням є використання ПЗ PGP або його подібного), після чого виписати сертифікат і встановити його собі в систему (приклади УЦ в Інтернеті: http://e-notary.ru/ і http://cryptopro.ru/certsrv/). Зазвичай цей процес відбувається автоматично. Після установки сертифіката, його можна буде побачити в себе в сховище особистих сертифікатів. Для того щоб переглянути його властивості, досить просто відкрити його. У властивостях можна побачити час дії сертифіката, ким він був виданий, кому був виданий, його універсальний

номер та інші властивості. Після одержання сертифікатів двома або більше користувачами від одного УЦ, відбувається організація найпростішої по архітектурі PKI. PKI – з одиночним УЦ. Користувачі, зберігши сертифікати у файл обмінюються ними (у такий спосіб відбувається обмін відкритими ключами) і починають захищену переписку. Перевірка дійсності отриманого відкритого ключа проводиться по електронному відбитку цього ключа. У найпростішому випадку досить подзвонити колезі, який вислав відкритий ключ і звірити з ним електронний відбиток ключа. Якщо він збігся – можна сміло починати захищену переписку, якщо немає – обмінятися ключами ще раз.

**Архітектури PKI**

В основному виділяють 5 видів архітектур PKI, це:

1. Проста PKI (одиночний УЦ)
2. Ієрархічна PKI
3. Мережна PKI
4. Крос-сертифіковані корпоративні PKI
5. Архітектура мостового УЦ

В основному PKI діляться на різні архітектури по наступних ознаках:

– кількість УЦ (а також кількість УЦ, які довіряють друг-другові);

– складність перевірки шляхи сертифікації;

– наслідку видачі зловмисника себе за УЦ.

Розглянемо більш докладно кожну з архітектур PKI окремо.

**Проста PKI**

Як уже говорилося вище, найпростіша з архітектур, це архітектура одиночного УЦ. У цьому випадку всі користувачі довіряють одному УЦ і листуються між собою. У даній архітектурі, якщо зловмисник видасть себе за УЦ, необхідно просто перевипустити всі виписані сертифікати й продовжити нормальну роботу.

**Ієрархічна PKI**

Ієрархічна структура – це найбільше часто зустрічаєма архітектура PKI. У цьому випадку на чолі всієї структури стоїть один Головний УЦ, якому всі довіряють і йому підкоряються нижчестоящі УЦ. Крім цього головного УЦ у структурі присутні ще не один УЦ, що підкоряється вищестоящому, котрому у свою чергу приписані які-небудь користувачі або нижчестоящі УЦ. Частний приклад ієрархічної PKI – корпоративна PKI. В ієрархічній PKI, навіть якщо зловмисник видав себе за будь-яку УЦ, мережа продовжує працювати без нього, а коли він відновлює нормальну працездатність – він просто знову включається в структуру.

**Мережна PKI**

Мережна архітектура PKI це також окремий випадок ієрархічної архітектури. Але в цьому випадку немає одного головного УЦ, якому всі довіряють. У цій архітектурі всі УЦ довіряють поруч знаходячимся УЦ, а кожний користувач довіряє тільки тому УЦ, у якого виписав сертифікат. У дану архітектуру PKI легко додається новий УЦ. У даній архітектурі найбільш складна побудова ланцюжка сертифікації.

**Архітектура крос-сертифікованої корпоративної PKI**

Даний вид архітектури можна розглядати як змішаний вид ієрархічної й мережний архітектур. Є кілька фірм, у кожної з яких організована яка те своя PKI, але вони хочуть спілкуватися між собою, у результаті чого виникає їх загальна міжфірмена PKI. В архітектурі крос-сертифікованої корпоративної PKI сама складна система ланцюжка сертифікації.

**Архітектура мостового УЦ**

Архітектура мостового УЦ розроблялася для того, щоб забрати недоліки складного процесу сертифікації в крос-сертифікованої корпоративної PKI. У цьому випадку всі компанії довіряють не якій те однієї або двом фірмам, а одному певному мостовому УЦ, що є практично їх головним УЦ, але він не є основним пунктом довіри, а виступає в ролі посередника між іншими УЦ.

**Опис DES**

DES являє собою блоковий шифр, він шифрує дані 64-бітовими блоками. З одного кінця алгоритму вводиться 64-бітовий блок відкритого тексту, а з іншого кінця виходить 64-бітовий блок шифротексту.

DES є симетричним алгоритмом: для шифрування й дешифрування використовуються однакові алгоритм і ключ (за винятком невеликих розходжень у використанні ключа). довжина ключа дорівнює 56 бітам. (ключ звичайно представляється 64-бітовим числом, але кожний восьмий біт використовується для перевірки парності й ігнорується, біти парності є найменшими значущими бітами байтів ключа). Ключ, що може бути будь-яким 56-бітним числом, можна змінити в будь-який момент часу. Ряд чисел вважаються слабкими ключами, але їх можна легко уникнути. Безпека повністю визначається ключем.

Процес шифрування складається із чотирьох етапів.

На першому з них виконується початкова перестановка (IP) 64-бітного вихідного тексту (забілювання), під час якої біти переставляються у відповідності зі стандартною таблицею.

Наступний етап складається з 16 раундів однієї й тієї ж функції, що використовує операції зсуву й підстановки.

На третьому етапі ліва й права половини виходу останньої (16-ої) ітерації міняються місцями.

Нарешті, на четвертому етапі виконується перестановка $IP^{-1}$ результату, отриманого на третьому етапі. Перестановка $IP^{-1}$ інверсна початковій перестановці.

У правій частині рисунку 3.1 показаний спосіб, яким використовується 56-бітний ключ. Спочатку ключ подається на вхід функції перестановки. Потім для кожного з 16 раундів підключ $K_i$ є комбінацією лівого циклічного зсуву й перестановки. Функція перестановки та сама для кожного раунду, але підключи $K_i$ для кожного раунду виходять різні внаслідок повторюваного зсуву бітів ключа.


Рисунок 3.1 – Загальна схема DES

На найпростішому рівні алгоритм не представляє нічого більшого, ніж комбінація двох основних методів шифрування: зсуву й дифузії. Фундаментальним будівельним блоком DES є застосування до тексту одиничної комбінації цих методів (підстановка, а за нею – перестановка), що залежить від ключа. Такий блок називається етапом. DES складається з 16 етапів, однакова комбінація методів застосовується до відкритого тексту 16 разів (рисунок 3.2).

Розглянемо більш докладно алгоритм шифрування. DES працює з 64-бітовим блоком відкритого тексту. Після первісної перестановки блок розбивається на праву й ліву половини довжиною по 32 біта. Потім виконується 16 етапів однакових дій, називаних функцією *f*, у яких дані поєднуються із ключем. Після шістнадцятого етапу права й ліва половини поєднуються й алгоритм завершується заключною перестановкою (зворотною відносно первісної). На кожному етапі біти ключа зрушуються, і потім з 56 біт ключа вибираються 48 біт. Права половина даних збільшується до 48 біт за допомогою перестановки з розширенням, поєднується за допомогою XOR с 48 бітами зміщеного й переставленого ключа, проходить через 8 S-блоків, утворюючи 32

нових біта, і переставляється знову. Ці чотири операції й виконуються функцією f. Потім результат функції f поєднується з лівою половиною за допомогою іншого XOR. У підсумку цих дій з'являється нова права половина, а стара права половина стає нової лівою. Ці дії повторюються 16 разів, утворюючи 16 етапів DES.

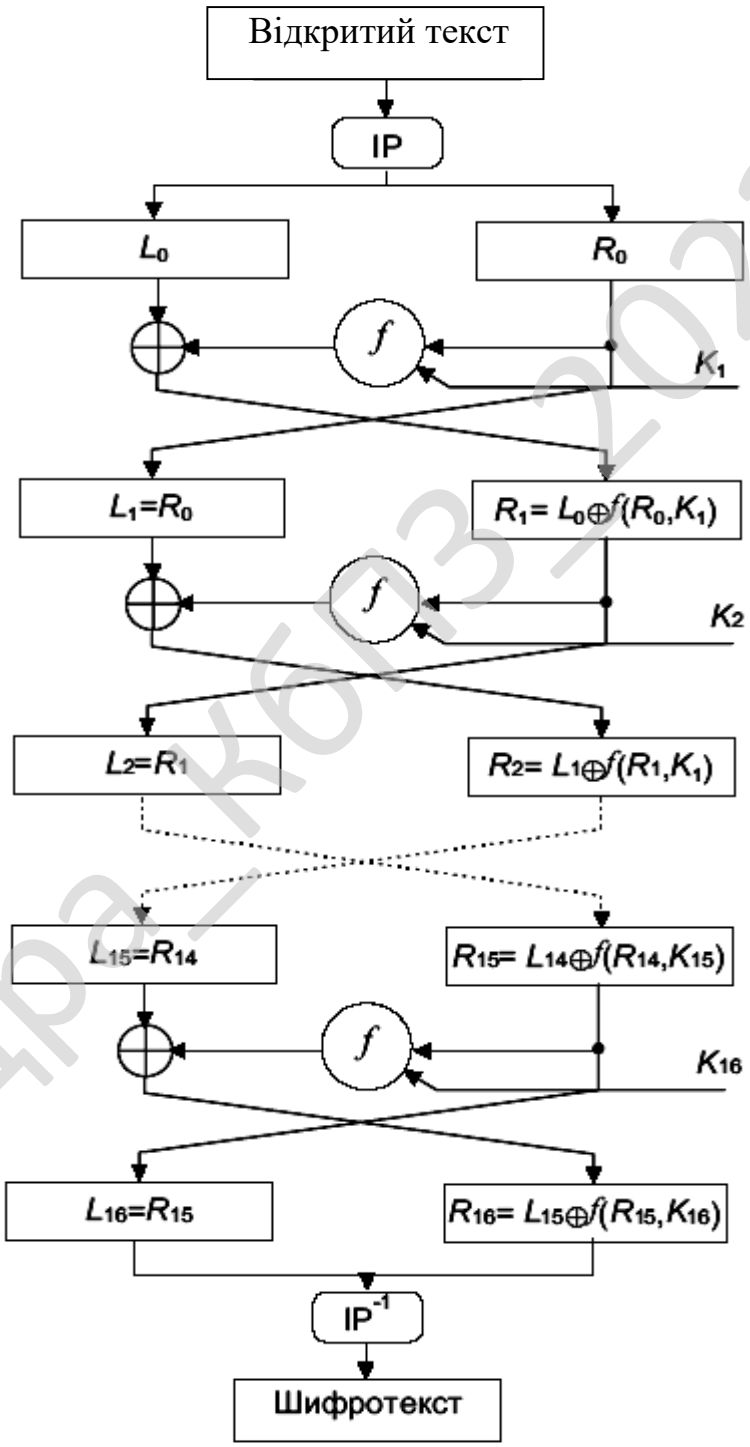


Рисунок 3.2 – Алгоритм DES

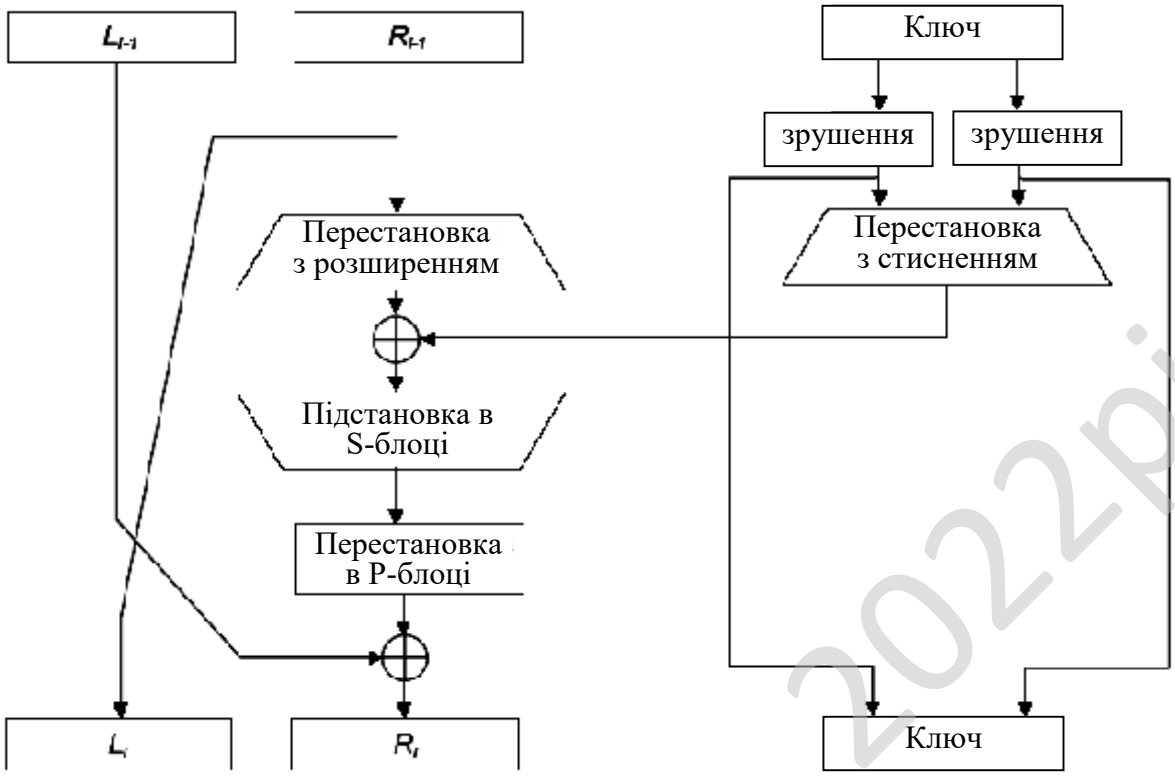


Рисунок 3.3 – Один етап DES

Якщо $B_i$ – це результат $i$-ої ітерації. $L_i$, і $R_i$, – ліва й права половини $B_i$, $K_i$ – 48-бітовий ключ для етапу $i$, а f – це функція, що виконують всі підстановки, перестановки й XOR з ключем, то етап можна представити як: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$.

**Початкова перестановка** виконується ще до етапу 1, при цьому вхідний блок переставляється.

Таблиця 3.1 – Початкова перестановка шифру DES

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

Початкова перестановка й відповідна заключна перестановка не впливають на безпеку DES. Так як програмна реалізація цієї багатобітної перестановки нелегка, у багатьох програмних реалізаціях DES початкова й заключна перестановки не використовуються. Хоча такий новий алгоритм не менш безпечний, чим DES, він не відповідає стандарту DES і, тому, не може називатися DES.

**Перетворення ключа:** 64-**б**ітовий ключ DES зменшується до 56-бітового ключа відкиданням кожного восьмого біта. Ці біти використовуються тільки для контролю парності, дозволяючи перевіряти правильність ключа. Після добування 56-бітового ключа для кожного з 16 етапів DES генерується новий 48-бітовий підключ і ці підключи, $K_i$, визначаються в такий спосіб:

Таблиця 3.2 – Перетворення ключа DES

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

По перше, 56-бітовий ключ ділиться на дві 28-бітових половинки. Потім, половинки циклічно зрушуються ліворуч на один або два біти залежно від етапу.

Таблиця 3.3 – Число біт зрушення залежно від етапу DES

| Етап | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Число | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Після зрушення вибирається 48 з 56 біт. Так як при цьому не тільки вибирається підмножина біт, але й змінюється їхній порядок, ця операція називається **перестановка зі стиском.** Її результатом є набір з 48 біт. Наприклад, біт зрушеного ключа в позиції 33 переміщається в позицію 35 результату, а 18-й біт зрушеного ключа відкидається.

Таблиця 3.4 – Перестановка зі стиском

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

Через зрушення для кожного підключа використовується відмінна підмножина біт ключа. Кожний біт використовується приблизно в 14 з 16 підключей, хоча не всі біти використовуються в точності однакове число раз

**Перестановка з розширенням:** ця операція розширює праву половину даних, $R_i$, від 32 до 48 біт. Так як при цьому не просто повторюються певні біти, але й змінюється їхній порядок, ця операція називається перестановкою з розширенням, У неї дві задачі: привести розмір правої половини у відповідність із ключем для операції XOR і одержати більше довгий результат, який можна буде стиснути в ході операції підстановки. Однак головний криптографічний зміст зовсім в іншому. За рахунок впливу одного біта на дві підстановки швидше зростає залежність біт результату від біт вихідних даних. Це називається **лавинним ефектом.** DES спроектований так, щоб якнайшвидше домогтися залежності кожного біта шифротексту від кожного біта відкритого тексту й кожного біта ключа.

Перестановка з розширенням показана на 9-й. Іноді вона називається Е-б**локом** (від expansion). для кожного 4-бітовий вхідний блоки перший і четвертий біт являють собою два біти вихідного блоку, а другий і третій біти – один біт вихідного блоку. В 7-й показано, які позиції результату відповідають яким позиціям вихідних даних. Наприклад, біт вхідного блоку в позиції 3 переміститься в позицію 4 вихідні блоки, а біт вхідного блоку в позиції 21 – у позиції 30 і 32 вихідного блоку.

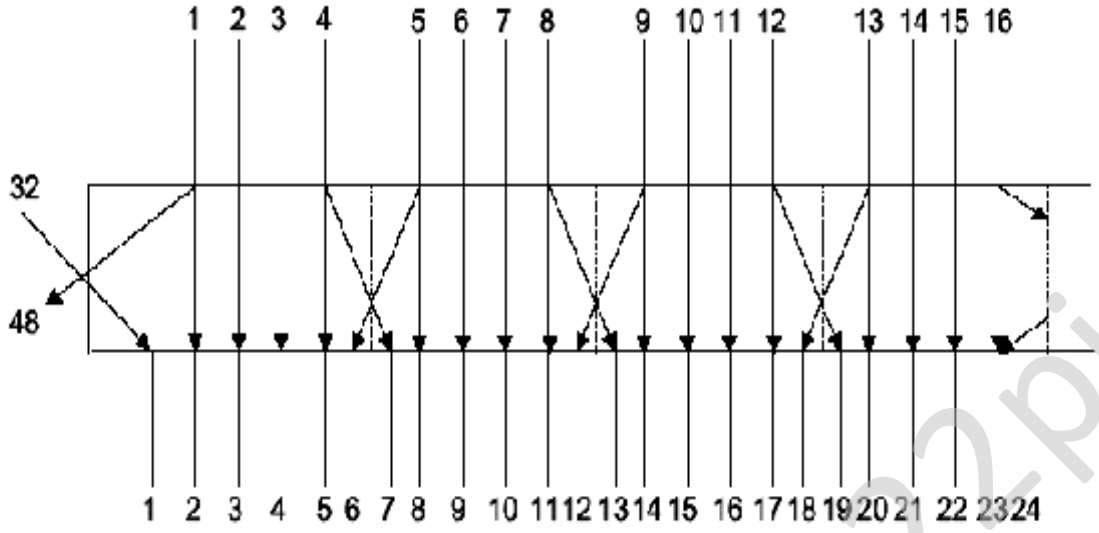


Рисунок 3.4 – Перестановка з розширенням

Хоча вихідний блок більше вхідного, кожний вхідний блок генерує унікальний вихідний блок

Таблиця 3.5 – Перестановка з розширенням DES

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

**Підстановка за допомогою S-блоків:** Після об'єднання стислого блоку з розширеним блоком за допомогою XOR над 48-бітовим результатом виконується операція підстановки. Підстановки виробляються у вісьмох блоках підстановки, або S-блоках (від Su*b*stitution). У кожного 8-блоку 6-бітовий вхід і 4-бітовий вихід, усього використовується вісім різних S-блоків. (для восьми S-блоків DES буде потрібно 256 байтів пам'яті.) 48 біт діляться на вісім 6-бітових підблока. Кожний окремий підблок обробляється окремим S-блоком: перший підблок – S-блоком 1, другий – S-блоком 2 і так далі.

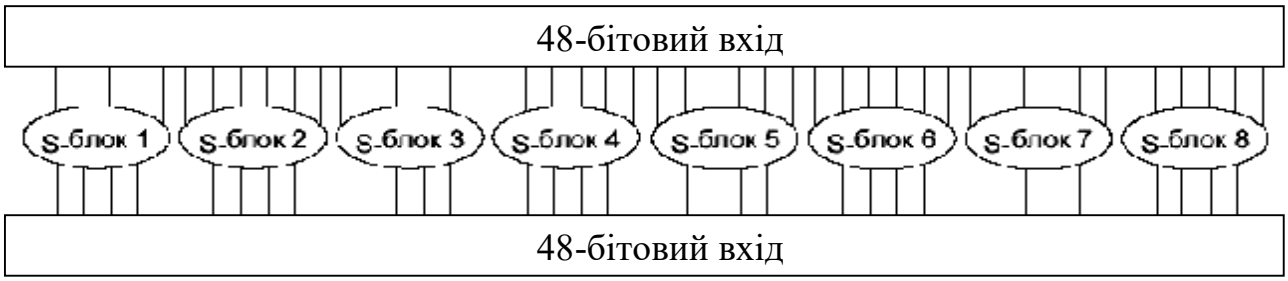


Рисунок 3.5 – Підстановка S-блоку

Кожний **S-блок** являє собою таблицю з 2 рядків і 16 стовпців. Кожний елемент у блоці є 4– бітовим числом. По 6 вхідних бітах S-блоку визначається, під якими номерами стовпців і рядків шукати вихідне значення.

**Перестановка за допомогою P-блоків:** 32-бітовий вихід підстановки за допомогою S-блоків, перетасовуються відповідно до P-блоку. Ця перестановка переміщає кожний вхідний біт в іншу позицію, жоден біт не використовується двічі, і жоден біт не ігнорується. Цей процес називається прямою перестановкою або просто перестановкою. Наприклад, біт 21 переміщається в позицію 4, а біт 4 – у позицію 31.

Таблиця 3.6 – перестановка за допомогою P-блоків

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

Нарешті, результат перестановки за допомогою P-блоку поєднується за допомогою XOR. с лівою половиною первісного 64-бітового блоку. Потім ліва й права половини міняються місцями, і починається наступний етап.

**Заключна перестановка** є зворотною відносно початкової перестановки. Зверніть увагу, що ліва й права половини не міняються місцями після останнього етапу DES, замість цього об'єднаний блок $R_{16}L_{16}$ використовується як вхід заключної перестановки. У цьому немає нічого особливого, перестановка половинок з наступним циклічним зрушенням привела б до точно такого ж

результату. Це зроблено для того, щоб алгоритм можна було використовувати як для шифрування, так і для дешифрування.

Таблиця 3.7 – Заключна перестановка DES

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

**Дешифрування DES**

Після всіх підстановок, перестановок, операцій XOR. і циклічних зрушень можна подумати, що алгоритм дешифрування, різко відрізняючись від алгоритму шифрування, точно так заплутаний. Навпроти, різні компоненти DES були підібрані так, щоб виконувалася дуже корисна властивість: для шифрування й дешифрування використовується той самий алгоритм. DES дозволяє використовувати для шифрування або дешифрування блоку ту саму функцію. Єдина відмінність полягає в тому, що ключі повинні використовуватися у зворотному порядку. Тобто, якщо на етапах шифрування використовувалися ключі $K_1$, $K_2$, $K_3$, ..., $K_{16}$, то ключами дешифрування будуть $K_{16}$, $K_{15}$, $K_{14}$, ..., $K_1$. Алгоритм, що створює ключ для кожного етапу, також циклічний. Ключ зрушується праворуч, а число позицій зрушення дорівнює 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

**Режими використання DES**

DES може використовується в чотирьох режимах.

1. Режим електронної кодової книги (ECB – Electronic Code Book): звичайне використання DES як блоковий шифр (рисунок 3.6). Кожний блок з 64 бітів незашифрованного тексту шифрується незалежно від інших блоків, із застосуванням того самого *ключа шифрування*. Типові додатки – безпечна передача одиночних значень (наприклад, криптографічного ключа).

3. Режим зчеплення блоків (CBC– Cipher Block Chaining) (рисунок 3.7). Вхід криптографічного алгоритму є результатом застосування операції XOR до наступного блоку незашифруванного тексту й попередньому блоку зашифруванного тексту. Кожний блок $C_i$ $i \geq 1$, перед черговим зашифруванням складається по модулю 2 з наступним блоком відкритого тексту $M_{i+1}$. Вектор $C_0$ – початковий вектор, він міняється щодня й зберігається в секреті. Типові додатки – загальна блокоорієнтована передача, автентифікація.

3. Режим зворотного зв'язка із шифртексту(CFB – Cipher Feed Back) (рисунок 3.8). У режимі CFB виробляється блокова "гама" $Z_0$, $Z_1$,...,$Z_i = DES_k(C_{i-1})$, $C_i = M_i \oplus Z_i$. Початковий вектор $C_0$ зберігається в секреті. Типові додатки – потокоорієнтована передача, автентифікація.

4. Режим зворотного зв'язку по виходу (OFB – Output Feed Back) (рисунок 3.9). аналогічний *CFB*, за винятком того, що на вхід алгоритму при шифруванні наступного блоку подається результат шифрування попереднього блоку; тільки після цього виконується операція XOR із черговими *i* бітами незашифруванного тексту. У режимі OFB виробляється блокова "гама" $Z_0,Z_1,... Z_i = DES_k(Z_{i-1})$, $C_i = M_i \oplus Z_i$, $i \geq 1$ Типові додатки – потокоорієнтована передача по зашумленому каналу (наприклад, супутниковий зв'язок).

Достоїнства й недоліки режимів.

Режим ECB просто реалізується, але може відбуватися проведення критоаналізу. У режимах ECB і OFB перекручування при передачі одного 64-бітового блоку шифртексту $C_i$ приводить до перекручування після розшифрування тільки відповідного відкритого блоку $M_i$ , тому такі режими використовуються для передачі по каналах зв'язку з великим числом перекручувань.

У режимах CBC і CFB перекручування при передачі одного блоку шифруванного тексту $C_i$ приводить до перекручування на приймачі не більше двох блоків відкритого тексту $M_i$, $M_{i+1}$ . Зміна $M_i$ приводиться до зміни всіх інших

блоків $M_{i+1}$, $M_{i+2}$,... Ця властивість використовується для вироблення коду автентифікації повідомлення.

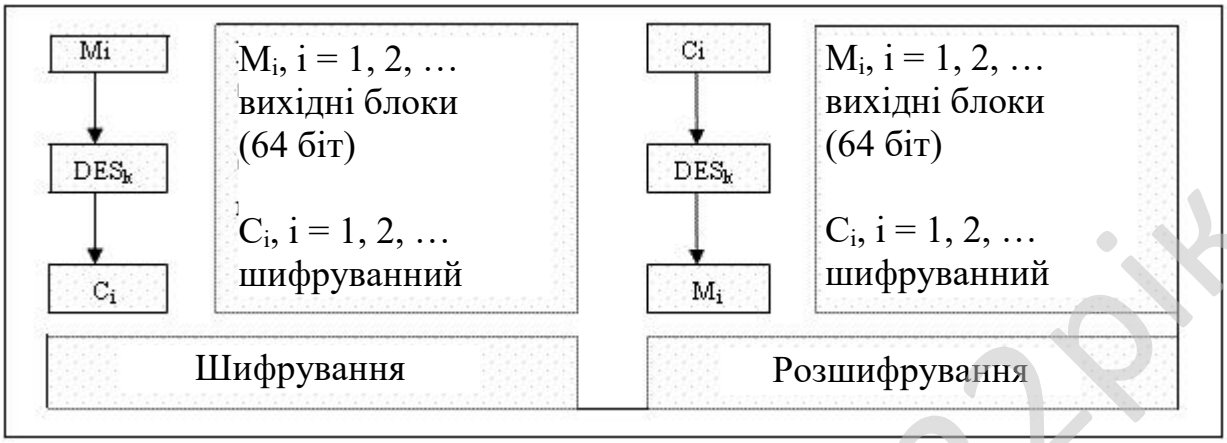


Рисунок 3.6 – Режим електронної кодової книги – ECB

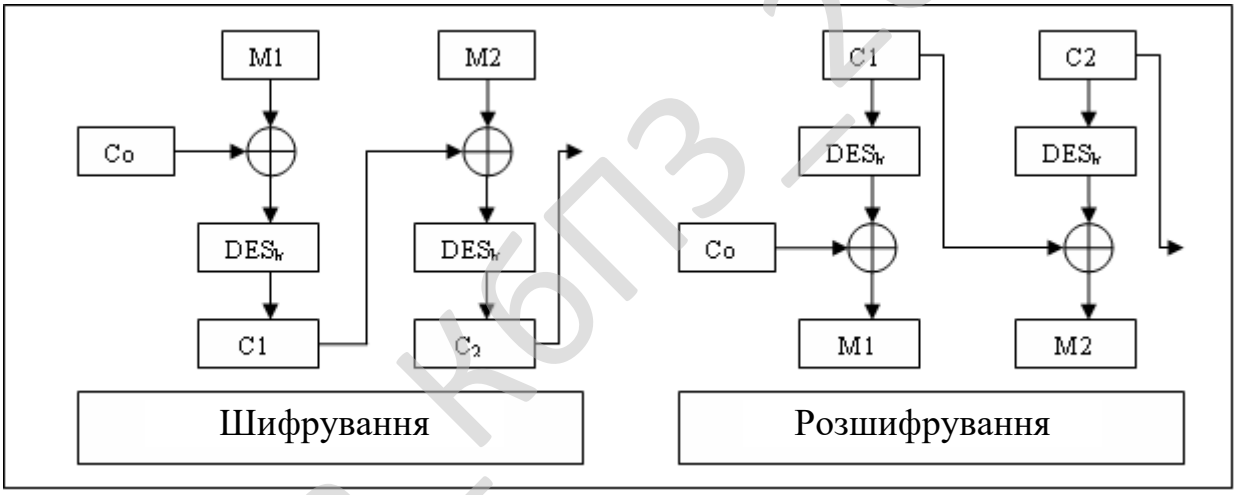


Рисунок 3.7– Режим зчеплення блоків – CBC

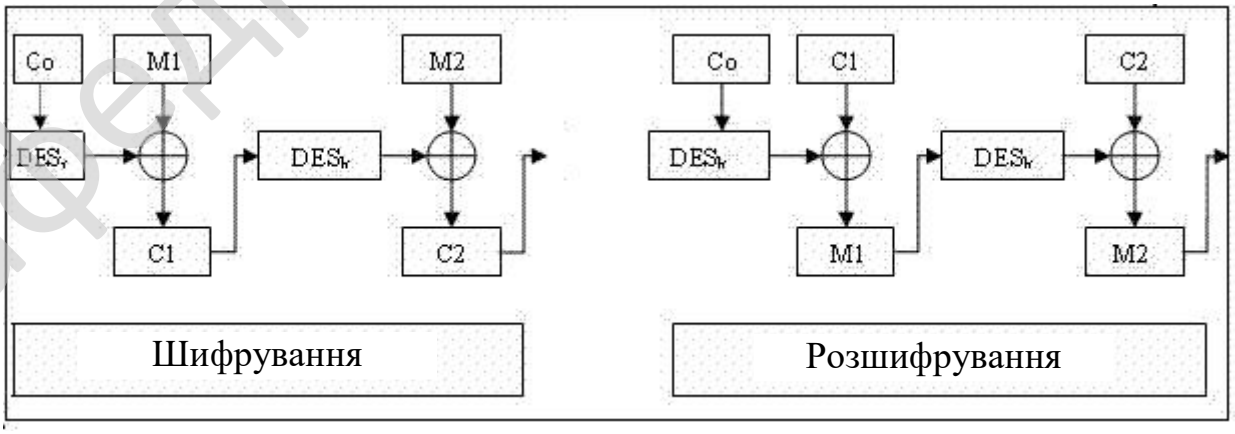


Рисунок 3.8 – Режим зворотного зв'язка із шифртексту – CFB

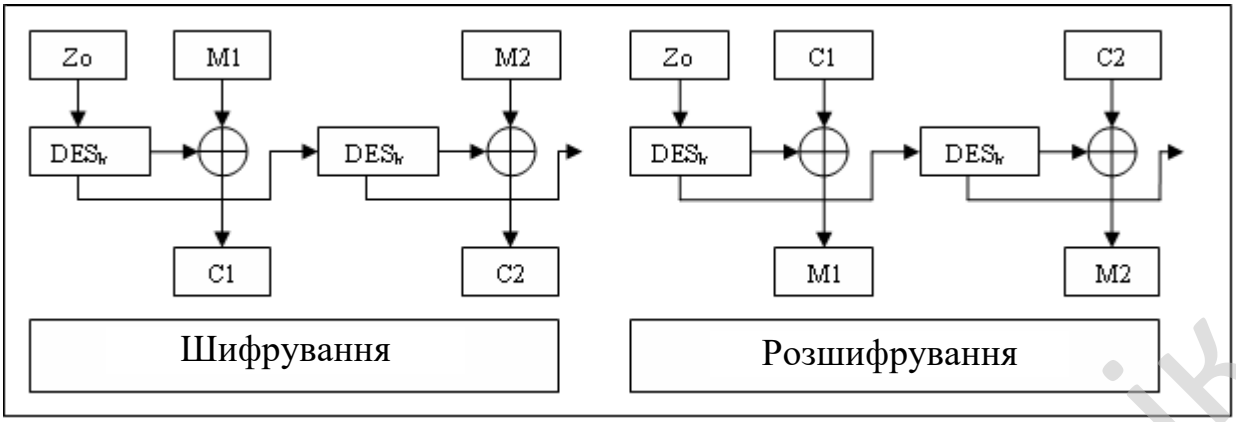


Рисунок 3.9 – Режим зворотного зв'язка по виходу – OFB

**Опис RSA**

Діффі й Хеллман визначили новий підхід до шифрування, що викликало до життя розробку алгоритмів шифрування, що задовольняють вимогам систем з відкритим ключем. Одним з перших результатів був алгоритм, розроблений в 1977 році Роном Ривестом, Ади Шамиром і Льоном Адлеманом і опублікований в 1978 році. З того часу алгоритм Rivest-Shamir-Adleman (RSA) широко застосовується практично у всіх додатках, що використовують криптографію з відкритим ключем.

Алгоритм заснований на використанні того факту, що задача **факторизації** (тобто розкладення числа на множники) є важкою, тобто легко перемножити два числа, у той час як не існує поліноміального алгоритму знаходження простих співмножників великого числа.

Алгоритм RSA являє собою блоковий алгоритм шифрування, де зашифруванні й незашифруванні дані є цілими між `0` і `n-1` для деякого `n`.

Дані шифруються блоками, кожний блок розглядається як число, менше деякого числа `n`. Шифрування і дешифрування мають наступний вигляд для деякого незашифруванного блоку M та зашифруванного блоку C:

$$C = M^e \pmod n, \tag{3.1}$$

$$M = C^d \pmod n = (M^e)^d \pmod n = M^{ed} \pmod n. \tag{3.2}$$

Як відправник, так і одержувач повинні знати значення n. Відправник знає значення e, одержувач знає значення d. Таким чином, відкритий ключ є

KU = {e, n} і закритий ключ є KR = {d, n}. При цьому повинні виконуватися наступні умови:

1. Можливість знайти значення e, d і n такі, що $M^{ed} = M \bmod n$ для всіх $M < n$.

2. Відносна легкість обчислення $M^e$ й $C^d$ для всіх значень $M < n$.

3. Неможливість визначити d, знаючи e та n.

**Створення ключів**

Вибрати прості p та q.

Обчислити $n = p \cdot q$.

Обчислити функцію Ейлера $\Phi(n) = (p - 1) * (q - 1)$.

Вибрати d взаємно просте з $\Phi(n)$: НСД $(\Phi(n), d) = 1$; $1 < d < \Phi(n)$.

Обчислити e: $d \cdot e \equiv 1 \bmod \Phi(n)$.

Відкритий ключ KU = {e, n} – публікується

Закритий ключ KR = {d, n} – тримається в таємниці

Примітка: p, q тримаються в таємниці, якщо вони стають відомі зловмиснику, то протокол дискредитовано.

**Шифрування**

Незашифрований текст: $M < n$.

Зашифрований текст: $C = M^e \pmod n$.

**Дешифрування**

Зашифрований текст: $C < n$.

Незашифрований текст: $M = C^d \pmod n$.

**Обчислювальні аспекти**

Розглянемо складність обчислень в алгоритмі RSA при створенні ключів і при шифруванні/дешифруванні.

**Шифрування/дешифрування**

Як шифрування, так і дешифрування включають піднесення цілого числа в цілий ступінь по модулю n. При цьому проміжні значення будуть величезними. Для того, щоб частково цього уникнути (тобто, щоб проміжні результати не

виходили за границі n), використовується наступна властивість модульної арифметики:

$$(a \cdot b) \bmod n = (a \bmod n) \cdot (b \bmod n). \quad (3.3)$$

Також алгоритм можна оптимізувати за допомогою ефективного використання показника ступеня, тому що у випадку RSA показники ступеня дуже великі. Припустимо, що необхідно обчислити $x^{16}$. Прямий підхід вимагає 15 множень. Однак можна домогтися того ж кінцевого результату за допомогою тільки чотирьох множень, якщо використовувати квадрат кожного проміжного результату: $x^2$, $x^4$, $x^8$, $x^{16}$.

**Створення ключів**

Створення ключів включає наступні задачі:

1. Визначити два простих числа p та q.

2. Вибрати e й обчислити d.

Насамперед, розглянемо проблеми, пов'язані з вибором p та q. Тому що значення $n = p \cdot q$ буде відомо будь-якому потенційному зловмисникові, для запобігання розкриття p та q ці прості числа повинні бути обрані з досить великої множини, тобто p та q повинні бути великими числами. З іншого боку, метод, що використовується для пошуку великого простого числа, повинен бути досить ефективним. У наш час не відомі алгоритми, які створюють довільно великі прості числа. Процедура, що використовується для цього, вибирає випадкове непарне число з необхідного діапазону й перевіряє, чи є воно простим. Якщо число не є простим, то знову вибирається випадкове число доти, поки не буде знайдено просте.

Були розроблені різні тести для визначення того, чи є число простим. Це тести імовірнісні, тобто тест показує, що дане число ймовірно є простим. Незважаючи на це, вони можуть виконуватися таким чином, що зроблять імовірність близькою до 1. Якщо *n* "провалює" тест, то воно не є простим. Якщо *n* "пропускає" тест, то *n* може як бути, так і не бути простим. Якщо *n* пропускає багато таких тестів, то можна з високим ступенем вірогідності сказати, що *n* є

простим. Це досить довга процедура, але вона виконується відносно рідко: тільки при створенні нової пари (KU, KR).

На складність обчислень також впливає те, яка кількість чисел буде відкинута перед тим, як буде знайдене просте число. Результат з теорії чисел, відомий як теорема простого числа, говорить, що простих чисел, розташованих біля $n$ у середньому одне на кожні ln ($n$) чисел. Таким чином, у середньому потрібно перевірити послідовність із ln ($n$) цілих, перш ніж буде знайдене просте число. Тому що всі парні числа можуть бути відкинуті без перевірки, то потрібно виконати приблизно ln ($n$)/2 перевірок. Наприклад, якщо просте число шукається в діапазоні величин $2^{200}$, то необхідно виконати біля ln ($2^{200}$) / 2 = 70 перевірок.

Вибравши прості числа p та $q$, далі необхідно вибрати значення $e$ так, щоб НСД(Ф($n$), $e$) = 1 і обчислити значення $d$, $d = e^{-1} \bmod Ф(n)$. Існує єдиний алгоритм, який називається розширеним алгоритмом Евкліда, що за фіксований час обчислює найбільший спільний дільник двох цілих і якщо цей спільний дільник дорівнює одиниці, визначає інверсне значення одного за модулем іншого. Таким чином, процедура полягає в генерації серії випадкових чисел і перевірці кожного відносно Ф($n$) доти, поки не буде знайдене число, взаємопросте з Ф($n$). Виникає питання, як багато випадкових чисел доведеться перевірити доти, поки не знайдеться потрібне число, що буде взаємопростим з Ф($n$). Результати показують, що ймовірність того, що два випадкових числа є взаємопростими, дорівнює 0.6.

### 3.2 Розробка структурної схеми

На рисунках 3.10 та 3.11 показані структурні схеми шифрування та дешифрування EFS. Більш докладно структурні зв’язки, які відбуваються у системі описані у пункті 3.1 даної магістерської роботи.

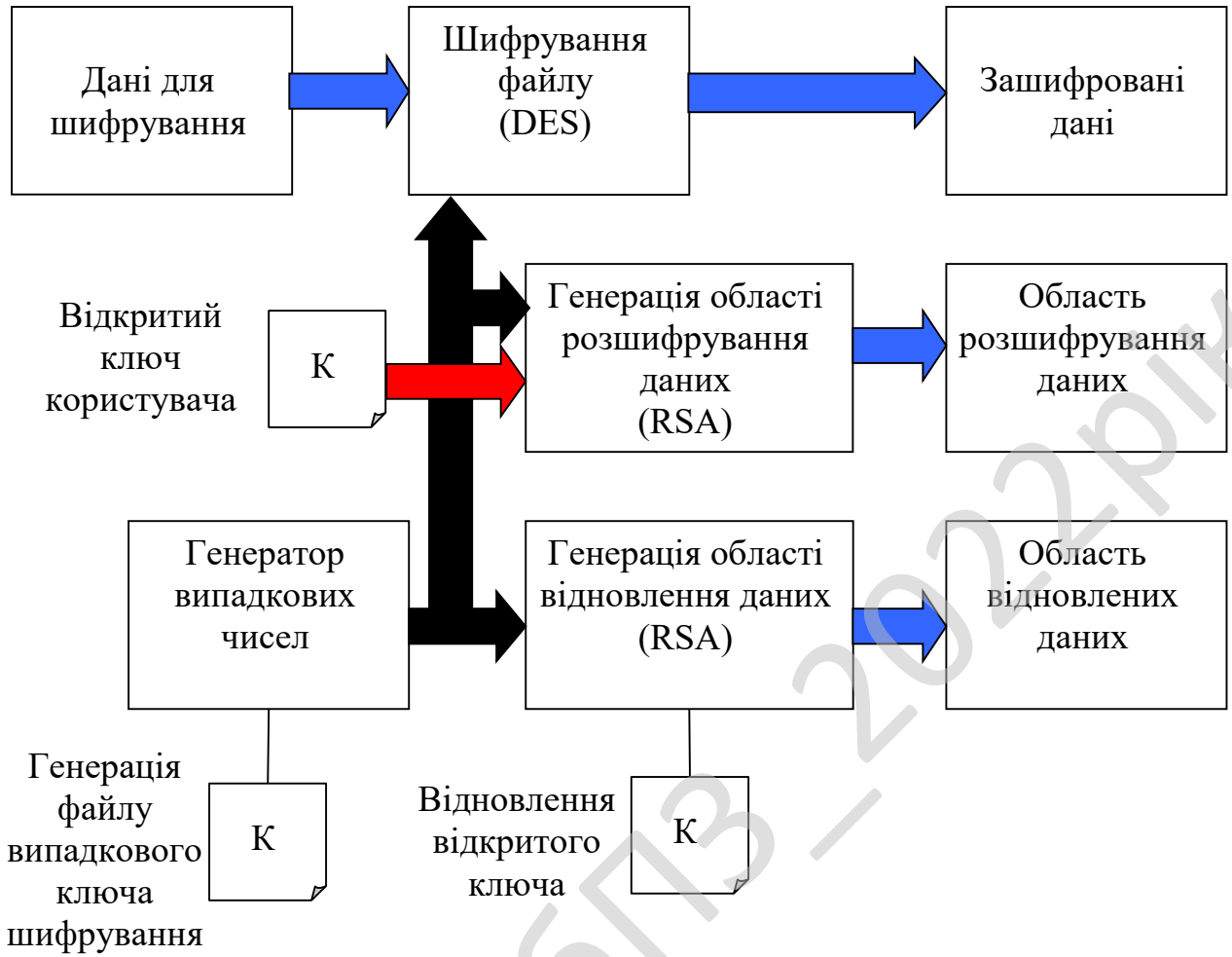


Рисунок 3.10 – Структурна схема шифрування EFS

Якщо коротко, то можливо сказати, що робота системи відбувається наступним чином. EFS працює, шифруючи кожний файл за допомогою алгоритму симетричного шифрування, що залежить від версії операційної системи й настроювань. При цьому використовується випадково-згенерований ключ для кожного файлу, називаний **File Encryption Key** (FEK), вибір симетричного шифрування на даному етапі пояснюється його швидкістю й більшою надійністю стосовно асиметричного шифрування. У данному магістерському проекті у якості симетричного алгоритму шифрування вибраний DES, у зв'язку з тим, що він є достатньо стійким та швидким.

FEK (випадковий для кожного файлу ключ симетричного шифрування) захищається шляхом асиметричного шифрування, що використовує відкритий ключ користувача файл, який шифрує, і алгоритм RSA (теоретично можливе використання інших алгоритмів асиметричного шифрування). RSA обраний тому, що він достаньо стійкий , для цих потреб, та виконується більш швидко, ніж інші алгоритми симметричного шифрування. Зашифрованний у такий спосіб ключ FEK зберігається в альтернативному потоці $EFS файлової системи NTFS. Для розшифрування даних, драйвер шифруванної файлової системи, прозоро для користувача, розшифровує FEK використовуючи закритий ключ користувача, а потім і необхідний файл за допомогою розшифрованного файлового ключа.

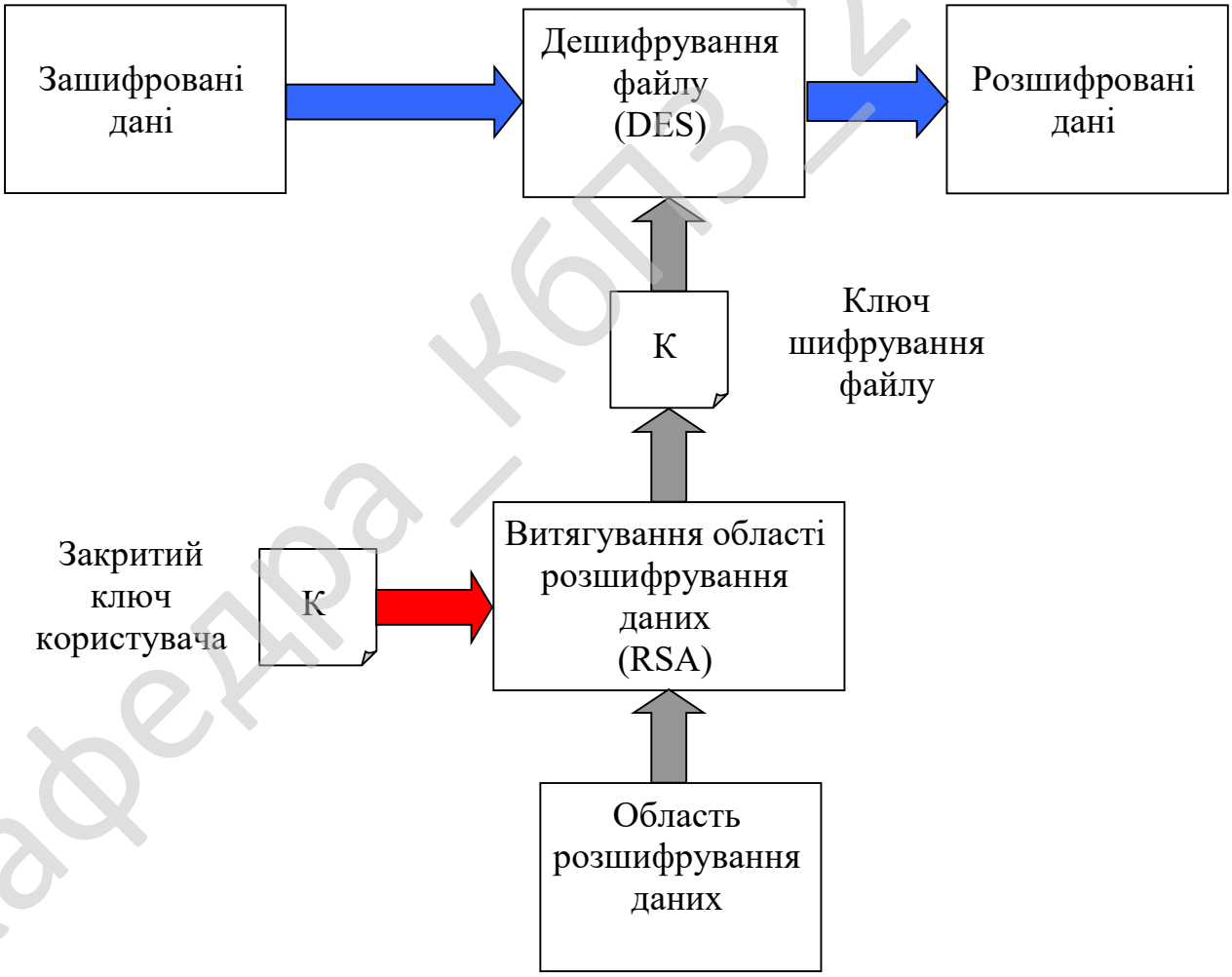


Рисунок 3.11 – Структурна схема дешифрування EFS

### 3.6 Розробка функціональної схеми

На рисунку 3.12 показана функціональна схема EFS.

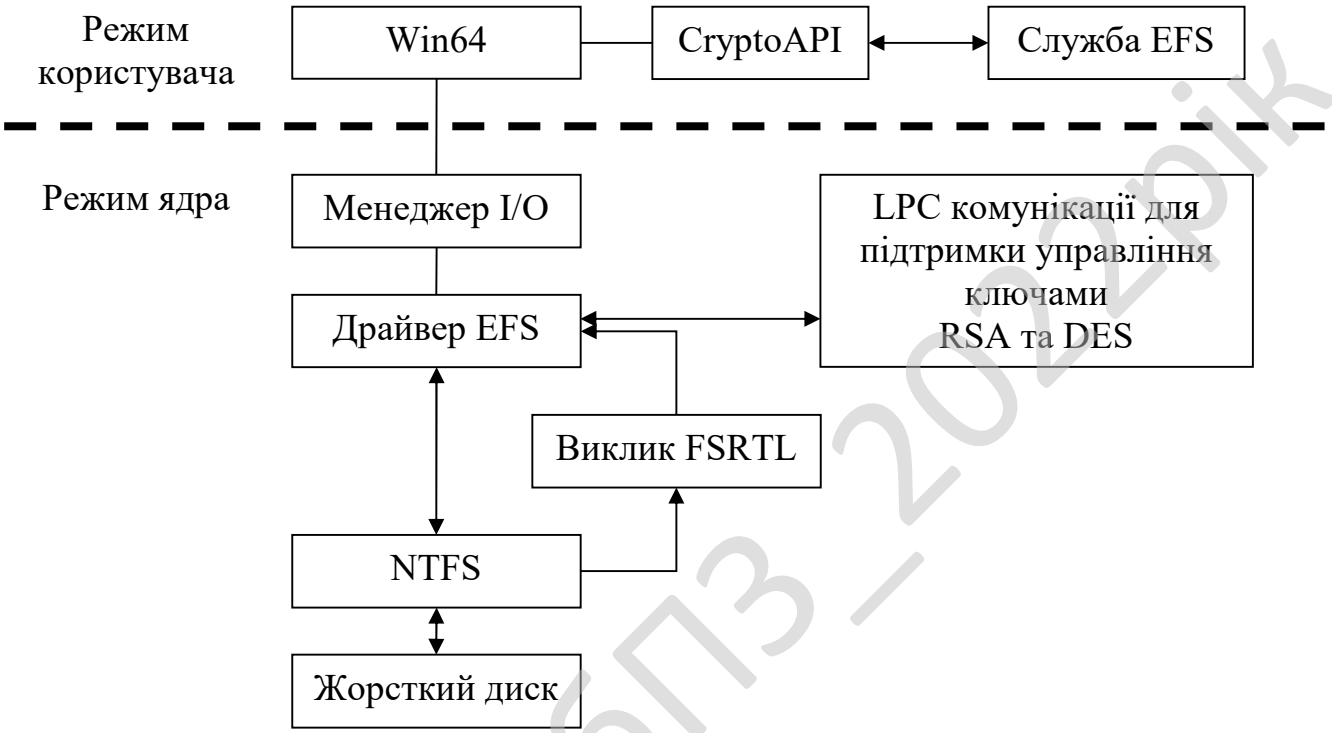


Рисунок 3.12 – Функціональна схема EFS

Функціонально EFS складається з наступних компонентів:

**Драйвер EFS**

Цей компонент розташований логічно на вершині NTFS. Він взаємодіє із сервісом EFS, одержує ключі шифрування файлів, поля DDF, DRF і інші дані керування ключами. Драйвер передає цю інформацію в FSRTL (file system runtime library, бібліотека часу виконання файлової системи) для прозорого виконання різних файлових системних операцій (наприклад, відкриття файлу, читання, запис, додавання даних у кінець файлу).

**Бібліотека часу виконання EFS (FSRTL)**

FSRTL – це модуль усередині драйвера EFS, що здійснює зовнішні виклики NTFS для виконання різних операцій файлової системи, таких як

читання, запис, відкриття зашифрованих файлів і каталогів, а також операцій шифрування, дешифрування, відновлення даних при записі на диск і читанні з диска. Незважаючи на те, що драйвер EFS і FSRTL реалізовані у вигляді одного компонента, вони ніколи не взаємодіють прямо. Для обміну повідомленнями між собою вони використовують механізм викликів NTFS. Це гарантує участь NTFS у всіх файлових операціях. Операції, реалізовані з використанням механізмів керування файлами, включають запис даних у файлові атрибути EFS (DDF і DRF) і передачу обчислених в EFS ключів FEK у бібліотеку FSRTL, тому що ці ключі повинні встановлюватися в контексті відкриття файлу. Такий контекст відкриття файлу дозволяє потім здійснювати непомітне шифрування й дешифрування файлів при записі й зчитуванні файлів з диска.

**Служба EFS**

Служба EFS є частиною підсистеми безпеки. Вона використовує існуючий порт зв'язку LPC між LSA (Local security authority, локальні засоби захисту) і працюючої в kernel-mode монітором безпеки для зв'язку із драйвером EFS.

Шина LPC (Low Pin Count або LPC bus**)** використовується в IBM PC-сумісних персональних комп'ютерах для підключення пристроїв, що не вимагають великої пропускної здатності до ЦП. До таких пристроїв ставляться завантажувальне ПЗП й контролери «застарілих» низькопродуктивних інтерфейсів передачі даних, такі як послідовний і паралельний інтерфейси, інтерфейс підключення маніпулятора «миша» і клавіатури, НЖМД, а з недавнього часу й пристроїв зберігання криптографічної інформації. Зазвичай контролер шини LPC розташований у південному мосту на материнській платі.

Шина LPC була введена фірмою Intel в 1998 році для заміни шини ISA. Хоча LPC фізично сильно відрізняється від ISA, програмна модель периферійних контролерів, що підключаються через LPC, залишилася колишньою. Це дозволило без доробок використовувати на комп'ютерах з LPC ПЗ, розроблене для керування периферійними контролерами, які підключалися до шини ISA.

У режимі користувача служба EFS взаємодіє із програмним інтерфейсом

CryptoAPI, надаючи ключі шифрування файлів і забезпечуючи генерацію DDF і DRF. Крім цього, служба EFS здійснює підтримку інтерфейсу Win64 API.

**Win64 API**

Забезпечує інтерфейс програмування для шифрування відкритих файлів, дешифрування й відновлення закритих файлів, прийому й передачі закритих файлів без їхньої попередньої розшифровки. Реалізований у вигляді стандартної системної бібліотеки advapi32.dll.

### 3.7 Розробка діаграми процесів

На рисунку 3.13 зображена діаграма процесів розробленої системи.

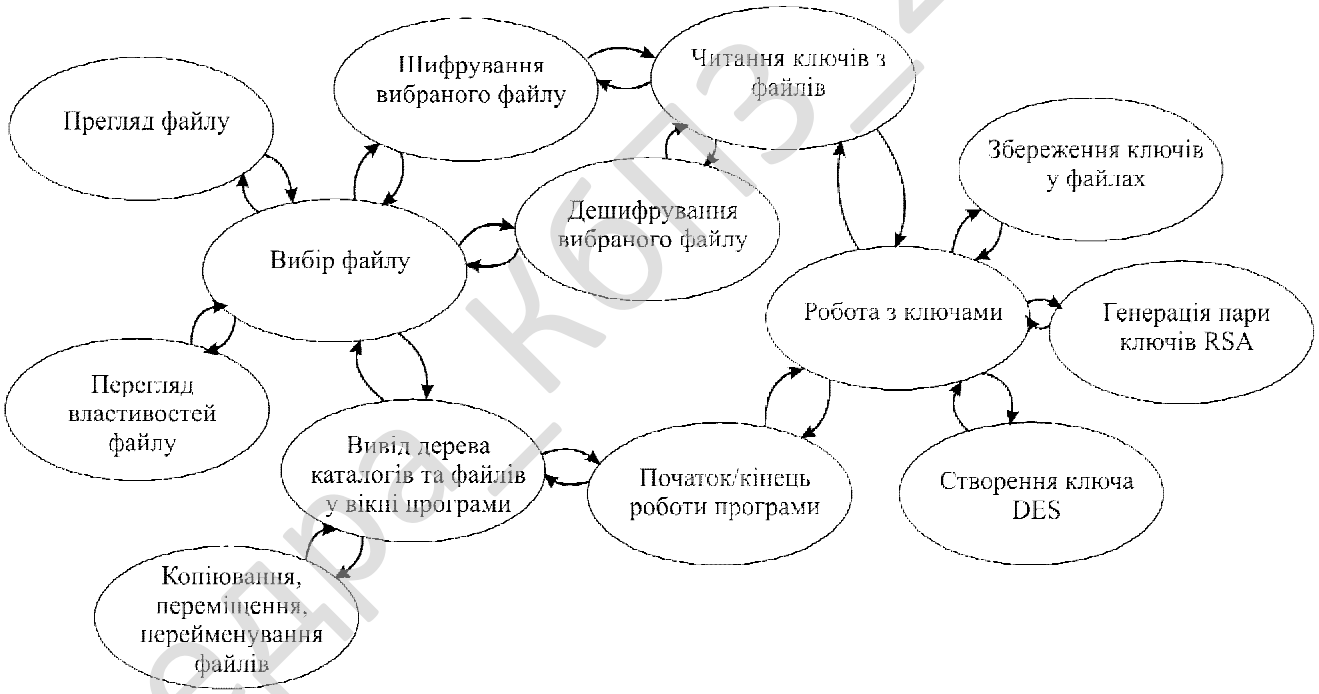


Рисунок 3.13 – Діаграма процесів системи

Першим у розробленій системі запускається процес виводу наявних на диску користувача каталогів та файлів у вікні програми. Потім для роботи з програмою користувач повинен створити ключ DES та згенерувати закритий та

відкритий ключі RSA. Створені ключі необхідно зберегти у файлах. Якщо ключі було створено раніше, то програмі необхідно вказати шлах до файлів з ними.

Користувач може копіювати, переміщювати, перейменовувати, відкривати та редагувати фали безпосередньо у вікні програми.

Для того, щоб захистити файл, необхідно виділити його, вказати програмі шлях до файлів ключів, за допомогою яких слід зашифрувати файл та запустити процес шифрування.

Для того, щоб розшифрувати файл, необхідно виділити його, вказати програмі шлях до файлів ключів, якими він був зашифрований та запустити процес дешифрування.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Блок-схема її роботи зображена на рисунку 4.1. Спочатку відбувається вивід на екран головного вікна програми та відображення в ньому дерева каталогів та файлів, що знаходяться на диску користувача.

Потім користувач може виконувати наступні дії:

- Створити нові ключі шифрування.
- Завантажити раніше створені ключі шифрування.
- Шифрувати файли.
- Дешифрувати файли.

Створення ключів шифрування відбувається наступним чином: спочатку створюється ключ DES, потім генеруються закритий та відкритий ключі RSA. Створені ключі зберігаються у файлах.

Для завантаження ключів слід вказати програмі шлях до файлів, що містять раніше створені ключі.

Для здійснення шифрування слід виділити у списку файлів ті, які необхідно зашифрувати, та запустити процес шифрування. Файли будуть зашифровані за допомогою вказаних програмі ключів.

Для розшифрування файлів, їх треба знову ж таки виділити та вказати програмі файли з ключами, якими вони були зашифровані. Після цього слід запустити процес дешифрування.

Якщо при дешифруванні були вказані вірні ключі, файл буде успішно розшифрований. У разі вказання невірних ключів, спроба зазнає невдачі.

Важливим моментом є правильне зберігання ключів. Ніхто крім власника не повинен мати доступ до закритого ключа RSA.

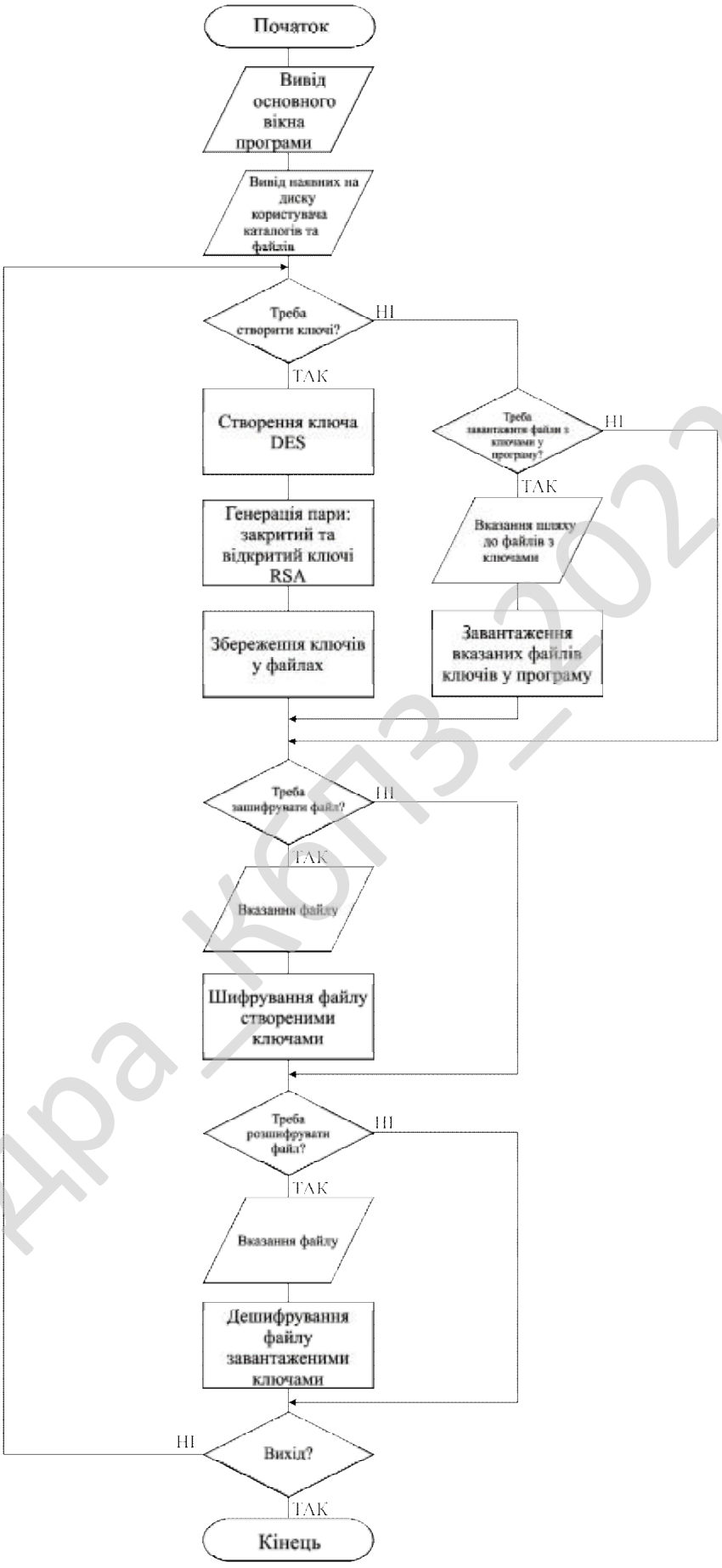


Рисунок 4.1 – Блок-схема роботи основної програми

**Технологія шифрування**

EFS використовує архітектуру Windows CryptoAPI. У її основі лежить технологія шифрування з відкритим ключем. Для шифрування кожного файлу випадковим чином генерується ключ шифрування файлу. При цьому для шифрування файлу може застосовуватися будь-який симетричний алгоритм шифрування. В EFS розробленої в даному магістерському проекті використовується алгоритм розповсюдженого стандарту DES.

Ключі шифрування EFS зберігаються в резидентному пулі пам'яті (сама EFS розташована в ядрі Windows 10/11), що виключає несанкціонований доступ до них через файл підкачування.

**Взаємодія з користувачем**

За замовчуванням EFS зконфігурована таким чином, що користувач може відразу почати використовувати шифрування файлів. Операція шифрування й зворотна підтримуються для файлів і каталогів. У тому випадку, якщо шифрується каталог, автоматично шифруються всі файли й підкаталоги цього каталогу. Необхідно відзначити, що якщо зашифрований файл переміщається або перейменовується із зашифрованого каталогу в незашифрований, то він однаково залишається зашифрованим.

Для того щоб зашифрувати каталог, користувачеві потрібно просто вибрати один або кілька каталогів і встановити прапорець шифрування у вікні розширених властивостей каталогу. Всі створювані пізніше файли й підкаталоги в цьому каталозі будуть також зашифровані. Таким чином, зашифрувати файл можна, просто скопіювавши (або перенесучи) його в "зашифрований" каталог.

Зашифровані файли зберігаються на диску в зашифрованому виді. При читанні файлу дані автоматично розшифровуються, а при записі – автоматично шифруються. Користувач може працювати із зашифрованими файлами так само, як і зі звичайними файлами, тобто відкривати й редагувати в текстовому редакторі Microsoft Word документи, редагувати малюнки в Adobe Photoshop або графічному редакторі Paint, і так далі.

Необхідно відзначити, що в жодному разі не можна шифрувати файли, які використовуються при запуску системи – у цей час особистий ключ користувача, за допомогою якого здійснюється дешифрування, ще недоступний. Це може привести до неможливості запуску системи! В EFS передбачена простий захист від таких ситуацій: файли з атрибутом "системний" не шифруються. Однак будьте уважні: це може створити "діру" у системі безпеки! Перевіряйте, чи не встановлений атрибут файлу "системний" для того, щоб переконатися, що файл дійсно буде зашифрований.

Важливо також пам'ятати про те, що зашифровані файли не можуть бути стиснуті засобами Windows 10/11 і навпаки. Іншими словами, якщо каталог стислий, його вміст не може бути зашифровано, а якщо вміст каталогу зашифрований, то він не може бути стислий.

У тому випадку, якщо буде потрібно дешифрування даних, необхідно просто зняти прапорці шифрування в обраних каталогів і файли й підкаталоги автоматично будуть дешифровані. Слід зазначити, що ця операція звичайно не потрібна, тому що EFS забезпечує "прозору" роботу із зашифрованими даними для користувача.

**Відновлення даних**

EFS забезпечує вбудовану підтримку відновлення даних на той випадок, якщо буде потрібно їх розшифрувати, але, за якимись причинами, це не може бути виконано звичайним шляхом. За замовчуванням, EFS автоматично згенерує ключ відновлення, установить сертифікат доступу в обліковому записі адміністратора й збереже його при першому вході в систему. Таким чином, адміністратор стає так званим агентом відновлення, і зможе розшифрувати будь-який файл у системі. Зрозуміло, політику відновлення даних можна змінити, і призначити як агент відновлення спеціальну людину, відповідальну за безпеку даних, або навіть кілька таких осіб.

Розглянемо більш детально алгоритми роботи підпрограм генерації ключів RSA, шифрування та дешифрування файлів (рисунки 4.2 – 4.4).


Рисунок 4.2 – Блок-схема роботи підпрограми генерації ключів RSA

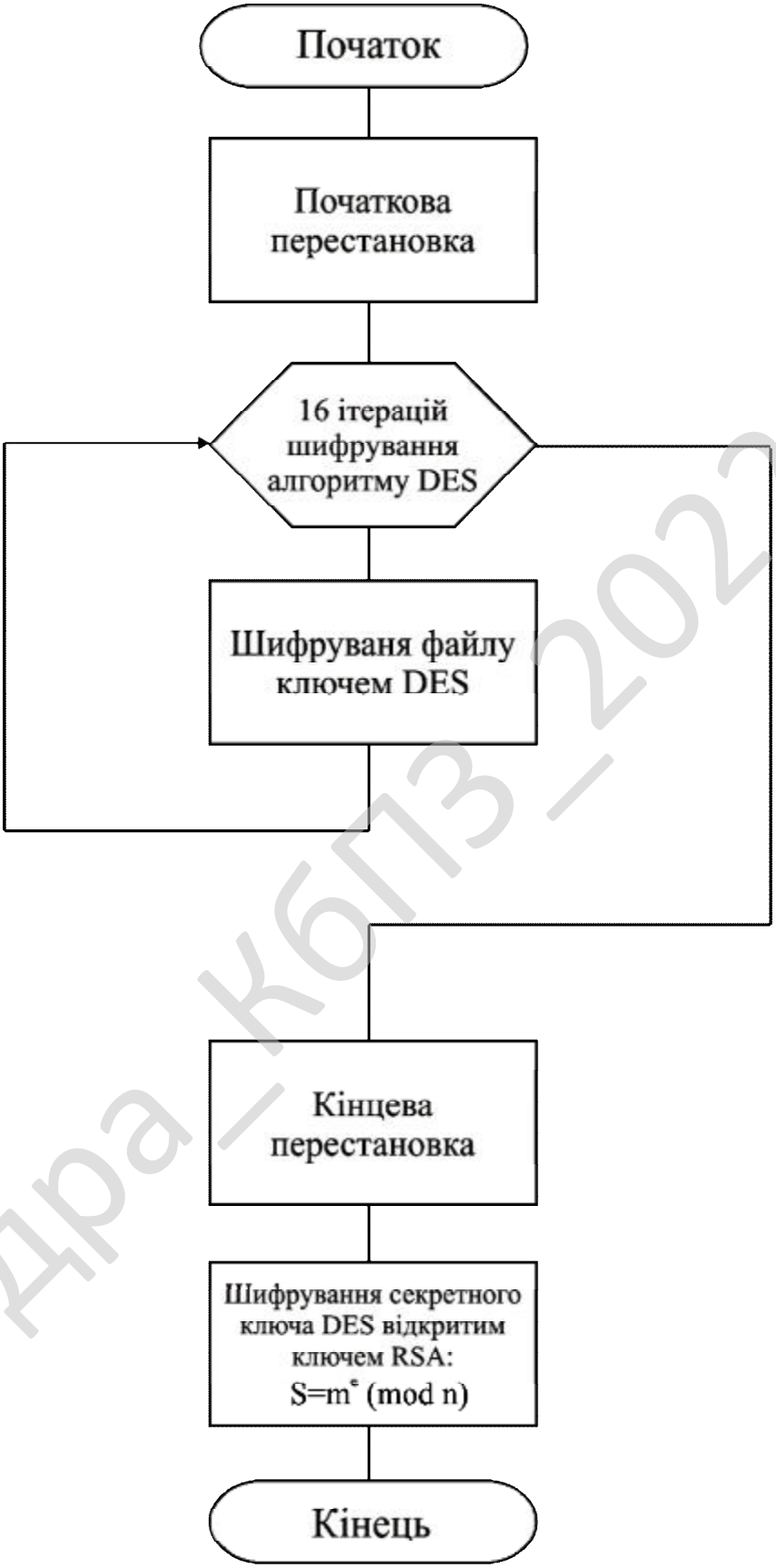


Рисунок 4.3 – Блок-схема роботи підпрограми шифрування

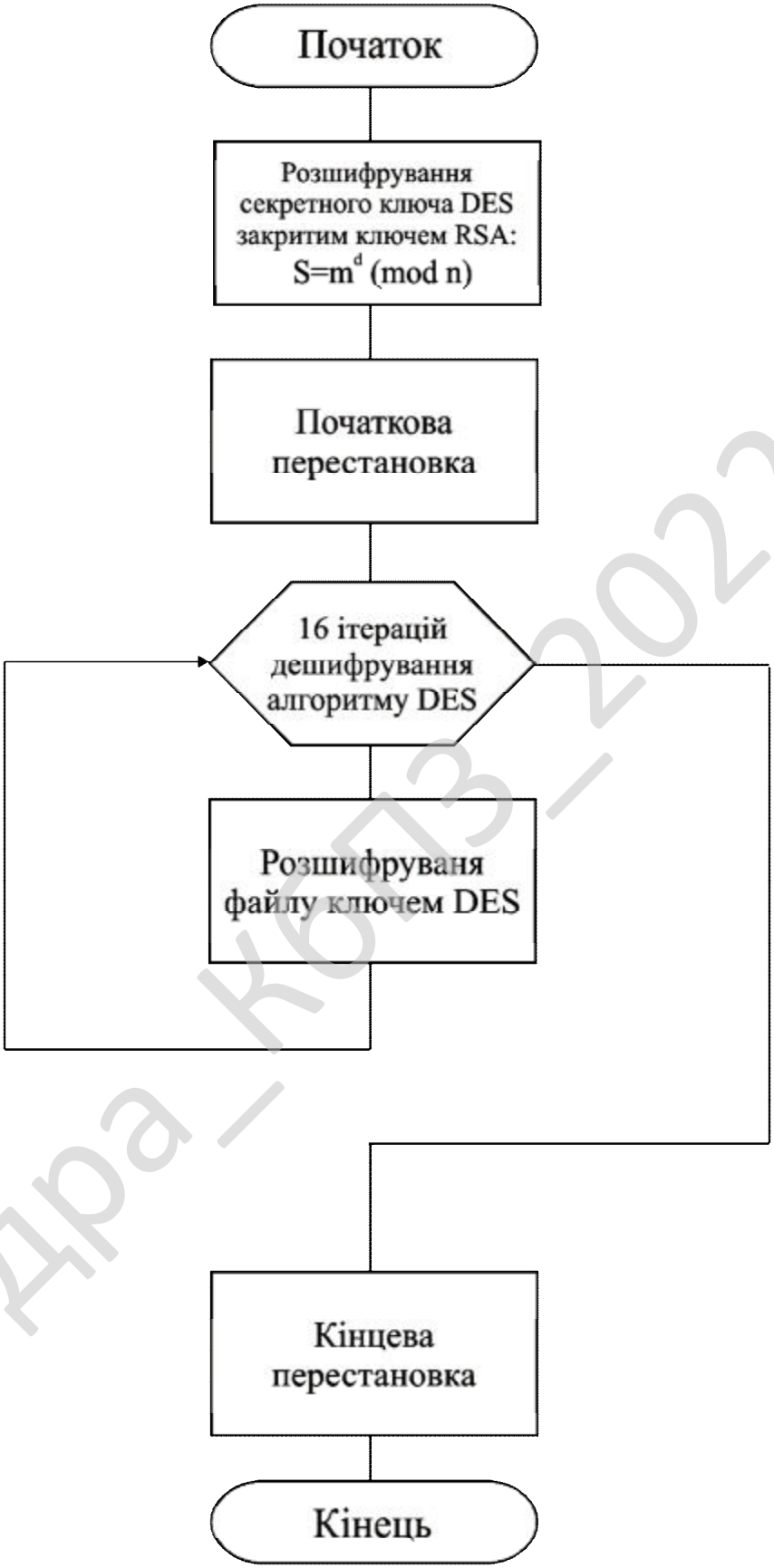


Рисунок 4.4 – Блок-схема роботи підпрограми дешифрування

Процедура генерації ключів RSA виглядає наступним чином.

```
procedure TRSA_keys.Button1Click(Sender: TObject);
begin
// Генерація p та q
   Base256StringToFGInt('3557', p);
   Base256StringToFGInt('2579', q);
   PrimeSearch(p);
   PrimeSearch(q);
   FGIntToBase256String(p, st);
   FGIntToBase256String(q, st);
 // Обчислення N
   FGIntMul(p, q, n);
   p.Number[1] := p.Number[1] - 1;
   q.Number[1] := q.Number[1] - 1;
   FGIntMul(p, q, phi);
   FGIntToBase10String(n, st);
   Edit_N.Text:=st;
// Обчислення E - ключ шифрування
//   Base10StringToFGInt('65537', e); // непарне число
//   Base10StringToFGInt('8171921', e);
   Base10StringToFGInt('14486581214143', e);
   Base10StringToFGInt('1', one);
   Base10StringToFGInt('2', two);
   FGIntGCD(phi, e, gcd);
   While FGIntCompareAbs(gcd, one) <> Eq Do
   Begin
      FGIntadd(e, two, temp);
      FGIntCopy(temp, e);
      FGIntGCD(phi, e, gcd);
   End;
   FGIntDestroy(two);
   FGIntDestroy(one);
   FGIntDestroy(gcd);
 // Обчислення D - ключ дешифрування
   FGIntModInv(e, phi, d);
   FGIntModInv(e, p, dp);
   FGIntModInv(e, q, dq);
   p.Number[1] := p.Number[1] + 1;
   q.Number[1] := q.Number[1] + 1;
   FGIntDestroy(phi);
   FGIntDestroy(nilgint);
```

```
   FGIntToBase10String(e, st);
   Edit_E.Text:=st;
   FGIntToBase10String(d, st);
   Edit_D.Text:=st;
end;
```

Процедури шифрування, дешифрування ключа DES ключем RSA та запис ключа DES у файл виглядають наступним чином.

```
//шифрування ключа DES
procedure TRSA_keys.Button4Click(Sender: TObject);
begin
 RSADecrypt(test, d, n, Nilgint, Nilgint, Nilgint, Nilgint, test);
       Edit3.Text:=test;
end;
//дешифрування ключа DES
procedure TRSA_keys.Button2Click(Sender: TObject);
var prom: string;
begin
   test := Edit3.Text;
   RSAEncrypt(test, e, n, test);
   Edit1.Text:=test;
end;
//запис ключів RSA у файли
procedure TRSA_keys.Button3Click(Sender: TObject);
begin
  AssignFile(tx1,'OpenKey.keys');
  AssignFile(tx2,'CloseKey.keys');
  ReWrite(tx1); CloseFile(tx1);
  ReWrite(tx2); CloseFile(tx2);
  Append(tx1);
  Append(tx2);
  FGIntToBase256String(n, st);
  ConvertBase256to64(st, st);
  WriteLn(tx1,st);
  WriteLn(tx2,st);
  FGIntToBase256String(e, st);
  ConvertBase256to64(st, st);
  WriteLn(tx1,st);
  FGIntToBase256String(d, st);
  ConvertBase256to64(st, st);
  WriteLn(tx2,st);
  CloseFile(tx1);
```

```
  CloseFile(tx2);
end;
//Запис ключа DES у файл
procedure TRSA_keys.Button5Click(Sender: TObject);
begin
AssignFile(tx3,'SecretKey.keys');
  ReWrite(tx3); CloseFile(tx3);
  Append(tx3);
  WriteLn(tx3,Edit1.Text);
  CloseFile(tx3);
end;
```

### 4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Madryga. Алгоритм Madryga складається із двох вкладених циклів. Зовнішній цикл повторюється вісім разів (для гарантії надійності число циклів можна збільшити) і полягає в застосуванні внутрішнього циклу до відкритого тексту. Внутрішній цикл перетворює відкритий текст у шифртекст і виконується однократно над кожним 8-бітовим блоком (байтом) відкритого тексту. Таким чином, весь відкритий текст послідовно вісім разів обробляється алгоритмом.

Ітерація внутрішнього циклу оперує з 3-байтовим вікном даних, називаним робочим кадром (рисунок 4.5). Це вікно зрушується на 1 байт за ітерацію. (При роботі з останніми 2 байтами дані покладаються циклічно замкнутими). Перші два байти робочого кадру циклічно зрушуються на змінне число позицій, а для останнього байта виконується операція XOR з декількома бітами ключа. У міру переміщення робочого кадру всі байти послідовно циклічно зрушуються й піддаються операції XOR із частинами ключа. Послідовні циклічні зрушення перемішують результати попередніх операцій XOR і циклічного зрушення, причому на циклічне зрушення впливають результати XOR. Завдяки цьому процес у цілому оборотний.

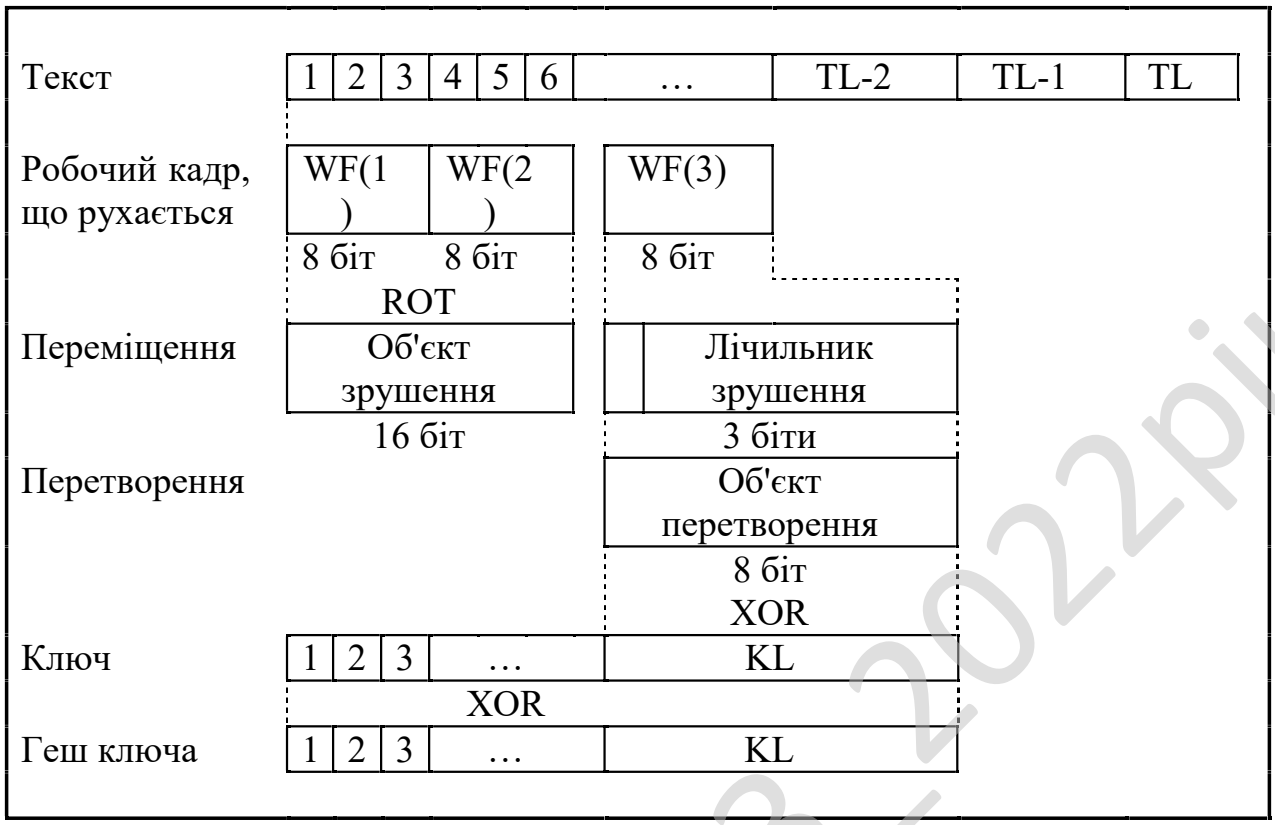


Рисунок 4.5 – Одна ітерація алгоритму Madryga

Оскільки кожний байт даних впливає на два байти ліворуч і на один байт праворуч від себе, після восьми проходів кожний байт шифртексту залежить від 16 байтів ліворуч і 8 байтів праворуч.

При шифруванні кожна ітерація внутрішнього циклу встановлює робочий кадр на передостанній байт відкритого тексту й циклічно переміщає його до третього з кінця байту відкритого тексту. Спочатку весь ключ піддається операції XOR з випадковою константою й потім циклічно зрушується вправо на 3 біти (ключ і дані рухаються в різних напрямках, щоб мінімізувати надлишкові операції з бітами ключа). Молодші три біти молодшого байта робочого кадру зберігаються, вони визначають циклічне зрушення інших двох байтів. Далі конкатенація двох старших байтом циклічно зрушується вліво на змінне число біт (від 0 до 7). Потім над молодшим байтом робочого кадру виконується

операція XOR з молодшим байтом ключа. Нарешті робочий кадр зміщається вправо на один байт і весь процес повторюється.

Випадкова константа призначена для перетворення ключа в псевдовипадкову послідовність. Довжина константи повинна бути рівній довжині ключа. При обміні даними абоненти повинні користуватися однієї й тією же константою. Для 64-бітового ключа Madryga рекомендує константу 0x0fle2d3c4b5a6978.

При розшифруванні процес повторюється у зворотному порядку. У кожній ітерації внутрішнього циклу робочий кадр установлюється на байт, третій ліворуч від останнього байта шифртексту, і циклічно зрушується у зворотному напрямку до байта, розташованого на 2 байти уліво відносно останнього байта шифртексту. 2 байти шифртексту в процесі циклічно зрушуються вправо, а ключ – уліво. Після циклічних зрушень виконується операція XOR.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Головне вікно програми зображене на рисунку 5.1.

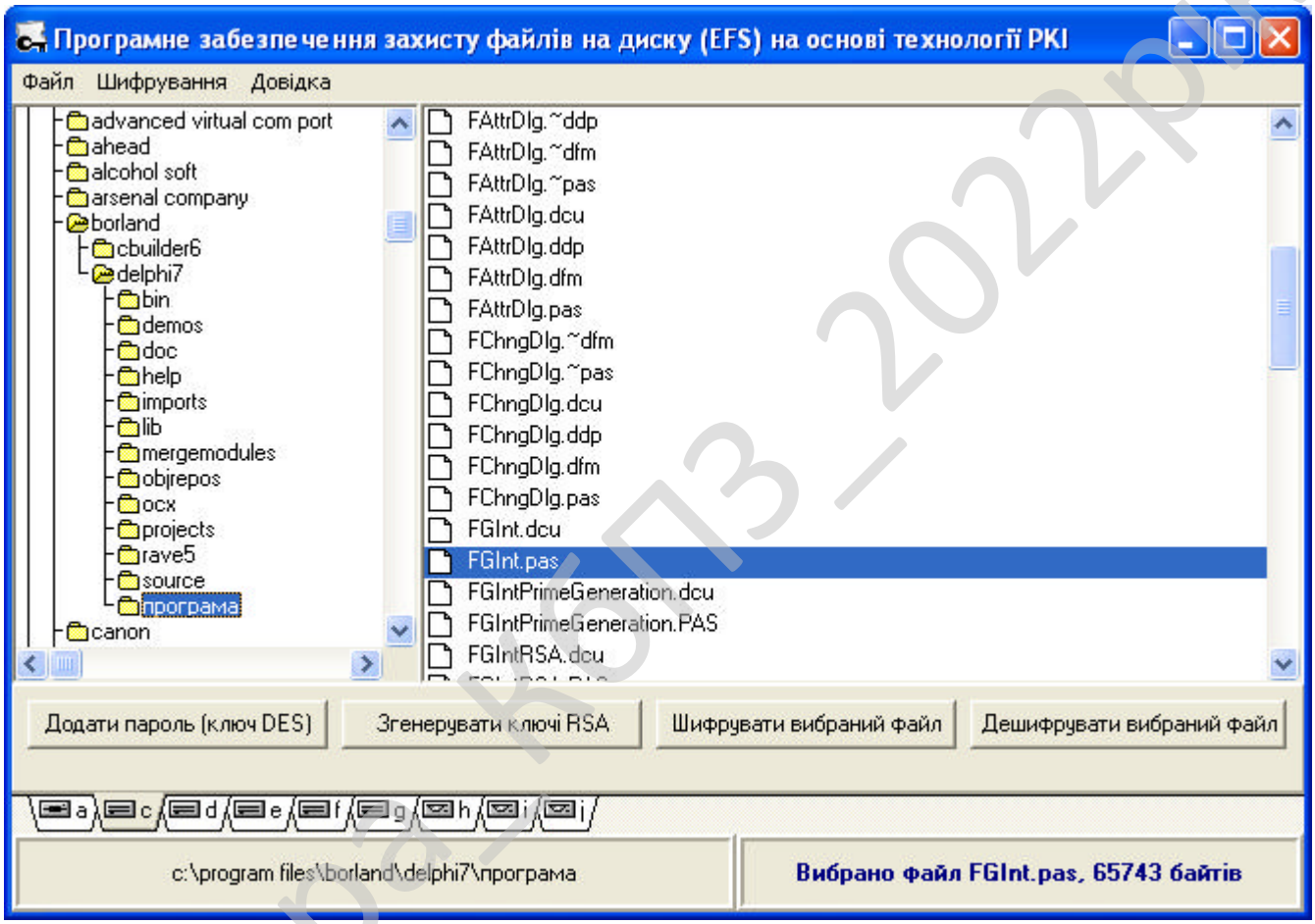


Рисунок 5.1 – Головне вікно програми

Як видно з рисунку програма виводить на екран дерево каталогів та файлів на вказаному диску та дозволяє шифрувати/дешифрувати виділені файли.

Головне вікно містить наступні кнопки:

– Додати пароль (ключ DES).

– Згенерувати ключі RSA.

– Шифрувати вибраний файл.

– Дешифрувати вибраний файл.

Більш повний перелік функцій містить меню користувача (рисунки 5.2-5.4).

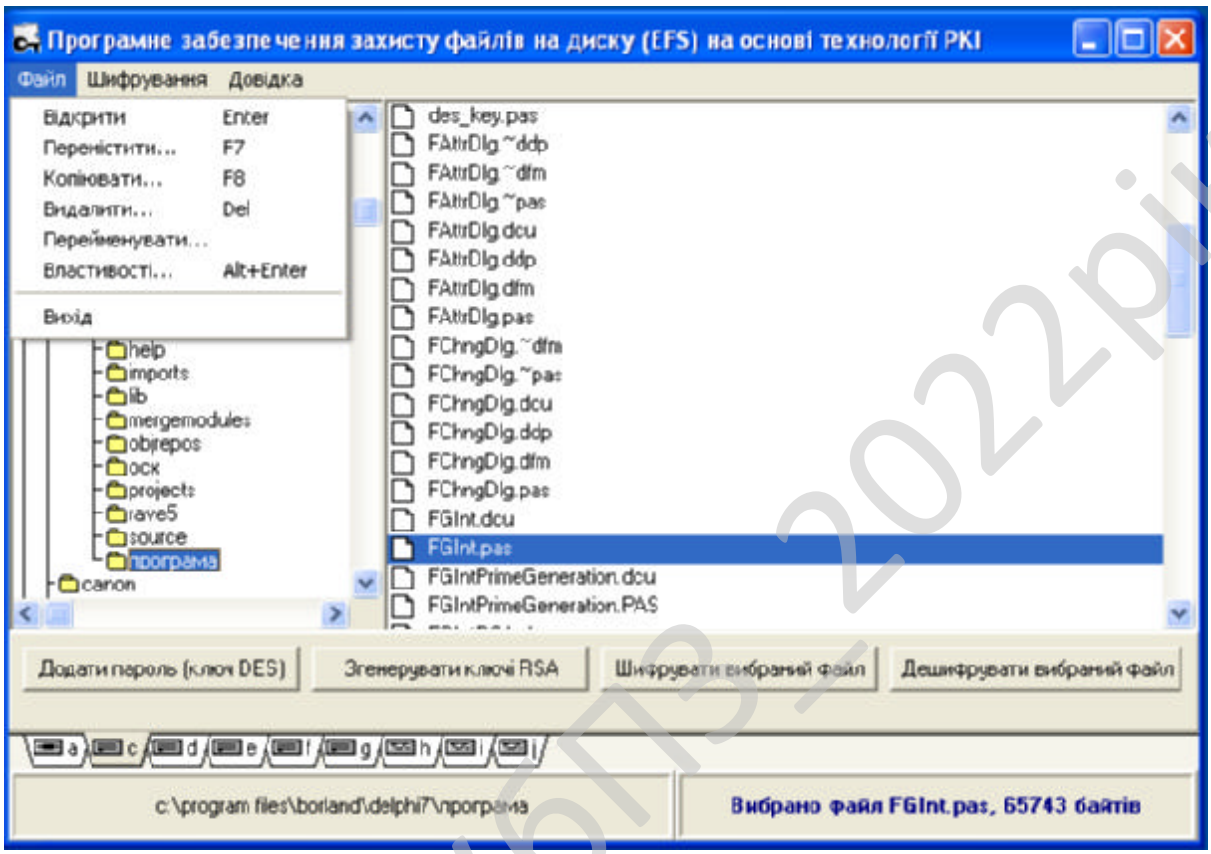


Рисунок 5.2 – Меню користувача "Файл"

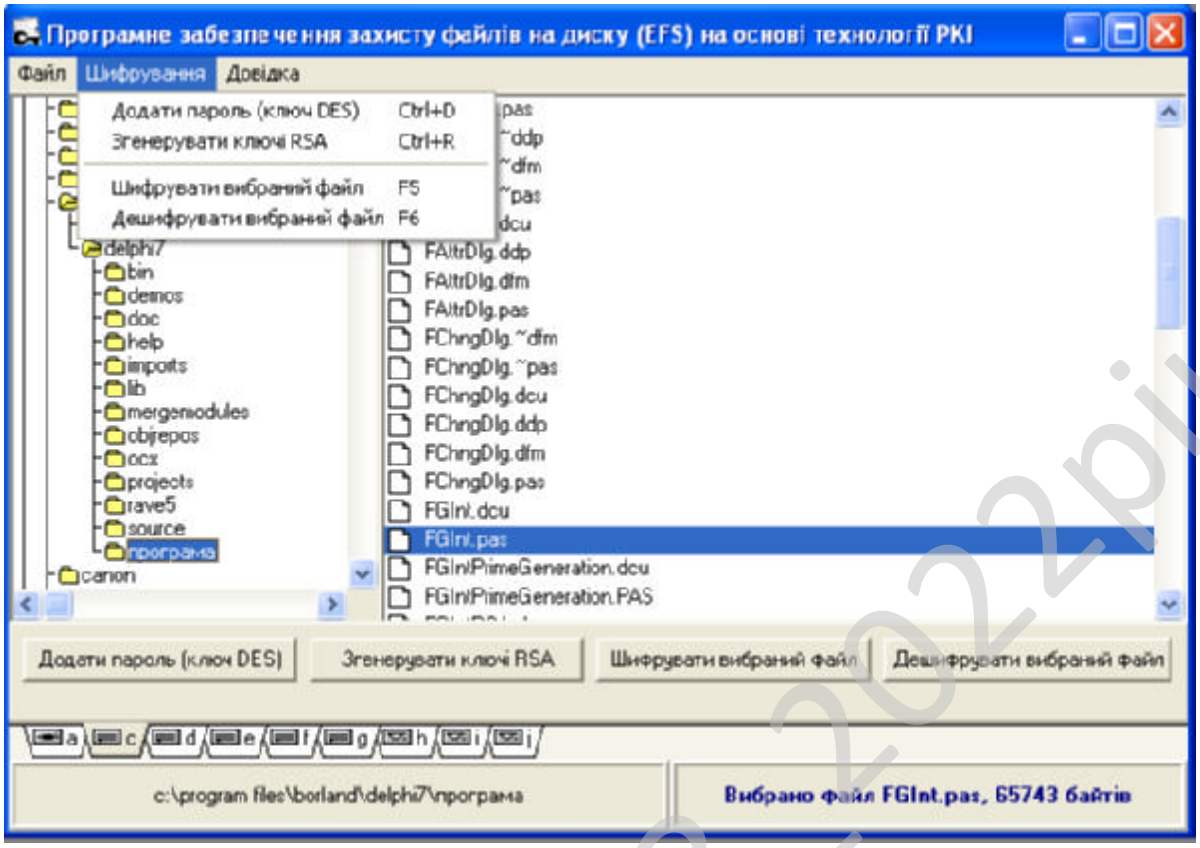


Рисунок 5.3 – Меню користувача "Шифрування"

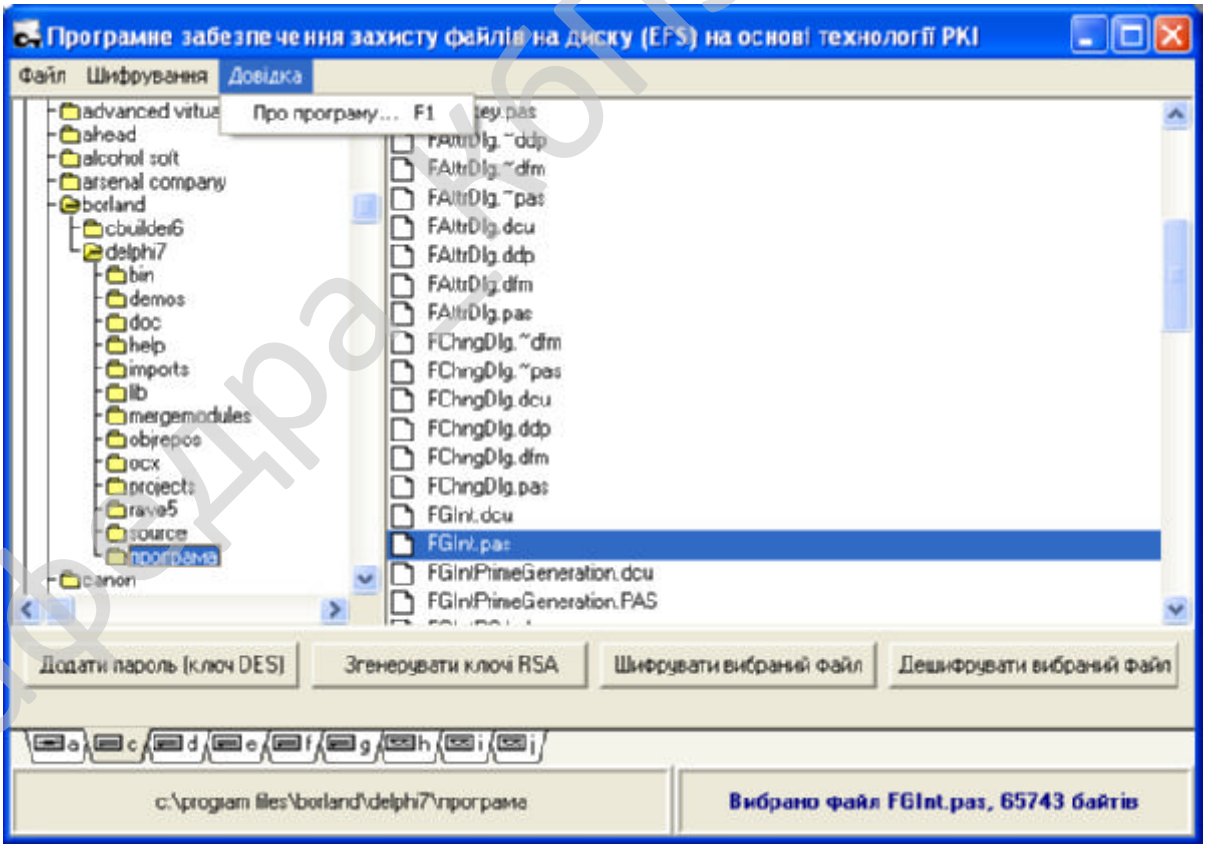


Рисунок 5.3 – Меню користувача "Довідка"

Як видно з рисунків програма дозволяє відкривати, переміщати, копіювати, видаляти, перейменовувати файли та переглядати їхні параметри. На рисунку 5.4 зображене вікно перегляду та зміни властивостей обраного файлу.

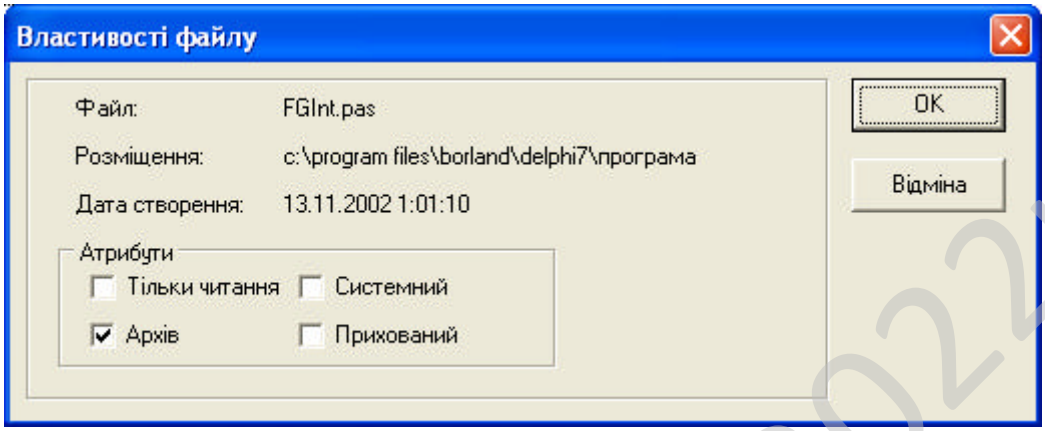


Рисунок 5.4 – Властивості файлу

Для здійснення шифрування файлів, спочатку необхідно створити секретний ключ DES та відкритий і закритий ключі RSA.

На рисунку 5.5 зображене вікно створення ключа DES. Для його створення користувач повинен ввести будь-яку послідовність символів, з точки зору безпеки бажано, щоб ця послідовність була не коротше 6 символів.

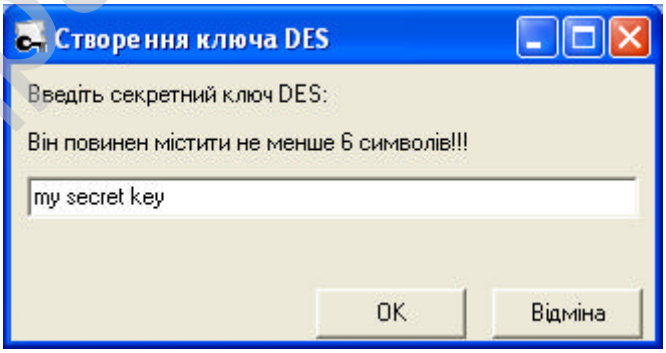


Рисунок 5.5 – Створення ключа DES

На рисунку 5.6 зображене вікно створення ключів RSA.

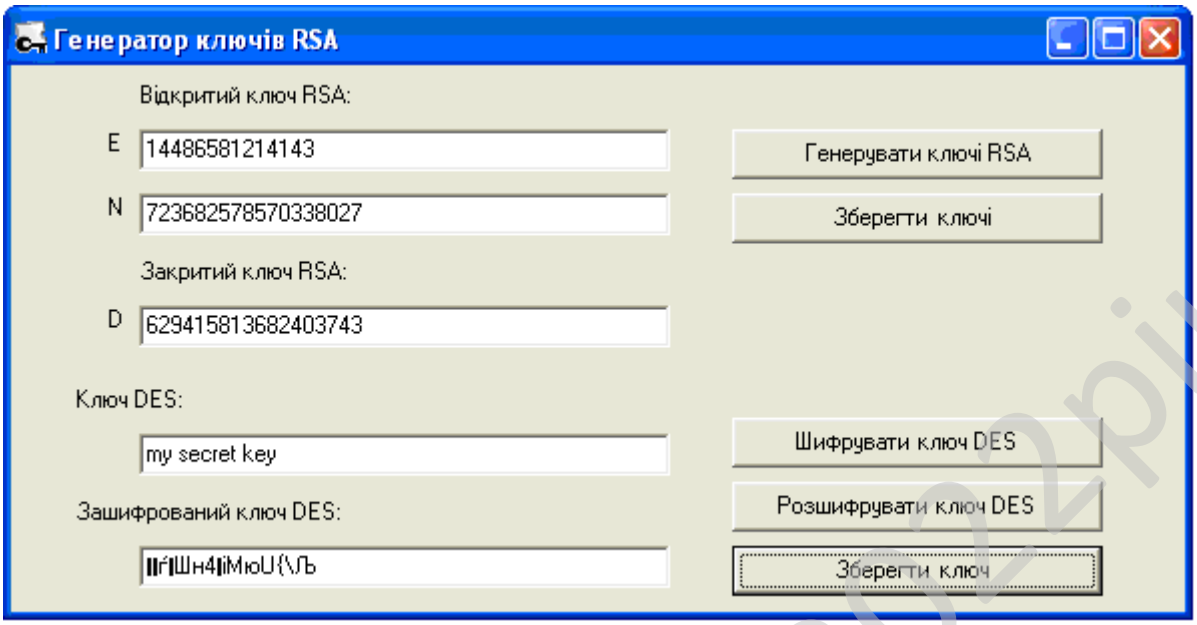


Рисунок 5.6 – Генерація ключів RSA

Спочатку користувач повинен натиснути кнопку "Генерувати ключі RSA", після чого програма випадковим чином згенерує ключі. Потім за допомогою кнопки "Зберегти ключі" слід записати їх у файли, окремо відкритий ключ, окремо закритий.

Далі за допомогою кнопки "Кодувати ключ DES" користувач кодує свій секретний ключ та записує у файл, використовуючи кнопку "Зберегти ключ".

Для здійснення шифрування файлу слід виділити мишею необхідний файл у списку та натиснути кнопку "Шифрувати". Після шифрування на екрані з'явиться повідомлення зображене на рисунку 5.7.

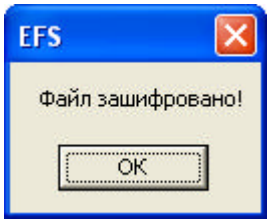


Рисунок 5.7 – Повідомлення після шифрування файлу

Для дешифрування файлу слід виділити його у списку файлів та натиснути кнопку "Дешифрувати". Після дешифрування на екрані з'явиться повідомлення зображене на рисунку 5.8.

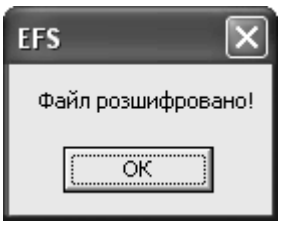


Рисунок 5.8 – Повідомлення після дешифрування файлу

Коротку довідку про розроблену програму можна переглянути натиснувши кнопку "Про програму…", після чого з'явиться вікно зображене на рисунку 5.9.

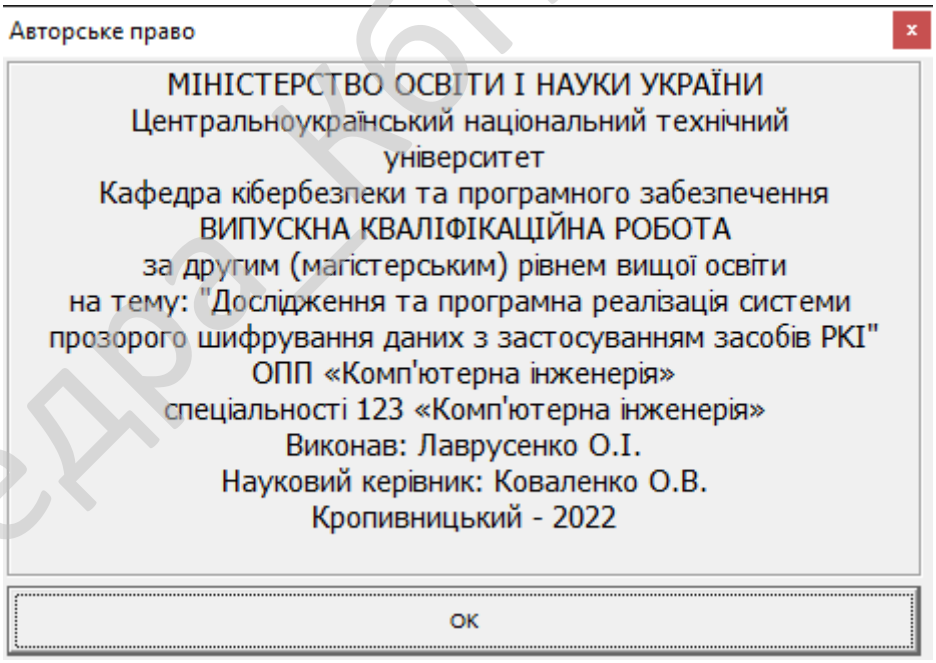


Рисунок 5.9 – Довідка

# 6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи прозорого шифрування даних з застосуванням засобів PKI.

*Метою розробки* є дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

*Об'єктом дослідження* є процес прозорого шифрування даних з застосуванням засобів PKI.

*Предметом дослідження* є методи прозорого шифрування даних з застосуванням засобів PKI.

*Методи дослідження* базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод прозорого шифрування даних з застосуванням засобів PKI.

– Розроблено вітчизняний продукт прозорого шифрування даних з застосуванням засобів PKI, який має більш широкі можливості, на відміну від існуючих аналогів.

# 7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

## 7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі було проведено дослідження та виконана програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

| Показники | Позначення | Характеристика або величина |
|---|---|---|
| 1 | 2 | 3 |
| 1. Кількість розроблених програм період, шт. | N | 1 |
| 2. Кількість екземплярів програм, шт. | Ne | 260 |
| 3. Запланований термін розробки, днів | Fpq | 48 (2 місяць) |
| 4. Група задачі підсистеми управління (1-6) | – | 1 |
| 5. Ступінь новизни задачі (А, Б, В, Г) | – | В |
| 6. Складність алгоритму (1, 2, 3) | – | 2 |

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 7. Кількість макетів вхідної інформації | – | 8 |
| 8. Кількість форм вихідної інформації. | – | 6 |
| 9. Мова програмування (1-6) | – | 2 |
| 10. Попередній досвід (1-6) | – | 3 |
| 11. Гнучкість проекту ПП (1-6) | – | 3 |
| 12. Детальність проекту ПП (1-6) | – | 1 |
| 13. Рівень спрацьованості колективу (1-6) | – | 2 |
| 14. Ступінь вимірності процесів (1-6) | – | 3 |
| 15. Необхідна надійність програмного забезпечення (1-6) | – | 3 |
| 16. Розмір бази даних (порівняно з розміром програми) (1-6) | – | 4 |
| 17. Складність кінцевого програмного продукту (1-6) | – | 5 |
| 18. Необхідний рівень забезпечення повторного використання (1-6) | – | 2 |
| 19. Документованість відповідно до планованого життєвого циклу (1-6) | – | 3 |
| 20. Вимоги до швидкодії ПП (1-6) | – | 3 |
| 21. Обмеження на розміри основного сховища даних (1-6) | – | 2 |
| 22. Різноманітність використовуваних обчислювальних платформ (1-6) | – | 4 |
| 23. Професійний рівень аналітиків (1-6) | – | 3 |
| 24. Професійний рівень програмістів (1-6) | – | 4 |
| 25. Постійність складу команди розробників (1-6) | – | 2 |
| 26. Досвід розробки додатків (1-6) | – | 1 |
| 27. Досвід роботи з обчислювальною платформою (1-6) | – | 2 |

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 28. Досвід роботи з мовою і інструментами середовища розробки (1-6) | – | 2 |
| 29. Досвід роботи з програмними інструментами розробки (1-6) | – | 3 |
| 30. Розробка ПЗ для декількох серверів одночасно (1-6) | – | 3 |
| 31. Вимоги до дотримання встановленого графіка робіт (1-6) | – | 2 |
| 32. Вартість ПЗ у розробника (НМА), грн. | – | 260000 |
| 33. Норматив додаткової зарплати, % : | $Н_д$ | 10 |
| 34. Норматив відрахувань у соціальні фонди, % | Нс | 22 |
| 35. Норматив загальногосподарських витрат, % | Нг | 15 |
| 36. Норматив витрат на освоєння нових мов програмування, % | Нп | 15 |
| 37. Рівень рентабельності програмної продукції, % | Ре | 40 |
| 38. Ставка податку на додану вартість, % | Ндв | 20 |

### 7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B, \qquad (7.1)$$

де: $A$ – коефіцієнт Боема, $A$ = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+4{,}22+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}2^{1{,}027} = 5{,}5 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 5{,}5 \cdot (1 \cdot 1{,}09 \cdot 1{,}30 \cdot 0{,}91 \cdot 1 \cdot 1 \cdot 1 \cdot 1{,}15 \cdot 1 \cdot 0{,}87 \cdot 1{,}10 \cdot$$

$$\cdot 1{,}22 \cdot 1{,}12 \cdot 1{,}10 \cdot 1 \cdot 1 \cdot 1{,}10) = 12{,}9 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

*S* – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 65 = 122 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

| Стадії розробки | Трудомісткість за типовими нормами та розрахунками | |
|---|---|---|
| | Величина, люд/дні | Підстава |
| Технічне завдання | 9 | Д5 |
| Ескізний проект | 10 | Д6 |
| Технічний проект | 15 | Д7 |
| Робочий проект | 122 | Ф 7.1-7.4 |
| Впровадження | 15 | Д13 |
| Всього | 171 | – |

### 7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \qquad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{171 \cdot 1}{48\text{-}3} = 3,8 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

| Найменування обладнання | Профілактичне обслуговування | | | |
|---|---|---|---|---|
| | Кількість хв. на один. обл. | Кількість облад-нання | Затрати часу в хв. | Затрати часу в год. |
| Системний блок ПК | 90 | 7 | 630 | 10,5 |
| Монітор | 60 | 7 | 420 | 7 |
| Клавіатура | 30 | 7 | 210 | 3,5 |
| Маніпулятор «мишка» | 30 | 7 | 210 | 3,5 |
| Принтер матричний | 60 | 0 | 0 | 0,0 |
| Принтер лазерний | 120 | 1 | 120 | 2 |
| Принтер струминний | 60 | 1 | 60 | 1 |
| Сканер | 20 | 1 | 20 | 0,33 |
| Концентратор-маршрутизатор | 30 | 2 | 60 | 1 |
| Кабельні господарства ЛОМ на 1 м.п. | 2,5 | 200 | 500 | 8,33 |
| Копіювальний апарат | 140 | 1 | 140 | 2,33 |
| Усього за рік: | | | $З_ч$ | 39,49 |

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1{,}2}, \tag{7.6}$$

$$Ф_{др}^{с} = \frac{40 \cdot 2}{1{,}2} = 67 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \tag{7.7}$$

$$Ч_{ел} = 67/(48 \cdot 8) = 0{,}2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Адміністратор загальної мережі, аналітик | Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi | 2 | 0,5 |
| | Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS) | 0,5 | |
| | Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ | 0,5 | |
| | Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет | 1 | |
| Всього | | 4 | |

Продовження таблиці 7.4

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Продакт-менеджер | Презентації нової продукції, пошук каналів збуту | 2 | 0,5 |
| | Підтримка постійних клієнтів | 1 | |
| | Оформлення договорів, ведення тендерів | 0,5 | |
| | Контроль взаєморозрахунків з постачальниками | 0,5 | |
| Всього | | 4 | |
| Дизайнер WEB | Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію | 2 | 0,5 |
| | Створення графічних і стилістичних елементів сайту | 1 | |
| | Оформлення банерів і промо-сторінок | 0,5 | |
| | Розміщення графіки і контенту на Інтернет сторінках | 0,5 | |
| Всього | | 4 | |
| Інженер верстальник | Розробка та верстка макетів рекламної продукції та технічної документації | 1 | 0,25 |
| | Верстка друкованих видань | 0,5 | |
| | Додрукова підготовка макетів | 0,25 | |
| | Розміщення графіки і контенту на Інтернет сторінках | 0,25 | |
| Всього | | 2 | |

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

| Посада | Кількість ставок | Середньомісячний оклад, грн. | Всього за період розробки, грн. |
|---|---|---|---|
| Керівник (ІТ-менеджер) | 1 | 17706 | 35412 |
| Продакт-менеджер | 0,5 | 15000 | 15000 |
| Інженер-програміст | 3,8 | 17500 | 133000 |
| Інженер-електронщик | 0,2 | 15000 | 6000 |
| Інженер-системотехнік | 0,25 | 15000 | 7500 |
| Адміністратор мережі | 0,5 | 15000 | 15000 |
| Системний програміст | 0,25 | 15000 | 7500 |
| Дизайнер WEB | 0,5 | 16000 | 16000 |
| Інженер-верстальник | 0,25 | 15000 | 7500 |
| Бухгалтер-економіст | 0,5 | 16000 | 16000 |
| Всього за період розробки | $R_{сп} = 7,75$ | - | $Ф_{роб} = 258912$ |

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{258912}{7,75 \cdot 48} = 696 \; грн.$$

## 7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 12 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м$^2$. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 12 \cdot 8 \cdot 20000 = 1920000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 192000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 12 \cdot 3500 = 42000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Комп'ютерторг за 25.10.22 – джерело

http://computorg.ua/ru/price.html

Таблиця 7.6 – Специфікація

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Персональний комп’ютер | | 11186 |
| Системний блок | Lenovo ThinkCentre M73 USFF | 6490 |
| Процесор | Intel Core i5-4570T, LGA1150, (2 (4) ядра п 2.9 - 3.6 GHz); Cache Memory 4 MB | |
| Системна плата | Lenovo ThinkCentre M73, Intel H81 Express 2x USB 3.0, 3x USB 2.0, 2x Audio Ports, 1 LAN (RJ-45), 1x VGA, 1x DP, UDIMM, PC3 12800 1600 МГц DDR3, два 240-контактни роз'єму DIMM, Gigabit Ethernet, Realte RTL8111GN, бездротова мережа 11b/g/r PCIe Half Mini Card, Intel Centrino® Wireless-N 105 | |
| Відеокарта | Integrated Intel HD Graphics 4600, частот графічної системи 200 MHz - 1.15 GHz | |
| Жорсткий диск | 240 GB SSD | |
| Оперативна пам’ять | 8 GB DDR3 | |
| DVD-привод | Не комплектується | |
| Корпус | ThinkCentre M73 USFF | |
| Кулер | – | – |
| Кардрідер внутрішній | USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black | |

Продовження таблиці 7.6

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| інше | Клавіатура, мишка | Подару-нок |
| Монітор | 22" TFT, ASUS VW223D ( 5ms, 300/3000: 1 170/160, D-SUB, Wide) | 3200 |
| Принтер лазерний | Canon i-SENSYS LBP6030W | 2700 |
| Принтер струминний | Epson Stylus Photo P50 (C11CA45341) + USB cable | 5500 |
| Сканер | Epson Perfection V37 Photo | 2970 |
| Копіювальний апарат | Canon i-SENSYS MF217W with Wi-Fi | 5965 |
| Пристрій безперебійного живлення | UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS) | 1496 |

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

| Найменування обчислювальної техніки | Кількість, шт. | Ціна за одиницю, грн. | Витрати на транспортування, монтаж та випробовування. | Загальна вартість, грн. |
|---|---|---|---|---|
| Персональні комп’ютери | 12 | 11186 | 13423,2 | 147655,2 |
| Принтер лаз. | 2 | 2700 | 540 | 5940 |
| Принтер струм. | 1 | 5500 | 550 | 6050 |

Продовження таблиці 7.7

| Найменування обчислювальної техніки | Кількість, шт. | Ціна за одиницю, грн. | Витрати на транспортування, монтаж та випробовування. | Загальна вартість, грн. |
|---|---|---|---|---|
| Сканери | 1 | 2970 | 297 | 3267 |
| Копіюв. апарат | 1 | 5965 | 596,5 | 6561,5 |
| Всього | – | – | – | 169473,7 |

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

| Групи та види основних фондів | Балансова вартість, грн. | Амортизація | |
|---|---|---|---|
| | | Норма, % | Відрахування, грн. |
| 1 | 2 | 3 | 4 |
| Група 3 | | | |
| 1. Будівлі | 1920000 | - | - |
| 2. Передавальні пристрої | 192000 | - | - |
| Всього по групі | 2112000 | 5 | 105600 |
| Група 4 | | | |
| 3. Обчислювальна техніка | 169474 | - | - |
| Всього по групі | 169474 | 50 | 84737 |
| Нематеріальні активи | | | |
| 4. Нематеріальні активи | 260000 | 10 | 26000 |

Продовження таблиці 7.8

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Група 5, 6 | | | |
| 5. Вимірювальні пристрої | 5190 | 25 | 1297,5 |
| 6. Транспортні засоби | 72500 | 20 | 14500 |
| 7. Господарський інвентар | 42000 | 25 | 10500 |
| Всього по групі | 119690 | - | 26297,5 |
| Разом | $К_р = 2661164$ | | $А_р = 242634,5$ |

Примітка: вартість автомобіля ЗАЗ Sens 2009 взята по даним з автосалону авто-РІА, джерело https://auto.ria.com/uk/auto_zaz_sens_33451541.html, складає 72500 грн.

### 7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 696 \cdot 171/260 = 458 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 458 \cdot 10 \cdot 0{,}01 = 46 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $H_c$ – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(458+46) = 111 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $H_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot H_г \cdot 0{,}01, \quad (7.14)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 458 \cdot 15 \cdot 0{,}01 = 69 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1 пачку паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=240 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 240 \cdot 1 = 240 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 100):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 33,93 грн./шт.

$$З_{М2} = 100 \cdot 33{,}93 = 3393 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (240 + 3393 + 1702)/260 = 21 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 458 \cdot 15 \cdot 0{,}01 = 69 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 260 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 242634 \cdot 2/(260 \cdot 12) = 156 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$С_п = 458+46+111+69+21+69+156 = 930 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \quad (7.22)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 930 = 379 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

| Найменування статей витрат | Позначення | Величина, грн |
|---|---|---|
| 1 | 2 | 3 |
| 1. Основна зарплата виконавців | $З_о$ | 458 |
| 2. Додаткова зарплата виконавців | $З_д$ | 46 |
| 3. Відрахування на соціальні потреби | $С_{оц}$ | 111 |
| 4. Загальногосподарські витрати | $Г_{осп}$ | 69 |
| 5. Витрати на матеріали | $З_М$ | 21 |
| 6. Освоєння нових операційних систем, мов програмування | $О_п$ | 69 |
| 7. Амортизація основних фондів | $А_м$ | 156 |
| 8. Повна собівартість програмного забезпечення | $С_п$ | 930 |
| 9. Плановий прибуток | $П_р$ | 379 |
| 10.Ціна підприємства $Ц_п = С_п + П_р$ | $Ц_п$ | 1309 |
| 11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{ов} \cdot Ц_п$ | $ПДВ$ | 261,8 |
| 12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$ | $Ц$ | 1570,8 |

## 7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та

пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

| Найменування капітальних вкладень | Сума за варіантами, грн. | |
|---|---|---|
| | Базовий | Новий |
| Вартість програмної продукції | – | 1571 |
| Всього капітальних витрат | – | 1571 |

### 7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

| Найменування статей витрат | Позначення | Сума витрат за варіантами, грн. | |
|---|---|---|---|
| | | Базовий | Новий |
| 1. Витрати на обслуговування системи | $З_р$ | 24156 | 13527 |
| 2. Витрати на електроенергію | $З_{ел}$ | 248 | 139 |
| 3. Витрати на амортизацію | $З_{ам}$ | 0 | 786 |
| Всього витрат за рік | $І$ | 24404 | 14452 |

Витрати на обслуговування системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування системи за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 250 годин на рік до 140 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 250 \cdot 72 \cdot 1{,}1 \cdot 1{,}22 = 24156\ грн,$$

до:

$$З_{р\ нов} = 140 \cdot 72 \cdot 1{,}1 \cdot 1{,}22 = 13527\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}45 \cdot 250 \cdot 2{,}2 = 248 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}45 \cdot 140 \cdot 2{,}2 = 139 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

| Групи основних фондів | Норма амортизації % | Балансова вартість, грн., за варіантами | | Сума відрахувань, грн за варіантами | |
|---|---|---|---|---|---|
| | | Базовий | Новий | Базовий | Новий |
| Програмна продукція | 50 | – | 1571 | – | 785,5 |
| Всього відрахувань | - | – | 1571 | – | 785,5 |

## 7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (1309 - 930) \cdot 260 - (0,05 \cdot 2112000 + 0,5 \cdot 169474 +$$
$$+0,25 \cdot 47190 + 0,2 \cdot 72500 + 0,1 \cdot 260000) \cdot 2/12 = 58101 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_p^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{549164}{(1309 - 930) \cdot 260 \cdot 12 / 2} = 0,9 \text{ року}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (24404\text{-}14452)\text{-}0,5 \cdot 1571 = 9167 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{1571}{24404 - 14452} = 0,16 \text{ року}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

| Найменування показників | Одиниця виміру | Величина |
|---|---|---|
| 1. Кількість екземплярів програми | Прим. | 260 |
| 2. Повна собівартість розробленої програми | Грн. | 930 |
| 3. Ціна розробленої програми | Грн. | 1309 |
| 4. Плановий прибуток від реалізації розробленої програми | Грн. | 379 |
| 5. Рентабельність програмної продукції | % | 40 |
| 6. Об’єм додаткових капітальних вкладень у виробника програмної продукції | Грн. | 2661164 |
| 7. Загальний прибуток від реалізації програмної продукції | Грн. | 98540 |
| 8. Величина економічного ефекту при виготовлені програмної продукції | Грн. | 58101 |
| 9. Період окупності додаткових капітальних вкладень у виробника програмної продукції | Років | 0,9 |
| 10.Об’єм додаткових капітальних вкладень у споживача програмної продукції | Грн. | 1571 |
| 11.Величина економічного ефекту у користувача програмної продукції | Грн. | 9167 |
| 12.Період окупності додаткових капітальних вкладень у користувача програмної продукції | Років | 0,16 |

## 7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

# 8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

## 8.1 Вступ

В охорону праці включають санітарно-гігієнічні, лікувально-профілактичні та організаційно-технічні системи правових і соціально-економічних заходів.

В кожній ІТ компанії є трудові відносини з працівниками. Згідно закону України “Про охорону праці” [3] кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

– Інструкцій з охорони праці по кожній професії і загальні.

– Положення про охорону праці.

– Накази з охорони праці.

– Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час

роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп’ютера на організм людини визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

## 8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

| Найменування | Значення, м |
|---|---|
| Ширина | 7 |
| Довжина | 8 |
| Висота | 3,3 |

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

| Геометрична характеристика | Одиниця виміру | Нормативне значення* | Фактичне значення |
|---|---|---|---|
| Площа, S | $м^2$ | не менше 6.0 | 6,2 |
| Об'єм, V | $м^3$ | не менше 20.0 | 20,5 |

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють 9 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об’єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-

обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри). Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами. У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

| Пора року | Оптимальні для Іа | | | Фактичні | | |
|---|---|---|---|---|---|---|
| | Температура, °С | Воло-гість,% | Швидкість повітря, м/с | Температура,°С | Воло-гість% | Швидкість повітря, м/с |
| Холодна | 22-24 | 40-60 | 0,1 | 22-24 | 40-55 | 0,1 |
| Тепла | 23-25 | 50-70 | 0,1 | 24-25 | 50-65 | 0,9 |

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер Xerox WorkCentre 3025BI (3025VBI), електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України "Природне і штучне освітлення" – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1], Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

### 8.3 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значенн;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга) [9].

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

### 8.4 Розрахункова частина

Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).


Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 7 м, довжина – 8 м, висота – 3,3 м.

У зазначеному приміщенні працює 9 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=ESKZ/n,$$

де:

*F* – світловий потік, що розраховується, Лм;

*E* – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

*S* – площа освітлюваного приміщення.

*K* – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

*Z* – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

*n* – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$)**, значення коефіцієнтів дорівнюють

$$\rho_{стін} = 50\% \text{ і } \rho_{стелі} = 50\%.$$

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де:

*S* – площа приміщення, *S* = 56 $м^2$*;*

*h* – розрахункова висота підвісу, *h* = 3,2 *м. (співпадає з в*исотою стелі, т.я. лампи освітлення закріплюються на стелі)*;*

*A* – ширина приміщення, *A = 7 м;*

*B* – довжина приміщення, *B =* 8 *м.*

Підставимо всі значення у формулу та визначимо індекса приміщення:

$$i=1,1.$$

Знаючи індекс приміщення, за знаходимо *n = 0,46 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним*

*типом лампам) [8].* Підставимо всі значення у формулу, визначемо світловий потік: *F=60260 Лм.*

Для розрахунку дудемо використовувати стельові *світлодіодні панелі Призма-72 6400К*, світловий потік яких $F_л$ = *7200* Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де:

*F* – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення: *N= 60260 / 7200=8,3 шт.*

Приймаємо необхідну кількість *світлодіодних світильників* 9 шт.

### 8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

# 9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи прозорого шифрування даних з застосуванням засобів PKI.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів прозорого шифрування даних з застосуванням засобів PKI.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем прозорого шифрування даних з застосуванням засобів PKI.

– Досліджена система прозорого шифрування даних з застосуванням засобів PKI.

– На основі отриманих результатів досліджень створена програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання прозорого шифрування даних з застосуванням засобів PKI.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10.4 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Madryga.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 9167 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,16 роки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Лаврусенко О.І. Дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI // Збірник праць молодих науковців ЦНТУ. – Вип. 13. – Кропивницький: ЦНТУ, 2022.

2. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

3. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011.

4. Смирнов С.А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А.К. Дидык, С.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

5. Смирнов С. А. Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Абу Таам Гани, А. А. Смирнов, А. В. Коваленко, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2014. – Вип. 9(125). – 105-110.

6. Смирнов С. А. Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов //

Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2014. – Вип. 4 (41). – С. 48-52.

7. Смирнов С. А. Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. –Х.: ХУПС, 2014. – № 4(17). – С. 90-95.

8. Смирнов С. А. Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, Н. С. Якименко, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2015. –Вип. 1(126). – С. 150-153.

9. Smirnov S.A. Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

10. Смирнов С. А. Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты даннях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2015. – № 3(43). – С. 100-107.

11. Смирнов С. А. Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України: наук. журн. –Х.: ХУПС, 2015. – № 3(20). – С. 134-141.

12. Смирнов С. А. Комплекс gert-моделей технологии облачной антивирусной защиты телекоммуникационной системы / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Безпека інформації: наук. – практ. журн. – К.: НАУ, 2015. – Т. 21, № 3. – С. 251-262.

13. Смирнов С. А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, А. К. Дидык, С. А. Смирнов // Системи озброєння і військова техніка: наук. журн. – Х.: ХУПС, 2016. – № 2 (46). – С. 146-149.

14. Смирнов С. А. Модели системы нейросетевых экспертов безопасной маршрутизации в облачных антивирусных системах / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Системи обробки інформації: зб. наук. праць. – Х.: ХУПС, 2016. – Вип. 3 (140). – С. 36-39.

15. Смирнов С. А. Метод безопасной маршрутизации на базовом множестве путей передачи метаданных в облачные антивирусные системы / В. Л. Бурячок, С. А. Смирнов // Системи управління, навігації та зв'язку. – Полтава, 2016. – Вип. 4(40). – С. 57-62.

16. Смирнов С. А. Способ контроля линий связи телекоммуникационной системы облачного антивируса / А. А. Смирнов, А. К. Дидык, А. Н. Дреев, С. А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2016. – № 2 (47). – С. 148-152.

17. Смирнов С. А. Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / В. Л. Бурячок, Мохамад Абу Таам Гани, С. А. Смирнов // Інформаційні технології та комп'ютерна інженерія: зб. тез доп. наук.-практ. конф., м. Кіровоград, 4 грудня 2014 р. – Кіровоград: КНТУ, 2014. – С. 168.

18. Смирнов С. А. Исследование математических моделей технологии распространения компьютерных виру сов / А. А. Смирнов, Мохамад Абу Таам Гани, С. А. Смирнов // Актуальні питання забезпечення кібернетичної безпеки та захисту інформації: зб. наук. праць міжнар. наук.-практ. конф., м. Київ, 25-28 лютого 2015 р. – К.: Європейський університет, 2015. – С. 90-91.

19. Смирнов С. А. Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Всеукраїнська науково-практична конференція

«Інформаційна безпека держави, суспільства та особистості», м. Кіровоград, 16 квітня 2015 р.: зб. тез доп. – Кіровоград: КНТУ, 2015. – С. 50-52.

20. Смирнов С. А. Разработка метода управления доступом в интеллектуальных узлах коммутации / А. А. Смирнов, Мохамад Абу Таам Гани, С. А. Смирнов // Проблеми і перспективи розвитку ІТ-індустрії: зб. тез VII міжнар. наук.-практ. конф., м. Харків, 17-18 квітня 2015 р. – Х.: ХНЕУ, 2015. – С. 14.

21. Смирнов С.А. Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Абу Таам Гани, С.А. Смирнов // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

22. Смирнов С. А. технология передачи сигнатур в облачные антивирусные системы для обеспечения защищенности телекоммуникационных сетей / А. А. Смирнов, С. А. Смирнов // Збірник тез V міжнародної науково-технічної конференції «ITSEC», Київ, 19-22 травня 2015 р. – К.: НАУ 2015. – С. 12-13.

23. Смирнов С. А. Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Інформаційна та економічна безпека (INFECO-2015): зб. тез II Міжнар. наук.-практ. Інтернет-конф., м. Харків, 21-22 травня 2015 р. – Х.: ХІБС УБС НБУ, 2015. – С. 20-24.

24. Смирнов С. А. Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Сборник тезисов XI международной конференции «Стратегия качества в промышленности и образовании», г. Варна, Болгария, 01-06 июня 2015 г. – Варна: ТУВ, 2015. – С. 488-491.

25. Смирнов С. А. Метод управления доступом к облачным телекоммуни-кационным ресурсам для обеспечения защиты даннях / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Комп'ютерні технології та інформаційна безпека: зб. тез доп. міжнар. наук.-практ. конф., м. Кіровоград, 2-3 липня 2015 р. – Кіровоград: КНТУ, 2015. – С. 4-5.

26. Смирнов С. А. Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Абу Таам Гани, А. А. Смирнов, С. А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації» (м. Затока, 7-9 вересня 2015 р.). – Одеса: ОНАЗ, 2015. – С. 90-94.

27. Смирнов С. А. Разработка комплекса gert-моделей технологии облачной антивирусной защиты телекоммуникационной системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційні технології та взаємодії» (IT & I): зб. тез II міжнар. наук.-практ. конф., м. Київ, 3-5 листопада 2015 р. – К.: КНУ ім. Тараса Шевченка, 2015. – С. 65-67.

28. Смирнов С. А. Разработка моделей телекоммуникационной системы формирования и обработки метаданных в облачных антивирусных системах / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Информационные и телекоммуникационные технологии: образование, наука, практика: сб. тезисов II междунар. научно-практ. конф., г. Алматы, Казахстан, 3-4 декабря 2015 г. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – С. 309-313.

29. Смирнов С. А. gert-модели технологии облачной антивирусной защиты / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Безпека українського суспільства в концепції вступу в постіндустріальне суспільство ЄС: зб. тез Круглого столу, м. Київ, 16 грудня 2015 р. – К.: Європейський університет, 2015. – С.41-43.

30. Смирнов С. А. Алгоритмы формирования множества маршрутов передачи метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Актуальні питання забезпечення

кібернетичної безпеки та захисту інформації: зб. наук. праць II Міжнар. наук.-практ. конф., м. Київ, 24-27 лютого 2016 р. – К.: Європейський університет, 2016. – С. 140-142.

31. Смирнов С. А. Разработка и реализация метода безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Securitea informationala 2015-2016: Conferenta internationala (editia a XII-a), Chisinau, Moldova, 3 martie 2016. – Chisinau: ADSEM, 2016. – С. 90-96.

32. Смирнов С. А. Алгоритм формирования базового множества маршрутов передачи метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформатика та системні науки (ІСН-2016): зб. тез VII всеукр. наук.-практ. конф., м. Полтава, 10-12 березня 2016 р. – Полтава: ПУЕТ, 2016. – С. 261-263.

33. Смирнов С. А. Система обработки и формирования начального состояния маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: зб. тез наук.-практ. конф., м. Київ, 10-11 березня 2016 р. – К.: КНУ ім. Тараса Шевченка, 2016. – С. 81-82.

34. Смирнов С. А. Алгоритм безопасной маршрутизации на базовом множестве путей передачи метаданных в программный сервер облачной антивирусной системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційна безпека та комп'ютерні технології (IS&CT): зб. тез міжнар. наук.-практ. конф., м. Кіровоград, 24-25 березня 2016 р. – Кіровоград: КНТУ, 2016. – С. 73.

35. Смирнов С. А. Исследование способа контроля линий связи телекоммуникационной системы для облачных антивирусов / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового

забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016), м. Харків, 30 березня – 1 квітня 2016 р. – Х.: НТУ «ХПІ», 2016. – С. 14.

36. Смирнов С. А. Разработка способа контроля линий связи телекоммуникационной системы для облачных антивирусов / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Матеріали XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування» (м. Кіровоград, 15-16 квітня 2016 р.). –Кіровоград: КНТУ, 2016. – С. 182-186.

37. Смирнов С. А. Разработка и исследование способа контроля линий связи телекоммуникационных сетей для облачных антивирусных систем / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Проблеми і перспективи розвитку ІТ-індустрії: VIII міжнар. наук.-практ. конф., м. Харків, 28-29 квітня 2016 р.: зб. тез. – Х.: ХНЕУ, 2016. – С. 48.

38. Смирнов С. А. Модель системы нейросетевых экспертов безопасной маршрутизации для облачных антивирусных систем / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Інформаційна та економічна безпека (INFECO-2016): зб. тез ІІІ міжнар. наук.-практ. конф., м. Харків, 28-30 кві. 2016 р. – Х.: ХННІ ДВНЗ «УБС», 2016. – С. 178-182.

39. Смирнов С. А. Метод безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. К. Дидык // Сборник тезисов XII международной конференции «Стратегия качества в промышленности и образовании» (г. Варна, Болгария, 30 мая – 02 июня 2016 г.). – Варна: ТУВ, 2016. – С. 581-585.

40. Смирнов С. А. Оценка эффективности метода безопасной маршрутизации метаданных в облачные антивирусные системы / А. А. Смирнов, С. А. Смирнов, А. С. Коваленко // РадіоЕлектроніка та ІнфоКомунікації: зб. тез першої наук. – техн. конф., м. Київ, 11-16 вересня 2016 р. – К.: НТУУ «КПІ», 2016. – С. 17.

41. Современные телекоммуникации. Технологии и экономика / [В.Л. Банкет, О.В. Бондаренко, П.П. Воробьенко и др.]; под ред. С.А. Довгого. – М.: Эко-Трендз, 2003. – 320 с.

42. Столлингс В. Современные компьютерные сети / Вильям Столлингс. –СПб.: Питер, 2003. – 778 с.

43. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. – 848 с.

44. Телекоммуникационные системы и сети: учебное пособие. В 3 томах / [В.В. Величко, Е.А. Субботин, В.П. Шувалов, А.Ф. Ярославцев]; под ред. В.П. Шувалова. – М.: Горячая линия-Телеком, 2005, т. 3 – 592 с.

45. Хайкин С. Нейронные сети: полный курс / С. Хайкин. – М.:Вильямс, 2006. – 1103 с.

46. Шелухин О.И. Фрактальные процессы в телекоммуникациях: моногр. / О.И. Шелухин, А.М. Тенякшев, А.В. Осин – М.: Радиотехника, 2003. – 480 с.

47. Elwalid, D. Mitra, I. Saniee, and I. Widjaja. Routing and Protection in GMPLS Networks: From Shortest Paths to Optimized Designs // Journal of lightwave technology. – 2003. – №21(11), P. 2828-28-38.

48. A.B. Bagula, M. Botha, and A.E Krzesinski. Online Traffic Engineering: The Least Interference Optimization Algorithm // IEEE Communications Society – 2004, P. 1232-1236.

49. Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Evaluating the Impact of Stale Link State on Quality-of-Service Routing // IEEE/ACM Transactions on Networking. – 2001. – №9(2), P. 162-176.

50. Basabi Chakraborty. Simultaneous Search for Multiple Routes using Genetic Algorithm / IEEE lntemational Conference on Computational oltelligencs for Measurement System and Applications Boston. MA, USA, 14-16, July 2004, P. 77-80/

51. Barakat, E. Altman, and W. Dabbous. On TCP performance in a heterogeneous network: a survey // IEEE Communications Magazine. – 10/11. – №38(1). – P. 40 – 46.

52. Carpenter G., A., Grossberg S. Pattern Recognition by Self-Organizing Neural Networks, Cambridge, MA, MIT Press, 1991.

53. Cloud security, Deep Dive series, August 2011 [Электронный ресурс]. – Режим доступа к ресурсу: http://www.slideshare.net/kimrenejensen/cloud-security-deep-dive-2011#14375029197881&fbinitialized

54. Chris Loeser, Andre Brinkmann, Ulrich Ruckert. Distributed Path Selection (DPS) a Traffic Engineering Protocol for IP-Networks / Proceedings of the 37th Hawaii International Conference on System Sciences – 2004, P. 1-8.

55. Dai Boong Lee and Hwangjun Song. Dynamic Class Selecting Mechanism for Guaranteed Service with Minimum Cost over Relative Differentiated-Services Networks / IEEE International Conference on Multimedia and Expo (ICME) – 2004, P. 237-240.

56. Gang Cheng and Nirwan Ansari. A New Heuristics For Finding The Delay Constrained Least Cost Path // IEEE GLOBECOM – 2003, P. 3711-3715.

57. Gang Cheng, Li Zhu, and Nirwan Ansari. A New Deterministic Traffic Model for Core-Stateless Scheduling // IEEE Transactions on communications. – 2006. – № 4, P. 704-713.

58. Hui Ma, Dongfeng Wang, Farokh Bastani, I-Ling Yen, Kendra Cooper. A Model and Methodology for Composition QoS Analysis of Embedded Systems / Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05) – 2005, P. 1-10.

59. H. Chaskar M. Eder, S. Nag "Considerations from the Service Management Research Group (SMRG) on Quality of Service (QoS) in the IP Network" // RFC-3387. September 2002

60. ITU-T Recommendations [Электронний ресурс]. – Режим доступа к ресурсу: http://www.itu.int/ITU-T/recommendations/index.aspx?ser=Y

61. Ji Li and Kwan L. Yeung. A Two-Step Approach to Restorable Dynamic QoS Routing // IEEE Communications Society. – 2004, P. 1166-1170.

62. Jianbin Wei and Cheng-Zhong Xu. Feedback Control Approaches for Quality of Service Guarantees in Web Servers // NAFIPS 2005. Annual Meeting of the North American Fuzzy Information Processing Society – 2005. P. 700-705

63. Jiang Hu and Sachin S. Sapatnekar. A Timing-Constrained Simultaneous Global Routing Algorithm // IEEE Transactions on computer-aided design of integrated circuits and systems. – 2002. – №21(9), P. 1025-1036.

64. Jeffrey O. Kephart, Steve R. White, Directed-Graph Epidemiological Model of Computer Viruses // IEEE Symposium on Security and Privacy, 1991. – P.343. [Электронный ресурс]. – Режим доступа к ресурсу: http://www.research.ibm.com/antivirus/SciPapers/Kephart/VIRIEEE/ virieee.gopher.html

65. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

66. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

67. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

68. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

69. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград:

КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

70. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

71. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

72. Центр післядипломної освіти та підвищення кваліфікації. – Режим доступу до ресурсу: https://cpo.stu.cn.ua

73. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

Додаток А
(обов'язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи прозорого шифрування даних з застосуванням засобів PKI.

## 2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 19-13 від 17.08.2022 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI.

## 4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи прозорого шифрування даних з застосуванням засобів PKI;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

### 5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

### 5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

#### 5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

#### 5.8.2 Мова програмування

Середовище Delphi 10.4.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

## 7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

## 8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

## 9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 111 аркушів.

## 10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2022 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 19.12.2022 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Коваленко О.В.

***Дослідження та програмна реалізація системи прозорого шифрування даних з застосуванням засобів PKI***

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 59

Літера: РП

Кропивницький – 2022 року

**Основна програма**

**Файл filmanex.dpr основної програми**

```
program FilManEx;

uses
  Forms,
  FMXWin in 'FMXWin.pas' {FMForm},
  FmxUtils in 'Fmxutils.pas',
  FAttrDlg in 'FAttrDlg.pas' {FileAttrForm},
  FChngDlg in 'FChngDlg.pas' {ChangeDlg},
  about in 'about.pas' {Form1},
  FGInt in 'FGInt.pas',
  FGIntPrimeGeneration in 'FGIntPrimeGeneration.PAS',
  FGIntRSA in 'FGIntRSA.PAS',
  RSA_keys in 'RSA_keys.pas' {RSA_keys},
  des_key in 'des_key.pas' {DES};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TFMForm, FMForm);
  Application.CreateForm(TFileAttrForm, FileAttrForm);
  Application.CreateForm(TChangeDlg, ChangeDlg);
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TRSA_keys, RSA_keys);
  Application.CreateForm(TDES, DES);
  Application.Run;
end.
```

**Файл FMXWin.pas – вивід дерева файлів та каталогів, шифрування, дешифрування**

```
unit FMXWin;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, FileCtrl, Grids, Outline, DirOutln, Tabs, ExtCtrls, Menus, about,
RSA_keys, des_key;

type
  TFMForm = class(TForm)
    StatusBar: TPanel;
    DirectoryPanel: TPanel;
    FilePanel: TPanel;
    DriveTabSet: TTabSet;
    DirectoryOutline: TDirectoryOutline;
    FileList: TFileListBox;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Open1: TMenuItem;
    Move1: TMenuItem;
    Copy1: TMenuItem;
    Delete1: TMenuItem;
    Rename1: TMenuItem;
    Properties1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    Floppy: TImage;
    Fixed: TImage;
    Network: TImage;
    CDRom: TImage;
    RamDisk: TImage;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    RSA1: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    procedure Exit1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure DirectoryOutlineChange(Sender: TObject);
    procedure FileListChange(Sender: TObject);
    procedure DriveTabSetMeasureTab(Sender: TObject; Index: Integer;
      var TabWidth: Integer);
    procedure DriveTabSetDrawTab(Sender: TObject; TabCanvas: TCanvas;
      R: TRect; Index: Integer; Selected: Boolean);
    procedure File1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure Properties1Click(Sender: TObject);
    procedure FileChange(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure FileListMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure DirectoryOutlineDragOver(Sender, Source: TObject; X,
      Y: Integer; State: TDragState; var Accept: Boolean);
    procedure DirectoryOutlineDragDrop(Sender, Source: TObject; X,
      Y: Integer);
    procedure FileListEndDrag(Sender, Target: TObject; X, Y: Integer);
```

```
    procedure DriveTabSetChange(Sender: TObject; NewTab: Integer;
      var AllowChange: Boolean);
    procedure N4Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);

  private
    procedure ConfirmChange(const ACaption, FromFile, ToFile: string);
  public
    { Public declarations }
  end;

var
  FMForm: TFMForm;

implementation

uses FmxUtils, FAttrDlg, FChngDlg;

{$R *.dfm}

procedure TFMForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TFMForm.FormCreate(Sender: TObject);
var
  Drive: Char;
  AddedIndex: Integer;
begin
//Вибір диску
  for Drive := 'a' to 'z' do
  begin
    case GetDriveType(PChar(Drive + ':\')) of
      DRIVE_REMOVABLE:
        AddedIndex := DriveTabSet.Tabs.AddObject(Drive, Floppy.Picture.Graphic);
      DRIVE_FIXED:
        AddedIndex := DriveTabSet.Tabs.AddObject(Drive, Fixed.Picture.Graphic);
      DRIVE_CDROM:
        AddedIndex := DriveTabSet.Tabs.AddObject(Drive, CDRom.Picture.Graphic);
      DRIVE_RAMDISK:
        AddedIndex := DriveTabSet.Tabs.AddObject(Drive,
RamDisk.Picture.Graphic);
      DRIVE_REMOTE:
        AddedIndex := DriveTabSet.Tabs.AddObject(Drive,
Network.Picture.Graphic);
    else
      AddedIndex := 0;
    end;
    if UpCase(Drive) = FileList.Drive then
      DriveTabSet.TabIndex := AddedIndex;
  end;
end;

procedure TFMForm.DriveTabSetChange(Sender: TObject; NewTab: Integer;
  var AllowChange: Boolean);
begin
  if not (csDesigning in ComponentState) then
  begin
    AllowChange := True;
    try
      with DriveTabSet do
        DirectoryOutline.Drive := Tabs[NewTab][1];
    except
      on EInOutError do
      begin
```

```
        AllowChange := False;
        with DriveTabSet do
          DirectoryOutline.Drive := Tabs[TabIndex][1];
        raise;
      end;
    end;
  end;
end;

procedure TFMForm.DirectoryOutlineChange(Sender: TObject);
begin
  FileList.Directory := DirectoryOutline.Directory;
  DirectoryPanel.Caption := DirectoryOutline.Directory;
end;

procedure TFMForm.FileListChange(Sender: TObject);
var
  TheFileName: string;
begin
  with FileList do
  begin
    if ItemIndex >= 0 then
    begin
      TheFileName := Items[ItemIndex];
      FilePanel.Caption := Format('Вибрано файл %s, %d байтів', [TheFileName,
GetFileSize(TheFileName)]);
    end
    else FilePanel.Caption := '';
  end;
end;

procedure TFMForm.DriveTabSetMeasureTab(Sender: TObject; Index: Integer;
  var TabWidth: Integer);
var
  BitmapWidth: Integer;
begin
  BitmapWidth := TBitmap(DriveTabSet.Tabs.Objects[Index]).Width;
  Inc(TabWidth, 2 + BitmapWidth);
end;

procedure TFMForm.DriveTabSetDrawTab(Sender: TObject; TabCanvas: TCanvas;
  R: TRect; Index: Integer; Selected: Boolean);
var
  Bitmap: TBitmap;
begin
  Bitmap := TBitmap(DriveTabSet.Tabs.Objects[Index]);
  with TabCanvas do
  begin
    Draw(R.Left, R.Top + 4, Bitmap);
    TextOut(R.Left + 2 + Bitmap.Width, R.Top + 2, DriveTabSet.Tabs[Index]);
  end;
end;

procedure TFMForm.File1Click(Sender: TObject);
var
  FileSelected: Boolean;
begin
  FileSelected := FileList.ItemIndex >= 0;
  Open1.Enabled := FileSelected;
  Delete1.Enabled := FileSelected;
  Copy1.Enabled := FileSelected;
  Move1.Enabled := FileSelected;
  Rename1.Enabled := FileSelected;
  Properties1.Enabled := FileSelected;
end;

procedure TFMForm.Delete1Click(Sender: TObject);
begin
  with FileList do
```

```
    if MessageDlg('Delete ' + FileName + '?', mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
      if DeleteFile(FileName) then Update;
end;

procedure TFMForm.Properties1Click(Sender: TObject);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    FilePathName.Caption := FileList.Directory;
    ChangeDate.Caption := DateTimeToStr(FileDateTime(FileList.FileName));
    Attributes := FileGetAttr(FileDirName.Caption);
    ReadOnly.Checked := (Attributes and faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> mrCancel then
    begin
      NewAttributes := Attributes;
      if ReadOnly.Checked then NewAttributes := NewAttributes or faReadOnly
      else NewAttributes := NewAttributes and not faReadOnly;
      if Archive.Checked then NewAttributes := NewAttributes or faArchive
      else NewAttributes := NewAttributes and not faArchive;
      if System.Checked then NewAttributes := NewAttributes or faSysFile
      else NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then NewAttributes := NewAttributes or faHidden
      else NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then
        FileSetAttr(FileDirName.Caption, NewAttributes);
    end;
  end;
end;

procedure TFMForm.ConfirmChange(const ACaption, FromFile, ToFile: string);
begin
  if MessageDlg(Format('%s %s to %s?', [ACaption, FromFile, ToFile]),
    mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  begin
    if ACaption = 'Move' then
      MoveFile(FromFile, ToFile)
    else if ACaption = 'Copy' then
      CopyFile(FromFile, ToFile)
    else if ACaption = 'Rename' then
      RenameFile(FromFile, ToFile);
    FileList.Update;
  end;
end;

procedure TFMForm.FileChange(Sender: TObject);
begin
  with ChangeDlg do
  begin
    if Sender = Move1 then Caption := 'Move'
    else if Sender = Copy1 then Caption := 'Copy'
    else if Sender = Rename1 then Caption := 'Rename'
    else Exit;
    CurrentDir.Caption := DirectoryOutline.Directory;
    FromFileName.Text := FileList.FileName;
    ToFileName.Text := '';
    if (ShowModal <> mrCancel) and (ToFileName.Text <> '') then
      ConfirmChange(Caption, FromFileName.Text, ToFileName.Text);
  end;
end;

procedure TFMForm.Open1Click(Sender: TObject);
begin
```

```
  with FileList do
  begin
    if HasAttr(FileName, faDirectory) then
      DirectoryOutline.Directory := FileName
    else ExecuteFile(FileName, '', Directory, SW_SHOW);
  end;
end;

procedure TFMForm.FileListMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  with Sender as TFileListBox do
  begin
    if ItemAtPos(Point(X, Y), True) >= 0 then
      BeginDrag(False);
  end;
end;

procedure TFMForm.DirectoryOutlineDragOver(Sender, Source: TObject; X,
  Y: Integer; State: TDragState; var Accept: Boolean);
begin
  Accept := (Source is TFileListBox) and (DirectoryOutline.GetItem(X, Y) > 0);
end;

procedure TFMForm.DirectoryOutlineDragDrop(Sender, Source: TObject; X,
  Y: Integer);
begin
  if Source is TFileListBox then
    with DirectoryOutline do
      ConfirmChange('Move', FileList.FileName, Items[GetItem(X, Y)].FullPath);
end;

procedure TFMForm.FileListEndDrag(Sender, Target: TObject; X, Y: Integer);
begin
  if Target <> nil then FileList.Update;
end;

procedure TFMForm.N4Click(Sender: TObject);
begin
Form1.Show;
end;

procedure TFMForm.Button2Click(Sender: TObject);
begin
RSA_keys.Show;
end;

procedure TFMForm.Button1Click(Sender: TObject);
begin
DES.Show;
end;

procedure TFMForm.Button3Click(Sender: TObject);
I:Integer;
S:String;
Begin
SetLength(Data,0);
I:=1;
While I<=Length(fileX) Do
  Begin
  S:=Copy(fileX,I,8);
  Data:=ConcatBits([Data,DESEncode(S, fileX)]);
  I:=I+8;
  End;
fileY:=BinToAnsiStr(Data);
MessageBox(Handle,   'Файл зашифровано!',   'EFS',MB_OK    );
end;
```

```
procedure TFMForm.Button4Click(Sender: TObject);
I:Integer;
Begin
SetLength(Data,0);
I:=1;
While I<=Length(fileY) Do
  Begin
Data:=ConcatBits([Data,DESDecode(Copy(fileY,I,8), fileX)]);
  I:=I+8;
  End;
fileX:=BinToAnsiStr(Data);
MessageBox(Handle,   'Файл розшифровано!',   'EFS',MB_OK   );
end;

end.
```

**Файл FmxUtils.pas – робота з файлами**

```
unit FmxUtils;

interface

uses SysUtils, Windows, Classes, Consts;

type
  EInvalidDest = class(EStreamError);
  EFCantMove = class(EStreamError);

procedure CopyFile(const FileName, DestName: string);
procedure MoveFile(const FileName, DestName: string);
function GetFileSize(const FileName: string): LongInt;
function FileDateTime(const FileName: string): TDateTime;
function HasAttr(const FileName: string; Attr: Word): Boolean;
function ExecuteFile(const FileName, Params, DefaultDir: string;
  ShowCmd: Integer): THandle;

implementation

uses Forms, ShellAPI, RtlConsts;

const
  SInvalidDest = 'Вказаний каталог %s не існує';
  SFCantMove = 'Не можу перемістити файл %s';

procedure CopyFile(const FileName, DestName: string);
var
  CopyBuffer: Pointer; { буфер для копіювання }
  BytesCopied: Longint;
  Source, Dest: Integer; { заголовки }
  Len: Integer;
  Destination: TFileName; { розширене місце розташування }
const
  ChunkSize: Longint = 8192; { копіювання в 8К блок }
begin
  Destination := ExpandFileName(DestName); { розширення шляху призначення }
  if HasAttr(Destination, faDirectory) then { якщо розширення є директорією... }
  begin
    Len :=  Length(Destination);
    if Destination[Len] = '\' then
      Destination := Destination + ExtractFileName(FileName) { ...clone file
name }
    else
      Destination := Destination + '\' + ExtractFileName(FileName); {
...клонуємо ім'я файлу }
  end;
GetMem(CopyBuffer, ChunkSize); { виділяємо пам'ять під буфер }
  try
    Source := FileOpen(FileName, fmShareDenyWrite); { відкриваємо файл джерела}
    if Source < 0 then raise EFOpenError.CreateFmt(SFOpenError, [FileName]);
    try
      Dest := FileCreate(Destination); { створюємо вихідний файл або
перезаписуємо }
      if Dest < 0 then raise EFCreateError.CreateFmt(SFCreateError,
[Destination]);
      try
        repeat
          BytesCopied := FileRead(Source, CopyBuffer^, ChunkSize); { читаємо
блок }
          if BytesCopied > 0 then { якщо ми прочитали... }
            FileWrite(Dest, CopyBuffer^, BytesCopied); { ...записуємо блок }
        until BytesCopied < ChunkSize; { працюємо поки не закінчуються блоки }
      finally
        FileClose(Dest); { закриваємо файл розширення }
```

```
      end;
    finally
      FileClose(Source); { закриваємо файл джерела }
    end;
  finally
    FreeMem(CopyBuffer, ChunkSize); { звільнення буфера }
  end;
end;


{ Процедура переміщення файла }

procedure MoveFile(const FileName, DestName: string);
var
  Destination: string;
begin
  Destination := ExpandFileName(DestName); { розширюємо шлях призначення  }
  if not RenameFile(FileName, Destination) then { переіменовуємо }
  begin
    if HasAttr(FileName, faReadOnly) then  { якщо тільки для читання... }
      raise EFCantMove.Create(Format(SFCantMove, [FileName])); { не в змозі
знищити }
      CopyFile(FileName, Destination); { копіюємо в буфер...}
//      DeleteFile(FileName); { ...та знищуємо оригінал }
  end;
end;

{ GetFileSize функція, повертає розмір файлу }

function GetFileSize(const FileName: string): LongInt;
var
  SearchRec: TSearchRec;
begin
  try
    if FindFirst(ExpandFileName(FileName), faAnyFile, SearchRec) = 0 then
      Result := SearchRec.Size
    else Result := -1;
  finally
    SysUtils.FindClose(SearchRec);
  end;
end;

function FileDateTime(const FileName: string): System.TDateTime;
begin
  Result := FileDateToDateTime(FileAge(FileName));
end;

function HasAttr(const FileName: string; Attr: Word): Boolean;
var
 FileAttr: Integer;
begin
  FileAttr := FileGetAttr(FileName);
  if FileAttr = -1 then FileAttr := 0;
  Result := (FileAttr and Attr) = Attr;
end;

function ExecuteFile(const FileName, Params, DefaultDir: string;
  ShowCmd: Integer): THandle;
var
  zFileName, zParams, zDir: array[0..79] of Char;
begin
  Result := ShellExecute(Application.MainForm.Handle, nil,
    StrPCopy(zFileName, FileName), StrPCopy(zParams, Params),
    StrPCopy(zDir, DefaultDir), ShowCmd);
end;

end.
```

**Файл des_key.pas – введення користувачем секретного ключа DES**

```
unit des_key;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TDES = class(TForm)
    EditDES: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  DES: TDES;

implementation

{$R *.dfm}

procedure TDES.Button1Click(Sender: TObject);
begin
RSA_keys.Edit1.Text:=Edit1.Text;
DES.Close;
end;

end.
```

**Файл RSA_keys.pas – генерація ключів RSA та шифрування ключа DES**

```
unit RSA_keys;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, FGInt, FGIntPrimeGeneration, FGIntRSA, StdCtrls, des_key;

type
  TRSA_keys = class(TForm)
    Button1: TButton;
    Label3: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Edit3: TEdit;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit_E: TEdit;
    Edit_N: TEdit;
    Edit_D: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label4: TLabel;
    Button5: TButton;
    Edit1: TEdit;
    Label122: TLabel;
    Label12: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  RSA_keys: TRSA_keys;
   n, e, d, dp, dq, p, q, phi, one, two, gcd, temp, nilgint : TFGInt;
   test, signature : String;
   ok : boolean;
   st: string;
   tx1: text;
   tx2: text;
   tx3: text;
implementation



{$R *.dfm}

procedure TRSA_keys.Button1Click(Sender: TObject);
begin

// Генерація p та q
```

```
   Base256StringToFGInt('3557', p);
   Base256StringToFGInt('2579', q);
   PrimeSearch(p);
   PrimeSearch(q);
   FGIntToBase256String(p, st);
   FGIntToBase256String(q, st);


 // Обчислення N

   FGIntMul(p, q, n);
   p.Number[1] := p.Number[1] - 1;
   q.Number[1] := q.Number[1] - 1;
   FGIntMul(p, q, phi);
   FGIntToBase10String(n, st);
   Edit_N.Text:=st;


// Обчислення E – ключ шифрування

//   Base10StringToFGInt('65537', e); // непарне число
//   Base10StringToFGInt('8171921', e);
   Base10StringToFGInt('14486581214143', e);
   Base10StringToFGInt('1', one);
   Base10StringToFGInt('2', two);
   FGIntGCD(phi, e, gcd);
   While FGIntCompareAbs(gcd, one) <> Eq Do
   Begin
      FGIntadd(e, two, temp);
      FGIntCopy(temp, e);
      FGIntGCD(phi, e, gcd);
   End;
   FGIntDestroy(two);
   FGIntDestroy(one);
   FGIntDestroy(gcd);

// Обчислення D – ключ дешифрування

   FGIntModInv(e, phi, d);
   FGIntModInv(e, p, dp);
   FGIntModInv(e, q, dq);
   p.Number[1] := p.Number[1] + 1;
   q.Number[1] := q.Number[1] + 1;
   FGIntDestroy(phi);
   FGIntDestroy(nilgint);


   FGIntToBase10String(e, st);
   Edit_E.Text:=st;
   FGIntToBase10String(d, st);
   Edit_D.Text:=st;


end;

//дешифрування ключа DES

procedure TRSA_keys.Button2Click(Sender: TObject);
var prom: string;
begin
   test := Edit3.Text;
   RSAEncrypt(test, e, n, test);
   Edit1.Text:=test;
end;

//запис ключів RSA у файли

procedure TRSA_keys.Button3Click(Sender: TObject);
begin
```

```
  AssignFile(tx1,'OpenKey.keys');
  AssignFile(tx2,'CloseKey.keys');

  ReWrite(tx1); CloseFile(tx1);
  ReWrite(tx2); CloseFile(tx2);
  Append(tx1);
  Append(tx2);
  FGIntToBase256String(n, st);
  ConvertBase256to64(st, st);
  WriteLn(tx1,st);
  WriteLn(tx2,st);
  FGIntToBase256String(e, st);
  ConvertBase256to64(st, st);
  WriteLn(tx1,st);
  FGIntToBase256String(d, st);
  ConvertBase256to64(st, st);
  WriteLn(tx2,st);
  CloseFile(tx1);
  CloseFile(tx2);

end;

//шифрування ключа DES


procedure TRSA_keys.Button4Click(Sender: TObject);
begin
 RSADecrypt(test, d, n, Nilgint, Nilgint, Nilgint, Nilgint, test);

       Edit3.Text:=test;

end;


//Запис ключа DES у файл

procedure TRSA_keys.Button5Click(Sender: TObject);
begin
AssignFile(tx3,'SecretKey.keys');
  ReWrite(tx3); CloseFile(tx3);
  Append(tx3);
  WriteLn(tx3,Edit1.Text);
  CloseFile(tx3);
end;

procedure TRSA_keys.FormCreate(Sender: TObject);
begin
Edit1.Text:=DES.EditDES.Text;
end;

end.
```

**Файл DES.pas – алгоритм шифрування DES**

```
unit DES;

interface

Uses Windows, Classes, SysUtils, Math, Dialogs;

Type
  TBitString = Array of Boolean;
  PBitString = ^TBitString;

  TSplitKeyParts = record
    C:TBitString;
    D:TBitString;
    end;
  TSplitKey = Array[0..16]Of TSplitKeyParts;

  TConcatKey = Array[0..15]Of TBitString;

  TIPKeyParts = record
    L:TBitString;
    R:TBitString;
    end;
  TIPKey = Array[0..16]OF TIPKeyParts;

//матриці перестановок

Const
DES_PC1:Array[0..55] Of Byte = (57,49,41,33,25,17,9,
                                1,58,50,42,34,26,18,
                                10,2,59,51,43,35,27,
                                19,11,3,60,52,44,36,
                                63,55,47,39,31,23,15,
                                7,62,54,46,38,30,22,
                                14,6,61,53,45,37,29,
                                21,13,5,28,20,12,4);

DES_PC2:Array[0..47] Of Byte = (14,17,11,24,1,5,
                                3,28,15,6,21,10,
                                23,19,12,4,26,8,
                                16,7,27,20,13,2,
                                41,52,31,37,47,55,
                                30,40,51,45,33,48,
                                44,49,39,56,34,53,
                                46,42,50,36,29,32);

DES_IP:Array[0..63] Of Byte = (58,50,42,34,26,18,10,2,
                               60,52,44,36,28,20,12,4,
                               62,54,46,38,30,22,14,6,
                               64,56,48,40,32,24,16,8,
                               57,49,41,33,25,17,9,1,
                               59,51,43,35,27,19,11,3,
                               61,53,45,37,29,21,13,5,
                               63,55,47,39,31,23,15,7);

DES_E:Array[0..47] Of Byte = (32,1,2,3,4,5,
                              4,5,6,7,8,9,
                              8,9,10,11,12,13,
                              12,13,14,15,16,17,
                              16,17,18,19,20,21,
                              20,21,22,23,24,25,
                              24,25,26,27,28,29,
                              28,29,30,31,32,1);

// S-блоки
```

```
S_BOXES:Array[0..7,0..3,0..15]Of Byte = (
((14,04,13,01,02,15,11,08,03,10,06,12,05,09,00,07),
  (00,15,07,04,14,02,13,01,10,06,12,11,09,05,03,08),
  (04,01,14,08,13,06,02,11,15,12,09,07,03,10,05,00),
  (15,12,08,02,04,09,01,07,05,11,03,14,10,00,06,13)),

((15,01,08,14,06,11,03,04,09,07,02,13,12,00,05,10),
   (03,13,04,07,15,02,08,14,12,00,01,10,06,09,11,05),
   (00,14,07,11,10,04,13,01,05,08,12,06,09,03,02,15),
   (13,08,10,01,03,15,04,02,11,06,07,12,00,05,14,09)),

((10,00,09,14,06,03,15,05,01,13,12,07,11,04,02,08),
   (13,07,00,09,03,04,06,10,02,08,05,14,12,11,15,01),
   (13,06,04,09,08,15,03,00,11,01,02,12,05,10,14,07),
   (01,10,13,00,06,09,08,07,04,15,14,03,11,05,02,12)),

((07,13,14,03,00,06,09,10,01,02,08,05,11,12,04,15),
   (13,08,11,05,06,15,00,03,04,07,02,12,01,10,14,09),
   (10,06,09,00,12,11,07,13,15,01,03,14,05,02,08,04),
   (13,15,00,06,10,01,13,08,09,04,05,11,12,07,02,14)),

((02,12,04,01,07,10,11,06,08,05,03,15,13,00,14,09),
   (14,11,02,12,04,07,13,01,05,00,15,10,03,08,09,06),
   (04,02,01,11,10,13,07,08,15,09,12,05,06,03,00,14),
   (11,08,12,07,01,14,02,13,06,15,00,09,10,04,05,03)),

((12,01,10,15,09,02,06,08,00,13,03,04,14,07,05,11),
   (10,15,04,02,07,12,09,05,06,01,13,14,00,11,03,08),
   (09,14,15,05,02,08,12,03,07,00,04,10,01,13,11,06),
   (04,03,02,12,09,05,15,10,11,14,01,04,06,00,08,13)),

((04,11,02,14,15,00,08,13,03,12,09,07,05,10,06,01),
   (13,00,11,07,04,09,01,10,14,03,05,12,02,15,08,06),
   (01,04,11,13,12,03,07,14,10,15,06,08,00,05,09,02),
   (06,11,13,08,01,04,10,07,09,05,00,15,14,02,03,12)),

((13,02,08,04,06,15,11,01,10,09,03,14,05,00,12,07),
   (01,15,13,08,10,03,07,04,12,05,06,11,00,14,09,02),
   (07,11,04,01,09,12,14,02,00,06,10,13,15,03,05,08),
   (02,01,14,07,04,10,08,13,15,12,09,00,03,05,06,11))
);

DES_P:Array[0..31] Of Byte = (16,7,20,21,
                              29,12,28,17,
                              1,15,23,26,
                              5,18,31,10,
                              2,8,24,14,
                              32,27,3,9,
                              19,13,30,6,
                              22,11,4,25);

DES_REVERSE_IP:Array[0..63] Of Byte = (40,8,48,16,56,24,64,32,
            39,7,47,15,55,23,63,31,
            38,6,46,14,54,22,62,30,
            37,5,45,13,53,21,61,29,
            36,4,44,12,52,20,60,28,
            35,3,43,11,51,19,59,27,
            34,2,42,10,50,18,58,26,
            33,1,41,9,49,17,57,25);

DES_LSH:Array[0..15] Of Byte = (1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1);

Function BinToInt(S:TBitString):Integer;
Function IntToBin(N:Integer;Precision:Integer=8):TBitString;

Function BinToStr(Bits:TBitString):String;
Function StrToBin(S:String):TBitString;
```

```
Function AnsiStrToBin(S:String; Zeroes:Boolean=True):TBitString;
Function BinToAnsiStr(Bits:TBitString):String;

Procedure CopyBits(Var Dest:TBitString; Source:TBitString; NBits:Integer);
Function ConcatBits(Bits:Array Of TBitString):TBitString;

Function DESEncode(S,Key:String):TBitString;
Function DESDecode(S,Key:String):TBitString;

Function GetPermutedKey(Key:TBitString):TBitString;
Function GetPermutedKey2(Key:TBitString):TBitString;

Function GetSplitKey(Key:TBitString):TSplitKey;
Function GetConcatKey(Key:TSplitKey):TConcatKey;
Function GetIPKey(M:TBitString; ConcatKey:TConcatKey):TIPKey;
Function GetF(R,K:TBitString):TBitString;
Function GetSBox(Index:Integer; T:TBitString):TBitString;
Function GetReverseIP(RL:TBitString):TBitString;
Procedure ReverseSubKeys(Var Keys:TConcatKey);

implementation

Function ConcatBits(Bits:Array Of TBitString):TBitString;
Var
I,C:Integer;
Begin
SetLength(Result,0);
For C:=0 To Length(Bits)-1 Do
  Begin
  SetLength(Result,Length(Result)+Length(Bits[C]));
  For I:=0 To Length(Bits[C])-1 Do
    Result[Length(Result)-Length(Bits[C])+I]:=Bits[C][I];
  End;
End;

Procedure CopyBits(Var Dest:TBitString; Source:TBitString; NBits:Integer);
Var
I:Integer;
Begin
SetLength(Dest,NBits);
For I:=0 To NBits-1 Do
  Dest[I]:=Source[I];
End;

Function BinToInt(S: TBitString): Integer;
Var
L,I:Integer;
Begin
Result:=0;
L:=Length(S);
IF L=0 Then
  Raise EConvertError.Create(спеціальна бітова строка довжиною 0 біт');
For I:=L-1 DownTo 0 Do
  Result:=Result+Ord(S[I])*Trunc(Power(2,L-I-1));
End;

Function IntToBin(N:Integer; Precision:Integer):TBitString;
Var
BitList:TList;
Bit:PBoolean;
Begin
SetLength(Result,0);
BitList:=TList.Create;
While N>0 Do
  Begin
  New(Bit);
  Bit^:=Boolean(N mod 2);
  BitList.Insert(0,Bit);
  N:=N div 2;
```

```
  End;
While BitList.Count<Precision Do
  Begin
  New(Bit);
  Bit^:=False;
  BitList.Insert(0,Bit);
  End;
For N:=0 To BitList.Count-1 Do
  Begin
  SetLength(Result,N+1);
  Bit:=BitList.Items[N];
  Result[N]:=Bit^;
  Dispose(Bit);
  End;
BitList.Free;
end;

Function AnsiStrToBin(S: String; Zeroes:Boolean):TBitString;
Var
Temp,B:TBitString;
L,I,J:Integer;
Begin
L:=0;
SetLength(Result,L);
SetLength(Temp,L);
SetLength(B,0);
For I:=1 To Length(S) Do
  Begin
  B:=IntToBin(Ord(S[I]));
  L:=L+Length(B);
  SetLength(Temp,L);
  For J:=0 To Length(B)-1 Do
    Temp[Length(Temp)-Length(B)+J]:=B[J];
  End;
Result:=Temp;
End;

Function BinToStr(Bits:TBitString):String;
Var
I,L:Integer;
Begin
Result:='';
L:=Length(Bits);
IF L=0 Then
  Raise EConvertError.Create('Спеціальна бітова строка довжиною 0 біт');
For I:=0 To L-1 Do
  IF Bits[I] Then Result:=Result+'1'
  Else Result:=Result+'0';
End;

Function StrToBin(S:String):TBitString;
Var
I:Integer;
Begin
SetLength(Result,0);
For I:=1 To Length(S) Do
  Begin
  IF (S[I]<>'1')And(S[I]<>'0') Then
    Raise EConvertError.Create(S+' некоректна бінарна строка');
  SetLength(Result,I);
  Result[I-1]:=Boolean(StrToInt(S[I]));
  End;
End;

Function BinToAnsiStr(Bits:TBitString):String;
Var
I:Integer;
B:TBitString;
Begin
```

```
Result:='';
SetLength(B,8);
I:=0;
While I<=Length(Bits)-8 Do
  Begin
  CopyMemory(B,Ptr(Integer(Bits)+I),8);
  Result:=Result+Char(BinToInt(B));
  Inc(I,8);
  End;
End;

Function GetPermutedKey(Key:TBitString):TBitString;
Var
I:Integer;
Begin
SetLength(Result,Length(DES_PC1));
For I:=0 To Length(DES_PC1)-1 Do
  Result[I]:=Key[DES_PC1[I]-1];
End;

Function GetPermutedKey2(Key:TBitString):TBitString;
Var
I:Integer;
Begin
SetLength(Result,Length(DES_PC2));
For I:=0 To Length(DES_PC2)-1 Do
  Result[I]:=Key[DES_PC2[I]-1];
End;

Function GetSplitKey(Key:TBitString):TSplitKey;
  Function LeftShift(Key:TBitString; N:Integer):TBitString;
  Var
  I,J:Integer;
  Temp:TBitString;
  Begin
  SetLength(Result,28);
  SetLength(Temp,28);
  For I:=0 To 27 Do
    Temp[I]:=Key[I];
  For J:=1 To N Do
    Begin
    For I:=1 To 27 Do
      Result[I-1]:=Temp[I];
    Result[27]:=Temp[0];
    For I:=0 To 27 Do
      Temp[I]:=Result[I];
    End;
  End;
Var
I,J:Integer;
Begin
For J:=1 To 16 Do
  Begin
  SetLength(Result[J].C,28);
  SetLength(Result[J].D,28);
  End;
CopyBits(Result[0].C,Key,28);
CopyBits(Result[0].D,TBitString(Integer(Key)+28),28);
For I:=1 To 16 Do
  Begin
  Result[I].C:=LeftShift(Result[I-1].C,DES_LSH[I-1]);
  Result[I].D:=LeftShift(Result[I-1].D,DES_LSH[I-1]);
  End;
End;

Function GetConcatKey(Key:TSplitKey):TConcatKey;
Var
I:Integer;
Temp:TBitString;
```

```
Begin
For I:=0 To 15 Do
  Begin
  SetLength(Result[I],56);
  Temp:=ConcatBits([Key[I+1].C,Key[I+1].D]);
  Result[I]:=GetPermutedKey2(Temp);
  End;
End;

Function GetIPKey(M:TBitString; ConcatKey:TConcatKey):TIPKey;
Var
I,J:Integer;
IP, F:TBitString;
Begin
For I:=0 To 16 Do
  Begin
  SetLength(Result[I].L,32);
  SetLength(Result[I].R,32);
  End;

SetLength(IP,64);
For I:=0 To Length(DES_IP)-1 Do
  IP[I]:=M[DES_IP[I]-1];

For I:=0 To 31 Do
  Result[0].L[I]:=IP[I];
For I:=32 To 63 Do
  Result[0].R[I-32]:=IP[I];

For I:=1 To 16 Do
  Begin
  Result[I].L:=Result[I-1].R;
  F:=GetF(Result[I-1].R,ConcatKey[I-1]);
  For J:=0 To 31 Do
    Result[I].R[J]:=Result[I-1].L[J] XOR F[J];
  End;
End;

Function GetF(R,K:TBitString):TBitString;
Var
I,J:Integer;
S,E,KE,F,T:TBitString;
Begin
SetLength(E,48);
For I:=0 To 47 Do
  E[I]:=R[DES_E[I]-1];

SetLength(KE,48);
For I:=0 To 47 Do
  KE[I]:=K[I] XOR E[I];

SetLength(T,6);
SetLength(F,0);
SetLength(S,4);
I:=0;
While I<48 Do
  Begin
  For J:=0 To 6 Do
    T[J]:=KE[J+I];
  S:=GetSBox(I div 6,T);
  F:=ConcatBits([F,S]);
  I:=I+6;
  End;
SetLength(Result,32);
For I:=0 To 31 Do
  Result[I]:=F[DES_P[I]-1];
End;

Function GetSBox(Index:Integer; T:TBitString):TBitString;
```

```
Var
Val,Row,Col:Integer;
Temp:TBitString;
Begin
SetLength(Result,4);
SetLength(Temp,2);
Temp[0]:=T[0];
Temp[1]:=T[5];
Row:=BinToInt(Temp);
SetLength(Temp,4);
CopyBits(Temp,TBitString(@T[1]),4);
Col:=BinToInt(Temp);
Val:=S_BOXES[Index,Row,Col];
SetLength(Result,4);
Result:=IntToBin(Val,4);
End;

Function GetReverseIP(RL:TBitString):TBitString;
Var
I:Integer;
Begin
SetLength(Result,64);
For I:=0 To Length(DES_REVERSE_IP)-1 Do
  Result[I]:=RL[DES_REVERSE_IP[I]-1];
End;

Procedure ReverseSubKeys(Var Keys:TConcatKey);
Var
I,L:Integer;
T:TBitString;
Begin
SetLength(T,48);
L:=Length(Keys);
For I:=0 To (L-1)Div 2 Do
  Begin
  T:=Keys[I];
  Keys[I]:=Keys[(L-I)-1];
  Keys[(L-I)-1]:=T;
  End;
End;

Function DESEncode(S,Key:String):TBitString;
Var
I:Integer;
K:TBitString;
M:TBitString;
RL:TBitString;
Kplus:TBitString;
SplitKey:TSplitKey;
ConcatKey:TConcatKey;
IPKey:TIPKey;
Begin
K:=AnsiStrToBin(Key);
Kplus:=GetPermutedKey(K);
SplitKey:=GetSplitKey(Kplus);
ConcatKey:=GetConcatKey(SplitKey);
M:=AnsiStrToBin(S);
IPKey:=GetIPKey(M,ConcatKey);
SetLength(RL,64);
For I:=0 To 31 Do
  Begin
  RL[I]:=IPKey[16].R[I];
  RL[I+32]:=IPKey[16].L[I];
  End;
RL:=GetReverseIP(RL);
Result:=RL;
End;

Function DESDecode(S,Key:String):TBitString;
```

```
Var
I:Integer;
K:TBitString;
M:TBitString;
RL:TBitString;
Kplus:TBitString;
SplitKey:TSplitKey;
ConcatKey:TConcatKey;
IPKey:TIPKey;
Begin
K:=AnsiStrToBin(Key);
Kplus:=GetPermutedKey(K);
SplitKey:=GetSplitKey(Kplus);
ConcatKey:=GetConcatKey(SplitKey);
ReverseSubKeys(ConcatKey);
M:=AnsiStrToBin(S);
IPKey:=GetIPKey(M,ConcatKey);
SetLength(RL,64);
For I:=0 To 31 Do
  Begin
  RL[I]:=IPKey[16].R[I];
  RL[I+32]:=IPKey[16].L[I];
  End;
RL:=GetReverseIP(RL);
Result:=RL;
End; end.
```

**Файл FGIntRSA.pas – алгоритм шифрування RSA**

```
Unit FGIntRSA;

Interface

Uses Windows, SysUtils, Controls, FGInt;

Procedure RSAEncrypt(P : String; Var exp, modb : TFGInt; Var E : String);
Procedure RSADecrypt(E : String; Var exp, modb, d_p, d_q, p, q : TFGInt; Var D :
String);
Procedure RSASign(M : String; Var d, n, dp, dq, p, q : TFGInt; Var S : String);
Procedure RSAVerify(M, S : String; Var e, n : TFGInt; Var valid : boolean);


Implementation


{$H+}




// Шифруємо рядок алгоритмом RSA, P^exp mod modb = E

Procedure RSAEncrypt(P : String; Var exp, modb : TFGInt; Var E : String);
Var
   i, j, modbits : longint;
   PGInt, temp, zero : TFGInt;
   tempstr1, tempstr2, tempstr3 : String;
Begin
   Base2StringToFGInt('0', zero);
   FGIntToBase2String(modb, tempstr1);
   modbits := length(tempstr1);
   convertBase256to2(P, tempstr1);
   tempstr1 := '111' + tempstr1;
   j := modbits - 1;
   While (length(tempstr1) Mod j) <> 0 Do tempstr1 := '0' + tempstr1;

   j := length(tempstr1) Div (modbits - 1);
   tempstr2 := '';
   For i := 1 To j Do
   Begin
      tempstr3 := copy(tempstr1, 1, modbits - 1);
      While (copy(tempstr3, 1, 1) = '0') And (length(tempstr3) > 1) Do
delete(tempstr3, 1, 1);
      Base2StringToFGInt(tempstr3, PGInt);
      delete(tempstr1, 1, modbits - 1);
      If tempstr3 = '0' Then FGIntCopy(zero, temp) Else
FGIntMontgomeryModExp(PGInt, exp, modb, temp);
      FGIntDestroy(PGInt);
      tempstr3 := '';
      FGIntToBase2String(temp, tempstr3);
      While (length(tempstr3) Mod modbits) <> 0 Do tempstr3 := '0' + tempstr3;
      tempstr2 := tempstr2 + tempstr3;
      FGIntdestroy(temp);
   End;

   While (tempstr2[1] = '0') And (length(tempstr2) > 1) Do delete(tempstr2, 1,
1);
   ConvertBase2To256(tempstr2, E);
   FGIntDestroy(zero);
End;


Дешифруємо рядок алгоритмом RSA, E^exp mod modb = D
// modb = p*q, d_p*e mod (p-1) = 1
// d_q*e mod (q-1)
```

```
Procedure RSADecrypt(E : String; Var exp, modb, d_p, d_q, p, q : TFGInt; Var D :
String);
Var
   i, j, modbits : longint;
   EGInt, temp, temp1, temp2, temp3, ppinvq, qqinvp, zero : TFGInt;
   tempstr1, tempstr2, tempstr3 : String;
Begin
   Base2StringToFGInt('0', zero);
   FGIntToBase2String(modb, tempstr1);
   modbits := length(tempstr1);
   convertBase256to2(E, tempstr1);
   While copy(tempstr1, 1, 1) = '0' Do delete(tempstr1, 1, 1);
   While (length(tempstr1) Mod modbits) <> 0 Do tempstr1 := '0' + tempstr1;
   If exp.Number = Nil Then
   Begin
      FGIntModInv(q, p, temp1);
      FGIntMul(q, temp1, qqinvp);
      FGIntDestroy(temp1);
      FGIntModInv(p, q, temp1);
      FGIntMul(p, temp1, ppinvq);
      FGIntDestroy(temp1);
   End;

   j := length(tempstr1) Div modbits;
   tempstr2 := '';
   For i := 1 To j Do
   Begin
      tempstr3 := copy(tempstr1, 1, modbits);
      While (copy(tempstr3, 1, 1) = '0') And (length(tempstr3) > 1) Do
delete(tempstr3, 1, 1);
      Base2StringToFGInt(tempstr3, EGInt);
      delete(tempstr1, 1, modbits);
      If tempstr3 = '0' Then FGIntCopy(zero, temp) Else
      Begin
         If exp.Number <> Nil Then FGIntMontgomeryModExp(EGInt, exp, modb, temp)
Else
         Begin
            FGIntMontgomeryModExp(EGInt, d_p, p, temp1);
            FGIntMul(temp1, qqinvp, temp3);
            FGIntCopy(temp3, temp1);
            FGIntMontgomeryModExp(EGInt, d_q, q, temp2);
            FGIntMul(temp2, ppinvq, temp3);
            FGIntCopy(temp3, temp2);
            FGIntAddMod(temp1, temp2, modb, temp);
            FGIntDestroy(temp1);
            FGIntDestroy(temp2);
         End;
      End;
      FGIntDestroy(EGInt);
      tempstr3 := '';
      FGIntToBase2String(temp, tempstr3);
      While (length(tempstr3) Mod (modbits - 1)) <> 0 Do tempstr3 := '0' +
tempstr3;
      tempstr2 := tempstr2 + tempstr3;
      FGIntdestroy(temp);
   End;

   If exp.Number = Nil Then
   Begin
      FGIntDestroy(ppinvq);
      FGIntDestroy(qqinvp);
   End;
   While (Not (copy(tempstr2, 1, 3) = '111')) And (length(tempstr2) > 3) Do
delete(tempstr2, 1, 1);
   delete(tempstr2, 1, 3);
   ConvertBase2To256(tempstr2, D);
   FGIntDestroy(zero);
End;
```

```
// підписуємо рядок RSA, M^d mod n = S
// n = p*q, dp*e mod (p-1) = 1
// dq*e mod (q-1)


Procedure RSASign(M : String; Var d, n, dp, dq, p, q : TFGInt; Var S : String);
Var
   MGInt, SGInt, temp, temp1, temp2, temp3, ppinvq, qqinvp : TFGInt;
Begin
   Base256StringToFGInt(M, MGInt);
   If d.Number <> Nil Then FGIntMontgomeryModExp(MGInt, d, n, SGInt) Else
   Begin
      FGIntModInv(p, q, temp);
      FGIntMul(p, temp, ppinvq);
      FGIntDestroy(temp);
      FGIntModInv(q, p, temp);
      FGIntMul(q, temp, qqinvp);
      FGIntDestroy(temp);
      FGIntMontgomeryModExp(MGInt, dp, p, temp1);
      FGIntMul(temp1, qqinvp, temp2);
      FGIntCopy(temp2, temp1);
      FGIntMontgomeryModExp(MGInt, dq, q, temp2);
      FGIntMul(temp2, ppinvq, temp3);
      FGIntCopy(temp3, temp2);
      FGIntAddMod(temp1, temp2, n, SGInt);
      FGIntDestroy(temp1);
      FGIntDestroy(temp2);
      FGIntDestroy(ppinvq);
      FGIntDestroy(qqinvp);
   End;
   FGIntToBase256String(SGInt, S);
   FGIntDestroy(MGInt);
   FGIntDestroy(SGInt);
End;


// Перевіряємо правильність алгоритму  RSA,
// Якщо M = S^e mod n тоді ok:=true інакше ok:=false

Procedure RSAVerify(M, S : String; Var e, n : TFGInt; Var valid : boolean);
Var
   MGInt, SGInt, temp : TFGInt;
Begin
   Base256StringToFGInt(S, SGInt);
   Base256StringToFGInt(M, MGInt);
   FGIntMod(MGInt, n, temp);
   FGIntCopy(temp, MGInt);
   FGIntMontgomeryModExp(SGInt, e, n, temp);
   FGIntCopy(temp, SGInt);
   valid := (FGIntCompareAbs(SGInt, MGInt) = Eq);
   FGIntDestroy(SGInt);
   FGIntDestroy(MGInt);
End;

End.
```

**Файл FGInt.pas – робота з великими числами для алгоритму RSA**

```
Unit FGInt;

Interface

Uses Windows, SysUtils, Controls, Math;

Type
   TCompare = (Lt, St, Eq, Er);
   TSign = (negative, positive);
   TFGInt = Record
      Sign : TSign;
      Number : Array Of int64;
   End;

Procedure zeronetochar8(Var g : char; Const x : String);
Procedure zeronetochar6(Var g : integer; Const x : String);
Procedure initialize8(Var trans : Array Of String);
Procedure initialize6(Var trans : Array Of String);
Procedure initialize6PGP(Var trans : Array Of String);
Procedure ConvertBase256to64(Const str256 : String; Var str64 : String);
Procedure ConvertBase64to256(Const str64 : String; Var str256 : String);
Procedure ConvertBase256to2(Const str256 : String; Var str2 : String);
Procedure ConvertBase64to2(Const str64 : String; Var str2 : String);
Procedure ConvertBase2to256(str2 : String; Var str256 : String);
Procedure ConvertBase2to64(str2 : String; Var str64 : String);
Procedure ConvertBase256StringToHexString(Str256 : String; Var HexStr : String);
Procedure ConvertHexStringToBase256String(HexStr : String; Var Str256 : String);
Procedure PGPConvertBase256to64(Var str256, str64 : String);
Procedure PGPConvertBase64to256(str64 : String; Var str256 : String);
Procedure PGPConvertBase64to2(str64 : String; Var str2 : String);
Procedure Base10StringToFGInt(Base10 : String; Var FGInt : TFGInt);
Procedure FGIntToBase10String(Const FGInt : TFGInt; Var Base10 : String);
Procedure FGIntDestroy(Var FGInt : TFGInt);
Function FGIntCompareAbs(Const FGInt1, FGInt2 : TFGInt) : TCompare;
Procedure FGIntAdd(Const FGInt1, FGInt2 : TFGInt; Var Sum : TFGInt);
Procedure FGIntChangeSign(Var FGInt : TFGInt);
Procedure FGIntSub(Var FGInt1, FGInt2, dif : TFGInt);
Procedure FGIntMulByInt(Const FGInt : TFGInt; Var res : TFGInt; by : int64);
Procedure FGIntMulByIntbis(Var FGInt : TFGInt; by : int64);
Procedure FGIntDivByInt(Const FGInt : TFGInt; Var res : TFGInt; by : int64; Var
modres : int64);
Procedure FGIntDivByIntBis(Var FGInt : TFGInt; by : int64; Var modres : int64);
Procedure FGIntModByInt(Const FGInt : TFGInt; by : int64; Var modres : int64);
Procedure FGIntAbs(Var FGInt : TFGInt);
Procedure FGIntCopy(Const FGInt1 : TFGInt; Var FGInt2 : TFGInt);
Procedure FGIntShiftLeft(Var FGInt : TFGInt);
Procedure FGIntShiftRight(Var FGInt : TFGInt);
Procedure FGIntShiftRightBy31(Var FGInt : TFGInt);
Procedure FGIntAddBis(Var FGInt1 : TFGInt; Const FGInt2 : TFGInt);
Procedure FGIntSubBis(Var FGInt1 : TFGInt; Const FGInt2 : TFGInt);
Procedure FGIntMul(Const FGInt1, FGInt2 : TFGInt; Var Prod : TFGInt);
Procedure FGIntSquare(Const FGInt : TFGInt; Var Square : TFGInt);
Procedure FGIntToBase2String(Const FGInt : TFGInt; Var S : String);
Procedure Base2StringToFGInt(S : String; Var FGInt : TFGInt);
Procedure FGIntToBase256String(Const FGInt : TFGInt; Var str256 : String);
Procedure Base256StringToFGInt(str256 : String; Var FGInt : TFGInt);
Procedure PGPMPIToFGInt(PGPMPI : String; Var FGInt : TFGInt);
Procedure FGIntToPGPMPI(FGInt : TFGInt; Var PGPMPI : String);
Procedure FGIntExp(Const FGInt, exp : TFGInt; Var res : TFGInt);
Procedure FGIntFac(Const FGInt : TFGInt; Var res : TFGInt);
Procedure FGIntShiftLeftBy31(Var FGInt : TFGInt);
Procedure FGIntDivMod(Var FGInt1, FGInt2, QFGInt, MFGInt : TFGInt);
Procedure FGIntDiv(Var FGInt1, FGInt2, QFGInt : TFGInt);
Procedure FGIntMod(Var FGInt1, FGInt2, MFGInt : TFGInt);
Procedure FGIntSquareMod(Var FGInt, Modb, FGIntSM : TFGInt);
```

```
Procedure FGIntAddMod(Var FGInt1, FGInt2, base, FGIntres : TFGInt);
Procedure FGIntMulMod(Var FGInt1, FGInt2, base, FGIntres : TFGInt);
Procedure FGIntModExp(Var FGInt, exp, modb, res : TFGInt);
Procedure FGIntModBis(Const FGInt : TFGInt; Var FGIntOut : TFGInt; b : longint;
head : int64);
Procedure FGIntMulModBis(Const FGInt1, FGInt2 : TFGInt; Var Prod : TFGInt; b :
longint; head : int64);
Procedure FGIntMontgomeryMod(Const GInt, base, baseInv : TFGInt; Var MGInt :
TFGInt; b : longint; head : int64);
Procedure FGIntMontgomeryModExp(Var FGInt, exp, modb, res : TFGInt);
Procedure FGIntGCD(Const FGInt1, FGInt2 : TFGInt; Var GCD : TFGInt);
Procedure FGIntLCM(Const FGInt1, FGInt2 : TFGInt; Var LCM : TFGInt);
Procedure FGIntTrialDiv9999(Const FGInt : TFGInt; Var ok : boolean);
Procedure FGIntRandom1(Var Seed, RandomFGInt : TFGInt);
Procedure FGIntRabinMiller(Var FGIntp : TFGInt; nrtest : integer; Var ok :
boolean);
Procedure FGIntBezoutBachet(Var FGInt1, FGInt2, a, b : TFGInt);
Procedure FGIntModInv(Const FGInt1, base : TFGInt; Var Inverse : TFGInt);
Procedure FGIntPrimetest(Var FGIntp : TFGInt; nrRMtests : integer; Var ok :
boolean);
Procedure FGIntLegendreSymbol(Var a, p : TFGInt; Var L : integer);
Procedure FGIntSquareRootModP(Square, Prime : TFGInt; Var SquareRoot : TFGInt);



Implementation

Var
   primes : Array[1..1228] Of integer =
      (3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127,
      131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
211, 223, 227, 229, 233, 239, 241, 251,
      257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347,
349, 353, 359, 367, 373, 379, 383, 389,
      397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479,
487, 491, 499, 503, 509, 521, 523, 541,
      547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631,
641, 643, 647, 653, 659, 661, 673, 677,
      683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787,
797, 809, 811, 821, 823, 827, 829, 839,
      853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947,
953, 967, 971, 977, 983, 991, 997, 1009,
      1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087,
1091, 1093, 1097, 1103, 1109, 1117, 1123,
      1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223,
1229, 1231, 1237, 1249, 1259, 1277, 1279,
      1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367,
1373, 1381, 1399, 1409, 1423, 1427, 1429,
      1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493,
1499, 1511, 1523, 1531, 1543, 1549, 1553,
      1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621,
1627, 1637, 1657, 1663, 1667, 1669, 1693,
      1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783,
1787, 1789, 1801, 1811, 1823, 1831, 1847,
      1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933,
1949, 1951, 1973, 1979, 1987, 1993, 1997,
      1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083,
2087, 2089, 2099, 2111, 2113, 2129, 2131,
      2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239,
2243, 2251, 2267, 2269, 2273, 2281, 2287,
      2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377,
2381, 2383, 2389, 2393, 2399, 2411, 2417,
      2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539,
2543, 2549, 2551, 2557, 2579, 2591, 2593,
      2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687,
2689, 2693, 2699, 2707, 2711, 2713, 2719,
      2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803,
2819, 2833, 2837, 2843, 2851, 2857, 2861,
```

2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969,
2971, 2999, 3001, 3011, 3019, 3023, 3037,
3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163,
3167, 3169, 3181, 3187, 3191, 3203, 3209,
3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313,
3319, 3323, 3329, 3331, 3343, 3347, 3359,
3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463,
3467, 3469, 3491, 3499, 3511, 3517, 3527,
3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607,
3613, 3617, 3623, 3631, 3637, 3643, 3659,
3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761,
3767, 3769, 3779, 3793, 3797, 3803, 3821,
3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917,
3919, 3923, 3929, 3931, 3943, 3947, 3967,
3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073,
4079, 4091, 4093, 4099, 4111, 4127, 4129,
4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231,
4241, 4243, 4253, 4259, 4261, 4271, 4273,
4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397,
4409, 4421, 4423, 4441, 4447, 4451, 4457,
4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561,
4567, 4583, 4591, 4597, 4603, 4621, 4637,
4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723,
4729, 4733, 4751, 4759, 4783, 4787, 4789,
4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909,
4919, 4931, 4933, 4937, 4943, 4951, 4957,
4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039,
5051, 5059, 5077, 5081, 5087, 5099, 5101,
5107, 5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227,
5231, 5233, 5237, 5261, 5273, 5279, 5281,
5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407,
5413, 5417, 5419, 5431, 5437, 5441, 5443,
5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531,
5557, 5563, 5569, 5573, 5581, 5591, 5623,
5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701,
5711, 5717, 5737, 5741, 5743, 5749, 5779,
5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857,
5861, 5867, 5869, 5879, 5881, 5897, 5903,
5923, 5927, 5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047,
6053, 6067, 6073, 6079, 6089, 6091, 6101,
6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211,
6217, 6221, 6229, 6247, 6257, 6263, 6269,
6271, 6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353,
6359, 6361, 6367, 6373, 6379, 6389, 6397,
6421, 6427, 6449, 6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551,
6553, 6563, 6569, 6571, 6577, 6581, 6599,
6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703,
6709, 6719, 6733, 6737, 6761, 6763, 6779,
6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869,
6871, 6883, 6899, 6907, 6911, 6917, 6947,
6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019,
7027, 7039, 7043, 7057, 7069, 7079, 7103,
7109, 7121, 7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213,
7219, 7229, 7237, 7243, 7247, 7253, 7283,
7297, 7307, 7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411, 7417,
7433, 7451, 7457, 7459, 7477, 7481, 7487,
7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561,
7573, 7577, 7583, 7589, 7591, 7603, 7607,
7621, 7639, 7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717,
7723, 7727, 7741, 7753, 7757, 7759, 7789,
7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879, 7883, 7901,
7907, 7919, 7927, 7933, 7937, 7949, 7951,
7963, 7993, 8009, 8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089,
8093, 8101, 8111, 8117, 8123, 8147, 8161,
8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237, 8243, 8263,
8269, 8273, 8287, 8291, 8293, 8297, 8311,
8317, 8329, 8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431,
8443, 8447, 8461, 8467, 8501, 8513, 8521,

```
      8527, 8537, 8539, 8543, 8563, 8573, 8581, 8597, 8599, 8609, 8623, 8627,
8629, 8641, 8647, 8663, 8669, 8677, 8681,
      8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753, 8761,
8779, 8783, 8803, 8807, 8819, 8821, 8831,
      8837, 8839, 8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941,
8951, 8963, 8969, 8971, 8999, 9001, 9007,
      9011, 9013, 9029, 9041, 9043, 9049, 9059, 9067, 9091, 9103, 9109, 9127,
9133, 9137, 9151, 9157, 9161, 9173, 9181,
      9187, 9199, 9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283,
9293, 9311, 9319, 9323, 9337, 9341, 9343,
      9349, 9371, 9377, 9391, 9397, 9403, 9413, 9419, 9421, 9431, 9433, 9437,
9439, 9461, 9463, 9467, 9473, 9479, 9491,
      9497, 9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623,
9629, 9631, 9643, 9649, 9661, 9677, 9679,
      9689, 9697, 9719, 9721, 9733, 9739, 9743, 9749, 9767, 9769, 9781, 9787,
9791, 9803, 9811, 9817, 9829, 9833, 9839,
      9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941,
9949, 9967, 9973);
   chr64 : Array[1..64] Of char = ('a', 'A', 'b', 'B', 'c', 'C', 'd', 'D', 'e',
'E', 'f', 'F',
      'g', 'G', 'h', 'H', 'i', 'I', 'j', 'J', 'k', 'K', 'l', 'L', 'm', 'M', 'n',
'N', 'o', 'O', 'p',
      'P', 'q', 'Q', 'r', 'R', 's', 'S', 't', 'T', 'u', 'U', 'v', 'V', 'w', 'W',
'x', 'X', 'y', 'Y',
      'z', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '=');
   PGPchr64 : Array[1..64] Of char = ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'K', 'L', 'M',
      'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b',
'c', 'd', 'e', 'f',
      'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'x', 'y',
      'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/');


{$H+}


Procedure zeronetochar8(Var g : char; Const x : String);
Var
   i : Integer;
   b : byte;
Begin
   b := 0;
   For i := 1 To 8 Do
   Begin
      If copy(x, i, 1) = '1' Then
         b := b Or (1 Shl (8 - I));
   End;
   g := chr(b);
End;


Procedure zeronetochar6(Var g : integer; Const x : String);
Var
   I : Integer;
Begin
   G := 0;
   For I := 1 To Length(X) Do
   Begin
      If I > 6 Then
         Break;
      If X[I] <> '0' Then
         G := G Or (1 Shl (6 - I));
   End;
   Inc(G);
End;


Procedure initialize8(Var trans : Array Of String);
```

```
Var
   c1, c2, c3, c4, c5, c6, c7, c8 : integer;
   x : String;
   g : char;
Begin
   For c1 := 0 To 1 Do
      For c2 := 0 To 1 Do
         For c3 := 0 To 1 Do
            For c4 := 0 To 1 Do
               For c5 := 0 To 1 Do
                  For c6 := 0 To 1 Do
                     For c7 := 0 To 1 Do
                        For c8 := 0 To 1 Do
                        Begin
                           x := '';
                           x := inttostr(c1) + inttostr(c2) + inttostr(c3) +
inttostr(c4) + inttostr(c5) + inttostr(c6) + inttostr(c7) + inttostr(c8);
                           zeronetochar8(g, x);
                           trans[ord(g)] := x;
                        End;
End;


Procedure initialize6(Var trans : Array Of String);
Var
   c1, c2, c3, c4, c5, c6 : integer;
   x : String;
   g : integer;
Begin
   For c1 := 0 To 1 Do
      For c2 := 0 To 1 Do
         For c3 := 0 To 1 Do
            For c4 := 0 To 1 Do
               For c5 := 0 To 1 Do
                  For c6 := 0 To 1 Do
                  Begin
                     x := '';
                     x := inttostr(c1) + inttostr(c2) + inttostr(c3) +
inttostr(c4) + inttostr(c5) + inttostr(c6);
                     zeronetochar6(g, x);
                     trans[ord(chr64[g])] := x;
                  End;
End;

Procedure initialize6PGP(Var trans : Array Of String);
Var
   c1, c2, c3, c4, c5, c6 : integer;
   x : String;
   g : integer;
Begin
   For c1 := 0 To 1 Do
      For c2 := 0 To 1 Do
         For c3 := 0 To 1 Do
            For c4 := 0 To 1 Do
               For c5 := 0 To 1 Do
                  For c6 := 0 To 1 Do
                  Begin
                     x := '';
                     x := inttostr(c1) + inttostr(c2) + inttostr(c3) +
inttostr(c4) + inttostr(c5) + inttostr(c6);
                     zeronetochar6(g, x);
                     trans[ord(PGPchr64[g])] := x;
                  End;
End;


// перетворюємо строки довжиною 256 біт у 64 біта

Procedure ConvertBase256to64(Const str256 : String; Var str64 : String);
```

```
Var
   temp : String;
   trans : Array[0..255] Of String;
   i, len6 : longint;
   g : integer;
Begin
   initialize8(trans);
   temp := '';
   For i := 1 To length(str256) Do temp := temp + trans[ord(str256[i])];
   While (length(temp) Mod 6) <> 0 Do temp := temp + '0';
   len6 := length(temp) Div 6;
   str64 := '';
   For i := 1 To len6 Do
   Begin
      zeronetochar6(g, copy(temp, 1, 6));
      str64 := str64 + chr64[g];
      delete(temp, 1, 6);
   End;
End;

// перетворюємо строки довжиною 64 біт у 256 біта

Procedure ConvertBase64to256(Const str64 : String; Var str256 : String);
Var
   temp : String;
   trans : Array[0..255] Of String;
   i, len8 : longint;
   g : char;
Begin
   initialize6(trans);
   temp := '';
   For i := 1 To length(str64) Do temp := temp + trans[ord(str64[i])];
   str256 := '';
   len8 := length(temp) Div 8;
   For i := 1 To len8 Do
   Begin
      zeronetochar8(g, copy(temp, 1, 8));
      str256 := str256 + g;
      delete(temp, 1, 8);
   End;
End;


// перетворюємо строки довжиною 256 біт у 2 біта

Procedure ConvertBase256to2(Const str256 : String; Var str2 : String);
Var
   trans : Array[0..255] Of String;
   i : longint;
Begin
   str2 := '';
   initialize8(trans);
   For i := 1 To length(str256) Do str2 := str2 + trans[ord(str256[i])];
End;

// перетворюємо строки довжиною 64 біт у 2 біта

Procedure ConvertBase64to2(Const str64 : String; Var str2 : String);
Var
   trans : Array[0..255] Of String;
   i : longint;
Begin
   str2 := '';
   initialize6(trans);
   For i := 1 To length(str64) Do str2 := str2 + trans[ord(str64[i])];
End;

// перетворюємо строки довжиною 2 біт у 256 біт
```

```
Procedure ConvertBase2to256(str2 : String; Var str256 : String);
Var
   i, len8 : longint;
   g : char;
Begin
   str256 := '';
   While (length(str2) Mod 8) <> 0 Do str2 := '0' + str2;
   len8 := length(str2) Div 8;
   For i := 1 To len8 Do
   Begin
      zeronetochar8(g, copy(str2, 1, 8));
      str256 := str256 + g;
      delete(str2, 1, 8);
   End;
End;

// перетворюємо строки довжиною 2 біта у 64 біта

Procedure ConvertBase2to64(str2 : String; Var str64 : String);
Var
   i, len6 : longint;
   g : integer;
Begin
   str64 := '';
   While (length(str2) Mod 6) <> 0 Do str2 := '0' + str2;
   len6 := length(str2) Div 6;
   For i := 1 To len6 Do
   Begin
      zeronetochar6(g, copy(str2, 1, 6));
      str64 := str64 + chr64[g];
      delete(str2, 1, 6);
   End;
End;


// перетворюємо строки довжиною 256 біт у 16 біта

Procedure ConvertBase256StringToHexString(Str256 : String; Var HexStr : String);
Var
   i : longint;
   b : byte;
Begin
   HexStr := '';
   For i := 1 To length(str256) Do
   Begin
      b := ord(str256[i]);
      If (b Shr 4) < 10 Then HexStr := HexStr + chr(48 + (b Shr 4))
      Else HexStr := HexStr + chr(55 + (b Shr 4));
      If (b And 15) < 10 Then HexStr := HexStr + chr(48 + (b And 15))
      Else HexStr := HexStr + chr(55 + (b And 15));
   End;
End;



// перетворюємо строки довжиною 16 біт у 256 біт

Procedure ConvertHexStringToBase256String(HexStr : String; Var Str256 : String);
Var
   i : longint;
   b, h1, h2 : byte;
Begin
   Str256 := '';
   For i := 1 To (length(Hexstr) Div 2) Do
   Begin
      h2 := ord(HexStr[2 * i]);
      h1 := ord(HexStr[2 * i - 1]);
      If h1 < 58 Then b := ((h1 - 48) Shl 4) Else b := ((h1 - 55) Shl 4);
```

```
      If h2 < 58 Then b := (b Or (h2 - 48)) Else b := (b Or (h2 - 55));
      Str256 := Str256 + chr(b);
   End;
End;


// перетворюємо строки довжиною 256 біт у 64 біта для PGP

Procedure PGPConvertBase256to64(Var str256, str64 : String);
Var
   temp, x, a : String;
   i, len6 : longint;
   g : integer;
   trans : Array[0..255] Of String;
Begin
   initialize8(trans);
   temp := '';
   For i := 1 To length(str256) Do temp := temp + trans[ord(str256[i])];
   If (length(temp) Mod 6) = 0 Then a := '' Else
      If (length(temp) Mod 6) = 4 Then
      Begin
         temp := temp + '00';
         a := '='
      End
      Else
      Begin
         temp := temp + '0000';
         a := '=='
      End;
   str64 := '';
   len6 := length(temp) Div 6;
   For i := 1 To len6 Do
   Begin
      x := copy(temp, 1, 6);
      zeronetochar6(g, x);
      str64 := str64 + PGPchr64[g];
      delete(temp, 1, 6);
   End;
   str64 := str64 + a;
End;

// перетворюємо строки довжиною 64 біт у 256 біт для PGP

Procedure PGPConvertBase64to256(str64 : String; Var str256 : String);
Var
   temp, x : String;
   i, j, len8 : longint;
   g : char;
   trans : Array[0..255] Of String;
Begin
   initialize6PGP(trans);
   temp := '';
   str256 := '';
   If str64[length(str64) - 1] = '=' Then j := 2 Else
      If str64[length(str64)] = '=' Then j := 1 Else j := 0;
   For i := 1 To (length(str64) - j) Do temp := temp + trans[ord(str64[i])];
   If j <> 0 Then delete(temp, length(temp) - 2 * j + 1, 2 * j);
   len8 := length(temp) Div 8;
   For i := 1 To len8 Do
   Begin
      x := copy(temp, 1, 8);
      zeronetochar8(g, x);
      str256 := str256 + g;
      delete(temp, 1, 8);
   End;
End;

// перетворюємо строки довжиною 64 біт у 2 біта для PGP
```

```
Procedure PGPConvertBase64to2(str64 : String; Var str2 : String);
Var
   i, j : longint;
   trans : Array[0..255] Of String;
Begin
   str2 := '';
   initialize6(trans);
   If str64[length(str64) - 1] = '=' Then j := 2 Else
      If str64[length(str64)] = '=' Then j := 1 Else j := 0;
   For i := 1 To (length(str64) - j) Do str2 := str2 + trans[ord(str64[i])];
   delete(str2, length(str2) - 2 * j + 1, 2 * j);
End;


// перетворюємо строки довжиною 10 біт у  FGInt

Procedure Base10StringToFGInt(Base10 : String; Var FGInt : TFGInt);
Var
   i, size : longint;
   j : int64;
   S : String;
   sign : TSign;

   Procedure GIntDivByIntBis1(Var GInt : TFGInt; by : int64; Var modres :
int64);
   Var
      i, size : longint;
      rest : int64;
   Begin
      size := GInt.Number[0];
      modres := 0;
      For i := size Downto 1 Do
      Begin
         modres := modres * 1000000000;
         rest := modres + GInt.Number[i];
         GInt.Number[i] := rest Div by;
         modres := rest Mod by;
      End;
      While (GInt.Number[size] = 0) And (size > 1) Do size := size - 1;
      If size <> GInt.Number[0] Then
      Begin
         SetLength(GInt.Number, size + 1);
         GInt.Number[0] := size;
      End;
   End;

Begin
   While (Not (Base10[1] In ['-', '0'..'9'])) And (length(Base10) > 1) Do
      delete(Base10, 1, 1);
   If copy(Base10, 1, 1) = '-' Then
   Begin
      Sign := negative;
      delete(Base10, 1, 1);
   End
   Else Sign := positive;
   While (length(Base10) > 1) And (copy(Base10, 1, 1) = '0') Do delete(Base10,
1, 1);
   size := length(Base10) Div 9;
   If (length(Base10) Mod 9) <> 0 Then size := size + 1;
   SetLength(FGInt.Number, size + 1);
   FGInt.Number[0] := size;
   For i := 1 To size - 1 Do
   Begin
      FGInt.Number[i] := StrToInt(copy(Base10, length(Base10) - 8, 9));
      delete(Base10, length(Base10) - 8, 9);
   End;
   FGInt.Number[size] := StrToInt(Base10);

   S := '';
```

```
   While (FGInt.Number[0] <> 1) Or (FGInt.Number[1] <> 0) Do
   Begin
      GIntDivByIntBis1(FGInt, 2, j);
      S := inttostr(j) + S;
   End;
   S := '0' + S;
   FGIntDestroy(FGInt);
   Base2StringToFGInt(S, FGInt);
   FGInt.Sign := sign;
End;


// перетворюємо  FGInt в рядок 10 біт

Procedure FGIntToBase10String(Const FGInt : TFGInt; Var Base10 : String);
Var
   S : String;
   j : int64;
   temp : TFGInt;
Begin
   FGIntCopy(FGInt, temp);
   Base10 := '';
   While (temp.Number[0] > 1) Or (temp.Number[1] > 0) Do
   Begin
      FGIntDivByIntBis(temp, 1000000000, j);
      S := IntToStr(j);
      While Length(S) < 9 Do S := '0' + S;
      Base10 := S + Base10;
   End;
   Base10 := '0' + Base10;
   While (length(Base10) > 1) And (Base10[1] = '0') Do delete(Base10, 1, 1);
   If FGInt.Sign = negative Then Base10 := '-' + Base10;
End;



// Видаляємо FGInt для звільнення пам'яті

Procedure FGIntDestroy(Var FGInt : TFGInt);
Begin
   FGInt.Number := Nil;
End;



Function FGIntCompareAbs(Const FGInt1, FGInt2 : TFGInt) : TCompare;
Var
   size1, size2, i : longint;
Begin
   FGIntCompareAbs := Er;
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   If size1 > size2 Then FGIntCompareAbs := Lt Else
      If size1 < size2 Then FGIntCompareAbs := St Else
      Begin
         i := size2;
         While (FGInt1.Number[i] = FGInt2.Number[i]) And (i > 1) Do i := i - 1;
         If FGInt1.Number[i] = FGInt2.Number[i] Then FGIntCompareAbs := Eq Else
            If FGInt1.Number[i] < FGInt2.Number[i] Then FGIntCompareAbs := St
Else
               If FGInt1.Number[i] > FGInt2.Number[i] Then FGIntCompareAbs :=
Lt;
      End;
End;


// Додаємо 2 FGInts, FGInt1 + FGInt2 = Sum

Procedure FGIntAdd(Const FGInt1, FGInt2 : TFGInt; Var Sum : TFGInt);
```

```
Var
   i, size1, size2, size : longint;
   rest : integer;
   Trest : int64;
Begin
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   If size1 < size2 Then FGIntAdd(FGInt2, FGInt1, Sum) Else
   Begin
      If FGInt1.Sign = FGInt2.Sign Then
      Begin
         Sum.Sign := FGInt1.Sign;
         SetLength(Sum.Number, size1 + 2);
         rest := 0;
         For i := 1 To size2 Do
         Begin
            Trest := FGInt1.Number[i] + FGInt2.Number[i] + rest;
            Sum.Number[i] := Trest And 2147483647;
            rest := Trest Shr 31;
         End;
         For i := (size2 + 1) To size1 Do
         Begin
            Trest := FGInt1.Number[i] + rest;
            Sum.Number[i] := Trest And 2147483647;
            rest := Trest Shr 31;
         End;
         size := size1 + 1;
         Sum.Number[0] := size;
         Sum.Number[size] := rest;
         While (Sum.Number[size] = 0) And (size > 1) Do size := size - 1;
         If Sum.Number[0] > size Then SetLength(Sum.Number, size + 1);
         Sum.Number[0] := size;
      End
      Else
      Begin
         If FGIntCompareAbs(FGInt2, FGInt1) = Lt Then FGIntAdd(FGInt2, FGInt1,
Sum)
         Else
         Begin
            SetLength(Sum.Number, size1 + 1);
            rest := 0;
            For i := 1 To size2 Do
            Begin
               Trest := 2147483648 + FGInt1.Number[i] - FGInt2.Number[i] + rest;
               Sum.Number[i] := Trest And 2147483647;
               If (Trest > 2147483647) Then rest := 0 Else rest := -1;
            End;
            For i := (size2 + 1) To size1 Do
            Begin
               Trest := 2147483648 + FGInt1.Number[i] + rest;
               Sum.Number[i] := Trest And 2147483647;
               If (Trest > 2147483647) Then rest := 0 Else rest := -1;
            End;
            size := size1;
            While (Sum.Number[size] = 0) And (size > 1) Do size := size - 1;
            If size < size1 Then
            Begin
               SetLength(Sum.Number, size + 1);
            End;
            Sum.Number[0] := size;
            Sum.Sign := FGInt1.Sign;
         End;
      End;
   End;
End;


Procedure FGIntChangeSign(Var FGInt : TFGInt);
Begin
```

```
   If FGInt.Sign = negative Then FGInt.Sign := positive Else FGInt.Sign :=
negative;
End;


// Віднімаємо 2 FGInts, FGInt1 - FGInt2 = dif

Procedure FGIntSub(Var FGInt1, FGInt2, dif : TFGInt);
Begin
   FGIntChangeSign(FGInt2);
   FGIntAdd(FGInt1, FGInt2, dif);
   FGIntChangeSign(FGInt2);
End;


// Перемножуємоа FGInt з цілим, FGInt * by = res, by < 1000000000

Procedure FGIntMulByInt(Const FGInt : TFGInt; Var res : TFGInt; by : int64);
Var
   i, size : longint;
   Trest, rest : int64;
Begin
   size := FGInt.Number[0];
   SetLength(res.Number, size + 2);
   rest := 0;
   For i := 1 To size Do
   Begin
      Trest := FGInt.Number[i] * by + rest;
      res.Number[i] := Trest And 2147483647;
      rest := Trest Shr 31;
   End;
   If rest <> 0 Then
   Begin
      size := size + 1;
      Res.Number[size] := rest;
   End
   Else SetLength(Res.Number, size + 1);
   Res.Number[0] := size;
   Res.Sign := FGInt.Sign;
End;


// перемножуємо FGInt з цілим, FGInt * by = res, by < 1000000000

Procedure FGIntMulByIntbis(Var FGInt : TFGInt; by : int64);
Var
   i, size : longint;
   Trest, rest : int64;
Begin
   size := FGInt.Number[0];
   SetLength(FGInt.Number, size + 2);
   rest := 0;
   For i := 1 To size Do
   Begin
      Trest := FGInt.Number[i] * by + rest;
      FGInt.Number[i] := Trest And 2147483647;
      rest := Trest Shr 31;
   End;
   If rest <> 0 Then
   Begin
      size := size + 1;
      FGInt.Number[size] := rest;
   End
   Else SetLength(FGInt.Number, size + 1);
   FGInt.Number[0] := size;
End;


// ділимо FGInt на ціле, FGInt = res * by + modres
```

```
Procedure FGIntDivByInt(Const FGInt : TFGInt; Var res : TFGInt; by : int64; Var
modres : int64);
Var
   i, size : longint;
   rest : int64;
Begin
   size := FGInt.Number[0];
   SetLength(res.Number, size + 1);
   modres := 0;
   For i := size Downto 1 Do
   Begin
      modres := modres Shl 31;
      rest := modres Or FGInt.Number[i];
      res.Number[i] := rest Div by;
      modres := rest Mod by;
   End;
   While (res.Number[size] = 0) And (size > 1) Do size := size - 1;
   SetLength(res.Number, size + 1);
   res.Number[0] := size;
   Res.Sign := FGInt.Sign;
End;


// ділимо FGInt на ціле, FGInt = FGInt * by + modres

Procedure FGIntDivByIntBis(Var FGInt : TFGInt; by : int64; Var modres : int64);
Var
   i, size : longint;
   rest : int64;
Begin
   size := FGInt.Number[0];
   modres := 0;
   For i := size Downto 1 Do
   Begin
      modres := modres Shl 31;
      rest := modres Or FGInt.Number[i];
      FGInt.Number[i] := rest Div by;
      modres := rest Mod by;
   End;
   While (FGInt.Number[size] = 0) And (size > 1) Do size := size - 1;
   If size <> FGInt.Number[0] Then
   Begin
      SetLength(FGInt.Number, size + 1);
      FGInt.Number[0] := size;
   End;
End;


// Беремо FGInt по модулю цілого числа, FGInt mod by = modres

Procedure FGIntModByInt(Const FGInt : TFGInt; by : int64; Var modres : int64);
Var
   i, size : longint;
   rest : int64;
Begin
   size := FGInt.Number[0];
   modres := 0;
   For i := size Downto 1 Do
   Begin
      modres := modres Shl 31;
      rest := modres + FGInt.Number[i];
      modres := rest Mod by;
   End;
End;


// повертаємо FGInt по модулю
```

```
Procedure FGIntAbs(Var FGInt : TFGInt);
Begin
   FGInt.Sign := positive;
End;


// Копіюємо FGInt1 в FGInt2

Procedure FGIntCopy(Const FGInt1 : TFGInt; Var FGInt2 : TFGInt);
Begin
   FGInt2.Sign := FGInt1.Sign;
   FGInt2.Number := Nil;
   FGInt2.Number := Copy(FGInt1.Number, 0, FGInt1.Number[0] + 1);
End;


// Зрушуємо FGInt уліво на 2 біта FGInt = FGInt * 2

Procedure FGIntShiftLeft(Var FGInt : TFGInt);
Var
   l, m : int64;
   i, size : longint;
Begin
   size := FGInt.Number[0];
   l := 0;
   For i := 1 To Size Do
   Begin
      m := FGInt.Number[i] Shr 30;
      FGInt.Number[i] := ((FGInt.Number[i] Shl 1) Or l) And 2147483647;
      l := m;
   End;
   If l <> 0 Then
   Begin
      setlength(FGInt.Number, size + 2);
      FGInt.Number[size + 1] := l;
      FGInt.Number[0] := size + 1;
   End;
End;


// Зрушуємо FGInt вправо на 2 біта, FGInt = FGInt div 2

Procedure FGIntShiftRight(Var FGInt : TFGInt);
Var
   l, m : int64;
   i, size : longint;
Begin
   size := FGInt.Number[0];
   l := 0;
   For i := size Downto 1 Do
   Begin
      m := FGInt.Number[i] And 1;
      FGInt.Number[i] := (FGInt.Number[i] Shr 1) Or l;
      l := m Shl 30;
   End;
   If (FGInt.Number[size] = 0) And (size > 1) Then
   Begin
      setlength(FGInt.Number, size);
      FGInt.Number[0] := size - 1;
   End;
End;


// FGInt = FGInt / 2147483648

Procedure FGIntShiftRightBy31(Var FGInt : TFGInt);
Var
   size : longint;
Begin
```

```
   size := FGInt.Number[0];
   If size > 1 Then
   Begin
      FGInt.Number := Copy(FGInt.Number, 1, Size);
      FGInt.Number[0] := size - 1;
   End
   Else FGInt.Number[1] := 0;
End;


// FGInt1 = FGInt1 + FGInt2, FGInt1 > FGInt2

Procedure FGIntAddBis(Var FGInt1 : TFGInt; Const FGInt2 : TFGInt);
Var
   i, size1, size2 : longint;
   rest : integer;
   Trest : int64;
Begin
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   rest := 0;
   For i := 1 To size2 Do
   Begin
      Trest := FGInt1.Number[i] + FGInt2.Number[i] + rest;
      rest := Trest Shr 31;
      FGInt1.Number[i] := Trest And 2147483647;
   End;
   For i := size2 + 1 To size1 Do
   Begin
      Trest := FGInt1.Number[i] + rest;
      rest := Trest Shr 31;
      FGInt1.Number[i] := Trest And 2147483647;
   End;
   If rest <> 0 Then
   Begin
      SetLength(FGInt1.Number, size1 + 2);
      FGInt1.Number[0] := size1 + 1;
      FGInt1.Number[size1 + 1] := rest;
   End;
End;


// FGInt1 = FGInt1 - FGInt2, використовується тільки якщо 0 < FGInt2 < FGInt1

Procedure FGIntSubBis(Var FGInt1 : TFGInt; Const FGInt2 : TFGInt);
Var
   i, size1, size2 : longint;
   rest : integer;
   Trest : int64;
Begin
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   rest := 0;
   For i := 1 To size2 Do
   Begin
      Trest := 2147483648 + FGInt1.Number[i] - FGInt2.Number[i] + rest;
      If (Trest > 2147483647) Then rest := 0 Else rest := -1;
      FGInt1.Number[i] := Trest And 2147483647;
   End;
   For i := size2 + 1 To size1 Do
   Begin
      Trest := 2147483648 + FGInt1.Number[i] + rest;
      If (Trest > 2147483647) Then rest := 0 Else rest := -1;
      FGInt1.Number[i] := Trest And 2147483647;
   End;
   i := size1;
   While (FGInt1.Number[i] = 0) And (i > 1) Do i := i - 1;
   If i < size1 Then
   Begin
```

```
      SetLength(FGInt1.Number, i + 1);
      FGInt1.Number[0] := i;
   End;
End;


// Перемножуємо 2 FGInts, FGInt1 * FGInt2 = Prod

Procedure FGIntMul(Const FGInt1, FGInt2 : TFGInt; Var Prod : TFGInt);
Var
   i, j, size, size1, size2 : longint;
   rest, Trest : int64;
Begin
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   size := size1 + size2;
   SetLength(Prod.Number, size + 1);
   For i := 1 To size Do Prod.Number[i] := 0;

   For i := 1 To size2 Do
   Begin
      rest := 0;
      For j := 1 To size1 Do
      Begin
         Trest := Prod.Number[j + i - 1] + FGInt1.Number[j] * FGInt2.Number[i] +
rest;
         Prod.Number[j + i - 1] := Trest And 2147483647;
         rest := Trest Shr 31;
      End;
      Prod.Number[i + size1] := rest;
   End;

   Prod.Number[0] := size;
   While (Prod.Number[size] = 0) And (size > 1) Do size := size - 1;
   If size < Prod.Number[0] Then
   Begin
      SetLength(Prod.Number, size + 1);
      Prod.Number[0] := size;
   End;
   If FGInt1.Sign = FGInt2.Sign Then Prod.Sign := Positive Else prod.Sign :=
negative;
End;


// Підводимо в квадрат FGInt, FGIntI = Square

Procedure FGIntSquare(Const FGInt : TFGInt; Var Square : TFGInt);
Var
   size, size1, i, j : longint;
   rest, Trest : int64;
Begin
   size1 := FGInt.Number[0];
   size := 2 * size1;
   SetLength(Square.Number, size + 1);
   Square.Number[0] := size;
   For i := 1 To size Do Square.Number[i] := 0;
   For i := 1 To size1 Do
   Begin
      Trest := Square.Number[2 * i - 1] + FGInt.Number[i] * FGInt.Number[i];
      Square.Number[2 * i - 1] := Trest And 2147483647;
      rest := Trest Shr 31;
      For j := i + 1 To size1 Do
      Begin
         Trest := Square.Number[i + j - 1] + 2 * FGInt.Number[i] *
FGInt.Number[j] + rest;
         Square.Number[i + j - 1] := Trest And 2147483647;
         rest := Trest Shr 31;
      End;
      Square.Number[i + size1] := rest;
```

```
   End;
   Square.Sign := positive;
   While (Square.Number[size] = 0) And (size > 1) Do size := size - 1;
   If size < 2 * size1 Then
   Begin
      SetLength(Square.Number, size + 1);
      Square.Number[0] := size;
   End;
End;


// Перетворюємо FGInt у бінарну рядок

Procedure FGIntToBase2String(Const FGInt : TFGInt; Var S : String);
Var
   i : longint;
   j : integer;
Begin
   S := '';
   For i := 1 To FGInt.Number[0] Do
   Begin
      For j := 0 To 30 Do S := inttostr(1 And (FGInt.Number[i] Shr j)) + S;
   End;
   While (length(S) > 1) And (S[1] = '0') Do delete(S, 1, 1);
   If S = '' Then S := '0';
End;


Procedure Base2StringToFGInt(S : String; Var FGInt : TFGInt);
Var
   i, j, size : longint;
Begin
   while (S[1]='0') and (length(S)>1) do delete(S,1,1);
   size := length(S) Div 31;
   If (length(S) Mod 31) <> 0 Then size := size + 1;
   SetLength(FGInt.Number, size + 1);
   FGInt.Number[0] := size;
   j := 1;
   FGInt.Number[j] := 0;
   i := 0;
   While length(S) > 0 Do
   Begin
      If S[length(S)] = '1' Then
         FGInt.Number[j] := FGInt.Number[j] Or (1 Shl i);
      i := i + 1;
      If i = 31 Then
      Begin
         i := 0;
         j := j + 1;
         If j <= size Then FGInt.Number[j] := 0;
      End;
      delete(S, length(S), 1);
   End;
   FGInt.Sign := positive;
End;


// перетворюємо FGInt у рядок 256 біт

Procedure FGIntToBase256String(Const FGInt : TFGInt; Var str256 : String);
Var
   temp1 : String;
   i, len8 : longint;
   g : char;
Begin
   FGIntToBase2String(FGInt, temp1);
   While (length(temp1) Mod 8) <> 0 Do temp1 := '0' + temp1;
   len8 := length(temp1) Div 8;
   str256 := '';
```

```
   For i := 1 To len8 Do
   Begin
      zeronetochar8(g, copy(temp1, 1, 8));
      str256 := str256 + g;
      delete(temp1, 1, 8);
   End;
End;


Procedure Base256StringToFGInt(str256 : String; Var FGInt : TFGInt);
Var
   temp1 : String;
   i : longint;
   trans : Array[0..255] Of String;
Begin
   temp1 := '';
   initialize8(trans);
   For i := 1 To length(str256) Do temp1 := temp1 + trans[ord(str256[i])];
   While (temp1[1] = '0') And (temp1 <> '0') Do delete(temp1, 1, 1);
   Base2StringToFGInt(temp1, FGInt);
End;

// Перетворюємо MPI (Multiple Precision Integer, для PGP) у FGInt

Procedure PGPMPIToFGInt(PGPMPI : String; Var FGInt : TFGInt);
Var
   temp : String;
Begin
   temp := PGPMPI;
   delete(temp, 1, 2);
   Base256StringToFGInt(temp, FGInt);
End;


Procedure FGIntToPGPMPI(FGInt : TFGInt; Var PGPMPI : String);
Var
   len, i : word;
   c : char;
   b : byte;
Begin
   FGIntToBase256String(FGInt, PGPMPI);
   len := length(PGPMPI) * 8;
   c := PGPMPI[1];
   For i := 7 Downto 0 Do If (ord(c) Shr i) = 0 Then len := len - 1 Else break;
   b := len Mod 256;
   PGPMPI := chr(b) + PGPMPI;
   b := len Div 256;
   PGPMPI := chr(b) + PGPMPI;
End;


// Піднімаємо у ступінь FGInt, FGInt^exp = res

Procedure FGIntExp(Const FGInt, exp : TFGInt; Var res : TFGInt);
Var
   temp2, temp3 : TFGInt;
   S : String;
   i : longint;
Begin
   FGIntToBase2String(exp, S);
   If S[length(S)] = '0' Then Base10StringToFGInt('1', res) Else
FGIntCopy(FGInt, res);
   FGIntCopy(FGInt, temp2);
   If length(S) > 1 Then
      For i := (length(S) - 1) Downto 1 Do
      Begin
         FGIntSquare(temp2, temp3);
         FGIntCopy(temp3, temp2);
         If S[i] = '1' Then
```

```
         Begin
            FGIntMul(res, temp2, temp3);
            FGIntCopy(temp3, res);
         End;
      End;
End;


// Розраховуємо FGInt! = FGInt * (FGInt - 1) * (FGInt - 2) * ... * 3 * 2 * 1

Procedure FGIntFac(Const FGInt : TFGInt; Var res : TFGInt);
Var
   one, temp, temp1 : TFGInt;
Begin
   FGIntCopy(FGInt, temp);
   Base10StringToFGInt('1', res);
   Base10StringToFGInt('1', one);

   While Not (FGIntCompareAbs(temp, one) = Eq) Do
   Begin
      FGIntMul(temp, res, temp1);
      FGIntCopy(temp1, res);
      FGIntSubBis(temp, one);
   End;

   FGIntDestroy(one);
   FGIntDestroy(temp);
End;


// FGInt = FGInt * 2147483648

Procedure FGIntShiftLeftBy31(Var FGInt : TFGInt);
Var
   f1, f2 : int64;
   i, size : longint;
Begin
   size := FGInt.Number[0];
   SetLength(FGInt.Number, size + 2);
   f1 := 0;
   For i := 1 To (size + 1) Do
   Begin
      f2 := FGInt.Number[i];
      FGInt.Number[i] := f1;
      f1 := f2;
   End;
   FGInt.Number[0] := size + 1;
End;


// Ділимо 2 FGInts, FGInt1 = FGInt2 * QFGInt + MFGInt, MFGInt позитивне

Procedure FGIntDivMod(Var FGInt1, FGInt2, QFGInt, MFGInt : TFGInt);
Var
   one, zero, temp1, temp2 : TFGInt;
   s1, s2 : TSign;
   i, j, s, t : longint;
Begin
   s1 := FGInt1.Sign;
   s2 := FGInt2.Sign;
   FGIntAbs(FGInt1);
   FGIntAbs(FGInt2);
   FGIntCopy(FGInt1, MFGInt);
   FGIntCopy(FGInt2, temp1);

   If FGIntCompareAbs(FGInt1, FGInt2) <> St Then
   Begin
      s := FGInt1.Number[0] - FGInt2.Number[0];
      setlength(QFGInt.Number, s + 2);
```

```
      QFGInt.Number[0] := s + 1;
      For t := 1 To s Do
      Begin
         FGIntShiftLeftBy31(temp1);
         QFGInt.Number[t] := 0;
      End;
      j := s + 1;
      QFGInt.Number[j] := 0;
      While FGIntCompareAbs(MFGInt, FGInt2) <> St Do
      Begin
         While FGIntCompareAbs(MFGInt, temp1) <> St Do
         Begin
            If MFGInt.Number[0] > temp1.Number[0] Then
               i := (2147483648 * MFGInt.Number[MFGInt.Number[0]] +
MFGInt.Number[MFGInt.Number[0] - 1]) Div (temp1.Number[temp1.Number[0]] + 1)
            Else i := MFGInt.Number[MFGInt.Number[0]] Div
(temp1.Number[temp1.Number[0]] + 1);
            If (i <> 0) Then
            Begin
               FGIntCopy(temp1, temp2);
               FGIntMulByIntBis(temp2, i);
               FGIntSubBis(MFGInt, temp2);
               QFGInt.Number[j] := QFGInt.Number[j] + i;
               If FGIntCompareAbs(MFGInt, temp2) <> St Then
               Begin
                  QFGInt.Number[j] := QFGInt.Number[j] + i;
                  FGIntSubBis(MFGInt, temp2);
               End;
            End Else
            Begin
               QFGInt.Number[j] := QFGInt.Number[j] + 1;
               FGIntSubBis(MFGInt, temp1);
            End;
         End;
         If MFGInt.Number[0] <= temp1.Number[0] Then
            If FGIntCompareAbs(temp1, FGInt2) <> Eq Then
            Begin
               FGIntShiftRightBy31(temp1);
               j := j - 1;
            End;
      End;
   End
   Else Base10StringToFGInt('0', QFGInt);
   s := QFGInt.Number[0];
   While (s > 1) And (QFGInt.Number[s] = 0) Do s := s - 1;
   If s < QFGInt.Number[0] Then
   Begin
      setlength(QFGInt.Number, s + 1);
      QFGInt.Number[0] := s;
   End;
   QFGInt.Sign := positive;

   FGIntDestroy(temp1);
   Base10StringToFGInt('0', zero);
   Base10StringToFGInt('1', one);
   If s1 = negative Then
   Begin
      If FGIntCompareAbs(MFGInt, zero) <> Eq Then
      Begin
         FGIntadd(QFGInt, one, temp1);
         FGIntCopy(temp1, QFGInt);
         FGIntDestroy(temp1);
         FGIntsub(FGInt2, MFGInt, temp1);
         FGIntCopy(temp1, MFGInt);
      End;
      If s2 = positive Then QFGInt.Sign := negative;
   End
   Else QFGInt.Sign := s2;
   FGIntDestroy(one);
```

```
   FGIntDestroy(zero);

   FGInt1.Sign := s1;
   FGInt2.Sign := s2;
End;


// Розраховуємо MFGInt

Procedure FGIntDiv(Var FGInt1, FGInt2, QFGInt : TFGInt);
Var
   one, zero, temp1, temp2, MFGInt : TFGInt;
   s1, s2 : TSign;
   i, j, s, t : longint;
Begin
   s1 := FGInt1.Sign;
   s2 := FGInt2.Sign;
   FGIntAbs(FGInt1);
   FGIntAbs(FGInt2);
   FGIntCopy(FGInt1, MFGInt);
   FGIntCopy(FGInt2, temp1);

   If FGIntCompareAbs(FGInt1, FGInt2) <> St Then
   Begin
      s := FGInt1.Number[0] - FGInt2.Number[0];
      setlength(QFGInt.Number, s + 2);
      QFGInt.Number[0] := s + 1;
      For t := 1 To s Do
      Begin
         FGIntShiftLeftBy31(temp1);
         QFGInt.Number[t] := 0;
      End;
      j := s + 1;
      QFGInt.Number[j] := 0;
      While FGIntCompareAbs(MFGInt, FGInt2) <> St Do
      Begin
         While FGIntCompareAbs(MFGInt, temp1) <> St Do
         Begin
            If MFGInt.Number[0] > temp1.Number[0] Then
               i := (2147483648 * MFGInt.Number[MFGInt.Number[0]] +
MFGInt.Number[MFGInt.Number[0] - 1]) Div (temp1.Number[temp1.Number[0]] + 1)
            Else i := MFGInt.Number[MFGInt.Number[0]] Div
(temp1.Number[temp1.Number[0]] + 1);
            If (i <> 0) Then
            Begin
               FGIntCopy(temp1, temp2);
               FGIntMulByIntBis(temp2, i);
               FGIntSubBis(MFGInt, temp2);
               QFGInt.Number[j] := QFGInt.Number[j] + i;
               If FGIntCompareAbs(MFGInt, temp2) <> St Then
               Begin
                  QFGInt.Number[j] := QFGInt.Number[j] + i;
                  FGIntSubBis(MFGInt, temp2);
               End;
            End Else
            Begin
               QFGInt.Number[j] := QFGInt.Number[j] + 1;
               FGIntSubBis(MFGInt, temp1);
            End;
         End;
         If MFGInt.Number[0] <= temp1.Number[0] Then
            If FGIntCompareAbs(temp1, FGInt2) <> Eq Then
            Begin
               FGIntShiftRightBy31(temp1);
               j := j - 1;
            End;
      End;
   End
   Else Base10StringToFGInt('0', QFGInt);
```

```
   s := QFGInt.Number[0];
   While (s > 1) And (QFGInt.Number[s] = 0) Do s := s - 1;
   If s < QFGInt.Number[0] Then
   Begin
      setlength(QFGInt.Number, s + 1);
      QFGInt.Number[0] := s;
   End;
   QFGInt.Sign := positive;

   FGIntDestroy(temp1);
   Base10StringToFGInt('0', zero);
   Base10StringToFGInt('1', one);
   If s1 = negative Then
   Begin
      If FGIntCompareAbs(MFGInt, zero) <> Eq Then
      Begin
         FGIntadd(QFGInt, one, temp1);
         FGIntCopy(temp1, QFGInt);
         FGIntDestroy(temp1);
         FGIntsub(FGInt2, MFGInt, temp1);
         FGIntCopy(temp1, MFGInt);
      End;
      If s2 = positive Then QFGInt.Sign := negative;
   End
   Else QFGInt.Sign := s2;
   FGIntDestroy(one);
   FGIntDestroy(zero);
   FGIntDestroy(MFGInt);

   FGInt1.Sign := s1;
   FGInt2.Sign := s2;
End;


// MFGInt = FGInt1 mod FGInt2

Procedure FGIntMod(Var FGInt1, FGInt2, MFGInt : TFGInt);
Var
   one, zero, temp1, temp2 : TFGInt;
   s1, s2 : TSign;
   i : int64;
   s, t : longint;
Begin
   s1 := FGInt1.Sign;
   s2 := FGInt2.Sign;
   FGIntAbs(FGInt1);
   FGIntAbs(FGInt2);
   FGIntCopy(FGInt1, MFGInt);
   FGIntCopy(FGInt2, temp1);

   If FGIntCompareAbs(FGInt1, FGInt2) <> St Then
   Begin
      s := FGInt1.Number[0] - FGInt2.Number[0];
      For t := 1 To s Do FGIntShiftLeftBy31(temp1);
      While FGIntCompareAbs(MFGInt, FGInt2) <> St Do
      Begin
         While FGIntCompareAbs(MFGInt, temp1) <> St Do
         Begin
            If MFGInt.Number[0] > temp1.Number[0] Then
               i := (2147483648 * MFGInt.Number[MFGInt.Number[0]] +
MFGInt.Number[MFGInt.Number[0] - 1]) Div (temp1.Number[temp1.Number[0]] + 1)
            Else i := MFGInt.Number[MFGInt.Number[0]] Div
(temp1.Number[temp1.Number[0]] + 1);
            If (i <> 0) Then
            Begin
               FGIntCopy(temp1, temp2);
               FGIntMulByIntBis(temp2, i);
               FGIntSubBis(MFGInt, temp2);
```

```
               If FGIntCompareAbs(MFGInt, temp2) <> St Then FGIntSubBis(MFGInt,
temp2);
            End Else FGIntSubBis(MFGInt, temp1);
//         If FGIntCompareAbs(MFGInt, temp1) <> St Then
FGIntSubBis(MFGInt,temp1);
         End;
         If MFGInt.Number[0] <= temp1.Number[0] Then
            If FGIntCompareAbs(temp1, FGInt2) <> Eq Then
FGIntShiftRightBy31(temp1);
      End;
   End;

   FGIntDestroy(temp1);
   Base10StringToFGInt('0', zero);
   Base10StringToFGInt('1', one);
   If s1 = negative Then
   Begin
      If FGIntCompareAbs(MFGInt, zero) <> Eq Then
      Begin
         FGIntSub(FGInt2, MFGInt, temp1);
         FGIntCopy(temp1, MFGInt);
      End;
   End;
   FGIntDestroy(one);
   FGIntDestroy(zero);

   FGInt1.Sign := s1;
   FGInt2.Sign := s2;
End;


// підводимо у квадрат FGInt за модулем  Modb, FGInt^2 mod Modb = FGIntSM

Procedure FGIntSquareMod(Var FGInt, Modb, FGIntSM : TFGInt);
Var
   temp : TFGInt;
Begin
   FGIntSquare(FGInt, temp);
   FGIntMod(temp, Modb, FGIntSM);
   FGIntDestroy(temp);
End;


// Додаємо 2 FGInts за модулем, (FGInt1 + FGInt2) mod base = FGIntres

Procedure FGIntAddMod(Var FGInt1, FGInt2, base, FGIntres : TFGInt);
Var
   temp : TFGInt;
Begin
   FGIntadd(FGInt1, FGInt2, temp);
   FGIntMod(temp, base, FGIntres);
   FGIntDestroy(temp);
End;


// Перемножуємо 2 FGInts за модулем, (FGInt1 * FGInt2) mod base = FGIntres

Procedure FGIntMulMod(Var FGInt1, FGInt2, base, FGIntres : TFGInt);
Var
   temp : TFGInt;
Begin
   FGIntMul(FGInt1, FGInt2, temp);
   FGIntMod(temp, base, FGIntres);
   FGIntDestroy(temp);
End;


// Підводимо у ступінь 2 FGInts за модулем, (FGInt1 ^ FGInt2) mod modb = res
```

```
Procedure FGIntModExp(Var FGInt, exp, modb, res : TFGInt);
Var
   temp2, temp3 : TFGInt;
   i : longint;
   S : String;
Begin
   If (Modb.Number[1] Mod 2) = 1 Then
   Begin
      FGIntMontgomeryModExp(FGInt, exp, modb, res);
      exit;
   End;
   FGIntToBase2String(exp, S);
   Base10StringToFGInt('1', res);
   FGIntcopy(FGInt, temp2);

   For i := length(S) Downto 1 Do
   Begin
      If S[i] = '1' Then
      Begin
         FGIntmulMod(res, temp2, modb, temp3);
         FGIntCopy(temp3, res);
      End;
      FGIntSquareMod(temp2, Modb, temp3);
      FGIntCopy(temp3, temp2);
   End;
   FGIntDestroy(temp2);
End;


// Процедура підводення у ступень за Монтгомері

Procedure FGIntModBis(Const FGInt : TFGInt; Var FGIntOut : TFGInt; b : longint;
head : int64);
Var
   i : longint;
Begin
   If b <= FGInt.Number[0] Then
   Begin
      FGIntOut.Number := Copy(FGInt.Number, 0, b + 1);
      FGIntOut.Number[b] := FGIntOut.Number[b] And head;
      i := b;
      While (FGIntOut.Number[i] = 0) And (i > 1) Do i := i - 1;
      If i < b Then SetLength(FGIntOut.Number, i + 1);
      FGIntOut.Number[0] := i;
      FGIntOut.Sign := positive;
   End Else FGIntCopy(FGInt, FGIntOut);
End;


Procedure FGIntMulModBis(Const FGInt1, FGInt2 : TFGInt; Var Prod : TFGInt; b :
longint; head : int64);
Var
   i, j, size, size1, size2, t : longint;
   rest, Trest : int64;
Begin
   size1 := FGInt1.Number[0];
   size2 := FGInt2.Number[0];
   size := min(b, size1 + size2);
   SetLength(Prod.Number, size + 1);
   For i := 1 To size Do Prod.Number[i] := 0;

   For i := 1 To size2 Do
   Begin
      rest := 0;
      t := min(size1, b - i + 1);
      For j := 1 To t Do
      Begin
         Trest := Prod.Number[j + i - 1] + FGInt1.Number[j] * FGInt2.Number[i] +
rest;
```

```
            Prod.Number[j + i - 1] := Trest And 2147483647;
            rest := Trest Shr 31;
         End;
         If (i + size1) <= b Then Prod.Number[i + size1] := rest;
      End;

      Prod.Number[0] := size;
      If size = b Then Prod.Number[b] := Prod.Number[b] And head;
      While (Prod.Number[size] = 0) And (size > 1) Do size := size - 1;
      If size < Prod.Number[0] Then
      Begin
         SetLength(Prod.Number, size + 1);
         Prod.Number[0] := size;
      End;
      If FGInt1.Sign = FGInt2.Sign Then Prod.Sign := Positive Else prod.Sign :=
negative;
End;


Procedure FGIntMontgomeryMod(Const GInt, base, baseInv : TFGInt; Var MGInt :
TFGInt; b : longint; head : int64);
Var
   m, temp, temp1 : TFGInt;
   r : int64;
Begin
   FGIntModBis(GInt, temp, b, head);
   FGIntMulModBis(temp, baseInv, m, b, head);
   FGIntMul(m, base, temp1);
   FGIntDestroy(temp);
   FGIntAdd(temp1, GInt, temp);
   FGIntDestroy(temp1);
   MGInt.Number := copy(temp.Number, b - 1, temp.Number[0] - b + 2);
   MGInt.Sign := positive;
   MGInt.Number[0] := temp.Number[0] - b + 1;
   FGIntDestroy(temp);
   If (head Shr 30) = 0 Then FGIntDivByIntBis(MGInt, head + 1, r)
   Else FGIntShiftRightBy31(MGInt);
   If FGIntCompareAbs(MGInt, base) <> St Then FGIntSubBis(MGInt, base);
   FGIntDestroy(temp);
   FGIntDestroy(m);
End;


Procedure FGIntMontgomeryModExp(Var FGInt, exp, modb, res : TFGInt);
Var
   temp2, temp3, baseInv, r : TFGInt;
   i, j, t, b : longint;
   S : String;
   head : int64;
Begin
   FGIntToBase2String(exp, S);
   t := modb.Number[0];
   b := t;

   If (modb.Number[t] Shr 30) = 1 Then t := t + 1;
   setlength(r.Number, t + 1);
   r.Number[0] := t;
   r.Sign := positive;
   For i := 1 To t Do r.Number[i] := 0;
   If t = modb.Number[0] Then
   Begin
      head := 2147483647;
      For j := 29 Downto 0 Do
      Begin
         head := head Shr 1;
         If (modb.Number[t] Shr j) = 1 Then
         Begin
            r.Number[t] := 1 Shl (j + 1);
            break;
```

```
         End;
      End;
   End
   Else
   Begin
      r.Number[t] := 1;
      head := 2147483647;
   End;

   FGIntModInv(modb, r, temp2);
   If temp2.Sign = negative Then FGIntCopy(temp2, BaseInv)
   Else
   Begin
      FGIntCopy(r, BaseInv);
      FGIntSubBis(BaseInv, temp2);
   End;
//   FGIntBezoutBachet(r, modb, temp2, BaseInv);
   FGIntAbs(BaseInv);
   FGIntDestroy(temp2);
   FGIntMod(r, modb, res);
   FGIntMulMod(FGInt, res, modb, temp2);
   FGIntDestroy(r);

   For i := length(S) Downto 1 Do
   Begin
      If S[i] = '1' Then
      Begin
         FGIntmul(res, temp2, temp3);
         FGIntDestroy(res);
         FGIntMontgomeryMod(temp3, modb, baseinv, res, b, head);
         FGIntDestroy(temp3);
      End;
      FGIntSquare(temp2, temp3);
      FGIntDestroy(temp2);
      FGIntMontgomeryMod(temp3, modb, baseinv, temp2, b, head);
      FGIntDestroy(temp3);
   End;
   FGIntDestroy(temp2);
   FGIntMontgomeryMod(res, modb, baseinv, temp3, b, head);
   FGIntCopy(temp3, res);
   FGIntDestroy(temp3);
   FGIntDestroy(baseinv);
End;


// розраховуємо найбільший загальний дільник 2 FGInts

Procedure FGIntGCD(Const FGInt1, FGInt2 : TFGInt; Var GCD : TFGInt);
Var
   k : TCompare;
   zero, temp1, temp2, temp3 : TFGInt;
Begin
   k := FGIntCompareAbs(FGInt1, FGInt2);
   If (k = Eq) Then FGIntCopy(FGInt1, GCD) Else
      If (k = St) Then FGIntGCD(FGInt2, FGInt1, GCD) Else
      Begin
         Base10StringToFGInt('0', zero);
         FGIntCopy(FGInt1, temp1);
         FGIntCopy(FGInt2, temp2);
         While (temp2.Number[0] <> 1) Or (temp2.Number[1] <> 0) Do
         Begin
            FGIntMod(temp1, temp2, temp3);
            FGIntCopy(temp2, temp1);
            FGIntCopy(temp3, temp2);
            FGIntDestroy(temp3);
         End;
         FGIntCopy(temp1, GCD);
         FGIntDestroy(temp2);
         FGIntDestroy(zero);
```

```
      End;
End;


// розраховуємо найменше загальне кратне 2 FGInts

Procedure FGIntLCM(Const FGInt1, FGInt2 : TFGInt; Var LCM : TFGInt);
Var
   temp1, temp2 : TFGInt;
Begin
   FGIntGCD(FGInt1, FGInt2, temp1);
   FGIntmul(FGInt1, FGInt2, temp2);
   FGIntdiv(temp2, temp1, LCM);
   FGIntDestroy(temp1);
   FGIntDestroy(temp2);
End;


// Знаходження взаємо простого FGInt до 9999 та зупинка коли знайдено таке
число, повертає ok=false

Procedure FGIntTrialDiv9999(Const FGInt : TFGInt; Var ok : boolean);
Var
   j : int64;
   i : integer;
Begin
   If ((FGInt.Number[1] Mod 2) = 0) Then ok := false
   Else
   Begin
      i := 0;
      ok := true;
      While ok And (i < 1228) Do
      Begin
         i := i + 1;
         FGIntmodbyint(FGInt, primes[i], j);
         If j = 0 Then ok := false;
      End;
   End;
End;


// Генератор випадкових чисел

Procedure FGIntRandom1(Var Seed, RandomFGInt : TFGInt);
Var
   temp, base : TFGInt;
Begin
   Base10StringToFGInt('281474976710656', base);
   Base10StringToFGInt('44485709377909', temp);
   FGIntMulMod(seed, temp, base, RandomFGInt);
   FGIntDestroy(temp);
   FGIntDestroy(base);
End;


// Тест на простоту числа FGIntp методом Рабіна-Мілера, повертає ok=true якщо
FGIntp пройшло тест

Procedure FGIntRabinMiller(Var FGIntp : TFGInt; nrtest : integer; Var ok :
boolean);
Var
   j, b, i : int64;
   m, z, temp1, temp2, temp3, zero, one, two, pmin1 : TFGInt;
   ok1, ok2 : boolean;
Begin
   randomize;
   j := 0;
   Base10StringToFGInt('0', zero);
   Base10StringToFGInt('1', one);
```

```
   Base10StringToFGInt('2', two);
   FGIntsub(FGIntp, one, temp1);
   FGIntsub(FGIntp, one, pmin1);

   b := 0;
   While (temp1.Number[1] Mod 2) = 0 Do
   Begin
      b := b + 1;
      FGIntShiftRight(temp1);
   End;
   m := temp1;

   i := 0;
   ok := true;
   Randomize;
   While (i < nrtest) And ok Do
   Begin
      i := i + 1;
      Base10StringToFGInt(inttostr(Primes[Random(1227) + 1]), temp2);
      FGIntMontGomeryModExp(temp2, m, FGIntp, z);
      FGIntDestroy(temp2);
      ok1 := (FGIntCompareAbs(z, one) = Eq);
      ok2 := (FGIntCompareAbs(z, pmin1) = Eq);
      If Not (ok1 Or ok2) Then
      Begin

         While (ok And (j < b)) Do
         Begin
            If (j > 0) And ok1 Then ok := false
            Else
            Begin
               j := j + 1;
               If (j < b) And (Not ok2) Then
               Begin
                  FGIntSquaremod(z, FGIntp, temp3);
                  FGIntCopy(temp3, z);
                  ok1 := (FGIntCompareAbs(z, one) = Eq);
                  ok2 := (FGIntCompareAbs(z, pmin1) = Eq);
                  If ok2 Then j := b;
               End
               Else If (Not ok2) And (j >= b) Then ok := false;
            End;
         End;

      End
   End;

   FGIntDestroy(zero);
   FGIntDestroy(one);
   FGIntDestroy(two);
   FGIntDestroy(m);
   FGIntDestroy(z);
   FGIntDestroy(pmin1);
End;


// Розраховуємо коефіцієнти з теореми Безу, FGInt1 * a + FGInt2 * b =
GCD(FGInt1, FGInt2)

Procedure FGIntBezoutBachet(Var FGInt1, FGInt2, a, b : TFGInt);
Var
   zero, r1, r2, r3, ta, gcd, temp, temp1, temp2 : TFGInt;
Begin
   If FGIntCompareAbs(FGInt1, FGInt2) <> St Then
   Begin
      FGIntcopy(FGInt1, r1);
      FGIntcopy(FGInt2, r2);
      Base10StringToFGInt('0', zero);
      Base10StringToFGInt('1', a);
```

```
      Base10StringToFGInt('0', ta);

      Repeat
         FGIntdivmod(r1, r2, temp, r3);
         FGIntDestroy(r1);
         r1 := r2;
         r2 := r3;

         FGIntmul(ta, temp, temp1);
         FGIntsub(a, temp1, temp2);
         FGIntCopy(ta, a);
         FGIntCopy(temp2, ta);
         FGIntDestroy(temp1);

         FGIntDestroy(temp);
      Until FGIntCompareAbs(r3, zero) = Eq;

      FGIntGCD(FGInt1, FGInt2, gcd);
      FGIntmul(a, FGInt1, temp1);
      FGIntsub(gcd, temp1, temp2);
      FGIntDestroy(temp1);
      FGIntdiv(temp2, FGInt2, b);
      FGIntDestroy(temp2);

      FGIntDestroy(ta);
      FGIntDestroy(r1);
      FGIntDestroy(r2);
      FGIntDestroy(gcd);
   End
   Else FGIntBezoutBachet(FGInt2, FGInt1, b, a);
End;


// Знахродимо мультпликативне зворотнє FGInt у кінцевому кільці позитивного
порядку

Procedure FGIntModInv(Const FGInt1, base : TFGInt; Var Inverse : TFGInt);
Var
   zero, one, r1, r2, r3, tb, gcd, temp, temp1, temp2 : TFGInt;
Begin
   Base10StringToFGInt('1', one);
   FGIntGCD(FGInt1, base, gcd);
   If FGIntCompareAbs(one, gcd) = Eq Then
   Begin
      FGIntcopy(base, r1);
      FGIntcopy(FGInt1, r2);
      Base10StringToFGInt('0', zero);
      Base10StringToFGInt('0', inverse);
      Base10StringToFGInt('1', tb);

      Repeat
         FGIntDestroy(r3);
         FGIntdivmod(r1, r2, temp, r3);
         FGIntCopy(r2, r1);
         FGIntCopy(r3, r2);

         FGIntmul(tb, temp, temp1);
         FGIntsub(inverse, temp1, temp2);
         FGIntDestroy(inverse);
         FGIntDestroy(temp1);
         FGIntCopy(tb, inverse);
         FGIntCopy(temp2, tb);

         FGIntDestroy(temp);
      Until FGIntCompareAbs(r3, zero) = Eq;

      If inverse.Sign = negative Then
      Begin
         FGIntadd(base, inverse, temp);
```

```
         FGIntCopy(temp, inverse);
      End;

      FGIntDestroy(tb);
      FGIntDestroy(r1);
      FGIntDestroy(r2);
   End;
   FGIntDestroy(gcd);
   FGIntDestroy(one);
End;


// Простй комбінований тест на простоту FGIntp

Procedure FGIntPrimetest(Var FGIntp : TFGInt; nrRMtests : integer; Var ok :
boolean);
Begin
   FGIntTrialdiv9999(FGIntp, ok);
   If ok Then FGIntRabinMiller(FGIntp, nrRMtests, ok);
End;


//Розрахунок символу Лагранжа

Procedure FGIntLegendreSymbol(Var a, p : TFGInt; Var L : integer);
Var
   temp1, temp2, temp3, temp4, temp5, zero, one : TFGInt;
   i : int64;
   ok1, ok2 : boolean;
Begin
   Base10StringToFGInt('0', zero);
   Base10StringToFGInt('1', one);
   FGIntMod(a, p, temp1);
   If FGIntCompareAbs(zero, temp1) = Eq Then
   Begin
      FGIntDestroy(temp1);
      L := 0;
   End
   Else
   Begin
      FGIntDestroy(temp1);
      FGIntCopy(p, temp1);
      FGIntCopy(a, temp2);
      L := 1;
      While FGIntCompareAbs(temp2, one) <> Eq Do
      Begin
         If (temp2.Number[1] Mod 2) = 0 Then
         Begin
            FGIntSquare(temp1, temp3);
            FGIntSub(temp3, one, temp4);
            FGIntDestroy(temp3);
            FGIntDivByInt(temp4, temp3, 8, i);
            If (temp3.Number[1] Mod 2) = 0 Then ok1 := false Else ok1 := true;
            FGIntDestroy(temp3);
            FGIntDestroy(temp4);
            If ok1 = true Then L := L * (-1);
            FGIntDivByIntBis(temp2, 2, i);
         End
         Else
         Begin
            FGIntSub(temp1, one, temp3);
            FGIntSub(temp2, one, temp4);
            FGIntMul(temp3, temp4, temp5);
            FGIntDestroy(temp3);
            FGIntDestroy(temp4);
            FGIntDivByInt(temp5, temp3, 4, i);
            If (temp3.Number[1] Mod 2) = 0 Then ok2 := false Else ok2 := true;
            FGIntDestroy(temp5);
            FGIntDestroy(temp3);
```

```
            If ok2 = true Then L := L * (-1);
            FGIntMod(temp1, temp2, temp3);
            FGIntCopy(temp2, temp1);
            FGIntCopy(temp3, temp2);
         End;
      End;
      FGIntDestroy(temp1);
      FGIntDestroy(temp2);
   End;
   FGIntDestroy(zero);
   FGIntDestroy(one);
End;


// Розрахунок квадратичного корня за модулем простого числа
// SquareRoot^2 mod Prime = Square

Procedure FGIntSquareRootModP(Square, Prime : TFGInt; Var SquareRoot : TFGInt);
Var
   one, n, b, s, r, temp, temp1, temp2, temp3 : TFGInt;
   a, L, i, j : longint;
Begin
   Base2StringToFGInt('1', one);
   Base2StringToFGInt('10', n);
   a := 0;
   FGIntLegendreSymbol(n, Prime, L);
   While L <> -1 Do
   Begin
      FGIntAddBis(n, one);
      FGIntLegendreSymbol(n, Prime, L);
   End;
   FGIntCopy(Prime, s);
   s.Number[1] := s.Number[1] - 1;
   While (s.Number[1] Mod 2) = 0 Do
   Begin
      FGIntShiftRight(s);
      a := a + 1;
   End;
   FGIntMontgomeryModExp(n, s, Prime, b);
   FGIntAdd(s, one, temp);
   FGIntShiftRight(temp);
   FGIntMontgomeryModExp(Square, temp, Prime, r);
   FGIntDestroy(temp);
   FGIntModInv(Square, Prime, temp1);

   For i := 0 To (a - 2) Do
   Begin
      FGIntSquareMod(r, Prime, temp2);
      FGIntMulMod(temp1, temp2, Prime, temp);
      FGIntDestroy(temp2);
      For j := 1 To (a - i - 2) Do
      Begin
         FGIntSquareMod(temp, Prime, temp2);
         FGIntDestroy(temp);
         FGIntCopy(temp2, temp);
         FGIntDestroy(temp2);
      End;
      If FGIntCompareAbs(temp, one) <> Eq Then
      Begin
         FGIntMulMod(r, b, Prime, temp3);
         FGIntDestroy(r);
         FGIntCopy(temp3, r);
         FGIntDestroy(temp3);
      End;
      FGIntDestroy(temp);
      FGIntDestroy(temp2);
      If i = (a - 2) Then break;
      FGIntSquareMod(b, Prime, temp3);
      FGIntDestroy(b);
```

```
      FGIntCopy(temp3, b);
      FGIntDestroy(temp3);
   End;

   FGIntCopy(r, SquareRoot);
   FGIntDestroy(r);
   FGIntDestroy(s);
   FGIntDestroy(b);
   FGIntDestroy(temp1);
   FGIntDestroy(one);
   FGIntDestroy(n);
End;


End.
```

**Файл FGIntPrimeGeneration.pas – генерація випадкових простих чисел для алгоритму RSA**

```
Unit FGIntPrimeGeneration;

Interface

Uses Windows, SysUtils, Controls, FGInt;

Procedure PrimeSearch(Var GInt : TFGInt);

Implementation


{$H+}


// Відбувається пошаговий пошук простого числа починаючи з GInt,
// якщо воно знайдено то зберігається у GInt

Procedure PrimeSearch(Var GInt : TFGInt);
Var
   temp, two : TFGInt;
   ok : Boolean;
Begin
   If (GInt.Number[1] Mod 2) = 0 Then GInt.Number[1] := GInt.Number[1] + 1;
   Base10StringToFGInt('2', two);
   ok := false;
   While Not ok Do
   Begin
      FGIntAdd(GInt, two, temp);
      FGIntCopy(temp, GInt);
      FGIntPrimeTest(GInt, 4, ok);
   End;
   FGIntDestroy(two);
End;

End.
```

**Файл about.pas – довідка**

```
unit about;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, jpeg;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Memo1: TMemo;
    Panel1: TPanel;
    procedure Panel1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Panel1Click(Sender: TObject);
begin
Form1.Close;
end;

end.
```