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
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA)”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Іванов М.Л.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *125 "*Кібербезпека*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Кібербезпека*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Іванову Максиму Леонідовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA)*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 57-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA)*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи кібербезпеки в промислову експлуатацію.
6. Висновки

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

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Іванов М.Л.
(прізвище та ініціали)

АНОТАЦІЯ

Іванов М.Л. Програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA). 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки у компонентній архітектурі сервісів (SCA).

Метою розробки є програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA).

Результат роботи – програмна реалізація системи кібербезпеки у компонентній архітектурі сервісів (SCA).

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 з ОС Windows 10/11.

Програму розроблено в середовищі Visual C#.

Ключові слова: кібербезпека, компонентна архітектура сервісів

ABSTRACT

Ivanov M.L. Software for a cybersecurity system in a component architecture of services (SCA). 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for a cybersecurity system in a component architecture of services (SCA).

The purpose of the development is software for a cybersecurity system in a component architecture of services (SCA).

The result of the work is a software implementation of a cybersecurity system in a component architecture of services (SCA).

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 are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in the Visual C# environment.

Keywords: cybersecurity, component service architecture

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АРМ	–	автоматизоване робоче місце
ЕЦП	–	електронний цифровий підпис
ЗКЦ	–	засвідчуючий й ключовий центр
ЗЦ	–	засвідчуючий центр
КЦ	–	ключовий центр
ЛОМ	–	локальна обчислювальна мережа
СКЗІ	–	система контролю та захисту інформації
ЦР	–	центр реєстрації
ЦУМ	–	центр управління мережею
SOA	–	Service-Oriented Architecture
SOAP	–	Simple Object Access Protocol

ВСТУП

Актуальність теми. Швидкий розвиток Інтернет-технологій урізноманітнив, з одного боку, категорії розподілених мультимедійних програм (наприклад, IPTV, VoIP, VoD, відеоконференції тощо), а з іншого боку – доступні мережеві технології (наприклад, високошвидкісні, бездротові, мобільні тощо). Ця важлива еволюція прикладного та мережевого рівнів глибоко вплинула на традиційний транспортний рівень. Дійсно, традиційні транспортні протоколи (тобто TCP і UDP) були добре підібрані для початкової моделі мережі з найкращими зусиллями. Однак для роботи з новими мережевими технологіями були потрібні спеціалізації транспортних механізмів (наприклад, розширення TCP для супутникових або Wi-Fi мереж). Аналогічно, нові протоколи, такі як DCCP і SCTP, були запропоновані для покращення базових транспортних послуг, щоб включити нові механізми (наприклад, більш адаптовані стратегії уникнення перевантажень мережі або підтримка багатоадресної мобільності). Однак ці часткові вдосконалення та нові розробки в області транспортних протоколів не ґрунтуються на багаторазовому та сервісно-орієнтованому підході (SCA), призначеному для легкого включення майбутніх розширень і спеціалізацій протоколів. У цьому документі представлено структуру композиції комунікаційних послуг, спрямовану на забезпечення основи для проектування та розвитку автономних транспортних послуг. Цей фреймворк призначений для динамічного створення багаторазових транспортних механізмів. Семантичний простір, що містить описи вимог користувача, транспортних компонентів і базових мережевих служб, включено в цю структуру, щоб керувати процесом створення транспортних послуг.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA).

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

– Огляд існуючих систем у компонентній архітектурі сервісів (SCA).

– Дослідження системи кібербезпеки у компонентній архітектурі сервісів (SCA).

– Програмна реалізація системи кібербезпеки у компонентній архітектурі сервісів (SCA).

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі у компонентній архітектурі сервісів (SCA).

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA), є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для захисту інформації у компонентній архітектурі сервісів (SCA). Це необхідно для забезпечення безпеки інформації, яка передається по системі, побудованій з використанням концепції сервіс-орієнтованої архітектури.

План реалізації безпеки SCA-додатку складається з виконання наступних кроків:

– Сформувати оптимальну робочу групу.

– Створити деталізований план проекту.

– Вести таблицю рішень по безпеці для процесу впровадження SCA.

– Створити проект використання принципу "безпека зсередини" і інфраструктури керування ризиками.

– Визначити зацікавлених осіб усередині організації ("своїх") і за її межами (сторонніх).

– Правильно вибрати й використовувати інструменти для збору вимог.

– При реалізації безпеки SCA-додатку дотримуватися схеми процесу SDLC.

– Ознайомитися з існуючими моделями й учитися на їхньому прикладі.

– Застосувати стандарти WS-Security.

– Розробити стандарти для сторонніх розроблювачів.

Виконавши всі ці дії, ви забезпечите собі гарну стартову точку в реалізації безпеки SCA-додатку.

1.2 Область застосування

Областю застосування системи є компонентна архітектура сервісів (SCA). Архітектура компонентів служб (SCA) складається з ряду специфікацій, що описують модель компонування додатків і систем у концепції архітектури на основі служб (SOA).

Архітектура на основі служб (SOA) – це інфраструктура, що поєднує окремі бізнес-функції й процеси, називані службами, у комплексні бізнес-додатки. В інфраструктурі SOA узагальнені бізнес-компонента представлені службами. SOA дозволяє структурувати ресурси IT у вигляді повторно використовуваних служб, які нежорстко зв'язані між собою й не залежать від платформи й реалізації. В SOA рішення представляються у вигляді комплексу служб, які зв'язані між собою за допомогою чітко певних інтерфейсів і контрактів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA), є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

ViPNet

Для створення інфраструктури електронного цифрового підпису в корпоративних обчислювальних мережах комерційних і державних організацій, що забезпечує виконання вимог Федерального закону "Про ЕЦП", високий рівень безпеки її функціонування й вимог до состава й технічних характеристик програмно-апаратного комплексу "Засвідчуючий центр", пропонується використовувати програмне забезпечення (ПЗ) технології ViPNet із криптографічним ядром "Домен-к", що має сертифікати ФАПСІ (класи КС1, КС2 – для криптографічного ядра "Домен-к" і 4 клас по вимогах ФАПСІ – для персонального мережного екрана ViPNet) і Держтехкомісії Росії (клас 1В – для автоматизованих систем і 3 клас – для міжмережних екранів), у наступному составі:

– **ПЗ ViPNet [Центр управління мережею] (ЦУМ)** призначено для реєстрації користувачів і управління безпекою створеної інфраструктури ЕЦП.

– **ПЗ ViPNet [Засвідчуючий й ключовий центр] (ЗКЦ)** призначено для випуску цифрових сертифікатів як власних користувачів (співробітників організації), так і зовнішніх користувачів (фізичних і/або юридичних осіб), які повинні мати можливість використовувати у відносинах з організацією ЕЦП (наприклад, у системах " клієнт-банк"). ПЗ ЦУМ і ЗКЦ поставляється в складі інсталяційного комплекту ViPNet [Адміністратор].

– **ПЗ ViPNet [Центр реєстрації]** призначено для реєстрації зовнішніх користувачів (фізичних і/або юридичних осіб) і одержання для них в ЗКЦ цифрових сертифікатів.

– **ПЗ ViPNet [Координатор]** призначений для виконання функцій міжмережного екрана й сервера керуючих і поштових повідомлень, через який здійснюється взаємодія з ЗКЦ.

– **ПЗ ViPNet [КриптоСервіс]**, призначено для забезпечення необхідної функціональності роботи з електронним цифровим підписом " Домен-к" (підпис, перевірка підпису й т.д.), а також для автоматизованого захищеного відновлення ключів, довідників і сертифікатів електронного цифрового підпису.

У якості додаткового ПЗ може бути використано:

– **ПЗ ViPNet [Клієнт] [Монітор]** – VPN і персональний мережний екран або ПЗ ViPNet [Персональний мережний екран] (сертифікований ФАПСІ для використання в органах державної влади) призначено для персонального мережного захисту комп'ютерів і виключення доступу до ключової інформації.

– **ПЗ ViPNet [Клієнт] [Ділова пошта]** – Outlook-подібний додаток, що має сертифікат Держтехкомісії по класу 1В и забезпечуюче всі завдання, необхідні для ділового корпоративного спілкування, Автопроцессінг файлів для автоматичного підпису й доставки файлів, юридично значимі підтвердження про доставку й прочитання документів.

Дане програмне забезпечення в захищеному варіанті забезпечує всі необхідні механізми використання ЕЦП із сертифікатами X.509, передбачені Законом про ЕЦП і міжнародними стандартами.

Алгоритми для хешування й підписи реалізовані у відповідності зі стандартами ДЕРЖСТАНДАРТ Р 34.10-94, Р 34.11-94, Р 34.10-2001. Алгоритм шифрування реалізований у відповідності зі стандартом 28147-89

ПЗ ViPNet [Засвідчуючий й ключовий центр] (ЗКЦ)

Програму ViPNet [Засвідчуючий й Ключовий Центр] можна умовно розділити на дві програми: **Ключовий центр** і **засвідчуючий центр**.

Засвідчуючий центр (ЗЦ) призначений для обслуговування наступних запитів: на видання сертифікатів ЕЦП, на відкликання, призупинення й поновлення припиненої дії сертифікатів абонентів, сформованих на мережних вузлах або в Центрах Реєстрації для зовнішніх користувачів.

Програмне забезпечення ЗЦ забезпечує наступну функціональність:

1. Генерація секретних і відкритих ключів Головних абонентів ЗЦ, сертифікатами яких засвідчуються сертифікати користувачів. Сертифікати головних абонентів можуть бути самопідписаними або завіреними вищестоящим ЗЦ.

2. Перше видання сертифіката підпису абонентів відбувається в ЗЦ разом з генерацією секретного ключа для нього. Подальше перевидання сертифіката може відбуватися як в ЗЦ одночасно з формуванням нового секретного ключа (для завдань, що вимагають централізованої генерації й розподілу ключів), так і по запиті користувача корпоративної мережі, сформованого на його мережному вузлі.

3. Видання й реєстрація сертифікатів ЕЦП по запиті абонентів мережі. Запит на сертифікат являє собою шаблон сертифіката, що містить інформацію про абонента, його новий відкритий ключ підпису, передбачуваний термін дії сертифіката, а також інші параметри, що відповідають стандарту X.509. Запит може бути зареєстрований або автоматично або в результаті дій адміністратора ЗЦ. Запит може бути відхилений. Після заповнення полів сертифіката сертифікат через Центр управління відправляється до користувача на комп'ютер.

4. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП абонентів мережі. Ці дії виконуються адміністратором ЗЦ. Довідник відкликаних сертифікатів розсилається абонентам мережі.

5. Реєстрація довідників сертифікатів ЕЦП головних абонентів інших ЗЦ ViPNet. Після перегляду сертифікатів головних абонентів інших ЗЦ і прийняття їх виробляється підпис такого довідника своїм головним абонентом (крос сертифікація). Завірений довідник розсилається по мережі у відповідності зі

зв'язками своїх абонентів, і використовуються при перевірці сертифікатів ЕЦП абонентів інших ЗЦ, що надіслали підписану інформацію на який-небудь вузол своєї мережі.

6. Аналогічним образом виконується імпорт, крос сертифікація й розсилання сертифікатів ЗЦ інших виробників на основі сертифікованих СКЗІ "Криптопро" і " Верба-про". Документи, підписані з використанням зазначених СКЗІ, будуть перевірені тільки при наявності на комп'ютері сертифіката відповідного ЗЦ, завіреного Головним абонентом свого ЗЦ. По міркуваннях безпеки ЗЦ ViPNet вимагає крос сертифікації навіть для сертифікатів інших ЗЦ, у ланцюжку сертифікатів яких утримується сертифікат, якому довіряє ЗЦ ViPNet,

7. Реєстрація довідників відкликаних сертифікатів ЕЦП із інших ЗЦ ViPNet. Такі довідники надходять із інших мереж автоматично, засвідчуються головним абонентом і розсилаються по мережі у відповідності зі зв'язками абонентів мережі. Імпорт довідників відкликаних сертифікатів з ЗЦ інших виробників не виробляється. Доступ до них здійснюється в процесі перевірки підпису по шляху, зазначеному в ЕЦП.

8. Обслуговування запитів зовнішніх користувачів. Зовнішній користувач реєструється на одному з пунктів реєстрації Адміністратором програми ViPNet [Центр Реєстрації] (ЦР). Адміністратор ЦР створює запит на сертифікат ЕЦП для зовнішнього користувача й відсилає його в ЗЦ для видання сертифіката. Запит на сертифікат перед відправленням в ЗЦ підписується ключем підпису цього Адміністратора. Після введення в дію сертифіката зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі мережі із установленим ПЗ ViPNet. Адміністратор ЦР може створювати запити на відкликання, призупинення дії, поновлення дії припиненого сертифіката ЕЦП зовнішніх користувачів.

9. Видання й реєстрація сертифікатів ЕЦП для зовнішніх користувачів виконується тільки по запиті із Центра реєстрації. Запит може бути зареєстрований або відхилений. Вторинні запити на сертифікати, якщо в них

немає змін, і видача сертифікатів можуть оброблятися й видаватися автоматично. Сертифікати через ЦУМ відправляється в ЗЦ

10. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП зовнішніх користувачів може відбуватися по запиті зі ЦР, або самим Адміністратором ЗЦ без запиту зі ЦР. Довідники відкликаних сертифікатів розсилаються по вузлах мережі.

11. Перегляд запитів і сертифікатів ЕЦП. У програмі ЗЦ можливий перегляд будь-яких запитів, сертифікатів, що діють списків відкликання, збереження їх у файл або вивід на друк.

12. Сервісні функції ЗЦ.

ЗЦ інформує адміністратора про витікання термінів дії різних сертифікатів за задане число днів до цього строку шляхом формування відповідних списків.

Автоматично формує архіви інформації через задані інтервали часу при наявності змін, що забезпечує можливість відновлення актуальної інформації.

Забезпечує різні режими функціонування ЗЦ.

Ключовий центр (КЦ) забезпечує захист інфраструктури ЕЦП за допомогою створення системи шифрування інформації у всіх процедурах управління інфраструктурою ЕЦП на основі симетричної схеми розподілу ключів:

– Захист секретних ключів ЕЦП, що централізовано розподіляються (персональні ключі зв'язку з ЗКЦ).

– Захист сертифікатів головних абонентів мережі (персональні ключі зв'язку з ЗКЦ).

– Захист транспортного рівня системи, що забезпечує роботу процедур запитів і одержання сертифікатів, доставки інших файлів інфраструктури ЕЦП (ключі зв'язків транспортного рівня).

– Генерацію паролів для захисту секретних ключів від несанкціонованого доступу. Тип пароля може бути – випадковий (пароль буде створений випадковим образом з різних слів, що утворять випадкову фразу, що запам'ятовується легко), власний (можна задати за бажанням, але не менш 5

символів), випадковий цифровий (сформується випадковим образом з різних цифр).

– Формування інших ключів, що забезпечують відновлення симетричної ключової інформації й роботу іншого ПЗ ViPNet.

Первісний набір симетричних ключів видається користувачеві в складі файлу ключового дистрибутива, зашифрованого паролем, і який користувач одержує при його первинній реєстрації. До складу ключового дистрибутива входить персональний ключ зв'язку з ЗКЦ, ключ зв'язку зі своїм ViPNet-координатором, первинний секретний ключ підпису й сертифікат, сертифікати головних абонентів ЗЦ. Інша ключова інформація надходить на комп'ютер після інсталяції ПЗ ViPNet[Криптосервіс] і в процесі відновлень.

ПЗ ViPNet [Центр управління мережею]

Програма Центра управління забезпечує:

– Реєстрацію вузлів і абонентів корпоративної мережі, реєстрацію "Центрів реєстрації" зовнішніх користувачів.

– Взаємодія з ЗКЦ і користувачами при управлінні сертифікатами.

– Формування захищених довідників доступу для вузлів мережі й довідників зв'язків вузлів і абонентів для ЗКЦ при штатній експлуатації й компрометації ключів абонентів.

– Інші функції.

Взаємодія абонентів з ЗКЦ виробляється тільки через Центр управління. При цьому обидві програм можуть установлюватися на один комп'ютер. Однак для підвищення рівня безпеки функціонування ЗКЦ передбачена можливість установки ЗКЦ і Центра управління на різних комп'ютерах.

ПЗ ViPNet [Центр реєстрації](ЦР)

Програма Центр Реєстрації призначена для реєстрації зовнішніх користувачів і одержання для них в ЗКЦ цифрових сертифікатів. Право працювати в даній програмі визначається програмою Центра управління шляхом

реєстрації вузла в завданні ЦР і формування відповідного довідника доступу. У системі може бути присутнім довільна кількість ЦР.

У Центрі Реєстрації при пред'явленні документів зовнішнім користувачем, що підтверджує його повноваження, створюється запит на сертифікат, виробляється відправлення його в ЗКЦ і здійснюється запровадження в дію виданого в ЗКЦ сертифіката. У Центрі Сертифікації сертифікат буде або вдоволений, або відхилений. Тільки запит на сертифікат зі статусом "удоволений" стає сертифікатом підпису, і цей сертифікат може бути введений у дію. Після введення в дію сертифіката, зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі із установленим ПЗ ViPNet.

Програма виконує наступні функції:

– Генерація секретного ключа підпису й збереження його на персональному ключовому носії зовнішнього користувача.

– Введення персональних даних для сертифіката зовнішнього користувача.

– Формування запиту на сертифікат.

– Підпис запиту на сертифікат ключем діючого адміністратора ЦР.

– Відправлення завіреного запиту в ЗКЦ (через ЦУМ).

– Прийом сертифікатів з ЗКЦ (відбувається автоматично).

– Перегляд запитів і прийнятих сертифікатів.

– Запровадження в дію сертифіката (збереження на персональному ключовому носії зовнішнього користувача).

– Формування запиту на відкликання сертифіката.

– Формування запиту на призупинення сертифіката.

– Формування запиту на поновлення сертифіката.

Крім того, програма виконує експорт сертифікатів у різних кодуваннях.

Всю процедуру створення запиту на одержання сертифіката забезпечує відповідний майстер.

Секретний ключ зовнішнього користувача і його сертифікат заносяться на його персональний носій. Це може бути дискета, E-Token, смарт-карта, Touch memory і інші.

Секретний ключ зашифровується на паролі, вироблюваному програмою. Тип пароля може бути один з наступних:

– Власний пароль.

– Випадкова фраза, що запам'ятовується легко.

– Власний цифровий пароль.

– Випадковий цифровий пароль.

ПЗ ViPNet [КриптоСервіс]

Програма ViPNet [КриптоСервіс] забезпечує необхідну функціональність роботи з електронним цифровим підписом (підпис, перевірка підпису й т.д.), а також автоматизоване захищене відновлення ключів, довідників і сертифікатів електронного цифрового підпису (ЕЦП). При використанні в якості клієнтського програмного забезпечення – ПЗ ViPNet [Клієнт] [Монітор] або ПЗ ViPNet [Клієнт] [Ділова пошта] установка ПЗ ViPNet [КриптоСервіс] не потрібно, тому що вся необхідна функціональність утримується в зазначених програмах.

ПЗ ViPNet [КриптоСервіс] містить набір функцій роботи з ЕЦП, необхідних для використання іншими додатками.

Як додатки, що використовує функції ЕЦП, може використовуватися "Ділова пошта" системи ViPNet, програма Microsoft Outlook, різні WEB – додатки, а також будь-які інші програми, якщо в них убудований виклик криптографічних функцій СКЗІ "Домен-к".

Користувач мережі, використовуючи меню програми, може в будь-який час створити для себе новий секретний ключ і відправити запит в ЗКЦ для одержання нового сертифіката. Однак користувач не може змінити поля в запиті, крім полів термінів дії. Редагування полів сертифіката можливо тільки в ЗКЦ.

Користувач може перемінити пароль, що захищає секретні ключі. Задати тип і термін дії пароля. Призначити строк, за який програма повинна повідомити

його про строк, що наближається, закінчення дії його сертифіката й запропонувати сформувати новий запит.

Транспортний рівень системи ViPNet

Транспортний рівень системи ViPNet використовується всіма компонентами ПЗ ViPNet і забезпечує гарантовану безпечну доставку інформації будь-якого типу, у тому числі всієї інформації, необхідної для функціонування інфраструктури ЕЦП і відновлення ключової інформації. Транспортний рівень функціонує автоматично, не вимагаючи уваги користувача, і використовує протокол TCP/IP як протокол мережного рівня.

Будь-який клієнтський модуль за замовчуванням відправляє зашифрований конверт із інформацією на свій ViPNet [Координатор], що у свою чергу відправляє цей конверт на інший ViPNet [Координатор] або клієнтові – адресатові. Якщо конверт – багатоадресний, то з вихідного конверта на Координаторі створюється необхідне число конвертів відповідно до кількості доступних адрес.

При установці з'єднання з Координатором виробляється сеанс криптографічної автентифікації, і потім відправлення й прийом наявних конвертів по спеціальному протоколі MFTP. Протокол MFTP забезпечує збереження "точки розриву" при розриві каналу зв'язку, що особливо важливо на комутируються линиях, що. Крім того, протокол MFTP на 20-30% менш надлишковий, чим протокол SMTP/POP3.

При необхідності можна настроїти транспортний модуль для передачі інформації прямо іншому вузлу по протоколу MFTP або використовувати поштові сервера SMTP/POP3 або використовувати мережні каталоги локальної мережі.

ПЗ ViPNet [Координатор]

Це багатофункціональний програмний модуль, що забезпечує наступну функціональність:

– міжмережне екранування відповідно до вимог Держтехкомісії по 3 класу. З ПЗ Координатор установлюється ЗКЦ і ЦУМ;

– виконання функцій поштового сервера для забезпечення безпечного обміну повідомленнями інфраструктури ЕЦП, що управляють повідомленнями системи захисту, а також поштовими повідомленнями додатків системи ViPNet;

– при необхідності:

– функції NAT (Network address translation) для VPN-з'єднань із локальної в суспільну мережу;

– туннелювання й шифрування трафіку комп'ютерів, установлених за ViPNet [Координатор] і взаємодіючих з комп'ютерами за іншими ViPNet [Координаторами] або із установленим ПЗ ViPNet [Клієнт].

Забезпечення безпеки функціонування інфраструктури ЕЦП у корпоративних мережах

Необхідний рівень безпеки (клас 1В) забезпечується використанням програмного забезпечення технології ViPNet, сертифікованого по зазначеному класі, а також по інших вимогах безпеки Держтехкомісії й ФАПСІ.

До основних технічних мір, які гарантують високий рівень безпеки використання інфраструктури ЕЦП, можна віднести наступні:

1. Весь інформаційний обмін, пов'язаний із забезпеченням роботи інфраструктури ЕЦП (одержання сертифікатів, їхнє відкликання, призупинення, поновлення, одержання й відновлення довідників відкликаних сертифікатів, довідників сертифікатів Головних абонентів мережі й ін.), виробляється в зашифрованому виді, що виключає будь-які стратегії модифікації, підміни, нав'язування помилкової інформації, несанкціонованого доступу до переданої інформації.

2. Весь службовий інформаційний обмін по ЕЦП забезпечується спеціалізованим захищеним транспортним модулем MFTP, що входить до складу сертифікованого продукту й використовуючим протокол мережного рівня TCP по портах 5000, 5001, 5002, що дозволяє шляхом настроювань для пропуску цього

протоколу на міжмережних екранах виключити можливі мережні атаки через стандартні протоколи.

3. Програмне забезпечення ViPNet на клієнтських станціях забезпечує криптографічний контроль цілісності довідників сертифікатів Головних абонентів ЗКЦ, що гарантує їхній захист від підміни. Всі інші довідники захищені від підміни криптографічними контрольними сумами й підписом ЗКЦ.

4. ЗКЦ ViPNet робить крос сертифікацію сертифікатів інших центрів, що засвідчують, що дозволяє центральній адміністрації контролювати надійність інших центрів, що засвідчують. Підпис особи, якому сертифікат виданий іншим центром, засвідчуючий, зізнається дійсної, тільки у випадку наявності на комп'ютері завіреного своїм ЗКЦ сертифіката цього іншого центра, засвідчуючий, що видали сертифікат даній особі.

5. Міжмережний екран ViPNet [Координатор], що захищає ЗКЦ, сполучений із сервером транспортного модуля MFTP. Мережні вузли мають можливість взаємодіяти тільки з координатором і не мають можливості прямої взаємодії з ЗКЦ і Центром управління, що дозволяє на міжмережному екрані припинити будь-які мережні атаки навіть із боку зареєстрованих користувачів.

6. Якщо на клієнтські комп'ютери можливі мережні атаки, то для захисту криптографічних модулів і ключів ЕЦП на них доцільна установка модуля ViPNet [Клієнт] або його складової ViPNet [Персональний мережний екран]. Модуль ViPNet [Клієнт] додатково до персонального мережного екрана створює зашифрований тунель до ViPNet [Координатора], блокуючи при цьому будь-який відкритий трафік, що повністю виключає будь-які мережні атаки на комп'ютер користувача.

Масштабованість рішення

Центр управління й ЗКЦ ViPNet можуть забезпечити автоматичну захищену взаємодію більш ніж з 65 000 мережними вузлами. При цьому в одному ЗКЦ може бути зареєстроване більш ніж 65 000 абонентів мережі для видачі їм сертифікатів. При цьому, робота відомчих ЗЦ сертифікується Засвідчуючим

Корнєвим центром. При необхідності легко можуть бути створені підвідомчі Центри управління й ЗКЦ (Корпоративні ЗЦ) з делегуванням їм прав по видачі сертифікатів і автоматичною взаємодією між собою.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу "сервер-клієнт", додатки баз даних і багато яких інших. В Visual C# 2012 є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку додатків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0

вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об’єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод Main – точку входу додатка – інкапсуюється у визначення класів. Клас може успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

На додаток до основних описаних об’єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

Архітектура платформи .NET Framework

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що відносяться до автоматичного збору сміття, обробки виключень і керуванню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки у компонентній архітектурі сервісів (SCA).

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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 кібербезпеки 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 кібербезпеки 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

Компонентна архітектура сервісів (SCA) – це безліч специфікацій, що описують модель побудови додатків і систем з використанням сервіс-орієнтованої архітектури (SOA). SCA розширює й доповнює попередні методи реалізації сервісів і ґрунтується на відкритих стандартах, таких як веб-сервіси.

По-суті, SCA забезпечує програмну модель для реалізації SOA.

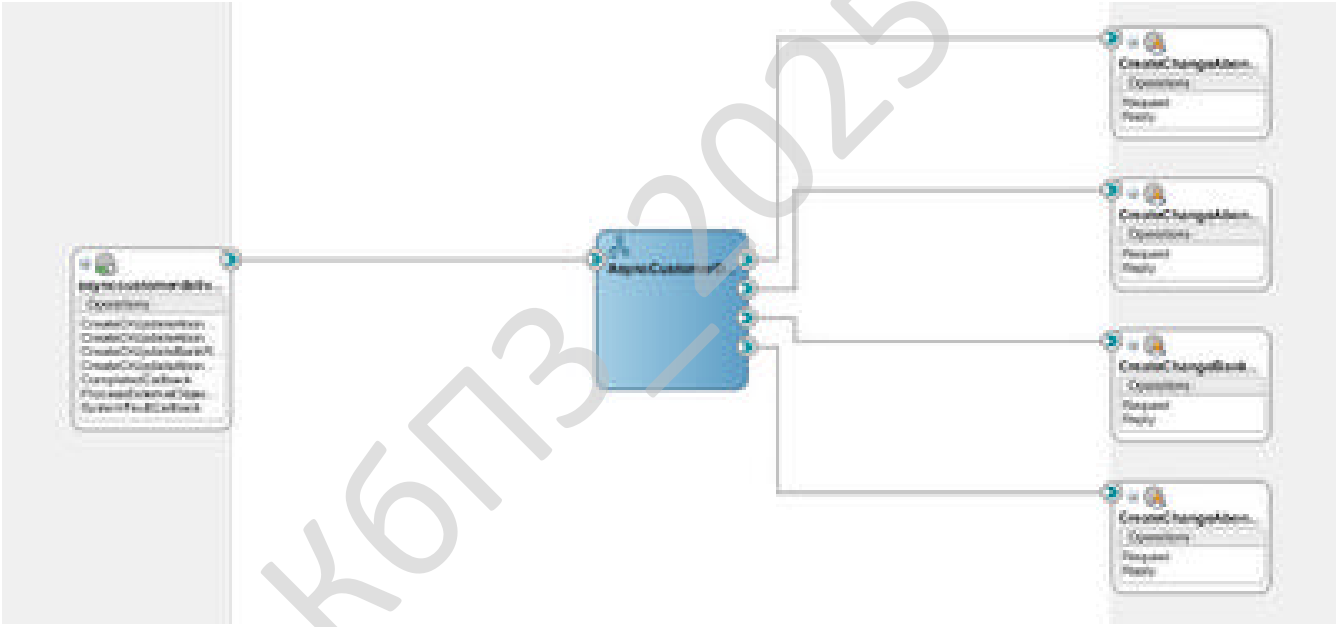

Рисунок 3.1 – Структура програмної моделі SCA

Програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA) базується на використанні WS Security.

Стандарт безпеки WS

Ось три основні механізми, описані стандартом WS Security:

– Підписання повідомлень SOAP – як це зробити таким чином, щоб забезпечити як цілісність, так і неспростовність.

– Шифрування повідомлень SOAP – стандарт пояснює, як можна шифрувати повідомлення SOAP таким чином, щоб забезпечити конфіденційність.

– Прикріплення маркерів безпеки – стандарт рекомендує додавати маркери безпеки таким чином, щоб допомогти встановити особу відправника.

Специфікація WS Security розглядає широкий спектр алгоритмів шифрування та форматів підпису, а також кілька доменів довіри. Стандарт також відкритий для різноманітних моделей маркерів безпеки, включаючи облікові дані користувача/пароля, квитки Kerberos, сертифікати X.509, твердження SAML і користувальницькі маркери. Ви можете знайти інформацію про семантику та формати маркерів у відповідних документах профілю.

Чи може WS Security працювати як комплексне рішення безпеки?

Стандарт WS-Security визначає заходи безпеки, включені в заголовок повідомлення SOAP. Це означає, що заходи безпеки WS працюють на прикладному рівні. Самі по собі ці механізми не забезпечують повного рішення безпеки, а сам WS Security не пропонує повної гарантії безпеки для веб-служб.

Специфіка WS Security слугує будівельним блоком, який слід використовувати разом з іншими розширеннями веб-служб, а також із протоколами вищого рівня, специфічними для програм. В ідеалі ваша безпека повинна включати різноманітні технології та моделі безпеки.

Зауважте, що довірче початкове керування ключами, об’єднання та узгодження технічних деталей (таких як шифри, алгоритми та формати) виходять за межі WS Security. Коли ви впроваджуєте та використовуєте структуру та синтаксис WS Security, ви несете відповідальність за те, щоб ваш результат не був уразливим.

WS Загрози безпеці та заходи протидії

Щоб захистити веб-сервіси, ви можете використовувати протокол HTTPS, який допомагає встановити безпечний зв’язок між клієнтом і сервером через

Інтернет. Для досягнення безпечного зв'язку протокол HTTPS використовує протокол Secure Sockets Layer (SSL), який гарантує, що клієнт і сервер мають цифровий сертифікат, який підтверджує їхню особу.

Нижче наведено кілька кроків, які виконуються під час зв'язку HTTPS між клієнтами та серверами.

1. Клієнт використовує SSL-сертифікат клієнта для надсилання запиту на певний сервер.

2. Сервер отримує сертифікат клієнта.

3. Сервер робить примітку в кеш-системі. Це гарантує, що сервер знає, що відповідь на цей запит має надходити лише до цього конкретного клієнта.

4. Щоб автентифікувати себе клієнту, сервер надсилає власний сертифікат. Ця перевірка відбувається, щоб переконатися, що клієнт досягає зв'язку з правильним сервером.

5. Усі майбутні зв'язки між клієнтом і сервером зашифровані. Якщо зловмисники намагаються зламати безпеку та отримати ці дані, шифрування може перешкодити їм використати дані.

Зазначений вище захист ефективний у деяких випадках, але не забезпечує повного захисту для веб-служб. Наприклад, коли клієнт спілкується з декількома серверами або коли клієнт спілкується з веб-сервером і базою даних одночасно. У цих сценаріях не всю інформацію можна передати через протокол HTTPS.

Безпека SOAP і WS

Специфікації безпеки WS рекомендують застосувати кілька заходів безпеки до протоколу безпеки SOAP. Ці заходи мають бути визначені в елементі заголовка SOAP, який може містити таку інформацію:

– Якщо повідомлення в тілі SOAP підписано будь-яким ключем безпеки, тоді цей ключ можна визначити в елементі заголовка.

– Якщо будь-який елемент у тілі SOAP зашифровано, тоді заголовок має містити всі необхідні ключі шифрування. Це гарантує, що повідомлення буде розшифровано, коли воно досягне місця призначення.

Ось як наведені вище методи автентифікації SOAP можуть сприяти безпеці кількох серверних середовищ:

– Лише хост-сервер може розшифрувати тіло SOAP – шифруючи тіло SOAP, ви гарантуєте, що його може розшифрувати лише веб-сервер, на якому розміщено вашу веб-службу.

– Сервер бази даних не може прочитати повідомлення – якщо повідомлення передається на сервер бази даних через запит HTTP, база даних не може його розшифрувати. Це тому, що він не має механізмів, необхідних для розшифровки повідомлення.

– Розшифрувати можна лише повідомлення протоколу SOAP – запит можна розшифрувати лише тоді, коли він досягає веб-сервера у формі протоколу SOAP. Потім сервер може розшифрувати повідомлення та надіслати відповідну відповідь назад клієнту.

Найкращі методи безпеки веб-служб

Окрім використання стандарту WS Security, ви можете забезпечити безпеку своїх веб-служб, дотримуючись цих найкращих практик.

Забезпечення конфіденційності транспортування

Конфіденційність транспортування може допомогти вам захиститися від таких атак, як Man-in-the-Middle (MITM), і атак підслуховування, які намагаються перехопити, видалити або змінити зв’язок, що надсилається на ваш сервер і з нього. Завжди слід припускати, що всі зв’язки з веб-службами та між ними містять конфіденційні дані.

Передача даних будь-якого типу, особливо конфіденційної або регламентованої інформації, а також автентифіковані сеанси, завжди повинні бути зашифровані за допомогою правильно налаштованих протоколів безпеки транспортного рівня (TLS). Протокол пропонує додаткові переваги, такі як захист від атак повторного відтворення та автентифікацію сервера.

Підтримуйте цілісність повідомлень

TLS може допомогти вам підтримувати цілісність даних у стані спокою. Це означає, що ви повинні застосувати TLS, навіть якщо вміст повідомлення вже зашифровано. Це тому, що TLS також захищає цілісність переданих даних.

Знімки хороші. Постійне тестування безпеки краще.

Сучасне тестування безпеки корпоративного рівня для Інтернету, API, бізнес-логіки та LLM зі швидкістю розгортання.

Відкриті ключі можуть захистити конфіденційність даних, але не захистити їх цілісність. Це пояснюється тим, що до відкритого ключа можуть отримати доступ інші. Крім того, шифрування з відкритим ключем не може підтримувати особу відправника.

Для підтримки цілісності даних XML необхідно використовувати цифрові підписи XML. Цей процес використовує закритий ключ відправника для шифрування. Це дозволяє одержувачам перевіряти підпис за допомогою відкритого ключа відправника.

Зберігайте конфіденційність повідомлень

Ви завжди повинні шифрувати дані за допомогою надійного ключа шифрування. В ідеалі довжина ключа має бути достатньою для запобігання атакам грубої сили. Ось кілька аспектів, які слід враховувати під час впровадження конфіденційності повідомлень:

– Визначайте повідомлення, що містять конфіденційні або регламентовані дані – завжди використовуйте надійні ключі шифрування, щоб захистити конфіденційні дані. Ви можете застосувати як шифрування повідомлень, так і транспортне шифрування, коли дані передаються.

– Визначте конфіденційні дані, які мають залишатися зашифрованими в стані спокою – завжди використовуйте надійні процеси шифрування, щоб захистити ці дані. У цьому випадку транспортного шифрування недостатньо.

Запобігання атак XML DoS

XML-атаки на відмову в обслуговуванні (DoS) дуже поширені у веб-службах і, мабуть, є найнебезпечнішими атаками, які відбуваються в середовищах веб-служб. Нижче наведено два методи, які можуть допомогти вам запобігти XML DoS:

– Перевірити – завжди перевіряти на розгортання об’єктів XML, завеликі корисні навантаження та рекурсивні корисні навантаження. Ви також повинні перевіряти надто довгі імена елементів, особливо у веб-службах на основі SOAP.

– Тест – розробка тестових прикладів, які можуть допомогти вам змоделювати та визначити, чи здатний синтаксичний аналізатор XML або засіб перевірки схеми захищатися від атак XML DoS.

Забезпечити доступність

Коли відбуваються атаки, веб-службі може знадобитися більше ресурсів, ніж доступно. Це може призвести до стану нестабільності та призвести до відмови в обслуговуванні. Під час налаштування використання ресурсів застосовуйте такі обмеження, щоб забезпечити доступність під час атак:

– Кількість циклів процесора – щоб забезпечити стабільність, ви повинні встановити це обмеження відповідно до очікуваної швидкості обслуговування.

– Обсяг пам’яті – щоб запобігти збоям системи, установіть обмеження на обсяг пам’яті, який може використовувати веб-служба.

– Кількість одночасних операцій – для забезпечення стабільності встановіть обмеження на кількість процесів, відкритих файлів і мережевих підключень, яким дозволено працювати одночасно.

Важливо віддати пріоритет безпеці та забезпечити безперервний захист своїх веб-служб. Однак безпека не повинна відбуватися за рахунок доступності. При правильному застосуванні ці заходи безпеки WS повинні допомогти вам підтримувати доступність під час атак. Коли системи виходять з ладу, ви зможете швидко й ефективно відновити нормальну роботу.

3.2 Розробка структурної схеми

Ідентифікація додатків або ділових процесів звичайно здійснюється за допомогою сертифікатів, і в цьому випадку управляти паролями на стороні клієнта не потрібно. Обіг із сертифікатами для зазначеного методу автентифікації описується в профілі X.509 Certificate Token Profile. Існують і інші профілі, приміром, для використання токенів мови розмітки тверджень безпеки (Security Assertion Markup Language, SAML) або Kerberos.

Двійкові або базовані на XML токени безпеки потрібні не тільки для автентифікації. Вони виконують ще одну функцію, являючи собою основу для транспорту або прив'язки ключів (Keys), застосовуваних у криптографії.

Шифрування

Щоб забезпечити захист конфіденційних даних, використовується криптографічне шифрування. Оскільки протокол SOAP базується на XML, то WS Security не визначає новий стандарт, а використовує специфікацію XML Encryption з W3C. Зашифровані дані і їхня метаінформація, у свою чергу, включаються в повідомлення у вигляді структур XML. Однак, відповідно до специфікації SOAP, не можна шифрувати елементи «конверт» (Envelope), «заголовок» (Header) і «тіло» (Body), оскільки вони задають структуру повідомлення й повинні бути читаємі завжди.

Принципово розрізняють два механізми шифрування: симетричне й асиметричне. При симетричному шифруванні (метод «секретного ключа» – Secret Key) для шифрування й дешифрування використовується загальний ключ, завжди доступним обом сторонам. При асиметричному шифруванні (алгоритм із відкритими ключами – Public Key) для шифрування й дешифрування застосовуються різні ключі, що істотно скорочує витрати зусиль на їхній розподіл: особистий ключ (Private Key) залишається у власника, а загальний ключ (Public Key) поширюється вільно. Однак у порівнянні із секретними ключами механізм відкритих ключів працює значно повільніше, тому обидва підходи часто поєднують, у результаті чого з'являються нові гібридні варіанти. Клієнт генерує

симетричний ключ сеансу (Session Key) і використовує його для симетричного шифрування більших обсягів даних. На закінчення симетричний ключ шифрується за допомогою асиметричного алгоритму, вкладається в повідомлення й надається в розпорядження сервісу.

Підпис (Signature)

Для підтвердження цілісності повідомлень застосовуються підписи. Вони дозволяють розпізнати неправомірні модифікації: зміна, видалення або додавання даних. Реалізація цього підходу в рамках WS Security опирається на стандарт XML Digital Signature від W3C. Принцип підписів заснований на створенні контрольних сум за допомогою спеціальних алгоритмів (дайджет). Результати приєднуються до повідомлення й передаються в частково зашифрованому виді. Сервісна сторона формує контрольну суму й порівнює неї зі значенням, присланим клієнтом. Оскільки в XML різні способи написання логічно ідентичні, перед формуванням контрольної суми необхідно зробити нормалізацію даних. Для цього використовуються стандартизовані алгоритми XML Canonicalization, також запозичені з W3C.

Крім того, підпису надають можливість установлення автентичності відправника. Цю інформацію можна використовувати в юридичних цілях для встановлення авторства.

Склад Комплексу

Комплекс складається з наступних компонентів:

1. Набір шлюзів кодування.
2. Центр генерації ключів.
3. Центр розподілу ключів.
4. Центр реєстрації мобільних клієнтів.
5. Центр підготовки електронних ключів мобільних клієнтів.
6. Мобільний клієнт.
7. Центр моніторингу.
8. Програма контролю цілісності.

Рисунок 3.2 – Структурна схема системи

3.3 Розробка функціональної схеми

Розглянемо функціональну схему розробленої системи. Вона зображена на рисунку 3.3. Як видно з рисунку система складається з наступних елементів:

1. Блок визначення алгоритму для формування ключа. Він може бути для симетричних та асиметричних алгоритмів. У першому випадку формується один ключ, який таємно розсилається учасникам зашифрованого обміну даними, у другому випадку генерується пара ключів: відкрий ключ – таємний ключ. Цей блок призначений для визначення по алгоритму шифрування параметрів та обмежень при формуванні ключа.

2. Генератор випадкових чисел. Призначений для генерації випадкових

чисел, які використовуються як при генерації ключів шифрування/дешифрування, так й для формування іншої інформації потрібної для авторизації та автентифікації учасників інформаційного обміну.

3. Блок генерації ключів. Призначений для генерації ключів для учасників інформаційного обміну.

4. Блок формування сертифікатів. Призначений для формування сертифікатів, які удостовірюють справжність ключів, та автентифікують користувачів.

5. База даних відкликаних сертифікатів, призначена для зберігання тих сертифікатів, які були зкомпроментовані.

6. Центр реєстрації.

Розглянемо ці блоки та їх функціональність більш детальніше.

Центр розподілу ключів

Центр розподілу ключів (ЦРК) призначений для обслуговування наступних запитів: на видання сертифікатів ЕЦП, на відкликання, призупинення й поновлення припиненої дії сертифікатів абонентів, сформованих на мережних вузлах або в Центрах Реєстрації для зовнішніх користувачів.

ЦРК забезпечує наступну функціональність:

1. Перше видання сертифіката підпису абонента відбувається в ЦРК разом з генерацією секретного ключа для нього. Подальше перевидання сертифіката може відбуватися як у ЦРК одночасно з формуванням нового секретного ключа (для завдань, що вимагають централізованої генерації й розподілу ключів), так і по запиту користувача корпоративної мережі, сформованого на його мережному вузлі.

2. Видання й реєстрація сертифікатів ЕЦП по запиту абонентів мережі. Запит на сертифікат являє собою шаблон сертифіката, що містить інформацію про абонента, його новий відкритий ключ підпису, передбачуваний термін дії сертифіката, а також інші параметри, що відповідають стандарту X.509. Запит може бути зареєстрований або автоматично або в результаті дій

адміністратора ЦРК. Запит може бути відхилений. Після заповнення полів сертифіката сертифікат через Центр керування відправляється до користувача на комп'ютер.

3. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП абонентів мережі. Ці дії виконуються адміністратором ЦРК. Довідник відкликаних сертифікатів розсилається абонентам мережі.

4. Реєстрація довідників сертифікатів ЕЦП головних абонентів інших ЦРК. Після перегляду сертифікатів головних абонентів інших ЦРК і прийняття їх виробляється підпис такого довідника своїм головним абонентом (крос сертифікація). Завірений довідник розсилається по мережі у відповідності зі зв'язками своїх абонентів, і використовуються при перевірці сертифікатів ЕЦП абонентів інших ЦРК, що надіслали підписану інформацію на який-небудь вузол своєї мережі.

5. Аналогічним чином виконується імпорт, крос сертифікація й розсилання сертифікатів ЦРК інших виробників на основі сертифікованих СКЗІ**.** Документи, підписані з використанням зазначених СКЗІ, будуть перевірені тільки при наявності на комп'ютері сертифіката відповідного ЦРК, завіреного Головним абонентом свого ЦРК. По міркуваннях безпеки ЦРК вимагає крос сертифікації навіть для сертифікатів інших ЦРК, у ланцюжку сертифікатів яких утримується сертифікат, якому довіряє ЦРК.

6. Реєстрація довідників відкликаних сертифікатів ЕЦП із інших ЦРК. Такі довідники надходять із інших мереж автоматично, засвідчуються головним абонентом і розсилаються по мережі у відповідності зі зв'язками абонентів мережі. Імпорт довідників відкликаних сертифікатів зі ЦРК інших виробників не виробляється. Доступ до них здійснюється в процесі перевірки підпису по шляху, зазначеному в ЕЦП.

7. Обслуговування запитів зовнішніх користувачів. Зовнішній користувач реєструється на одному з пунктів реєстрації. Адміністратор створює запит на сертифікат ЕЦП для зовнішнього користувача й відсилає його в ЦРК для видання

сертифіката. Запит на сертифікат перед відправленням у ЦРК підписується ключем підпису цього Адміністратора. Після введення в дію сертифіката зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі мережі із установленим ПЗ. Адміністратор може створювати запити на відкликання, призупинення дії, поновлення дії припиненого сертифіката ЕЦП зовнішніх користувачів.

8. Видання й реєстрація сертифікатів ЕЦП для зовнішніх користувачів виконується тільки по запиту із Центра реєстрації (ЦР). Запит може бути зареєстрований або відхилений. Вторинні запити на сертифікати, якщо в них немає змін, і видача сертифікатів можуть оброблятися й видаватися автоматично. Сертифікати відправляється в ЦРК.

9. Відкликання сертифікатів, призупинення дії сертифікатів, поновлення дії сертифікатів ЕЦП зовнішніх користувачів може відбуватися по запиту зі ЦР, або самим Адміністратором ЦРК без запиту зі ЦР. Довідники відкликаних сертифікатів розсилаються по вузлах мережі.

10. Перегляд запитів і сертифікатів ЕЦП.

11. Розбір конфліктних ситуацій і експертизи правочинності й дійсності електронних документів, підписаних ЕЦП, виробляється відповідно до Документа "СКЗІ. Порядок розбору конфліктних ситуацій, пов'язаних із застосуванням ЕЦП".

12. Сервісні функції ЦРК.

ЦРК інформує адміністратора про витікання термінів дії різних сертифікатів за задане число днів до цього строку шляхом формування відповідних списків.

Автоматично формує архіви інформації через задані інтервали часу при наявності змін, що забезпечує можливість відновлення актуальної інформації.

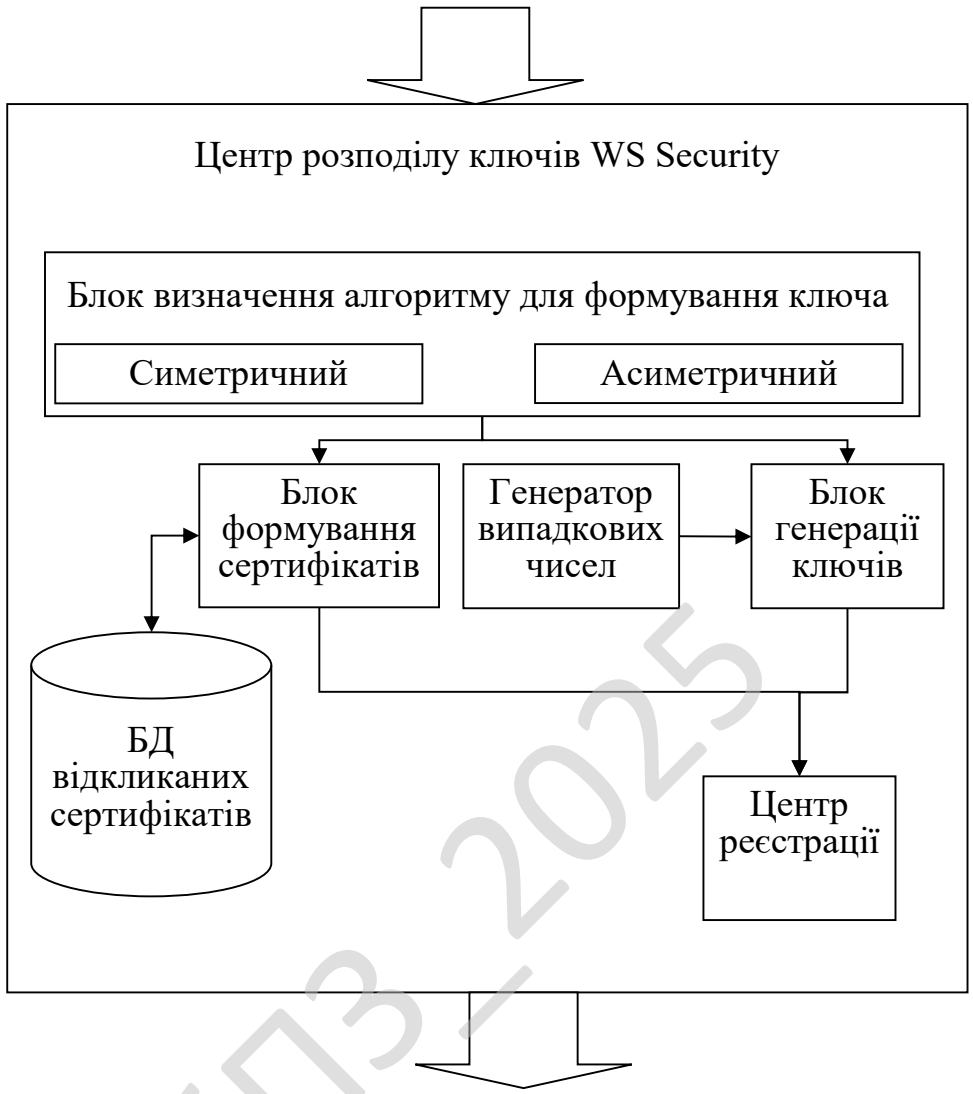

Рисунок 3.3 – Функціональна схема центру розподілу ключів

Ключовий центр

Ключовий центр (КЦ) забезпечує захист інфраструктури ЕЦП за допомогою створення системи шифрування інформації у всіх процедурах керування інфраструктурою ЕЦП на основі симетричної схеми розподілу ключів:

– Захист секретних ключів ЕЦП, що розподіляються централізовано.

– Захист сертифікатів абонентів мережі.

– Захист транспортного рівня системи, що забезпечує роботу процедур запитів і одержання сертифікатів, доставки інших файлів інфраструктури ЕЦП.

– Генерацію паролів для захисту секретних ключів від несанкціонованого доступу. Тип пароля може бути – випадковий (пароль буде створений випадковим образом з різних слів, що утворять випадкову фразу, що запам'ятовується легко), власний (можна задати за бажанням, але не менш 5 символів), випадковий цифровий (сформується випадковим образом з різних цифр).

– Формування інших ключів, що забезпечують відновлення симетричної ключової інформації й роботу іншого ПЗ.

Первісний набір симетричних ключів видається користувачеві в складі файлу ключового дистрибутива, зашифрованого на паролі, і який користувач одержує при його первинній реєстрації. До складу ключового дистрибутива входить персональний ключ зв'язку з УКЦ, ключ зв'язку зі своїм координатором, первинний секретний ключ підпису й сертифікат, сертифікати головних абонентів ЦРК. Інша ключова інформація надходить на комп'ютер після інсталяції ПЗ і в процесі відновлень.

Центр керування мережею

Центра керування забезпечує:

– Реєстрацію вузлів і абонентів корпоративної мережі, реєстрацію "Центрів реєстрації" зовнішніх користувачів.

– Взаємодія зі ЦРК і користувачами при керуванні сертифікатами.

– Формування захищених довідників доступу для вузлів мережі й довідників зв'язків вузлів і абонентів для УКЦ при штатній експлуатації й компрометації ключів абонентів.

– Інші функції.

Взаємодія абонентів зі ЦРК виробляється тільки через Центр керування мережею.

Центр реєстрації

Центр Реєстрації призначений для реєстрації зовнішніх користувачів і одержання для них цифрових сертифікатів. У Центрі Реєстрації при пред'явленні документів зовнішнім користувачем, що підтверджує його повноваження, створюється запит на сертифікат, виробляється відправлення його в ЦРК і здійснюється запровадження в дію виданого в ЦРК сертифіката. У Центрі Сертифікації сертифікат буде або задоволений, або відхилений. Тільки запит на сертифікат зі статусом "задоволений" стає сертифікатом підпису, і цей сертифікат може бути уведений у дію. Після введення в дію сертифіката, зовнішній користувач зможе користуватися ним (підписувати документи) на будь-якому вузлі із установленим ПЗ.

Центр Реєстрації виконує наступні функції:

– Генерація секретного ключа підпису й збереження його на персональному ключовому носії зовнішнього користувача.

– Уведення персональних даних для сертифіката зовнішнього користувача.

– Формування запиту на сертифікат.

– Підпис запиту на сертифікат ключем діючого адміністратора ЦР.

– Відправлення завіреного запиту в ЦРК

– Прийом сертифікатів зі ЦРК (відбувається автоматично).

– Перегляд запитів і прийнятих сертифікатів.

– Запровадження в дію сертифіката (збереження на персональному ключовому носії зовнішнього користувача).

– Формування запиту на відкликання сертифіката.

– Формування запиту на призупинення сертифіката.

– Формування запиту на поновлення сертифіката.

Крім того, Центр Реєстрації виконує експорт сертифікатів у різних кодуваннях.

Секретний ключ зовнішнього користувача і його сертифікат заносяться на його персональний носій. Це може бути дискета, компактний диск (CD), E-Token, смарт-карта, Touch memory і інші.

Секретний ключ зашифровується на паролі, вироблюваному програмою. Тип пароля може бути один з наступних:

– Власний пароль.

– Випадкова фраза, що запам'ятовується легко.

– Власний цифровий пароль.

– Випадковий цифровий пароль.

Розподіл відкритих ключів

Розглянемо основні алгоритми розподілу ключів.

На сьогоднішній день відомі наступні методи розподілу відкритих ключів:

– індивідуальне публічне оголошення відкритих ключів сервісами;

– використання привселюдно доступного каталогу відкритих ключів;

– участь авторитетного джерела відкритих ключів;

– сертифікати відкритих ключів.

Розглянемо кожний з перерахованих методів.

При *індивідуальному публічному оголошенні відкритих ключів* будь-яка сторона, що бере участь в обміні повідомленнями (X), може надати свій відкритий ключ (K_O) будь-якій іншій стороні. Недоліком даного підходу є неможливість забезпечити автентифікацію відправника відкритого ключа (K_O). Тобто, при даному підході в порушника з'являється можливість фальсифікації сервісів.

Використання привселюдно доступного каталогу відкритих ключів дозволяє домогтися більше високого ступеня захисту інформації й мережі побудованої з використанням сервіс-компонентної архітектури. У цьому випадку за ведення й поширення публічного каталогу повинна відповідати надійна організація (уповноважений об'єкт). При цьому повинні дотримуватися наступні правила.

1. Сервіси повинні реєструвати свої відкриті ключі в публічному каталозі, що веде вповноважений об'єкт.

2. Реєстрація повинна проходити по заздалегідь захищених каналах зв'язку.

3. Уповноважений об'єкт повинен періодично публікувати каталог відкритих ключів.

Недоліком даного підходу є наступне. Якщо порушникові вдасться змінити записи, що зберігаються в каталозі відкритих ключів, то він зможе авторитетно видавати фальсифіковані відкриті ключі й, отже, виступати від імені кожного з учасників обміну даними й читати повідомлення, призначені будь-якому сервісу.

Участь авторитетного джерела відкритих ключів. Обов'язковою умовою даного варіанта розподілу відкритих ключів сервісів є умова, що авторитетне джерело відкритих ключів має свій секретний ключ, і кожний сервіс знає його відкритий ключ. При цьому виконується наступний порядок дій:

1. Сервіс 1 надсилає запит авторитетному джерелу відкритих ключів про поточне значення відкритого ключа сервісу 2. При цьому вказується дата й час запиту (д.вр.).

2. Авторитетне джерело, використовуючи свій секретний ключ $K_{\text{С}}^{\text{А}}$, шифрує й передає повідомлення сервісу 1 $Y_1 = E_{K_{\text{С}}^{\text{А}}}\left(K_{\text{О}}^{\text{П2}}, \text{д. вр.}\right)$, у якому втримується наступна інформація:

– $K_{\text{О}}^{\text{П2}}$ – відкритий ключ сервісу 2;

– буд. вр. – дата й час відправлення повідомлення.

3. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, шифрує й передає сервісу 2 шифроване повідомлення $Y_2 = E_{K_{\text{О}}^{\text{П2}}}\left(ID_1, N_1\right)$, що містить:

– ID_1 – ідентифікатор відправника (сервіс 1);

– N_1 – унікальну мітку даного повідомлення.

4, 5. Сервіс 2, одержавши шифроване повідомлення $Y_2 = E_{K_{\text{О}}^{\text{П2}}}\left(ID_1, N_1\right)$, дешифрує його за допомогою свого секретного ключа $K_{\text{С}}^{\text{П2}}$ $\left(ID_1, N_1\right) = D_{K_{\text{С}}^{\text{П2}}}\left(Y_2\right)$ й

відповідно до ідентифікатора ID_1, аналогічно з пунктами 1 і 2 вище перерахованих дій одержує від авторитетного джерела відкритий ключ сервісу 1 $K_{\text{О}}^{\text{П1}}$.

6. Сервіс 2, використовуючи $K_{\text{О}}^{\text{П1}}$, посилає сервісу 1 шифроване повідомлення $Y_4 = E_{K_{\text{О}}^{\text{П1}}}(N_1, N_2)$, де N_2 – унікальна мітка даного повідомлення.

7. Сервіс 1 шифрує за допомогою відкритого ключа $K_{\text{О}}^{\text{П2}}$ повідомлення Y, призначене сервісу 1 і передає $Y_5 = E_{K_{\text{О}}^{\text{П2}}}(Y, N_2)$.

Наведений варіант розподілу відкритих ключів має деякі недоліки:

– щораз, коли сервіс має намір передати інформацію новому адресатові, то він повинен звертатися до авторитетного джерела з метою одержання відкритого ключа;

– каталог імен і відкритих ключів, підтримуваний авторитетним джерелом, є привабливим місцем для порушника передачі інформації сервісів.

Сценарій розподілу відкритих ключів із застосуванням сертифікатів відкритих ключів. Обов'язковою умовою даного варіанта розподілу відкритих ключів сервісів є умова, що авторитетне джерело сертифікатів має свій секретний ключ $K_{\text{С}}^{\text{А}}$, і кожний сервіс знає його відкритий ключ $K_{\text{О}}^{\text{А}}$. При цьому виконується наступний порядок дій:

1. Сервіс 1 генерує пари ключів $[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}]$ (відповідно, відкритий і секретний) і по захищеному каналу зв'язку звертається до авторитетного джерела сертифікатів з метою одержання сертифіката.

2. Авторитетне джерело шифрує за допомогою свого секретного ключа $K_{\text{С}}^{\text{А}}$ сертифікат $C_{\text{П1}} = E_{\text{Кс}}^{\text{А}}[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}]$ і видає його сервісу 1. Сертифікат містить:

– $K_{\text{О}}^{\text{П1}}$ – відкритий ключ сервісу 1 (даний ключ сервіс 1 сам згенерував і передав авторитетному джерелу для сертифікації);

– $ID_{\text{П1}}$ – ідентифікатор сервісу 1;

– $T_{\text{П1}}$ – термін дії сертифіката сервісу.

3. Сервіс 1 пересилає свій сертифікат $C_{П1} = E^{A}_{Kc}\left[K^{П1}_{O}, ID_{П1}, T_{П1}\right]$, отриманий від авторитетного джерела, сервісу 2. Останній, знаючи відкритий ключ авторитетного джерела сертифікатів K^{A}_{O}, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом $D^{A}_{Ko}\left[C_{П1}\right] = D^{A}_{Ko}\left[E^{A}_{Kc}\left[K^{П1}_{O}, ID_{П1}, T_{П1}\right]\right] = \left[K^{П1}_{O}, ID_{П1}, T_{П1}\right]$.

4. 5, 6. Сервіс 2 виконує аналогічні дії, які були виконані сервісом 1 у пунктах 1, 2 і 3. Тобто одержує від авторитетного джерела сертифікат $C_{П2} = E^{A}_{Kc}\left[K^{П2}_{O}, ID_{П2}, T_{П2}\right]$. Пересилає його сервісу 1. Останній, знаючи відкритий ключ авторитетного джерела сертифікатів K^{A}_{O}, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом

$$D^{A}_{Ko}\left[C_{П2}\right] = D^{A}_{Ko}\left[E^{A}_{Kc}\left[K^{П2}_{O}, ID_{П2}, T_{П2}\right]\right] = \left[K^{П2}_{O}, ID_{П2}, T_{П2}\right].$$

У результаті перерахованих дій сервіси обмінялися відкритими ключами й готові до передачі й прийому сервісних повідомлень.

Застосування криптосистеми з відкритим ключем для розподілу секретних ключів

На сьогоднішній день існує кілька підходів застосування криптосистеми з відкритим ключем для розподілу секретних ключів [7]. Розглянемо деякі з них.

Простий розподіл секретних ключів складається у виконанні наступних дій:

1. Сервіс 1 генерує пари ключів $\left[K^{П1}_{O}, K^{П1}_{C}\right]$, відповідно, відкритий і секретний.

2. Сервіс 1 передає сервісу 2 повідомлення $X_{П1} = \left[ID_{П1}, K^{П1}_{O}\right]$, де $ID_{П1}$ – ідентифікатор сервісу 1.

3. Сервіс 2, одержавши повідомлення $X_{П1} = \left[ID_{П1}, K^{П1}_{O}\right]$ від сервісу 1, так само генерує свою пару ключів $\left[K^{П2}_{O}, K^{П2}_{C}\right]$.

4. Сервіс 2, використовуючи відкритий ключ $K^{П1}_{O}$ сервісу 1, шифрує й передає повідомлення $Y_{П2} = E_{K^{П1}_{O}}\left[ID_{П2}, K^{П2}_{C}\right]$ сервісу 1.

5. Сервіс 1 знищує свій секретний ключ $K_{\text{С}}^{\text{П1}}$, а сервіс 2 знищує відкритий ключ сервісу 1 $K_{\text{О}}^{\text{П1}}$.

Таким чином, обидва сервіси мають сеансовий (секретний) ключ $K_{\text{С}}^{\text{П2}}$ і можуть використовувати його для передачі інформації, захищеної традиційним шифруванням. По закінченні сеансу передачі інформації ключ $K_{\text{С}}^{\text{П2}}$ знищується. Однак даний підхід уразливий для активних порушень. Дійсно, якщо порушник має можливість впровадження в з'єднання між сервісами, те, виконуючи наступні дії, він буде мати можливість знати секретний (сеансовий) ключ.

1. Сервіс 1 генерує пари ключів $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$ і передає сервісу 2 повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$.

2. Порушник перехоплює повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$, створює власну пару ключів $\left[K_{\text{О}}^{\text{Н}}, K_{\text{С}}^{\text{Н}}\right]$ і передає сервісу 2 повідомлення $X_{\text{Н}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{Н}}\right]$.

3. Сервіс 2, одержавши повідомлення $X_{\text{Н}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{Н}}\right]$, генерує свою пару ключів $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$, шифрує (використовуючи відкритий ключ порушника $K_{\text{О}}^{\text{Н}}$) і передає повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{Н}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$ сервісу 1.

4. Порушник перехоплює повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{Н}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$, дешифрує його $\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor = D_{K_{\text{С}}^{\text{Н}}}\left[Y_{\text{П2}}\right]$, визначає сеансовий ключ $K_{\text{С}}^{\text{П2}}$ і передає сервісу 2 повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{П1}}}\left\lfloor ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right\rfloor$.

У результаті обидва сервіси мають сеансовий ключ $K_{\text{С}}^{\text{П2}}$, однак не будуть підозрювати, що він теж відомий і порушникові.

Сценарій розподілу секретних ключів із забезпеченням конфіденційності й автентичності і складається у виконанні наступних дій.

1. Сервіси генерують пари ключів, відповідно $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$, $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$, і обмінюються між собою відкритими ключами $K_{\text{О}}^{\text{П1}}$ й $K_{\text{О}}^{\text{П2}}$.

2. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, передає сервісу 2 повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor ID_{\text{П1}}, N_1 \right\rfloor$, що містить: свій ідентифікатор – $ID_{\text{П1}}$; N_1 – унікальна мітка даних повідомлень.

3. Сервіс 2, використовуючи $K_{\text{О}}^{\text{П1}}$, передає сервісу 1 повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{П1}}}\left\lfloor N_2, N_1 \right\rfloor$, що містить N_1 і N_2 - унікальні мітки даного повідомлення. Наявність мітки N_1 переконує сервісу 1 у тім, що тільки сервіс 2 міг дешифрувати повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor ID_{\text{П1}}, N_1 \right\rfloor$.

4. Сервіс 1, використовуючи $K_{\text{О}}^{\text{П2}}$, передає сервісу 2 повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}\left\lfloor N_2 \right\rfloor$, що містить унікальну мітку N_2. Дане повідомлення виконує функцію підтвердження для сервісу 2, що його респондентом є сервіс 1.

5. Сервіс 1 генерує секретний (сеансовий) ключ $K_{\text{С}}$, що двічі шифрується з використанням: свого секретного ключа $E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]$ й відкритого ключа сервісу 2 $E_{K_{\text{О}}^{\text{П2}}}[\, E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]\,]$. Після виконання процедури шифрування повідомлення $Y_{\text{П1}} = E_{K_{\text{О}}^{\text{П2}}}[\, E_{K_{\text{С}}^{\text{П1}}}[K_{\text{С}}]\,]$ передається сервісу 2. Останній, маючи відкритий ключ сервісу 1 і свій секретний ключ, дешифрує отримане повідомлення.

У результаті перераховані дії обидва сервіси мають секретний (сеансовий) ключ $K_{\text{С}}$.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Інтерфейс ПЗ.

– Припинення строку дії

– Зміна статусу сертифікату.

– Використання сертифіката.

– Зберігання сертифіката.

– Реєстрація/видання сертифіката.

– Розповсюдження сертифіката.

– Генерація пари ключів.

– Транспортування ключа.

– Ініціалізація ключа.

– Знищення ключа.

– Використання ключа.

– Резервне копіювання ключа.

– Архівування ключа.

– Обмежене використання ключа.

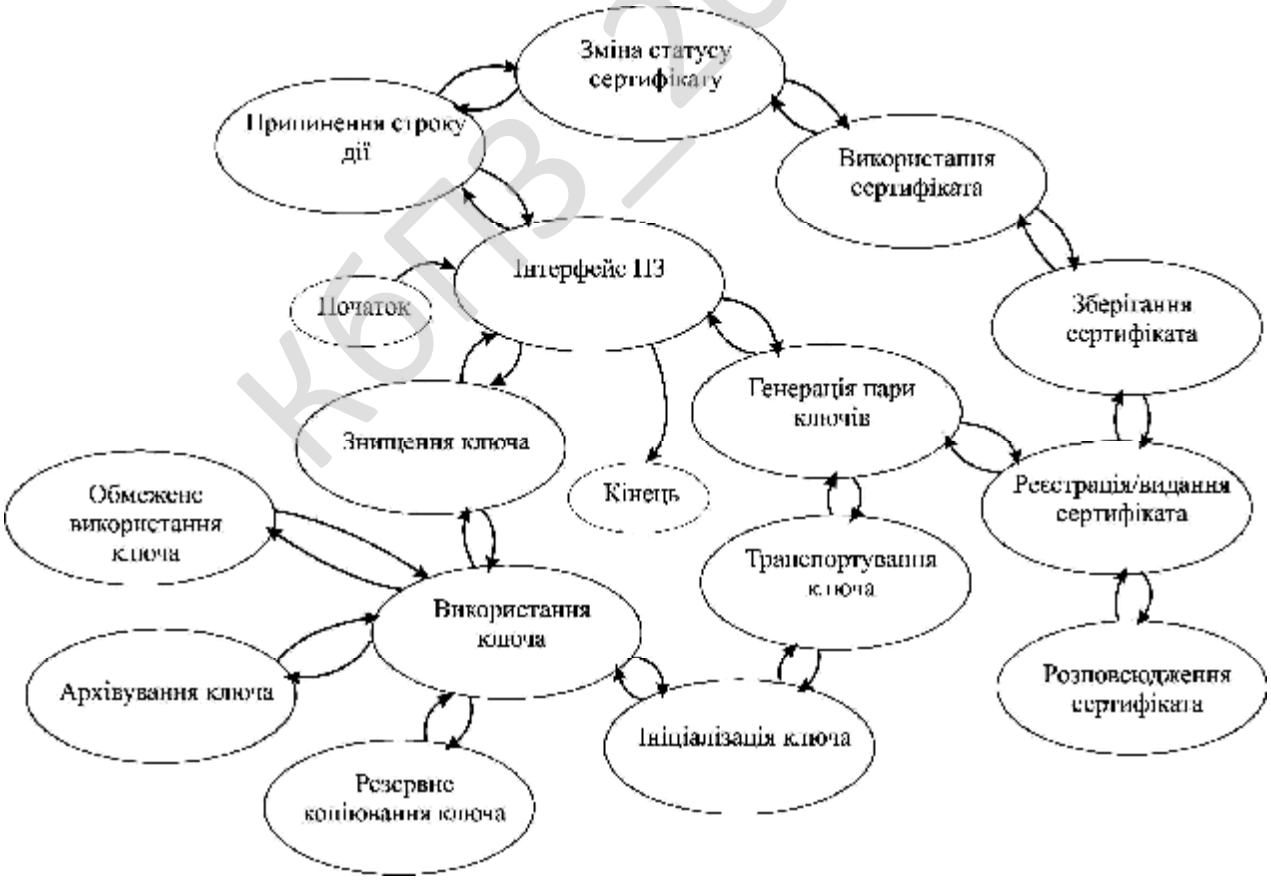

Рисунок 3.4 – Діаграма взаємодії процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми. З якої видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

При розробці використовувались концепції діаграм діяльності. Тобто в UML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Це фундаментальна одиниця визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії.

Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності.

Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

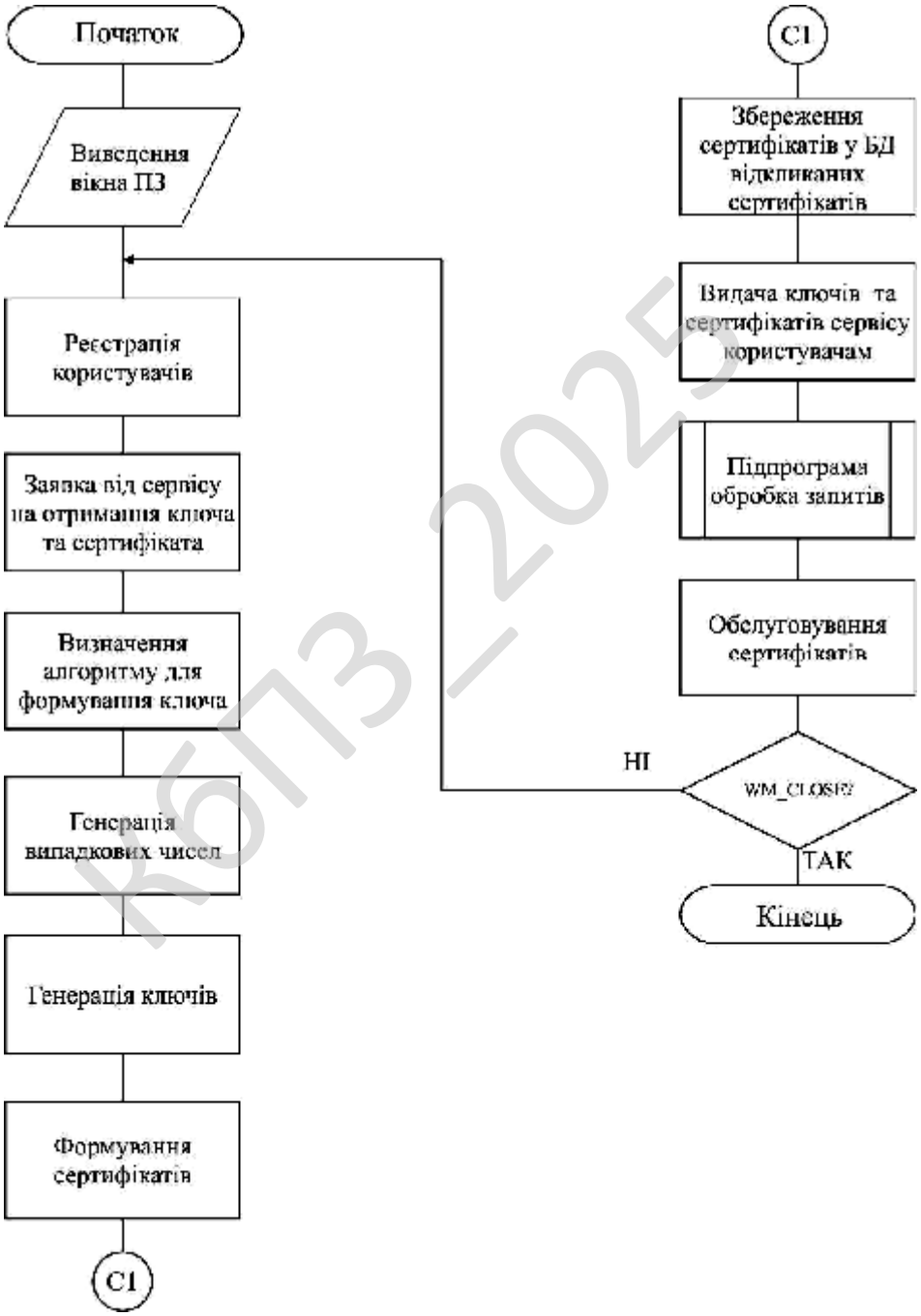

Рисунок 4.1 – Блок схема основної програми

Нижче приведемо частину коду основної програми, яка реалізує ці алгоритми.

У якості алгоритмів шифрування даних у мережі, для яких здійснюється розподіл ключів, обрані алгоритми RSA та DSA.

```
//створення ключів RSA
    private static void Create_RSAKeys()
    {
      CspParameters csp = new CspParameters();
      csp.KeyContainerName = "RSA Тест";
      const int PROV_RSA_FULL = 1;
      csp.ProviderType = PROV_RSA_FULL;
      const int AT_KEYEXCHANGE = 1;
// const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_KEYEXCHANGE;
      _RSACryptoServiceProvider rsa =
          new _RSACryptoServiceProvider(1024, csp);
      rsa.PersistKeyInCsp = false;
```

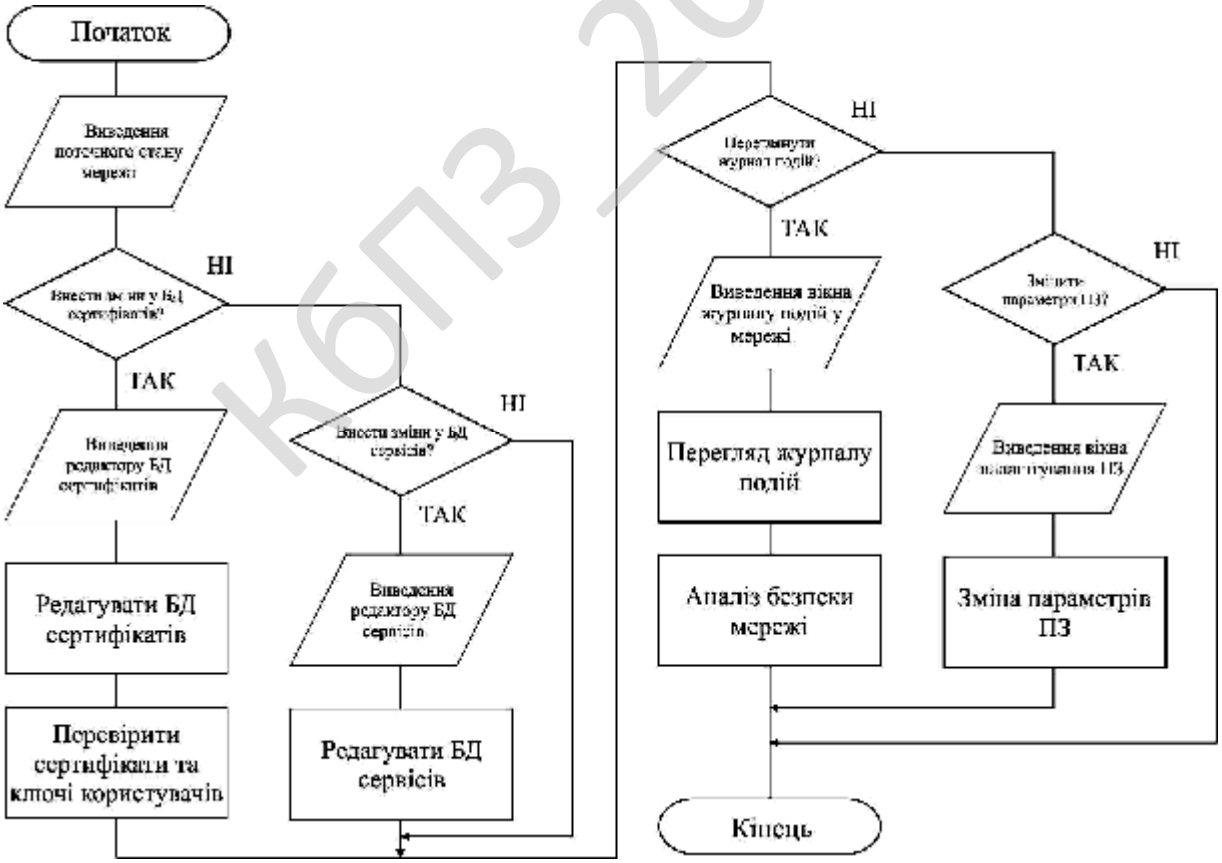

Рисунок 4.2 – Блок схема підпрограми

```
// Ключ шифрування
      AsnKeyBuilder.AsnMessage key = null;
// Таємний ключ
      _RSAParameters privateKey = rsa.ExportParameters(true);
      key = AsnKeyBuilder.PrivateKeyToPKCS8(privateKey);
      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("private.rsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }
//тестування ключів DSA
    private static void TestDsaKeys()
    {
      CreateDsaKeys();
      LoadDsaPrivateKey();
      LoadDsaPublicKey();
    }
//тестування ключів RSA
    private static void Test_RSAKeys()
    {
      Create_RSAKeys();
      Load_RSAPrivateKey();
      Load_RSAPublicKey();
    }
// Відкритий ключ
      _RSAParameters publicKey = rsa.ExportParameters(false);
      key = AsnKeyBuilder.PublicKeyToX509(publicKey);
      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("public.rsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }

      rsa.Clear();
    }
// Створення ключів DSA
    private static void CreateDsaKeys()
    {
      CspParameters csp = new CspParameters();
      csp.KeyContainerName = "DSA Тест";
```

```
      const int PROV_DSS_DH = 13;
      csp.ProviderType = PROV_DSS_DH;
// Не використовуйте AT_EXCHANGE для створення. Це алгоритм підпису
      const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_SIGNATURE;
      DSACryptoServiceProvider dsa =
          new DSACryptoServiceProvider(1024, csp);
      dsa.PersistKeyInCsp = false;
// Ключ шифрування
      AsnKeyBuilder.AsnMessage key = null;
// Таємний ключ
      DSAParameters privateKey = dsa.ExportParameters(true);
      key = AsnKeyBuilder.PrivateKeyToPKCS8(privateKey);
      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("private.dsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }
// Редагування таємного ключа RSA
    private static void Load_RSAPrivateKey()
    {
//
// Редагування Таємного ключа
//   PKCS#8 формат
//
      AsnKeyParser keyParser =
        new AsnKeyParser("private.rsa.cs.ber");
      _RSAParameters privateKey = keyParser.Parse_RSAPrivateKey();
//
// Ініціалізація CSP
//   Подавляє створення нового ключа
//
      CspParameters csp = new CspParameters();
      csp.KeyContainerName = "RSA  Тест";
      const int PROV_RSA_FULL = 1;
      csp.ProviderType = PROV_RSA_FULL;
      const int AT_KEYEXCHANGE = 1;
// const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_KEYEXCHANGE;
//
// Ініціалізація криптопровайдер
```

```
//
      _RSACryptoServiceProvider rsa =
        new _RSACryptoServiceProvider(csp);
      rsa.PersistKeyInCsp = false;
//
//
//
      rsa.ImportParameters(privateKey);

      rsa.Clear();
    }
// Відкритий ключ
      DSAParameters publicKey = dsa.ExportParameters(false);
      key = AsnKeyBuilder.PublicKeyToX509(publicKey);
      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("public.dsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }

      dsa.Clear();
    }
// Редагування таємного ключа DSA
    private static void LoadDsaPrivateKey()
    {
// Редагування таємного ключа
//   PKCS#8 формат
      AsnKeyParser keyParser =
        new AsnKeyParser("private.dsa.cs.ber");
      DSAParameters privateKey = keyParser.ParseDSAPrivateKey();
// Ініціалізація CSP
//   Подавляє створення нового ключа
      CspParameters csp = new CspParameters();
      csp.KeyContainerName = "DSA Тест";
// Не використовуйтеPROV_DSS_DH для редагування.
// const int PROV_DSS_DH = 13;
      const int PROV_DSS = 3;
      csp.ProviderType = PROV_DSS;
// const int AT_EXCHANGE = 1;
      const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_SIGNATURE;
```

```
//
// Ініціалізація криптопровайдера
//
      DSACryptoServiceProvider dsa =
        new DSACryptoServiceProvider(csp);
      dsa.PersistKeyInCsp = false;
//
//
      dsa.ImportParameters(privateKey);
      dsa.Clear();
    }
// Редагування відкритого ключа DSA
    private static void LoadDsaPublicKey()
    {
//
// Редагування відкритого ключа
//   X.509 формат
//
      AsnKeyParser keyParser =
        new AsnKeyParser("public.dsa.cs.ber");
      DSAParameters publicKey = keyParser.ParseDSAPublicKey();
//
// Ініціалізація CSP
//   Подавляє створення нового ключа
//
      CspParameters csp = new CspParameters();
// const int PROV_DSS_DH = 13;
      const int PROV_DSS = 3;
      csp.ProviderType = PROV_DSS;
      const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_SIGNATURE;
      csp.KeyContainerName = "DSA Test (OK to Delete)";
//
// Ініціалізація криптопровайдера
//
      DSACryptoServiceProvider dsa =
        new DSACryptoServiceProvider(csp);
      dsa.PersistKeyInCsp = false;
//
//
//
      dsa.ImportParameters(publicKey);
```

```
      dsa.Clear();
    }
// Редагування відкритого ключа RSA
    private static void Load_RSAPublicKey()
    {
//
// Редагування Відкритого ключа
//   X.509 формат
//
      AsnKeyParser keyParser =
        new AsnKeyParser("public.rsa.cs.ber");
      _RSAParameters publicKey = keyParser.Parse_RSAPublicKey();
//
// Ініціалізація CSP
//   Подавляє створення нового ключа
//
      CspParameters csp = new CspParameters();
      csp.KeyContainerName = "RSA  Тест";
      const int PROV_RSA_FULL = 1;
      csp.ProviderType = PROV_RSA_FULL;
      const int AT_KEYEXCHANGE = 1;
// const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_KEYEXCHANGE;
//
// Ініціалізація криптопровайдера
//
      _RSACryptoServiceProvider rsa =
        new _RSACryptoServiceProvider(csp);
      rsa.PersistKeyInCsp = false;
//
      rsa.ImportParameters(publicKey);
      rsa.Clear();
    }
  }
```


4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Sinople – симетричний блоковий криптоалгоритм, побудований на основі незбалансованої «мережі Фейстеля». Алгоритм розроблено у 2003 році.

Основні вимоги до алгоритму при його розробці:

– Можливість програмної і апаратної реалізації.

– Висока швидкість.

– Простота.

– Низькі вимоги до пам'яті.

– Високий рівень безпеки.

Алгоритм заснований на 32-розрядних операціях і має 64 раунду, серед яких два типи – C і D. D раунди спроектовані для досягнення максимальної дифузії, C раунди – для досягнення перемішування. F-функція D раунду використовує один з елементів блоку даних (D[3]) та поточного з'єднання (K[r]) для трансформації 3-х елементів блоку даних. F-функція C раунду, навпаки, використовує перші три елемента блоку даних і поточний з'єднання (K[r]) для трансформації останнього елемента блоку даних (D[3]). Раунди D-типу виконуються до раундів C-типу. Додавання ключів з даними проводиться тільки через таблиці замін. Операції XOR (додавання по модулю 2) обов'язково поєднуються з операціями ADD (додавання по модулю 2^{32}).

Таблиці замін спочатку запозичені з алгоритму MARS і містять 512 32-розрядних елементів, проте були жорстко проаналізовано на предмет посилення.

Ключове розклад було спроектовано з урахуванням вимог:

– Простота

– Використовується та ж процедура, що і при шифруванні та розшифрування

– Установка ключа займає менше часу, ніж зашифрування

– Виключення еквівалентних ключів

– Виключення слабких ключів

Алгоритм, згідно із заявою авторів, стійкий до лінійного і диференціального аналізу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні розділи:

– Меню.

– Функціональні кнопки.

– Ресурси.

У вищезазначених розділах реалізується наступний функціонал:

– З’єднання.

– Розподіл ключів.

– Сертифікати.

– Ресурси.

– Журнал.

– Параметри.

– Довідка.

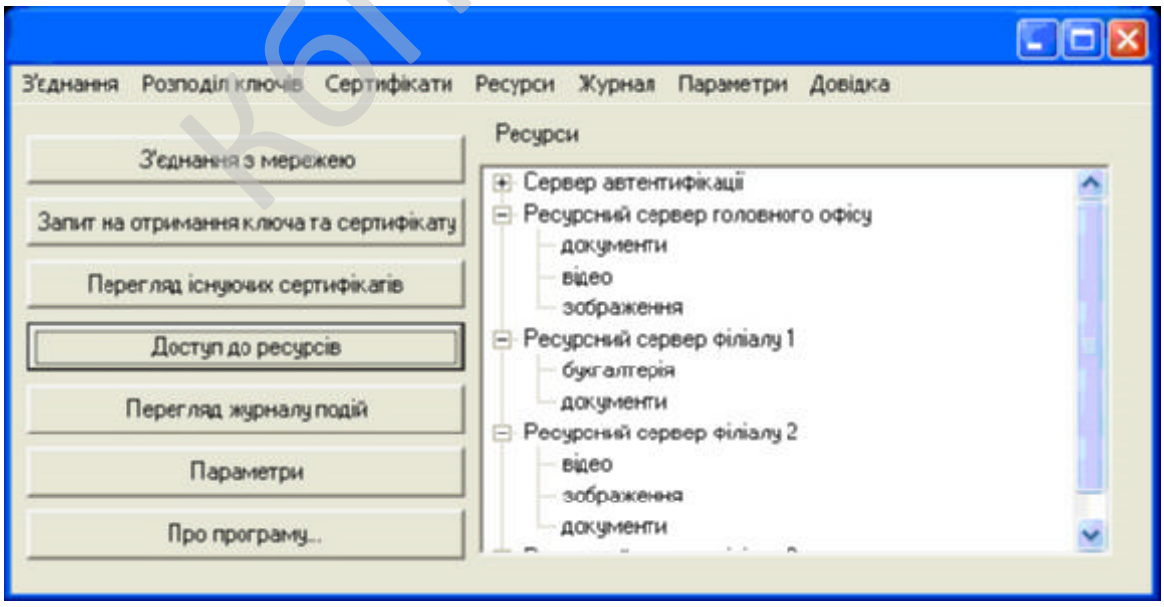

Рисунок 5.1– Основне вікно програми

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

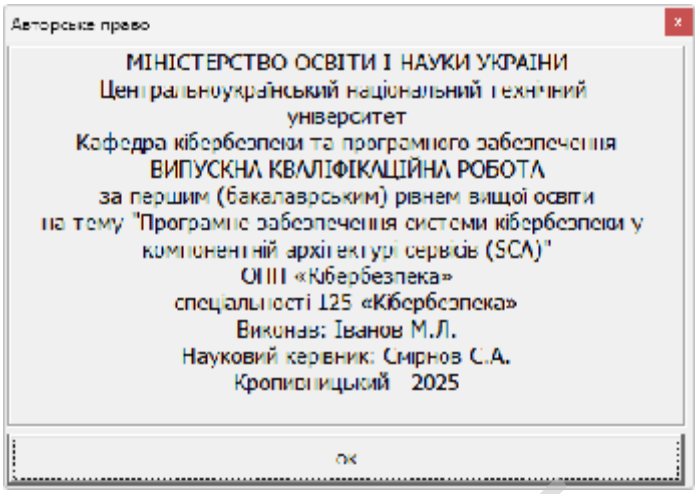

Рисунок 5.2 – Авторське право

Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми. В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ - «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки у компонентній архітектурі сервісів (SCA).

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем у компонентній архітектурі сервісів (SCA).

– Досліджена система у компонентній архітектурі сервісів (SCA).

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки у компонентній архітектурі сервісів (SCA).

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання у компонентній архітектурі сервісів (SCA).

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 Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки у компонентній архітектурі сервісів (SCA). 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 кібербезпеки 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 Sinople.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Massimo Bertaccini. Cryptography Algorithms. Packt Publishing. 2022. 358 p.

2. Alyssa Miller. Cybersecurity Career Guide. Manning Publications. 2022. 368 p.

3. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

4. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

5. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

6. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

7. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

8. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

9. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

10. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

11. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

12. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

13. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

14. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

15. Kuznetsov, O., Frontoni, E., Kandiy, S., Smirnova, T., Prokopov, S., Bilanovych, A. «New Cost Function for S-boxes Generation by Simulated Annealing Algorithm». Lecture Notes on Data Engineering and Communications Technologies, 2023. vol 180. pp. 310-320. Springer, Cham.

16. Kuznetsov, O., Frontoni, E., Kandiy, S., Smirnov, O., Ulianovska, Y., Kobylianska, O. «Heuristic Search for Nonlinear Substitutions for Cryptographic Applications». *Lecture Notes on Data Engineering and Communications Technologies,* 2023. vol 180. Springer, Cham. pp. 288-298.

17. Kuznetsov, O., Kuznetsova, Y., Smirnov, O., Kostenko, O., Zvieriev, V. «Evaluating Hashing Algorithms in the Age of ASIC Resistance». *CEUR Workshop Proceedings*, 2023, 3628, pp. 93-105.

18. Kuznetsov O., Frontoni E., Kuznetsova Ye., Smirnov O., Chevardin V. «Achieving Enhanced Security in Biometric Authentication: A Rigorous Analysis of Code-Based Fuzzy Extractor». *CEUR Workshop Proceedings,* Volume 3624, 2023, pp. 330-339.

19. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

20. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

21. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

22. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

23. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

24. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

25. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

26. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

27. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

28. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

29. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

30. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

31. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

32. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

33. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

34. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

35. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

36. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

37. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

38. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

39. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

40. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

41. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

42. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

43. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

44. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

45. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

46. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

47. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

48. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

49. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and*

Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.399-405.

50. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

51. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

52. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки у компонентній архітектурі сервісів (SCA).

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 57-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA).

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) 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 кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки у компонентній архітектурі сервісів (SCA);

– 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, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

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 Visual C#.

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 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 63 аркуші.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 2.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки у компонентній архітектурі сервісів (SCA)

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 91

Літера: РП

Кропивницький – 2025 року

Файл Program.cs – основна програма

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

// [assembly: CLSCompliant(true)]
namespace CSInteropKeys
{
  class Program
  {
    internal static void Main(/* string[] аргументи */)
    {
      TestRsaKeys();

      TestDsaKeys();
    }
//тестування ключів DSA
    private static void TestDsaKeys()
    {
      CreateDsaKeys();

      LoadDsaPrivateKey();

      LoadDsaPublicKey();
    }
//тестування ключів RSA
    private static void TestRsaKeys()
    {
      CreateRsaKeys();

      LoadRsaPrivateKey();

      LoadRsaPublicKey();
    }
//створення ключів RSA
    private static void CreateRsaKeys()
    {
      CspParameters csp = new CspParameters();

      csp.KeyContainerName = "RSA Тест";

      const int PROV_RSA_FULL = 1;
      csp.ProviderType = PROV_RSA_FULL;

      const int AT_KEYEXCHANGE = 1;
      // const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_KEYEXCHANGE;

      RSACryptoServiceProvider rsa =
          new RSACryptoServiceProvider(1024, csp);
      rsa.PersistKeyInCsp = false;

      // Ключ шифрування
      AsnKeyBuilder.AsnMessage key = null;

      // Таємний ключ
      RSAParameters privateKey = rsa.ExportParameters(true);
      key = AsnKeyBuilder.PrivateKeyToPKCS8(privateKey);

      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("private.rsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }
```

```
      // Відкритий ключ
      RSAParameters publicKey = rsa.ExportParameters(false);
      key = AsnKeyBuilder.PublicKeyToX509(publicKey);

      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("public.rsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }


      rsa.Clear();
    }
// Створення ключів Dsa
    private static void CreateDsaKeys()
    {
      CspParameters csp = new CspParameters();

      csp.KeyContainerName = "DSA Тест";

      const int PROV_DSS_DH = 13;
      csp.ProviderType = PROV_DSS_DH;

      // Не використовуйте AT_EXCHANGE для створення. Це
      //  алгоритм підпису
      const int AT_SIGNATURE = 2;
      csp.KeyNumber = AT_SIGNATURE;

      DSACryptoServiceProvider dsa =
          new DSACryptoServiceProvider(1024, csp);
      dsa.PersistKeyInCsp = false;

      // Ключ шифрування
      AsnKeyBuilder.AsnMessage key = null;

      // Таємний ключ
      DSAParameters privateKey = dsa.ExportParameters(true);
      key = AsnKeyBuilder.PrivateKeyToPKCS8(privateKey);

      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("private.dsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }

      // Відкритий ключ
      DSAParameters publicKey = dsa.ExportParameters(false);
      key = AsnKeyBuilder.PublicKeyToX509(publicKey);

      using (BinaryWriter writer = new BinaryWriter(
          new FileStream("public.dsa.cs.ber", FileMode.Create,
              FileAccess.ReadWrite)))
      {
        writer.Write(key.GetBytes());
      }


      dsa.Clear();
    }

    private static void LoadDsaPrivateKey()
    {
      //
      // Редагування таємного ключа
      //   PKCS#8 формат
      //
```

```
  AsnKeyParser keyParser =
    new AsnKeyParser("private.dsa.cs.ber");

  DSAParameters privateKey = keyParser.ParseDSAPrivateKey();

  //
  // Ініціалізація CSP
  //   Подавляє створення нового ключа
  //
  CspParameters csp = new CspParameters();
  csp.KeyContainerName = "DSA Тест";

  // Не використовуйтеPROV_DSS_DH для редагування.
  // const int PROV_DSS_DH = 13;
  const int PROV_DSS = 3;
  csp.ProviderType = PROV_DSS;

  // const int AT_EXCHANGE = 1;
  const int AT_SIGNATURE = 2;
  csp.KeyNumber = AT_SIGNATURE;

  //
  // Ініціалізація криптопровайдера
  //
  DSACryptoServiceProvider dsa =
    new DSACryptoServiceProvider(csp);
  dsa.PersistKeyInCsp = false;

  //
  //
  dsa.ImportParameters(privateKey);

  dsa.Clear();
}

private static void LoadDsaPublicKey()
{
  //
  // Редагування відкритого ключа
  //   X.509 формат
  //
  AsnKeyParser keyParser =
    new AsnKeyParser("public.dsa.cs.ber");

  DSAParameters publicKey = keyParser.ParseDSAPublicKey();

  //
  // Ініціалізація CSP
  //   Подавляє створення нового ключа
  //
  CspParameters csp = new CspParameters();

  // const int PROV_DSS_DH = 13;
  const int PROV_DSS = 3;
  csp.ProviderType = PROV_DSS;

  const int AT_SIGNATURE = 2;
  csp.KeyNumber = AT_SIGNATURE;

  csp.KeyContainerName = "DSA Test (OK to Delete)";

  //
  // Ініціалізація криптопровайдера
  //
  DSACryptoServiceProvider dsa =
    new DSACryptoServiceProvider(csp);
  dsa.PersistKeyInCsp = false;

  //
```

```
  //
  //
  dsa.ImportParameters(publicKey);


  dsa.Clear();
}

private static void LoadRsaPrivateKey()
{
  //
  // Редагування Таємного ключа
  //   PKCS#8 формат
  //
  AsnKeyParser keyParser =
    new AsnKeyParser("private.rsa.cs.ber");

  RSAParameters privateKey = keyParser.ParseRSAPrivateKey();

  //
  // Ініціалізація CSP
  //   Подавляє створення нового ключа
  //
  CspParameters csp = new CspParameters();
  csp.KeyContainerName = "RSA  Тест";

  const int PROV_RSA_FULL = 1;
  csp.ProviderType = PROV_RSA_FULL;

  const int AT_KEYEXCHANGE = 1;
  // const int AT_SIGNATURE = 2;
  csp.KeyNumber = AT_KEYEXCHANGE;

  //
  // Ініціалізація криптопровайдер
  //
  RSACryptoServiceProvider rsa =
    new RSACryptoServiceProvider(csp);
  rsa.PersistKeyInCsp = false;

  //
  //
  //
  rsa.ImportParameters(privateKey);


  rsa.Clear();
}

private static void LoadRsaPublicKey()
{
  //
  // Редагування Відкритого ключа
  //   X.509 формат
  //
  AsnKeyParser keyParser =
    new AsnKeyParser("public.rsa.cs.ber");

  RSAParameters publicKey = keyParser.ParseRSAPublicKey();

  //
  // Ініціалізація CSP
  //   Подавляє створення нового ключа
  //
  CspParameters csp = new CspParameters();
  csp.KeyContainerName = "RSA  Тест";

  const int PROV_RSA_FULL = 1;
  csp.ProviderType = PROV_RSA_FULL;
```

```
        const int AT_KEYEXCHANGE = 1;
        // const int AT_SIGNATURE = 2;
        csp.KeyNumber = AT_KEYEXCHANGE;

        //
        // Ініціалізація криптопровайдера
        //
        RSACryptoServiceProvider rsa =
          new RSACryptoServiceProvider(csp);
        rsa.PersistKeyInCsp = false;

        //
        //
        //
        rsa.ImportParameters(publicKey);


        rsa.Clear();
      }
    }
}
```

Файл AsnKeyBuilder.cs – створення ключів

```
using System;
using System.Text;
using System.Diagnostics;
using System.Globalization;
using System.Collections.Generic;
using System.Security.Cryptography;

/// <summary>
/// Файл у якому реалізований клас, який працює на основі стандарту ASN.1, кодує
дані за допомогою PKCS#8 PrivateKeyInfo  та X.509 PublicKeyInfo.
Використовується для експорту ключів RSA або DSA. для використання на Java або
інших не-XML мовах програмування .
/// </summary>
///
/// <remarks>
/// Дубина Віталій Олександрович
/// </remarks>

namespace CSInteropKeys
{
  class AsnKeyBuilder
  {
    internal class AsnMessage
    {
      private byte[] m_octets;
      private String m_format;

      internal int Length
      {
        get
        {
          if (null == m_octets) { return 0; }
          return m_octets.Length;
        }
        // set { m_length = value; }
      }

      internal AsnMessage(byte[] octets, String формат)
      {
        m_octets = octets;
        m_format = формат;
      }

      internal byte[] GetBytes()
      {
        if (null == m_octets)
        { return new byte[] { }; }

        return m_octets;
      }
      internal String GetFormat()
      { return m_format; }
    }

    internal class AsnType
    {
      // Конструктори
      // Не виставляється по замовчуванню - повині бути визначені теги та дані

      public AsnType(byte tag, byte octet)
      {
        m_raw = false;
        m_tag = new byte[] { tag };
        m_octets = new byte[] { octet };
      }
```

```
public AsnType(byte tag, byte[] octets)
{
  m_raw = false;
  m_tag = new byte[] { tag };
  m_octets = octets;
}

public AsnType(byte tag, byte[] length, byte[] octets)
{
  m_raw = true;
  m_tag = new byte[] { tag };
  m_length = length;
  m_octets = octets;
}

private bool m_raw;

private bool Raw
{
  get { return m_raw; }
  set { m_raw = value; }
}

// Встановлення та зчитування
private byte[] m_tag;
public byte[] Tag
{
  get
  {
    if (null == m_tag)
      return EMPTY;
    return m_tag;
  }
  // set { m_tag = value; }
}

private byte[] m_length;
public byte[] Length
{
  get
  {
    if (null == m_length)
      return EMPTY;
    return m_length;
  }
  // set { m_length = value; }
}

private byte[] m_octets;
public byte[] Octets
{
  get
  {
    if (null == m_octets)
    { return EMPTY; }
    return m_octets;
  }
  set
  { m_octets = value; }
}

// методи
internal byte[] GetBytes()
{
  // Створення вихідних параметрів користувачів
  // повертає байти....
  if (true == m_raw)
  {
    return Concatenate(
```

```
      new byte[][] { m_tag, m_length, m_octets }
    );
  }

  SetLength();

  // Спеціальний випадок
  // Пусте значення не має довжини
  if (0x05 == m_tag[0])
  {
    return Concatenate(
      new byte[][] { m_tag, m_octets }
    );
  }

  return Concatenate(
    new byte[][] { m_tag, m_length, m_octets }
  );
}

private void SetLength()
{
  if (null == m_octets)
  {
    m_length = ZERO;
    return;
  }

  // Спеціальний випадок
  // Пусте значення не має довжини
  if (0x05 == m_tag[0])
  {
    m_length = EMPTY;
    return;
  }

  byte[] length = null;

  // Length: 0 <= l < 0x80
  if (m_octets.Length < 0x80)
  {
    length = new byte[1];
    length[0] = (byte)m_octets.Length;
  }
  // 0x80 < length <= 0xFF
  else if (m_octets.Length <= 0xFF)
  {
    length = new byte[2];
    length[0] = 0x81;
    length[1] = (byte)((m_octets.Length & 0xFF));
  }

  //
  // Ми повинні майже ніколи не бачити їх...
  //

  // 0xFF < length <= 0xFFFF
  else if (m_octets.Length <= 0xFFFF)
  {
    length = new byte[3];
    length[0] = 0x82;
    length[1] = (byte)((m_octets.Length & 0xFF00) >> 8);
    length[2] = (byte)((m_octets.Length & 0xFF));
  }

  // 0xFFFF < length <= 0xFFFFFF
  else if (m_octets.Length <= 0xFFFFFF)
  {
    length = new byte[4];
```

```
      length[0] = 0x83;
      length[1] = (byte)((m_octets.Length & 0xFF0000) >> 16);
      length[2] = (byte)((m_octets.Length & 0xFF00) >> 8);
      length[3] = (byte)((m_octets.Length & 0xFF));
    }
    // 0xFFFFFF < length <= 0xFFFFFFFF
    else
    {
      length = new byte[5];
      length[0] = 0x84;
      length[1] = (byte)((m_octets.Length & 0xFF000000) >> 24);
      length[2] = (byte)((m_octets.Length & 0xFF0000) >> 16);
      length[3] = (byte)((m_octets.Length & 0xFF00) >> 8);
      length[4] = (byte)((m_octets.Length & 0xFF));
    }

    m_length = length;
  }

  private byte[] Concatenate(byte[][] values)
  {
    // Пусте значення на вході, пусте значення на виході
    if (IsEmpty(values))
      return new byte[] { };

    int length = 0;
    foreach (byte[] b in values)
    {
      if (null != b) length += b.Length;
    }

    byte[] cated = new byte[length];

    int current = 0;
    foreach (byte[] b in values)
    {
      if (null != b)
      {
        Array.Copy(b, 0, cated, current, b.Length);
        current += b.Length;
      }
    }

    return cated;
  }
};

private static byte[] ZERO = new byte[] { 0 };
private static byte[] EMPTY = new byte[] { };

// PublicKeyInfo (X.509 сумісно) повідомлення
/// <summary>
/// Повертає AsnMessage представлене у X.509 PublicKeyInfo.
/// </summary>
/// <param name="publicKey"> DSA ключ шифрування.</param>
/// <returns>Повертає AsnType представлене у
/// X.509 PublicKeyInfo.</returns>
/// <seealso cref="PrivateKeyToPKCS8(DSAParameters)"/>
/// <seealso cref="PrivateKeyToPKCS8(RSAParameters)"/>
/// <seealso cref="PublicKeyToX509(RSAParameters)"/>
internal static AsnMessage PublicKeyToX509(DSAParameters publicKey)
{
  // Значення типів не повинні бути пусті
  // Debug.Assert(null != publicKey);

  /* *
  * SEQUENCE              // PrivateKeyInfo
  * +- SEQUENCE           // AlgorithmIdentifier
  * |  +- OID             // 1.2.840.10040.4.1
```

```
  * |  +- SEQUENCE        // DSS-параметри (Опційні параметри)
  * |    +- INTEGER (P)
  * |    +- INTEGER (Q)
  * |    +- INTEGER (G)
  * +- BITSTRING          // PublicKey
  *    +- INTEGER(Y)      // DSAPublicKey Y
  * */

  // DSA параметри
  AsnType p = CreateIntegerPos(publicKey.P);
  AsnType q = CreateIntegerPos(publicKey.Q);
  AsnType g = CreateIntegerPos(publicKey.G);

  // послідовність- DSA-параметрів
  AsnType dssParams = CreateSequence(new AsnType[] { p, q, g });

  // OID - пакує 1.2.840.10040.4.1
  //   { 0x2A, 0x86, 0x48, 0xCE, 0x38, 0x04, 0x01 }
  AsnType oid = CreateOid("1.2.840.10040.4.1");

  // послідовність
  AsnType algorithmID = CreateSequence(new AsnType[] { oid, dssParams });

  // Відкритий ключ Y
  AsnType y = CreateIntegerPos(publicKey.Y);
  AsnType key = CreateBitString(y);

  // послідовність'A'
  AsnType publicKeyInfo =
    CreateSequence(new AsnType[] { algorithmID, key });

  return new AsnMessage(publicKeyInfo.GetBytes(), "X.509");
}

// PublicKeyInfo (X.509 сумісно) повідомлення
/// <summary>
/// Повертає AsnMessage представлене у X.509 PublicKeyInfo.
/// </summary>
/// <param name="publicKey"> RSA ключ шифрування.</param>
/// <returns>Повертає AsnType представлене у
/// X.509 PublicKeyInfo.</returns>
/// <seealso cref="PrivateKeyToPKCS8(DSAParameters)"/>
/// <seealso cref="PrivateKeyToPKCS8(RSAParameters)"/>
/// <seealso cref="PublicKeyToX509(DSAParameters)"/>
internal static AsnMessage PublicKeyToX509(RSAParameters publicKey)
{
  // Значення типів не повинні бути пусті
  // Debug.Assert(null != publicKey);

  /* *
  * SEQUENCE              // PrivateKeyInfo
  * +- SEQUENCE           // AlgorithmIdentifier
  *    +- OID             // 1.2.840.113549.1.1.1
  *    +- Null            // Опційні параметри
  * +- BITSTRING          // PrivateKey
  *    +- SEQUENCE        // RSAPrivateKey
  *       +- INTEGER(N)   // N
  *       +- INTEGER(E)   // E
  * */

  // OID - пакує 1.2.840.113549.1.1.1
  //   { 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01 }
  AsnType oid = CreateOid("1.2.840.113549.1.1.1");
  AsnType algorithmID =
    CreateSequence(new AsnType[] { oid, CreateNull() });

  AsnType n = CreateIntegerPos(publicKey.Modulus);
  AsnType e = CreateIntegerPos(publicKey.Exponent);
  AsnType key = CreateBitString(
```

```
    CreateSequence(new AsnType[] { n, e })
  );

  AsnType publicKeyInfo =
    CreateSequence(new AsnType[] { algorithmID, key });

  return new AsnMessage(publicKeyInfo.GetBytes(), "X.509");
}

// PKCS #8, частина 6 (PrivateKeyInfo) повідомлення
// !!!!!!!!!!!!!!! Незашифрованому вигляді !!!!!!!!!!!!!!!
/// <summary>
/// повертає AsnMessage представлене у незашифрованому вигляді
/// PKCS #8 PrivateKeyInfo.
/// </summary>
/// <param name="privateKey"> DSA ключ шифрування.</param>
/// <returns>Повертає AsnType представлене у незашифрованому вигляді
/// PKCS #8 PrivateKeyInfo.</returns>
/// <seealso cref="PrivateKeyToPKCS8(RSAParameters)"/>
/// <seealso cref="PublicKeyToX509(DSAParameters)"/>
/// <seealso cref="PublicKeyToX509(RSAParameters)"/>
internal static AsnMessage PrivateKeyToPKCS8(DSAParameters privateKey)
{
  // Значення типів не повинні бути пусті
  // Debug.Assert(null != privateKey);

  /* *
  * SEQUENCE              // PrivateKeyInfo
  * +- INTEGER(0)         // Версія (v2010)
  * +- SEQUENCE           // AlgorithmIdentifier
  * |  +- OID             // 1.2.840.10040.4.1
  * |  +- SEQUENCE        // DSS-параметри (Опційні параметри)
  * |    +- INTEGER (P)
  * |    +- INTEGER (Q)
  * |    +- INTEGER (G)
  * +- OCTETSTRING        // PrivateKey
  *    +- INTEGER(X)   // DSAPrivateKey X
  * */

       AsnType version = CreateInteger(ZERO);

  // Область параметрів
  AsnType p = CreateIntegerPos(privateKey.P);
  AsnType q = CreateIntegerPos(privateKey.Q);
  AsnType g = CreateIntegerPos(privateKey.G);

  AsnType dssParams = CreateSequence(new AsnType[] { p, q, g });

  // OID - пакує 1.2.840.10040.4.1
  //   { 0x2A, 0x86, 0x48, 0xCE, 0x38, 0x04, 0x01 }
  AsnType oid = CreateOid("1.2.840.10040.4.1");

  // AlgorithmIdentifier
  AsnType algorithmID = CreateSequence(new AsnType[] { oid, dssParams });

  // Таємний ключ X
  AsnType x = CreateIntegerPos(privateKey.X);
  AsnType key = CreateOctetString(x);

  // послідовність
  AsnType privateKeyInfo =
    CreateSequence(new AsnType[] { version, algorithmID, key });

  return new AsnMessage(privateKeyInfo.GetBytes(), "PKCS#8");
}

// PKCS #8, параграф  6 (PrivateKeyInfo) повідомлення
// !!!!!!!!!!!!!!! Незашифрований вигляд !!!!!!!!!!!!!!!
/// <summary>
```

```
    /// повертає AsnMessage представлене у незашифрованому вигляді
    /// PKCS #8 PrivateKeyInfo.
    /// </summary>
    /// <param name="privateKey">The RSA ключ шифрування.</param>
    /// <returns>Повертає AsnType представлене у незашифрованому вигляді
    /// PKCS #8 PrivateKeyInfo.</returns>
    /// <seealso cref="PrivateKeyToPKCS8(DSAParameters)"/>
    /// <seealso cref="PublicKeyToX509(DSAParameters)"/>
    /// <seealso cref="PublicKeyToX509(RSAParameters)"/>
    internal static AsnMessage PrivateKeyToPKCS8(RSAParameters privateKey)
    {
      // Значення типів не повинні бути пусті
      // Debug.Assert(null != privateKey);

      /* *
      * SEQUENCE                  // PublicKeyInfo
      * +- INTEGER(0)             // Версія - 0 (v2010)
      * +- SEQUENCE               // AlgorithmIdentifier
      *    +- OID                 // 1.2.840.113549.1.1.1
      *    +- NULL                // Опційні параметри
      * +- OCTETSTRING            // PrivateKey
      *    +- SEQUENCE            // RSAPrivateKey
      *       +- INTEGER(0)       // Версія - 0 (v2010)
      *       +- INTEGER(N)
      *       +- INTEGER(E)
      *       +- INTEGER(D)
      *       +- INTEGER(P)
      *       +- INTEGER(Q)
      *       +- INTEGER(DP)
      *       +- INTEGER(DQ)
      *       +- INTEGER(Inv Q)
      * */

      AsnType n = CreateIntegerPos(privateKey.Modulus);
      AsnType e = CreateIntegerPos(privateKey.Exponent);
      AsnType d = CreateIntegerPos(privateKey.D);
      AsnType p = CreateIntegerPos(privateKey.P);
      AsnType q = CreateIntegerPos(privateKey.Q);
      AsnType dp = CreateIntegerPos(privateKey.DP);
      AsnType dq = CreateIntegerPos(privateKey.DQ);
      AsnType iq = CreateIntegerPos(privateKey.InverseQ);

      // Версія - 0 (v2010)
      AsnType version = CreateInteger(new byte[] { 0 });

      // octstring = OCTETSTRING(SEQUENCE(INTEGER(0)INTEGER(N)...))
      AsnType key = CreateOctetString(
        CreateSequence(new AsnType[] { version, n, e, d, p, q, dp, dq, iq })
      );

      // OID - пакує 1.2.840.113549.1.1.1
      //   { 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01 }
      AsnType algorithmID = CreateSequence(new AsnType[] {
CreateOid("1.2.840.113549.1.1.1"), CreateNull() }
      );

      // PrivateKeyInfo
      AsnType privateKeyInfo =
        CreateSequence(new AsnType[] { version, algorithmID, key });

      return new AsnMessage(privateKeyInfo.GetBytes(), "PKCS#8");
    }

    /// <summary>
    /// <para>Впорядкована сукупність одного або більше типів.
    /// Повертає AsnType представлене у ASN.1 зашифрованій послідовності.</para>
    /// <para>Якщо AsnType містить пусте значення, пуста послідовність (довжина
0)
    /// повертається.</para>
```

```
    /// </summary>
    /// <param name="value"> AsnType складається з
    /// одиночних зашифрованих значень .</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// зашифрованій послідовності.</returns>
    /// <seealso cref="CreateSet(AsnType)"/>
    /// <seealso cref="CreateSet(AsnType[])"/>
    /// <seealso cref="CreateSetOf(AsnType)"/>
    /// <seealso cref="CreateSetOf(AsnType[])"/>
    /// <seealso cref="CreateSequence(AsnType)"/>
    /// <seealso cref="CreateSequence(AsnType[])"/>
    /// <seealso cref="CreateSequenceOf(AsnType)"/>
    /// <seealso cref="CreateSequenceOf(AsnType[])"/>
    internal static AsnType CreateSequence(AsnType value)
    {
      // Повинно дорівнюватись 1...
      Debug.Assert(!IsEmpty(value));

      // Вимагає одного або більше значень
      if (IsEmpty(value))
      { throw new ArgumentException("Послідовність вимагає наявності хоча б
одного значення"); }

      // послідовність: Tag 0x30 (16, Universal, Constructed)
      return new AsnType(0x30, value.GetBytes());
    }

    /// <summary>
    /// <para>Впорядкована сукупність одного або більше типів.
    /// Повертає AsnType представлене у ASN.1 зашифрованій послідовності.</para>
    /// <para> Якщо AsnType містить пусте значення,
    /// пуста послідовність (довжина дорівнює 0) повертається.</para>
    /// </summary>
    /// <param name="values">Массив AsnType складається з
    /// значень, які потрібно шифрувати.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує Set.</returns>
    /// <seealso cref="CreateSet(AsnType)"/>
    /// <seealso cref="CreateSet(AsnType[])"/>
    /// <seealso cref="CreateSetOf(AsnType)"/>
    /// <seealso cref="CreateSetOf(AsnType[])"/>
    /// <seealso cref="CreateSequence(AsnType)"/>
    /// <seealso cref="CreateSequence(AsnType[])"/>
    /// <seealso cref="CreateSequenceOf(AsnType)"/>
    /// <seealso cref="CreateSequenceOf(AsnType[])"/>
    internal static AsnType CreateSequence(AsnType[] values)
    {
      // Повинно дорівнюватись 1...
      Debug.Assert(!IsEmpty(values));

      // Вимагає одного або більше значень
      if (IsEmpty(values))
      { throw new ArgumentException("Послідовність вимагає наявності хочаб
одного значення"); }

      // послідовність: Tag 0x30 (16, Universal, Constructed)
      return new AsnType((0x10 | 0x20), Concatenate(values));
    }

    /// <summary>
    /// <para>Упорядкована сукупність нульова, один або більше типів.
    /// Повертає AsnType представлене у ASN.1 зашифрованій послідовності.</para>
    /// <para> Якщо значення AsnType містить пусте значення,
    /// повертається пуста послідовність (довжина дорівнює 0).</para>
    /// </summary>
    /// <param name="value"> AsnType складається з
    /// одиночних зашифрованих значень .</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// зашифрованій послідовності.</returns>
```

```
    /// <seealso cref="CreateSet(AsnType)"/>
    /// <seealso cref="CreateSet(AsnType[])"/>
    /// <seealso cref="CreateSetOf(AsnType)"/>
    /// <seealso cref="CreateSetOf(AsnType[])"/>
    /// <seealso cref="CreateSequence(AsnType)"/>
    /// <seealso cref="CreateSequence(AsnType[])"/>
    /// <seealso cref="CreateSequenceOf(AsnType)"/>
    /// <seealso cref="CreateSequenceOf(AsnType[])"/>
    internal static AsnType CreateSequenceOf(AsnType value)
    {
      // З ASN.1 списку учасників ключового обміну
      if (IsEmpty(value))
      { return new AsnType(0x30, EMPTY); }

      // послідовність: Tag 0x30 (16, Universal, Constructed)
      return new AsnType(0x30, value.GetBytes());
    }

    /// <summary>
    /// <para>Упорядкована сукупність нульова, один або більше типів.
    /// Повертає AsnType представлене у ASN.1 зашифрованій послідовності.</para>
    /// <para> Якщо масив AsnType містить пусте значення абои масив з довжиною
0,
    /// пуста послідовність (довжина дорівнює 0) повертається.</para>
    /// </summary>
    /// <param name="values"> AsnType складається з
    /// значень, які потрібно шифрувати.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// зашифрованій послідовності.</returns>
    /// <seealso cref="CreateSet(AsnType)"/>
    /// <seealso cref="CreateSet(AsnType[])"/>
    /// <seealso cref="CreateSetOf(AsnType)"/>
    /// <seealso cref="CreateSetOf(AsnType[])"/>
    /// <seealso cref="CreateSequence(AsnType)"/>
    /// <seealso cref="CreateSequence(AsnType[])"/>
    /// <seealso cref="CreateSequenceOf(AsnType)"/>
    /// <seealso cref="CreateSequenceOf(AsnType[])"/>
    internal static AsnType CreateSequenceOf(AsnType[] values)
    {
      // З ASN.1 Списку учасників ключового обміну
      if (IsEmpty(values))
      { return new AsnType(0x30, EMPTY); }

      // послідовність: Tag 0x30 (16, Universal, Constructed)
      return new AsnType(0x30, Concatenate(values));
    }

    /// <summary>
    /// <para> завантажується нульова послідовність, один або декілька біт.
Повертає
    /// AsnType представлене у ASN.1 зашифрованого рядка біт.</para>
    /// <para>Якщо октет містить пусте значення або довжина дорівнює 0, він
пустий (довжина дорівнює 0)
    /// повертається рядок біт.</para>
    /// </summary>
    /// <param name="octets"> MSB (big endian) byte[] представлене у
    /// зашифрованому рядку біт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує рядок біт.</returns>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(AsnType[])"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateBitString(byte[] octets)
    {
```

```
      // BitString: Tag 0x03 (3, Universal, Primitive)
      return CreateBitString(octets, 0);
    }

    /// <summary>
    /// <para> завантажується нульова послідовність, один або декілька біт.
Повертає
    /// AsnType представлене у ASN.1 зашифрованого рядка біт.</para>
    /// <para>unusedBits дописується в кінець рядка біт,
    /// не є стартовим рядком біт. unusedBits повинен бути меншим ніж 8
    /// (розмір октету). Описаний у ITU X.680, параграф  32.</para>
    /// <para>Якщо октет містить пусте значення або довжина дорівнює 0, він
пустий (довжина дорівнює 0)
    /// повертається рядок біт.</para>
    /// </summary>
    /// <param name="octets"> MSB (big endian) byte[] представлене у
    /// зашифрованому рядку біт.</param>
    /// <param name="unusedBits">Кількість невикористаних розрядів у
зашифрованому рядку біт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує рядок біт.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(AsnType[])"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateBitString(byte[] octets, uint unusedBits)
    {
      if (IsEmpty(octets))
      {
        // Пустий рядок (октет)
        return new AsnType(0x03, EMPTY);
      }

      if (!(unusedBits < 8))
      { throw new ArgumentException("Невикористаних бітів повинно бути менше
8."); }

      byte[] b = Concatenate(new byte[] { (byte)unusedBits }, octets);
      // BitString: Tag 0x03 (3, Universal, Primitive)
      return new AsnType(0x03, b);
    }

    /// <summary>
    /// Завантажується нульова послідовність, один або декілька біт. Повертає
    /// AsnType представлене у ASN.1 зашифрованого рядка біт.
    /// Якщо значення містить пусте значення, він пустий (довжина дорівнює 0)
рядок біт повертається
    ///
    /// </summary>
    /// <param name="value"> AsnType зашифрований об’єкт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує рядок біт.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType[])"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateBitString(AsnType value)
    {
      if (IsEmpty(value))
      { return new AsnType(0x03, EMPTY); }
```

```
      // BitString: Tag 0x03 (3, Universal, Primitive)
      return CreateBitString(value.GetBytes(), 0x00);
    }

    /// <summary>
    /// завантажується нульова послідовність, один або декілька біт. Повертає
    /// AsnType представлене у ASN.1 зашифрованого рядка біт.
    /// Якщо значення містить пусте значення, він пустий (довжина дорівнює 0)
рядок біт повертається
    ///.
    /// </summary>
    /// <param name="values"> AsnType зашифрований об’єкт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує рядок біт.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateBitString(AsnType[] values)
    {
      if (IsEmpty(values))
      { return new AsnType(0x03, EMPTY); }

      // BitString: Tag 0x03 (3, Universal, Primitive)
      return CreateBitString(Concatenate(values), 0x00);
    }

    /// <summary>
    /// <para> завантажується нульова послідовність, один або декілька біт.
Повертає
    /// AsnType представлене у ASN.1 зашифрованого рядка біт.</para>
    /// <para>Якщо октет містить пусте значення або довжина дорівнює 0, він
пустий (довжина дорівнює 0)
    /// повертається рядок біт.</para>
    /// <para>Якщо перетворення не відбулося, рядок біт повертає частину
    /// рядка біт. Частина рядка біт закінчує октет перед точкою помилки (не
включає октет, у якому сталась помилка, та наступні октети).</para>
    /// </summary>
    /// <param name="value"> MSB (big endian) byte[] представлене у
    /// зашифрованому рядку біт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує рядок біт.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateBitString(String value)
    {
      if (IsEmpty(value))
      { return CreateBitString(EMPTY); }

      // Будь які невикористовані біти?
      int lstrlen = value.Length;
      int unusedBits = 8 - (lstrlen % 8);
      if (8 == unusedBits) { unusedBits = 0; }

      for (int i = 0; i < unusedBits; i++)
      { value += "0"; }

      // Визначаємо число октетів
      int loctlen = (lstrlen + 7) / 8;
```

```
      List<byte> octets = new List<byte>();
      for (int i = 0; i < loctlen; i++)
      {
        String s = value.Substring(i * 8, 8);
        byte b = 0x00;

        try
        { b = Convert.ToByte(s, 2); }

        catch (FormatException /*e*/) { unusedBits = 0; break; }
        catch (OverflowException /*e*/) { unusedBits = 0; break; }

        octets.Add(b);
      }

      // BitString: Tag 0x03 (3, Universal, Primitive)
      return CreateBitString(octets.ToArray(), (uint)unusedBits);
    }

    /// <summary>
    /// завантажується нульова послідовність, один або декілька октетів.
Повертає
    /// ASN.1 зашифрований рядок октетів. Якщо октет містить пусте значення або
довжина
    /// дорівнює 0, він пустий (довжина дорівнює 0)  рядок октетів повертається.
    /// </summary>
    /// <param name="value"> MSB (big endian) byte[] представлене у
    ///  зашифрованому рядку октетів.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому рядку октетів.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateOctetString(byte[] value)
    {
      if (IsEmpty(value))
      {
        // Пустий рядок (октет)
        return new AsnType(0x04, EMPTY);
      }

      // OctetString: Tag 0x04 (4, Universal, Primitive)
      return new AsnType(0x04, value);
    }

    /// <summary>
    /// завантажується нульова послідовність, один або декілька октетів.
Повертає
    /// byte[] представлене у ASN.1 зашифрований рядок октетів.
    /// Якщо октет містить пусте значення або довжина дорівнює 0, він пустий
(довжина дорівнює 0)
    /// рядок октетів повертається.
    /// </summary>
    /// <param name="value"> AsnType зашифрований об’єкт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому рядку октетів.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateOctetString(AsnType value)
    {
      if (IsEmpty(value))
```

```
      {
        // Пустий рядок (октет)
        return new AsnType(0x04, 0x00);
      }

      // OctetString: Tag 0x04 (4, Universal, Primitive)
      return new AsnType(0x04, value.GetBytes());
    }

    /// <summary>
    /// завантажується нульова послідовність, один або декілька октетів.
Повертає
    /// byte[] представлене у ASN.1 зашифрований рядок октетів.
    /// Якщо октет містить пусте значення або довжина дорівнює 0, він пустий
(довжина дорівнює 0)
    /// рядок октетів повертається.
    /// </summary>
    /// <param name="values"> AsnType зашифрований об’єкт.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому рядку октетів.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(String)"/>
    internal static AsnType CreateOctetString(AsnType[] values)
    {
      if (IsEmpty(values))
      {
        // Пустий рядок (октет)
        return new AsnType(0x04, 0x00);
      }

      // OctetString: Tag 0x04 (4, Universal, Primitive)
      return new AsnType(0x04, Concatenate(values));
    }

    /// <summary>
    /// <para> завантажується нульова послідовність, один або декілька біт.
Повертає
    /// AsnType представлене у ASN.1 зашифрований рядок октетів.</para>
    /// <para>Якщо октет містить пусте значення або довжина дорівнює 0, він
пустий (довжина дорівнює 0)
    ///  рядок октетів повертається.</para>
    /// <para>Якщо перетворення відбулося з помилкою, the рядок біт повертає
частину
    /// рядок біт. Частина рядку октетів дописується у кінець октету перед
    /// точкою помилки (не вклбчає октет, який
    /// не був розібраний, або підпослідовність октетів).</para>
    /// </summary>
    /// <param name="value">Рядок представлений у
    ///  зашифрованому рядку октетів.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому рядку октетів.</returns>
    /// <seealso cref="CreateBitString(byte[])"/>
    /// <seealso cref="CreateBitString(byte[], uint)"/>
    /// <seealso cref="CreateBitString(String)"/>
    /// <seealso cref="CreateBitString(AsnType)"/>
    /// <seealso cref="CreateOctetString(byte[])"/>
    /// <seealso cref="CreateOctetString(AsnType)"/>
    /// <seealso cref="CreateOctetString(AsnType[])"/>
    internal static AsnType CreateOctetString(String value)
    {
      if (IsEmpty(value))
      { return CreateOctetString(EMPTY); }

      // Визначаємо число октетів
```

```
      int len = (value.Length + 255) / 256;

      List<byte> octets = new List<byte>();
      for (int i = 0; i < len; i++)
      {
        String s = value.Substring(i * 2, 2);
        byte b = 0x00;

        try
        { b = Convert.ToByte(s, 16); }
        catch (FormatException /*e*/) { break; }
        catch (OverflowException /*e*/) { break; }

        octets.Add(b);
      }

      // OctetString: Tag 0x04 (4, Universal, Primitive)
      return CreateOctetString(octets.ToArray());
    }

    /// <summary>
    /// <para>Повертає AsnType представлене у  ASN.1 зашифрованому
    /// цілому. Шифрування октетів цим методом не модифіковано.</para>
    /// <para>Якщо октет містить пусте значення або нульову довжину, метод
повертає
    /// AsnType який еквівалентний CreateInteger(byte[]{0})..</para>
    /// </summary>
    /// <param name="value"> MSB (big endian) byte[] представлене у
    /// зашифрованому цілому значенні.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому цілому.</returns>
    /// <example>
    /// ASN.1 зашифрованому  0:
    /// <code>CreateInteger(null)</code>
    /// <code>CreateInteger(new byte[]{0x00})</code>
    /// <code>CreateInteger(new byte[]{0x00, 0x00})</code>
    /// </example>
    /// <example>
    /// ASN.1 зашифрованому  1:
    /// <code>CreateInteger(new byte[]{0x01})</code>
    /// </example>
    /// <seealso cref="CreateIntegerPos"/>
    /// <seealso cref="CreateIntegerNeg"/>
    internal static AsnType CreateInteger(byte[] value)
    {
      //
      if (IsEmpty(value))
      { return CreateInteger(ZERO); }

      return new AsnType(0x02, value);
    }

    /// <summary>
    /// <para>Повертає AsnType представлене у  позитивному ASN.1 зашифрованому
    /// цілому. Якщо старші біти найбільш значимого байта встановлені. метод
додає 0x00 у октети перед величино, щоб гарантувати позитивне значення у
додатку.</para>
    /// <para>Якщо октет містить пусте значення або нульову довжину, метод
повертає
    /// AsnType який еквівалентний CreateInteger(byte[]{0})..</para>
    /// </summary>
    /// <param name="value"> MSB (big endian) byte[] представлене у
    /// зашифрованому цілому значенні.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// розшифрованому позитивному цілому .</returns>
    /// <example>
    /// ASN.1 зашифрованому  0:
    /// <code>CreateIntegerPos(null)</code>
    /// <code>CreateIntegerPos(new byte[]{0x00})</code>
```

```
    /// <code>CreateIntegerPos(new byte[]{0x00, 0x00})</code>
    /// </example>
    /// <example>
    /// ASN.1 зашифрованому  1:
    /// <code>CreateInteger(new byte[]{0x01})</code>
    /// </example>
    /// <seealso cref="CreateInteger"/>
    /// <seealso cref="CreateIntegerNeg"/>
    internal static AsnType CreateIntegerPos(byte[] value)
    {
      byte[] i = null, d = Duplicate(value);

      if (IsEmpty(d)) { d = ZERO; }

      // Пов’язання двох представлень.
      // Якщо у першому байті встановлений вищий біт, додаємо додатковий байт
0x00
      if (d.Length > 0 && d[0] > 0x7F)
      {
        i = new byte[d.Length + 1];
        i[0] = 0x00;
        Array.Copy(d, 0, i, 1, value.Length);
      }
      else
      {
        i = d;
      }

      // Ціле: Tag 0x02 (2, Universal, Primitive)
      return CreateInteger(i);
    }

    /// <summary>
    /// <para>Повертає негативне ASN.1 зашифрованому  цілому. Якщо вищий біт
найбільш значимого байта встановлений, то ціле число вважається
негативним</para>
    /// <para> Якщо  вищий біт найбільш значимого байта
    /// <bold>не</bold> встановлений, ціле число буде представлено двома
числами, щоб сформувати негативне ціле.</para>
    /// <para>Якщо октет містить пусте значення або нульову довжину, метод
повертає
    /// AsnType який еквівалентний CreateInteger(byte[]{0})..</para>
    /// </summary>
    /// <param name="value"> MSB (big endian) byte[] представлене у
    /// зашифрованому цілому значенні.</param>
    /// <returns>Повертає негативне ASN.1 зашифроване ціле.</returns>
    /// <example>
    /// ASN.1 зашифрованому  0:
    /// <code>CreateIntegerNeg(null)</code>
    /// <code>CreateIntegerNeg(new byte[]{0x00})</code>
    /// <code>CreateIntegerNeg(new byte[]{0x00, 0x00})</code>
    /// </example>
    /// <example>
    /// ASN.1 зашифрованому  -1 (другий байт 0xFF):
    /// <code>CreateIntegerNeg(new byte[]{0x01})</code>
    /// </example>
    /// <example>
    /// ASN.1 зашифрованому  -2 (другий байт 0xFE):
    /// <code>CreateIntegerNeg(new byte[]{0x02})</code>
    /// </example>
    /// <example>
    /// ASN.1 зашифрованому  -1:
    /// <code>CreateIntegerNeg(new byte[]{0xFF})</code>
    /// <code>CreateIntegerNeg(new byte[]{0xFF,0xFF})</code>
    /// вже негативне, з тих пір, як вищий біт встановлений.</example>
    /// <example>
    /// ASN.1 зашифрованому  -255 (другий байт 0xFF, 0x01):
    /// <code>CreateIntegerNeg(new byte[]{0x00,0xFF})</code>
    /// </example>
```

```
/// <example>
/// ASN.1 зашифрованому  -255 (другий байт 0xFF, 0xFF, 0x01):
/// <code>CreateIntegerNeg(new byte[]{0x00,0x00,0xFF})</code>
/// </example>
/// <seealso cref="CreateInteger"/>
/// <seealso cref="CreateIntegerPos"/>
internal static AsnType CreateIntegerNeg(byte[] value)
{
  // Це краще, для того, щоб добавляти '0', або
  //   визначати ціле?.
  if (IsEmpty(value))
  { return CreateInteger(ZERO); }

  // Не упорядковано
  // byte[] повинен мати путь для сенсу
  if (IsZero(value))
  { return CreateInteger(value); }

  //
  // У цій точці ми знаємо, що у нас є як мінімум один октет
  //

  // Це ціле вже негативне?
  if (value[0] >= 0x80)
  // Шифруємо без модифікацій
  { return CreateInteger(value); }

  // Немає необхідності дублювати
  byte[] c = Compliment2s(value);

  return CreateInteger(c);
}

/// <summary>
/// Повертає AsnType представлене у ASN.1 зашифрованому  пустому значенні.
/// </summary>
/// <returns>Повертає AsnType представлене у ASN.1
/// дешифрує песте значення.</returns>
internal static AsnType CreateNull()
{
  return new AsnType(0x05, new byte[] { 0x00 });
}

/// <summary>
/// Видалає додавання 0x00 октетів з byte[] октетів. Ций
/// метод повинен повертати пустий масив байтів (довжина дорівнює 0).
/// </summary>
/// <param name="octets">Массив октетів для перетворень.</param>
/// <returns> byte[] з додаванням  0x00 для видалення октетів.</returns>
internal static byte[] TrimStart(byte[] octets)
{
  if (IsEmpty(octets) || IsZero(octets))
  { return new byte[] { }; }

  byte[] d = Duplicate(octets);

  // Позиція першого ненульового значення
  int pos = 0;
  foreach (byte b in d)
  {
    if (0 != b) { break; }
    pos++;
  }

  // Не потребує трансформації
  if (pos == d.Length)
  { return octets; }

  // Розподілення масиву  трансформації
```

```
  byte[] t = new byte[d.Length - pos];

  // Копіювання
  Array.Copy(d, pos, t, 0, t.Length);

  return t;
}

/// <summary>
/// Видалення 0x00 октету з byte[] октетів. Цей
/// метод повинен повернути він пустий масив байт (довжина дорівнює 0).
/// </summary>
/// <param name="octets">Массив октетів для перетворень.</param>
/// <returns> byte[] з 0x00 для видалення октетів.</returns>
internal static byte[] TrimEnd(byte[] octets)
{
  if (IsEmpty(octets) || IsZero(octets))
  { return EMPTY; }

  byte[] d = Duplicate(octets);

  Array.Reverse(d);

  d = TrimStart(d);

  Array.Reverse(d);

  return d;
}

/// <summary>
/// Повертає AsnType представлене у ASN.1 зашифрованому  OID.
/// Якщо перетворення відбулося з помилкою, результат є частиною пертворення
/// до точки помилки. якщо oid рядок містить пусте значення або
/// не добре формований, він повертає пустий byte[].
/// </summary>
/// <param name="value">Рядок представлений у ідентифікаторі об’єкту
///  повинен бути зашифрований .</param>
/// <returns>Повертає AsnType представлене у ASN.1
/// дешифрує ідентифікатор об’єкту.</returns>
/// <example>Наступне назначає  зашифрованому  AsnType
/// для  RSA ключа до oid:
/// <code>AsnType oid = CreateOid("1.2.840.113549.1.1.1")</code>
/// </example>
/// <seealso cref="CreateOid(byte[])"/>
internal static AsnType CreateOid(String value)
{

  if (IsEmpty(value))
    return null;

  String[] tokens = value.Split(new Char[] { ' ', '.' });


  if (IsEmpty(tokens))
    return null;

  // Аналіз/обробка значення
  UInt64 a = 0;

  // Один або декілька рядків доступні
  List<UInt64> arcs = new List<UInt64>();

  foreach (String t in tokens)
  {
    // Не пусті, або погано відформовані рядки...
    if (t.Length == 0) { break; }

    try { a = Convert.ToUInt64(t, CultureInfo.InvariantCulture); }
```

```
        catch (FormatException /*e*/) { break; }
        catch (OverflowException /*e*/) { break; }

        arcs.Add(a);
      }


      if (0 == arcs.Count)
        return null;

      // Октети, які  потрібно повертати викликаючій функції
      List<byte> octets = new List<byte>();

      // Потрібно зберігати список невеликим
      // список повинен мати як мінімум один пункт...
      if (arcs.Count >= 1) { a = arcs[0] * 40; }
      if (arcs.Count >= 2) { a += arcs[1]; }
      octets.Add((byte)(a));

      // Додаються остальні аргументи(підідентифікатори)
      for (int i = 2; i < arcs.Count; i++)
      {
        // Тимчасовий розробник списка для усіх аргументів
        List<byte> temp = new List<byte>();

        // поточний аргумент (підідентифікатор)
        UInt64 arc = arcs[i];

        // Побудова аргументів(підідентифікатор) у вигляді байтового масиву
        // масив будується реверсивно (LSB до MSB).
        do
        {
          // Кожний вхід  формується з молодшого 7 біта (0x7F).
          // Встановлюється старший біт для усіх входів (0x80) згідно X.680. Ми
          // будемо скидувати старший біт останього байта пізніше.
          temp.Add((byte)(0x80 | (arc & 0x7F)));
          arc >>= 7;
        } while (0 != arc);

        // Захвачується результуючий масив. виконується цикл до/пока, до тих пір
пока є хоча б одна величина у масиві.
        byte[] t = temp.ToArray();

        // Старший біт скидується у  byte t[0]
        // t[0] є LSB після реверсування масиву.
        t[0] = (byte)(0x7F & t[0]);

        // MSB перший...
        Array.Reverse(t);

        // Додаємо до результуючого масиву
        foreach (byte b in t)
        { octets.Add(b); }
      }

      return CreateOid(octets.ToArray());
    }

    /// <summary>
    /// Повертає AsnType представлене у ASN.1 зашифрованому  OID.
    /// Якщо перетворення відбулося з помилкою, результат є частиною пертворення
    /// (до точки помилки). Якщо октет містить пусте значення,
    /// пустий byte[] повертається.
    /// </summary>
    /// <param name="value">The packed byte[] представлене у зашифрованому
ідентифікаторі об’єкту
    ///.</param>
    /// <returns>Повертає AsnType представлене у ASN.1
    /// дешифрує ідентифікатор об’єкту.</returns>
```

```
    /// <example>Наступне назначає  зашифрованому  AsnType for a RSA
    /// key to oid:
    /// <code>// Packed 1.2.840.113549.1.1.1
    /// byte[] rsa = new byte[] { 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01,
0x01, 0x01 };
    /// AsnType = CreateOid(rsa)</code>
    /// </example>
    /// <seealso cref="CreateOid(String)"/>
    internal static AsnType CreateOid(byte[] value)
    {

      if (IsEmpty(value))
      { return null; }

      // OID: Tag 0x06 (6, Universal, Primitive)
      return new AsnType(0x06, value);
    }

    private static byte[] Compliment1s(byte[] value)
    {
      if (IsEmpty(value))
      { return EMPTY; }

      // Створює копію масиву октетів
      byte[] c = Duplicate(value);

      for (int i = c.Length - 1; i >= 0; i--)
      {
        c[i] = (byte)~c[i];
      }

      return c;
    }

    private static byte[] Compliment2s(byte[] value)
    {
      if (IsEmpty(value))
      { return EMPTY; }

      if (IsZero(value))
      { return Duplicate(value); }

      // Створює копію масиву октетів
      byte[] d = Duplicate(value);

      int carry = 1;
      for (int i = d.Length - 1; i >= 0; i--)
      {
        d[i] = (byte)~d[i];

        // Додаємо
        int j = d[i] + carry;

        //  Записуємо назад
        d[i] = (byte)(j & 0xFF);

        //  Визначаємо наступний перенос
        if (0x100 == (j & 0x100))
        { carry = 1; }
        else
        { carry = 0; }
      }

      // Масив переносів (нам можливо потрібно розрахувати  'd'
      byte[] c = null;
      if (1 == carry)
      {
        c = new byte[d.Length + 1];
```

```
    // Розширений підпис....
    c[0] = (byte)0xFF;

    Array.Copy(d, 0, c, 1, d.Length);
  }
  else
  {
    c = d;
  }

  return c;
}

private static byte[] Concatenate(AsnType[] values)
{
  // Пусте значення на вході, пусте значення на виході
  if (IsEmpty(values))
    return new byte[] { };

  int length = 0;
  foreach (AsnType t in values)
  {
    if (null != t)
    { length += t.GetBytes().Length; }
  }

  byte[] cated = new byte[length];

  int current = 0;
  foreach (AsnType t in values)
  {
    if (null != t)
    {
      byte[] b = t.GetBytes();

      Array.Copy(b, 0, cated, current, b.Length);
      current += b.Length;
    }
  }

  return cated;
}

private static byte[] Concatenate(byte[] first, byte[] second)
{
  return Concatenate(new byte[][] { first, second });
}

private static byte[] Concatenate(byte[][] values)
{
  // Пусте значення на вході, пусте значення на виході
  if (IsEmpty(values))
    return new byte[] { };

  int length = 0;
  foreach (byte[] b in values)
  {
    if (null != b)
    { length += b.Length; }
  }

  byte[] cated = new byte[length];

  int current = 0;
  foreach (byte[] b in values)
  {
    if (null != b)
    {
      Array.Copy(b, 0, cated, current, b.Length);
```

```
      current += b.Length;
    }
  }

  return cated;
}

private static byte[] Duplicate(byte[] b)
{
  if (IsEmpty(b))
  { return EMPTY; }

  byte[] d = new byte[b.Length];
  Array.Copy(b, d, b.Length);

  return d;
}

private static bool IsZero(byte[] octets)
{
  if (IsEmpty(octets))
  { return false; }

  bool allZeros = true;
  for (int i = 0; i < octets.Length; i++)
  {
    if (0 != octets[i])
    { allZeros = false; break; }
  }
  return allZeros;
}

private static bool IsEmpty(byte[] octets)
{
  if (null == octets || 0 == octets.Length)
  { return true; }

  return false;
}

private static bool IsEmpty(String s)
{
  if (null == s || 0 == s.Length)
  { return true; }

  return false;
}

private static bool IsEmpty(String[] strings)
{
  if (null == strings || 0 == strings.Length)
    return true;

  return false;
}

private static bool IsEmpty(AsnType value)
{
  if (null == value)
  { return true; }

  return false;
}

private static bool IsEmpty(AsnType[] values)
{
  if (null == values || 0 == values.Length)
    return true;
```

```
      return false;
    }

    private static bool IsEmpty(byte[][] arrays)
    {
      if (null == arrays || 0 == arrays.Length)
        return true;

      return false;
    }
  }
}
```

Файл AsnKeyParser.cs – робота з ключами

```
using System;
using System.IO;
using System.Text;
using System.Diagnostics;
using System.Globalization;
using System.Collections.Generic;
using System.Security.Cryptography;

namespace CSInteropKeys
{
  class AsnKeyParser
  {
    private AsnParser parser;

    internal AsnKeyParser(String pathname)
    {
      using (BinaryReader reader = new BinaryReader(
          new FileStream(pathname, FileMode.Open, FileAccess.Read)))
      {
        FileInfo info = new FileInfo(pathname);

        parser = new AsnParser(reader.ReadBytes((int)info.Length));
      }
    }

    internal static byte[] TrimLeadingZero(byte[] values)
    {
      byte[] r = null;
      if ((0x00 == values[0]) && (values.Length > 1))
      {
        r = new byte[values.Length - 1];
        Array.Copy(values, 1, r, 0, values.Length - 1);
      }
      else
      {
        r = new byte[values.Length];
        Array.Copy(values, r, values.Length);
      }

      return r;
    }

    internal static bool EqualOid(byte[] first, byte[] second)
    {
      if (first.Length != second.Length)
      { return false; }

      for (int i = 0; i < first.Length; i++)
      {
        if (first[i] != second[i])
        { return false; }
      }

      return true;
    }

    internal RSAParameters ParseRSAPublicKey()
    {
      RSAParameters Parameters = new RSAParameters();

      // Поточні величини
      byte[] value = null;

      // Перевірка правильності роботи
      int length = 0;
```

```
      // Точка перевірки
      int position = parser.CurrentPosition();

      // Послідовність ігнорування  - PublicKeyInfo
      length = parser.NextSequence();
      if (length != parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
послідовності. ");
          sb.AppendFormat("Зазначено: {0}, Залишається: {1}",
          length.ToString(CultureInfo.InvariantCulture),
          parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - AlgorithmIdentifier
      length = parser.NextSequence();
      if (length > parser.RemainingBytes())
      {
        StringBuilder sb = new StringBuilder("Неправильний розмір ідентифікатора
алгоритму ");
        sb.AppendFormat("Зазначено: {0}, Залишається: {1}",
          length.ToString(CultureInfo.InvariantCulture),
          parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();
      // Захват OID
      value = parser.NextOID();
      byte[] oid = { 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01 };
      if (!EqualOid(value, oid))
      { throw new BerDecodeException("Очікуване OID 1.2.840.113549.1.1.1",
position); }

      // Опційні параметри
      if (parser.IsNextNull())
      {
        parser.NextNull();
        // value = parser.Next();
      }
      else
      {
        //
        value = parser.Next();
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // ігноруємо BitString - PublicKey
      length = parser.NextBitString();
      if (length > parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір Публічного
Ключа");
        sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
          (parser.RemainingBytes()).ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();
```

```
      // Послідовність ігнорування  - RSAPublicKey
      length = parser.NextSequence();
      if (length < parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір Публічного
Ключа RSA");
          sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
          parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      параметри.Modulus = TrimLeadingZero(parser.NextInteger());
      параметри.Exponent = TrimLeadingZero(parser.NextInteger());

      Debug.Assert(0 == parser.RemainingBytes());

      return параметри;
    }

    internal RSAParameters ParseRSAPrivateKey()
    {
      RSAParameters Parameters = new RSAParameters();

      // Поточні величини
      byte[] value = null;

      // Точка перевірки
      int position = parser.CurrentPosition();

      // Перевірка правильності роботи
      int length = 0;

      // Послідовність ігнорування  - PrivateKeyInfo
      length = parser.NextSequence();
      if (length != parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
послідовності.");
          sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();
      // Версія
      value = parser.NextInteger();
      if (0x00 != value[0])
      {
          StringBuilder sb = new StringBuilder("Неправильна версія
PrivateKeyInfo");
        BigInteger v = new BigInteger(value);
        sb.AppendFormat("Очікувана: 0, Зазначена: {0}", v.ToString(10));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - AlgorithmIdentifier
      length = parser.NextSequence();
      if (length > parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
ідентифікатора алгоритму");
          sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
```

```
            parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        // Точка перевірки
        position = parser.CurrentPosition();

        // Захват OID
        value = parser.NextOID();
        byte[] oid = { 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01 };
        if (!EqualOid(value, oid))
        { throw new BerDecodeException("Очікуване OID 1.2.840.113549.1.1.1",
position); }

        // Опційні параметри
        if (parser.IsNextNull())
        {
          parser.NextNull();
          // value = parser.Next();
        }
        else
        {
          //
          value = parser.Next();
        }

        // Точка перевірки
        position = parser.CurrentPosition();

        // Ігноруємо OctetString - PrivateKey
        length = parser.NextOctetString();
        if (length > parser.RemainingBytes())
        {
            StringBuilder sb = new StringBuilder("Неправильний розмір Закритого
Ключа. ");
            sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        // Точка перевірки
        position = parser.CurrentPosition();

        // Послідовність ігнорування  - RSAPrivateKey
        length = parser.NextSequence();
        if (length < parser.RemainingBytes())
        {
            StringBuilder sb = new StringBuilder("Неправильний розмір Закритого
Ключа RSA");
            sb.AppendFormat("Зазначено: {0},Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        // Точка перевірки
        position = parser.CurrentPosition();
        // Версія
        value = parser.NextInteger();
        if (0x00 != value[0])
        {
            StringBuilder sb = new StringBuilder("Неправильна версія Закритого
Ключа RSA");
          BigInteger v = new BigInteger(value);
          sb.AppendFormat("Очікувана: 0, Зазначена: {0}", v.ToString(10));
          throw new BerDecodeException(sb.ToString(), position);
        }
```

```
      параметри.Modulus = TrimLeadingZero(parser.NextInteger());
      параметри.Exponent = TrimLeadingZero(parser.NextInteger());
      параметри.D = TrimLeadingZero(parser.NextInteger());
      параметри.P = TrimLeadingZero(parser.NextInteger());
      параметри.Q = TrimLeadingZero(parser.NextInteger());
      параметри.DP = TrimLeadingZero(parser.NextInteger());
      параметри.DQ = TrimLeadingZero(parser.NextInteger());
      параметри.InverseQ = TrimLeadingZero(parser.NextInteger());

      Debug.Assert(0 == parser.RemainingBytes());

      return параметри;
    }

    internal DSAParameters ParseDSAPublicKey()
    {
      DSAParameters параметри = new DSAParameters();

      // Поточні величини
      byte[] value = null;

      // Поточна позиція
      int position = parser.CurrentPosition();
      // Перевірка правильності роботиз
      int length = 0;

      // Послідовність ігнорування  - PublicKeyInfo
      length = parser.NextSequence();
      if (length != parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
послідовності.");
        sb.AppendFormat("Зазначено: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - AlgorithmIdentifier
      length = parser.NextSequence();
      if (length > parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
ідентифікатора алгоритму");
          sb.AppendFormat("Зазначено: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Захват OID
      value = parser.NextOID();
      byte[] oid = { 0x2a, 0x86, 0x48, 0xce, 0x38, 0x04, 0x01 };
      if (!EqualOid(value, oid))
      { throw new BerDecodeException("Очікуване OID 1.2.840.10040.4.1",
position); }


      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - DSS-параметрів
      length = parser.NextSequence();
```

```
      if (length > parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір DSS-
параметрів.");
        sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Наступні три параметри еліптичної кривої
      параметри.P = TrimLeadingZero(parser.NextInteger());
      параметри.Q = TrimLeadingZero(parser.NextInteger());
      параметри.G = TrimLeadingZero(parser.NextInteger());

      // Ігноруємо BitString - PrivateKey
      parser.NextBitString();

      // Відкритий ключ
      параметри.Y = TrimLeadingZero(parser.NextInteger());

      Debug.Assert(0 == parser.RemainingBytes());

      return параметри;
    }

    internal DSAParameters ParseDSAPrivateKey()
    {
      DSAParameters параметри = new DSAParameters();

      // Поточні величини
      byte[] value = null;

      // Поточна позиція
      int position = parser.CurrentPosition();
      // Перевірка правильності роботи
      int length = 0;

      // Послідовність ігнорування  - PrivateKeyInfo
      length = parser.NextSequence();
      if (length != parser.RemainingBytes())
      {
          StringBuilder sb = new StringBuilder("Неправильний розмір
послідовності.");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();
      // Версія
      value = parser.NextInteger();
      if (0x00 != value[0])
      {
        throw new BerDecodeException("Неправильна версія PrivateKeyInfo",
position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - AlgorithmIdentifier
      length = parser.NextSequence();
      if (length > parser.RemainingBytes())
      {
        StringBuilder sb = new StringBuilder("Неправильний розмір ідентифікатора
алгоритму");
```

```
        sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Точка перевірки
      position = parser.CurrentPosition();
      // Захват OID
      value = parser.NextOID();
      byte[] oid = { 0x2a, 0x86, 0x48, 0xce, 0x38, 0x04, 0x01 };
      if (!EqualOid(value, oid))
      { throw new BerDecodeException("Очікуване OID 1.2.840.10040.4.1",
position); }

      // Точка перевірки
      position = parser.CurrentPosition();

      // Послідовність ігнорування  - DSS-параметрів
      length = parser.NextSequence();
      if (length > parser.RemainingBytes())
      {
        StringBuilder sb = new StringBuilder("Невірний розмір DSS-параметрів");
        sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
          length.ToString(CultureInfo.InvariantCulture),
parser.RemainingBytes().ToString(CultureInfo.InvariantCulture));
        throw new BerDecodeException(sb.ToString(), position);
      }

      // Наступні три параметри еліптичної кривої
      параметри.P = TrimLeadingZero(parser.NextInteger());
      параметри.Q = TrimLeadingZero(parser.NextInteger());
      параметри.G = TrimLeadingZero(parser.NextInteger());

      // Ігноруємо OctetString - PrivateKey
      parser.NextOctetString();

      // Таємний ключ
      параметри.X = TrimLeadingZero(parser.NextInteger());

      Debug.Assert(0 == parser.RemainingBytes());

      return параметри;
    }
  }

  class AsnParser
  {
    private List<byte> octets;
    private int initialCount;

    internal AsnParser(byte[] values)
    {
      octets = new List<byte>(values.Length);
      octets.AddRange(values);

      initialCount = octets.Count;
    }

    internal int CurrentPosition()
    {
      return initialCount - octets.Count;
    }

    internal int RemainingBytes()
    {
      return octets.Count;
    }
```

```
    private int GetLength()
    {
      int length = 0;

      // Точка перевірки
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();

        if (b == (b & 0x7f)) { return b; }
        int i = b & 0x7f;

        if (i > 4)
        {
            StringBuilder sb = new StringBuilder("Невірна довжина кодування");
            sb.AppendFormat("Довжина використовує (0) октету",
            i.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        while (0 != i--)
        {
          // зсув уліво
          length <<= 8;

          length |= GetNextOctet();
        }
      }
      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}

      return length;
    }

    internal byte[] Next()
    {
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();

        int length = GetLength();
        if (length > RemainingBytes())
        {
            StringBuilder sb = new StringBuilder("Невірний розмір");
            sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        return GetOctets(length);
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
    }

    internal byte GetNextOctet()
    {
      int position = CurrentPosition();

      if (0 == RemainingBytes())
      {
```

```
          StringBuilder sb = new StringBuilder("Невірний розмір");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            1.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        byte b = GetOctets(1)[0];

        return b;
      }

      internal byte[] GetOctets(int octetCount)
      {
        int position = CurrentPosition();

        if (octetCount > RemainingBytes())
        {
            StringBuilder sb = new StringBuilder("Невірний розмір");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            octetCount.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        byte[] values = new byte[octetCount];

        try
        {
          octets.CopyTo(0, values, 0, octetCount);
          octets.RemoveRange(0, octetCount);
        }

        catch (ArgumentOutOfRangeException ex)
        { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}

        return values;
      }

      internal bool IsNextNull()
      {
        return 0x05 == octets[0];
      }

      internal int NextNull()
      {
        int position = CurrentPosition();

        try
        {
          byte b = GetNextOctet();
          if (0x05 != b)
          {
            StringBuilder sb = new StringBuilder("Очікування Null.");
            sb.AppendFormat("Зазначений ідентифікатор: {0}",
b.ToString(CultureInfo.InvariantCulture));
            throw new BerDecodeException(sb.ToString(), position);
          }

          // Next octet must be 0
          b = GetNextOctet();
          if (0x00 != b)
          {
              StringBuilder sb = new StringBuilder("Null має ненульовий розмір.
");
            sb.AppendFormat("Розмір: {0}",
b.ToString(CultureInfo.InvariantCulture));
            throw new BerDecodeException(sb.ToString(), position);
```

```
        }

        return 0;
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
    }

    internal bool IsNextSequence()
    {
      return 0x30 == octets[0];
    }

    internal int NextSequence()
    {
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();
        if (0x30 != b)
        {
            StringBuilder sb = new StringBuilder("Очікувана послідовність");
            sb.AppendFormat("Зазначений ідентифікатор: {0}",
            b.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        int length = GetLength();
        if (length > RemainingBytes())
        {
            StringBuilder sb = new StringBuilder("Неправильний розмір
послідовності. ");
            sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        return length;
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
    }

    internal bool IsNextOctetString()
    {
      return 0x04 == octets[0];
    }

    internal int NextOctetString()
    {
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();
        if (0x04 != b)
        {
          StringBuilder sb = new StringBuilder("Очікуваний рядок октету");
          sb.AppendFormat("Зазначений ідентифікатор: {0}",
b.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }
```

```
        int length = GetLength();
        if (length > RemainingBytes())
        {
          StringBuilder sb = new StringBuilder("Неправильний розмір рядка
октету");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        return length;
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа, position, ex);
}
    }

    internal bool IsNextBitString()
    {
      return 0x03 == octets[0];
    }

    internal int NextBitString()
    {
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();
        if (0x03 != b)
        {
          StringBuilder sb = new StringBuilder("Очікуваний Рядок біт.");
          sb.AppendFormat("Зазначений ідентифікатор: {0}",
b.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        int length = GetLength();

        // Ми повинні поглинати біти, які не використовуються, де перший октет
величина, що остається
        b = octets[0];
        octets.RemoveAt(0);
        length--;

        if (0x00 != b)
        { throw new BerDecodeException("Перший октет BitString має бути 0",
position); }

        return length;
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
    }

    internal bool IsNextInteger()
    {
      return 0x02 == octets[0];
    }

    internal byte[] NextInteger()
    {
      int position = CurrentPosition();

      try
```

```
      {
        byte b = GetNextOctet();
        if (0x02 != b)
        {
          StringBuilder sb = new StringBuilder("Очікуване значення");
          sb.AppendFormat("Зазначений ідентифікатор: {0}",
b.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        int length = GetLength();
        if (length > RemainingBytes())
        {
          StringBuilder sb = new StringBuilder("Невірний розмір змінної");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        return GetOctets(length);
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
    }

    internal byte[] NextOID()
    {
      int position = CurrentPosition();

      try
      {
        byte b = GetNextOctet();
        if (0x06 != b)
        {
          StringBuilder sb = new StringBuilder("Очікуваний ідентифікатор
об'єкта. ");
          sb.AppendFormat("Зазначений ідентифікатор: {0}",
            b.ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        int length = GetLength();
        if (length > RemainingBytes())
        {
          StringBuilder sb = new StringBuilder("Неправильний розмір
ідентифікатора об'єкта");
          sb.AppendFormat("Вказано: {0}, Залишилося: {1}",
            length.ToString(CultureInfo.InvariantCulture),
            RemainingBytes().ToString(CultureInfo.InvariantCulture));
          throw new BerDecodeException(sb.ToString(), position);
        }

        byte[] values = new byte[length];

        for (int i = 0; i < length; i++)
        {
          values[i] = octets[0];
          octets.RemoveAt(0);
        }

        return values;
      }

      catch (ArgumentOutOfRangeException ex)
      { throw new BerDecodeException("Помилка при розборі Ключа", position, ex);
}
```

```
        }
    }
}
```

Файл BigInteger.cs – робота з великими числами

```
      //*********************************************************************
************
//
// Реалізовані оператори +, -, *, /, %, >>, <<, ==, !=, >, <, >=, <=, &, |, ^,
++, --, ~
//
// Характеристики
// --------
// 1) Арифметичні операції, які включають великі прості числа.
// 2)Тест на простоту, використовуючий малу теорему Ферма та метод Рабіна Мілера
//    Методи Соловея страшенна та Лукаса для побудови псевдо простих чисел.
// 3) Взяття логарифма по модулю з зменшенням Баретта.
// 4) Інверсія по модулю.
// 5) Генератор псевдовипадкових чисел.
// 6) Генератор чисел, з яких складаються прості.
//
//
//
//       byte[] pseudoPrime1 = { (byte)0x00,
//             (byte)0x85, (byte)0x84, (byte)0x64, (byte)0xFD, (byte)0x70,
(byte)0x6A,
//             (byte)0x9F, (byte)0xF0, (byte)0x94, (byte)0x0C, (byte)0x3E,
(byte)0x2C,
//             (byte)0x74, (byte)0x34, (byte)0x05, (byte)0xC9, (byte)0x55,
(byte)0xB3,
//             (byte)0x85, (byte)0x32, (byte)0x98, (byte)0x71, (byte)0xF9,
(byte)0x41,
//             (byte)0x21, (byte)0x5F, (byte)0x02, (byte)0x9E, (byte)0xEA,
(byte)0x56,
//             (byte)0x8D, (byte)0x8C, (byte)0x44, (byte)0xCC, (byte)0xEE,
(byte)0xEE,
//             (byte)0x3D, (byte)0x2C, (byte)0x9D, (byte)0x2C, (byte)0x12,
(byte)0x41,
//             (byte)0x1E, (byte)0xF1, (byte)0xC5, (byte)0x32, (byte)0xC3,
(byte)0xAA,
//             (byte)0x31, (byte)0x4A, (byte)0x52, (byte)0xD8, (byte)0xE8,
(byte)0xAF,
//             (byte)0x42, (byte)0xF4, (byte)0x72, (byte)0xA1, (byte)0x2A,
(byte)0x0D,
//             (byte)0x97, (byte)0xB1, (byte)0x31, (byte)0xB3,
//       };
//
//
//*********************************************************************
******

using System;

public class BigInteger
{
  // Максимальна довжина BigInteger  у пристрої (4 байта)
  // можливо змінити для підвищення рівня точності.

  private const int maxLength = 1024;

  // прості числа менші 2000, дя можливості тестування генератора простих чисел

  public static readonly int[] primesBelow2000 = {
        2, 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 };


  private uint[] data = null;             // запам/ятованы байти з Big Integer
  public int dataLength;                 // число значущих символів
used


  //*********************************************************************
  // Конструктор(Любе значення для BigInteger дорівнює 0
  //*********************************************************************

  public BigInteger()
  {
    data = new uint[maxLength];
    dataLength = 1;
  }


  //*********************************************************************
  // Конструктор(Любе значення з заданою довжиною  )
  //*********************************************************************

  public BigInteger(long value)
  {
    data = new uint[maxLength];
    long tempVal = value;

    // Копіювання байтів з довжиною до BigInteger без любого передположення
довжини довгого типу даних

    dataLength = 0;
    while (value != 0 && dataLength < maxLength)
    {
      data[dataLength] = (uint)(value & 0xFFFFFFFF);
      value >>= 32;
      dataLength++;
    }

    if (tempVal > 0)         // перевірка переповнення для +ve значення
    {
      if (value != 0 || (data[maxLength - 1] & 0x80000000) != 0)
          throw (new ArithmeticException("Позитивне переповнення в
конструкторі."));
    }
    else if (tempVal < 0)    // перевірка недоповнення  для  -ve значення
    {
```

```
      if (value != -1 || (data[dataLength - 1] & 0x80000000) == 0)
          throw (new ArithmeticException("Негативне переповнення в
конструкторі."));
    }

    if (dataLength == 0)
      dataLength = 1;
  }


  //*********************************************************************
  // Конструктор(Любе значення передбачене типом ulong)
  //*********************************************************************

  public BigInteger(ulong value)
  {
    data = new uint[maxLength];

    // Копіювання байтів з ulong до BigInteger без любого передположення
    // довжини типу даних ulong

    dataLength = 0;
    while (value != 0 && dataLength < maxLength)
    {
      data[dataLength] = (uint)(value & 0xFFFFFFFF);
      value >>= 32;
      dataLength++;
    }

    if (value != 0 || (data[maxLength - 1] & 0x80000000) != 0)
        throw (new ArithmeticException("Позитивне переповнення в
конструкторі."));

    if (dataLength == 0)
      dataLength = 1;
  }



  //*********************************************************************
  // Конструктор(Любе значення передбачене типом BigInteger)
  //*********************************************************************

  public BigInteger(BigInteger bi)
  {
    data = new uint[maxLength];

    dataLength = bi.dataLength;

    for (int i = 0; i < dataLength; i++)
      data[i] = bi.data[i];
  }


  //*********************************************************************
  // Конструктор(Любе значення передбачене типом рядок разрядів   //
визначеної тибази
  //
  // Example (base 10)
  // -----------------
  // Ініціалізує "a" з любого значення  1234 у базі 10
  //      BigInteger a = new BigInteger("1234", 10)
  //
  // Ініціалізує "a" з любого значення  -1234
  //      BigInteger a = new BigInteger("-1234", 10)
  //
  // Example (base 16)
  // -----------------
  // Ініціалізує "a" з любого значення  0x1D4F у базі 16
```

```
  //      BigInteger a = new BigInteger("1D4F", 16)
  //
  // Ініціалізує "a" з любого значення  -0x1D4F
  //      BigInteger a = new BigInteger("-1D4F", 16)
  //
  // Відмітимо, що величини рядка визначеного <sign><magnitude>
  // формат.
  //
  //*********************************************************************

  public BigInteger(string value, int radix)
  {
    BigInteger multiplier = new BigInteger(1);
    BigInteger result = new BigInteger();
    value = (value.ToUpper()).Trim();
    int limit = 0;

    if (value[0] == '-')
      limit = 1;

    for (int i = value.Length - 1; i >= limit; i--)
    {
      int posVal = (int)value[i];

      if (posVal >= '0' && posVal <= '9')
        posVal -= '0';
      else if (posVal >= 'A' && posVal <= 'Z')
        posVal = (posVal - 'A') + 10;
      else
        posVal = 9999999;       // довільно великий


      if (posVal >= radix)
          throw (new ArithmeticException("Невірна лінія в конструкторі."));
      else
      {
        if (value[0] == '-')
          posVal = -posVal;

        result = result + (multiplier * posVal);

        if ((i - 1) >= limit)
          multiplier = multiplier * radix;
      }
    }

    if (value[0] == '-')     // негативні значення
    {
      if ((result.data[maxLength - 1] & 0x80000000) == 0)
          throw (new ArithmeticException("Негативне переповнення в
конструкторі."));
    }
    else    // позитивні значення
    {
      if ((result.data[maxLength - 1] & 0x80000000) != 0)
          throw (new ArithmeticException("Позитивне переповнення в
конструкторі."));
    }

    data = new uint[maxLength];
    for (int i = 0; i < result.dataLength; i++)
      data[i] = result.data[i];

    dataLength = result.dataLength;
  }


  //*********************************************************************
  // Конструктор(Любе значення передбачене типом Массив байтів
```

```
  //
  // нижній індекс у віхідному масиві байтів (i.e [0]) повинно містити
  // найбільш значимий байт числа, й верхній індекс повинен містити найменш
значимий байт числа
  // Ініціалізує "a" з любого значення  0x1D4F у базі 16
  //      byte[] temp = { 0x1D, 0x4F };
  //      BigInteger a = new BigInteger(temp)
  //
  // Відмітимо, що метод ініціалізації не допускає
  // визначення знаку.
  //
  //*********************************************************************

  public BigInteger(byte[] inData)
  {
    dataLength = inData.Length >> 2;

    int leftOver = inData.Length & 0x3;
    if (leftOver != 0)         // довжина не ділиться на 4
      dataLength++;


    if (dataLength > maxLength)
        throw (new ArithmeticException("Байт переповнення в конструкторі."));

    data = new uint[maxLength];

    for (int i = inData.Length - 1, j = 0; i >= 3; i -= 4, j++)
    {
      data[j] = (uint)((inData[i - 3] << 24) + (inData[i - 2] << 16) +
                       (inData[i - 1] << 8) + inData[i]);
    }

    if (leftOver == 1)
      data[dataLength - 1] = (uint)inData[0];
    else if (leftOver == 2)
      data[dataLength - 1] = (uint)((inData[0] << 8) + inData[1]);
    else if (leftOver == 3)
      data[dataLength - 1] = (uint)((inData[0] << 16) + (inData[1] << 8) +
inData[2]);


    while (dataLength > 1 && data[dataLength - 1] == 0)
      dataLength--;

    //Console.WriteLine("Len = " + dataLength);
  }


  //*********************************************************************
  // Конструктор(Любе значення передбачене типом Массив байтів
  // спеціальної довжини.)
  //*********************************************************************

  public BigInteger(byte[] inData, int inLen)
  {
    dataLength = inLen >> 2;

    int leftOver = inLen & 0x3;
    if (leftOver != 0)         // довжина не ділиться на 4
      dataLength++;

    if (dataLength > maxLength || inLen > inData.Length)
        throw (new ArithmeticException("Байт переповнення в конструкторі."));


    data = new uint[maxLength];

    for (int i = inLen - 1, j = 0; i >= 3; i -= 4, j++)
```

```
    {
      data[j] = (uint)((inData[i - 3] << 24) + (inData[i - 2] << 16) +
                       (inData[i - 1] << 8) + inData[i]);
    }

    if (leftOver == 1)
      data[dataLength - 1] = (uint)inData[0];
    else if (leftOver == 2)
      data[dataLength - 1] = (uint)((inData[0] << 8) + inData[1]);
    else if (leftOver == 3)
      data[dataLength - 1] = (uint)((inData[0] << 16) + (inData[1] << 8) +
inData[2]);


    if (dataLength == 0)
      dataLength = 1;

    while (dataLength > 1 && data[dataLength - 1] == 0)
      dataLength--;

    //Console.WriteLine("Len = " + dataLength);
  }


  //**********************************************************************
  // Конструктор(Любе значення передбачене типом Массив unsigned integers)
  //**********************************************************************

  public BigInteger(uint[] inData)
  {
    dataLength = inData.Length;

    if (dataLength > maxLength)
        throw (new ArithmeticException("Байт переповнення в конструкторі."));

    data = new uint[maxLength];

    for (int i = dataLength - 1, j = 0; i >= 0; i--, j++)
      data[j] = inData[i];

    while (dataLength > 1 && data[dataLength - 1] == 0)
      dataLength--;

    //Console.WriteLine("Len = " + dataLength);
  }


  //**********************************************************************
  // Переповнення оператора визначення типу .
  // Для BigInteger bi = 10;
  //**********************************************************************

  public static implicit operator BigInteger(long value)
  {
    return (new BigInteger(value));
  }

  public static implicit operator BigInteger(ulong value)
  {
    return (new BigInteger(value));
  }

  public static implicit operator BigInteger(int value)
  {
    return (new BigInteger((long)value));
  }

  public static implicit operator BigInteger(uint value)
  {
```

```
    return (new BigInteger((ulong)value));
  }


  //*********************************************************************
  // Переповнення оператора додавання
  //*********************************************************************

  public static BigInteger operator +(BigInteger bi1, BigInteger bi2)
  {
    BigInteger result = new BigInteger();

    result.dataLength = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;

    long carry = 0;
    for (int i = 0; i < result.dataLength; i++)
    {
      long sum = (long)bi1.data[i] + (long)bi2.data[i] + carry;
      carry = sum >> 32;
      result.data[i] = (uint)(sum & 0xFFFFFFFF);
    }

    if (carry != 0 && result.dataLength < maxLength)
    {
      result.data[result.dataLength] = (uint)(carry);
      result.dataLength++;
    }

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;


    // перевірка переповнення
    int lastPos = maxLength - 1;
    if ((bi1.data[lastPos] & 0x80000000) == (bi2.data[lastPos] & 0x80000000) &&
       (result.data[lastPos] & 0x80000000) != (bi1.data[lastPos] & 0x80000000))
    {
      throw (new ArithmeticException());
    }

    return result;
  }


  //*********************************************************************
  // Переповнення оператора unary ++
  //*********************************************************************

  public static BigInteger operator ++(BigInteger bi1)
  {
    BigInteger result = new BigInteger(bi1);

    long val, carry = 1;
    int index = 0;

    while (carry != 0 && index < maxLength)
    {
      val = (long)(result.data[index]);
      val++;

      result.data[index] = (uint)(val & 0xFFFFFFFF);
      carry = val >> 32;

      index++;
    }

    if (index > result.dataLength)
      result.dataLength = index;
```

```
    else
    {
      while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
        result.dataLength--;
    }

    // перевірка переповнення
    int lastPos = maxLength - 1;

    // переповнення якщо початкова величина була +ve але ++ викликає зміну знаку
    // на негативне.

    if ((bi1.data[lastPos] & 0x80000000) == 0 &&
       (result.data[lastPos] & 0x80000000) != (bi1.data[lastPos] & 0x80000000))
    {
      throw (new ArithmeticException("Overflow in ++."));
    }
    return result;
  }


  //*********************************************************************
  // Переповнення оператора віднімання
  //*********************************************************************

  public static BigInteger operator -(BigInteger bi1, BigInteger bi2)
  {
    BigInteger result = new BigInteger();

    result.dataLength = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;

    long carryIn = 0;
    for (int i = 0; i < result.dataLength; i++)
    {
      long diff;

      diff = (long)bi1.data[i] - (long)bi2.data[i] - carryIn;
      result.data[i] = (uint)(diff & 0xFFFFFFFF);

      if (diff < 0)
        carryIn = 1;
      else
        carryIn = 0;
    }

    // реєстр над негативними
    if (carryIn != 0)
    {
      for (int i = result.dataLength; i < maxLength; i++)
        result.data[i] = 0xFFFFFFFF;
      result.dataLength = maxLength;
    }
    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    // перевірка переповнення

    int lastPos = maxLength - 1;
    if ((bi1.data[lastPos] & 0x80000000) != (bi2.data[lastPos] & 0x80000000) &&
       (result.data[lastPos] & 0x80000000) != (bi1.data[lastPos] & 0x80000000))
    {
      throw (new ArithmeticException());
    }

    return result;
  }
```

```
//*********************************************************************
// Переповнення оператора unary --
//*********************************************************************

public static BigInteger operator --(BigInteger bi1)
{
  BigInteger result = new BigInteger(bi1);

  long val;
  bool carryIn = true;
  int index = 0;

  while (carryIn && index < maxLength)
  {
    val = (long)(result.data[index]);
    val--;

    result.data[index] = (uint)(val & 0xFFFFFFFF);

    if (val >= 0)
      carryIn = false;

    index++;
  }

  if (index > result.dataLength)
    result.dataLength = index;

  while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
    result.dataLength--;

  // перевірка переповнення
  int lastPos = maxLength - 1;

  // переповнення якщо початкова величина була -ve but -- викликає зміну знаку
  // на позитивний.

  if ((bi1.data[lastPos] & 0x80000000) != 0 &&
     (result.data[lastPos] & 0x80000000) != (bi1.data[lastPos] & 0x80000000))
  {
    throw (new ArithmeticException("Втрата значення в --."));
  }

  return result;
}


//*********************************************************************
// Переповнення оператора множення
//*********************************************************************

public static BigInteger operator *(BigInteger bi1, BigInteger bi2)
{
  int lastPos = maxLength - 1;
  bool bi1Neg = false, bi2Neg = false;

  // take the absolute значення inputs
  try
  {
    if ((bi1.data[lastPos] & 0x80000000) != 0)     // bi1 негативне
    {
      bi1Neg = true; bi1 = -bi1;
    }
    if ((bi2.data[lastPos] & 0x80000000) != 0)     // bi2 негативне
    {
      bi2Neg = true; bi2 = -bi2;
    }
  }
  catch (Exception) { }
```

```
    BigInteger result = new BigInteger();

    // перемножування беззнакових величин
    try
    {
      for (int i = 0; i < bi1.dataLength; i++)
      {
        if (bi1.data[i] == 0) continue;

        ulong mcarry = 0;
        for (int j = 0, k = i; j < bi2.dataLength; j++, k++)
        {
          // k = i + j
          ulong val = ((ulong)bi1.data[i] * (ulong)bi2.data[j]) +
                       (ulong)result.data[k] + mcarry;

          result.data[k] = (uint)(val & 0xFFFFFFFF);
          mcarry = (val >> 32);
        }

        if (mcarry != 0)
          result.data[i + bi2.dataLength] = (uint)mcarry;
      }
    }
    catch (Exception)
    {
      throw (new ArithmeticException("Переповнення множення"));
    }


    result.dataLength = bi1.dataLength + bi2.dataLength;
    if (result.dataLength > maxLength)
      result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    // перевірка переповнення (результат у -ve)
    if ((result.data[lastPos] & 0x80000000) != 0)
    {
      if (bi1Neg != bi2Neg && result.data[lastPos] == 0x80000000)    // інший
знак
      {
        // Заголовок – спеціальний випадок коли результат множення
        // максимальне негативне число з двох складових.

        if (result.dataLength == 1)
          return result;
        else
        {
          bool isMaxNeg = true;
          for (int i = 0; i < result.dataLength - 1 && isMaxNeg; i++)
          {
            if (result.data[i] != 0)
              isMaxNeg = false;
          }

          if (isMaxNeg)
            return result;
        }
      }

      throw (new ArithmeticException("Переповнення множення"));
    }

    // Якщо вхід має інший знак, то результат у -ve
    if (bi1Neg != bi2Neg)
      return -result;
```

```
  return result;
}


//*********************************************************************
// Переповнення операторів unary <<
//*********************************************************************

public static BigInteger operator <<(BigInteger bi1, int shiftVal)
{
  BigInteger result = new BigInteger(bi1);
  result.dataLength = shiftLeft(result.data, shiftVal);

  return result;
}


// найменш значимі біти у більш низькій частині буферу

private static int shiftLeft(uint[] buffer, int shiftVal)
{
  int shiftAmount = 32;
  int bufLen = buffer.Length;

  while (bufLen > 1 && buffer[bufLen - 1] == 0)
    bufLen--;

  for (int count = shiftVal; count > 0; )
  {
    if (count < shiftAmount)
      shiftAmount = count;

    //Console.WriteLine("shiftAmount = {0}", shiftAmount);

    ulong carry = 0;
    for (int i = 0; i < bufLen; i++)
    {
      ulong val = ((ulong)buffer[i]) << shiftAmount;
      val |= carry;

      buffer[i] = (uint)(val & 0xFFFFFFFF);
      carry = val >> 32;
    }

    if (carry != 0)
    {
      if (bufLen + 1 <= buffer.Length)
      {
        buffer[bufLen] = (uint)carry;
        bufLen++;
      }
    }
    count -= shiftAmount;
  }
  return bufLen;
}


//*********************************************************************
// Переповнення оператору unary >>
//*********************************************************************

public static BigInteger operator >>(BigInteger bi1, int shiftVal)
{
  BigInteger result = new BigInteger(bi1);
  result.dataLength = shiftRight(result.data, shiftVal);
```

```
    if ((bi1.data[maxLength - 1] & 0x80000000) != 0) // негативне
    {
      for (int i = maxLength - 1; i >= result.dataLength; i--)
        result.data[i] = 0xFFFFFFFF;

      uint mask = 0x80000000;
      for (int i = 0; i < 32; i++)
      {
        if ((result.data[result.dataLength - 1] & mask) != 0)
          break;

        result.data[result.dataLength - 1] |= mask;
        mask >>= 1;
      }
      result.dataLength = maxLength;
    }

    return result;
  }


  private static int shiftRight(uint[] buffer, int shiftVal)
  {
    int shiftAmount = 32;
    int invShift = 0;
    int bufLen = buffer.Length;

    while (bufLen > 1 && buffer[bufLen - 1] == 0)
      bufLen--;

    //Console.WriteLine("bufLen = " + bufLen + " buffer.Length = " +
buffer.Length);

    for (int count = shiftVal; count > 0; )
    {
      if (count < shiftAmount)
      {
        shiftAmount = count;
        invShift = 32 - shiftAmount;
      }

      //Console.WriteLine("shiftAmount = {0}", shiftAmount);

      ulong carry = 0;
      for (int i = bufLen - 1; i >= 0; i--)
      {
        ulong val = ((ulong)buffer[i]) >> shiftAmount;
        val |= carry;

        carry = ((ulong)buffer[i]) << invShift;
        buffer[i] = (uint)(val);
      }

      count -= shiftAmount;
    }

    while (bufLen > 1 && buffer[bufLen - 1] == 0)
      bufLen--;

    return bufLen;
  }


  //*********************************************************************
  // Переповнення оператора NOT (одна складова)
  //*********************************************************************

  public static BigInteger operator ~(BigInteger bi1)
```

```
  {
    BigInteger result = new BigInteger(bi1);

    for (int i = 0; i < maxLength; i++)
      result.data[i] = (uint)(~(bi1.data[i]));

    result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    return result;
  }


  //*********************************************************************
  // Переповнення оператора  NEGATE (дві складові)
  //*********************************************************************

  public static BigInteger operator -(BigInteger bi1)
  {


    if (bi1.dataLength == 1 && bi1.data[0] == 0)
      return (new BigInteger());

    BigInteger result = new BigInteger(bi1);

    // одна складова
    for (int i = 0; i < maxLength; i++)
      result.data[i] = (uint)(~(bi1.data[i]));

    // Додаємо одиницю до результуючої однієї складової
    long val, carry = 1;
    int index = 0;

    while (carry != 0 && index < maxLength)
    {
      val = (long)(result.data[index]);
      val++;

      result.data[index] = (uint)(val & 0xFFFFFFFF);
      carry = val >> 32;

      index++;
    }

    if ((bi1.data[maxLength - 1] & 0x80000000) == (result.data[maxLength - 1] &
0x80000000))
        throw (new ArithmeticException("Переповнення заперечення.\n"));

    result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;
    return result;
  }


  //*********************************************************************
  // Переповнення оператора рівності
//*********************************************************************

  public static bool operator ==(BigInteger bi1, BigInteger bi2)
  {
    return bi1.Equals(bi2);
  }
```

```
  public static bool operator !=(BigInteger bi1, BigInteger bi2)
  {
    return !(bi1.Equals(bi2));
  }


  public override bool Equals(object o)
  {
    BigInteger bi = (BigInteger)o;

    if (this.dataLength != bi.dataLength)
      return false;

    for (int i = 0; i < this.dataLength; i++)
    {
      if (this.data[i] != bi.data[i])
        return false;
    }
    return true;
  }


  public override int GetHashCode()
  {
    return this.ToString().GetHashCode();
  }


  //*********************************************************************
  // Переповнення оператора нерівності
  //*********************************************************************

  public static bool operator >(BigInteger bi1, BigInteger bi2)
  {
    int pos = maxLength - 1;

    // bi1 є негативне, bi2 є позитивне
    if ((bi1.data[pos] & 0x80000000) != 0 && (bi2.data[pos] & 0x80000000) == 0)
      return false;

    // bi1 позитивне, bi2 is негативне
    else if ((bi1.data[pos] & 0x80000000) == 0 && (bi2.data[pos] & 0x80000000)
!= 0)
      return true;

    // той же знак
    int len = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;
    for (pos = len - 1; pos >= 0 && bi1.data[pos] == bi2.data[pos]; pos--) ;

    if (pos >= 0)
    {
      if (bi1.data[pos] > bi2.data[pos])
        return true;
      return false;
    }
    return false;
  }


  public static bool operator <(BigInteger bi1, BigInteger bi2)
  {
    int pos = maxLength - 1;

    // bi1 негативне, bi2 позитивне
    if ((bi1.data[pos] & 0x80000000) != 0 && (bi2.data[pos] & 0x80000000) == 0)
      return true;

    // bi1 позитивне, bi2 негативне
```

```
    else if ((bi1.data[pos] & 0x80000000) == 0 && (bi2.data[pos] & 0x80000000)
!= 0)
      return false;

    // той же знак
    int len = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;
    for (pos = len - 1; pos >= 0 && bi1.data[pos] == bi2.data[pos]; pos--) ;

    if (pos >= 0)
    {
      if (bi1.data[pos] < bi2.data[pos])
        return true;
      return false;
    }
    return false;
  }


  public static bool operator >=(BigInteger bi1, BigInteger bi2)
  {
    return (bi1 == bi2 || bi1 > bi2);
  }


  public static bool operator <=(BigInteger bi1, BigInteger bi2)
  {
    return (bi1 == bi2 || bi1 < bi2);
  }


  //*********************************************************************
  // Частна функція, яка підтримує ділення двох чисел з ділителем,що має більше
ніж одну цифру
  //
  //  //*********************************************************************

  private static void multiByteDivide(BigInteger bi1, BigInteger bi2, BigInteger
outQuotient, BigInteger outRemainder)
  {
    uint[] result = new uint[maxLength];

    int remainderLen = bi1.dataLength + 1;
    uint[] remainder = new uint[remainderLen];

    uint mask = 0x80000000;
    uint val = bi2.data[bi2.dataLength - 1];
    int shift = 0, resultPos = 0;

    while (mask != 0 && (val & mask) == 0)
    {
      shift++; mask >>= 1;
    }

    //Console.WriteLine("shift = {0}", shift);
    //Console.WriteLine("Before bi1 Len = {0}, bi2 Len = {1}", bi1.dataLength,
bi2.dataLength);

    for (int i = 0; i < bi1.dataLength; i++)
      remainder[i] = bi1.data[i];
    shiftLeft(remainder, shift);
    bi2 = bi2 << shift;

    /*
    Console.WriteLine("bi1 Len = {0}, bi2 Len = {1}", bi1.dataLength,
bi2.dataLength);
    Console.WriteLine("dividend = " + bi1 + "\ndivisor = " + bi2);
    for(int q = remainderLen - 1; q >= 0; q--)
            Console.Write("{0:x2}", remainder[q]);
```

```
    Console.WriteLine();
    */

    int j = remainderLen - bi2.dataLength;
    int pos = remainderLen - 1;

    ulong firstDivisorByte = bi2.data[bi2.dataLength - 1];
    ulong secondDivisorByte = bi2.data[bi2.dataLength - 2];

    int divisorLen = bi2.dataLength + 1;
    uint[] dividendPart = new uint[divisorLen];

    while (j > 0)
    {
      ulong dividend = ((ulong)remainder[pos] << 32) + (ulong)remainder[pos -
1];
      //Console.WriteLine("dividend = {0}", dividend);

      ulong q_hat = dividend / firstDivisorByte;
      ulong r_hat = dividend % firstDivisorByte;

      //Console.WriteLine("q_hat = {0:X}, r_hat = {1:X}", q_hat, r_hat);

      bool done = false;
      while (!done)
      {
        done = true;

        if (q_hat == 0x100000000 ||
           (q_hat * secondDivisorByte) > ((r_hat << 32) + remainder[pos - 2]))
        {
          q_hat--;
          r_hat += firstDivisorByte;

          if (r_hat < 0x100000000)
            done = false;
        }
      }

      for (int h = 0; h < divisorLen; h++)
        dividendPart[h] = remainder[pos - h];

      BigInteger kk = new BigInteger(dividendPart);
      BigInteger ss = bi2 * (long)q_hat;

      //Console.WriteLine("ss before = " + ss);
      while (ss > kk)
      {
        q_hat--;
        ss -= bi2;
        //Console.WriteLine(ss);
      }
      BigInteger yy = kk - ss;

      //Console.WriteLine("ss = " + ss);
      //Console.WriteLine("kk = " + kk);
      //Console.WriteLine("yy = " + yy);

      for (int h = 0; h < divisorLen; h++)
        remainder[pos - h] = yy.data[bi2.dataLength - h];

      /*
      Console.WriteLine("dividend = ");
      for(int q = remainderLen - 1; q >= 0; q--)
             Console.Write("{0:x2}", remainder[q]);
      Console.WriteLine("\n************ q_hat = {0:X}\n", q_hat);
      */

      result[resultPos++] = (uint)q_hat;
```

```
      pos--;
      j--;
    }

    outQuotient.dataLength = resultPos;
    int y = 0;
    for (int x = outQuotient.dataLength - 1; x >= 0; x--, y++)
      outQuotient.data[y] = result[x];
    for (; y < maxLength; y++)
      outQuotient.data[y] = 0;

    while (outQuotient.dataLength > 1 && outQuotient.data[outQuotient.dataLength
- 1] == 0)
      outQuotient.dataLength--;

    if (outQuotient.dataLength == 0)
      outQuotient.dataLength = 1;

    outRemainder.dataLength = shiftRight(remainder, shift);

    for (y = 0; y < outRemainder.dataLength; y++)
      outRemainder.data[y] = remainder[y];
    for (; y < maxLength; y++)
      outRemainder.data[y] = 0;
  }


  //*********************************************************************
  // Частна функція, яка підтримує ділення двох чисел з ділителем,що має тільки
одну цифру.
  //*********************************************************************

  private static void singleByteDivide(BigInteger bi1, BigInteger bi2,
BigInteger outQuotient, BigInteger outRemainder)
  {
    uint[] result = new uint[maxLength];
    int resultPos = 0;

    // Копіювання ділителя до пам'яті
    for (int i = 0; i < maxLength; i++)
      outRemainder.data[i] = bi1.data[i];
    outRemainder.dataLength = bi1.dataLength;

    while (outRemainder.dataLength > 1 &&
outRemainder.data[outRemainder.dataLength - 1] == 0)
      outRemainder.dataLength--;

    ulong divisor = (ulong)bi2.data[0];
    int pos = outRemainder.dataLength - 1;
    ulong dividend = (ulong)outRemainder.data[pos];

    //Console.WriteLine("divisor = " + divisor + " dividend = " + dividend);
    //Console.WriteLine("divisor = " + bi2 + "\ndividend = " + bi1);

    if (dividend >= divisor)
    {
      ulong quotient = dividend / divisor;
      result[resultPos++] = (uint)quotient;

      outRemainder.data[pos] = (uint)(dividend % divisor);
    }
    pos--;

    while (pos >= 0)
    {
      //Console.WriteLine(pos);
```

```
      dividend = ((ulong)outRemainder.data[pos + 1] << 32) +
(ulong)outRemainder.data[pos];
      ulong quotient = dividend / divisor;
      result[resultPos++] = (uint)quotient;

      outRemainder.data[pos + 1] = 0;
      outRemainder.data[pos--] = (uint)(dividend % divisor);
      //Console.WriteLine(">>>> " + bi1);
    }

    outQuotient.dataLength = resultPos;
    int j = 0;
    for (int i = outQuotient.dataLength - 1; i >= 0; i--, j++)
      outQuotient.data[j] = result[i];
    for (; j < maxLength; j++)
      outQuotient.data[j] = 0;

    while (outQuotient.dataLength > 1 && outQuotient.data[outQuotient.dataLength
- 1] == 0)
      outQuotient.dataLength--;

    if (outQuotient.dataLength == 0)
      outQuotient.dataLength = 1;

    while (outRemainder.dataLength > 1 &&
outRemainder.data[outRemainder.dataLength - 1] == 0)
      outRemainder.dataLength--;
  }


  //*********************************************************************
  // Переповнення оператора ділення
  //*********************************************************************

  public static BigInteger operator /(BigInteger bi1, BigInteger bi2)
  {
    BigInteger quotient = new BigInteger();
    BigInteger remainder = new BigInteger();

    int lastPos = maxLength - 1;
    bool divisorNeg = false, dividendNeg = false;

    if ((bi1.data[lastPos] & 0x80000000) != 0)     // bi1 негативне
    {
      bi1 = -bi1;
      dividendNeg = true;
    }
    if ((bi2.data[lastPos] & 0x80000000) != 0)     // bi2 негативне
    {
      bi2 = -bi2;
      divisorNeg = true;
    }

    if (bi1 < bi2)
    {
      return quotient;
    }

    else
    {
      if (bi2.dataLength == 1)
        singleByteDivide(bi1, bi2, quotient, remainder);
      else
        multiByteDivide(bi1, bi2, quotient, remainder);

      if (dividendNeg != divisorNeg)
        return -quotient;

      return quotient;
```

```
    }
  }


  //*********************************************************************
  // Переповнення оператора взяття по модулю
  //*********************************************************************

  public static BigInteger operator %(BigInteger bi1, BigInteger bi2)
  {
    BigInteger quotient = new BigInteger();
    BigInteger remainder = new BigInteger(bi1);

    int lastPos = maxLength - 1;
    bool dividendNeg = false;

    if ((bi1.data[lastPos] & 0x80000000) != 0)     // bi1 негативне
    {
      bi1 = -bi1;
      dividendNeg = true;
    }
    if ((bi2.data[lastPos] & 0x80000000) != 0)     // bi2 негативне
      bi2 = -bi2;

    if (bi1 < bi2)
    {
      return remainder;
    }

    else
    {
      if (bi2.dataLength == 1)
        singleByteDivide(bi1, bi2, quotient, remainder);
      else
        multiByteDivide(bi1, bi2, quotient, remainder);

      if (dividendNeg)
        return -remainder;

      return remainder;
    }
  }


  //*********************************************************************
  // Переповнення оператора побітового AND
  //*********************************************************************

  public static BigInteger operator &(BigInteger bi1, BigInteger bi2)
  {
    BigInteger result = new BigInteger();

    int len = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;

    for (int i = 0; i < len; i++)
    {
      uint sum = (uint)(bi1.data[i] & bi2.data[i]);
      result.data[i] = sum;
    }

    result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    return result;
  }
```

```
  //*********************************************************************
  // Переповнення оператора побітового OR
  //*********************************************************************

  public static BigInteger operator |(BigInteger bi1, BigInteger bi2)
  {
    BigInteger result = new BigInteger();

    int len = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;

    for (int i = 0; i < len; i++)
    {
      uint sum = (uint)(bi1.data[i] | bi2.data[i]);
      result.data[i] = sum;
    }

    result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    return result;
  }


  //*********************************************************************
  // Переповнення оператора побітового XOR
  //*********************************************************************

  public static BigInteger operator ^(BigInteger bi1, BigInteger bi2)
  {
    BigInteger result = new BigInteger();

    int len = (bi1.dataLength > bi2.dataLength) ? bi1.dataLength :
bi2.dataLength;

    for (int i = 0; i < len; i++)
    {
      uint sum = (uint)(bi1.data[i] ^ bi2.data[i]);
      result.data[i] = sum;
    }

    result.dataLength = maxLength;

    while (result.dataLength > 1 && result.data[result.dataLength - 1] == 0)
      result.dataLength--;

    return result;
  }


  //*********************************************************************
  // повертає max(this, bi)
  //*********************************************************************

  public BigInteger max(BigInteger bi)
  {
    if (this > bi)
      return (new BigInteger(this));
    else
      return (new BigInteger(bi));
  }


  //*********************************************************************
  // повертає min(this, bi)
  //*********************************************************************
```

```
public BigInteger min(BigInteger bi)
{
  if (this < bi)
    return (new BigInteger(this));
  else
    return (new BigInteger(bi));

}


//***************************************************************************
// Повертає абсолютне значення
//***************************************************************************

public BigInteger abs()
{
  if ((this.data[maxLength - 1] & 0x80000000) != 0)
    return (-this);
  else
    return (new BigInteger(this));
}


//***************************************************************************
// повертає рядок представлений у BigInteger у базі 10.
//***************************************************************************

public override string ToString()
{
  return ToString(10);
}


//***************************************************************************
// повертає рядок представлений  у BigInteger у sign-and-magnitude
// форматі у певному вигляді.
//
// Приклад
// -------
// Якщо значення BigInteger is -255 у базі 10, то
// ToString(16) повертає "-FF"
//
//***************************************************************************

public string ToString(int radix)
{
  if (radix < 2 || radix > 36)
    throw (new ArgumentException("Radix має бути >= 2 and <= 36"));

  string charSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  string result = "";

  BigInteger a = this;

  bool негативне = false;
  if ((a.data[maxLength - 1] & 0x80000000) != 0)
  {
    negative = true;
    try
    {
      a = -a;
    }
    catch (Exception) { }
  }

  BigInteger quotient = new BigInteger();
  BigInteger remainder = new BigInteger();
  BigInteger biRadix = new BigInteger(radix);
```

```
  if (a.dataLength == 1 && a.data[0] == 0)
    result = "0";
  else
  {
    while (a.dataLength > 1 || (a.dataLength == 1 && a.data[0] != 0))
    {
      singleByteDivide(a, biRadix, quotient, remainder);

      if (remainder.data[0] < 10)
        result = remainder.data[0] + result;
      else
        result = charSet[(int)remainder.data[0] - 10] + result;

      a = quotient;
    }
    if (negative)
      result = "-" + result;
  }

  return result;
}


//*****************************************************************************
// повертає a hex рядок, який містить BigInteger
//
// приклади
// -------
// 1) Якщо значення BigInteger є 255 у базі 10, тоді
//    ToHexString() повертає "FF"
//
// 2) Якщо значення BigInteger є -255 у базі 10, тоді
//    ToHexString() повертає ".....FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF01",
//    які є дві складові представляючі -255.
//
//*****************************************************************************

public string ToHexString()
{
  string result = data[dataLength - 1].ToString("X");

  for (int i = dataLength - 2; i >= 0; i--)
  {
    result += data[i].ToString("X8");
  }

  return result;
}



//*****************************************************************************
// Введення у ступінь по модулю
//*****************************************************************************

public BigInteger modPow(BigInteger exp, BigInteger n)
{
  if ((exp.data[maxLength - 1] & 0x80000000) != 0)
      throw (new ArithmeticException("Тільки позитивні показники"));

  BigInteger resultNum = 1;
  BigInteger tempNum;
  bool thisnegative = false;

  if ((this.data[maxLength - 1] & 0x80000000) != 0)   // негативне this
  {
    tempNum = -this % n;
    thisnegative = true;
```

```
    }
    else
      tempNum = this % n;  // ensures (tempNum * tempNum) < b^(2k)

    if ((n.data[maxLength - 1] & 0x80000000) != 0)   // негативне n
      n = -n;

    // розраховує constant = b^(2k) / m
    BigInteger constant = new BigInteger();

    int i = n.dataLength << 1;
    constant.data[i] = 0x00000001;
    constant.dataLength = i + 1;

    constant = constant / n;
    int totalBits = exp.bitCount();
    int count = 0;

    // виконує возведення у квадрат та возведення у ступінь
    for (int pos = 0; pos < exp.dataLength; pos++)
    {
      uint mask = 0x01;
      //Console.WriteLine("pos = " + pos);

      for (int index = 0; index < 32; index++)
      {
        if ((exp.data[pos] & mask) != 0)
          resultNum = BarrettReduction(resultNum * tempNum, n, constant);

        mask <<= 1;

        tempNum = BarrettReduction(tempNum * tempNum, n, constant);


        if (tempNum.dataLength == 1 && tempNum.data[0] == 1)
        {
          if (thisnegative && (exp.data[0] & 0x1) != 0)    //odd exp
            return -resultNum;
          return resultNum;
        }
        count++;
        if (count == totalBits)
          break;
      }
    }

    if (thisnegative && (exp.data[0] & 0x1) != 0)    //odd exp
      return -resultNum;

    return resultNum;
  }



  //*********************************************************************
  // Швидке розрахування модульного зменшення, використовує зменшення Баретта..
  // Потребується x < b^(2k), де b у базі.  У цьому випадку база є
  // 2^32 (модуль).
  //
  //*********************************************************************

  private BigInteger BarrettReduction(BigInteger x, BigInteger n, BigInteger
constant)
  {
    int k = n.dataLength,
        kPlusOne = k + 1,
        kMinusOne = k - 1;

    BigInteger q1 = new BigInteger();
```

```
// q1 = x / b^(k-1)
for (int i = kMinusOne, j = 0; i < x.dataLength; i++, j++)
  q1.data[j] = x.data[i];
q1.dataLength = x.dataLength - kMinusOne;
if (q1.dataLength <= 0)
  q1.dataLength = 1;


BigInteger q2 = q1 * constant;
BigInteger q3 = new BigInteger();

// q3 = q2 / b^(k+1)
for (int i = kPlusOne, j = 0; i < q2.dataLength; i++, j++)
  q3.data[j] = q2.data[i];
q3.dataLength = q2.dataLength - kPlusOne;
if (q3.dataLength <= 0)
  q3.dataLength = 1;


// r1 = x mod b^(k+1)
// тобто отримує нижні(k+1) слова
BigInteger r1 = new BigInteger();
int lengthToCopy = (x.dataLength > kPlusOne) ? kPlusOne : x.dataLength;
for (int i = 0; i < lengthToCopy; i++)
  r1.data[i] = x.data[i];
r1.dataLength = lengthToCopy;


// r2 = (q3 * n) mod b^(k+1)
// часткове множення q3 та n

BigInteger r2 = new BigInteger();
for (int i = 0; i < q3.dataLength; i++)
{
  if (q3.data[i] == 0) continue;

  ulong mcarry = 0;
  int t = i;
  for (int j = 0; j < n.dataLength && t < kPlusOne; j++, t++)
  {
    // t = i + j
    ulong val = ((ulong)q3.data[i] * (ulong)n.data[j]) +
                (ulong)r2.data[t] + mcarry;

    r2.data[t] = (uint)(val & 0xFFFFFFFF);
    mcarry = (val >> 32);
  }

  if (t < kPlusOne)
    r2.data[t] = (uint)mcarry;
}
r2.dataLength = kPlusOne;
while (r2.dataLength > 1 && r2.data[r2.dataLength - 1] == 0)
  r2.dataLength--;

r1 -= r2;
if ((r1.data[maxLength - 1] & 0x80000000) != 0)        // негативне
{
  BigInteger val = new BigInteger();
  val.data[kPlusOne] = 0x00000001;
  val.dataLength = kPlusOne + 1;
  r1 += val;
}

while (r1 >= n)
  r1 -= n;

return r1;
```

```
  }

  //*********************************************************************
  // повертає gcd(this, bi) - НЗД - Найбільший загальний дільник
  //*********************************************************************

  public BigInteger gcd(BigInteger bi)
  {
    BigInteger x;
    BigInteger y;

    if ((data[maxLength - 1] & 0x80000000) != 0)     // негативне
      x = -this;
    else
      x = this;

    if ((bi.data[maxLength - 1] & 0x80000000) != 0)     // негативне
      y = -bi;
    else
      y = bi;

    BigInteger g = y;

    while (x.dataLength > 1 || (x.dataLength == 1 && x.data[0] != 0))
    {
      g = x;
      x = y % x;
      y = g;
    }

    return g;
  }


  //*********************************************************************
  // Заповнює з визначеною сумою довільних біт
  //*********************************************************************

  public void genRandomBits(int bits, Random rand)
  {
    int dwords = bits >> 5;
    int remBits = bits & 0x1F;

    if (remBits != 0)
      dwords++;

    if (dwords > maxLength)
        throw (new ArithmeticException("Кількість необхідних бітів>
MaxLength."));

    for (int i = 0; i < dwords; i++)
      data[i] = (uint)(rand.NextDouble() * 0x100000000);

    for (int i = dwords; i < maxLength; i++)
      data[i] = 0;

    if (remBits != 0)
    {
      uint mask = (uint)(0x01 << (remBits - 1));
      data[dwords - 1] |= mask;

      mask = (uint)(0xFFFFFFFF >> (32 - remBits));
      data[dwords - 1] &= mask;
    }
    else
      data[dwords - 1] |= 0x80000000;

    dataLength = dwords;
```

```
  if (dataLength == 0)
    dataLength = 1;
}


//*********************************************************************
// Повертає позицію останього найбільш значущого біту у BigInteger.
//
// Результат дорівнює 0 , якщо значення BigInteger дорівнює 0 ...0000 0000
//      Результат у 1, якщо значення BigInteger дорівнює 0 ...0000 0001
//      Результат у 2, якщо значення BigInteger дорівнює 0 ...0000 0010
//      Результат у 3, якщо значення BigInteger дорівнює 0 ...0000 0011
//
//*********************************************************************

public int bitCount()
{
  while (dataLength > 1 && data[dataLength - 1] == 0)
    dataLength--;

  uint value = data[dataLength - 1];
  uint mask = 0x80000000;
  int bits = 32;

  while (bits > 0 && (value & mask) == 0)
  {
    bits--;
    mask >>= 1;
  }
  bits += ((dataLength - 1) << 5);

  return bits;
}


//*********************************************************************
// перевірка псевдовипдковості чисел використовуючи малу теорему Ферма
//*********************************************************************

public bool FermatLittleTest(int confidence)
{
  BigInteger thisVal;
  if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
    thisVal = -this;
  else
    thisVal = this;

  if (thisVal.dataLength == 1)
  {
    // тестуємо малі числа
    if (thisVal.data[0] == 0 || thisVal.data[0] == 1)
      return false;
    else if (thisVal.data[0] == 2 || thisVal.data[0] == 3)
      return true;
  }

  if ((thisVal.data[0] & 0x1) == 0)     // парні числа
    return false;

  int bits = thisVal.bitCount();
  BigInteger a = new BigInteger();
  BigInteger p_sub1 = thisVal - (new BigInteger(1));
  Random rand = new Random();

  for (int round = 0; round < confidence; round++)
  {
    bool done = false;
```

```
    while (!done)           // генеруємо a < n
    {
      int testBits = 0;

      // переконайтесь, що "a" має щонайменше 2 біта
      while (testBits < 2)
        testBits = (int)(rand.NextDouble() * bits);

      a.genRandomBits(testBits, rand);

      int byteLen = a.dataLength;

      // переконайтесь, що "a" не дорівнює  0
      if (byteLen > 1 || (byteLen == 1 && a.data[0] != 1))
        done = true;
    }

    // перевірка існуючого вказівника
    BigInteger gcdTest = a.gcd(thisVal);
    if (gcdTest.dataLength == 1 && gcdTest.data[0] != 1)
      return false;

    // розрахунок a^(p-1) mod p
    BigInteger expResult = a.modPow(p_sub1, thisVal);

    int resultLen = expResult.dataLength;

    // не дорівнює  просте є a^(p-1) mod p != 1

    if (resultLen > 1 || (resultLen == 1 && expResult.data[0] != 1))
    {
      //Console.WriteLine("a = " + a.ToString());
      return false;
    }
  }

  return true;
}


//***********************************************************************
// Перевірка простоти заснована на тесті Рабіна-Мілера
//
//
//***********************************************************************

public bool RabinMillerTest(int confidence)
{
  BigInteger thisVal;
  if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
    thisVal = -this;
  else
    thisVal = this;

  if (thisVal.dataLength == 1)
  {
    // тестування малих чисел
    if (thisVal.data[0] == 0 || thisVal.data[0] == 1)
      return false;
    else if (thisVal.data[0] == 2 || thisVal.data[0] == 3)
      return true;
  }

  if ((thisVal.data[0] & 0x1) == 0)     // парні числа
    return false;


  // розраховує значення s та t
  BigInteger p_sub1 = thisVal - (new BigInteger(1));
```

```
    int s = 0;

    for (int index = 0; index < p_sub1.dataLength; index++)
    {
      uint mask = 0x01;

      for (int i = 0; i < 32; i++)
      {
        if ((p_sub1.data[index] & mask) != 0)
        {
          index = p_sub1.dataLength;      // Останавлює зовнішній цикл
          break;
        }
        mask <<= 1;
        s++;
      }
    }

    BigInteger t = p_sub1 >> s;

    int bits = thisVal.bitCount();
    BigInteger a = new BigInteger();
    Random rand = new Random();

    for (int round = 0; round < confidence; round++)
    {
      bool done = false;

      while (!done)           // генеруємо a < n
      {
        int testBits = 0;

        // переконайтесь, що "a" має щонайменше 2 біти
        while (testBits < 2)
          testBits = (int)(rand.NextDouble() * bits);

        a.genRandomBits(testBits, rand);

        int byteLen = a.dataLength;

        // переконайтесь, що "a" не дорівнює  0
        if (byteLen > 1 || (byteLen == 1 && a.data[0] != 1))
          done = true;
      }

      // перевірка існуючого вказівника
      BigInteger gcdTest = a.gcd(thisVal);
      if (gcdTest.dataLength == 1 && gcdTest.data[0] != 1)
        return false;

      BigInteger b = a.modPow(t, thisVal);

      /*
      Console.WriteLine("a = " + a.ToString(10));
      Console.WriteLine("b = " + b.ToString(10));
      Console.WriteLine("t = " + t.ToString(10));
      Console.WriteLine("s = " + s);
      */

      bool result = false;

      if (b.dataLength == 1 && b.data[0] == 1)         // a^t mod p = 1
        result = true;

      for (int j = 0; result == false && j < s; j++)
      {
        if (b == p_sub1)         // a^((2^j)*t) mod p = p-1 for some 0 <= j <=
s-1
        {
```

```
        result = true;
        break;
      }

      b = (b * b) % thisVal;
    }

    if (result == false)
      return false;
  }
  return true;
}


//*****************************************************************************
// Перевірка простоти заснована на методі Соловея Страсена (Критерій Ейлера)
//*****************************************************************************

public bool SolovayStrassenTest(int confidence)
{
  BigInteger thisVal;
  if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
    thisVal = -this;
  else
    thisVal = this;

  if (thisVal.dataLength == 1)
  {
    // тестування малих чисел
    if (thisVal.data[0] == 0 || thisVal.data[0] == 1)
      return false;
    else if (thisVal.data[0] == 2 || thisVal.data[0] == 3)
      return true;
  }

  if ((thisVal.data[0] & 0x1) == 0)     // парні числа
    return false;


  int bits = thisVal.bitCount();
  BigInteger a = new BigInteger();
  BigInteger p_sub1 = thisVal - 1;
  BigInteger p_sub1_shift = p_sub1 >> 1;

  Random rand = new Random();

  for (int round = 0; round < confidence; round++)
  {
    bool done = false;

    while (!done)           // генеруємо a < n
    {
      int testBits = 0;

      // переконайтесь, що "a" має щонайменше 2 біта
      while (testBits < 2)
        testBits = (int)(rand.NextDouble() * bits);

      a.genRandomBits(testBits, rand);

      int byteLen = a.dataLength;

      // переконайтесь, що "a" не дорівнює  0
      if (byteLen > 1 || (byteLen == 1 && a.data[0] != 1))
        done = true;
    }

    // перевірка існуючого вказівника
    BigInteger gcdTest = a.gcd(thisVal);
```

```
      if (gcdTest.dataLength == 1 && gcdTest.data[0] != 1)
        return false;

      // розрахунок a^((p-1)/2) mod p

      BigInteger expResult = a.modPow(p_sub1_shift, thisVal);
      if (expResult == p_sub1)
        expResult = -1;

      // розрахунок символу Якобі
      BigInteger jacob = Jacobi(a, thisVal);

      //Console.WriteLine("a = " + a.ToString(10) + " b = " +
thisVal.ToString(10));
      //Console.WriteLine("expResult = " + expResult.ToString(10) + " Jacob = "
+ jacob.ToString(10));

      // Якщо they are different then it не дорівнює  prime
      if (expResult != jacob)
        return false;
    }

    return true;
  }


  //*********************************************************************
  // Реалізація тесту псевдосильних чисел Лукаса
  //
  //*********************************************************************

  public bool LucasStrongTest()
  {
    BigInteger thisVal;
    if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
      thisVal = -this;
    else
      thisVal = this;

    if (thisVal.dataLength == 1)
    {
      // тестування малих чисел
      if (thisVal.data[0] == 0 || thisVal.data[0] == 1)
        return false;
      else if (thisVal.data[0] == 2 || thisVal.data[0] == 3)
        return true;
    }

    if ((thisVal.data[0] & 0x1) == 0)     // парні числа
      return false;

    return LucasStrongTestHelper(thisVal);
  }


  private bool LucasStrongTestHelper(BigInteger thisVal)
  {
    // Зроблено тестом (вибране D)
    // Встановлене D перший елемент послідовності
    // 5, -7, 9, -11, 13, ... Для якої J(D,n) = -1
    // Встановимо P = 1, Q = (1-D) / 4

    long D = 5, sign = -1, dCount = 0;
    bool done = false;

    while (!done)
    {
      int Jresult = BigInteger.Jacobi(D, thisVal);
```

```
  if (Jresult == -1)
    done = true;    // J(D, this) = 1
  else
  {
    if (Jresult == 0 && Math.Abs(D) < thisVal)       // визначен дільник
      return false;

    if (dCount == 20)
    {
      // перевірка введення у квадрат
      BigInteger root = thisVal.sqrt();
      if (root * root == thisVal)
        return false;
    }

    //Console.WriteLine(D);
    D = (Math.Abs(D) + 2) * sign;
    sign = -sign;
  }
  dCount++;
}

long Q = (1 - D) >> 2;

/*
Console.WriteLine("D = " + D);
Console.WriteLine("Q = " + Q);
Console.WriteLine("(n,D) = " + thisVal.gcd(D));
Console.WriteLine("(n,Q) = " + thisVal.gcd(Q));
Console.WriteLine("J(D|n) = " + BigInteger.Jacobi(D, thisVal));
*/

BigInteger p_add1 = thisVal + 1;
int s = 0;

for (int index = 0; index < p_add1.dataLength; index++)
{
  uint mask = 0x01;

  for (int i = 0; i < 32; i++)
  {
    if ((p_add1.data[index] & mask) != 0)
    {
      index = p_add1.dataLength;      // Останавлює зовнішній цикл
      break;
    }
    mask <<= 1;
    s++;
  }
}

BigInteger t = p_add1 >> s;

// розрахунок constant = b^(2k) / m
// для зменшення Баретта
BigInteger constant = new BigInteger();

int nLen = thisVal.dataLength << 1;
constant.data[nLen] = 0x00000001;
constant.dataLength = nLen + 1;

constant = constant / thisVal;

BigInteger[] lucas = LucasSequenceHelper(1, Q, t, thisVal, constant, 0);
bool isPrime = false;

if ((lucas[0].dataLength == 1 && lucas[0].data[0] == 0) ||
   (lucas[1].dataLength == 1 && lucas[1].data[0] == 0))
{
```

```
      // u(t) = 0 or V(t) = 0
      isPrime = true;
    }

    for (int i = 1; i < s; i++)
    {
      if (!isPrime)
      {
        // Дублює індекси
        lucas[1] = thisVal.BarrettReduction(lucas[1] * lucas[1], thisVal,
constant);
        lucas[1] = (lucas[1] - (lucas[2] << 1)) % thisVal;

        //lucas[1] = ((lucas[1] * lucas[1]) - (lucas[2] << 1)) % thisVal;

        if ((lucas[1].dataLength == 1 && lucas[1].data[0] == 0))
          isPrime = true;
      }

      lucas[2] = thisVal.BarrettReduction(lucas[2] * lucas[2], thisVal,
constant);     //Q^k
    }


    if (isPrime)     // Додавання перевіки на композитність чисел
    {
      // Якщо n is prime and gcd(n, Q) == 1, then
      // Q^((n+1)/2) = Q * Q^((n-1)/2) is congruent to (Q * J(Q, n)) mod n

      BigInteger g = thisVal.gcd(Q);
      if (g.dataLength == 1 && g.data[0] == 1)        // gcd(this, Q) == 1
      {
        if ((lucas[2].data[maxLength - 1] & 0x80000000) != 0)
          lucas[2] += thisVal;

        BigInteger temp = (Q * BigInteger.Jacobi(Q, thisVal)) % thisVal;
        if ((temp.data[maxLength - 1] & 0x80000000) != 0)
          temp += thisVal;

        if (lucas[2] != temp)
          isPrime = false;
      }
    }

    return isPrime;
  }


  //***********************************************************************
  // Визначає чи є число ймовірно початком, для використання тесту Рабіна-
Мілера. Перед застосуванням тесту, число протестовано на делимість простим <
2000
  //
  // повертає true якщо число підходить.
  //***********************************************************************

  public bool isProbablePrime(int confidence)
  {
    BigInteger thisVal;
    if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
      thisVal = -this;
    else
      thisVal = this;


    // тест на делимість простого числа < 2000
    for (int p = 0; p < primesBelow2000.Length; p++)
    {
      BigInteger divisor = primesBelow2000[p];
```

```
      if (divisor >= thisVal)
        break;

      BigInteger resultNum = thisVal % divisor;
      if (resultNum.IntValue() == 0)
      {
        /*
Console.WriteLine("Не просте!  Ділиться на {0}\n",
                          primesBelow2000[p]);
        */
        return false;
      }
    }

    if (thisVal.RabinMillerTest(confidence))
      return true;
    else
    {
      //Console.WriteLine("Не просте!  Помилка проходження тестуна простоту\n");
      return false;
    }
  }


  //////////////////////////////////////////////////////////////////////
  // Визначає що BigInteger - імовірно просте число використовуюче комбінацію
двох базових тестів: тест на сильне просте число та тест Лукаса на сильне просте
число
  //
  //////////////////////////////////////////////////////////////////////

  public bool isProbablePrime()
  {
    BigInteger thisVal;
    if ((this.data[maxLength - 1] & 0x80000000) != 0)        // негативне
      thisVal = -this;
    else
      thisVal = this;

    if (thisVal.dataLength == 1)
    {
      // тестування малих чисел
      if (thisVal.data[0] == 0 || thisVal.data[0] == 1)
        return false;
      else if (thisVal.data[0] == 2 || thisVal.data[0] == 3)
        return true;
    }

    if ((thisVal.data[0] & 0x1) == 0)     // парні числа
      return false;


    // тест на делимість простого числа < 2000
    for (int p = 0; p < primesBelow2000.Length; p++)
    {
      BigInteger divisor = primesBelow2000[p];

      if (divisor >= thisVal)
        break;

      BigInteger resultNum = thisVal % divisor;
      if (resultNum.IntValue() == 0)
      {
        //Console.WriteLine("Не просте! ділиться на {0}\n",
        //                  primesBelow2000[p]);

        return false;
      }
```

```
    }

    // виконує базовий 2 Тест Рабіна-Мілера

    // розраховує значення s та t
    BigInteger p_sub1 = thisVal - (new BigInteger(1));
    int s = 0;

    for (int index = 0; index < p_sub1.dataLength; index++)
    {
      uint mask = 0x01;

      for (int i = 0; i < 32; i++)
      {
        if ((p_sub1.data[index] & mask) != 0)
        {
          index = p_sub1.dataLength;      // Останавлює зовнішній цикл
          break;
        }
        mask <<= 1;
        s++;
      }
    }

    BigInteger t = p_sub1 >> s;

    int bits = thisVal.bitCount();
    BigInteger a = 2;

    // b = a^t mod p
    BigInteger b = a.modPow(t, thisVal);
    bool result = false;

    if (b.dataLength == 1 && b.data[0] == 1)         // a^t mod p = 1
      result = true;

    for (int j = 0; result == false && j < s; j++)
    {
      if (b == p_sub1)         // a^((2^j)*t) mod p = p-1 для деяких 0 <= j <=
s-1
      {
        result = true;
        break;
      }

      b = (b * b) % thisVal;
    }

    // Якщо число є сильним простим числом , то воно підвергається тесту Лукаса
    if (result)
      result = LucasStrongTestHelper(thisVal);

    return result;
  }



  //***********************************************************************
  // Повертає молодші 4 байта BigInteger у типу int.
  //***********************************************************************

  public int IntValue()
  {
    return (int)data[0];
  }


  //***********************************************************************
  // Повертає молодші 8 байтів BigInteger типу long.
```

```
  //*********************************************************************

  public long LongValue()
  {
    long val = 0;

    val = (long)data[0];
    try
    {       // exception if maxLength = 1
      val |= (long)data[1] << 32;
    }
    catch (Exception)
    {
      if ((data[0] & 0x80000000) != 0) // негативне
        val = (int)data[0];
    }

    return val;
  }


  //*********************************************************************
  // розраховує символ Якобі для  a та b.
  //*********************************************************************

  public static int Jacobi(BigInteger a, BigInteger b)
  {
    // Якобі визначається тільки для непарних цілих чисел
    if ((b.data[0] & 0x1) == 0)
      throw (new ArgumentException("Якобі визначається тільки для непарних цілих
чисел "));

    if (a >= b) a %= b;
    if (a.dataLength == 1 && a.data[0] == 0) return 0;  // a == 0
    if (a.dataLength == 1 && a.data[0] == 1) return 1;  // a == 1

    if (a < 0)
    {
      if ((((b - 1).data[0]) & 0x2) == 0)       //if( (((b-1) >> 1).data[0] &
0x1) == 0)
        return Jacobi(-a, b);
      else
        return -Jacobi(-a, b);
    }

    int e = 0;
    for (int index = 0; index < a.dataLength; index++)
    {
      uint mask = 0x01;

      for (int i = 0; i < 32; i++)
      {
        if ((a.data[index] & mask) != 0)
        {
          index = a.dataLength;      // Останавлює зовнішній цикл
          break;
        }
        mask <<= 1;
        e++;
      }
    }

    BigInteger a1 = a >> e;

    int s = 1;
    if ((e & 0x1) != 0 && ((b.data[0] & 0x7) == 3 || (b.data[0] & 0x7) == 5))
      s = -1;

    if ((b.data[0] & 0x3) == 3 && (a1.data[0] & 0x3) == 3)
```

```
      s = -s;

    if (a1.dataLength == 1 && a1.data[0] == 1)
      return s;
    else
      return (s * Jacobi(b % a1, a1));
  }
```

```
//*********************************************************************
  // Генерує в позитивному BigInteger попередньо просте число.
  //*********************************************************************

  public static BigInteger genPseudoPrime(int bits, int confidence, Random rand)
  {
    BigInteger result = new BigInteger();
    bool done = false;

    while (!done)
    {
      result.genRandomBits(bits, rand);
      result.data[0] |= 0x01;       // make it odd

      // тест на простоту
      done = result.isProbablePrime(confidence);
    }
    return result;
  }


  //*********************************************************************
  // Генерує випадкове число використовуючи gcd(number, this) = 1
  //*********************************************************************

  public BigInteger genCoPrime(int bits, Random rand)
  {
    bool done = false;
    BigInteger result = new BigInteger();

    while (!done)
    {
      result.genRandomBits(bits, rand);
      //Console.WriteLine(result.ToString(16));

      // gcd тест
      BigInteger g = result.gcd(this);
      if (g.dataLength == 1 && g.data[0] == 1)
        done = true;
    }

    return result;
  }


  //*********************************************************************
  // Повертає зворотнє по модулю.  ставиться ArithmeticException якщо
  // інверсію неможливо виконати.  (gcd(this, modulus) != 1)
  //*********************************************************************

  public BigInteger modInverse(BigInteger modulus)
  {
    BigInteger[] p = { 0, 1 };
    BigInteger[] q = new BigInteger[2];    // часні
    BigInteger[] r = { 0, 0 };             // загальні
```

```
    int step = 0;

    BigInteger a = modulus;
    BigInteger b = this;

    while (b.dataLength > 1 || (b.dataLength == 1 && b.data[0] != 0))
    {
      BigInteger quotient = new BigInteger();
      BigInteger remainder = new BigInteger();

      if (step > 1)
      {
        BigInteger pval = (p[0] - (p[1] * q[0])) % modulus;
        p[0] = p[1];
        p[1] = pval;
      }

      if (b.dataLength == 1)
        singleByteDivide(a, b, quotient, remainder);
      else
        multiByteDivide(a, b, quotient, remainder);

      /*
      Console.WriteLine(quotient.dataLength);
      Console.WriteLine("{0} = {1}({2}) + {3}  p = {4}", a.ToString(10),
                        b.ToString(10), quotient.ToString(10),
remainder.ToString(10),
                        p[1].ToString(10));
      */

      q[0] = q[1];
      r[0] = r[1];
      q[1] = quotient; r[1] = remainder;

      a = b;
      b = remainder;

      step++;
    }

    if (r[0].dataLength > 1 || (r[0].dataLength == 1 && r[0].data[0] != 1))
      throw (new ArithmeticException("No inverse!"));

    BigInteger result = ((p[0] - (p[1] * q[0])) % modulus);

    if ((result.data[maxLength - 1] & 0x80000000) != 0)
      result += modulus;  // отримано найменші позитивні модулі

    return result;
  }


  //*********************************************************************
  // Повертає значення BigInteger у масиві байтів.  Нижній
  // індекс  міститься у MSB.
  //*********************************************************************

  public byte[] getBytes()
  {
    int numBits = bitCount();

    int numBytes = numBits >> 3;
    if ((numBits & 0x7) != 0)
      numBytes++;

    byte[] result = new byte[numBytes];

    //Console.WriteLine(result.Length);
```

```
  int pos = 0;
  uint tempVal, val = data[dataLength - 1];

  if ((tempVal = (val >> 24 & 0xFF)) != 0)
    result[pos++] = (byte)tempVal;
  if ((tempVal = (val >> 16 & 0xFF)) != 0)
    result[pos++] = (byte)tempVal;
  else if (pos > 0)
    pos++;
  if ((tempVal = (val >> 8 & 0xFF)) != 0)
    result[pos++] = (byte)tempVal;
  else if (pos > 0)
    pos++;
  if ((tempVal = (val & 0xFF)) != 0)
    result[pos++] = (byte)tempVal;

  for (int i = dataLength - 2; i >= 0; i--, pos += 4)
  {
    val = data[i];
    result[pos + 3] = (byte)(val & 0xFF);
    val >>= 8;
    result[pos + 2] = (byte)(val & 0xFF);
    val >>= 8;
    result[pos + 1] = (byte)(val & 0xFF);
    val >>= 8;
    result[pos] = (byte)(val & 0xFF);
  }

  return result;
}


//*********************************************************************
// Встановлює значення спеціального біта у 1
// Найменша позиція значимого біту дорівнює 0 .
//*********************************************************************

public void setBit(uint bitNum)
{
  uint bytePos = bitNum >> 5;             // ділиться на 32
  byte bitPos = (byte)(bitNum & 0x1F);    // беремо молодші 5 бітів

  uint mask = (uint)1 << bitPos;
  this.data[bytePos] |= mask;

  if (bytePos >= this.dataLength)
    this.dataLength = (int)bytePos + 1;
}


//*********************************************************************
// Встановлює значення спеціального біта у 0
// Найменша позиція значимого біту дорівнює 0 .
//*********************************************************************

public void unsetBit(uint bitNum)
{
  uint bytePos = bitNum >> 5;

  if (bytePos < this.dataLength)
  {
    byte bitPos = (byte)(bitNum & 0x1F);

    uint mask = (uint)1 << bitPos;
    uint mask2 = 0xFFFFFFFF ^ mask;

    this.data[bytePos] &= mask2;

    if (this.dataLength > 1 && this.data[this.dataLength - 1] == 0)
```

```
        this.dataLength--;
      }
    }
    //*********************************************************************
    // повертає значення яке є еквівалентним цілому квадратному корню
    //  з BigInteger.
    //
    //
    //*********************************************************************

    public BigInteger sqrt()
    {
      uint numBits = (uint)this.bitCount();

      if ((numBits & 0x1) != 0)        // непарна кількість бітів
        numBits = (numBits >> 1) + 1;
      else
        numBits = (numBits >> 1);

      uint bytePos = numBits >> 5;
      byte bitPos = (byte)(numBits & 0x1F);

      uint mask;

      BigInteger result = new BigInteger();
      if (bitPos == 0)
        mask = 0x80000000;
      else
      {
        mask = (uint)1 << bitPos;
        bytePos++;
      }
      result.dataLength = (int)bytePos;

      for (int i = (int)bytePos - 1; i >= 0; i--)
      {
        while (mask != 0)
        {
          // догадка
          result.data[i] ^= mask;

          // догадка знімається, якщо квадрат більше цього
          if ((result * result) > this)
            result.data[i] ^= mask;

          mask >>= 1;
        }
        mask = 0x80000000;
      }
      return result;
    }


    //*********************************************************************
    // Повертає k_th число послідовності Лукаса зменшення по модулю  n.
    //
  //*********************************************************************

    public static BigInteger[] LucasSequence(BigInteger P, BigInteger Q,
  BigInteger k, BigInteger n)
    {
      if (k.dataLength == 1 && k.data[0] == 0)
      {
        BigInteger[] result = new BigInteger[3];

        result[0] = 0; result[1] = 2 % n; result[2] = 1 % n;
        return result;
      }
```

```
    // розрахунок constant = b^(2k) / m
    // для зменшення Баретта
    BigInteger constant = new BigInteger();

    int nLen = n.dataLength << 1;
    constant.data[nLen] = 0x00000001;
    constant.dataLength = nLen + 1;

    constant = constant / n;

    // розраховує значення s та t
    int s = 0;

    for (int index = 0; index < k.dataLength; index++)
    {
      uint mask = 0x01;

      for (int i = 0; i < 32; i++)
      {
        if ((k.data[index] & mask) != 0)
        {
          index = k.dataLength;      // Останавлює зовнішній цикл
          break;
        }
        mask <<= 1;
        s++;
      }
    }

    BigInteger t = k >> s;

    //Console.WriteLine("s = " + s + " t = " + t);
    return LucasSequenceHelper(P, Q, t, n, constant, s);
  }


  //************************************************************************
  // виконує пірахунок к-го елемента у послідовності Лукаса
  //
  // k повинно бути непарним.  Т.я. LSB == 1
  //************************************************************************

  private static BigInteger[] LucasSequenceHelper(BigInteger P, BigInteger Q,
                                                  BigInteger k, BigInteger n,
                                                  BigInteger constant, int s)
  {
    BigInteger[] result = new BigInteger[3];

    if ((k.data[0] & 0x00000001) == 0)
      throw (new ArgumentException("Аргумент k повиний бути непарним."));

    int numbits = k.bitCount();
    uint mask = (uint)0x1 << ((numbits & 0x1F) - 1);

    // v = v0, v1 = v1, u1 = u1, Q_k = Q^0

    BigInteger v = 2 % n, Q_k = 1 % n,
               v1 = P % n, u1 = Q_k;
    bool flag = true;

    for (int i = k.dataLength - 1; i >= 0; i--)     // ітерація у двійковому
розширенні k
    {
      //Console.WriteLine("раунд");
      while (mask != 0)
      {
        if (i == 0 && mask == 0x00000001)        // останій біт
          break;
```

```
        if ((k.data[i] & mask) != 0)             // біт встановлено
        {
          // індекс продубльовано з доповненням

          u1 = (u1 * v1) % n;

          v = ((v * v1) - (P * Q_k)) % n;
          v1 = n.BarrettReduction(v1 * v1, n, constant);
          v1 = (v1 - ((Q_k * Q) << 1)) % n;

          if (flag)
            flag = false;
          else
            Q_k = n.BarrettReduction(Q_k * Q_k, n, constant);

          Q_k = (Q_k * Q) % n;
        }
        else
        {
          // індекс продубльовано
          u1 = ((u1 * v) - Q_k) % n;

          v1 = ((v * v1) - (P * Q_k)) % n;
          v = n.BarrettReduction(v * v, n, constant);
          v = (v - (Q_k << 1)) % n;

          if (flag)
          {
            Q_k = Q % n;
            flag = false;
          }
          else
            Q_k = n.BarrettReduction(Q_k * Q_k, n, constant);
        }

        mask >>= 1;
      }
      mask = 0x80000000;
    }

    // у цій точці u1 = u(n+1) and v = v(n)
    // з тих пір як останій біт завжди буде дорівнювати 1, необхідно
трасформувати u1 до u(2n+1) та v до  v(2n+1)

    u1 = ((u1 * v) - Q_k) % n;
    v = ((v * v1) - (P * Q_k)) % n;
    if (flag)
      flag = false;
    else
      Q_k = n.BarrettReduction(Q_k * Q_k, n, constant);

    Q_k = (Q_k * Q) % n;


    for (int i = 0; i < s; i++)
    {
      // індекс продубльовано
      u1 = (u1 * v) % n;
      v = ((v * v) - (Q_k << 1)) % n;

      if (flag)
      {
        Q_k = Q % n;
        flag = false;
      }
      else
        Q_k = n.BarrettReduction(Q_k * Q_k, n, constant);
    }
```

```
  result[0] = u1;
  result[1] = v;
  result[2] = Q_k;

  return result;
}


//*********************************************************************
// Тест на коректність застосування Tests /, %, * та + операцій
//*********************************************************************

public static void MulDivTest(int rounds)
{
  Random rand = new Random();
  byte[] val = new byte[64];
  byte[] val2 = new byte[64];

  for (int count = 0; count < rounds; count++)
  {
    // генеруємо 2 числа випадкової довжини
    int t1 = 0;
    while (t1 == 0)
      t1 = (int)(rand.NextDouble() * 65);

    int t2 = 0;
    while (t2 == 0)
      t2 = (int)(rand.NextDouble() * 65);

    bool done = false;
    while (!done)
    {
      for (int i = 0; i < 64; i++)
      {
        if (i < t1)
          val[i] = (byte)(rand.NextDouble() * 256);
        else
          val[i] = 0;

        if (val[i] != 0)
          done = true;
      }
    }

    done = false;
    while (!done)
    {
      for (int i = 0; i < 64; i++)
      {
        if (i < t2)
          val2[i] = (byte)(rand.NextDouble() * 256);
        else
          val2[i] = 0;

        if (val2[i] != 0)
          done = true;
      }
    }

    while (val[0] == 0)
      val[0] = (byte)(rand.NextDouble() * 256);
    while (val2[0] == 0)
      val2[0] = (byte)(rand.NextDouble() * 256);

    Console.WriteLine(count);
    BigInteger bn1 = new BigInteger(val, t1);
    BigInteger bn2 = new BigInteger(val2, t2);
```

```
      // Визначіть значення та остаток від ділення
      // номер першого встановіть другим.

      BigInteger bn3 = bn1 / bn2;
      BigInteger bn4 = bn1 % bn2;

      // Перерахуйте числа
      BigInteger bn5 = (bn3 * bn2) + bn4;

      // Переконайтесь, що вони – ті ж
      if (bn5 != bn1)
      {
        Console.WriteLine("Помилка у " + count);
        Console.WriteLine(bn1 + "\n");
        Console.WriteLine(bn2 + "\n");
        Console.WriteLine(bn3 + "\n");
        Console.WriteLine(bn4 + "\n");
        Console.WriteLine(bn5 + "\n");
        return;
      }
    }
  }


  //*****************************************************************
  // Тест коректного застосування функції возведення у ступінь по модулю, яка
використовується у  RSA  шифруванні та дешифруванні (використовується для
розрахунку ключів шифрування та дешифрування).
  //*****************************************************************

  public static void RSATest(int rounds)
  {
    Random rand = new Random(1);
    byte[] val = new byte[64];

    // таємний та відкритий ключ
    BigInteger bi_e = new
BigInteger("a932b948feed4fb2b692609bd22164fc9edb59fae7880cc1eaff7b3c9626b7e5b241
c27a974833b2622ebe09beb451917663d47232488f23a117fc97720f1e7", 16);
    BigInteger bi_d = new
BigInteger("4adf2f7a89da93248509347d2ae506d683dd3a16357e859a980c4f77a4e2f7a01fae
289f13a851df6e9db5adaa60bfd2b162bbbe31f7c8f828261a6839311929d2cef4f864dde65e556c
e43c89bbbf9f1ac5511315847ce9cc8dc92470a747b8792d6a83b0092d2e5ebaf852c85cacf34278
efa99160f2f8aa7ee7214de07b7", 16);
    BigInteger bi_n = new
BigInteger("e8e77781f36a7b3188d711c2190b560f205a52391b3479cdb99fa010745cbeba5f2a
dc08e1de6bf38398a0487c4a73610d94ec36f17f3f46ad75e17bc1adfec99839589f45f95ccc94cb
2a5c500b477eb3323d8cfab0c8458c96f0147a45d27e45a4d11d54d77684f65d48f15fafcc1ba208
e71e921b9bd9017c16a5231af7f", 16);

    Console.WriteLine("e =\n" + bi_e.ToString(10));
    Console.WriteLine("\nd =\n" + bi_d.ToString(10));
    Console.WriteLine("\nn =\n" + bi_n.ToString(10) + "\n");

    for (int count = 0; count < rounds; count++)
    {
      // генеруємо дані випадкової довжини
      int t1 = 0;
      while (t1 == 0)
        t1 = (int)(rand.NextDouble() * 65);

      bool done = false;
      while (!done)
      {
        for (int i = 0; i < 64; i++)
        {
          if (i < t1)
            val[i] = (byte)(rand.NextDouble() * 256);
          else
```

```
            val[i] = 0;

          if (val[i] != 0)
            done = true;
        }
      }

      while (val[0] == 0)
        val[0] = (byte)(rand.NextDouble() * 256);

      Console.Write("Round = " + count);

      // шифруємо та дешифруємо дані
      BigInteger bi_data = new BigInteger(val, t1);
      BigInteger bi_encrypted = bi_data.modPow(bi_e, bi_n);
      BigInteger bi_decrypted = bi_encrypted.modPow(bi_d, bi_n);

      // порівняння
      if (bi_decrypted != bi_data)
      {
        Console.WriteLine("\nПомилка у раунді " + count);
        Console.WriteLine(bi_data + "\n");
        return;
      }
      Console.WriteLine(" <Тест пройдено>.");
    }

  }


  //*********************************************************************
  // Тестується правильна реалізація возведеня у ступінь по модулю та обратна
функція по модулю, яке використовується у шифруванні та дешифруванні RSA. Два
псевдовипадкові числа встановлені, але для кожного раунду тестування генеруються
окремі ключі
  //*********************************************************************

  public static void RSATest2(int rounds)
  {
    Random rand = new Random();
    byte[] val = new byte[64];

    byte[] pseudoPrime1 = {
                          (byte)0x85, (byte)0x84, (byte)0x64, (byte)0xFD,
(byte)0x70, (byte)0x6A,
                          (byte)0x9F, (byte)0xF0, (byte)0x94, (byte)0x0C,
(byte)0x3E, (byte)0x2C,
                          (byte)0x74, (byte)0x34, (byte)0x05, (byte)0xC9,
(byte)0x55, (byte)0xB3,
                          (byte)0x85, (byte)0x32, (byte)0x98, (byte)0x71,
(byte)0xF9, (byte)0x41,
                          (byte)0x21, (byte)0x5F, (byte)0x02, (byte)0x9E,
(byte)0xEA, (byte)0x56,
                          (byte)0x8D, (byte)0x8C, (byte)0x44, (byte)0xCC,
(byte)0xEE, (byte)0xEE,
                          (byte)0x3D, (byte)0x2C, (byte)0x9D, (byte)0x2C,
(byte)0x12, (byte)0x41,
                          (byte)0x1E, (byte)0xF1, (byte)0xC5, (byte)0x32,
(byte)0xC3, (byte)0xAA,
                          (byte)0x31, (byte)0x4A, (byte)0x52, (byte)0xD8,
(byte)0xE8, (byte)0xAF,
                          (byte)0x42, (byte)0xF4, (byte)0x72, (byte)0xA1,
(byte)0x2A, (byte)0x0D,
                          (byte)0x97, (byte)0xB1, (byte)0x31, (byte)0xB3,
                };

    byte[] pseudoPrime2 = {
                          (byte)0x99, (byte)0x98, (byte)0xCA, (byte)0xB8,
(byte)0x5E, (byte)0xD7,
```

```
                        (byte)0xE5, (byte)0xDC, (byte)0x28, (byte)0x5C,
(byte)0x6F, (byte)0x0E,
                        (byte)0x15, (byte)0x09, (byte)0x59, (byte)0x6E,
(byte)0x84, (byte)0xF3,
                        (byte)0x81, (byte)0xCD, (byte)0xDE, (byte)0x42,
(byte)0xDC, (byte)0x93,
                        (byte)0xC2, (byte)0x7A, (byte)0x62, (byte)0xAC,
(byte)0x6C, (byte)0xAF,
                        (byte)0xDE, (byte)0x74, (byte)0xE3, (byte)0xCB,
(byte)0x60, (byte)0x20,
                        (byte)0x38, (byte)0x9C, (byte)0x21, (byte)0xC3,
(byte)0xDC, (byte)0xC8,
                        (byte)0xA2, (byte)0x4D, (byte)0xC6, (byte)0x2A,
(byte)0x35, (byte)0x7F,
                        (byte)0xF3, (byte)0xA9, (byte)0xE8, (byte)0x1D,
(byte)0x7B, (byte)0x2C,
                        (byte)0x78, (byte)0xFA, (byte)0xB8, (byte)0x02,
(byte)0x55, (byte)0x80,
                        (byte)0x9B, (byte)0xC2, (byte)0xA5, (byte)0xCB,
                };
```

```
    BigInteger bi_p = new BigInteger(pseudoPrime1);
    BigInteger bi_q = new BigInteger(pseudoPrime2);
    BigInteger bi_pq = (bi_p - 1) * (bi_q - 1);
    BigInteger bi_n = bi_p * bi_q;

    for (int count = 0; count < rounds; count++)
    {
      // генеруємо таємний та відкритий ключ
      BigInteger bi_e = bi_pq.genCoPrime(512, rand);
      BigInteger bi_d = bi_e.modInverse(bi_pq);

      Console.WriteLine("\ne =\n" + bi_e.ToString(10));
      Console.WriteLine("\nd =\n" + bi_d.ToString(10));
      Console.WriteLine("\nn =\n" + bi_n.ToString(10) + "\n");

      // генеруємо дані випадкової довжини
      int t1 = 0;
      while (t1 == 0)
        t1 = (int)(rand.NextDouble() * 65);

      bool done = false;
      while (!done)
      {
        for (int i = 0; i < 64; i++)
        {
          if (i < t1)
            val[i] = (byte)(rand.NextDouble() * 256);
          else
            val[i] = 0;

          if (val[i] != 0)
            done = true;
        }
      }

      while (val[0] == 0)
        val[0] = (byte)(rand.NextDouble() * 256);

      Console.Write("Раунд = " + count);

      // шифруєм та дешифруєм дані
      BigInteger bi_data = new BigInteger(val, t1);
      BigInteger bi_encrypted = bi_data.modPow(bi_e, bi_n);
      BigInteger bi_decrypted = bi_encrypted.modPow(bi_d, bi_n);

      // порівняння
      if (bi_decrypted != bi_data)
```

```
      {
        Console.WriteLine("\nпомилка в раунді " + count);
        Console.WriteLine(bi_data + "\n");
        return;
      }
      Console.WriteLine(" <Пройдено>.");
    }

  }


  //*********************************************************************
  // Тестування на коректність реалізації методу  sqrt().
  //*********************************************************************

  public static void SqrtTest(int rounds)
  {
    Random rand = new Random();
    for (int count = 0; count < rounds; count++)
    {
      // генеруємо дані випадкової довжини
      int t1 = 0;
      while (t1 == 0)
        t1 = (int)(rand.NextDouble() * 1024);

      Console.Write("Раунд = " + count);

      BigInteger a = new BigInteger();
      a.genRandomBits(t1, rand);

      BigInteger b = a.sqrt();
      BigInteger c = (b + 1) * (b + 1);

      // check that b is the largest integer such that b*b <= a
      if (c <= a)
      {
        Console.WriteLine("\nПомилка в раунді " + count);
        Console.WriteLine(a + "\n");
        return;
      }
      Console.WriteLine(" <Тест пройдено>.");
    }
  }
}
```

Файл BerDecodeError.cs – обробка помилок

```
  using System;
using System.Text;
using System.Security.Permissions;
using System.Runtime.Serialization;

namespace CSInteropKeys
{
  [Serializable]
  public sealed class BerDecodeException : Exception, ISerializable
  {
    private int m_position;
    public int Position
    { get { return m_position; } }

    public override string повідомлення
    {
      get
      {
        StringBuilder sb = new StringBuilder(base.Message);
        sb.AppendFormat(" (Позиція {0}){1}",
          m_position, Environment.NewLine);
        return sb.ToString();
      }
    }
    public BerDecodeException()
      : base() { }
    public BerDecodeException(String message)
      : base(message) { }
    public BerDecodeException(String message, Exception ex)
      : base(message, ex) { }
    public BerDecodeException(String message, int position)
      : base(message) { m_position = position; }
    public BerDecodeException(String message, int position, Exception ex)
      : base(message, ex) { m_position = position; }
    private BerDecodeException(SerializationInfo info, StreamingContext context)
      : base(info, context)
    { m_position = info.GetInt32("Позиція"); }
    [SecurityPermission(SecurityAction.Demand, SerializationFormatter = true)]
    public override void GetObjectData(SerializationInfo info, StreamingContext
context)
    {
      base.GetObjectData(info, context);
      info.AddValue("Позиція", m_position);
    }
  }
}
```

Файл AboutBox.cs - вікно довідки про програму

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Reflection;

namespace CSInteropKeys
{
    partial class AboutBox : Form
    {
        public AboutBox()
        {
            InitializeComponent();

            //  Ініціалізація AboutBox на дисплей з даними про продукт.
            this.Text = String.Format("About {0}", AssemblyTitle);
            this.labelProductName.Text = AssemblyProduct;
            this.labelVersion.Text = String.Format("Version {0}",
AssemblyVersion);
            this.labelCopyright.Text = AssemblyCopyright;
            this.labelCompanyName.Text = AssemblyCompany;
            this.textBoxDescription.Text = AssemblyDescription;
        }

        #region Assembly Attribute Accessors

        public string AssemblyTitle
        {
            get
            {
                // Усі атрибути назви містяться у цій сборці
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttribut
e), false);
                // Якщо є по крайній мірі один атрибут назви
                if (attributes.Length > 0)
                {
                    // Вибираємо перший атрибут
                    AssemblyTitleAttribute titleAttribute =
(AssemblyTitleAttribute)attributes[0];
                    // Якщо він не дорівнює «» він повертає пустий рядок,
                    if (titleAttribute.Title != "")
                        return titleAttribute.Title;
                }

                return
System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeB
ase);
            }
        }
        public string AssemblyVersion
        {
            get
            {
                return
Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get
            {
                // Отримуємо усі описи атрибутів
```

```
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAt
tribute), false);
                // Якщо там не довільне значення атрибутів, то повертається
пустий рядок
                if (attributes.Length == 0)
                    return "";
                // Якщо там  значення атрибутів, то повертається його величина
                return
((AssemblyDescriptionAttribute)attributes[0]).Description;
            }
        }

        public string AssemblyProduct
        {
            get
            {
                // Отримуємо усі атрибути програмного продукту
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttrib
ute), false);
                // Якщо там не довільне значення атрибутів програмного продукту,
то повертається пустий рядок
                if (attributes.Length == 0)
                    return "";
                // Якщо там  значення атрибутів програмного продукту, то
повертається його величина
                return ((AssemblyProductAttribute)attributes[0]).Product;
            }
        }

        public string AssemblyCopyright
        {
            get
            {
                // Отримуємо усі копірайтні атрибути програмного продукту
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAttr
ibute), false);
                // Якщо там не довільне значення атрибутів копірайту, то
повертається пустий рядок
                if (attributes.Length == 0)
                    return "";
                // Якщо там значення атрибутів копірайту, то повертається його
величина
                return ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
            }
        }

        public string AssemblyCompany
        {
            get
            {
                // Отримуємо усі атрибути місця де був реалізований дипломний
проект
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttrib
ute), false);
                // Якщо там не довільне значення атрибутів місця де був
реалізований дипломний проект, то повертається пустий рядок
                if (attributes.Length == 0)
                    return "";
                // Якщо там значення атрибутів місця де був реалізований
дипломний проект, то повертається його величина
                return ((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion
```

```
        private void textBoxDescription_TextChanged(object sender, EventArgs e)
        {

        }
    }
}
```