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

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

**ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА**
**за другим (магістерським) рівнем вищої освіти**
на тему
# “Дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-21М-1,4
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Ковальчук В.М.
«____» _____ 2022 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Якименко Н.М.
«____» _____ 2022 р.
Рецензент _____
_____



м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

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

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

*Ковальчуку Володимиру Миколайовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК*

2. Керівник роботи *Якименко Наталія Миколаївна, канд. фіз.-мат. наук, доцент*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 18-13 від 17.08.2022 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК*

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

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

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

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

6. Консультанти розділів роботи

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

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

## КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 6 » вересня 2022 р.

Підпис керівника
_____ Якименко Н.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2022 р.

Підпис здобувача
_____ Ковальчук В.М.
(прізвище та ініціали)

# АНОТАЦІЯ

**Ковальчук В.М. Дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2022.**

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи хмарних сервісів з використанням ЦСК.

Метою розробки є дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК.

Об'єктом дослідження є процес хмарних сервісів з використанням ЦСК.

Предметом дослідження є методи хмарних сервісів з використанням ЦСК.

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

Результат роботи – програмна реалізація системи хмарних сервісів з використанням ЦСК.

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

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

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

Програму розроблено в середовищі Visual C#.

**Ключові слова**: комп'ютерні науки, хмарні сервіси, центр сертифікації ключів

## ABSTRACT

**Kovalchuk V.M. Research and software implementation of the system of cloud services using KCC. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.**

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the system of cloud services using KCC.

The purpose of the development is the research and software implementation of the system of cloud services using KCC.

The object of the study is the process of cloud services using KCC.

The subject of research is methods of cloud services using KCC.

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

The result of the work is the software implementation of the cloud services system using KCC.

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

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

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

The program was developed in the Visual C# environment.



**Keywords:** computer science, cloud services, key certification center

**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 |
| ЦСК | – | центр сертифікацій та сертифікації та розподілу ключів |
| ЦР | – | центр реєстрації |
| ЦУМ | – | центр управління мережею |
| SOA | – | Service-Oriented Architecture |
| SOAP | – | Simple Object Access Protocol |

# ВСТУП

**Актуальність теми**. Магістерська робота присвячена питанням забезпечення безпеки інформації в сервіс-орієнтованих хмарних архітектурах, за рахунок розробки центру сертифікації та розподілу ключів (ЦСК). При згадуванні абревіатури SOA (Service-Oriented Architecture) більшість IT-фахівців першою справою згадують Web-сервіси й протокол HTTP, які є складовими хмарних сервісів, хоча цей термін позначає набагато більше широке поняття.

Хмарна архітектура SOA зовсім незалежить від мов програмування, платформ або протокольних специфікацій, за допомогою яких сервіси розробляються, а також від того, де й за допомогою чого вони розгорнуті.

Практично хмарна архітектура SOA вимагає наявності не тільки сервісів, але й засобів, за допомогою яких ці сервіси можуть бути виявлені й підключені незалежно від нижчележачої інфраструктури. SOA – це не продукт або специфікація. Хмарна архітектура ретельно вибудовується – складається з множини компонентів, таких, як сервери додатків, що зв'язують ПЗ, репозиторій і навіть спеціалізовані пакети централізованого управління SOA.

Строго говорячи, SOA не можна відносити ні до нової реалізації CORBA, ні до оновленої хмарної архітектури RMI (Remote Method Invocation). Ключовий компонент SOA – сервіс. Сервіси тут є бізнес-функціями, призначеними для забезпечення погодженої роботи великих, що складаються з множини частин додатків.

По суті, це будівельні блоки для відбиття бізнес-логіки в розроблювальних додатках. А кінцевим місцем, де сервіси "живуть", є сервер додатків, будь то WebLogic від BEA Systems, WebSphere від IBM, Application Server від Oracle або Java AS від Sun Microsystems.

Функції, або операції, в SOAP (Simple Object Access Protocol) повинні бути інтуїтивно зрозумілими й відповідати своїм назвам – наприклад,

submitPurchaseOrder (“підтвердити замовлення на покупку”) або validateCustomerAccounts (“перевірити особовий рахунок замовника”).

На відміну від звичайних додатків сервіс в хмарній архітектурі SOA призначається для використання всім реалізованим бізнес-функціям. У той час як звичайні корпоративні додатки містять у собі схожі фрагменти бізнес-логіки або навіть дублюють окремі об'єкти – наприклад, об'єкт клієнтського замовлення, – в хмарній архітектурі SOA вам потрібно запустити лише єдиний екземпляр такої бізнес-функції.

Таким чином, можливо повторно використовувати функціональність у середовищі із множинними додатками й швидко коректувати бізнес-логіку, для того щоб мати можливість пристосовуватися до мінливих умов ринку. У цьому й складається головна перевага SOA.

Зі зміною єдиного екземпляра бізнес-функції в SOA автоматично вносяться корективи й в усі додатки, що опираються на цю функцію. Так що, наприклад, будь-які зміни в правилах ціноутворення або політики знижок застосовуються у всіх додатках.

Аналогічно будь-які зміни в підтримуючій інфраструктурі залишаються прозорими для всіх додатків, що використовують сервіси. Наприклад, якщо ви переходите з однієї версії бази даних на іншу, то будуть модифіковані лише пов'язані з нею сервіси, оскільки додатки в хмарній архітектурі SOA працюють із усіма інфраструктурними додатками тільки за допомогою сервісів. У недавнім минулому зміни в сполучному ПЗ або в базі даних змушували переробляти всю систему, включаючи клієнтські настільні додатки.

**Мета й завдання дослідження.** Метою роботи є дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК.

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

– Огляд існуючих систем хмарних сервісів з використанням ЦСК.

– Дослідження системи хмарних сервісів з використанням ЦСК.

– Програмна реалізація системи хмарних сервісів з використанням ЦСК.

*Об'єктом дослідження* є процес хмарних сервісів з використанням ЦСК.

*Предметом дослідження* є методи хмарних сервісів з використанням ЦСК.

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

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

– Удосконалено метод хмарних сервісів з використанням ЦСК.

– Розроблено вітчизняний продукт хмарних сервісів з використанням ЦСК, який має більш широкі можливості, на відміну від існуючих аналогів.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі хмарних сервісів з використанням ЦСК.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Одне з важливих переваг хмарної архітектури SOA – її надійність, особливо якщо мова заходить про її реалізацію на базі Web-сервісів і специфікації SOAP, що найбільше часто використовують протокол HTTP.

Справа в тому, що HTTP не планувався для цілей гарантованої доставки, тому сам по собі він не забезпечує надійного виконання критично важливих транзакцій.

Для рішення цієї проблеми були створені два щодо нових конкуруючого стандарту – WS-RM (Web Services Reliable Messaging) і WS-R (Web Services Reliability), але сучасні реалізації SOA для забезпечення гарантованої доставки використовують більше традиційні методи, – наприклад, ESB (Enterprise Service Bus).

Сполучне ПЗ ESB може служити опорою вашої хмарної архітектури SOA, забезпечуючи гарантовану доставку повідомлень, обробку виняткових ситуацій і використання всіх можливостей моделі “ публікація-підписка”. Воно працює так само, як і традиційне сполучне ПЗ обміну повідомленнями, таке, як MQ Series фірми IBM, EMS фірми Tibco або JMS (Java Message Service), що виконує маршрутизацію повідомлень із їхнім проміжним зберіганням.

Сервіси на сервері додатків “контактують” із шиною ESB, запускаючи асинхронні сервісні механізми й гарантовану доставку повідомлень. ПЗ ESB також може служити як засіб інтеграції, що дозволяє множини додатків здійснювати доступ до однієї й тієї ж транзакції в межах системи.

Іншим важливим компонентом добре спланованої хмарної архітектури SOA є централізований репозиторій, що містить довідник всіх сервісів, доступних на підприємстві.

Тут підійде будь-яка їхня класифікація, що щонайкраще відповідає організації. Вона повинна забезпечувати групування сервісів по бізнес-функціям, підрозділам, джерелам даних або навіть за версією вихідного коду. Використовуйте таку класифікацію, що дозволить вашим розроблювачам швидко знаходити сервіси, у яких вони бідують для створення нових додатків.

Зазначену функціональність в SOA забезпечує реєстр, заснований на стандарті UDDI (Universal Description, Discovery and Integration), який можна розглядати як довідник "жовті сторінки" Web-сервісів. У ньому перераховані компанії й пов'язані з ними сервіси (всі використовувані тут стандарти – UDDI, WSDL і SOAP – перебувають у віданні організації OASIS). Дані механізми дозволяють організувати доступ ваших партнерів по бізнесі до реєстру, тим самим забезпечуючи можливість швидкої інтеграції ваших додатків.

**Адміністрування й безпека**

Одним з важливих питань побудови хмарної архітектури SOA є її адміністрування й забезпечення безпеки.

При наявності сотень сервісів, готових до застосування, у користувача виникає потреба в централізованій системі адміністрування, за допомогою якої можна було б здійснювати їхній моніторинг і захист.

Існує безліч систем адміністрування SOA (на базі агентів і без них). Продукти SOAPStation компанії Actional, Management Foundation, Service-Level Manager і Exception Manager фірми AmberPoint, Gateway від Reactivity і Service Manager компанії SOA Software є найбільш популярними засобами, призначеними для адміністрування сервісів у рамках усього підприємства. Компанії Computer Associates і Hewlett-Packard зі своїми продуктами Unicenter і OpenView відповідно також пропонують програмні засоби для оперативного адміністрування й моніторингу SOA.

Адміністрування хмарної архітектури SOA означає як моніторинг працездатності окремих сервісів, так і реєстрацію й перевірку необхідних

функцій відповідно до різних законодавчих актів, що регламентують правила й строки зберігання бухгалтерської звітності й інших ділових документів.

Програмні продукти по адмініструванню SOA також полегшують розподіл сервісів, автоматично відслідковуючи ненадійні з'єднання й проблеми взаємодії корпоративних мереж за допомогою засобів захвата повідомлень, по своїх можливостях набагато переважаючі традиційні аналізатори мережевих протоколів.

Залежно від потреб організації ви можете забезпечити безпеку хмарної архітектури SOA за допомогою звичайних продуктів управління на базі протоколу SOAP або інших спеціальних продуктів.

Програмні продукти, відповідальні за безпеку, такі, як XS40 компанії DataPower, Sentry і Xwall від ForumSystems, Gateway від Reactivity і XML Guardian від Sarvega, являють собою єдину точку доступу, через яку повинні проходити всі сервісні запити.

Безпеку цілком припустимо забезпечувати й на рівні сервера додатків, на якому розміщаються сервіси, але це є нездійсненним завданням, якщо ваш партнер по бізнесу буде взаємодіяти з вашою архітектурою SOA.

Зовнішні Web-сервіси або шлюз XML є більше гнучкими засобами для надання доступу партнерові (або клієнтові) до вашої SOA, значно полегшуюче життя розроблювачам додатків.

Якщо взяти до уваги той факт, що хмарна архітектура SOA не має чіткого набору поєднуваних систем, те немає нічого страшного в тому, щоб розділити функції забезпечення безпеки й адміністрування сервісів.

Система призначена для реалізації центру сертифікації та розподілу ключів у сервіс-орієнтованих хмарних архітектурах. Це необхідно для забезпечення безпеки інформації, яка передається по системі, побудованій з використанням концепції сервіс-орієнтованої хмарної архітектури.

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

– Сформувати оптимальну робочу групу.

– Створити деталізований план проекту.

– Вести таблицю рішень по безпеці для процесу впровадження SOA.

– 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 SOA-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 SOA-додатку.

Розглянемо деякі із цих кроків більш детально.

Необхідно вести офіційну документацію високорівневої динаміки системи безпеки SOA-додатку за допомогою таблиці рішень по безпеці для процесу впровадження SOA. У цій таблиці за допомогою зацікавлених бізнес-учасників всі додатки діляться на три категорії, а саме:

– Додатки, які будуть входити в архітектуру SOA.

– Додатки, яким необхідно взаємодіяти з SOA-додатками.

– Додатки, які не будуть входити в SOA або взаємодіяти з SOA-додатками.

Кожної окремої категорії для наочності добре привласнити визначений колір. Об'єкти в першій категорії позначені червоним, у другій – жовтогарячим, а в третій категорії – жовтим кольором.

Потім робоча група по безпеці ділить додатки за рівнем вимог до безпеки на три категорії – висока, середня й низька безпека. Наприклад, додатки або сервіси, що мають справу з інформацією з ринку й відомостями, які не піддаються розголошенню, потрапили б у більше високу категорію безпеки, тоді як додатки або сервіси, які звертаються тільки до даних, що перебувають у відкритому доступі, можуть бути поміщені в саму нижчу категорію.

З огляду на величезний обсяг інформації, який необхідно зібрати, кількість архітектурних артефактів, які необхідно написати, і конкретні сервіси безпеки SOA-додатку, які необхідно спроектувати, робочій групі по безпеці SOA-додатку варто дотримуватися стандартної схеми процедури Життєвий цикл розробки програмного забезпечення (Software Development Life Cycle, SDLC):

– Ідентифікація вимог до безпеки й обмежень.

– Виявлення й збір вимог до безпеки системи.

– Створення безпечного архітектурного проекту.

– Документація деталізованого безпечного проекту SOA.

– Реалізація SOA (включаючи керування SOA).

– Тестування.

– Розміщення.

– Обслуговування.

На перший погляд ці кроки можуть здатися очевидними, але робітники групи по безпеці рідко дотримують схеми SDLC.

Перш ніж робоча група приступиться до розробки рішень, вона повинна зібрати вимоги до систем. Існують як явні, так і неявні вимоги до реалізацій забезпечення безпеки. Що стосується явних вимог, гарною стартовою точкою буде збір вимог від кожної із зацікавлених осіб. Відносно неявних вимог до безпеки корисно використовувати інфраструктури забезпечення безпеки, наприклад, тріаду конфіденційність, цілісність і відслідкуємість (confidentiality, integrity, and accountability, CIA); з обліком якої складається список конкретних вимог до всіх систем безпеки. Тріада CIA являє собою широко використовувану

модель гарантування інформації (information assurance, IA), що визначає конфіденційність, цілісність і доступність як основні характеристики безпеки всіх інформаційних систем.

Члени робочої групи повинні продумати, як дана SOA-реалізація буде забезпечувати конфіденційність системи (приватність даних), і спроектувати точні схеми процесу із вказівкою докладного опису того, яким образом до повідомлень у процесі передачі будуть звертатися тільки авторизовані законні одержувачі, індивідуальні користувачі, процеси або пристрої. Розголошення повідомлень неавторизованим сутностям, наприклад, користувачам-зловмисникам, що здійснюють несанкціоноване перехоплення мережевих пакетів, є порушенням конфіденційності, і SOA-реалізації повинні визначати, де й коли в границях всієї системи буде використовуватися криптографія.

Аналогічно, моделі безпеки SOA-додатку вимагають дотримання цілісності, або гарантії того, що повідомлення не було змінено в процесі передачі. Система безпеки SOA-додатку відповідає за забезпечення того, що інформація не була змінена в процесі передачі від джерела до одержувача (цілісність даних) і за гарантування того, що відправник цієї інформації є тим, за кого себе видає (цілісність джерела); і одержувач також є тим, за кого себе видає (цілісність одержувача). Цілісність даних може бути скомпрометована, якщо інформація навмисне або випадково ушкоджена або змінена до того, як неї прочитав законний одержувач. Цілісність джерела вважається скомпрометованою в тому випадку, якщо агент фальсифікує особисті дані й відправляє некоректну інформацію одержувачеві. Для забезпечення цілісності даних використовуються такі механізми, як алгоритм гешування й цифрові підписи.

Крім того, одним з вимог безпеки SOA-додатку є своєчасний і надійний доступ до сервісів даних для авторизованих користувачів (доступність). SOA-реалізації повинні гарантувати, що ці ресурси або інформація будуть доступні, коли в них виникне необхідність; це означає, що доступ до ресурсів може бути здійснений на швидкості, достатньої для того, щоб віддалена система могла

виконати своє завдання запропонованим образом. Безумовно, можливі випадки, коли заходи щодо захисту конфіденційності й цілісності прийняті, але атакуючий проте може зробити ресурси менш доступними, чим потрібно, або взагалі недоступними. Зокрема, коли як брокер повідомлень використовуються компоненти SOA-системи, такі як ESB, то для забезпечення її доступності й надійності необхідно вказати в документації вимоги до безпеки SOA-додатку, що необхідно використовувати протоколи високої доступності, мережеві хмарної архітектури з резервуванням і апаратне забезпечення системи, що не має ні однієї одиночної точки відмови. Робоча група по безпеці SOA-додатку повинна забезпечити всеосяжне виявлення всіх зазначених областей і гарантувати, що відповідні контрольні приклади будуть задокументовані й зможуть проілюструвати визначені вимоги.

Після того як робоча група по безпеці SOA-додатку приділила якийсь час обговоренню всіх описаних вимог, члени групи повинні з'ясувати, чи не можна виконати їх за допомогою інструментів сторонніх виробників. Вони повинні написати програми сервісів безпеки SOA, які будуть задовольняти конкретним вимогам. Щоб не винаходити колесо, краще й корисніше ознайомитися із уже наявними моделями й довідатися, які розробки вже були зроблені до того, як дана робоча група перейшла до етапу високорівневого проектування. Нижче представлений список і опис звичайний використовуваних службових сервісів, які можуть придатися в SOA-реалізації. Більш докладно приводиться список:

– сервіс аудита загального призначення;

– сервіс авторизації для керування доступом;

– сервіс забезпечення конфіденційності;

– сервіс конверсії повноважень доступу;

– сервіс поновлення повноважень доступу;

– сервіс делегування;

– сервіс спостереження за периметром міжмережевого екрана;

– сервіс установлення особистості;

– сервіс зіставлення особистості;

– сервіс забезпечення інформаційної цілісності;

– міждомений сервіс безпеки;

– сервіс забезпечення неможливості відмови від авторства;

– сервіс маршрутизації й QoS (гарантованої якості обслуговування);

– сервіс настроювання політик безпеки;

– сервіс зміни політик;

– сервіс забезпечення приватності;

– сервіс для роботи із профілем (сервіс користувальницьких профілів);

– сервіс забезпечення якості встановлення особистості;

– сервіс захисту від атак відмови в обслуговуванні (denial of service, DoS);

– сервіс керування забезпеченням безпеки;

– сервіс узгодження протоколів;

– сервіс оцінки доступності сервісів безпеки;

– сервіс єдиного входу в систему;

– сервіс установлення довіри.

Можливо, вам знадобляться не всі ці сервіси. Співробітники робочої групи по безпеці SOA-додатку повинні зрівняти вимоги з кожним сервісом, потім створити модель безпеки SOA для всіх сервісів, необхідних для задоволення вимог.

Після того як будуть виконані описані дії й буде зібрано досить інформації, можна перейти перегляду стандартів WS-Security і з'ясувати, які з них застосовні до вашої конкретної реалізації безпеки SOA-додатку.

Як тільки моделі стануть деталізованими, необхідно вивчити докладні стандарти безпеки SOA-додатку, що входять в WS-Security, і зрозуміти, як вони співвідносяться один з одним і з вимогами до моделі безпеки SOA-додатку. Ці стандарти безпеки будуть використовуватися для формування безпечних повідомлень у всієї SOA-реалізації.

## 1.2 Область застосування

Областю застосування системи, яка розробляється є сервіс-орієнтована хмарна архітектура. Розглянемо більш детально цю технологію програмування.

**Сервіс-орієнтовна хмарна архитекту́ра** (SOA, service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні сервісів (служб) зі стандартизованими інтерфейсами.

В основі SOA лежать принципи багаторазового використання функціональних елементів IT, ліквідації дублювання функціональності в ПЗ, уніфікації типових операційних процесів, забезпечення перекладу операційної моделі компанії на централізовані процеси й функціональну організацію на основі промислової платформи інтеграції.

Компоненти програми можуть бути розподілені по різних вузлах мережі, і пропонуються як незалежні, слабко зв'язані, замінні сервіси-додатки. Програмні комплекси, розроблені відповідно до SOA, часто реалізуються як набір веб-сервісів, інтегрованих за допомогою відомих стандартних протоколів (SOAP, WSDL, і т.п.)

Інтерфейс компонентів SOA-програми надає інкапсуляцію деталей реалізації конкретного компонента (ОС, платформи, мови програмування, вендора, і т.п.) від інших компонентів. Таким чином, SOA надає гнучкий і елегантний спосіб комбінування й багаторазового використання компонентів для побудови складних розподілених програмних комплексів.

SOA добре зарекомендувала себе для побудови великих корпоративних програмних додатків. Цілий ряд розроблювачів і інтеграторів пропонують інструменти й рішення на основі SOA (наприклад, платформи IBM WebSphere, Oracle/BEA Aqualogic, Microsoft Windows Communication Foundation, SAP NetWeaver, ІОК Юпітер, TIBCO, Diasoft).

OASIS (Організація по поширенню відкритих стандартів структурованої інформації) визначає SOA у такий спосіб (OASIS Reference Model for Service

Oriented Architecture V 1.0): Сервісно-орієнтована хмарна архітектура – це парадигма організації й використання розподілених інформаційних ресурсів таких як: додатки й дані, що перебувають у сфері відповідальності різних власників, для досягнення бажаних результатів споживачем, яким може бути: кінцевий користувач або інший додаток.

Основна причина появи SOA – мрія індустрії програмування про заміну «кустарного» кодування програм «від і до» на «промислове» складання додатків з «стандартних комплектуючих», як в автомобільної, або інших «традиційних» галузях промисловості.

Для великих інформаційних систем, рівня підприємства, і вище:

– скорочення витрат при розробці додатків, за рахунок впорядкування процесу розробки;

– розширення повторного використання коду;

– незалежність від використовуваних платформ, інструментів, мов розробки;

– підвищення масштабованості створюваних систем;

– поліпшення керованості створюваних систем.

**Принципи SOA**

1. Хмарна архітектура, як така, не прив'язана до якоїсь визначеної технології.

2. Незалежність організації системи від використовуваної обчислювальної платформи (платформ).

3. Незалежність організації системи від застосовуваних мов програмування.

4. Використання сервісів, незалежних від конкретних додатків, з однаковими інтерфейсами доступу до них.

5. Організація сервісів як слабо-зв’язаних компонентів для побудови систем

**Інші SOA-концепції**

Хмарна архітектура не прив'язана до якоїсь визначеної технології. Вона може бути реалізована з використанням широкого спектра технологій, включаючи такі технології як REST, RPC, DCOM, CORBA або веб-сервіси. SOA може бути реалізована використовуючи один із цих протоколів і, наприклад, може використовувати, додатково, механізм файлової системи, для обміну даними.

Головне, що відрізняє SOA, це використання незалежних сервісів, із чітко визначеними інтерфейсами, які, для виконання своїх завдань, можуть бути викликані якимось стандартним способом, за умови, що сервіси заздалегідь нічого не знають про додаток, що їх викличе, а додаток не знає, яким образом сервіси виконують своє завдання.

SOA також може розглядатися як стиль хмарної архітектури інформаційних систем, що дозволяє створювати додатка, побудовані шляхом комбінації слабо-зв'язаних і взаємодіючих сервісів. Ці сервіси взаємодіють на основі якого-небудь строго визначеного платформенно-незалежного і язиково-незалежного інтерфейсу (наприклад, WSDL). Визначення інтерфейсу приховує мовно-залежну реалізацію сервісу.

Таким чином, системи, засновані на SOA, можуть бути незалежні від технологій розробки й платформ (таких як Java, .NET і т.д.). Приміром, сервіси, написані на C#, що працюють на платформах .Net і сервіси на Java, що працюють на платформах Java EE, можуть бути з однаковим успіхом викликані загальним складеним додатком. Додатки, що працюють на одних платформах, можуть викликати сервіси, що працюють на інших платформах, що полегшує повторне використання компонентів.

SOA може підтримувати інтеграцію й консолідацію операцій у складі складних систем, однак SOA не визначає й не надає методологій або фреймворків для документування сервісів.

Мови високого рівня, такі як BPEL, або специфікації, такі як WS-CDL і WS-Coordination, розширюють концепцію сервісу, надаючи метод оркестрації, для об'єднання дрібних сервісів у більше великі бізнес-сервіси, які, у свою чергу, можуть бути включені до складу технологічних процесів і бізнес-процесів, реалізованих у вигляді складених додатків або порталів.

Використання компонентної хмарної архітектури (SCA) для реалізації SOA – це область поточних досліджень.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Розглянемо існуючі програмні продукти, що реалізують центр сертифікації та розподілу ключів.

Проведений аналіз показав, що на сучасному етапі найбільшим попитом користуються наступні продукти:

– комплекс "Тропа-джет";

– ViPNet.

**Комплекс «Тропа-джет"**

Основними компонентами комплексу «Тропа-джет" є:

– Центр генерації й сертифікації ключів.

– Центр сертифікації та розподілу ключів.

– Шлюз кодування (Кодуючий модуль).

– Точка реєстрації мобільних клієнтів.

– Мобільний клієнт.

**Центр генерації й сертифікації ключів**

Центр генерації й сертифікації ключів служить для генерації пар "секретний ключ – відкритий ключ", а також є репозитарієм всіх відомих системі ключів.

Центр генерації ключів установлюється на комп'ютер, що не має мережевих з'єднань. Він призначений для виготовлення секретних і відкритих ключів, підготовки ключових дискет, зберігання еталонних копій ключів, а також підпису сертифікатів ключем адміністратора

**Центр сертифікації та розподілу ключів**

У завдання Центра сертифікації та розподілу ключів входить роздача ключів і управління контуром безпеки, а також виконання наступних функцій:

– Одержання зі змінного носія відкритих ключів Шлюзів.

– Видача будь-якому Шлюзу сертифікатів відкритих ключів будь-яких інших Шлюзів і інформації про відповідні сегменти мережі.

– Розсилання Шлюзам повідомлень про зміни структури закритої мережі.

– Зберігання інформації про структуру мережі.

Центр реалізований у вигляді програмного комплексу, що виконує функції зберігання й видачі відкритих ключів кодування по мережевому запиту від Шлюзів кодування. Центр сертифікації та розподілу ключів може бути встановлений або на окремому (виділеному) комп'ютері, або разом з одним зі Шлюзів кодування.

**Шлюз кодування**

Шлюз є основним модулем комплексу, що забезпечує маршрутизацію, фільтрацію й кодування пакетів. Кожний Шлюз призначений для захисту визначеної групи локальних мереж. На комп'ютері-шлюзі встановлюється ядерний модуль із функціями кодування й декодування й запускається програма автентифікації. Шлюз виконує наступні функції:

– Фільтрація трафіку (розподіл на кодуємий і некодуємий потоки).

– Кодування трафіку (кодуємий потік).

– Автентифікація з іншими Шлюзами.

– Виробіток і виконання процедури зміни сеансових ключів.

– Реєстрація подій у Центрі моніторингу.

– Забезпечення власного захисту.

**Точка реєстрації мобільних клієнтів і Мобільний клієнт**

Доступ до корпоративних даних, які захищаються, для мобільних абонентів, не підключених до локальних мереж, які захищаються, забезпечується

за допомогою таких засобів комплексу «Тропа-джет", як Точка реєстрації мобільних клієнтів і програмне забезпечення Мобільний клієнт.

Точка реєстрації мобільних клієнтів являє собою спеціальний шлюз кодування для підключення довільної кількості мобільних клієнтів. Мобільний клієнт являє собою програмний модуль, що працює під управлінням ОС Windows 10/11 і використовуючий апаратні ключі eToken для автентифікації абонента в VPN.

**Центр моніторингу**

Центр моніторингу являє собою мережеве автоматизоване робоче місце із установленим на ньому набором програм, що здійснюють збір і аналіз протоколів, що надходять від всіх модулів комплексу.

**Програма контролю цілісності**

Комплекс «Тропа-джет" містить у собі засобу формування й перевірки контрольних сум файлів. Ці засоби оформлені у вигляді Програми контролю цілісності. Ця Програма призначена для виявлення змін, додавань і видалень файлів і повідомлення системного адміністратора про ці події.

**Автоматизоване робоче місце адміністратора**

Настроювання й адміністрування компонентів комплексу «Тропа-джет" здійснюються централізовано з автоматизованого робочого місця (АРМ) адміністратора безпеки за допомогою графічного інтерфейсу або командного рядка. Віддалене управління здійснюється по захищеному каналі. АРМ забезпечує автентифікацію адміністраторів і розмежування доступу до функцій адміністрування.

**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» у наступному складі:

– **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 міжмережевого 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 мережевий екран або ПЗ ViPNet [Персональний мережевий екран] призначено для персонального мережевого захисту комп'ютерів і виключення доступу до ключової інформації.

– **ПЗ ViPNet [Клієнт] [Ділова пошта]** – Outlook-подібний додаток, що забезпечує 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.

**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 мережевих 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 сертифікації та розподілу ключів), так і по запиті користувача корпоративної мережі, сформованого на його мережевому 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 сертифікації та 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 мережевого 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 мережеві каталоги локальної мережі.

**ПЗ ViPNet [Координатор]**

Це багатофункціональний програмний модуль, що забезпечує наступну функціональність:

– міжмережеве 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 мережевого рівня TCP по портах 5000, 5001, 5002, що дозволяє шляхом настроювань для пропуску цього протоколу на міжмережевих екранах виключити можливі мережеві 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. Міжмережевий екран ViPNet [Координатор], що захищає ЗКЦ, сполучений із сервером транспортного модуля MFTP. Мережеві вузли мають можливість взаємодіяти тільки з координатором і не мають можливості прямої взаємодії з ЗКЦ і Центром управління, що дозволяє на міжмережевому екрані припинити будь-які мережеві атаки навіть із боку зареєстрованих користувачів.

6. Якщо на клієнтські комп'ютери можливі мережеві атаки, то для захисту криптографічних модулів і ключів ЕЦП на них доцільна установка модуля ViPNet [Клієнт] або його складової ViPNet [Персональний мережевий екран]. Модуль

ViPNet [Клієнт] додатково до персонального мережевого екрана створює зашифрований тунель до ViPNet [Координатора], блокуючи при цьому будь-який відкритий трафік, що повністю виключає будь-які мережеві атаки на комп'ютер користувача.

**Масштабованість рішення**

Центр управління й ЗКЦ ViPNet можуть забезпечити автоматичну захищену взаємодію більш ніж з 65 000 мережевими 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 застосунків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські застосунки Windows, веб-служби XML, розподілені компоненти, застосунки типу “ сервер-клієнт”, застосунки баз даних і багато яких інших. В Visual C# є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку застосунків 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` – крапку входу застосунка – інкапсулюється 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.

**Хмарна 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.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для керуванням рядками для розбивки XML, і закінчуючи елементами керування Windows Forms. У звичайному застосунку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

### 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи хмарних сервісів з використанням ЦСК.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

### Опис технології WS Security

Надаючи вільно зв'язані сервіси, сервіс-орієнтована хмарна архітектура дозволяє гнучко реагувати на постійно мінливі ділові процеси. При цьому необхідно приділити увагу не тільки функціональним аспектам, але й створенню гнучкої інфраструктури безпеки, оскільки зміни ділових процесів роблять на неї серйозний вплив. Приміром, залучення нових ділових партнерів або включення конфіденційних відомостей у важливі корпоративні процеси вимагає адекватного стандартизованого рішення для забезпечення безпеки.

Як основна технологія забезпечення безпеки повідомлень на базі SOAP (Simple Object Access Protocol) міцно закріпився стандарт безпеки служб Web (Web Services Security, WS Security), ратифікований OASIS, організацією по розвитку стандартів структурованої інформації. WS Security складається із цілого пакета специфікацій і множини механізмів, які комбінуються відповідно до необхідного сценарію застосування.

До честі творців стандартів у рамках SOA вони приділили підвищену увагу безпеки при розробці цих стандартів. Механізми безпеки органічно вбудовуються в концепцію Web-сервісів і дозволяють не тільки уникнути основних проблем, але й істотно підвищити ефективність як механізмів захисту, так і засобів керування політикою безпеки.

### Стандарти

Основний пул стандартів безпеки Web-сервісів розробляється в рамках консорціуму OASIS. Структуру специфікацій безпеки SOA можна зобразити у вигляді наступної ієрархічної конструкції (рисунок 3.1).

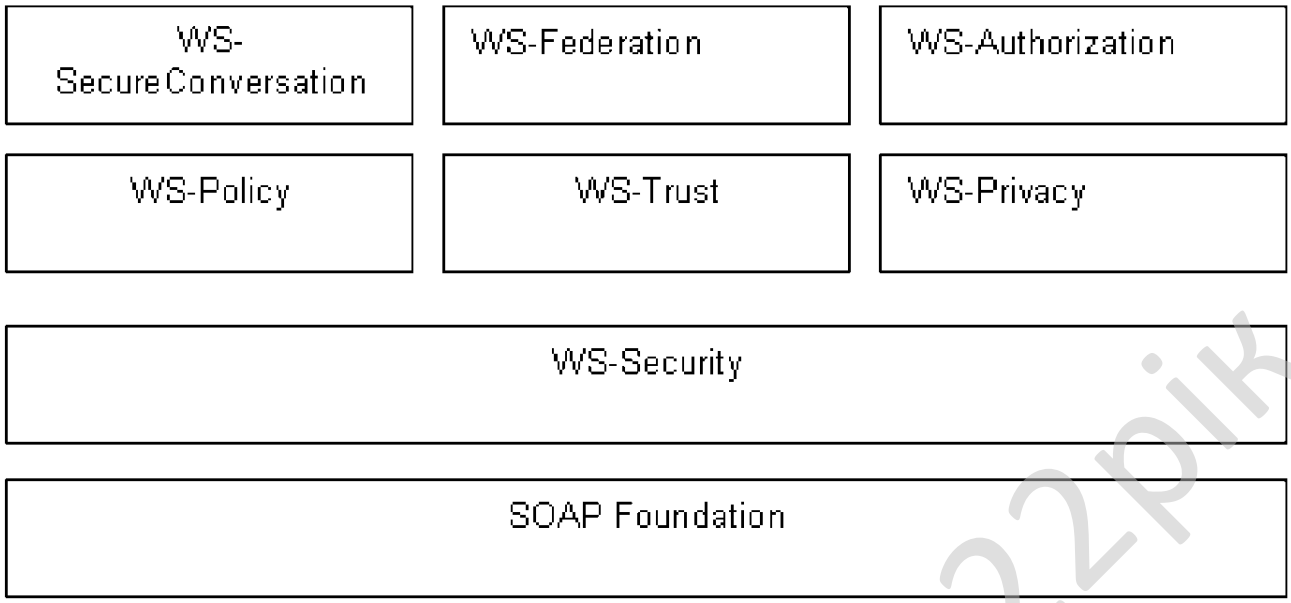


Рисунок 3.1 – Структура пула стандартів безпеки Web-сервісів

Розглянемо ці стандарти:

– Базові стандарти (SOAP Foundation) містять у собі специфікації XML Signature і XML Encryption, які визначають відповідно формати ЕЦП і шифрування SOAP-транзакцій. Дані специфікації ніяк не обмежують список алгоритмів шифрування й ЕЦП, що робить вбудовування українського ДСТУ в SOA-архітектуру неважким завданням. Також до базових понять можна віднести інформацію в складі SOAP-заголовка (security-token, маркер безпеки), використовувану для автентифікації й авторизації запиту. Наприклад, security-token може містити в собі сертифікат X.509 і/або ім'я/пароль. Одним з видів security-token є SAML (Security Assertion Markup Language), що включає в себе інформацію про статус автентифікації, авторизації й атрибутах учасників транзакції. Це дозволяє забезпечити побудову відносин довіри (trust) в SOA-хмарній архітектурі й виключити необхідність автентифікації/авторизації для кожного запиту.

– WS-Security визначає базові механізми й формати використання security-token у складі SOAP-запитів. Основною метою WS-Security є абстрагування

реалізації політик безпеки Web-сервісів від конкретних методів (наприклад, протоколів автентифікації й авторизації). За допомогою уточнюючих специфікацій, описаних нижче, WS-Security дозволяє досягти сумісності методів реалізації політик безпеки, описаних з використанням даних стандартів.

– WS-Policy визначає шаблони й правила опису політики безпеки для Web-сервісів.

– WS-Trust описує правила організації довірених відносин між учасниками Web-взаємодії.

– WS-Privacy визначає формати політики конфіденційності при обміні SOAP-Повідомленнями.

– WS-SecureConversation регламентує правила безпечного обміну повідомленнями в SOA-хмарній архітектурі.

– WS-Federation є специфікацією, що визначає встановлення довірених відносин між різними доменами безпеки.

– WS-Authorization описує формати опису правил розмежування доступу до Web-сервісів.

Отже, стає ясно, що безпека в SOA-хмарній архітектурі описується досить великим набором специфікацій. Втішно, однак, що даний набір є невід'ємною частиною пула стандартів SOA і розробляється одночасно з ним. Це дає підстави думати, що додатки в складі Web-сервісної хмарної архітектури можуть створюватися безпечними вже на стадії проектування.

**Хмарна архітектура**

Розглянемо тепер типову архітектуру безпеки Web-сервісів, що застосовується в більшості рішень корпоративного рівня.

Ключові завдання, покладені на таку архітектуру:

– Керування доступом до Web-сервісів і однократна автентифікація (Single Sign-on, SSO). Призначено для забезпечення однократної автентифікації, авторизації й аудита Web-сервісів.

– Централізоване керування політикою безпеки. Дозволяє мінімізувати необхідність дублювання зусиль для застосування політики безпеки для кожного Web-сервісу за допомогою використання централізованої інфраструктури безпеки, не вимагаючи при цьому переробки самих Web-сервісів.

– Уніфікація процесу моніторингу. Дозволяє проводити аудит роботи Web-сервісів, що показує, які користувачі (додатка) здійснювали доступ до Web-сервісів, які дії вони виконували і які дані при цьому передавали.

– Маршрутизація запитів до Web-служб. Дозволяє, аналізуючи вміст запиту, проводити його перетворення й перенапрямок до того або інший Web-сервісу.

**Схема керування захистом в SOA-хмарній архітектурі**

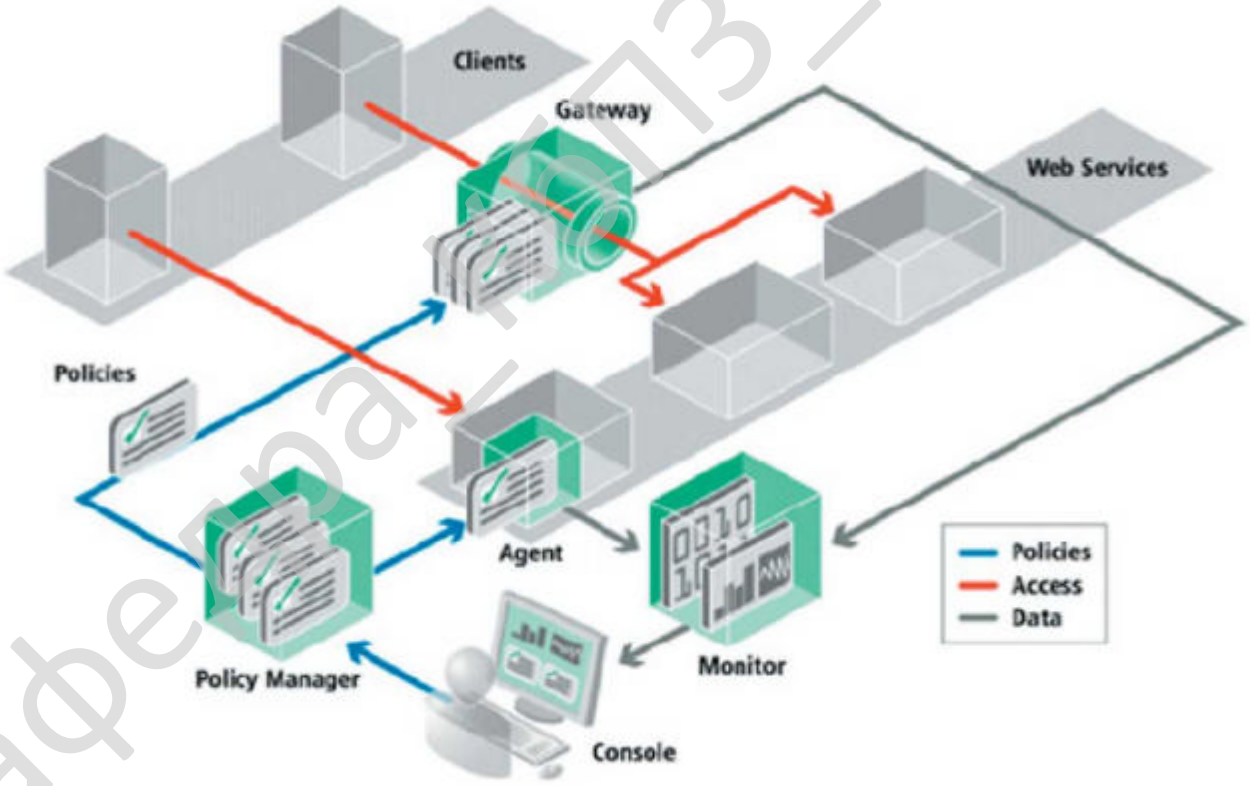


Рисунок 3.2 – Схема керування захистом в SOA-хмарній архітектурі

До складу такої схеми входять наступні компоненти:

– менеджер політик (Policy Manager);

– компоненти застосування політики: агенти (Agents) і шлюзи (Gateways);

– панель моніторингу (Monitor).

Менеджер політик – це графічний інструмент для визначення нових політик безпеки й експлуатації, зберігання політик, а також для керування поширенням і відновленням політик на агентах і шлюзах.

Компоненти застосування політик діляться на шлюзи (Policy Gateways) і агенти (Policy Agents). Шлюзи політик установлюються перед групою додатків або сервісів, перехоплюючи запити до цих додатків з метою застосування політик, підвищуючи безпеку вже встановлених додатків і додаючи в них нові правила. Агенти політик забезпечують додатковий диференційований рівень безпеки й розміщаються на серверах додатків, що забезпечують виконання додатка або сервісу. Таким чином, забезпечується можливість автентифікації й авторизації запитів до Web-сервісів по наявним на підприємстві репозитаріям користувачів (наприклад, LDAP-каталог).

На панелі моніторингу адміністратор може задати рівні якості обслуговування для кожного додатка, визначити правила видачі попереджень і повідомлень, якщо додаток перевищить заданий рівень якості обслуговування.

Таким чином, архітектуру безпеки SOA можна побудувати без переробки безпосередньо Web-сервісів. Це одне з основних достоїнств наявності стандартів безпеки, що є частиною загального пула стандартів SOA.

**Базові концепції**

OASIS прийняла стандарт WS Security у березні 2004 р. як доповнення до протоколу SOAP. До теперішнього часу він визнаний цілком зрілим і придатним до застосування. WS Security не визначає ніяких нових технологій, а опирається на вже існуючі стандарти, приміром, XML Encryption, XML Signature, сертифікати X.509 або різні криптографічні алгоритми. Базова концепція ґрунтується на механізмах повідомлень, тому замість захисту, орієнтованої на

транспорт, можливе забезпечення безпеки від краю до краю (End-to-End Security), приміром, за допомогою протоколу SSL. Такий підхід необхідний, щоб уникнути виникнення наскрізних комунікаційних структур у межах SOA, а також забезпечити передачу асинхронних повідомлень або використання проміжних станцій (приміром, сервісної шини підприємства – Enterprise Service Bus, ESB).

Основне завдання WS Security – забезпечення цілісності, конфіденційності й автентичності повідомлення і його відправника при одночасному збереженні відкритості для розширень. Основними елементами стандарту є наступні базові механізми (рисунок 3.3): токени безпеки, шифрування, підписи й оцінки про час.

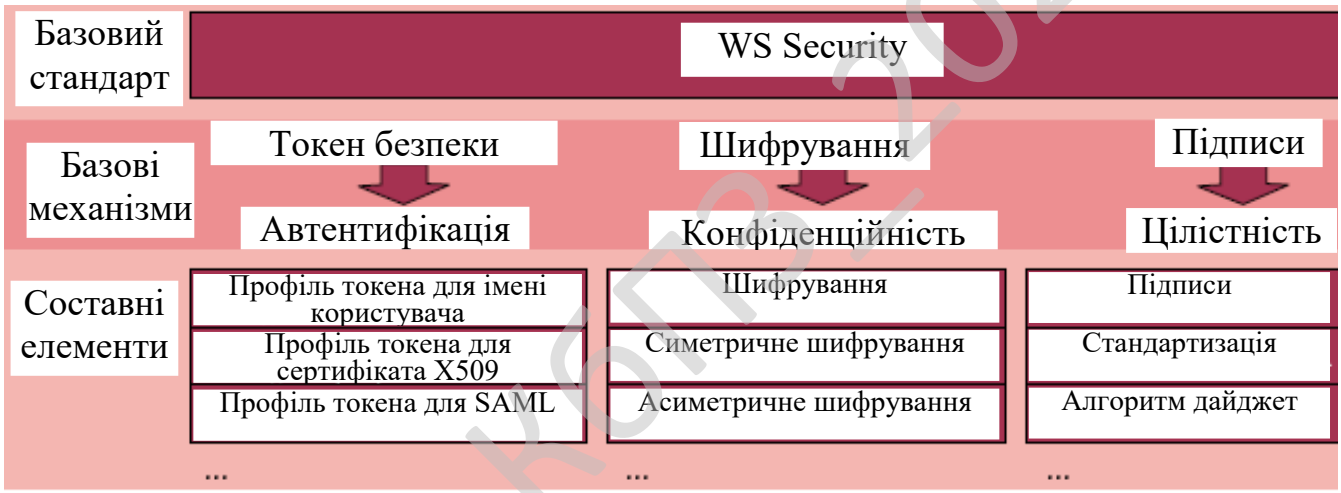


Рисунок 3.3 – Базові механізми WS Security

**Токени безпеки (Security Token).** Автентифікація відправника – базова передумова для забезпечення контролю доступу (Access Control) з боку сервісу, а крім того, вона необхідна для організації обліку й контролю. Підтвердження ідентифікації (Credentials), без яких неможлива автентифікація, передаються усередині повідомлення у вигляді токенів. Сама автентифікація не входить до складу WS Security – це самостійний процес провайдеру послуг. Для різних форматів токенів OASIS пропонує окремі специфікації у вигляді профілів WS Security. Так, «Профіль токена з ім'ям користувача» (Username Token Profile) регулює алгоритм широко розповсюдженого методу автентифікації

користувача за допомогою ідентифікаційного номера (User ID) і відповідного пароля.

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.

**Оцінка про час (Timestamp).** Ідея послуг у рамках SOA має на увазі, що сервіси повинні робити визначену дію й у такий спосіб підтримувати взаємодію без обліку стану (Stateless). Однак даний принцип комунікації без установлення сеансу відкриває простір для атак скидання (Replay), що коли атакує повторно відправляє або повідомлення цілком, або окремі їхні частини. Щоб перешкодити таким атакам, необхідно гарантувати унікальність повідомлень, для чого кожне з них одержує свій ідентифікаційний номер (Message ID), що сервіс перевіряє на предмет його унікальності. Тому ідентифікаційні номери вже отриманих повідомлень необхідно зберігати. Термін дії, а виходить, і час зберігання окремих ідентифікаційних номерів повідомлень на стороні сервісу обмежується оцінкою, що втримується в повідомленні, про час.

Крім використання традиційної структури оцінок про час у заголовку безпеки (Security Header), токен з ім'ям користувача пропонує власне керування оцінками про час щоб уникнути несанкціонованого повторного використання даних для автентифікації. Ідентифікаційний номер повідомлення повинен відповідати специфікації WS Addressing. А токени з ім'ям користувача одержують випадкове криптографічне значення (Nonce).

У рамках SOA кожний зі згаданих чотирьох базових механізмів охоплює лише один аспект забезпечення безпеки. Сформувати цілісне рішення можна лише за умови взаємодії всіх компонентів. Цілком традиційна комбінація механізмів безпеки на основі повідомлень (WS Security), орієнтованих на транспорт (SSL). Сценарій, представлений на рисунку 3.4, докладно роз'ясняє необхідність використання комбінації різних механізмів.

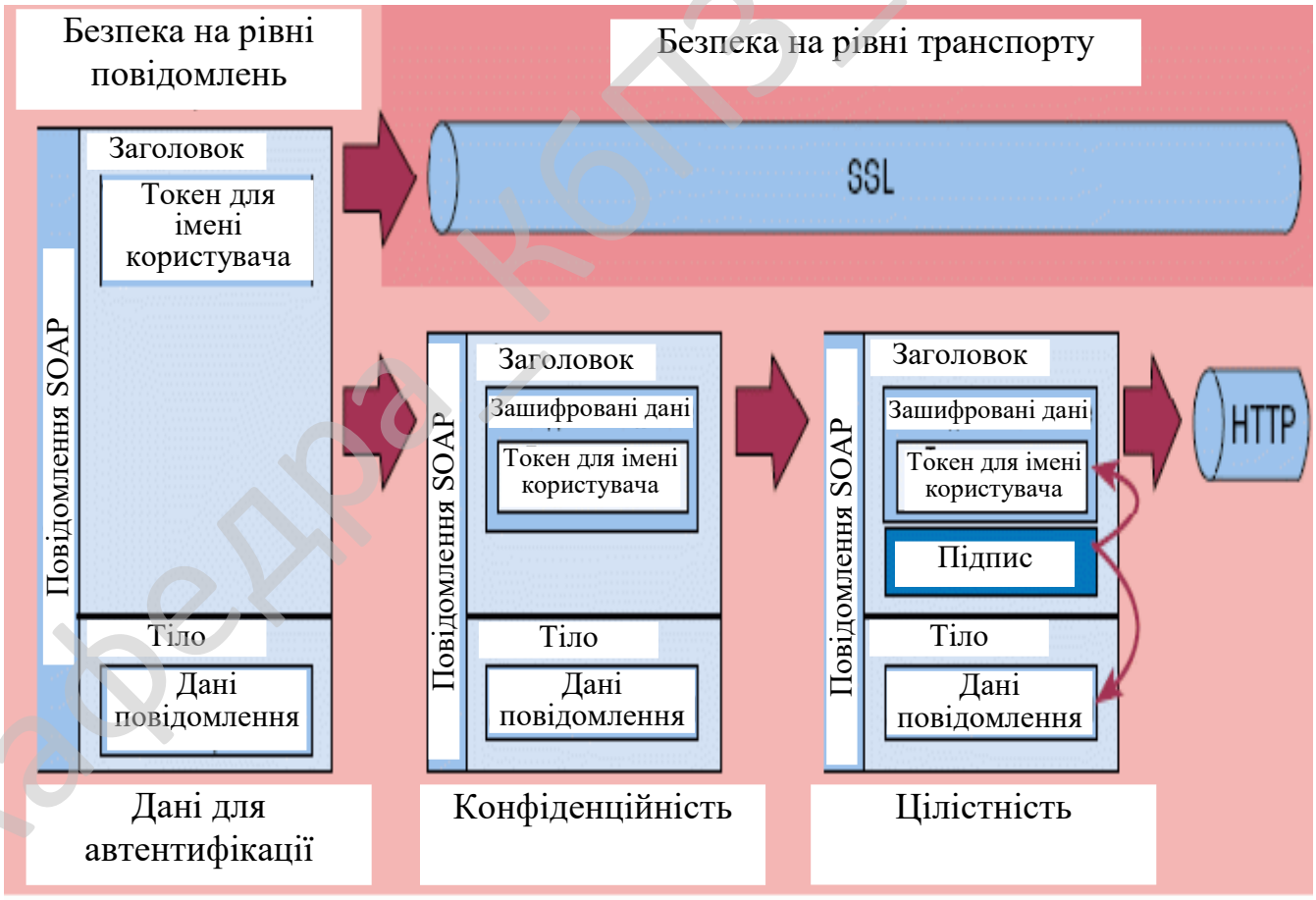


Рисунок 3.4 – Взаємодія токенів, шифрування й підписів WS Security

**Автентифікація.** Будь-який контроль доступу на стороні сервісу припускає автентифікацію клієнта. Сервісній стороні необхідно мати відомості про підтвердження ідентифікації. У випадку методу з користувальницьким ідентифікаційним номером і паролем WS Security надає механізм токенів з ім'ям користувача, де пароль є конфіденційною інформацією, тому необхідно запобігти його зчитуванню в процесі транспорту. Шифрування необхідно, навіть якщо механізм, визначений у специфікації токенів з ім'ям користувача, припускає передачу пароля тільки у вигляді контрольної суми. При використанні контрольної суми, що читається, виникає погроза атаки методом підбора пароля (Brute Force) шляхом перевірки всіх можливих комбінацій, оскільки паролі обмежені по довжині й набору символів.

Крім того, у випадку застосування контрольної суми пароля сервісній стороні знадобиться пароль відкритим текстом. Тому даний підхід у багатьох випадках неприйнятне або адміністрування паролів зажадає додаткових заходів безпеки.

**Конфіденційність.** Запобігти розкраданню пароля під час пересилання повідомлення покликане шифрування токену з ім'ям користувача. У деяких випадках досить застосувати широко розповсюджений протокол SSL. Однак необхідно врахувати, що внаслідок принципу з'єднання двох точок, властивого SSL, використання проміжних вузлів, приміром, сервісної шини підприємства (Enterprise Service Bus, ESB), неможливо, і захист даних після їхньої передачі не забезпечується.

У той же час механізм шифрування WS Security надає метод на основі повідомлень: вихідні дані шифруються й заміняються за допомогою алгоритму шифрування XML. Додатково в повідомлення вкладається метаінформація, приміром, про використані алгоритми або ключі, і тепер воно може передаватися навіть за допомогою незахищених протоколів (приміром, HTTP), а конфіденційність даних не піддається погрозі.

**Цілісність.** У використаному як приклад сценарії відсутня зв'язок між токеном з ім'ям користувача, що перебуває в заголовку SOAP, і даними в тілі SOAP. У результаті виникає погроза підміни ключових елементів повідомлення. Зокрема, зашифрована інформація про користувача, зазначена в заголовку, може бути постачена підробленим запитом сервісу. Однак ця проблема легко вирішується за допомогою підписів. Механізми підписів, використовувані в WS Security, «скріплюють» трохи пространственно розділених блоків даних, з яких складається повідомлення, що дозволяє перевірити цілісність усього повідомлення або окремих його частин. У контексті безпеки на базі повідомлень підпису виконують роль елементарних конструктивних компонентів і зачіпають не тільки тему цифрових підписів.

**Унікальність повідомлення.** Для того щоб запобігти повторному відправленню повідомлення (атака Replay), на сервісній стороні необхідно перевірити унікальність повідомлення. Для цього до повідомлення, представленому в стандартизованому виді, додається ідентифікаційний номер. Стандарт WS Addressing, визначений в W3C, передбачає, серед іншого, завдання ідентифікаційного номера повідомлення, що допомагає встановити його унікальність. Визначена в рамках WS Security структура вказує час створення повідомлення й закінчення строку його дії.

На закінчення потрібно відзначити, що ідентифікаційні номери повідомлень, як і оцінки про час, повинні бути прив'язані до існуючих блоків даних (інформація про користувачів і дані повідомлень). Для цього потрібно розширити діапазон охвату підпису, що дозволяє включити нові елементи при контролі цілісності.

**Підписи і їхні завдання**

Завдяки своїй інфраструктурі на основі повідомлень, сервіси Web підтримують можливість включення будь-яких проміжних інстанцій (Intermediaries) між кінцевими точками. С допомогою такої хмарної архітектури можна розширити функціональність сервісів Web. Крім того, ця хмарна

архітектура стає основою для організації поділу відповідальності за реалізацію властивостей сервісу, особливо вимог, не пов'язаних з функціональністю (якість сервісу – Quality of Services, QoS). При виклику сервісу повідомлення із запитом і відповіддю повинні пройти через проміжні інстанції, причому кожна витягає з повідомлення дані, необхідні для виконання її завдань, і, якщо знадобиться, постачає його додатковою інформацією. Відповідно, необхідно, щоб визначені частини повідомлень були придатні для читання й зміни проміжними інстанціями. Так, за допомогою даних WS Addressing з повідомлення можна управляти функціями маршрутизації в межах ESB.

Для забезпечення конфіденційності й цілісності повідомлення, з одного боку, і читаності й розширюваності, з іншої, до механізмів безпеки пред'являються підвищені вимоги. Приміром, шифрування всього повідомлення за допомогою протоколу SSL перешкоджало б гнучкому використанню проміжних інстанцій. Крім того, стандарт SOAP вимагає, щоб конверт, заголовок і тіло повідомлення представлялися в незашифрованому виді.

Підписи виконують кілька важливих завдань для забезпечення всебічної безпеки в рамках такої вільно зв'язаної хмарної архітектури. Крім загальновідомої ролі цифрового підпису, вони надають механізми для перевірки цілісності й автентичності частин повідомлення. Через основну роль підписів необхідно бути в курсі їхніх базових принципів.

**Цілісність даних**

Підпису, крім іншого, дозволяють перевірити цілісність окремих блоків даних і розпізнати маніпуляції з повідомленнями (зміна, видалення й додавання даних). Для цього за допомогою спеціального криптографічного алгоритму створення гешу (приміром, SHA1, MD5) розраховуються контрольні суми для важливих блоків даних (Message Digest). Геш-алгоритми – необоротні функції, і відновлення вихідних даних за відомим значенням гешу неможливо. Крім того, його величина для різних даних не повинна збігатися (відсутність колізій). Для перевірки геша приймаюча сторона ще раз розраховує контрольну суму й

порівнює отриманий результат із присланим. Якщо обоє значення ідентичні, то цілісність даних дотримана, тоді як в інших випадках велика ймовірність змін повідомлення. Однак цей механізм не дозволяє встановити, які саме зміни внесені, оскільки підпису повідомляють тільки про вірний або невірний результат.

Принцип підписів базується на формуванні контрольних сум обома сторонами (відправником і одержувачем), тому однакова форма подання даних є обов'язковою умовою. Приміром, для того щоб різне написання XML не привело до різниці контрольних сум, варто ввести проміжний етап – нормалізацію XML (Canonicalization).

Однак одного доказу цілісності окремих блоків даних недостатньо для підтвердження автентичності всього повідомлення – потрібно забезпечити єдність окремих блоків. Для цієї мети створюється загальна контрольна сума шляхом об'єднання значення гешу для всіх блоків даних. У результаті здійснюється криптографічний зв'язок блоків, що не залежить від їхнього положення усередині повідомлення: таким чином, загальна контрольна сума дає можливість перевірити автентичність усього повідомлення. Крім того, так можна розпізнати маніпуляцію зі значеннями гешу окремих блоків даних. У процесі транспорту повідомлення загальна контрольна сума захищається від зміни за допомогою криптографічного механізму шифрування. Для реалізації автентичності повідомлення можна застосовувати симетричне шифрування (приміром, HMAC). При цьому ключ, спільно використовуваний відправником і одержувачем, або передається заздалегідь, або створюється відправником у момент передачі, а потім відправляється в зашифрованому виді разом з повідомленням.

Крім перевірки цілісності даних і автентичності повідомлень, підпису надають можливість автентифікації відправника всього повідомлення або його частин (рисунок 3.5).

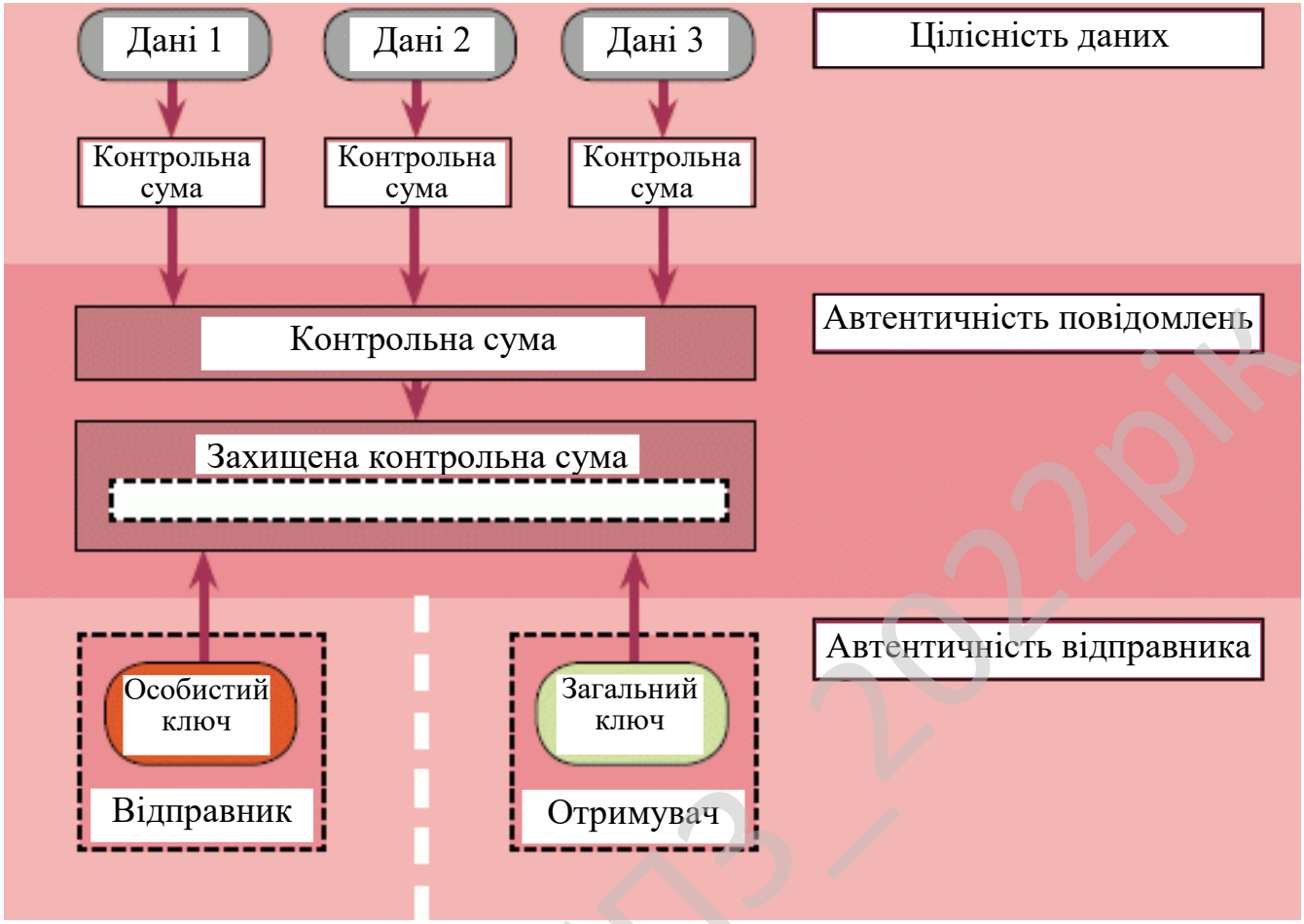


Рисунок 3.5 – Перевірка цілісності даних і автентичності повідомлень, а також автентифікації відправника всього повідомлення або його частин

Це властивість відомо як цифровий підпис. У цьому випадку застосовується не симетричний алгоритм шифрування загальної контрольної суми, а асиметричний алгоритм із застосуванням відкритих ключів: відправник використовує власний ключ для шифрування значення гешу, а прочитати його можна лише за допомогою відкритого ключа, сертифікат якого надає інформацію про власника особистого ключа. Якщо сертифікат, а виходить, і відкритий ключ, викликає довіру, то з його допомогою можна визначити відправника повідомлення.

На перший погляд, набір стандартів і реалізація хмарної архітектури безпеки в SOA здаються нетривіальними. Але його переваги переважують всі складності опису й реалізації. До них відносяться:

– Відділення політики безпеки сервісів від самих сервісів дозволяє побудувати універсальні сервіси захисту для всіх бізнес-додатків без необхідності втручання в бізнес-логікові й "прошивання" функцій безпеки в код бізнесів-додатків.

– Чітке розмежування експертизи. Розроблювачі сервісів формують бізнес-логіку, архітектори й адміністратори визначають політику безпеки й керування.

– Єдина точка керування політикою ІБ.

– Зниження витрат на адміністрування, оскільки зміни в політику безпеки вносяться централізовано, а не в кожному Web-сервісі. Крім того, аудит безпеки для всіх сервісів ведеться з єдиної точки адміністрування.

– Спрощення підтримки й внесення змін у середовище керування й забезпечення безпеки Web-сервісів за рахунок використання єдиних сервісів безпеки для всіх Web-сервісних додатків.

Визначенено, що такий значний набір переваг SOA з погляду безпеки послужить достатнім стимулом для співробітників підрозділів ІБ підтримати зусилля своїх колег з IT-підрозділів по побудові Web-сервісної хмарної архітектури.

### 3.2 Розробка структурної схеми

Одним з найважливіших завдань забезпечення інформаційної безпеки в сервіс-орієнтованих хмарних архітектурах (SOA) є захист потоків корпоративних даних, переданих по каналах загального користування, у тому числі й через Internet. Перспективним методом надійного захисту інформації є метод кодування даних.

Для рішення цього завдання необхідно здійснити кодування інформації на виході з локальної мережі й декодування вхідних у неї даних. Ці функції реалізуються спеціальними програмними або програмно-апаратними

засобами. Якщо захист сегмента корпоративної мережі вже забезпечений міжмережевим екраном, природно покласти на нього також виконання функцій кодування й декодування.

Для реалізації можливостей кодування/декодування повинне бути виконане попередній (початковий) розподіл ключів. Сучасні технології пропонують для цього цілий ряд методів. Після сертифікації та розподілу ключів з'являється можливість здійснення процесу виробітку спільних секретних ключів, що обслуговують сеанс спілкування абонентів.

У результаті кодування весь обмін даними між територіально-віддаленими локальними мережами є захищеним і для користувачів виглядає як обмін усередині однієї локальної мережі, при цьому від користувачів не потрібно застосування яких-небудь додаткових захисних засобів.

**Комплекс кодування міжмережевих потоків**

Програмний комплекс кодування міжмережевих потоків (ККМП) реалізує функції кодування міжмережевих інформаційних потоків у мережах передачі даних протоколу TCP/IP для забезпечення обміну інформацією між територіально-віддаленими локальними мережами. Це забезпечується за допомогою організації віртуальних захищених мереж (Virtual Private Networks – VPN).

Комплекс виконує наступні функції:

– **Кодування міжмережевих потоків.** Функції кодування міжмережевих інформаційних потоків у відкритих мережах передачі даних виконуються шляхом організації VPN. Кожна мережа в складі VPN захищена своїм модулем, що кодує, установлюваним у точці її з'єднання із зовнішніми мережами. Інформація, що захищається, кодується на передавальному модулі й декодується на приймаючому, тобто передається у відкритому виді в межах локальних мереж і в кодованому – за їхніми межами. Кодований трафік передається по протоколу IPsec.

**– Створення контуру безпеки.** Розроблена система дозволяє сформувати

контур безпеки, що поєднує IP-адреси всіх абонентів, що мають доступ у віртуальну захищену мережу. Абонентами VPN можуть бути цілі мережі, підмережі й окремі робітники станції. Крім того, що кодує модуль може бути встановлений на окрему робочу станцію.

**– Вибіркове кодування трафіку.** Формування контуру безпеки служить для поділу трафіку на кодуємий і некодуємий потоки.

**– Модуль, що кодує**. Розроблена система робить виділення пакетів, які необхідно кодувати, на підставі IP-адрес відправника пакета й одержувача пакета й, крім того, перевірки інтерфейсу, через який проходить пакет.

**– Управління ключовою системою.** У розробленій системі реалізована несиметрична ключова система, коли потенційні учасники обміну даними використовують пари довгострокових секретних й відкритих ключів кодування. Кодування здійснюється на основі сеансових ключів, автоматично сформованих за допомогою довгострокових ключів і що мають обмежений час життя. Комплекс здійснює всі необхідні дії по управлінню ключами: генерацію й розподіл довгострокових ключів, виробіток сеансових ключів абонентів, сертифікацію відкритих ключів у довіреному центрі, планову й позаштатну зміну ключів кодування.

**– Реєстрація подій, моніторинг і управління міжмережевими потоками.** Розроблена система здійснює збір і зберігання статистичної й службової інформації про всі штатні й позаштатні події, що виникають при автентифікації вузлів, передачі кодованої інформації, обмеженні доступу абонентів ЛОМ. Засоби моніторингу проводять збір і аналіз протоколів реєстрації від всіх модулів комплексу по кодованому каналі.

**– Захист з'єднань із мобільними клієнтами.** До складу віртуальної захищеної мережі можуть входити мобільні користувачі – віддалені комп'ютери, що підключаються по виділеним або каналам зв'язку, що комутуються. Основною відмінністю Мобільного клієнта є динамічно-призначувана IP-адреса. Носієм ключової інформації для них є електронний ключ eToken.

**Состав Комплексу**

Комплекс складається з наступних компонентів:

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

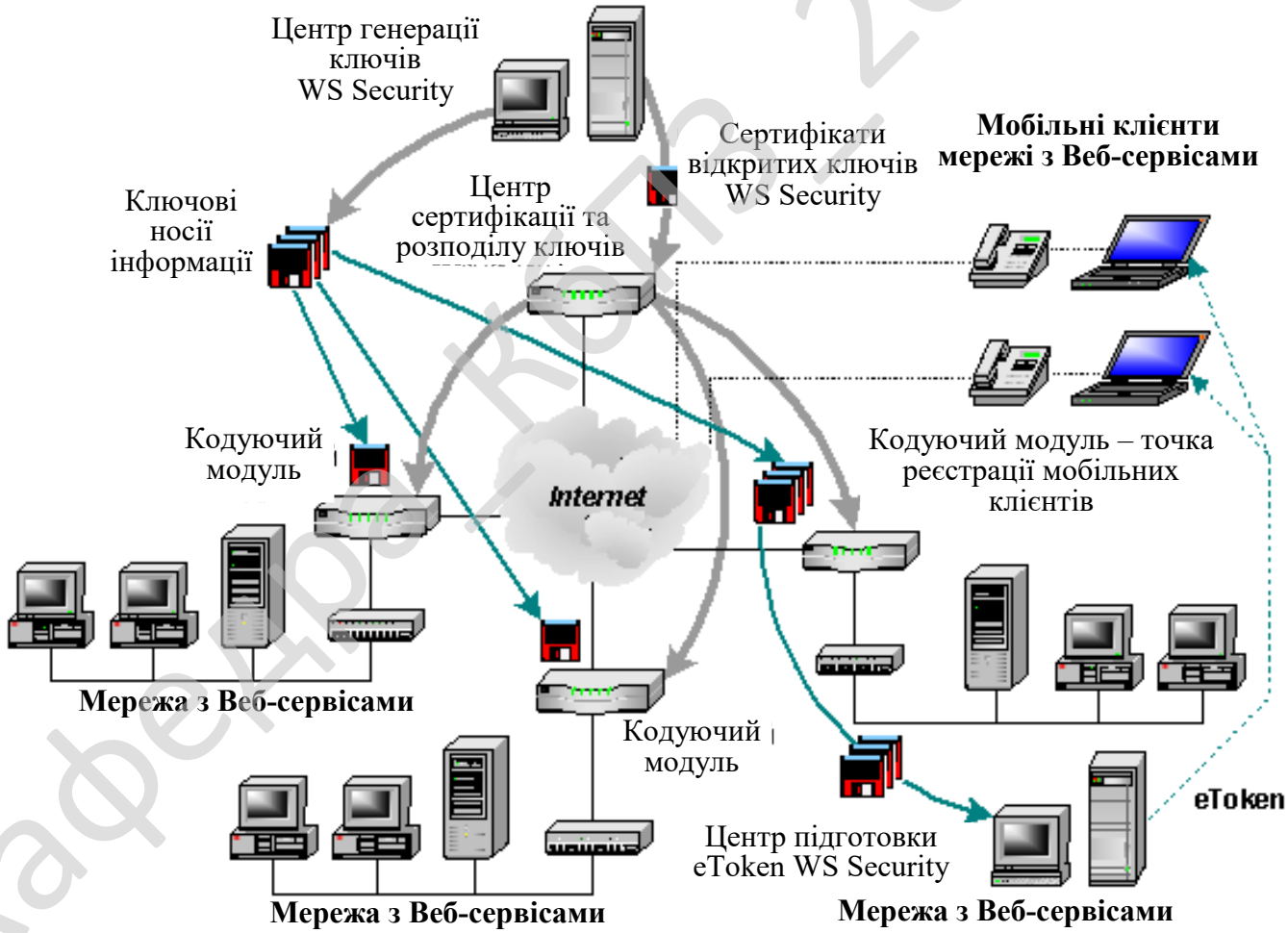


Рисунок 3.6 – Структурна схема системи

**Шлюз із модулем, що кодує/декодувальним**

Шлюз є основним модулем комплексу, що виконує функції маршрутизації, фільтрації й кодування пакетів. Кожний Шлюз призначений для закриття визначеної групи локальних мереж. На комп'ютері-шлюзі встановлюється ядерний модуль с функціями кодування й декодування й запускається програма автентифікації. Функціями шлюзу є:

– Фільтрація трафіку (розподіл на кодуємий/некодуємий потоки).

– Кодування трафіку (кодуємий потік).

– Автентифікація з іншими Шлюзами.

– Реєстрація подій у Центрі моніторингу.

– Забезпечення власного захисту.

**Центр сертифікації та розподілу ключів**

Центр сертифікації та розподілу ключів здійснює управління контуром безпеки, а також виконує наступні функції:

– Одержання зі змінного носія відкритих ключів Шлюзів.

– Видачу будь-якому Шлюзу відкритих ключів будь-яких інших Шлюзів і інформації про відповідні сегменти структури мережі.

– Розсилання Шлюзам повідомлень про зміни структури закритої мережі.

– Вироблення і виконання процедури зміни сеансових ключів.

– Зберігання інформації про структуру мережі.

Центр реалізований у вигляді програмного комплексу, що виконує функції зберігання й видачі відкритих ключів кодування по мережевому запиті від модулів кодування. Центр сертифікації та розподілу ключів може бути встановлений або на окремому (виділеному) комп'ютері, або разом з одним зі Шлюзів кодування.

**Центр генерації ключів**

Даний модуль служить для генерації пар комплементарних ключів, а також є репозитарієм всіх відомих системі ключів. У функції Центра генерації ключів входить:

– генерація пар відкритого й секретного ключів модулів, що кодують;

– генерація пари ключів для сертифікації (еталонного завірення) відкритих ключів модулів, що кодують;

– генерація сертифікатів відкритих ключів, підписаних секретним ключем сертифікації;

– приміщення підписаних сертифікатів відкритих ключів на змінні носії;

– зберігання еталонних копій сертифікованих відкритих ключів в архіві.

Центр генерації ключів – програма, що виконується на ізольованому автоматизованому робочому місці.

**Центр реєстрації ключів**

Центр реєстрації ключів служить репозитарієм всіх відомих системі ключів. У його функції входить:

– Введення зі змінного носія відкритого ключа.

– Введення зі змінного носія закритого ключа Адміністратора безпеки.

– Підпис нового ключа ключем Адміністратора безпеки.

– Приміщення підписаного відкритого ключа в архів довгострокового зберігання й на змінний носій.

– Зберігання еталонних копій сертифікованих (зареєстрованих) відкритих ключів.

Центр реєстрації ключів виконаний у вигляді програми, що виконується на ізольованому автоматизованому робочому місці й призначеної для сертифікації (еталонного завірення) відкритих ключів.

**Центр реєстрації мобільних клієнтів і Мобільний клієнт**

Для забезпечення доступу до корпоративних даних, які захищаються, мобільних абонентів, не підключених до локальних мереж, які захищаються, використовується Центр реєстрації мобільних клієнтів і програмне забезпечення мобільного клієнта комплексу.

Центр реєстрації мобільних клієнтів являє собою спеціальний модуль, що кодує, для підключення довільної кількості мобільних клієнтів.

Мобільний клієнт являє собою програмний модуль, що працює під управлінням ОС Windows і використовуючий апаратні ключі для автентифікації абонента в VPN.

**Центр моніторингу**

Центр моніторингу являє собою мережеве автоматизоване робоче місце із установленим на ньому набором програм, що здійснюють збір і аналіз протоколів, що надходять від всіх модулів комплексу.

**Програма контролю цілісності**

Комплекс містить у собі засобу формування й перевірки контрольних сум файлів. Ці засоби реалізовані у вигляді Програми контролю цілісності, що призначена для визначення й повідомлення системного Адміністратора про зміну, додавання й видалення файлів.

**Адміністрування комплексу**

Настроювання й адміністрування компонентів комплексу здійснюється централізовано з робочого місця Адміністратора безпеки за допомогою графічного інтерфейсу або командного рядка. Віддалене управління здійснюється по захищеному каналі. Комплекс забезпечує автентифікацію Адміністраторів і розмежування доступу до функцій адміністрування.

**Основні особливості комплексу**

Основними особливостями розробленої системи є:

– Повнофункціональна схема управління ключами, що дозволяє здійснювати динамічний розподіл ключів з використанням довіреного центра сертифікації, перевірку дійсності ключової інформації й оповіщення систем кодування про компрометацію ключів.

– Висока надійність функціонування, забезпечувана засобами контролю цілісності, протоколювання й аудита, стійкості до збоїв і відновлення у випадку збоїв і відмов.

– Прозорість кодування переданих даних для абонентів і використовуваного ними програмного забезпечення.

– Висока продуктивність (робота в мережі 100 Мбіт/с без істотного впливу на пропускну здатність).

– Забезпечення необхідної якості сервісу (QoS) і підтримка роботи із сервісами, що пред'являють високі вимоги до величин тимчасових затримок ( IP-телефонія, відеоконференцзв’язок).

– Можливість використання в комплексі з міжмережевими екранами, антивірусними рішеннями й засобами контекстного аналізу.

– Використання відкритих стандартів – протокол тунелювання мережевих пакетів відповідає стандартам IETF IPsec.

### 3.3 Розробка функціональної схеми

Розглянемо функціональну схему розробленої системи. Вона зображена на рисунку 3.7.

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 зкмпроментовані.

6. Центр реєстрації.

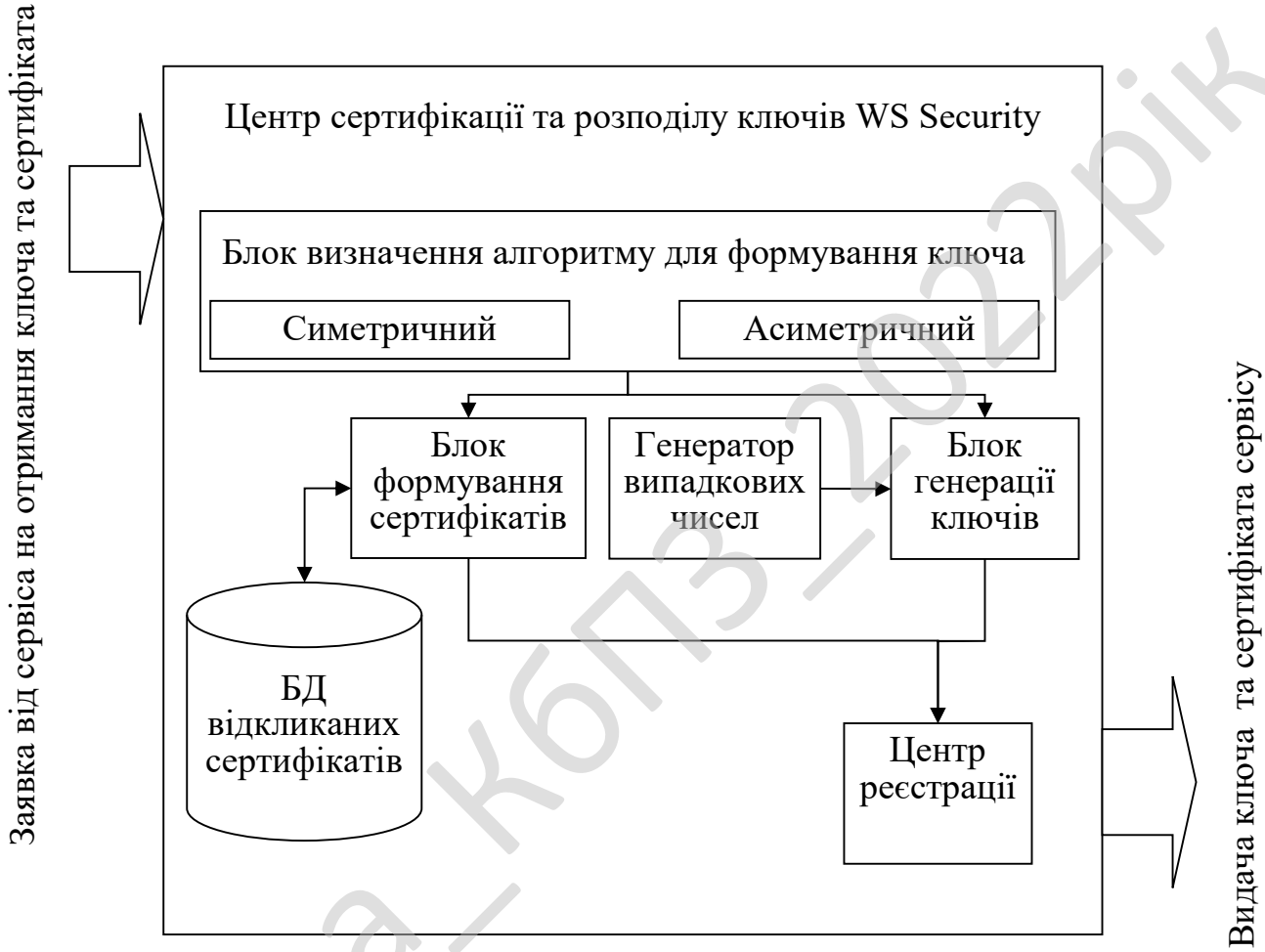


Рисунок 3.7 – Функціональна схема центру сертифікації та розподілу ключів

Розглянемо ці блоки та їх функціональність більш детальніше.

**Центр сертифікації та розподілу ключів**

Центр сертифікації та розподілу ключів (ЦРК) призначений для обслуговування наступних запитів: на видання сертифікатів ЕЦП, на відкликання, призупинення й поновлення припиненої дії сертифікатів абонентів,

сформованих на мережевих 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 сертифікації та розподілу ключів), так і по запиту користувача корпоративної мережі, сформованого на його мережевому 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.

**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 сертифікації та 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 сертифікації та 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-орієнтованої хмарної 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 $Y_4 = E_{K_{\text{О}}^{\text{П1}}}\left(N_1, N_2\right)$, 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 $Y_5 = E_{K_{\text{О}}^{\text{П2}}}\left(Y, N_2\right)$.

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 $\left[K_{\text{О}}^{\text{П1}}, K_{C}^{\text{П1}}\right]$ (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 $C_{\text{П1}} = E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$ 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 $C_{\text{П1}} = E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$, отриманий від авторитетного джерела, сервісу 2. Останній, знаючи відкритий ключ авторитетного джерела сертифікатів $K_{\text{О}}^{\text{А}}$, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом $D_{K_{\text{О}}}^{\text{А}}\left[C_{\text{П1}}\right] = D_{K_{\text{О}}}^{\text{А}}\left[E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]\right] = \left[K_{\text{О}}^{\text{П1}}, ID_{\text{П1}}, T_{\text{П1}}\right]$.

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 $C_{\text{П2}} = E_{\text{Кс}}^{\text{А}}\left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right]$. Пересилає його сервісу 1. Останній, знаючи відкритий

ключ авторитетного джерела сертифікатів $K_{\text{О}}^{\text{А}}$, має можливість прочитати й упевнитися, що отримане повідомлення є сертифікатом

$$D_{K\text{о}}^{\text{А}}[C_{\text{П2}}] = D_{K\text{о}}^{\text{А}}\left[E_{K\text{с}}^{\text{А}}\left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right]\right] = \left[K_{\text{О}}^{\text{П2}}, ID_{\text{П2}}, T_{\text{П2}}\right].$$

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 $\left[K_{\text{О}}^{\text{П1}}, K_{\text{С}}^{\text{П1}}\right]$, відповідно, відкритий і секретний.

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

3. Сервіс 2, одержавши повідомлення $X_{\text{П1}} = \left[ID_{\text{П1}}, K_{\text{О}}^{\text{П1}}\right]$ від сервісу 1, так само генерує свою пару ключів $\left[K_{\text{О}}^{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$.

4. Сервіс 2, використовуючи відкритий ключ $K_{\text{О}}^{\text{П1}}$ сервісу 1, шифрує й передає повідомлення $Y_{\text{П2}} = E_{K_{\text{О}}^{\text{П1}}}\left[ID_{\text{П2}}, K_{\text{С}}^{\text{П2}}\right]$ 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* зображений на малюнку 2.10 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 $Y_{П1} = E_{K_O^{П2}}\lfloor ID_{П1}, N_1 \rfloor$.

4. Сервіс 1, використовуючи $K_O^{П2}$, передає сервісу 2 повідомлення $Y_{П1} = E_{K_O^{П2}}\lfloor N_2 \rfloor$, 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 $K_C$, що двічі шифрується з використанням: свого секретного ключа $E_{K_C^{П1}}[K_C]$ й відкритого ключа сервісу 2 $E_{K_O^{П2}}[\ E_{K_C^{П1}}[K_C]\ ]$. Після виконання процедури шифрування повідомлення $Y_{П1}=E_{K_O^{П2}}[\ E_{K_C^{П1}}[K_C]\ ]$ 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 $K_C$.

### 3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.8.

З рисунку видно, що на початку роботи програми відбувається процес початку/кінця роботи програмного продукту.

Цей процес взаємодіє взаємодіє з наступними процесами:

– Процес генерації пари ключів.

– Процес припинення строку дії сертифіката.

– Процес знищення ключа.

Процес генерації пари ключів у свою чергу взаємодіє з наступними процесами:

– Процес транспортування ключа.

– Процес реєстрації/видання сертифіката.

Процес реєстрації/видання сертифіката з однієї сторони взаємодіє з процесом розповсюдження сертифіката, а з іншої сторони взаємодіє з процесом зберігання сертифіката, який є першим процесом, у ланцюжку життєвого циклу сертифікатів.

Життєвий цикл сертифікатів складається з послідовної взаємодії один з одним наступних процесів:

– Процес зберігання сертифіката.

– Процес використання сертифіката.

– Процес зміни статуса сертифіката.

– Процес припинення строку дії сертифіката.

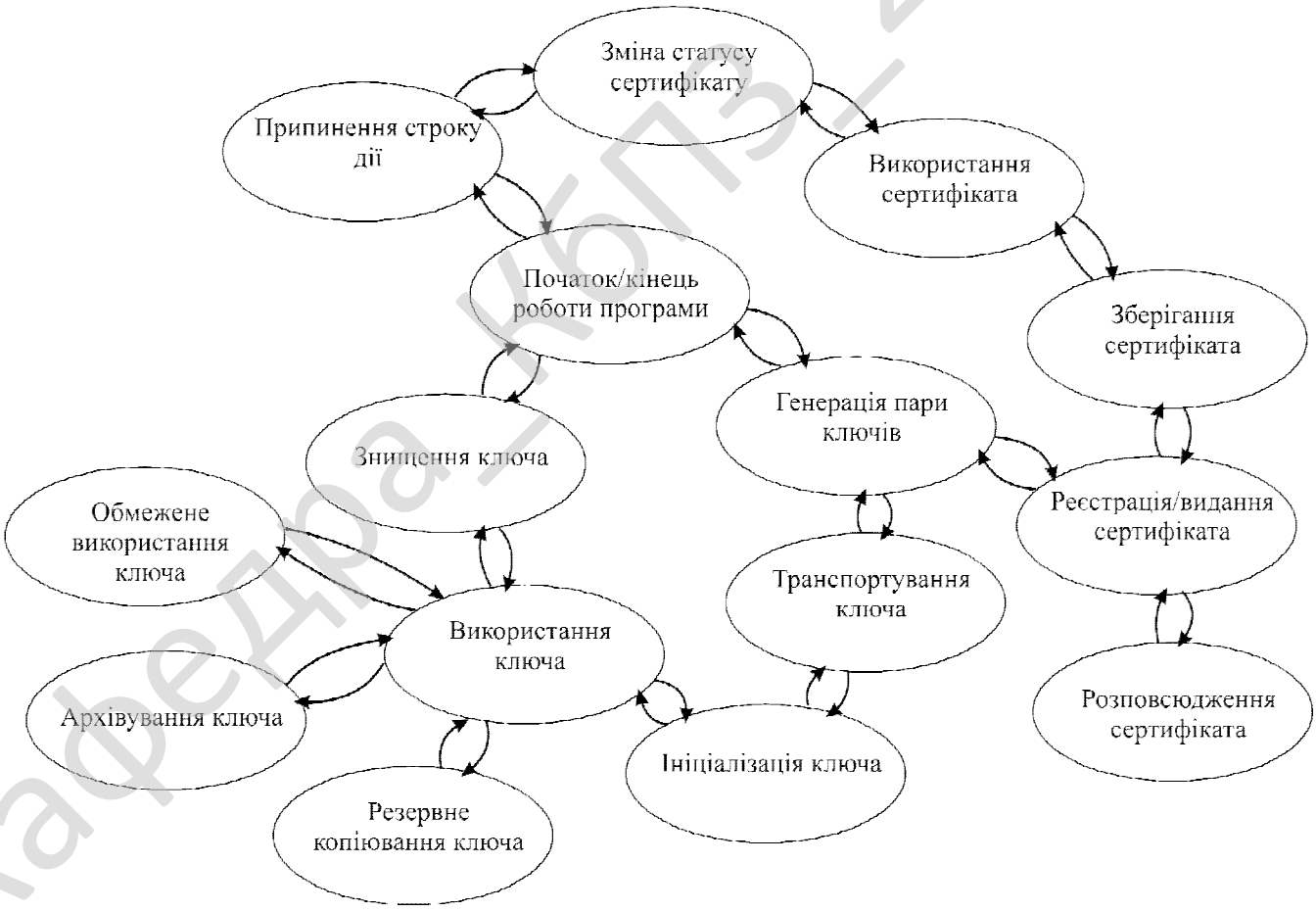


Рисунок 3.8 – Діаграма процесів розробленої системи

Розглянувши життєвий цикл сертифіката, перейдемо до життєвого циклу сертифікації та розподілу ключів.

Цей цикл починається з процесу генерації пари ключів.

Після цього запускається процес транспортування ключа.

Процес транспортування ключа взаємодіє з процесом ініціалізації ключа.

Останній процес взаємодіє з процесом використання ключа.

Процес використання ключа взаємодіє з наступними процесами:

– Процес ініціалізації ключа.

– Процес резервного копіювання ключа.

– Процес архівування ключа.

– Процес обмеженого використання ключа.

– Процес знищення ключа.

Процес знищення ключа є завершуючим процесом життєвого циклу центру сертифікації та розподілу ключів сервіс-орієнтованих архітектур. Відповідно він взаємодіє з процесом початку/кінця роботи програми.

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

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

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема алгоритму роботи основної програми зображена на рисунку 4.1.

Після запуску програми на екран виводиться головне вікно програми.

Після виведення головного вікна програми відбувається реєстрація користувачів.

Наступним етапом є заявка від сервісу на отримання ключа та сертифіката.

Після цього відбувається визначення алгоритму для формування ключа.

Наступним кроком є генерація випадкових чисел.

За генерацією випадкових чисел, відбувається генерація ключів, у якій беруть участь згенеровані випадкові числа.

Після того, як згенеровані ключі, відбувається формування сертифікатів.

Сформовані сертифікати зберігаються у базі даних відкликаних сертифікатів.

Після виконання усіх підготовчих процедур центру сертифікації та розподілу ключів сервіс-орієнтованої хмарної архітектури, відбувається видача згенерованих ключів, та сформованих сертифікатів сервісам користувачів.

Далі починається робота користувачів з отриманими ключами й сертифікатами.

Під цей час обробляються запити зареєстрованих користувачів та відбувається обслуговування сертифікатів.

На цьому основна програма закінчує свою роботу.


Рисунок 4.1 – Блок-схема алгоритму роботи основної програми

На рисунку 4.2 зображена блок-схема алгоритму роботи підпрограми обробки запитів.

З неї ми бачимо, що ця підпрограма працює наступним чином.

Спершу відбувається введення параметрів з’єднання з сервером.

Після цього очікується вибір користувача, з’єднуватися з сервером, або ні.

Якщо він вирішує не з’єднуватися, то програма перебуває у режимі очікування.

Якщо ж він вирішує з’єднуватися, то відбувається підключення до сервера сертифікації та розподілу ключів сервіс-орієнтованої хмарної архітектури.

Першим кроком є перевірка сертифікату на дійсність. Якщо він не дійсний, то виконуються наступні дії:

– Генерується та відправляється запит на одержання сертифіката.

– Отримується сертифікат та закритий ключ.

Наступним кроком є вибір користувачем встановленого захищеного з’єднання.

Якщо він не вибирає захищене з’єднання, то відбувається виведення попередження, про те, що йде робота по незахищеному каналу, та починається робота з відкритими ресурсами мережі по незахищеному каналу.

Якщо ж користувач обирає захищене з’єднання, то відбувається обмін ключами з сервером, та встановлення захищеного з’єднання.

Після цього йде перевірка, чи насправді встановилося захищене з’єднання.

Якщо воно не встановилося, то відбувається перехід ло перевірки дійсності сертифікатів.

Якщо ж захищене з’єднання встановилося, то відбувається робота з закритими ресурсами мережі по захищеному каналу.

Після виконання усіх вище перерахованих кроків, користувач обирає, працювати йому далі з підпрограмою обробки запитів, або ні.

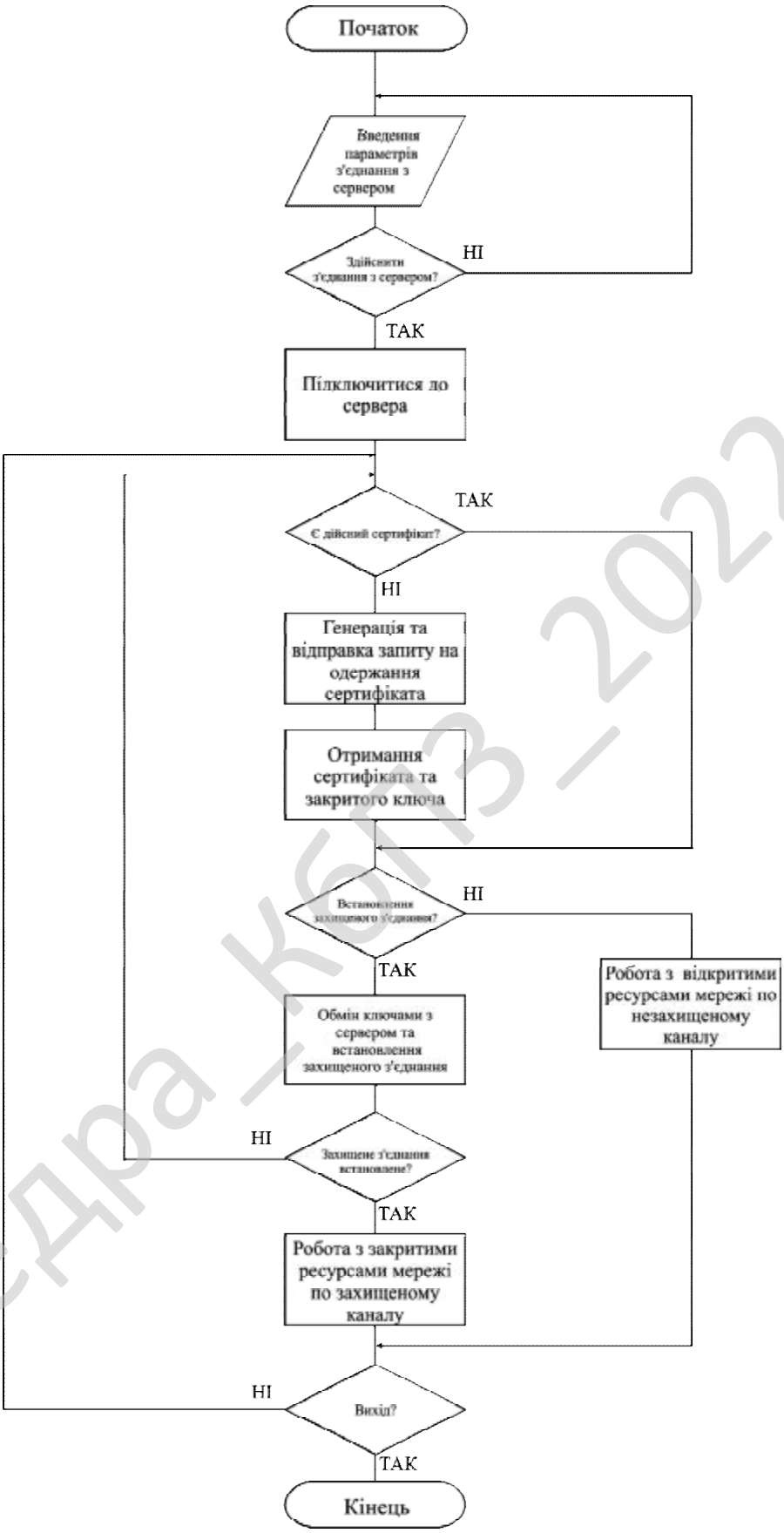


Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми обробки запитів користувачів

На рисунку 4.3 зображено блок-схему алгоритму роботи підпрограми обслуговування сертифікатів.

Робота цієї підпрограми складається з виконання наступних кроків.

Спершу відбувається виведення поточного стану мережі, її сервисів та сертифікатів користувачів.

Після цього користувач обирає вносити, або ні зміни у базу даних сертифікатів.

Якщо треба внести зміни, то відбувається поетапне виконання наступних кроків:

– Виведення редактору бази даних сертифікатів.

– Редагування бази даних сертифікатів.

– Перевірити сертифікати та ключі користувачів.

Якщо ж зміни вносити не треба, то відбувається вибір користувачем чи внести зміни в базу даних сервісів, чи ні.

Якщо потрібно внести зміни у базу даних сервісів, то відбувається послідовність наступних кроків:

– Виводиться редактор бази даних сервісів.

– Редагується база даних сервісів.

Після виконання вищеперерахованих дій, користувач обирає переглядати, або ні йому журнал подій.

Якщо він обирає перегляд журналу подій, то відбувається виконання наступних дій:

– Виведення вікна журналу подій у мережі.

– Перегляд журналу подій.

– Аналіз безпеки мережі.

Якщо ж користувач не бажає провести аудит подій, переглянувши журнал подій, то він може обрати меню зміни параметрів системи.

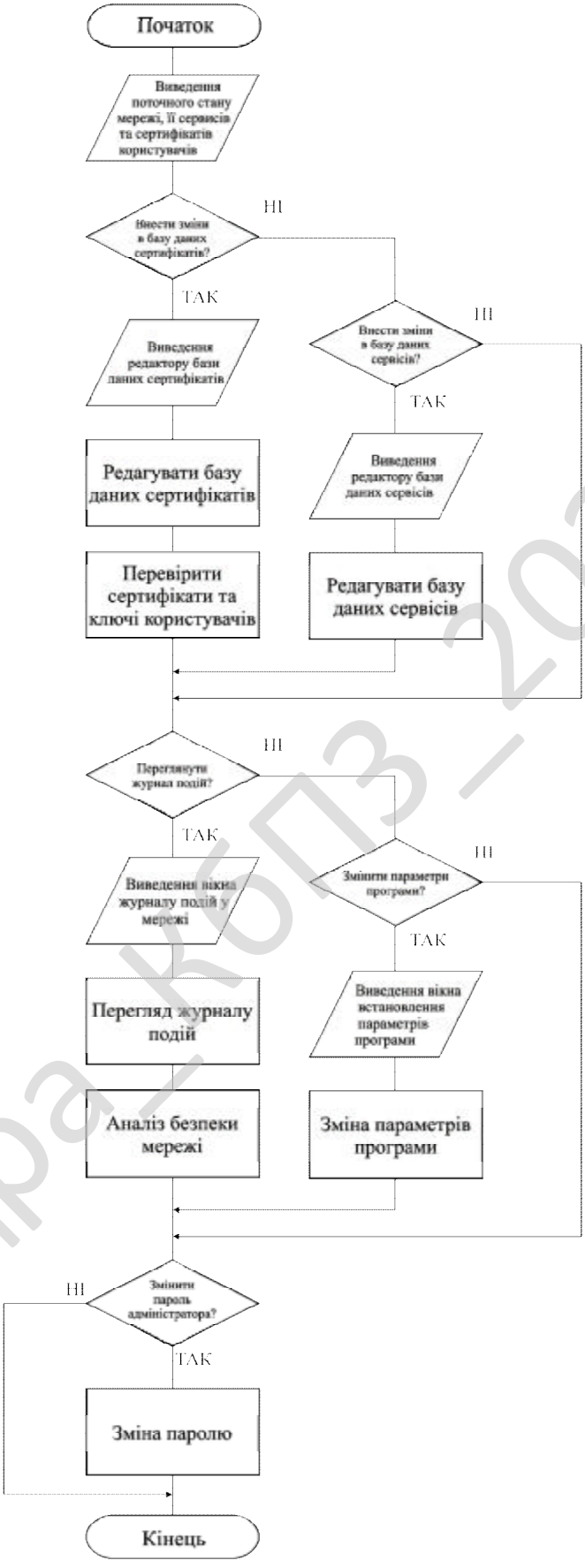


Рисунок 4.3 – Блок-схема алгоритму роботи підпрограми обслуговування сертифікатів

Якщо користувач обирає зміну параметрі системи, то відбувається виконання наступних кроків роботи підпрограми:

– Виведення вікна встановлення параметрів програми.

– Зміна параметрів програми.

Останнім пунктом, який може вибрати користувач є можливість зміни паролю адміністратора.

Якщо він обирає можливість зміни паролю адміністратора, то він його змінює.

На цьому підпрограма обслуговування сертифікатів завершує свою роботу.

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.

```
//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();
    }
// 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 таємного ключа RSA
    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();
    }
// Редагування відкритого ключа RSA
    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();
    }
  }
```

## 4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму UMAC (код автентифікації повідомлення на основі

універсального гешування) – один з видів коду автентичності повідомлень (MAC).

Швидка «універсальна» функція використовується, для того, щоб гешувати вхідне повідомлення M у короткий рядок. До цього рядка потім застосовується функція XOR із псевдовипадковим значенням, у результаті чого ми одержуємо тег UMAC:

де K1 і K2 – секретні випадкові ключі, які мають одержувач і відправник.

Звідси видне, що безпека UMAC залежить від того, яким випадковим способом відправник і одержувач вибрали таємну геш-функцію й псевдовипадкову послідовність. При цьому значення Nonce міняється кожний такт. Через використання Nonce, приймач і передавач повинні знати час відправлення повідомлення й принцип створення значення Nonce. Замість цього можна використовувати в якості Nonce будь-яке інше неповторюване значення, наприклад порядковий номер повідомлення. При цьому даний номер не зобов'язано бути секретним, головне щоб він не повторювався.

UMAC розрахований на використання 32-х, 64-х, 92-х, і 128-бітових тегів, залежно від необхідного рівня безпеки. UMAC звичайно використовується разом з алгоритмом шифрування AES.

Функція створення ключа й псевдовипадкової послідовності

Створення псевдовипадкових байтів необхідно для роботи UHASH і при створенні тегів

**Вибір блокового шифру**

Для своєї роботи UMAC використовує блоковий шифр, вибір якого визначають наступні константи:

– BLOCLEN – довжина, у байтах, блоку з яким працює блоковий шифр.

– KEYLEN – довжина, у байтах, ключа блокового шифру.

При цьому використовується функція

– ENCIPHER(K,P) – зашифрувати рядок P з BLOCLEN байтів, використовуючи ключ K.

Приклад: якщо використовується AES з 16-байтним ключем, то BLOCLEN буде рівним 16( тому що AES працює з 16-байтними блоками).

**KDF – функція створення ключа**

Ця функція генерує послідовність псевдовипадкових байтів, використовуваних для ключових геш-функцій.

Вхід:

– K – рядок довжиною KEYLEN байт. / / Ключ блокового шифру.

– Index – ненегативне ціле число менше, чим 2 ^ 64.

– Numbytes – ненегативне ціле число менше, чим 2 ^ 64.

Вихід:

– Y – рядок довжини numbytes байт.

**PDF: функція створення псевдовипадкового числа**

Ця функція ухвалює ключ і даний час і повертає псевдовипадкове число для використання його в тегу покоління. За допомогою цієї функції можуть бути отримані числа довжиною 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– Nonce – рядок довжиною від 1 до BLOCKLEN байт.

– Taglen – ціле число 4, 8, 12 або 16.

Вихід:

– Y – послідовність байтів довжини taglen.

**Генерація UMAC-тегів**

Генерація UMAC-тегів відбувається за допомогою UHASH функції при використанні Nonce значенні й отриманої до цього рядка. Їхня довжина може бути 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– M – рядок довжиною менше 267 біт.

– Nonce – випадкове число від 1 до BLOCKLEN байт.

– Taglen – ціле 4, 8, 12 або 16.

Вихід:

– Тег, послідовність байтів довжиною taglen.

Алгоритм обчислення тегів:

Hashedmassage = UHASH(K, M, Taglen)

Pad = PDF(K, nonce, Taglen)

Tag = Pad xor Hashedmassage

**UMAC-32 UMAC-64 UMAC-96 UMAC-128**

Дані позначення містять у своїй назві певне значення довжини тегу:

– UMAC-32 ( K, M, Nonce) = UMAC (K, M, Nonce, 4).

– UMAC-64 ( K, M, Nonce) = UMAC (K, M, Nonce, 8).

– UMAC-96 ( K, M, Nonce) = UMAC (K, M, Nonce, 12).

– UMAC-128 (K, M, Nonce) = UMAC (K, M, Nonce, 16).

**Універсальна функція гешування(UHASH)**

UHASH – універсальна функція гешування, серцевина алгоритму UMAC. UHASH – функція працює в три етапи. Спочатку до вхідного повідомлення застосовується L1-HASH, потім до цього результату застосовується L2-HASH і, нарешті, до результату застосовується L3-HASH . Якщо при цьому довжина вхідного повідомлення не більш 1024 біт, то L2-HASH не використовується. Тому що функція L3-hash повертає тільки слово довжини 4 байта, те якщо потрібно одержати геш довжини більше 4 байт, здійснюється кілька ітерацій даної трирівневої схеми.

**Універсальна функція**

Нехай функція гешування вибирається із класу геш-функцій H, які відображають повідомлення в D, набір усіляких образів повідомлення. Цей клас називається універсальним, якщо для яких-небудь окремих пар повідомлень, існує на множини H/D функцій, функція, яка відображає їх в елемент D. Зміст цієї

функції в тому, що якщо третя сторона прагне замінити одне повідомлення іншим, але при цьому вважає, що геш-функція була обрана абсолютно випадково, те ймовірність не виявлення підміни стороною, що ухвалює, прагне до 1/D.

**L1-hash – перший етап**

L1-hash розбиває повідомлення на шматки з 1024 байт і до кожного шматка застосовує алгоритм гешування називаний NH. Вихідний результат алгоритму NH в 128 раз менше вхідного.

**L2-hash – другий етап**

L2-hash працює з виходом L1-hash, використовує поліноміальний алгоритм POLY. Другий етап гешування використовується, тільки якщо довжина вхідного повідомлення більше 16 мегабайт. Використання алгоритму POLY потрібно для того, щоб уникнути тимчасову атаку. На виході з алгоритму POLY виходить 16 байтне число.

**L3-hash – третій етап**

Цей етап потрібно для того щоб з вихідних 16 байтів алгоритму L2-hash одержати 4-байтне значення.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблене програмне забезпечення призначене для хмарних сервісів з використанням ЦСК.

Програмно-апаратні вимоги:

– Загальний обсяг ОЗП: 512 Мбайт.

– Вільний простір на жорсткому диску: 15 Мбайт.

– Операційна система Microsoft Windows 10/11.

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1

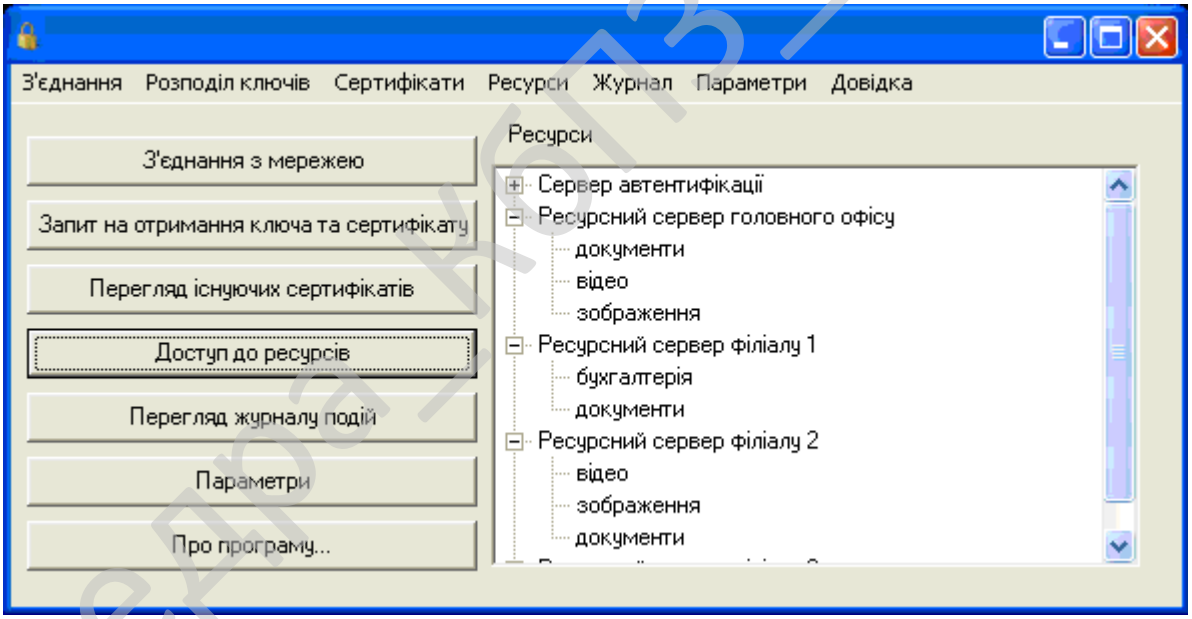


Рисунок 5.1 – Основне вікно програми

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку «Про програму…», після чого на екрані з’явиться вікно показане на рисунку 5.2.

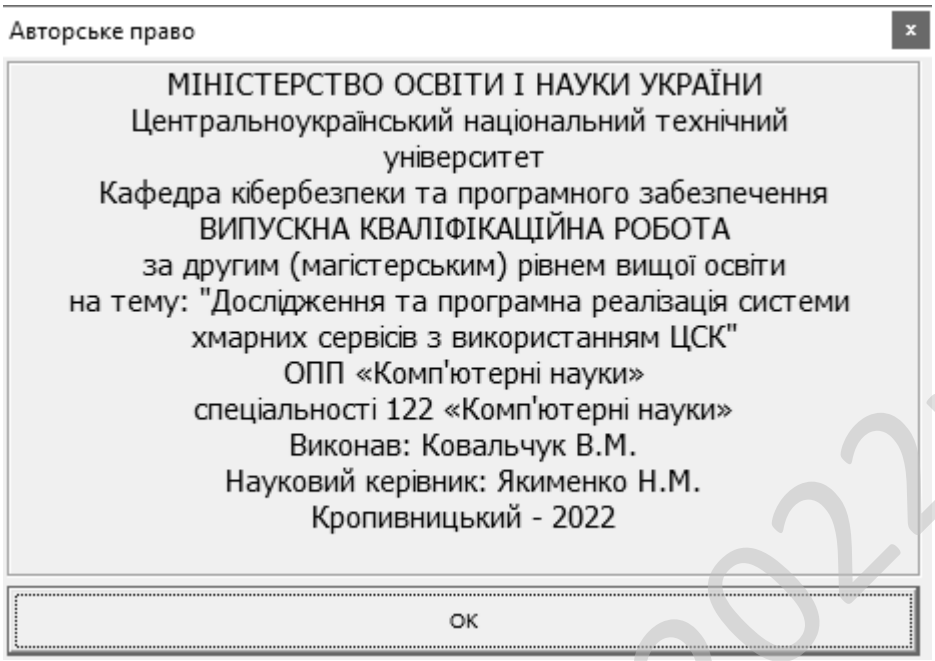


Рисунок 5.2 – Довідка

# 6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи хмарних сервісів з використанням ЦСК.

*Метою розробки* є дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК.

*Об'єктом дослідження* є процес хмарних сервісів з використанням ЦСК.

*Предметом дослідження* є методи хмарних сервісів з використанням ЦСК.

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

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

– Удосконалено метод хмарних сервісів з використанням ЦСК.

– Розроблено вітчизняний продукт хмарних сервісів з використанням ЦСК, який має більш широкі можливості, на відміну від існуючих аналогів.

# 7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

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

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 24 днів (один місяць).

В магістерській роботі було проведено дослідження та виконана програмна реалізація системи хмарних сервісів з використанням ЦСК.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

| Показники | Позначення | Характеристика або величина |
|---|---|---|
| 1 | 2 | 3 |
| 1. Кількість розроблених програм період, шт. | N | 1 |
| 2. Кількість екземплярів програм, шт. | Ne | 320 |
| 3. Запланований термін розробки, днів | Fpq | 24 (1 місяць) |
| 4. Група задачі підсистеми управління (1-6) | – | 1 |
| 5. Ступінь новизни задачі (А, Б, В, Г) | – | Б |
| 6. Складність алгоритму (1, 2, 3) | – | 2 |

Продовження таблиці 7.1

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

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 28. Досвід роботи з мовою і інструментами середовища розробки (1-6) | – | 2 |
| 29. Досвід роботи з програмними інструментами розробки (1-6) | – | 3 |
| 30. Розробка ПЗ для декількох серверів одночасно (1-6) | – | 3 |
| 31. Вимоги до дотримання встановленого графіка робіт (1-6) | – | 2 |
| 32. Вартість ПЗ у розробника (НМА), грн. | – | 32000 |
| 33. Норматив додаткової зарплати, % : | $Н_д$ | 10 |
| 34. Норматив відрахувань у соціальні фонди, % | Нс | 22 |
| 35. Норматив загальногосподарських витрат, % | Нг | 15 |
| 36. Норматив витрат на освоєння нових мов програмування, % | Нп | 15 |
| 37. Рівень рентабельності програмної продукції, % | Ре | 50 |
| 38. Ставка податку на додану вартість, % | Ндв | 20 |

### 7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

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

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

B – показник ступеня, що визначається співвідношенням:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 50 = 94 \text{ люд/день.}$$

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

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

| Стадії розробки | Трудомісткість за типовими нормами та розрахунками | |
|---|---|---|
| | Величина, люд/дні | Підстава |
| Технічне завдання | 9 | Д5 |
| Ескізний проект | 10 | Д6 |
| Технічний проект | 15 | Д7 |
| Робочий проект | 94 | Ф 7.1- 7.4 |
| Впровадження | 15 | Д13 |
| Всього | 143 | – |

### 7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

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

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

$$Ч = \frac{143 \cdot 1}{24\text{-}3} = 6,8 \text{ ставки.}$$

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

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

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

| Найменування обладнання | Профілактичне обслуговування | | | |
|---|---|---|---|---|
| | Кількість хв. на один. обл. | Кількість облад-нання | Затрати часу в хв. | Затрати часу в год. |
| Системний блок ПК | 90 | 10 | 900 | 15 |
| Монітор | 60 | 10 | 600 | 10 |
| Клавіатура | 30 | 10 | 300 | 5 |
| Маніпулятор «мишка» | 30 | 10 | 300 | 5 |
| Принтер матричний | 60 | 0 | 0 | 0,0 |
| Принтер лазерний | 120 | 2 | 240 | 4 |
| Принтер струминний | 60 | 1 | 60 | 1 |
| Сканер | 20 | 1 | 20 | 0,33 |
| Концентратор-маршрутизатор | 30 | 2 | 60 | 1 |
| Кабельні господарства ЛОМ на 1 м.п. | 2,5 | 300 | 750 | 12,5 |
| Копіювальний апарат | 140 | 1 | 140 | 2,33 |
| Усього за рік: | | | $З_ч$ | 56,16 |

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{56 \cdot 1}{1,2} = 47 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 47/(24 \cdot 8) = 0{,}25 \text{ ставки.}$$

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

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

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

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Адміністратор загальної мережі, аналітик | Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi | 2 | 0,5 |
| | Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS) | 0,5 | |
| | Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ | 0,5 | |
| | Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет | 1 | |
| Всього | | 4 | |

Продовження таблиці 7.4

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

| Посада | Кількість ставок | Середньомісячний оклад, грн. | Всього за період розробки, грн. |
|---|---|---|---|
| Керівник (ІТ-менеджер) | 0,25 | 21000 | 5250 |
| Продакт-менеджер | 0,5 | 17136 | 8568 |
| Інженер-програміст | 6,8 | 19000 | 129200 |
| Інженер-електронщик | 0,25 | 16000 | 4000 |
| Інженер-системотехнік | 0,25 | 16000 | 4000 |
| Адміністратор мережі | 0,5 | 19000 | 9500 |
| Системний програміст | 0,25 | 16000 | 4000 |
| Дизайнер WEB | 0,5 | 16000 | 8000 |
| Інженер-верстальник | 0,25 | 18000 | 4500 |
| Бухгалтер-економіст | 0,5 | 15000 | 7500 |
| Всього за період розробки | $R_{сп} = 10{,}05$ | - | $Ф_{роб} = 184518$ |

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{184518}{10{,}05 \cdot 24} = 765 \; грн.$$

## 7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

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

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

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

$$Б_{уд} = 13 \cdot 8 \cdot 20000 = 2080000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 208000 грн.

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

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

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

$$І_{нв} = 13 \cdot 3500 = 45500 \text{ грн.}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за пропозицією інтернет ресурсу hotline за 24.10.22 – джерело https://hotline.ua

Таблиця 7.6 – Специфікація

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Персональний комп’ютер | | 11457 |
| Системний блок | | 7509 |
| Процесор | Intel Core i7-4770K 3.5GHz/5GT/s/8MI (BX80646I74770K) s1150 BOX | - |
| Системна плата | MSI B85M-E45 Socket 1150 Intel B85 OEM Refurbished | - |
| Відеокарта | nVidia GeForce 8600 GTS, 256 MB GDDR3 128-bit / 2x DVI 1x S-Video | - |
| Жорсткий диск | SSD 240 Gb | - |
| Оперативна пам’ять | Kingston DDR3 4GB (KVR1333D3N9/2G Intel/AMD | - |
| DVD-привод | - | - |
| Корпус | ATX Middle Tower GIGABYTE GZ-X Silver 500W (GZ-X4 Silver) | - |
| Кулер | – | – |
| Кардрідер внутрішній | USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black | - |
| інше | Клавіатура, мишка | Подару-нок |
| Монітор | 22" TFT, ASUS VW223D ( 5ms, 300/3000: 1 170/160, D-SUB, Wide) | 2600 |
| Принтер лазерний | Canon i-SENSYS LBP6030W | 2700 |

Продовження таблиці 7.6

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Принтер струминний | Epson Stylus Photo P50 (C11CA45341) + USB cable | 5500 |
| Сканер | Epson Perfection V37 Photo | 2970 |
| Копіювальний апарат | Canon i-SENSYS MF217W with Wi-Fi | 5965 |
| Пристрій безперебійног живлення | UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS) | 1348 |

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

| Найменування обчислювальної техніки | Кількість, шт. | Ціна за одиницю, грн. | Витрати на транспортування, монтаж та випробовування. | Загальна вартість, грн. |
|---|---|---|---|---|
| Персональні комп’ютери | 13 | 11457 | 14894,1 | 163835,1 |
| Принтер лаз. | 2 | 2700 | 540 | 5940 |
| Принтер струм. | 1 | 5500 | 550 | 6050 |
| Сканери | 1 | 2970 | 297 | 3267 |
| Копіюв. апарат | 1 | 5965 | 596,5 | 6561,5 |
| Всього | – | – | – | 185653,6 |

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

| Групи та види основних фондів | Балансова вартість, грн. | Амортизація | |
|---|---|---|---|
| | | Норма, % | Відрахування, грн. |
| 1 | 2 | 3 | 4 |
| Група 3 | | | |
| 1. Будівлі | 2080000 | - | - |
| 2. Передавальні пристрої | 208000 | - | - |
| Всього по групі | 2288000 | 5 | 114400 |
| Група 4 | | | |
| 3. Обчислювальна техніка | 185654 | - | - |
| Всього по групі | 185654 | 50 | 92827 |
| Група 5, 6 | | | |
| 4. Вимірювальні пристрої | 3999 | 25 | 1000 |
| 5. Транспортні засоби | 97500 | 20 | 19500 |
| 6. Господарський інвентар | 45500 | 25 | 11375 |
| Всього по групі | 146999 | - | 31875 |
| 7. Нематеріальні активи | 32000 | 10 | 3200 |
| Разом | $К_р$ = 2652653 | | $А_р$ = 242302 |

Примітка: вартість автомобіля взята по даним з прайс-листа автосалону автотрейдінг, вкладки автобазар, джерело http: //www.auto-trading.com.ua/sale/lot20772.html, складає 97500 грн.

## 7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

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

$$З_о = 765 \cdot 143/320 = 342 \text{ грн.}$$

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

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

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

$$З_д = 342 \cdot 10 \cdot 0{,}01 = 34 \text{ грн.}$$

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

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

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

$$С_{оц} = 0{,}01 \cdot 22(342+34) = 83 \text{ грн.}$$

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

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

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 342 \cdot 15 \cdot 0{,}01 = 51 \text{ грн.}$$

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

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

де: $З_{М1}$ – вартість паперу, грн.; $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.; $З_{М3}$ – вартість фарби, картріджей, тонеру, грн.; $N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,2 пачек паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 200 \cdot 0,2 = 40 \text{ грн.}$$

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

$$З_{М2} = \sum Ц_д\,, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23 грн./шт., DVD-R box – 32,15 грн./шт.

$$З_{М2} = 32,15 \cdot 100 = 3215 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

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

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (40 + 3215 + 1702)/320 = 15 \text{ грн.}$$

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

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

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

$$О_п = 342 \cdot 15 \cdot 0,01 = 51 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 320 прим.):

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

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

$$A_м = 242302 \cdot 1/(320 \cdot 12) = 63 \text{ грн.}$$

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

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

$$C_п = 342+34+83+51+15+51+63 = 639 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 639 = 320 \text{ грн.}$$

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

| Найменування статей витрат | Позначення | Величина, грн |
|---|---|---|
| 1 | 2 | 3 |
| 1. Основна зарплата виконавців | $З_о$ | 342 |
| 2. Додаткова зарплата виконавців | $З_д$ | 34 |
| 3. Відрахування на соціальні потреби | $С_{оц}$ | 83 |
| 4. Загальногосподарські витрати | $Г_{осп}$ | 51 |
| 5. Витрати на матеріали | $З_М$ | 15 |
| 6. Освоєння нових операційних систем, мов програмування | $О_п$ | 51 |

Продовження таблиці 7.9

| 1 | 2 | 3 |
|---|---|---|
| 7. Амортизація основних фондів | $А_м$ | 63 |
| 8. Повна собівартість програмного забезпечення | $С_п$ | 639 |
| 9. Плановий прибуток | $П_р$ | 320 |
| 10.Ціна підприємства $Ц_п = С_п + П_р$ | $Ц_п$ | 959 |
| 11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$ | $ПДВ$ | 191,8 |
| 12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$ | $Ц$ | 1150,8 |

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

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

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

| Найменування капітальних вкладень | Сума за варіантами, грн. | |
|---|---|---|
| | Базовий | Новий |
| Вартість програмної продукції | – | 1151 |
| Всього капітальних витрат | – | 1151 |

### 7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

| Найменування статей витрат | Позначення | Сума витрат за варіантами, грн. | |
|---|---|---|---|
| | | Базовий | Новий |
| 1. Витрати на технічне обслуговування | $З_р$ | 25498 | 19800 |
| 2. Витрати на електроенергію | $З_{ел}$ | 0 | 0 |
| 3. Витрати на амортизацію | $З_{ам}$ | 0 | 576 |
| Всього витрат за рік | $І$ | 25498 | 20376 |

Витрати на профілактичні роботи:

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

де: $Т_р$ – кількість годин обслуговування кожного комп'ютера за рік, *год.*;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 190 годин на рік до 150 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\,баз} = 190 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 25498 \text{ грн},$$

до:

$$З_{р\,нов} = 150 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 19800 \text{ грн}.$$

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

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

$$З_{ел\ баз} = З_{ел\ нов}$$

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

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

| Групи основних фондів | Норма амортизації % | Балансова вартість, грн., за варіантами | | Сума відрахувань, грн за варіантами | |
|---|---|---|---|---|---|
| | | Базовий | Новий | Базовий | Новий |
| Програмна продукція | 50 | – | 1151 | – | 575,5 |
| Всього відрахувань | - | – | 1151 | – | 575,5 |

## 7.8 Визначення економічної ефективності програмної продукції

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

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

$$E_в = (Ц_п - С_п) \cdot N_е - \sum_{i=1}^{m} E_{р_m} \cdot К_{р_m}, \quad (7.25)$$

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

$$Ев = (959 - 639) \cdot 320 - (0,05 \cdot 2288000 + 0,5 \cdot 185654 + \\ + 0,25 \cdot 49499 + 0,2 \cdot 97500 + 0,1 \cdot 32000) \cdot 1/12 = 82208\ грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$Т_в = \frac{К_р^*}{(Ц_п - С_п) \cdot N_е}, \quad (7.26)$$

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

$$T_в = \frac{364653}{(959-639)\cdot 320\cdot 12/1} = 0{,}3 \ року .$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

| Найменування показників | Одиниця виміру | Величина |
|---|---|---|
| 1. Кількість екземплярів програми | Прим. | 320 |
| 2. Повна собівартість розробленої програми | Грн. | 639 |
| 3. Ціна розробленої програми | Грн. | 959 |
| 4. Плановий прибуток від реалізації розробленої програми | Грн. | 320 |
| 5. Рентабельність програмної продукції | % | 50 |
| 6. Об’єм додаткових капітальних вкладень у виробника програмної продукції | Грн. | 2652653 |
| 7. Загальний прибуток від реалізації програмної продукції | Грн. | 102400 |
| 8. Величина економічного ефекту при виготовлені програмної продукції | Грн. | 82208 |
| 9. Період окупності додаткових капітальних вкладень у виробника програмної продукції | Років. | 0,3 |
| 10.Об’єм додаткових капітальних вкладень у споживача програмної продукції | Грн. | 1151 |
| 11.Величина економічного ефекту у користувача програмної продукції | Грн. | 4547 |
| 12.Період окупності додаткових капітальних вкладень у користувача програмної продукції | Років | 0,22 |

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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

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

$$E_{сп} = (25498\text{-}20376)\text{-}0{,}5 \cdot 1151 = 4547 \text{ грн.}$$

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

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

$$T_{сп} = \frac{1151}{(25498\text{-}20376)} = 0{,}22 \ року .$$

**7.9 Висновки**

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

# 8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

## 8.1 Вступ

Охорона праці – система збереження життя і здоров'я працівників у процесі трудової діяльності, що включає правові, соціально-економічні, організаційні, технічні, санітарно-гігієнічні, лікувально-профілактичні, реабілітаційні та інші заходи.

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

– Інструкцій з охорони праці по кожній професії і загальні.

– Положення про охорону праці.

– Накази з охорони праці.

– Журнали реєстрації та інструктажу.

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

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

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

з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [2].

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової діяльності. Їх праця стала більш інтенсивною, напруженою і вимагає значних витрат розумової, емоційної і фізичної енергії. Це призвело до необхідності у знаходженні комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку. Охорона здоров'я робітників, забезпечення безпеки умов праці, ліквідація та профілактика професійних захворювань і виробничого травматизму складає одну з головних турбот людського суспільства.

### 8.2 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов’язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об’єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Передбачено також створення підрозділу добровільної пожежної охорони та пожежно-рятувальної команди в його складі.

Встановлений режим включає порядки з описом місць спеціального призначення та правила їх користування та утримання, наприклад:

– евакуаційних шляхів;

– так званих «курилок»;

– місць складування продукції та сировини;

– стоянки транспорту.

Також встановлюється порядок роботи та технічного обслуговування:

– вентиляційного устаткування;

– засобів пожежогасіння і захисту від загорянь;

– нагрівальних приладів;

– електрообладнання.

Розробляються і впроваджуються правила роботи з відкритим вогнем і горючими матеріалами. Створюються графіки проходження інструктажів з пожежної безпеки співробітників, а також порядок і терміни перевірок знань пожежно-технічного мінімуму, в тому числі, тих працівників, які відповідальні за цю ділянку роботи на підприємстві. При цьому можуть передбачатися внутрішні лекції, семінари, тренінги та практичні заняття на підприємстві, а також зовнішні – на базі спеціалізованих навчальних центрів з професійними викладачами. Важливою складовою протипожежного режиму на будь-якому об’єкті є розробка і впровадження порядку дій при виникненні пожежі. Неодмінно має бути план евакуації, описано, як повинні відключатися електроустановки, що і в якій послідовності необхідно робити співробітникам. Відповідно, для кожного об’єкта, кожного приміщення (крім коридорів, санвузлів, басейнів і подібних приміщень), окремих видів робіт складаються інструкції, за якими повинен працювати персонал, залучений на певних ділянках і в виконанні окремих видів робіт. За інструкціями проводиться навчання (інструктаж) персоналу з подальшим контролем знань. Детально про те, як розробити протипожежний режим, прописати порядки та інструкції, пояснюють на тематичних курсах і семінарах. [4]

## 8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців it-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців it-індустрії і стандартів підприємств, центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів. Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців it-індустрії.

Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Пропозиції щодо підвищення працездатності it-фахівців, розіб'ємо на декілька категорій:

Середовище і розпорядок праці. Для мінімізації негативних ефектів, що пов'язані з перевтомленням it-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги. Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють it-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження it-фахівців, і подальшої неможливості ними виконувати свої функції. Також, за можливості, нами пропонується введення практики віддаленої праці it-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

Фізичні і психоемоційні чинники. Першим і найважливішим чинником, що впливає на працездатність it-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку. Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві it-галузі. Тому нами пропонується закупівля тільки меблів , які пошли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, тімбілдінг, спортивні змагання і естафети. Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

### 8.4 Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

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

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який

повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга) [9].

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

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

### 8.5 Розрахункова частина

Для захисного штучного заземлення будемо застосовувати вертикальні електроди з сталевого прокату круглого перерізу діаметром 35 мм., довжиною *L=2 м.,* та горизонтальний електрод – металева полоса з перетином 35·4 мм. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – по контуру (прямокутником).

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина (питомий опір $\rho_2 = 40$ *Ом·м)*. Умовна товщина верхнього шару грунта: *H=0,6 м*. Відстань між вертикальними заземлювачами (електродами) *A=3 м*. Глибина закладення горизонтального контура заземлення *t=0,75 м*. Опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

**Розрахунок**

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,75+2/2=1,75 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi\rho = 1,36 \cdot 40 = 54,5 \text{ Ом·м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті [10];

$\rho_2 = 40$ *Ом·м.* – табличне значення питомого опору нижнього шару грунта (глина) [10].

Діаметр вертикального електрода (задан):

$$D_в = 35 \text{ мм.} = 0,035 \text{ м.}$$

Відношення $A/L=3/2=1,5$.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [10]:

$$R_O=0,366(\rho/L)[\lg(2L/D_в)+(1/2)\lg((4T+L)/(4T-L))]=$$

$$=0,366(54,5/2)[(\lg(2\cdot 2/0,035)+(1/2)\lg((4\cdot 1,75+2)/(4\cdot 1,75-2))]=$$

$$=21,7 \text{ Ом.}$$

Визначаємо коефіцієнт екранування вертикальних електродів $K_{ев}=0,53$ при орієнтовній кількості вертикальних електродів, яке дорівнює 5 [10].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без врахування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом*:

$$N=R_O/(K_{ев} R_{ЗН})= 21,7/(0,53\cdot 4)= 10,2 \approx 10 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси:

$$L_П = 1,05\cdot A\cdot N = 1,05\cdot 3\cdot 10 = 32,3 \approx 32 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [10]:

$$R_П = 0,366(\rho\cdot K_П/L_П)\lg(2(L_П\cdot L_П)/(B\cdot t))=$$

$$=0,366(40\cdot 5/40)\cdot\lg((2\cdot 40^2)/(0,035\cdot 0,75))=11,14 \text{ Ом.}$$

де $K_П=5$ – табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної кліматичної зони для з'єднуючої полоси [10]:

$B = 35$ *мм.* $= 0,035$ *м.* - ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [10]:

$$R =(R_О \cdot R_П)/( R_О \cdot \eta_П + N \cdot R_П \cdot Кев)=$$

$$=(21,7 \cdot 11,14)/(21,7 \cdot 0,55 + 10 \cdot 11,14 \cdot 0,53)=3,4 \text{ Ом.}$$

де $\eta_П = 0,55$ – табличне значення коефіцієнта екранування з'єднуючої полоси [10].

Умова $R \leq R_{ЗН}$ виконується ( $3,4 \leq 4$).

Так як при *10* вертикальних електродах $R$ суттєво більше $R_{ЗН}$, зменшимо кількість вертикальних електродів $N$ до *9* і виконаємо перерахунок. У результаті остаточно отримали: $R = 3,9$ *Ом. при* кількості вертикальних електродів $N=9$.

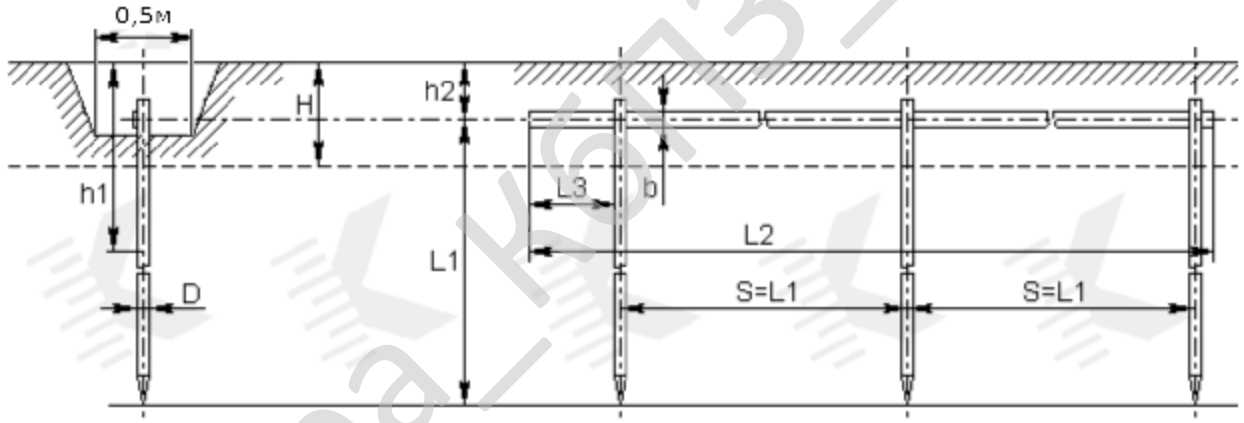


Рисунок 8.1 – Схема штучного заземлення

### 8.6 Висновки до розділу

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

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд питань пожежної безпеки, небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів о питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп'ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

# 9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи хмарних сервісів з використанням ЦСК.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів хмарних сервісів з використанням ЦСК.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем хмарних сервісів з використанням ЦСК.

– Досліджена система хмарних сервісів з використанням ЦСК.

– На основі отриманих результатів досліджень створена програмна реалізація системи хмарних сервісів з використанням ЦСК.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання хмарних сервісів з використанням ЦСК.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм UMAC.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 4547 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,22 роки.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Ковальчук В.М. Дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК // Збірник праць молодих науковців ЦНТУ. – Вип. 13. – Кропивницький: ЦНТУ, 2022.

2. 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, pp. 1-12. **(Scopus).**

3. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». *Sensors (Basel, Switzerland)* Volume 22, Issue 16, 6223, 2022. (**Scopus**).

4. 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. **(Scopus).**

5. 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. **(Scopus).**

6. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». *SN Computer Science,* Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w **(Scopus).**

7. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». *2021 IEEE 8th*

*International Conference on Problems of Infocommunications, Science and Technology (PIC S&T),* 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143 **(Scopus).**

8. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». *CEUR Workshop Proceedings* Volume 3101, 2021, Pages 192-207. **(Scopus).**

9. 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.* **(Scopus)**.

10. 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. **(Scopus)**.

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

12. Smirnov O.A., 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. **(Scopus)**.

13. 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. **(Scopus)**.

14. 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. **(Scopus)**.

15. 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. **(Scopus)**.

16. 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. **(Scopus)**.

17. 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. **(Scopus**).

18. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379. **(Scopus)**.

19. 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. **(Scopus)**.

20. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645. **(Scopus)**.

21. 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., **(Scopus)**.

22. 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. **(Scopus)**.

23. 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. **(Scopus)**.

24. 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. **(Scopus)**.

25. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629. **(Scopus)**.

26. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884. **(Scopus)**.

27. Smirnov, O., Kuznetsov, A., Prokopovych-Tkachenko, D. «Hiding Data in Images Using a Pseudo-Random Sequence». *ISCI'2020: Information Security in Critical Infrastructures.* ***Collective monograph***. Edited by Ivan D. Gorbenko, Victor A. Krasnobayev and Alexandr A. Kuznetsov. ASC Academic Publishing, USA, 2020. pp. 46-59. – ISBN: 978-1-7362833-0-1 (Hardback), ISBN: 978-1-7362833-1-8 (Ebook).

28. Smirnov, O., Kuznetsov, A., Shekhanin, K., Chepurko, I. Detecting Hidden Information in FAT. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures**. Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 412-429. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

29. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. **Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

30. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

31. Смірнов О.А., Дрєєва Г.М., «Метод генерування фрактального трафіку за допомогою моделі генератора на графі» у Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139.

32. Смирнов А.А., Коваленко А.В. Комплекс математических моделей технологии тестирования WEB-приложений. Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

33. Смирнов А.А., Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения. Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

34. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98. 2022.

35. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

36. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного

захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

37. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

38. Смирнов А., Кузнецов А., Кузнецова Т. «Шумоподобные дискретные сигналы для асинхронных систем кодового разделения радиоканалов». *Радиотехника*, № 2(205), 175–183. 2021.

39. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». *CEUR Workshop Proceedings* Volume 2732, 2020, Pages 214-227.

40. Смірнов, О.А., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю.Усік П.С., «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». *Проблеми телекомунікацій*. № 1(26). С. 83-96. 2020.

41. Смирнов А.А, Кузнецов А.А., Киян А.С., Кузнецова Е.А. «Сокрытие данных на основе адресации шумоподобных сигналов». *Всеукраїнський міжвідомчий науково-технічний збірник "Радіотехніка"* – Харьків: ХНУРЕ. – 2020. – Вип. 203. – С. 38-49.

42. Смирнов А.А., Дудан А.В., Смирнова Т.В. «Формализация структуры технологического процесса электродугового напыления». *Сборник научных трудов «Актуальные вопросы машиноведения».* Объединенный институт машиностроения Национальной Академии Наук Беларуси. №9. С. 308-312, 2020.

43. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

44. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

45. А.А. Смирнов, Т.В. Смирнова, А.Н. Дреев, А.В. Дудан. «Оптимизация технологического процесса восстановления и упрочнения поверхностей с заданными характеристиками в виде облачного сервиса». Вестник Полоцкого государственного университета. Серия В, Промышленность. Прикладные науки. Республика Беларусь – 2020. – № 3. – С. 50-61.

46. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

47. О.А. Смірнов, Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», *Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура* № 1 (11). с. 48-57, 2019.

48. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

49. Смірнов О.А., Смірнова Т.В., Солових Є.К., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

50. Смірнов О.А., Смірнова Т.В.,. Дрєєв О.М, «Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням», Системи управління, навігації та зв'язку, № 2 (54). с. 149-154, 2019.

51. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

52. Смирнов А.А., Лысенко И.А., Информационная технология проектирования тестовых наборов на основе требований к программному обеспечению, Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

53. Смірнов О.А., Мелешко Є.В., Хох В.Д., Дослідження методів аудиту систем управління інформаційною безпекою, Системи управління, навігації та зв'язку. – Випуск 1 (41). – Полтава: ПолтНТУ. – 2017. – С. 38-42.

54. Державні будівельні норми України: ДБН В.2.5-28:2018. – Режим доступу до ресурсу: https://goo.su/9AkQ

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

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

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

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

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

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

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

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

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

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

Додаток А
(обов'язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи хмарних сервісів з використанням ЦСК.

## 2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 18-13 від 17.08.2022 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК.

## 4 Джерела розробки

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

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи хмарних сервісів з використанням ЦСК;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

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

### 5.4 Вимоги до архітектури

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

### 5.5 Вимоги до надійності

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

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

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

### 5.8 Вимоги до інформаційної і програмної сумісності

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

#### 5.8.1 Обладнання

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

#### 5.8.2 Мова програмування

Середовище 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 Економічні вимоги

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

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

## 8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинна бути розглянута пожежна безпека.

## 9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 132 аркуші.

## 10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 25.12.2022 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Якименко Н.М.

***Дослідження та програмна реалізація системи хмарних сервісів з використанням ЦСК***

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 91

Літера: РП

Кропивницький – 2022 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>
/// Коваленко А.О.
/// </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 була реалізована магістерська
робота
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttrib
ute), false);
                // Якщо там не довільне значення атрибутів місця де була
реалізована магістерська робота, то повертається пустий рядок
                if (attributes.Length == 0)
                    return "";
                // Якщо там значення атрибутів місця де була реалізована
магістерська робота, то повертається його величина
                return ((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion
```

```
        private void textBoxDescription_TextChanged(object sender, EventArgs e)
        {

        }
    }
}
```