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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти

на тему

“Програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Рябовол Я.В.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук
_____ Смірнова Т.В.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2024 року

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

Рябоволу Ярославу Вадимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі*

2. Керівник роботи *Смірнова Тетяна Віталіївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 136-02 від 01.04.2024 року

3. Строк подання студентом роботи до захисту *23.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

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

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

7. Дата видачі завдання « 17 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

Підпис керівника
_____ Смірнова Т.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2024 р.

Підпис здобувача
_____ Рябовол Я.В.
(прізвище та ініціали)

АНОТАЦІЯ

Рябовол Я.В. Програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

Метою розробки є програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

Результат роботи – програмна реалізація системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

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

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

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

Програму розроблено в середовищі Visual C++.

Ключові слова: кібербезпека, PKI, корпоративна мережа

ABSTRACT

Riabovol Ya.V. Distributed access rights cyber security system software based on PKI technology in a corporate network. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the cyber security system of distributed access rights based on PKI technology in the corporate network.

The purpose of the development is the software of the cyber security system of distributed access rights based on the PKI technology in the corporate network.

The result of the work is the software implementation of the cyber security system of distributed access rights based on PKI technology in the corporate network.

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 a PC with Windows 10/11 OS.

The program was developed in the Visual C++ environment.

Keywords: cyber security, PKI, corporate network

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЕН	–	електронний нотаріус
ЕЦП	–	електроний цифровий підпис
СУБД	–	система управління базами даних
УЦ	–	удостовірюючий центр
CRL	–	список відкликаних сертифікатів
DVCS	–	Data Validation and Certification Server Protocols
http	–	HyperText Transfer Protocol – протокол передачі гіпертексту
HTTPS	–	розширення протоколу HTTP, підтримуюче шифрування
IMAP	–	протокол доступу до електроної пошти Інтернету
MAC	–	код автентифікації повідомлення
OCSP	–	Online Certificate Status Protocol
PKI	–	інфраструктура відкритих ключів
POP3	–	Post Office Protocol Version 3 – протокол поштового відділення
SMTP	–	простий протокол передачі пошти
SSL	–	Secure Sockets Layer – рівень захищених сокетів
SQL	–	Structured Query Language – мова структурованих запросів
TCP	–	протокол управління передачею
TLS	–	криптографічний протокол, захищаючий передачу даних
TSP	–	Time-Stamp Protocol
URL	–	єдиний показник ресурсів
XMPP	–	росширяємий протокол обміну повідомленями й інформацією про присутність

ВСТУП

Актуальність теми. Відправник і одержувач повідомлень при їхній взаємодії в комп'ютерній мережі підкоряються певним правилам по дотриманню послідовності дій між ними. Такі правила, називані протоколом, гарантують не тільки безпеку повідомлень, але й автентифікація кореспондентів. Тому вибір протоколів розподілу ключів у мережі являє собою важливу проблему.

У цей час розподіл ключів між користувачами реалізується двома способами:

– прямим обміном сеансовими ключами;

– створенням одного або декількох центрів розподілу ключів.

У зв'язку із цим можливі наступні ситуації організації обміну ключами:

– прямий обмін ключами;

– обмін через посередника;

– обмін через декілька посередників.

Як правило, процедура розподілу ключів застосовується разом із процедурою перевірки дійсності учасників обміну інформацією. При цьому можливі варіанти протоколів розподілу ключів із секретним і відкритим ключем, тобто на основі одноключових і двоключових методів.

Протоколи розподілу ключів з використанням одноключових методів (із секретним ключем) існують для двох ситуацій:

– прямого обміну;

– обміну через посередника.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

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

– Огляд існуючих систем розподілених прав доступу на основі технології PKI у корпоративній мережі.

– Дослідження системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

– Програмна реалізація системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі розподілених прав доступу на основі технології PKI у корпоративній мережі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення, яке розроблено у даному бакалаврському проекті, призначено для реалізації системи розподілених прав доступу на основі технології PKI у корпоративній мережі. Інфраструктура відкритих ключів PKI (Public Key Infrastructure) – це комплексна підсистема, що реалізує в рамках інформаційної системи організації функціонування набору служб безпеки по підтримці гарантованих процедур конфіденційності й строгої автентичності доступу за рахунок використання засобу шифрування й механізму цифрового підпису. Термін PKI можна розшифрувати, як сукупність апаратного й програмного забезпечення, людей і процедур, необхідних для керування, зберігання, розподілу й відкликання сертифікатів, що базується на криптографії з відкритим ключем. Таким чином, підтримуючи PKI організація встановлює в корпоративній системі довірче середовище, що гарантує авторство, дійсність і цілісність для циркулюючої в ній цифрової інформації (даних), а також забезпечує дія фактора "невідказуємості від авторства" або "невідрекаємості" при здійсненні бізнес процесів з використанням технологій електронного документообігу й комунікацій.

Стратегія створення підсистеми керування цифровими сертифікатами в автоматизованій інформаційній системі підприємства припускає використання PKI для наступних цілей:

– Забезпечення механізму строгої автентифікації в домені с використанням технології апаратних USB-ключів (смарт-карт технології)

– Організація захищеного обміну електронної пошти (шифрування переписки, використання механізму електронного цифрового підпису)

– Організація VPN (віртуальні частки мережі) для захищених з'єднань вилучених користувачів і філіальних мереж організації

– Організація захищених порталів (доступ через Web), системи розмежування доступу до сайтів, порталам і додаткам.

1.2 Область застосування

Областю застосування програмного забезпечення, розробленого під час виконання бакалаврського проектування, є корпоративна мережа. Корпоративна мережа – комунікаційна система, що належить і/або керована єдиною організацією відповідно до правил цієї організації. Корпоративна мережа відрізняється від мережі, наприклад, інтернет провайдеру тим, що правила розподілу IP адрес, роботи з інтернет ресурсами й т.д. єдині для всієї корпоративної мережі, у той час як провайдер контролює тільки магістральний сегмент мережі, дозволяючи своїм клієнтам самостійно управляти їхніми сегментами мережі, які можуть бути як частиною адресного простору провайдеру, так і бути сховані механізмом мережної трансляції адрес за одним або декількома адресами провайдеру.

Корпоративна мережа – це мультисервісна мережа передачі даних, що працює під єдиним керуванням і призначена для задоволення власних виробничих потреб компанії й організації. Корпоративна інформація – це інформація, розголошення або несанкціонована зміна якої може привести до величезних фінансових втрат.

Тому корпоративна мережа – закрита структура з досить високим ступенем захисту, доступ до якої ззовні заборонений взагалі або строго обмежений, а доступ до інформації усередині її розмежований з використанням адміністративних і технічних методів.

Для забезпечення захисту даних у корпоративних мережах можуть використовуватися різні організаційно-технічні методи (виділення

відповідальних фахівців, застосування списків контролю доступу, використання VPN і т.п.). Їхня сукупність називається комплексною системою захисту інформації.

Корпоративна мережа – база для послуг:

– корпоративна IP – телефонія;

– корпоративна електронна пошта;

– розподілені сховища даних;

– відеоконференцзв'язок;

– інші телекомунікаційні послуги.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Admin-PKI

Програма « Admin-PKI» забезпечує:

– генерацію секретних ключових носіїв і парних (секретних і відкритий) ключів для російських і закордонних криптографічних алгоритмів: ДСТ 34.10-2001 (у форматі СКЗІ Крипто-КОМ 3.1, ДСТ 34.10-94 (у форматі СКЗІ Крипто-КОМ 3.1 і більше ранньої версії СКЗІ Крипто-КОМ 3.0), RSA, DSA (у форматі PKCS#8);

– генерацію запитів сертифікатів у форматі PKCS#10;

– можливість апаратної підтримки датчика випадкових чисел (ДСЧ) при наявності програмно-апаратних комплексів захисту від несанкціонованого доступу: Акорд-амдз із контролером 4+ (розроблювач – ОКБ САПР), електронного замка Соболь (розроблювач – НИП «Информзахист»), пристроїв криптографічного захисту даних КРИПТОН-4К/16 Замок і КРИПТОН-4/PCI (розроблювач – фірма «Анкад»);

– підтримку декількох типів ключових носіїв для зберігання секретних ключів (магнітні носії (гнучкі диски), електронні ключі eToken (R2/PRO) і пристрою eToken SmartCard виробництва «Aladdin Knowledge Systems Ltd», електронні таблетки Touch-Memory виробництва «Dallas Semiconductor», USB Flash Hard Drive);

– відображення параметрів сертифіката X.509 (формату зберігання PEM або DER) у зручний для користувача вид;

– друк атрибутів запитів і сертифікатів у складі шаблона документа довільного змісту;

– пошук на ключовому носії файлу секретного ключа, що відповідає заданому сертифікату його відкритого ключа;

– можливість керування режимами генерації ключів і запитів через конфігураційні файли, формовані адміністратором системи;

– резервне копіювання ключових носіїв, що забезпечує зміна вектора стану програмного датчика випадкових чисел на резервній копії;

– можливість роботи із програмою в інтерактивному режимі (через графічний інтерфейс) і в режимі запуску модуля, що виконується, з опціями з командного рядка.

Crypto4 PKI

Crypto4 PKI це набір інструментів розроблений для керування сертифікатами стандарту X.509, списками відкликаних сертифікатів і запитами на сертифікати.

1. Генератор сертифікатів – майстер, що дозволяє генерувати сертифікати й запити на сертифікати. Генератор сертифікатів може створювати як самопідписані сертифікати, так і підписані іншими сертифікатами. Сертифікат може бути створений з нуля (коли ви вводите всю необхідну інформацію вручну) або ж використовуючи заздалегідь створений запит на сертифікат. Профілі генератора сертифікатів можна використовувати для створення декількох сертифікатів заснованих на одному шаблоні. Генератор сертифікатів дозволяє задавати алгоритми й довжину ключа, передбачувані цілі використання й місце розташування списку відкликаних сертифікатів. Ви можете зберегти сертифікат у будь-якому форматі з CER, PEM, PFX(PKCS#12) і SPC/PVK.

2. Конвертер сертифікатів дозволяє переводити сертифікат (і при необхідності секретний ключ) з одного формату в інший. Перетворення сертифікатів це розповсюджене завдання оскільки різні додатки використовують різні формати. Підтримувані формати сертифікатів: CER (стандартний DER-Формат), PEM (формат DER, представлений у кодуванні BASE64), PFX

(PKCS#12) і P7B/SPC (PKCS#7). Підтримувані формати секретних ключів: CER, PEM, PFX і PVK.

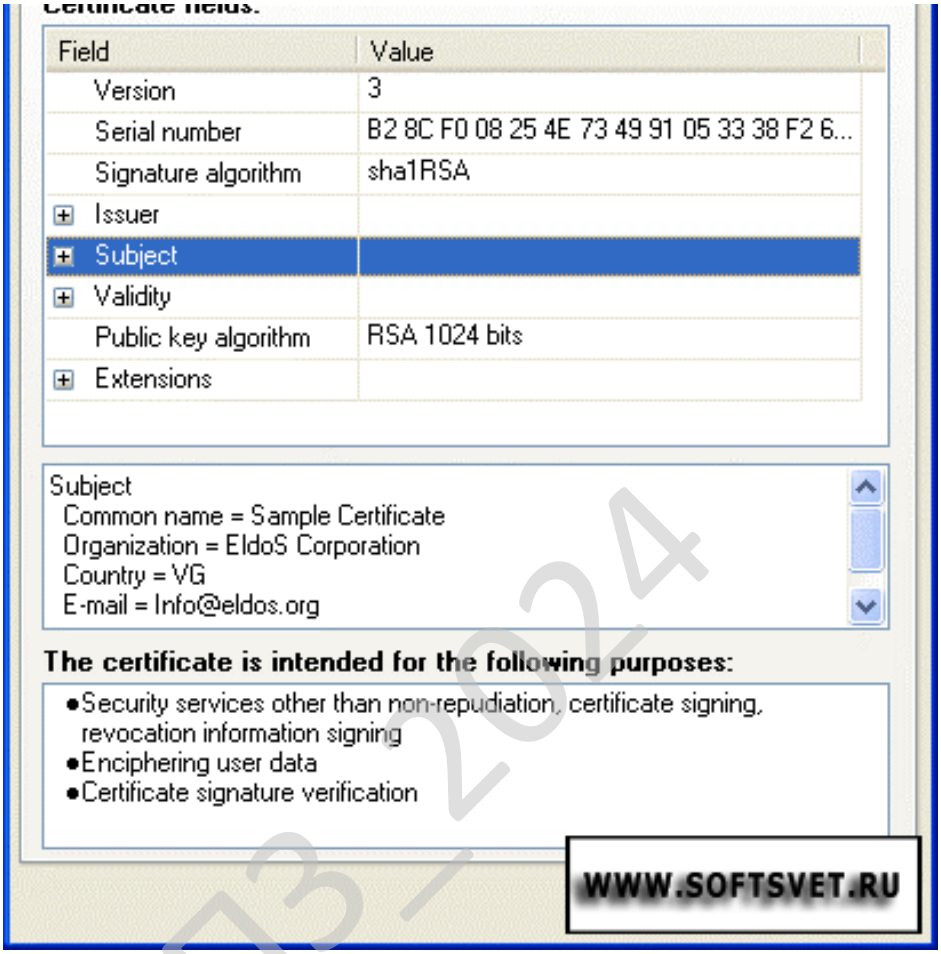

Рисунок 2.1 – Інтерфейс користувача Crypto4 PKI

Шифр-PKI

Система «Шифр-PKI» являє собою універсальну ієрархічну масштабовану систему криптографічного захисту даних і керування відкритими ключами.

«Шифр-PKI» містить набір програмних засобів побудови корпоративної інфраструктури відкритих ключів (Public Key Infrastructure – PKI), що забезпечує довірчі взаємини між кореспондентами в процесі обміну даними.

Основою довірчих відносин є цифрові сертифікати. Сертифікат однозначно зв'язує відкритий ключ кореспондента з його ім'ям і видається центром сертифікації ключів. Наявність сертифіката, що належить

кореспондентові, дозволяє вести з ним шифрований обмін даними, перевіряти електронні цифрові підписи прийнятих від нього повідомлень і здійснювати строгу автентифікацію джерела їхньої передачі.

«Шифр-PKI» дозволяє створити інфраструктуру відкритих ключів, що включає наступні елементи:

– головний центр сертифікації ключів (ГЦСК), що володіє самопідписаним (рутовим) сертифікатом і здійснюючою видачею цифрових сертифікатів для регіональних (підлеглих) центрів сертифікації;

– регіональні центри сертифікації ключів (РЦСК), що володіють сертифікатом ГЦСК і здійснюють видачу персональних цифрових сертифікатів користувачам регіонів;

– центри реєстрації (ЦР), що здійснюють передачу запитів на сертифікацію в РЦСК і прийом сертифікатів для користувачів, територіально вилучених від місця розташування РЦСК;

– засоби генерації користувальницьких ключів і формування запитів на сертифікацію;

– виконавчі програмні комплекси (бібліотеки), що виконують шифрування й цифровий підпис даних, які можна використовувати в різних автоматизованих системах.

Залежно від потреб замовника на базі засобів системи «Шифр-PKI» можуть бути створені підсистеми керування ключами й сертифікатами різних конфігурацій:

– трирівнева ієрархічна, що включає головний і трохи регіональних (підлеглих головному) центрів сертифікації ключів, у кожному регіоні – кілька центрів реєстрації (підлеглих своєму регіональному центру);

– дворівнева ієрархічна, що включає один центр сертифікації ключів і кілька центрів реєстрації;

– проста конфігурація, що знайшла широке застосування в системах типу Клієнт-Банк, що складається з одного центра сертифікації ключів.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Для реалізації програми мною була використана мова програмування Visual C++. У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка застосунків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка застосунків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки застосунків Windows дозволяють автоматизувати процес створення застосунка. Для цього використовуються генератори застосунків. Програміст відповідає на питання генератора застосунків і визначає властивості застосунка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор застосунків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої застосунки. Подібні засоби автоматизованого створення застосунків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики застосунка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні застосунки, а також застосунки, що не мають головного вікна, – замість нього

використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину застосунка програмістові прийдеться розробляти самостійно. Вихідний текст застосунка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон застосунка – це вже половина всієї роботи. Вихідні тексти застосунків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти застосунків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої застосунки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-застосунки на базі об’єктно-орієнтованого підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

Одною з основних переваг роботи з MFC є можливість багаторазового використання того самого коду. В зв’язку з тим, що бібліотека містить багато елементів, загальних для всіх Windows-застосунків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою

об’єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв’язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

Оскільки MFC являє собою набір класів, написаних мовою C++, тому програми, написані з використанням MFC, повинна бути в той же час програмами на C++. Для цього необхідно володіти відповідними знаннями. Для початку необхідно вміти створювати власні класи, розуміти принципи спадкування й уміти перевизначати віртуальні функції. Хоча програми, що використовують бібліотеку MFC, звичайно не містять занадто специфічних елементів з арсеналу C++, для їхнього написання проте потрібні солідні знання в даній області.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Інфраструктура відкритих ключів (англ. PKI – Public Key Infrastructure) – набір засобів (технічних, матеріальних, людських і т.д.), розподілених служб і компонентів, у сукупності використовуваних для підтримки криптозадач на основі закритого й відкритого ключів.

В основі PKI лежить використання криптографічної системи з відкритим ключем і кілька основних принципів:

– закритий ключ відомий тільки його власникові;

– удостовірюючий центр, створює сертифікат відкритого ключа, у такий спосіб засвідчуючи цей ключ;

– ніхто не довіряє один одному, але всі довіряють центру, що засвідчує;

– удостовірюючий центр, підтверджує або спростовує приналежність відкритого ключа заданій особі, що володіє відповідним закритим ключем.

Фактично, PKI являє собою систему, основним компонентом якої є удостовірюючий центр, і користувачі, взаємодіючі між собою за допомогою

Об'єкти PKI

PKI реалізується в моделі клієнт-сервер, тобто перевірка якої-небудь інформації, надаваною інфраструктурою може відбуватися тільки з ініціативи клієнта.

Основні компоненти PKI

Удостовірюючий центр (УЦ) є основною структурою, що формує цифрові сертифікати підлеглих центрів сертифікації й кінцевих користувачів. УЦ є головним керуючим компонентом PKI:

– він є довіреною третьою стороною (trusted third party);

– це сервер, що здійснює керування сертифікатами.

Сертифікат відкритого ключа (найчастіше просто сертифікат) – це дані користувача і його відкритий ключ, скріплені підписом центра, що засвідчує. Випускаючи сертифікат відкритого ключа, що засвідчує центр тим самим підтверджує, що особа, пойменована в сертифікаті, володіє секретним ключем, що відповідає цьому відкритому ключу.

Реєстраційний центр (РЦ) – необов'язковий компонент системи, призначений для реєстрації користувачів. Для цих цілей РЦ звичайно надає web-інтерфейс. Удостовірюючий центр, довіряє реєстраційному центру перевірку інформації про суб'єкта. Реєстраційний центр, перевіривши правильність інформації, підписує її своїм ключем і передає центру, що засвідчує, що, перевіривши ключ реєстраційного центра, виписує сертифікат. Один реєстраційний центр може працювати з декількома центрами, що засвідчують (тобто складатися в декількох PKI), один удостовірюючий центр, може працювати з декількома реєстраційними центрами. Іноді, що засвідчує центр виконує функції реєстраційного центра.

Репозиторій – сховище, що містить сертифікати й списки відкликаних сертифікатів (СВС) і, що служить для поширення цих об'єктів серед користувачів. У Законі «Про електронний підпис» він називається реєстр сертифікатів ключів підписів.

Архів сертифікатів – сховище всіх виданих коли-або сертифікатів (включаючи сертифікати зі строком, що закінчився, дії). Архів використовується для перевірки дійсності електронного підпису, який засвідчувалися документи.

Центр запитів – необов'язковий компонент системи, де кінцеві користувачі можуть запросити або відкликати сертифікат.

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

Основні завдання

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

– забезпечення конфіденційності інформації;

– забезпечення цілісності інформації;

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

– забезпечення можливості підтвердження зроблених користувачами дій з інформацією (невідмовляємість, або апелюємість – англ. non-repudiation).

PKI прямо не реалізує авторизацію, довіру, іменування суб'єктів криптографії, захист інформації або ліній зв'язку, але може використовуватися як одна зі складових при їхній реалізації.

Основна ідея

Завданням PKI є визначення політики випуску цифрових сертифікатів, видача їх і анулювання, зберігання інформації, необхідної для наступної перевірки правильності сертифікатів. У число додатків, що підтримують PKI, входять: захищена електронна пошта, протоколи платежів, електронні чеки, електронний обмін інформацією, захист даних у мережах із протоколом IP, електронні форми й документи з електронним цифровим підписом (ЕЦП).

Діяльність інфраструктури керування відкритими ключами здійснюється на основі регламенту системи. Інфраструктура відкритих ключів ґрунтується на використанні принципів криптографічної системи з відкритим ключем. Інфраструктура керування відкритими ключами складається із центра сертифікації (удостовірюючого центра – УЦ), кінцевих користувачів, і опціональних компонентів: центра реєстрації й мережного довідника.

PKI оперує в роботі сертифікатами. Сертифікат – це електронний документ, що містить електронний ключ користувача, – відкритий або ж ключову пари (keypair), – інформацію про користувача, якому належить сертифікат, що засвідчує підпис центра видачі сертифікатів (УЦ) і інформацію про термін дії сертифіката.

Для того, щоб клієнт міг працювати із центром, що засвідчує, необхідно включити центр у список довірених. Після включення в цей список, будь-який

сертифікат, виданий довіреним центром, уважається достовірним, а його власник – гідним довіри.

Удостовірюючий центр, також публікує й списки відкликаних сертифікатів (Certificate Revocation List/CRL), які можуть використовувати клієнти інфраструктури відкритого ключа, коли вирішують питання про довіру сертифікату користувача й/або комп'ютера.

Ключова пара – це набір, що складається із двох ключів: закритого ключа (private key) і відкритого ключа (public key). Ці ключі створюються разом, є комплементарними по відношенню друг до друга (те, що зашифровано за допомогою відкритого ключа можна розшифрувати, тільки маючи закритий ключ, а підпис зроблену за допомогою закритого ключа можна перевірити використовуючи відкритий ключ).

Створюється пара ключів або центром видачі сертифікатів (удостовірюючим центром), за запитом користувача, або ж самим користувачем за допомогою спеціального програмного забезпечення. Користувач робить запит на сертифікат, після чого, після процедури ідентифікації користувача, центр видає йому сертифікат зі своїм підписом. Цей підпис свідчить про те, що даний сертифікат виданий саме цим центром видачі сертифікатів і ніким іншим.

Закритий ключ використовується для підпису даних, відкритий ключ у свою чергу використовується для шифрування даних. Відкритий ключ відомий всім, а закритий ключ зберігається в таємниці. Власник закритого ключа завжди зберігає його в захищеному сховищі й ні за яких умов не повинен допустити того, щоб цей ключ став відомим зловмисникам або іншим користувачам. Якщо ж закритий ключ усе таки стане відомий зловмисникам, то він уважається скомпрометованим і повинен бути відкликаний і замінений. Тільки власник закритого ключа може підписати дані, а також розшифрувати дані, які були зашифровані відкритим ключем, що відповідає закритому ключу власника. Підпис на даних або листі гарантує авторство отриманої інформації й те, що інформація в процесі передачі не піддалася змінам. Підпис двійкового коду

гарантує, що дане програмне забезпечення дійсно зроблене зазначеною компанією й не містить шкідливого коду, якщо компанія це декларує.

Розберемо докладніше наступні моменти:

– У чому полягає робота УЦ.

– Як відбувається видача сертифіката, обмін відкритими ключами і як зрозуміти, що відкритий ключ, що ми маємо, не фальшивий.

– Які бувають PKI.

УЦ і його робота

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

Процес роботи із сертифікатами

Для того щоб одержати сертифікат, потрібно знайти який-небудь УЦ в інтернеті (альтернативним рішенням є використання ПЗ PGP або ньому подібних), після чого виписати сертифікат і встановити його собі в систему. Звичайно цей процес відбувається автоматично. Після установки сертифіката його можна буде побачити в себе в сховище особистих сертифікатів. Для того щоб переглянути його властивості, досить просто відкрити його. (Для операційних систем сімейства Windows: Пуск -> Виконати -> certmgr.msc -> OK). У властивостях можна побачити час дії сертифіката, ким він був виданий, кому був виданий, його унікальний номер та інші властивості. Після одержання сертифікатів двома або більше користувачами від одного УЦ, відбувається організація найпростішої по архітектурі PKI. PKI – з одиночним УЦ. Користувачі, зберігши сертифікати у файл обмінюються ними (у такий спосіб відбувається обмін відкритими ключами) і починають захищену переписку. Перевірка дійсності отриманого відкритого ключа проводиться по електронному відбитку цього ключа. У найпростішому випадку досить подзвонити колезі який

вислав відкритий ключ і звірити з ним електронний відбиток ключа. Якщо він збігся – можна сміло починати захищену переписку, якщо немає – обмінятися ключами ще раз.

Архітектури PKI

В основному виділяють 5 видів архітектур PKI, це:

– проста PKI (одиночний УЦ);

– ієрархічна PKI;

– мережна PKI;

– крос-сертифіковані корпоративні PKI.

– архітектура мостового УЦ

В основному PKI діляться на різні архітектури по наступних ознаках:

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

– складність перевірки шляхи сертифікації;

– наслідку видачі зловмисника себе за УЦ.

Розглянемо більш докладно кожну з архітектур PKI окремо.

Проста PKI

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

Ієрархічна PKI

Ієрархічна структура – це найбільше що часто зустрічається архітектура PKI. У цьому випадку на чолі всієї структури коштує один Головний УЦ, якому всі довіряють і йому підкоряються нижчестоящі УЦ. Крім цього головного УЦ у структурі присутні ще не один УЦ, що підкоряється вищестоящому, котрому у свою чергу приписані які-небудь користувачі або нижчестоящі УЦ. Часний приклад ієрархічної PKI – корпоративна PKI. Наприклад якщо в нас є одна більша фірма, у якої в підпорядкуванні безліч філій по всій країні. У головному

будинку фірми є головний УЦ і в кожній філії є УЦ, що підкоряється головному. В ієрархічної PKI, навіть якщо зловмисник видав себе за який – або УЦ, мережа продовжує працювати без нього, а коли він відновлює нормальну працездатність – він просто знову включається в структуру.

Мережна PKI

Мережна архітектура PKI будується як мережа довіри, численні УЦ якої надають PKI-сервіси й зв'язані одноранговими, тобто рівноправними, відносинами. Але в цьому випадку немає одного головного УЦ, якому всі довіряють. У цій архітектурі всі УЦ довіряють поруч вартим УЦ, а кожний користувач довіряє тільки тому УЦ, у якого виписав сертифікат. УЦ випускають сертифікати друг для друга; пара сертифікатів описує двосторонні відносини довіри. У дану архітектуру PKI легко додається новий УЦ, для цього йому потрібно обмінятися сертифікатами, принаймні, з одним УЦ, що вже входить у мережу. У даній архітектурі найбільш складна побудова ланцюжка сертифікації.

Мережні PKI мають велику гнучкість, тому що мають численні пункти довіри. Компрометація одного УЦ не відбивається на мережний PKI у цілому: УЦ які випустили сертифікати для скомпрометованого УЦ, просто анулюють їх, тим самим видаляючи з інфраструктури ненадійний УЦ. У результаті не порушується робота користувачів, пов'язаних з іншими центрами, що засвідчують, – вони як і раніше можуть покладатися на надійні пункти довіри й захищено зв'язуватися з іншими користувачами своєї PKI. Компрометація мережного PKI приводить або до того, що звертається робота одного УЦ разом з його співтовариством користувачів, або, якщо стали ненадійними кілька УЦ, до того, що PKI розпадається на кілька менших інфраструктур. Відновлення після компрометації мережний PKI відбувається простіше, ніж ієрархічної, насамперед, тому що компрометація зачіпає менше користувачів.

Побудувати шлях сертифікації в мережі досить складно, оскільки цей процес не детермінований і є численні варіанти формування ланцюга сертифікатів. Одні з них приводять до побудови правильного шляху, інші –

заводять у тупик. Із цієї причини валідація шляху сертифікації часто виконується одночасно з його побудовою, частиною цього процесу є видалення невірних галузей. Для побудови правильного шляху використовується кілька додаткових полів сертифікатів.

Архітектура крос-сертифікованої корпоративної PKI

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

Архітектура мостового УЦ

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

Впровадження PKI

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

Етап 1. Аналіз вимог до системи.

Етап 2. Визначення архітектури.

Етап 3. Визначення регламенту.

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

Етап 5. Інтеграція.

Етап 6. Розгортання.

Етап 7. Експлуатація.

Авторизація

Сертифікати можуть використовуватися для підтвердження особистості користувача й завдання повноважень, якими він наділений. У числі повноважень суб'єкта сертифіката може бути, наприклад, право переглядати інформацію або дозвіл вносити зміни в матеріал, представлений на web-сервері.

Термінологія PKI

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

– Сертифікат – електронний документ, що містить електронний ключ користувача, інформацію про користувача, що засвідчує підпис центра видачі сертифікатів і інформацію про термін дії сертифіката.

– Закритий ключ – ключ, що зберігається в безпечному сховищі, створений з використанням алгоритмів шифрування, що має свій унікальний електронний відбиток і, що використовується для одержання зашифрованих даних і підпису даних.

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

– Електронний відбиток (fingerprint) – це інформація за допомогою якої можна перевірити, чи є отриманий відкритий ключ саме тим, що був відісланий відправником. Електронні відбитки відкритого й закритого ключа однієї пари ідентичні, тому звіривши відбиток отриманого ключа (наприклад, по телефоні) з відбитком закритого ключа відправника, можна встановити відповідність відкритого ключа закритому.

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

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

Терміни, які необхідні для загального розуміння:

– Шлях довіри – ланцюжок документів, що дозволяє впевнитися, що пред'явлений сертифікат був виданий довіреним центром; останньою ланкою в цьому ланцюжку є пред'явлений сертифікат, початковим – сертифікат кореневого довіреного центра сертифікації, а проміжними – сертифікати, видані проміжним центрам сертифікації. Особливістю шляху довіри є те, що при втраті довіри до початкової ланки ланцюжка (кореневому центру сертифікації) губиться довіра до всього ланцюжка, тобто до всім виданим даним центром сертифікатам і до пред'явленого в тому числі.

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

– Кореневі центри сертифікації – центри сертифікації, яким довіряють споконвічно всі, або керуючись політикою підприємства, або через передвстановлені настроювання сховища сертифікатів, і які можуть перебувати на початку шляху довіри.

– Довірені центри сертифікації – список центрів сертифікації, яким довіряють власники сертифікатів. Щоб зробити який або центр довіреним, досить одержати від нього сертифікат і внести його в список довірених центрів.

3.2 Розробка структурної схеми

Засоби розробленої системи умовно розділені на дві частини.

Перша частина – базовий комплекс. До складу базового комплексу входять наступні елементи:

– АРМ головного центра сертифікації ключів.

– АРМ регіонального центра сертифікації ключів PKI у корпоративній мережі.

– Модуль генерації ключів PKI у корпоративній мережі.

– Модуль криптографічних функцій.

– Модуль інтерфейсу до електронної пошти.

Друга частина системи – засоби розширення функціональності наступного состава:

– Модуль-агент регіонального центра сертифікації ключів PKI у корпоративній мережі (ЦР).

– Модуль керування ключами клієнта.

– Модуль-служба доступу до довідника сертифікатів X.509.

Основа системи

Всі криптографічні перетворення в системі виконуються за допомогою сертифікованого програмного виробу, що являє собою бібліотеки функцій криптографічних перетворень.

Коротка характеристика системи

Система забезпечує:

– генерацію криптографічних ключів PKI у корпоративній мережі;

– керування сертифікатами відкритих ключів PKI у корпоративній мережі користувачів і персоналу автоматизованих систем і комплексів;

– криптографічний захист даних у розподілених автоматизованих системах і комплексах різного призначення.

Система підтримує ключі наступних типів:

– ключі цифрового підпису – секретні й відкриті ключі стандарту ДСТ 34310-95, ДСТУ4145-2002;

– ключі шифрування даних – симетричні ключі стандарту ДСТ 28147-89, DES, AES;

– ключі керування – ключі шифрування даних, вироблені відповідно до вимог протоколу PKI.

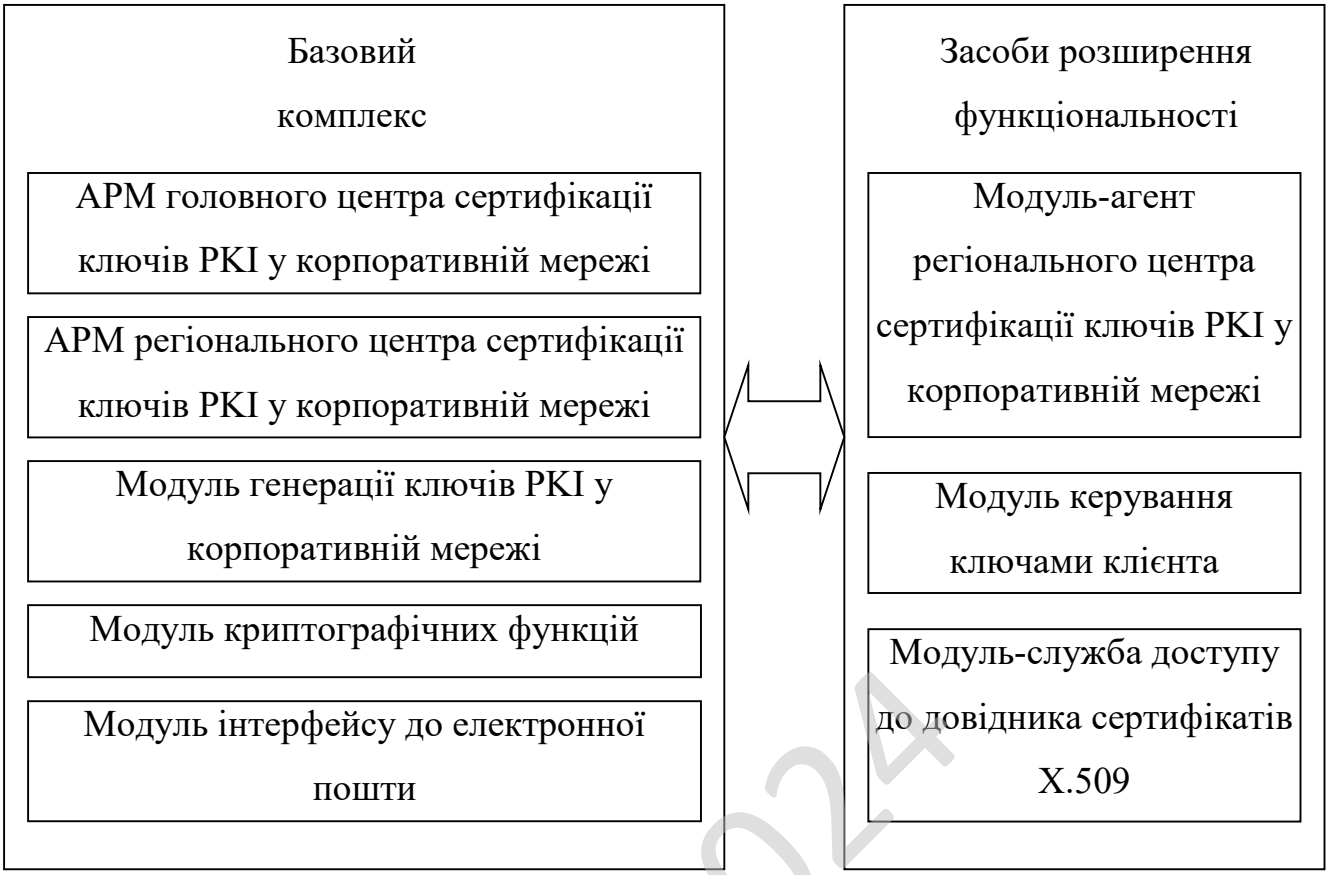

Рисунок 3.1 – Структурна схема системи

Генерацію криптографічних ключів PKI у корпоративній мережі користувачів і персоналу автоматизованих систем і комплексів здійснюють:

– Модуль генерації ключів PKI у корпоративній мережі (для виконавців центрального офісу, філії, де встановлюється регіональний центр сертифікації ключів PKI у корпоративній мережі).

– Модуль – агент регіонального центра сертифікації ключів PKI у корпоративній мережі (для територіально віддалених підрозділів, наприклад).

– Модуль зміни ключів PKI у корпоративній мережі (для територіально віддалених підрозділів).

– Модуль керування ключами клієнта (для виконавців – посадових осіб віддаленого клієнта).

Генерація криптографічних ключів PKI у корпоративній мережі в системі виконується за принципом «сам для себе». Засоби генерації ключів PKI у

корпоративній мережі дозволяють вибрати носій (HDD, Flash, Touch Memory (iButton), смарт-карта) для збереження секретних ключів PKI у корпоративній мережі. Секретні ключі зберігаються на носії в зашифрованому виді. Відкриті ключі у вигляді запиту на сертифікацію передаються в сертифікаційний центр, де виробляється сертифікат відкритих ключів PKI у корпоративній мережі користувача.

Керування ключами й сертифікатами користувачів забезпечують:

– АРМ головного центра сертифікації ключів PKI у корпоративній мережі (у центральному офісі).

– АРМ регіонального центра сертифікації ключів PKI у корпоративній мережі (у центральному офісі й філії).

– Модуль-агент регіонального центра сертифікації ключів PKI у корпоративній мережі (у територіально віддалених підрозділах,).

– Модуль керування ключами клієнта (у клієнтів банку).

– Модуль інтерфейсу до електронної пошти.

– Модуль – служба доступу до довідника сертифікатів X.509.

Засоби керування ключами дозволяють:

– підтвердити вірогідність і відправлення запитів на сертифікацію;

– виробити сертифікат відкритих ключів PKI у корпоративній мережі по отриманому запиті;

– здійснити адресне розсилання сертифікатів X.509;

– вести контроль термінів дії сертифікатів X.509 (запровадження в дію й вивід з дії сертифікатів X.509 відповідно до призначеного строку);

– відзивати сертифікати;

– формувати й розсилати списки відкликаних сертифікатів X.509;

– припиняти дії сертифікатів X.509 на основі даних списку відкликаних сертифікатів X.509;

– виконувати широкий набір сервісних функцій (відображення таблиць сертифікатів X.509, пошук сертифікатів X.509 за різними критеріями, перенос виведених з дії сертифікатів X.509 в архів, і ін.).

Доставку ключових і службових повідомлень у процесі керування сертифікатами забезпечує **Модуль інтерфейсу до електронної пошти**, що реалізує протоколи SMTP і POP3. Крім цього, даний модуль може використовуватися для розкладки ключових повідомлень по заданих каталогах, що дозволяє використовувати його при обміні файлами за протоколом FTP.

Виконавчими пристроями системи є модуль криптографічних функцій (динамічні бібліотеки для WIN32, WIN64).

Модулі криптографічних функцій реалізовані у вигляді динамічних бібліотек (*.DLL для WIN32, WIN64).

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

У процесі роботи система забезпечує наступні функціональні можливості:

– уведення секретного ключа, його розшифрування й збереження в пам'яті;

– зупинку роботи при невідповідності уведеного секретного ключа сертифікату, що зберігається в довіднику, або якщо використання цього сертифіката заборонено;

– пошук у довіднику й вибір сертифіката, необхідного для виконання криптоперетворень;

– перевірку автентичності й повноважності обраного сертифіката;

– хешування даних;

– постановку й перевірку електронного цифрового підпису;

– виробіток ключів PKI у корпоративній мережі зв'язку за протоколом PKI;

– генерацію ключів PKI у корпоративній мережі шифрування даних;

– виробіток імітовставки й шифрування даних;

– розшифрування даних і контроль їхньої цілісності шляхом перевірки імітовставки.

Рисунок 3.2 – Функціональна схема системи

Наявність механізму профілювання сертифікатів X.509

Профіль сертифіката – це однозначно ідентифікуєма, іменована сукупність загальних властивостей сертифікатів X.509 (повноваження, призначення, термін дії, місце сертифікації й т.д.), що дозволяє об'єднати сертифікати в певну групу.

Завдяки цьому, засобу системи гнучко й органічно вписуються практично в будь-які автоматизовані системи Замовника, будучи фактично незалежними від їх технологічних і функціональних властивостей.

Сертифікат X.509 v3

Сертифікат X.509 v3 визначається в такий спосіб. Для обчислення підпису дані, які повинні бути підписані, представляються з використанням ASN.1 однозначних правил подання (DER).

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate     TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue     BIT STRING
}
TBSCertificate  ::= SEQUENCE  {
    version [0] EXPLICIT Version DEFAULT v1,
    serialNumber CertificateSerialNumber,
    signature AlgorithmIdentifier,
    issuer Name,
    validity Validity,
    subject Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID [1] IMPLICIT
        UniqueIdentifier OPTIONAL,
    ---i якщо є присутнім, версія повинна
    ---i бути v2 або v3
    subjectUniqueID [2] IMPLICIT
        UniqueIdentifier OPTIONAL,
    ---i якщо є присутнім, версія повинна бути
    ---i v2 або v3
    extensions [3] EXPLICIT Extensions OPTIONAL
    ---i якщо є присутнім, версія повинна бути v3
}
Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
Validity ::= SEQUENCE {
    notBefore Time,
    notAfter  Time
}
Time ::= CHOICE {
    utcTime     UTCTime,
```

```
    generalTime GeneralizedTime
}
UniqueIdentifier ::= BIT STRING
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm        AlgorithmIdentifier,
    subjectPublicKey BIT STRING
}
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE  {
      extnID    OBJECT IDENTIFIER,
      critical  BOOLEAN DEFAULT FALSE,
      extnValue OCTET STRING
}
```

Поля сертифіката. Сертифікат є послідовність трьох обов'язкових полів: tbsCertificate, signatureAlgorithm і signatureValue.

– tbsCertificate. Поле містить імена суб'єкта й випускаючого, відкритий ключ, пов'язаний із суб'єктом, період дійсності й іншу пов'язану із цим сертифікатом інформацію. Поля докладно описані далі; tbsCertificate звичайно включають розширення, які теж будуть описані нижче.

– signatureAlgorithm. Поле signatureAlgorithm містить ідентифікатор криптографічного алгоритму, використовуваного УЦ для підписування даного сертифіката. Існують стандартні алгоритми, які повинні підтримуватися всіма реалізаціями, але конкретна реалізація може підтримувати й інші алгоритми.

Ідентифікатор алгоритму визначається наступної ASN.1-структурою:

```
AlgorithmIdentifier ::= SEQUENCE  {
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

Ідентифікатор алгоритму використовується для визначення криптографічного алгоритму. Компонент OBJECT IDENTIFIER ідентифікує алгоритм (такий як DSA з SHA-1). Компоненти поля параметрів змінюються відповідно до зазначеного алгоритму. Поле повинне містити той же самий ідентифікатор алгоритму, що й поле підпису в tbsCertificate.

– signatureValue. Поле signatureValue містить цифровий підпис, обчислений для поля tbsCertificate, записаному в DER-поданні ASN.1. Це означає, що поле tbsCertificate, представлене як ASN.1 DER, використовується як вхід у функцію підпису. Отримане значення підпису представлене як BIT STRING і включено в поле підпису. Деталі даного процесу можуть відрізнятися для кожного конкретного алгоритму підпису. Створенням даного підпису УЦ підтверджує дійсність інформації в поле tbsCertificate. Зокрема, УЦ підтверджує зв'язок між матеріалом відкритого ключа й суб'єктом сертифіката.

– TBSCertificate. Послідовність TBSCertificate містить інформацію, пов'язану із суб'єктом сертифіката й УЦ, що випустив сертифікат. Кожний TBSCertificate містить імена суб'єкта й випускаючого, відкритий ключ, пов'язаний із суб'єктом, період дійсності, номер версії й серійний номер сертифіката; деякі поля можуть (але це не обов'язково) містити унікальний ідентифікатор. Розглянемо синтаксис і семантику таких полів. TBSCertificate звичайно включає розширення. Розглянемо також найбільше часто використовувані в Internet розширення.

– Version. Дане поле описує версію подання сертифіката. Якщо використовуються розширення, то версія повинна бути 3 (значення – 2). Якщо розширення не зазначені, але UniqueIdentifier представлений, версія може бути 2 (значення – 1); але версія може бути й 3. Якщо представлені тільки базові поля, версія може бути 1 (значення в сертифікаті опущене як значення за замовчуванням); але версія може бути 2 або 3. Реалізації повинні бути готові приймати будь-яку версію сертифіката. Як мінімум конформні реалізації повинні розпізнавати версію 3 сертифікатів.

– Serial number. Серійний номер повинен бути позитивним цілим, призначуваним УЦ для кожного сертифіката. Він повинен бути унікальним для кожного сертифіката, випущеного даним УЦ. Таким чином, ім'я що выпустили й серійний номер однозначно визначають сертифікат. Cas повинні забезпечувати,

щоб серійні номери були ненегативними цілими. Уважається, що серійні номери можуть мати довжину до 20 октетів.

– Signature. Дане поле містить ідентифікатор алгоритму, використовуваного УЦ для підписування сертифіката.

– Дане поле повинне містити той же самий ідентифікатор алгоритму, що й поле signatureAlgorithm в Certificate. Зміст необов'язкового поля параметрів залежить від конкретного алгоритму.

– Issuer. Поле issuer ідентифікує того, хто підписав і випустив сертифікат. Поле issuer повинне містити непусте унікальне ім'я (DN). Ім'я визначається у відповідності з наступної ASN.1 структурою:

```
Name ::= CHOICE { RDNSequence }
RDNSequence ::= SEQUENCE OF
    RelativeDistinguishedName
RelativeDistinguishedName ::=
    SET OF AttributeTypeAndValue
AttributeTypeAndValue ::= SEQUENCE {
    type   AttributeType,
    value  AttributeValue
}
AttributeType ::= OBJECT IDENTIFIER
AttributeValue ::= ANY DEFINED BY
    AttributeType
```

Ім'я описує ієрархічне ім'я, що складається з атрибутів, таких, наприклад, як назва країни, і відповідних значень, таких як RU. Тип компонента AttributeValue визначається значенням AttributeType. Стандарт X.509 не обмежує набір типів атрибутів, які можуть з'явитися в ім'ї. Проте, стандартом рекомендується підтримувати наступні типи атрибутів в іменах випускаючі й суб'єкта:

– Країна.

– Організація.

– Організаційна одиниця.

– Позначення унікального імені.

– Назва штату або регіону.

– Загальноприйняте ім'я (наприклад, Іванов Іван).

– Серійний номер.

Додатково можуть бути присутнім деякі інші типи атрибутів в іменах випускаючі й суб'єкта, наприклад:

– Локалізація.

– Заголовок.

– По батькові.

– Призначене ім'я.

– Ініціали.

– Псевдонім.

– Спеціальна назва (наприклад, "Jr.", " 3-ій" або "IV").

Також може бути присутнім атрибут domainComponent. DNS надає собою ієрархічну систему позначення ресурсів. Даний атрибут надає зручний механізм для організацій, які хочуть використовувати унікальні DN імена паралельно зі своїми DNS-Іменами. Це не заміняє dNSName компонент альтернативного поля ім'я. Стандарт не вимагає конвертувати такі імена в DNS-Імена.

Сторона, що перевіряє, повинна обробляти поля унікального ім'я випускаючого й унікального ім'я суб'єкта для одержання ланцюжка імен при перевірці дійсності сертифікаційного шляху. Ланцюжок імен виходить у випадку відповідності унікального ім'я випускаючого в першому сертифікаті ім'я суб'єкта в сертифікаті УЦ.

Зручна виконавча частина системи

Модулі криптографічних функцій реалізовані у вигляді набору динамічних бібліотек під Win32.

Крім цього, система містить у собі Модуль ЕЦП і спрямованого шифрування файлів, що дозволяють здійснювати ЕЦП і шифрування файлів як в інтерактивному, так і в автоматичному режимі.

Повна автоматизації процедур керування сертифікатами

Весь обмін між різними компонентами системи, починаючи від відправлення запиту на сертифікат виконавця, і до одержання сертифіката й здійснюється по електронних каналах зв'язку, при цьому всі дані, передані в каналах зв'язки, захищені ЕЦП. Всі процедури по зміні ключів PKI у корпоративній мережі виконуються динамічно, без необхідності зупинки роботи автоматизований систем.

Зручний механізм настроювання правил розсилання сертифікатів X.509 і стоп-аркушів

Засоби системи дозволяють, як Розроблювачеві на етапі адаптації по вимогах Замовника, так і користувачам у процесі експлуатації, створювати й змінювати правила, по яких головний центр сертифікації ключів і/ або регіональний центр сертифікації ключів автоматично відправлять вироблений сертифікат (стоп-аркуш) заданому одержувачеві, або заданій групі одержувачів. Сертифікати (стоп-аркуші) розсилаються у вигляді ключових повідомлень.

Доставка ключових повідомлень може виконуватися Модулем інтерфейсу до електронної пошти, що входить у комплект поставки системи, або будь-якими іншими транспортними засобами Замовника.

Гнучка організація доступу до сертифікатів відкритих ключів PKI у корпоративній мережі

Виконавчі пристрої системи при виконанні криптоперетворень використовують сертифікати відкритих ключів PKI у корпоративній мережі, які зберігаються в довіднику сертифікатів X.509.

Доступ до сертифікатів у довіднику може здійснюватися двома способами:

– перший спосіб: прямій (поділюваний) доступ до файлу таблиці довідника сертифікатів X.509;

– другий спосіб: через Модуль – службу доступу до довідника сертифікатів X.509 за протоколом TCP/IP.

При цьому, як ми вже відзначали вище, відновлення довідника здійснюється динамічно без необхідності зупинки.

Зміна ключів PKI у корпоративній мережі виконавців безпосередньо на своєму робочому місці

При використанні в системі Модуля зміни ключів PKI у корпоративній мережі в кожного виконавця з'являється можливість планової зміни ключів PKI у корпоративній мережі безпосередньо на своєму робочому місці, за пропозицією модуля криптофункцій.

При цьому доставка запиту на сертифікат у регіональному центрі сертифікації ключів може здійснюватися через поділюваний каталог, якщо виконавець і регіональний центр сертифікації ключів перебувають у єдиній локальній мережі, або за протоколом TCP/IP через Службу доступу до довідника сертифікатів X.509, при віддаленому розташуванні виконавця.

Легкість роботи із системою

Система має інтуїтивно зрозумілий користувальницький інтерфейс і повністю документована. Документація розрахована на рядового користувача. Система легко встановлюється й настроюється. В експлуатації надійна й невимоглива до ресурсів.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

Після запуску розробленої програми відбувається генерація пари ключів. Один зі згенерованих ключів регіструється та видається користувачу як сертифікат, інший ключ стає секретним закритим ключем користувача.

Розглянемо процеси, що стосуються закритого ключа користувача. Після

його створення, ключ транспортується до його власника, а також відбувається його резервне копіювання. Після цього ключ ініціалізується та використовується його власником доки не закінчиться строк його дії.

Розглянемо процеси, що відносяться до життєвого циклу сертифікату.

Після його реєстрації відбувається розповсюдження сертифіката, потім він зберігається у базі даних Центру сертифікатів та зберігається там поки не закінчиться його строк дії. Користувач може використовувати свій сертифікат поки той не змінить свій статус і не припинить дію.

Рисунок 3.3 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків. Спершу відбувається виведення головного вікна ПЗ, введення параметрів з'єднання з сервером та запит здійснити з'єднання з сервером.

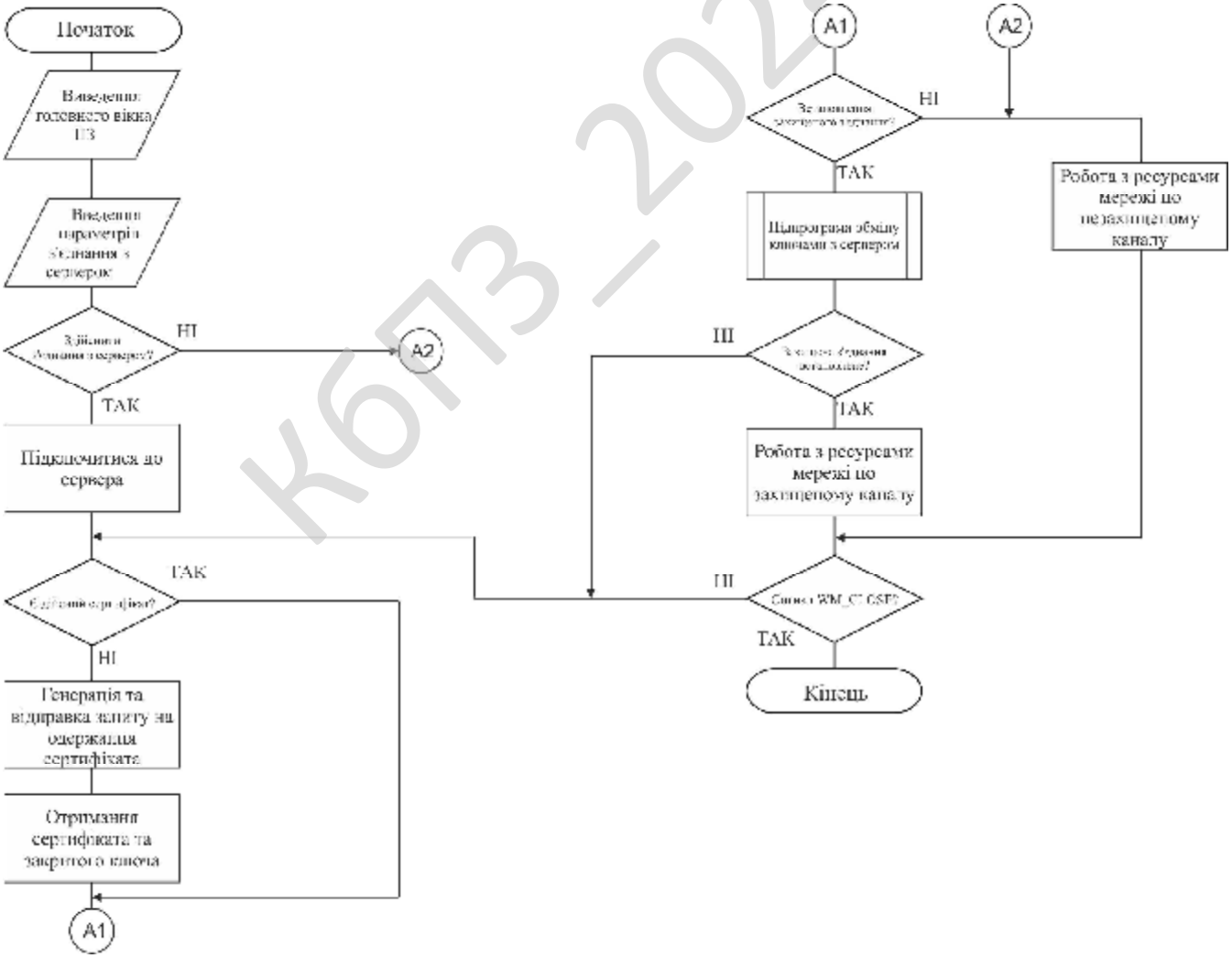

Рисунок 4.1 – Блок-схема основної програми

Після чого проходе підключення до сервера та якщо немає дійсного сертифікату проходе генерація та відправка запиту на одержання сертифіката, отримання сертифіката та закритого ключа.

Якщо є запит встановлення захищеного з'єднання проходе виклик підпрограми обміну ключами з сервером (рисунок 4.2).

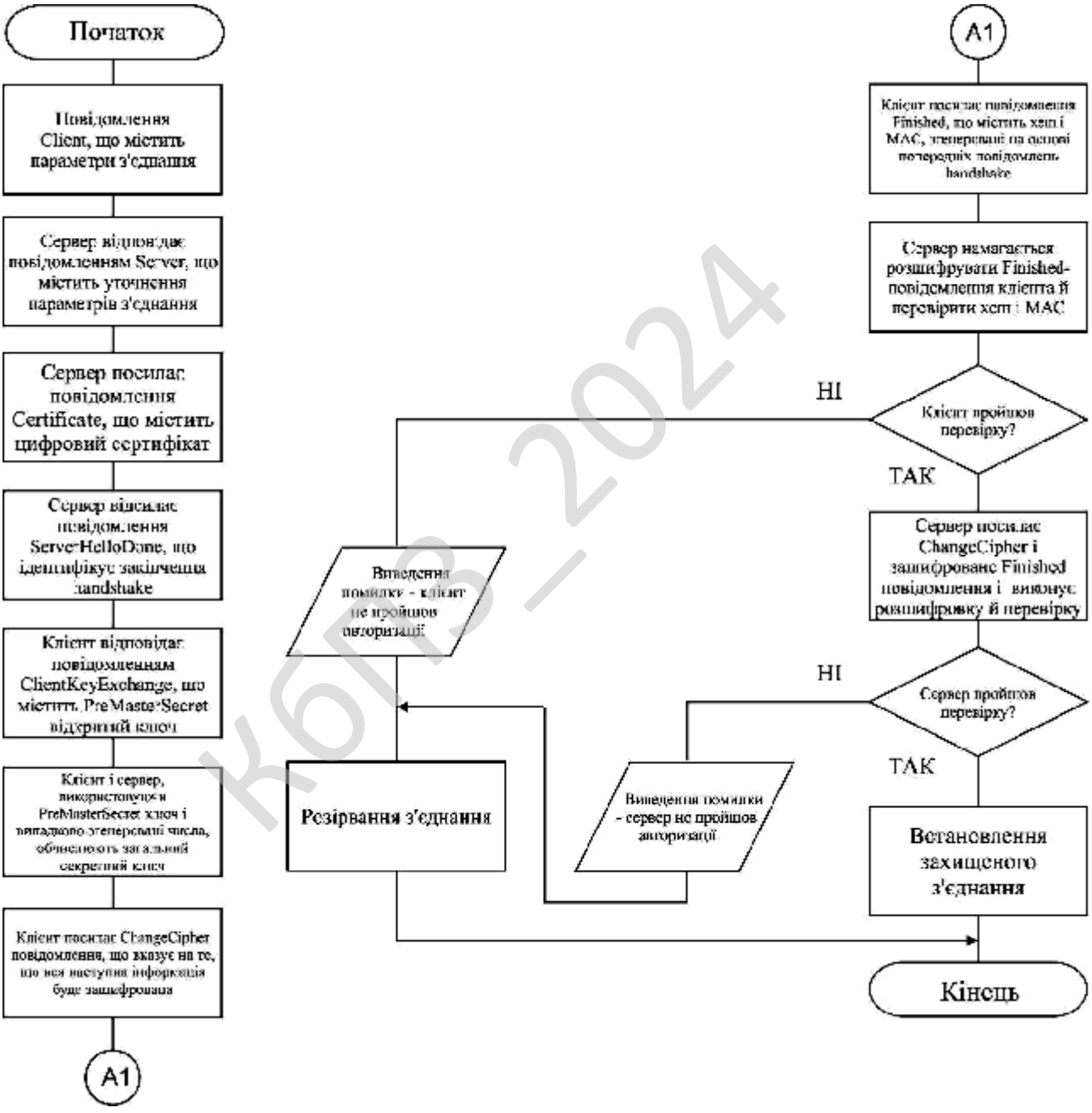

Рисунок 4.2 – Блок-схема підпрограми

У підпрограмі проходе обмін повідомленнями між клієнтом та сервером:

– повідомлення Client, що містить параметри з'єднання;

– сервер відповідає повідомленням Server, що містить уточнення параметрів з'єднання;

– сервер посилає повідомлення Certificate, що містить цифровий сертифікат сервера;

– сервер відсилає повідомлення ServerHelloDone, що ідентифікує закінчення handshake;

– клієнт відповідає повідомленням ClientKeyExchange, що містить PreMasterSecret відкритий ключ;

– клієнт і сервер, використовуючи PreMasterSecret ключ і випадково згенеровані числа, обчислюють загальний секретний ключ;

– клієнт посилає ChangeCipher повідомлення, що вказує на те, що вся наступна інформація буде зашифрована;

– клієнт посилає повідомлення Finished, що містить хеш і MAC, згенеровані на основі попередніх повідомлень handshake;

– сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити хеш і MAC.

Далі проходе запит – клієнт пройшов перевірку. Якщо так сервер посилає ChangeCipher і зашифроване Finished повідомлення і виконує розшифровку й перевірку та проводиться наступний запит – сервер пройшов перевірку.

Якщо так проводиться встановлення захищеного з'єднання та повернення до основної блок схеми.

Де проходе запит – захищене з'єднання встановлене. Якщо так проводиться робота з ресурсами мережі по захищеному каналу.

Якщо є сигнал WM_CLOSE програмне забезпечення завершується.

Процедура посилання клієнтом повідомлення Client_Hello та його обробки виглядає наступним чином:

```
void SSLConnection::PerformHandShake(Object* state)
    {
```

```
        DWORD           dwSSPIFlags;
        DWORD           dwSSPIOutFlags;
        TimeStamp       tsExpiry;
        SECURITY_STATUS scRet = S_OK;
        dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT   |
                      ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR  |
                      ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;
//
//  Ініціалізуємо повідомлення ClientHello та генеруємо токен.
//
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);
        OutBuffer.SetSecurityBufferToken(0, NULL, 0);

       IntPtr ptrServerName = System::Runtime::InteropServices::Marshal::
StringToCoTaskMemAnsi(m_ServerIP);
        scRet = m_pSecurityFunc->InitializeSecurityContextA(
                       m_phClientCreds, NULL,
                        static_cast<SEC_CHAR*>(ptrServerName.ToPointer()),
                        dwSSPIFlags,0,
                        SECURITY_NATIVE_DREP,
                        NULL, 0, m_phContext, &OutBuffer,
                        &dwSSPIOutFlags, &tsExpiry);
        System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);
        if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
        {
            DoRenegotiate(this);
        }
        else if(scRet != SEC_I_CONTINUE_NEEDED)
        {
            throw new Common::Exceptions::SSLException
       (S"InitializeSecurityContext Помилкове. Помилка: ", scRet);
        }
        m_bInHandShake = true;
// Відправлення відповіді на сервер, якщо він готовий.
        if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
        {
            bool bSent = DispatchSend(static_cast<char*>(
       OutBuffer[0].pvBuffer), OutBuffer[0].cbBuffer, state);
            if(!bSent)
            {
                throw new Common::Exceptions::SSLSendException
       (S"Відправлення помилки Сервера.");
```

```
            }
        }
    }
```

Далі сервер посилає повідомлення Certificate, що містить цифровий сертифікат сервера та повідомлення ServerHelloDone, що ідентифікує закінчення handshake.

Клієнт відповідає повідомленням ClientKeyExchange, що містить PreMasterSecret відкритий ключ.

Після цього клієнт і сервер, використовуючи PreMasterSecret ключ і випадково згенеровані числа, обчислюють спільний секретний ключ.

Вся інша інформація про ключ буде отримана із спільно секретного ключа (і згенерованих клієнтом і сервером випадкових значень).

Клієнт посилає ChangeCipherSpec повідомлення, що вказує на те, що вся наступна інформація буде зашифрована встановленим у процесі handshake алгоритмом, використовуючи спільний секретний ключ.

Клієнт посилає повідомлення Finished, що містить хеш і MAC, згенеровані на основі попередніх повідомлень handshake.

Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити хеш і MAC.

Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване.

У разі вдалої перевірки клієнта, сервер посилає ChangeCipherSpec і зашифроване Finished повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

Якщо і клієнт і сервер пройшли перевірку, то встановлюється захищене з'єднання.

Процедура ініціалізації рукопотискання (handshake) у розробленій у результаті виконання роботи програмі виглядає наступним чином:

```
void SSLConnection::InitiateHandShake(String* ipAddres, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state)
    {
```

```
        if(m_ServerIP != NULL)
// Йдемо по списку серверів
        {
            Dispose();
            Init();
//ініціалізуємо
        }
        m_ServerIP = ipAddres;
// визначаємо IP адресу сервера
        try
        {
            SetupCredentials(thumbPrint, prot);
//Встановлюємо автентифікатор
            PerformHandShake(state);
        }
        catch(Common::Exceptions::SSLException*)
        {
            Dispose();
            throw;
        }
    }
```

Процедура перегляду рукопотискань клієнтом описується наступним чином:

```
    bool SSLConnection::ClientHandshakeLoop(void* IoBuffer, int& ActualLen,
                                            SecBuffer *pExtraData, Object* state)
    {
        CAutoSecBuffer<2> InBuffer(m_pSecurityFunc, false);
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);

        TimeStamp tsExpiry;
        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT |
                            ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR |
                            ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;

        SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
        while(scRet == SEC_I_CONTINUE_NEEDED || scRet ==
SEC_I_INCOMPLETE_CREDENTIALS)
         {
//
// Встановлення вхідних буферів. Буфер 0 використовує шифрування в даних
```

```
/// отриманих з серверу. Schannel читає усі дані або тільки признаки.
// Відложені дані будуть накопичуватися у буфері  1 та надавати
// буферу тип SECBUFFER_EXTRA.
//
            InBuffer.SetSecurityBufferToken(0, IoBuffer, ActualLen);
            InBuffer.SetSecurityBufferEmpty(1);
            OutBuffer.SetSecurityBufferToken(0, NULL, 0);
            scRet = m_pSecurityFunc->InitializeSecurityContextA(m_phClientCreds,
                                              m_phContext,
                                              NULL,
                                              dwSSPIFlags,
                                              0,
                                              SECURITY_NATIVE_DREP,
                                        &InBuffer,
                                        0,
                                        NULL,
                                        &OutBuffer,
                                        &dwSSPIOutFlags,
                                        &tsExpiry);
// Якщо InitializeSecurityContext працює правильно
// (або якщо помилка припустима), відправляємо зміст вихідного
// буфера на сервер.
            if(scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED ||
              (FAILED(scRet)
               && (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR)))
            {
                if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
                {
              bool bSent =DispatchSend(static_cast<char*>
                                        (OutBuffer[0].pvBuffer),
                                        OutBuffer[0].cbBuffer, state);
                    if(!bSent)
                    {
    throw new Common::Exceptions::SSLSendException
              (S"Відправлення на сервер не відбулося.");
                    }
                }
                OutBuffer.FreeBuffer(0);
            }
            //
// Якщо InitializeSecurityContext повертає SEC_E_INCOMPLETE_MESSAGE,
//  тоді ми  читаємо наступні данні з сервера.
```

```
//
            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
//Викликаємо повідомлення для збереження змісту
// буферу у разі помилки при подальшому вводі;
            }
//
// Якщо InitializeSecurityContext повертає SEC_E_OK, тоді
// рукопотискання завершено успішно.
//
            if(scRet == SEC_E_OK)
            {
                //
// Якщо "додатковий" буфер містить дані - то це закодована
// інформація для прикладного протоколу OSI. Це повинно
// бути збережено. Данні для прикладного рівня будуть
// декодовані з  DecryptMessage.
//
                if(InBuffer[1].BufferType == SECBUFFER_EXTRA)
                {
                    pExtraData->pvBuffer = malloc(InBuffer[1].cbBuffer);
                    if(pExtraData->pvBuffer == NULL)
                    {
                        throw new OutOfMemoryException();
                    }
                    MoveMemory(pExtraData->pvBuffer,(BYTE*)IoBuffer + (ActualLen -

                                InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                    pExtraData->cbBuffer   = InBuffer[1].cbBuffer;
                    pExtraData->BufferType = SECBUFFER_TOKEN;
                }
                else
                {
                    pExtraData->pvBuffer   = NULL;
                    pExtraData->cbBuffer   = 0;
                    pExtraData->BufferType = SECBUFFER_EMPTY;
                }
//
// Для виходу зберігається у БД
//
                m_bInHandShake = false;
                if(DoServerCertVerify != NULL)
```

```
                {
                    IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemUni(m_ServerIP);
                    Common::Misc::CeriticateInfo ServCertInfo;
                    VerifyCertificate(true, m_pSecurityFunc, m_phContext,
static_cast<wchar_t*>(ptrServerName.ToPointer()), 0, &ServCertInfo);
                        DoServerCertVerify(ServCertInfo);

System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);
                    DoServerCertVerify = NULL; //заборона інших викликів під час
виконання процедури renegotiation.
                }
                if(DoHandShakeSuccess != NULL)
                {
                    DoHandShakeSuccess();
                }
                break;
            }
//
// Крапка невиправної помилки .
//
            if(FAILED(scRet))
            {
                throw new Common::Exceptions::SSLException(S"Руко потискання з
сервером помилкове. Помилка: ", scRet);
            }
//
// Якщо InitializeSecurityContext повертає SEC_I_INCOMPLETE_CREDENTIALS,
// тоді сервер автентифікує клієнта.
//
            if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
            {
                if(DoRenegotiate != NULL)
                    DoRenegotiate(this);
                SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
                continue;
            }
//
// Копіюємо любі відложені дані з «додаткового»
// буфера й виконуємо наступний раунд.
//
            if ( InBuffer[1].BufferType == SECBUFFER_EXTRA )
```

```
            {
                int temp = ActualLen;
                MoveMemory(IoBuffer,(BYTE*)IoBuffer + (ActualLen -
                          InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                ActualLen = InBuffer[1].cbBuffer;
            }
            else
            {
                ActualLen = 0;
                break;
            }
        }
        return true;
    }
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SHACAL-2, який шифрує дані 256-бітними блоками з використанням 512-бітного ключа. Допускається використання ключів менших розмірів (не менш 128 біт), які доповнюються бітовими нулями до 512 біт.

Шифруємий блок даних ділиться на 8 фрагментів по 32 біта (які позначені буквами *A*...*H*). Алгоритм виконує 64 раунду перетворень, у кожному з яких дані фрагменти обробляються в такий спосіб:

$$T = H_i + S_1(E_i) + Ch(E_i, F_i, G_i) + M_i + K_i,$$

$$H_{i+1} = G_i,$$

$$G_{i+1} = F_i,$$

$$F_{i+1} = E_i,$$

$$E_{i+1} = D_i + T,$$

$$D_{i+1} = C_i,$$

$$C_{i+1} = B_i,$$

$$B_{i+1} = A_i,$$

$$A_{i+1} = T + S_0(A_i) + Maj(A_i, B_i, C_i).$$

де T – тимчасова змінна.

Використовувані функції визначені в такий спосіб:

$$S_0(x) = (x >>> 2) \oplus (x >>> 13) \oplus (x >>> 22),$$

$$S_1(x) = (x >>> 6) \oplus (x >>> 11) \oplus (x >>> 25),$$

$$Ch(x, y, z) = (x \,\&\, y) \oplus (x' \,\&\, z),$$

$$Maj(x, y, z) = (x \,\&\, y) \oplus (x \,\&\, z) \oplus (y \,\&\, z),$$

де >>> – операція побітового циклічного зрушення вправо.

Константи, що модифікують, M_i ($i = 0…63$) наведено нижче (одна за одною від M_0 до M_{63}):

428A2F98	71374491	B5C0FBCF	E9B5DBA5
3956C25B	59F111F1	923F82A4	AB1C5ED5
D807AA98	12835B01	243185BE	550C7DC3
72BE5D74	80DEB1FE	9BDC06A7	C19BF174
E49B69C1	EFBE4786	0FC19DC6	240CA1CC
2DE92C6F	4A7484AA	5CB0A9DC	76F988DA
983E5152	A831C66D	B00327C8	BF597FC7
C6E00BF3	D5A79147	06CA6351	14292967
27B70A85	2E1B2138	4D2C6DFC	53380D13
650A7354	766A0ABB	81C2C92E	92722C85
A2BFE8A1	A81A664B	C24B8B70	C76C51A3
D192E819	D6990624	F40E3585	106AA070
19A4C116	1E376C08	2748774C	34B0BCB5
391C0CB3	4ED8AA4A	5B9CCA4F	682E6FF3
748F82EE	78A5636F	84C87814	8CC70208
90BEFFFA	A4506CEB	BEF9A3F7	C67178F2

Фрагменти розширеного ключа $K_0…K_{63}$ обчислюються в процесі процедури розширення ключа, що виконується в такий спосіб:

Етап 1. 512-бітний вихідний ключ шифрування ділиться на 16 фрагментів по 32 біта $K_0…K_{15}$...

Етап 2. Інші фрагменти розширеного ключа $K_{16}…K_{63}$ обчислюються з перших 16 фрагментів у такий спосіб:

$$K_i = O_1(K_{i-2}) + K_{i-7} + O_0(K_{i-15}) + K_{i-16},$$

де функції O_0 і O_1 визначені так:

$$O_0(x) = (x >>> 7) \oplus (x >>> 18) \oplus (x >> 3),$$

$$O_1(x) = (x >>> 17) \oplus (x >>> 19) \oplus (x >> 10),$$

де >> – операція побітового зрушення (не циклічного) вправо.

Раунди розшифрування алгоритму виконуються у зворотній послідовності:

$$T = A_{i+1} + S_0'(B_{i+1}) + Maj'(B_{i+1}, C_{i+1}, D_{i+1}) + 2,$$

$$H_i = T + S_1'(F_{i+1}) + Ch'(F_{i+1}, G_{i+1}, H_{i+1}) + M_i' + K_i' + 4,$$

$$G_i = H_{i+1},$$

$$F_i = G_{i+1},$$

$$E_i = F_{i+1},$$

$$D_i = E_{i+1} + T' + 1,$$

$$C_i = D_{i+1},$$

$$B_i = C_{i+1},$$

$$A_i = B_{i+1}.$$

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків:

– Меню: Сертифікати; Плагіни; Налаштування; Довідка.

– Лівого бокового блоку: Серверні дані (IP адреса, порт); Підключення з використанням одного з двох запропонованих алгоритмів захисту; Відключитись; Видалити; Додати; Налаштувати; Вікно авторського права (рисунок 5.2).

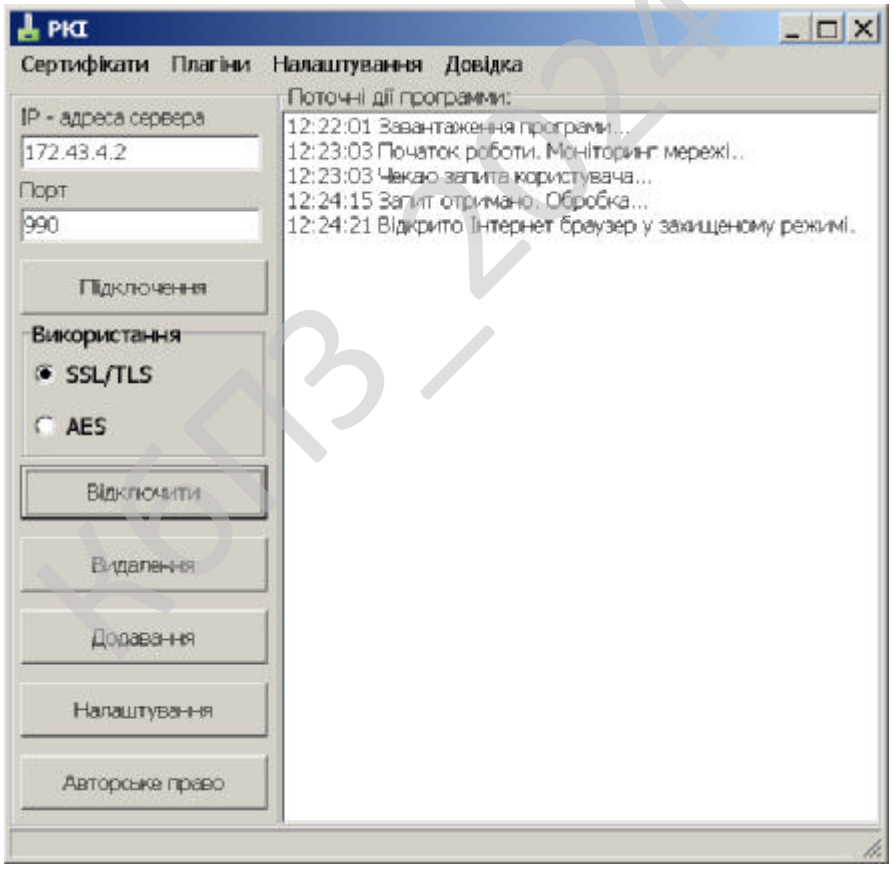

Рисунок 5.1 – Головне вікно програми

Програма проста у використанні й не вимагає установки на комп'ютері користувача іншого програмного забезпечення або баз даних.

Використовуючи комбінацію тунелювання, шифрування, автентифікації й контролю доступу, програма надає користувачам захищений спосіб доступу через Інтернет.

Після запуску програми треба ввести у сервер та порт, по якому буде відбуватися передача даних. Слід вибрати алгоритм захисту, який буде використовуватися під час роботи з мережою.

Користувач має можливість використати алгоритм захисту інформації SSL/TLS або AES.

Якщо користувач перший раз використовує програму та не має сертифікату, або строк дії його сертифіката закінчився, то необхідно згенерувати запит на одержання сертифікату.

Після того як користувач вказав IP-адресу сервера та порт передачі даних, вибрав алгоритм захисту інформації та отримав сертифікат та закритий ключ, він може здійснити з'єднання з сервером по мережі і відкрити Інтернет браузер у захищеному режимі.

На рисунку 5.2 зображено форму авторського права. Обрана ліцензія – вільне розповсюдження (freeware).

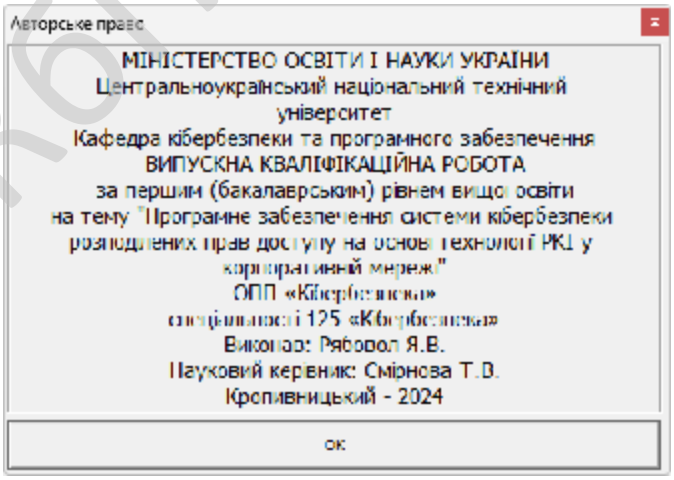

Рисунок 5.2 – Довідка розробника

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем розподілених прав доступу на основі технології PKI у корпоративній мережі.

– Досліджена система розподілених прав доступу на основі технології PKI у корпоративній мережі.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання розподілених прав доступу на основі технології PKI у корпоративній мережі.

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

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

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SHACAL-2.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

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

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

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

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

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

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

8. 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,

9. 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.

10. 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.

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

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

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

14. 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.*

15. 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.

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

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

18. 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.

19. 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.

20. 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.

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

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

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

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

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

30. 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.

31. 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.

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

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

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

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

36. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

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

38. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes»,

2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

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

40. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

41. 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.

42. 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.

43. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

44. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

45. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична*

конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

46. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

47. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

48. 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.

49. 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.

50. 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.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 136-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 61 аркуш.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 10.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Смірнова Т.В.

Програмне забезпечення системи кібербезпеки розподілених прав доступу на основі технології PKI у корпоративній мережі

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 38

Літера: РП

Кропивницький – 2024 року

Файл PKIServer.cpp – Серверна частина

```
#include "StdAfx.h"
#include "PKIserver.h"
#include <vcclr.h>

namespace PKI
{
namespace Server
{
    PKIServer::PKIServer(void)
    {
        try
        {
                  m_pCS = __nogc new CRITICAL_SECTION();
            m_pSChannelCred = __nogc new SCHANNEL_CRED();
            m_phServerCreds = __nogc new CredHandle();
            m_pID2Clients = new CLIENTS_HM_TYPE();
        }
        catch(const std::bad_alloc&)
        {
            throw new OutOfMemoryException();
        }
            InitializeCriticalSection(m_pCS);
        m_bAskClientForAuth = false;
        SecInvalidateHandle(m_phServerCreds);
        ZeroMemory(m_pSChannelCred, sizeof(SCHANNEL_CRED));
        m_pSecurityFunc = NULL;
        m_hSecurity = NULL;
        SecurityFunctionTable* pSecurityFunc = m_pSecurityFunc;
        if(!LoadSecurityLibrary(m_hSecurity, pSecurityFunc))
            throw new Common::Exceptions::PKIException(S"Failed to load security
dll.");
        m_pSecurityFunc = pSecurityFunc;
    }
    bool PKIServer::SSPINegotiateLoop(void* IoBuffer, int& ActualLen, SecBuffer
*pExtraData, Guid ClientID, Object* state)
    {
        TimeStamp           tsExpiry;
        CAutoSecBuffer<2>    InBuffer(m_pSecurityFunc, false);
        CAutoSecBuffer<1>    OutBuffer(m_pSecurityFunc, true);
        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ASC_REQ_SEQUENCE_DETECT |
                            ASC_REQ_REPLAY_DETECT   |
                            ASC_REQ_CONFIDENTIALITY |
                            ASC_REQ_EXTENDED_ERROR  |
                            ASC_REQ_ALLOCATE_MEMORY |
                            ASC_REQ_STREAM;

        CLIENTDATA* pClientData;
        GetClientIDAssoc(ClientID, pClientData);
        if(m_bAskClientForAuth)
        {
            dwSSPIFlags |= ASC_REQ_MUTUAL_AUTH;
        }

        SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;

        while( scRet == SEC_I_CONTINUE_NEEDED)
        {
            //
            // InBuffers[1] є для зовнішніх даних
            //  SSPI/SCHANNEL
            //
            InBuffer.SetSecurityBufferToken(0, IoBuffer, ActualLen);
            InBuffer.SetSecurityBufferEmpty(1);
            OutBuffer.SetSecurityBufferToken(0, NULL, 0);
```

```
            scRet = m_pSecurityFunc->AcceptSecurityContext(
                            m_phServerCreds,
                            SecIsValidHandle(&(pClientData-
>hContext))?&(pClientData->hContext):NULL,
                            &InBuffer,
                            dwSSPIFlags,
                            SECURITY_NATIVE_DREP,
                            &pClientData->hContext,
                            &OutBuffer,
                            &dwSSPIOutFlags,
                            &tsExpiry);

            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
                //викликання повідомленя для збереження в буфер
                return false;
            }
            if(scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED ||
                (FAILED(scRet) && (0 != (dwSSPIOutFlags &
ISC_RET_EXTENDED_ERROR))))
            {
                if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL )
                {
                    bool bSent = DispatchSend(static_cast<const
char*>(OutBuffer[0].pvBuffer), OutBuffer[0].cbBuffer, ClientID, state);
                    if(!bSent)
                    {
                                    throw new
Common::Exceptions::PKISendException(S"Відправлення на сервер не відбулося.");
                    }
                    OutBuffer.FreeBuffer(0);
                }
            }
            if ( scRet == SEC_E_OK )
            {
                // відкладені дані
                // Якщо додатковий буфер містить дані, - це зашифровані дані для
прикладного рівня. Повинно бути збережено. Прикладний рівень дешифрує це з
DecryptMessage.
                //
                if(InBuffer[1].BufferType == SECBUFFER_EXTRA)
                {
                    pExtraData->pvBuffer = malloc(InBuffer[1].cbBuffer);
                    if(pExtraData->pvBuffer == NULL)
                    {
                        throw new OutOfMemoryException();
                    }

                    MoveMemory(pExtraData->pvBuffer,(BYTE*)IoBuffer + (ActualLen
- InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);

                    pExtraData->cbBuffer   = InBuffer[1].cbBuffer;
                    pExtraData->BufferType = SECBUFFER_TOKEN;
                }
                else
                {
                    pExtraData->pvBuffer   = NULL;
                    pExtraData->cbBuffer   = 0;
                    pExtraData->BufferType = SECBUFFER_EMPTY;
                }
                pClientData->bInHandShakeLoop = false;
                if(DoClientCertVerify != NULL)
                {
                    Common::Misc::CeriticateInfo ServCerInfo;
                    VerifyCertificate(false, m_pSecurityFunc,&(pClientData-
>hContext), NULL, 0, &ServCerInfo);
                        DoClientCertVerify(ServCerInfo);
                }
                if(DoHandShakeSuccess != NULL)
```

```
                {
                    DoHandShakeSuccess(ClientID);
                }
                break;
            }
            else if(FAILED(scRet))
            {
               throw new Common::Exceptions::PKIException(S"Рукопотискання з
сервером не відбулося. Помилка: ", scRet);
            }

            if ( InBuffer[1].BufferType == SECBUFFER_EXTRA )
            {
                MoveMemory(IoBuffer,(BYTE*)IoBuffer + (ActualLen -
InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                ActualLen = InBuffer[1].cbBuffer;
            }
            else if(scRet == SEC_I_CONTINUE_NEEDED)
            {
                ActualLen = 0;
                break;
            }
            else
            {
                //не повинно відбуватися
                //беремо новий Common::Exceptions::PKIException(S"Неправильна
умова. Помилка: ", scRet);
            }
        }
        return true;
    }

    void PKIServer::DisconnectFromClient(Guid ClientID, Object* state)
    {
        DWORD dwType = SCHANNEL_SHUTDOWN;
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, false);
        OutBuffer.SetSecurityBufferToken(0, &dwType, sizeof(dwType));

        CLIENTDATA* pClientData;
        GetClientIDAssoc(ClientID, pClientData);
        SECURITY_STATUS Status = m_pSecurityFunc-
>ApplyControlToken(&(pClientData->hContext), &OutBuffer);

        if(FAILED(Status))
        {
            RemoveClient(ClientID);
            throw new Common::Exceptions::PKIException(S"Помилка з'єднання.
Помилка: ", Status);
        }

        DWORD dwSSPIFlags = ASC_REQ_SEQUENCE_DETECT |
                            ASC_REQ_REPLAY_DETECT   |
                            ASC_REQ_CONFIDENTIALITY |
                            ASC_REQ_EXTENDED_ERROR  |
                            ASC_REQ_ALLOCATE_MEMORY |
                            ASC_REQ_STREAM;

        OutBuffer.SetSecurityBufferToken(0, NULL, 0);

        DWORD     dwSSPIOutFlags;
        TimeStamp tsExpiry;
        Status = m_pSecurityFunc->AcceptSecurityContext(
                       m_phServerCreds,
                       &(pClientData->hContext),
                       NULL,
                       dwSSPIFlags,
                       SECURITY_NATIVE_DREP,
                       NULL,
                       &OutBuffer,
```

```
                        &dwSSPIOutFlags,
                        &tsExpiry);

        if(FAILED(Status))
        {
            RemoveClient(ClientID);
            throw new Common::Exceptions::PKIException("Помилка відключення
InitializeSecurityContext.");
        }

        char* pbMessage = static_cast<char*>(OutBuffer[0].pvBuffer);
        DWORD cbMessage = OutBuffer[0].cbBuffer;

        if(pbMessage != NULL && cbMessage != 0)
        {
            bool bRead = DispatchSend(pbMessage, cbMessage, ClientID, state);
            if(!bRead)
            {
                throw new Common::Exceptions::PKISendException(S"Відправлення на
сервер не відбулося.");
            }
            m_pSecurityFunc->FreeContextBuffer(pbMessage);
        }
        RemoveClient(ClientID);
    }
    void PKIServer::EncryptSend(Byte data[], int ActualLen, Guid ClientID,
Object* state)
    {
        CLIENTDATA* pClientData;
        GetClientIDAssoc(ClientID, pClientData);
        SecPkgContext_StreamSizes Sizes;
          int IoBufferLength = GetMaxChunkSize(Sizes, ClientID);
        IoBufferLength += Sizes.cbHeader + Sizes.cbTrailer;
#ifdef _DEBUG
        if(GetMaxChunkSize(Sizes, ClientID) < (DWORD)data->Length)
            throw new Common::Exceptions::PKIException("Розмір даних
специфікації не є правильним.");
#endif

        CAutoSecBuffer<4> Buffers(m_pSecurityFunc, false);

            char* pbIoBuffer = (char*)malloc(IoBufferLength);
        if(pbIoBuffer == NULL)
            throw new OutOfMemoryException();

        Marshal::Copy(data, 0, pbIoBuffer + Sizes.cbHeader, ActualLen);

        Buffers.SetSecurityBufferStreamHeader(0, pbIoBuffer, Sizes.cbHeader);
        Buffers.SetSecurityBufferData(1, pbIoBuffer + Sizes.cbHeader,
ActualLen);
        Buffers.SetSecurityBufferStreamTrailer(2, pbIoBuffer + Sizes.cbHeader +
ActualLen, Sizes.cbTrailer);
        Buffers.SetSecurityBufferEmpty(3);
        SECURITY_STATUS scRet = m_pSecurityFunc->EncryptMessage(&(pClientData-
>hContext), 0, &Buffers, 0);

            if(FAILED(scRet) && scRet != SEC_E_CONTEXT_EXPIRED)
            {
            free(pbIoBuffer);
                throw new Common::Exceptions::PKIException(S"Помилка шифрування
повідомлення. Помилка: ", scRet);
            }

            int OutBufferLen =
Buffers[0].cbBuffer+Buffers[1].cbBuffer+Buffers[2].cbBuffer;

        if(!DispatchSend(static_cast<char*>(pbIoBuffer), OutBufferLen, ClientID,
state))
        {
```

```
            free(pbIoBuffer);
            throw new Common::Exceptions::PKISendException(S"Помилка
відправлення. Помилка: ", scRet);
        }
        free(pbIoBuffer);
    }

    void PKIServer::DecryptData(Byte data[], Int32 ActualLen, Guid ClientID,
Object* state)
    {
        CLIENTDATA* pClientData=NULL;
        try
        {
                GetClientIDAssoc(ClientID, pClientData);
        }
        catch(Common::Exceptions::PKIServerFailedToFindExistingClientID*)
        {
            //новий клієнт ініціалізує потрібні дані для наступного з'єднання
            pClientData = new CLIENTDATA(m_pSecurityFunc);
            String* sClientID = ClientID.ToString();
            const wchar_t __pin* pClientID = PtrToStringChars(sClientID);
            try
            {
                        EnterCriticalSection(m_pCS);
                    (*m_pID2Clients)[pClientID] = pClientData;
                        LeaveCriticalSection(m_pCS);
            }
            catch(const std::bad_alloc&)
            {
                        LeaveCriticalSection(m_pCS);
                throw new OutOfMemoryException();
            }
        }
        //додаємо попередній відкладений буфер
        ActualLen += pClientData->secExtraBuffer.cbBuffer;
        BYTE* pReadBuff = (BYTE*)malloc(ActualLen);
        if(pReadBuff == NULL)
            new OutOfMemoryException();
                Marshal::Copy(data, 0, pReadBuff+pClientData-
>secExtraBuffer.cbBuffer, ActualLen-pClientData->secExtraBuffer.cbBuffer);
        if(pClientData->secExtraBuffer.cbBuffer > 0)
        {
            //копіюємо з попередніх відкладені дані в початок
            MoveMemory(pReadBuff, pClientData->secExtraBuffer.pvBuffer,
pClientData->secExtraBuffer.cbBuffer);
            free(pClientData->secExtraBuffer.pvBuffer);
            pClientData->secExtraBuffer.cbBuffer = 0;
            pClientData->secExtraBuffer.pvBuffer = NULL;
        }
        SecBuffer ExtraBuffer={0};
        if(pClientData->bInHandShakeLoop)
        {
            if(!SSPINegotiateLoop(pReadBuff, ActualLen, &ExtraBuffer, ClientID,
state))
            {
                Вхідний буфер містить тільки фрагмент закодованих даних.
Необхідно зберегти фрагмент та чекати надходження наступних даних
                pClientData->secExtraBuffer.pvBuffer = pReadBuff;
                pClientData->secExtraBuffer.cbBuffer = ActualLen;
                return;
            }
            if(ExtraBuffer.cbBuffer == 0)
            {
                free(pReadBuff);
                return;
            }
            else if(ExtraBuffer.pvBuffer)
            {
```

```
                //зберігаємо зовнішні дані та відправляємо їх до розшифрованого
повідомлення
                MoveMemory(pReadBuff, ExtraBuffer.pvBuffer,
ExtraBuffer.cbBuffer);
                ActualLen = ExtraBuffer.cbBuffer;
                free(ExtraBuffer.pvBuffer);
                ExtraBuffer.pvBuffer = NULL;
                ExtraBuffer.cbBuffer =0;
            }
        }

        while(true)
        {
            CAutoSecBuffer<4> Buffers(m_pSecurityFunc, false);
            Buffers.SetSecurityBufferData(0, pReadBuff, ActualLen);
            Buffers.SetSecurityBufferEmpty(1);
            Buffers.SetSecurityBufferEmpty(2);
            Buffers.SetSecurityBufferEmpty(3);
            CLIENTDATA* pClientData;
            GetClientIDAssoc(ClientID, pClientData);
            SECURITY_STATUS scRet = m_pSecurityFunc-
>DecryptMessage(&(pClientData->hContext), &Buffers, 0, NULL);

            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
                Вхідний буфер містить тільки фрагмент закодованих даних.
Необхідно зберегти фрагмент та чекати надходження наступних даних
                //
                pClientData->secExtraBuffer.pvBuffer = pReadBuff;
                pClientData->secExtraBuffer.cbBuffer = ActualLen;
                //pReadBuff визволений на наступному вході
                return;
            }
            if( scRet != SEC_E_OK &&
                scRet != SEC_I_RENEGOTIATE &&
                scRet != SEC_I_CONTEXT_EXPIRED)
            {
                free(pReadBuff);
                throw new Common::Exceptions::PKIException("Помилка
дешифрування. Помилка: ", scRet);
            }

            // Клієнт надсилає повідомлення про кінець сесії
            if(scRet == SEC_I_CONTEXT_EXPIRED)
                {
                //Шифруємо пустий буфер у відповідності зі специфікацією
                EncryptSend(new Byte[0], 0, ClientID, state);
                free(pReadBuff);
                //Dispose();
                RemoveClient(ClientID);
                throw new Common::Exceptions::PKIException("Помилка
дешифрування. Заповнення закінчено.");
                }

            // Локальні дані й (Опціонально) зовнішні буфера.
                SecBuffer* pDataBuffer = NULL;
                SecBuffer* pExtraBuffer=NULL;
            for(int i = 1; i < 4; i++)
            {
                if(pDataBuffer == NULL && Buffers[i].BufferType ==
SECBUFFER_DATA)
                {
                    pDataBuffer = &Buffers[i];
                }
                if(pExtraBuffer == NULL && Buffers[i].BufferType ==
SECBUFFER_EXTRA)
                {
                    pExtraBuffer = &Buffers[i];
                }
```

```
            }

            // Відображеня типу декодування даних.
            if(pDataBuffer && pDataBuffer->cbBuffer > 0)
            {
                DispatchPlainData(pDataBuffer->pvBuffer, pDataBuffer->cbBuffer,
ClientID, state);
            }

            // Переміщуємо додаткові дані на вхідний буфер, змінюємо довжину й
виробляємо шифрування
            if(pExtraBuffer != NULL)
            {
                MoveMemory(pReadBuff, pExtraBuffer->pvBuffer, pExtraBuffer-
>cbBuffer);
                ActualLen = pExtraBuffer->cbBuffer;
            }
            else if(scRet == S_OK)
                    break;
            if(scRet == SEC_I_RENEGOTIATE)
            {
                // Клієнт готовий до іншого рукопотискання.
                pClientData->bInHandShakeLoop=true;
                int dummy =0;
                if(pExtraBuffer != NULL)
                    SSPINegotiateLoop(pReadBuff, ActualLen, &ExtraBuffer,
ClientID, state);
                else
                    SSPINegotiateLoop(NULL, dummy, &ExtraBuffer, ClientID,
state);
                // Переміщуємо інші додаткові дані у вхідний буфер.
                if(ExtraBuffer.pvBuffer != NULL)
                {
                    MoveMemory(pReadBuff, ExtraBuffer.pvBuffer,
ExtraBuffer.cbBuffer);
                    ActualLen = ExtraBuffer.cbBuffer;
                    free(ExtraBuffer.pvBuffer);
                    ExtraBuffer.pvBuffer = NULL;
                    ExtraBuffer.cbBuffer =0;
                }
                else
                    break;
            }
        }
          free(pReadBuff);
    }
    void PKIServer::DispatchPlainData(void* pData, long Len, Guid ClientID,
Object* state)
    {
        Byte data[] = new Byte[Len];
        Marshal::Copy(IntPtr(pData), data, 0, Len);
        DoPlainData(data, ClientID, state);
    }
    bool PKIServer::DispatchSend(const char* pbMessage, DWORD cbMessage, Guid
ClientID, Object* state)
    {
        Byte data[] = new Byte[cbMessage];
        Marshal::Copy(IntPtr((void*)pbMessage), data, 0, cbMessage);
        return DoWrite(data,ClientID, state);
    }
    DWORD PKIServer::GetMaxChunkSize(SecPkgContext_StreamSizes& Sizes, Guid
ClientID)
    {
        CLIENTDATA* pClientData;
        GetClientIDAssoc(ClientID, pClientData);
        SECURITY_STATUS scRet = m_pSecurityFunc-
>QueryContextAttributesA(&(pClientData-
>hContext),SECPKG_ATTR_STREAM_SIZES,&Sizes);
        if(scRet != SEC_E_OK)
```

```
        {
            throw new Common::Exceptions::PKIException(S"Взятий максимальний
розмір блоку PKI помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    void PKIServer::SetupCredentials(Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot)
    {
        TimeStamp       tsExpiry;
        SECURITY_STATUS Status;
        HCERTSTORE      hCertStore;
        PCCERT_CONTEXT  pCertContext = NULL;

        // Вдкриваємо "MY" сертифікаційні записи, у Internet Explorer записах
клієнтських сертифікатів.
        hCertStore = CertOpenSystemStore(0, _T("MY"));
        if(hCertStore == NULL)
        {
           throw new Common::Exceptions::PKIException(String::Concat(S"помилка
відкриття MY сертифікаційних записів. Помилка: ", Convert::ToString((unsigned
int)GetLastError())));
        }
        if(thumbPrint != NULL && thumbPrint->Length > 0)
        {
            int HashLen = thumbPrint->Length;
            BYTE* pbData = (BYTE*)malloc(HashLen);
            Marshal::Copy(thumbPrint, 0, pbData, HashLen);
            CRYPT_HASH_BLOB hash={HashLen, pbData};
            SetLastError(0);
            pCertContext =
CertFindCertificateInStore(hCertStore,X509_ASN_ENCODING,

0,CERT_FIND_HASH,&hash,NULL);
            free(pbData);
            Status = GetLastError();
            if(pCertContext == NULL)
            {
                CertCloseStore(hCertStore, 0);
                hCertStore = NULL;
                throw new
Common::Exceptions::PKIException(String::Concat(S"Помилка сертифікаційної
інформації. Помилка: ", Convert::ToString((unsigned int)Status)));
            }
        }
        m_pSChannelCred->dwVersion  = SCHANNEL_CRED_VERSION;
        if(pCertContext != NULL)
        {
            m_pSChannelCred->cCreds     = 1;
            m_pSChannelCred->paCred     = &pCertContext;
        }
        m_pSChannelCred->grbitEnabledProtocols = prot;
        m_pSChannelCred->dwFlags = 0;

        Status = m_pSecurityFunc->AcquireCredentialsHandleA( NULL,
// Найменуваня автентифікатору
                                                             UNISP_NAME_A,
// Найменування пакету

SECPKG_CRED_INBOUND,   // Прапор використання
                                                             NULL,
// Крапка підключення ID
                                                             m_pSChannelCred,
// Пакет спеціальних даних
                                                             NULL,
// Вказник на GetKey()
                                                             NULL,
// Зашифровані значення в GetKey()
```

```
                                                m_phServerCreds,        // (out)
Рукопотискання
                                                                        &tsExpiry);
// (out) Час життя (опція)
        if(Status != SEC_E_OK)
        {
            if(pCertContext != NULL)
            {
                CertCloseStore(hCertStore, 0);
                hCertStore = NULL;
                CertFreeCertificateContext(pCertContext);
            }
            throw new Common::Exceptions::PKIException(String::Concat(S"Помилка
створення автентифікатору. Помилка: ", Convert::ToString((int)Status)));
        }
        //
        // Звільнення контексту сертифікату. Копія була створенна у каналі.
        //
        if(pCertContext != NULL)
        {
            CertCloseStore(hCertStore, 0);
            CertFreeCertificateContext(pCertContext);
            pCertContext = NULL;
        }
    }
    void PKIServer::GetClientIDAssoc(Guid ClientID, CLIENTDATA __nogc*&
pClientData)
    {
        String* sClientID = ClientID.ToString();
        const wchar_t __pin* pClientID = PtrToStringChars(sClientID);
            EnterCriticalSection(m_pCS);
        CLIENTS_HM_TYPE::iterator iter = m_pID2Clients-
>find(std::wstring(pClientID));
        if(iter != m_pID2Clients->end())
        {
            pClientData = iter->second;
        }
        else
        {
                  LeaveCriticalSection(m_pCS);
                  throw new
Common::Exceptions::PKIServerFailedToFindExistingClientID("PKI сервер не знайшов
id клієнта");
        }
            LeaveCriticalSection(m_pCS);
    }
    void PKIServer::RemoveClient(Guid ClientID)
    {
        String* sClientID = ClientID.ToString();
        const wchar_t __pin* pClientID = PtrToStringChars(sClientID);
            EnterCriticalSection(m_pCS);
        CLIENTS_HM_TYPE::iterator iter = m_pID2Clients->find(pClientID);
        if(iter != m_pID2Clients->end())
        {
            delete iter->second;
            iter->second = NULL;
            m_pID2Clients->erase(pClientID);
        }
            LeaveCriticalSection(m_pCS);
    }

    void PKIServer::AskForRenegotiate(Guid ClientID, Object* state)
    {
        CLIENTDATA* pClientData;
        GetClientIDAssoc(ClientID, pClientData);
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);
        OutBuffer.SetSecurityBufferToken(0, NULL, 0);
        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ASC_REQ_SEQUENCE_DETECT |
```

```
                                ASC_REQ_REPLAY_DETECT   |
                                ASC_REQ_CONFIDENTIALITY |
                                ASC_REQ_EXTENDED_ERROR  |
                                ASC_REQ_ALLOCATE_MEMORY |
                                ASC_REQ_STREAM          |
                                ASC_REQ_MUTUAL_AUTH;
        SECURITY_STATUS scRet = m_pSecurityFunc->AcceptSecurityContext(
                                                m_phServerCreds,
                                                &(pClientData->hContext),
                                                NULL,
                                                dwSSPIFlags,
SECURITY_NATIVE_DREP,
                                                &(pClientData->hContext),
                                                &OutBuffer, &dwSSPIOutFlags,
NULL);
       if(scRet != SEC_E_OK)
           throw new Common::Exceptions::PKIException(S"Запит помилковий.
Помилка: ", scRet);
       if(OutBuffer[0].cbBuffer > 0 && OutBuffer[0].pvBuffer != NULL)
       {
            bool bSent = DispatchSend(static_cast<const
char*>(OutBuffer[0].pvBuffer), OutBuffer[0].cbBuffer, ClientID, state);
            if(!bSent)
            {
                throw new Common::Exceptions::PKISendException(S"Відправлення на
сервер не відбулося.");
            }
       }
    }
    void PKIServer::Dispose(bool disposing)
    {
            if(m_pCS != NULL)
            {
                  DeleteCriticalSection(m_pCS);
                  delete m_pCS;
                  m_pCS = NULL;
            }
        if(SecIsValidHandle(m_phServerCreds))
        {
            m_pSecurityFunc->FreeCredentialsHandle(m_phServerCreds);
            SecInvalidateHandle(m_phServerCreds);
        }
        delete m_pSChannelCred;
        delete m_phServerCreds;
        m_pSChannelCred = NULL;
        m_phServerCreds = NULL;
        if(m_pSecurityFunc != NULL)
        {
            FreeLibrary(m_hSecurity);
            m_hSecurity = NULL;
            m_pSecurityFunc = NULL;
        }
            //
        if(m_pID2Clients != NULL)
        {
            CLIENTS_HM_TYPE::iterator iter = m_pID2Clients->begin();
            for(; iter != m_pID2Clients->end(); iter++)
            {
                delete iter->second;
                iter->second = NULL;
            }
            m_pID2Clients->erase(m_pID2Clients->begin(), m_pID2Clients->end());
            delete m_pID2Clients;
            m_pID2Clients = NULL;
        }
                }
}
}
```

Файл PKIServer.h – бібліотека для файлу PKIServer.cpp

```
#pragma once

#include "PKIcommon.h"

namespace PKI
{
    namespace Server
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WritePKI(Byte data[], Guid ClientID, Object*
state);
        //Процес дешифрування даних
        public __delegate void PlainData(Byte data[], Guid ClientID, Object*
state);
        //інформація сертифікату клієнта, Опціонально
        public __delegate void VerifyClientCert(Common::Misc::CeriticateInfo
ClientCertInfo);
        //Рукопотискання з клієнтом відбулося
        public __delegate void HandShakeSuccess(Guid ClientID);

        __sealed public __gc class PKIServer : public IDisposable
        {
        public:
            PKIServer();
            ~PKIServer(){Dispose(false);}
        public:
            //роз'єднання з поточним клієнтом id
            void DisconnectFromClient(Guid ClientID, Object* state);
            //очищення
            void Dispose() { Dispose(true); GC::SuppressFinalize(this);}
            //шифруємо дані, шифровані дані повертаються в WritePKI
            void EncryptSend(Byte data[], int ActualLen, Guid ClientID, Object*
state);
            // розшифруємо дані, розшифровані дані повертаються в PlainData
callback
            void DecryptData(Byte data[], Int32 ActualLen, Guid ClientID,
Object* state);
            //видаляємо клієнта по id з внутрішнього списку клієнтів
            void RemoveClient(Guid ClientID);
            //максимальний розмір даних для посилки/отримання за один раз
            int MaxDataChunkSize(Guid ClientID)
            {
                SecPkgContext_StreamSizes notused; return
GetMaxChunkSize(notused, ClientID);
            }
            //визначаємо автентифікатори, certThumbPrint  - хеш сертифікату
            void SetupCredentials(Byte certThumbPrint[],
Common::Misc::SecurityProviderProtocol prot);
            //запит клієнта на автентифікатор та отримання нового
автентифікатора
            void AskForRenegotiate(Guid ClientID, Object* state);
        public:
            //рекомендований початковий розмір, коли починається рукопотискання.
Це максимальний розмір токену автентифікації
            __property int get_MaxInitialChunkSize()
            {
                PSecPkgInfo psecInfo;
                SECURITY_STATUS scRet = QuerySecurityPackageInfo(UNISP_NAME,
&psecInfo);
                if (scRet != SEC_E_OK)
                    throw new Common::Exceptions::PKIException(S"Взятий
максимальний розмір точену PKI помилковий. Помилка: ", scRet);
                return psecInfo->cbMaxToken;
            }
            //запит клієнта на забезпечення сертифікату або ні
```

```
            __property void set_AskClientForAuth(bool value)
            {
                m_bAskClientForAuth = value;
            }
        public:
            //Повертаємо зашифровані дані
            WritePKI* DoWrite;
            //Повертаємо розшифровані дані
            PlainData*  DoPlainData;
            //Опціонально
            VerifyClientCert* DoClientCertVerify;
            //Опціонально
            HandShakeSuccess* DoHandShakeSuccess;
        ////////////////////////////////////////////////////////////////
        private:
            bool SSPINegotiateLoop(void*, int&, SecBuffer*, Guid, Object*);
            bool DispatchSend(const char*, DWORD, Guid, Object*);
            void DispatchPlainData(void*, long, Guid, Object*);
            DWORD GetMaxChunkSize(SecPkgContext_StreamSizes&, Guid);
            void Dispose(bool disposing);
        private:
            __nogc struct CLIENTDATA
            {
                CtxtHandle hContext;
                SecBuffer  secExtraBuffer;
                bool       bInHandShakeLoop;
                SecurityFunctionTable __nogc* pSecurityFunc;
                CLIENTDATA(SecurityFunctionTable __nogc*
pSecFunc):bInHandShakeLoop(true),pSecurityFunc(pSecFunc)
                {
                    SecInvalidateHandle(&hContext);
                    secExtraBuffer.BufferType = -1;
                    secExtraBuffer.cbBuffer = 0;
                    secExtraBuffer.pvBuffer = NULL;
                }
                ~CLIENTDATA()
                {
                    if(secExtraBuffer.cbBuffer > 0)
                    {
                        free(secExtraBuffer.pvBuffer);
                        secExtraBuffer.cbBuffer = 0;
                        secExtraBuffer.pvBuffer = NULL;
                    }

                    pSecurityFunc->DeleteSecurityContext(&hContext);
                    SecInvalidateHandle(&hContext);
                }
            };
        private:
            void GetClientIDAssoc(Guid, CLIENTDATA __nogc*&);
        private:
                  CRITICAL_SECTION __nogc* m_pCS;
            SCHANNEL_CRED __nogc*    m_pSChannelCred;
            CredHandle __nogc*            m_phServerCreds;
            SecurityFunctionTable __nogc* m_pSecurityFunc;
            HMODULE m_hSecurity;
            //typedef std::hash_map<std::wstring, CLIENTDATA __nogc*,
WStringComp> CLIENTS_HM_TYPE;
            typedef std::map<std::wstring, CLIENTDATA __nogc*, WStringComp>
CLIENTS_HM_TYPE;
            CLIENTS_HM_TYPE __nogc* m_pID2Clients;
            bool m_bAskClientForAuth;
        };
    }
}
```

Основна програма

PKI.vcproj – Файл конфігурації проекту

```
<?xml version="1.0" encoding = "Windows-1252"?>
<VisualStudioProject
      ProjectType="Visual C++"
      Version="7.00"
      Name="PKI"
      ProjectGUID="{B5EB056C-668A-41F0-9C4B-871A0D900D85}"
      Keyword="ManagedCProj">
      <Platforms>
            <Platform
                  Name="Win32"/>
      </Platforms>
      <Configurations>
            <Configuration
                  Name="Debug|Win32"
                  OutputDirectory="Debug"
                  IntermediateDirectory="Debug"
                  ConfigurationType="2"
                  CharacterSet="2"
                  ManagedExtensions="TRUE">
                  <Tool
                        Name="VCCLCompilerTool"
                        Optimization="0"
                        PreprocessorDefinitions="WIN32;_DEBUG"
                        MinimalRebuild="FALSE"
                        BasicRuntimeChecks="0"
                        RuntimeLibrary="1"
                        UsePrecompiledHeader="3"
                        WarningLevel="3"
                        DebugInformationFormat="3"/>
                  <Tool
                        Name="VCCustomBuildTool"/>
                  <Tool
                        Name="VCLinkerTool"
                        OutputFile="$(OutDir)/PKI.dll"
                        LinkIncremental="2"
                        GenerateDebugInformation="TRUE"/>
                  <Tool
                        Name="VCMIDLTool"/>
                  <Tool
                        Name="VCPostBuildEventTool"/>
                  <Tool
                        Name="VCPreBuildEventTool"/>
                  <Tool
                        Name="VCPreLinkEventTool"/>
                  <Tool
                        Name="VCResourceCompilerTool"/>
                  <Tool
                        Name="VCWebServiceProxyGeneratorTool"/>
                  <Tool
                        Name="VCWebDeploymentTool"/>
            </Configuration>
            <Configuration
                  Name="Release|Win32"
                  OutputDirectory="Release"
                  IntermediateDirectory="Release"
                  ConfigurationType="2"
                  CharacterSet="2"
                  ManagedExtensions="TRUE">
                  <Tool
                        Name="VCCLCompilerTool"
                        Optimization="2"
                        InlineFunctionExpansion="1"
                        PreprocessorDefinitions="WIN32;NDEBUG"
                        MinimalRebuild="FALSE"
```

```
                BasicRuntimeChecks="0"
                RuntimeLibrary="2"
                UsePrecompiledHeader="3"
                WarningLevel="3"/>
            <Tool
                Name="VCCustomBuildTool"/>
            <Tool
                Name="VCLinkerTool"
                OutputFile="$(OutDir)/PKI.dll"
                LinkIncremental="1"
                GenerateDebugInformation="TRUE"/>
            <Tool
                Name="VCMIDLTool"/>
            <Tool
                Name="VCPostBuildEventTool"/>
            <Tool
                Name="VCPreBuildEventTool"/>
            <Tool
                Name="VCPreLinkEventTool"/>
            <Tool
                Name="VCResourceCompilerTool"/>
            <Tool
                Name="VCWebServiceProxyGeneratorTool"/>
            <Tool
                Name="VCWebDeploymentTool"/>
        </Configuration>
    </Configurations>
    <Files>
        <Filter
            Name="Source Files"
            Filter="cpp;c;cxx;def;odl;idl;hpj;bat;asm">
            <File
                RelativePath="AssemblyInfo.cpp">
            </File>
            <File
                RelativePath="PKI.cpp">
            </File>
            <File
                RelativePath="PKIServer.cpp">
            </File>
            <File
                RelativePath="Stdafx.cpp">
                <FileConfiguration
                    Name="Debug|Win32">
                    <Tool
                        Name="VCCLCompilerTool"
                        UsePrecompiledHeader="1"/>
                </FileConfiguration>
                <FileConfiguration
                    Name="Release|Win32">
                    <Tool
                        Name="VCCLCompilerTool"
                        UsePrecompiledHeader="1"/>
                </FileConfiguration>
            </File>
            <File
                RelativePath="PKIcommon.cpp">
            </File>
        </Filter>
        <Filter
            Name="Header Files"
            Filter="h;hpp;hxx;hm;inl;inc">
            <File
                RelativePath="PKI.h">
            </File>
            <File
                RelativePath="PKIServer.h">
            </File>
            <File
```

```
                        RelativePath="Stdafx.h">
                    </File>
                    <File
                        RelativePath="PKIcommon.h">
                    </File>
                </Filter>
                <Filter
                    Name="Resource Files"

        Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;r">
                </Filter>
                <File
                    RelativePath="ReadMe.txt">
                </File>
        </Files>
        <Globals>
        </Globals>
</VisualStudioProject>
```

Файл PKI.cpp основної програми

```
#include "stdafx.h"
#include "PKI.h"
#include <new>
namespace PKI
{
namespace Client
{
    PKIConnection::PKIConnection()
    {
        Init();
    }

    void PKIConnection::InitiateHandShake(String* ipAddres, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state)
    {
        if(m_ServerIP != NULL)
        {
            Dispose();
            Init();
        }
        m_ServerIP = ipAddres;
        try
        {
            SetupCredentials(thumbPrint, prot);
            PerformHandShake(state);
        }
        catch(Common::Exceptions::PKIException*)
        {
            Dispose();
            throw;
        }
    }

    void PKIConnection::PerformHandShake(Object* state)
    {
        DWORD           dwSSPIFlags;
        DWORD           dwSSPIOutFlags;
        TimeStamp       tsExpiry;
        SECURITY_STATUS scRet = S_OK;

        dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT   |
                      ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR  |
                      ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;
        //
        //  Ініціалізуємо повідомлення ClientHello та генеруємо токен.
        //
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);
        OutBuffer.SetSecurityBufferToken(0, NULL, 0);

        IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemAnsi(m_ServerIP);
        scRet = m_pSecurityFunc->InitializeSecurityContextA( m_phClientCreds,
NULL,
                        static_cast<SEC_CHAR*>(ptrServerName.ToPointer()),
                        dwSSPIFlags,0,
                        SECURITY_NATIVE_DREP,
                        NULL, 0, m_phContext, &OutBuffer,
                        &dwSSPIOutFlags, &tsExpiry);
        System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);

        if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
        {
            DoRenegotiate(this);
        }
        else if(scRet != SEC_I_CONTINUE_NEEDED)
        {
```

```
            throw new
Common::Exceptions::PKIException(S"InitializeSecurityContext Помилкове. Помилка:
", scRet);
        }

        m_bInHandShake = true;

        // Відправлення відповіді на сервер, якщо він готовий.
        if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
        {
            bool bSent = DispatchSend(static_cast<char*>(OutBuffer[0].pvBuffer),
OutBuffer[0].cbBuffer, state);
            if(!bSent)
            {
                throw new Common::Exceptions::PKISendException(S"Відправлення
помилки Сервера.");
            }
        }
    }
    bool PKIConnection::ClientHandshakeLoop(void* IoBuffer, int& ActualLen,
SecBuffer *pExtraData, Object* state)
    {
        CAutoSecBuffer<2> InBuffer(m_pSecurityFunc, false);
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);

        TimeStamp tsExpiry;

        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT |
                            ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR
|
                            ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;

        SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;

        while(scRet == SEC_I_CONTINUE_NEEDED || scRet ==
SEC_I_INCOMPLETE_CREDENTIALS)
          {
            //
            // Встановлення вхідних буферів. Буфер 0 використовує шифрування в
даних отриманих з серверу. Schannel читає усі дані або тільки признаки.
Відложені дані будуть накопичуватися у буфері  1 та надавати буферу тип
SECBUFFER_EXTRA.
            //

            InBuffer.SetSecurityBufferToken(0, IoBuffer, ActualLen);
            InBuffer.SetSecurityBufferEmpty(1);
            OutBuffer.SetSecurityBufferToken(0, NULL, 0);
            scRet = m_pSecurityFunc->InitializeSecurityContextA(m_phClientCreds,
                                        m_phContext,
                                        NULL,
                                        dwSSPIFlags,
                                        0,
                                        SECURITY_NATIVE_DREP,
                                        &InBuffer,
                                        0,
                                        NULL,
                                        &OutBuffer,
                                        &dwSSPIOutFlags,
                                        &tsExpiry);
            //
            // Якщо InitializeSecurityContext працює правильно (або якщо помилка
припустима), відправляємо зміст вихідного буфера на сервер.
            //

            if(scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED ||
(FAILED(scRet)
               && (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR)))
            {
```

```
                if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
                {
                    bool bSent =
DispatchSend(static_cast<char*>(OutBuffer[0].pvBuffer), OutBuffer[0].cbBuffer,
state);
                    if(!bSent)
                    {
                                    throw new
Common::Exceptions::PKISendException(S"Відправлення на сервер не відбулося.");
                    }
                }
                OutBuffer.FreeBuffer(0);
            }
            //
            // Якщо InitializeSecurityContext повертає SEC_E_INCOMPLETE_MESSAGE,
тоді ми  читаємо наступні данні з сервера.
            //
            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
                //Викликаємо повідомлення для збереження змісту буферу у разі
помилки при подальшому вводі;
            }
            //
            // Якщо InitializeSecurityContext повертає SEC_E_OK, тоді
рукопотиск5ання завершено успішно.
            //
            if(scRet == SEC_E_OK)
            {
                //
                // Якщо "додатковий" буфер містить дані – то це закодована
інформація для прикладного протоколу OSI. Це повинно бути збережено. Данні для
прикладного рівня будуть декодовані з  DecryptMessage.
                //
                if(InBuffer[1].BufferType == SECBUFFER_EXTRA)
                {
                    pExtraData->pvBuffer = malloc(InBuffer[1].cbBuffer);
                    if(pExtraData->pvBuffer == NULL)
                    {
                        throw new OutOfMemoryException();
                    }

                    MoveMemory(pExtraData->pvBuffer,(BYTE*)IoBuffer + (ActualLen
- InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);

                    pExtraData->cbBuffer   = InBuffer[1].cbBuffer;
                    pExtraData->BufferType = SECBUFFER_TOKEN;
                }
                else
                {
                    pExtraData->pvBuffer   = NULL;
                    pExtraData->cbBuffer   = 0;
                    pExtraData->BufferType = SECBUFFER_EMPTY;
                }

                //
                // Для виходу зберігається у БД
                //
                m_bInHandShake = false;
                if(DoServerCertVerify != NULL)
                {
                    IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemUni(m_ServerIP);
                    Common::Misc::CeriticateInfo ServCertInfo;
                    VerifyCertificate(true, m_pSecurityFunc, m_phContext,
static_cast<wchar_t*>(ptrServerName.ToPointer()), 0, &ServCertInfo);
                        DoServerCertVerify(ServCertInfo);

System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);
```

```
                    DoServerCertVerify = NULL; //заборона інших викликів під час
виконання процедури renegotiation.
                }
                if(DoHandShakeSuccess != NULL)
                {
                    DoHandShakeSuccess();
                }
                break;
            }

            //
            // Крапка невиправної помилки .
            //

            if(FAILED(scRet))
            {
                throw new Common::Exceptions::PKIException(S"Рукопотискання з
сервером помилкове. Помилка: ", scRet);
            }
            //
            // Якщо InitializeSecurityContext повертає
SEC_I_INCOMPLETE_CREDENTIALS,
            // тоді сервер автентифікує клієнта.
            //
            if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
            {
                if(DoRenegotiate != NULL)
                    DoRenegotiate(this);
                SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
                continue;
            }
            //
            // Копіюємо любі відложені дані з «додаткового» буфера й виконуємо
наступний раунд.
            //
            if ( InBuffer[1].BufferType == SECBUFFER_EXTRA )
            {
                int temp = ActualLen;
                MoveMemory(IoBuffer,(BYTE*)IoBuffer + (ActualLen -
InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                ActualLen = InBuffer[1].cbBuffer;
            }
            else
            {
                ActualLen = 0;
                break;
            }
        }
        return true;
    }

    void PKIConnection::LoadNewClientCredentials(Byte certhash[])
    {
        CredHandle                                  hCreds;
        SecPkgContext_IssuerListInfoEx   IssuerListInfo;
        PCCERT_CHAIN_CONTEXT             pChainContext;
        CERT_CHAIN_FIND_BY_ISSUER_PARA   FindByIssuerPara;
        PCCERT_CONTEXT                        pCertContext;
        TimeStamp                             tsExpiry;
        SECURITY_STATUS                       Status;
        HCERTSTORE                      hCertStore;
        //
        // Читаємо список надійних джерел з schannel.
        //
        Status = m_pSecurityFunc->QueryContextAttributesA(m_phContext,

      SECPKG_ATTR_ISSUER_LIST_EX,(PVOID)&IssuerListInfo);
        if(Status != SEC_E_OK)
        {
```

```
            throw new Common::Exceptions::PKIException(S"Формування
автентифікатора зазнало поразки. Помилка: ", Status);
        }
        //
        // Перераховуємо сертифікати клієнта.
        //
        ZeroMemory(&FindByIssuerPara, sizeof(FindByIssuerPara));
        FindByIssuerPara.cbSize = sizeof(FindByIssuerPara);
        FindByIssuerPara.pszUsageIdentifier = szOID_PKIX_KP_CLIENT_AUTH;
        FindByIssuerPara.dwKeySpec = 0;
        FindByIssuerPara.cIssuer   = IssuerListInfo.cIssuers;
        FindByIssuerPara.rgIssuer  = IssuerListInfo.aIssuers;
        pChainContext = NULL;
        hCertStore = CertOpenSystemStore(0, _T("MY"));
        if(hCertStore == NULL)
        {
            throw new Common::Exceptions::PKIException(String::Concat(S"Помилка
відкриття запам’ятовуваних  MY Certificate. Помилка: ",
Convert::ToString((unsigned int)GetLastError())));
        }
        while(TRUE)
        {
            // Пошук сертифіката.
            pChainContext = CertFindChainInStore(hCertStore,
                                                 X509_ASN_ENCODING,
                                                 0,
                                                 CERT_CHAIN_FIND_BY_ISSUER,
                                                 &FindByIssuerPara,
                                                 pChainContext);
            if(pChainContext == NULL)
            {
                break;
            }

            // Get pointer to leaf certificate context.
            pCertContext = pChainContext->rgpChain[0]->rgpElement[0]-
>pCertContext;

            // Створення канального автентифікатора.
            m_pSChannelCred->cCreds = 1;
            m_pSChannelCred->paCred = certhash == NULL? NULL:&pCertContext;
            DWORD dwLen =0;
            if(certhash != NULL &&
CertGetCertificateContextProperty(pCertContext, CERT_HASH_PROP_ID, NULL,
&dwLen))
            {
               if(dwLen != certhash->Length)
               {
                    continue;
               }
               void* pCertHash = malloc(dwLen);
               if(pCertHash == NULL)
                   throw new OutOfMemoryException();
               if(CertGetCertificateContextProperty(pCertContext,
CERT_HASH_PROP_ID, pCertHash, &dwLen))
               {
                    void* pCertHashGiven =malloc(certhash->Length);
                    if(pCertHashGiven == NULL)
                    {
                        free(pCertHash);
                        throw new OutOfMemoryException();
                    }
                    Marshal::Copy(certhash, 0, pCertHashGiven, certhash-
>Length);
                    if(memcmp(pCertHashGiven, pCertHash, certhash->Length) != 0)
                    {
                        free(pCertHashGiven);
                        free(pCertHash);
                        continue;
```

```
                    }
                    free(pCertHashGiven);
                    free(pCertHash);
                }

            }

            Status = m_pSecurityFunc->AcquireCredentialsHandleA(
                                NULL,                   // Найменуваня
автентифікатору
                                UNISP_NAME_A,            // Найменування пакету
                                SECPKG_CRED_OUTBOUND,   // Прапор використання
                                NULL,                   // Крапка підключення ID
                                m_pSChannelCred,          // Пакет спеціальних
даних
                                NULL,                   // Вказник на GetKey()
                                NULL,                   // Зашифровані значення
в GetKey()
                                &hCreds,                // (out) Рукопотискання
                                &tsExpiry);             // (out) Час життя
(опція)
            if(Status != SEC_E_OK)
            {
                continue;
            }
            // Знищуємо старі автентифікатори.
            CertFreeCertificateChain(pChainContext);
            m_pSecurityFunc->FreeCredentialsHandle(m_phClientCreds);
            *m_phClientCreds = hCreds;
            break;
        }
        CertCloseStore(hCertStore, 0);
        hCertStore = NULL;
    }
    DWORD PKIConnection::GetMaxChunkSize(SecPkgContext_StreamSizes& Sizes)
    {
        SECURITY_STATUS scRet = m_pSecurityFunc-
>QueryContextAttributesA(m_phContext,SECPKG_ATTR_STREAM_SIZES,&Sizes);
        if(scRet != SEC_E_OK)
        {
            throw new Common::Exceptions::PKIException(S"Максимальни PKI розмір
помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    bool PKIConnection::Disconnect(Object* state)
    {
        //
        // Увідомлення каналу про закінчення сеансу зв’язку .
        //

        DWORD dwType = SCHANNEL_SHUTDOWN;
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, false);
        OutBuffer.SetSecurityBufferToken(0, &dwType, sizeof(dwType));

        SECURITY_STATUS Status = m_pSecurityFunc->ApplyControlToken(m_phContext,
&OutBuffer);

        if(FAILED(Status))
        {
            throw new Common::Exceptions::PKIException(S"Помлка з’єднання.
Помилка: ", Status);
        }
        //
        // Будуємо повідомлення про закриття PKI
        //
        DWORD dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT   |
                            ISC_REQ_REPLAY_DETECT     |
                            ISC_REQ_CONFIDENTIALITY   |
```

```
                                ISC_RET_EXTENDED_ERROR    |
                                ISC_REQ_ALLOCATE_MEMORY   |
                                ISC_REQ_STREAM;

        OutBuffer.SetSecurityBufferToken(0, NULL, 0);
        TimeStamp tsExpiry;
        DWORD     dwSSPIOutFlags;
        Status = m_pSecurityFunc->InitializeSecurityContextA(
                        m_phClientCreds,
                        m_phContext,
                        NULL,
                        dwSSPIFlags,
                        0,
                        SECURITY_NATIVE_DREP,
                        NULL,
                        0,
                        m_phContext,
                        &OutBuffer,
                        &dwSSPIOutFlags,
                        &tsExpiry);

        if(FAILED(Status))
        {
            throw new Common::Exceptions::PKIException("Помилка в відключенні
InitializeSecurityContext.");
        }

        char* pbMessage = static_cast<char*>(OutBuffer[0].pvBuffer);
        DWORD cbMessage = OutBuffer[0].cbBuffer;
        //
        // Відправляємо повідомлення про закриття сервер.
        //
        if(pbMessage != NULL && cbMessage != 0)
        {
            bool bRead = DispatchSend(pbMessage, cbMessage, state);
            if(!bRead)
            {
                throw new Common::Exceptions::PKISendException(S"Відправлення на
сервер не відбулося.");
            }

            // Звільняємо вихідний буфер.
                m_pSecurityFunc->FreeContextBuffer(pbMessage);
        }

        if(SecIsValidHandle(m_phContext))
        {
            Dispose();
        }
        return true;
    }

    void PKIConnection::EncryptSend(Byte data[], int ActualLen, Object* state)
    {
        SecPkgContext_StreamSizes Sizes;
          int IoBufferLength = GetMaxChunkSize(Sizes);
        IoBufferLength += Sizes.cbHeader + Sizes.cbTrailer;
#ifdef _DEBUG
        if(GetMaxChunkSize(Sizes) < (DWORD)ActualLen)
            throw new Common::Exceptions::PKIException("Визначенний розмір
недійсний.");
#endif

        CAutoSecBuffer<4> Buffers(m_pSecurityFunc, false);

            char* pbIoBuffer = (char*)malloc(IoBufferLength);
        if(pbIoBuffer == NULL)
            throw new OutOfMemoryException();
```

```
        Marshal::Copy(data, 0, pbIoBuffer + Sizes.cbHeader, ActualLen);

        Buffers.SetSecurityBufferStreamHeader(0, pbIoBuffer, Sizes.cbHeader);
        Buffers.SetSecurityBufferData(1, pbIoBuffer + Sizes.cbHeader,
ActualLen);
        Buffers.SetSecurityBufferStreamTrailer(2, pbIoBuffer + Sizes.cbHeader +
ActualLen, Sizes.cbTrailer);
        Buffers.SetSecurityBufferEmpty(3);

        SECURITY_STATUS scRet = m_pSecurityFunc->EncryptMessage(m_phContext, 0,
&Buffers, 0);

            if(FAILED(scRet) && scRet != SEC_E_CONTEXT_EXPIRED)
            {
            free(pbIoBuffer);
                throw new Common::Exceptions::PKIException(S"EncryptMessage
помилкове. Помилка: ", scRet);
            }

            int OutBufferLen =
Buffers[0].cbBuffer+Buffers[1].cbBuffer+Buffers[2].cbBuffer;

        if(!DispatchSend(static_cast<char*>(pbIoBuffer), OutBufferLen, state))
        {
            free(pbIoBuffer);
            throw new Common::Exceptions::PKISendException(S"Відправлення
помилкове. Помилка: ", scRet);
        }

        free(pbIoBuffer);
    }

    void PKIConnection::DecryptData(Byte data[], Int32 ActualLen, Object* state)
    {
        SecBuffer ExtraBuffer={0};
        //Додаємо попередній відкладений буфер
        ActualLen += m_SecExtraBuffer.cbBuffer;
        BYTE* pReadBuff = (BYTE*)malloc(ActualLen);
        if(pReadBuff == NULL)
            new OutOfMemoryException();
        //копіюємо з managed до unmanaged, в позиції після додаткових даних,
якщо є
        Marshal::Copy(data, 0, pReadBuff+m_SecExtraBuffer.cbBuffer, ActualLen-
m_SecExtraBuffer.cbBuffer);
        if(m_SecExtraBuffer.cbBuffer > 0)
        {
            //копія з попередніх відкладених даних до початку/ перед новим
            MoveMemory(pReadBuff, m_SecExtraBuffer.pvBuffer,
m_SecExtraBuffer.cbBuffer);
            free(m_SecExtraBuffer.pvBuffer);
            m_SecExtraBuffer.cbBuffer = 0;
            m_SecExtraBuffer.pvBuffer = NULL;
        }
        if(m_bInHandShake)
        {
            if(!ClientHandshakeLoop(pReadBuff, ActualLen, &ExtraBuffer, state))
            {
                // Вхідний буфер містить тільки фрагмент закодованих даних.
Необхідно зберегти фрагмент й чекати надходження наступних даних..
                m_SecExtraBuffer.pvBuffer = pReadBuff;
                m_SecExtraBuffer.cbBuffer = ActualLen;
                return;
            }
            if(ExtraBuffer.cbBuffer == 0)
            {
                free(pReadBuff);
                return;
            }
            else if(ExtraBuffer.pvBuffer)
```

```
            {
                //зберігаємо закодовані дані та пересилаємо їх для декодування
повідомлення
                MoveMemory(pReadBuff, ExtraBuffer.pvBuffer,
ExtraBuffer.cbBuffer);
                ActualLen = ExtraBuffer.cbBuffer;
                free(ExtraBuffer.pvBuffer);
                ExtraBuffer.pvBuffer = NULL;
                ExtraBuffer.cbBuffer =0;
            }
        }

        while(true)
        {
            CAutoSecBuffer<4> Buffers(m_pSecurityFunc, false);
            Buffers.SetSecurityBufferData(0, pReadBuff, ActualLen);
            Buffers.SetSecurityBufferEmpty(1);
            Buffers.SetSecurityBufferEmpty(2);
            Buffers.SetSecurityBufferEmpty(3);

            SECURITY_STATUS scRet = m_pSecurityFunc->DecryptMessage(m_phContext,
&Buffers, 0, NULL);

            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
                Вхідний буфер містить тільки фрагмент закодованих даних.
Необхідно зберегти фрагмент та чекати надходження наступних даних
                //
                m_SecExtraBuffer.pvBuffer = pReadBuff;
                m_SecExtraBuffer.cbBuffer = ActualLen;
                //pReadBuff визволяється на наступному вході
                return;
            }
            if( scRet != SEC_E_OK &&
                scRet != SEC_I_RENEGOTIATE &&
                scRet != SEC_I_CONTEXT_EXPIRED)
            {
                free(pReadBuff);
                throw new Common::Exceptions::PKIException("Помилка
дешифрування. Помилка: ", scRet);
            }

            // Сервер посилає повідомлення про кінець сессії
            if(scRet == SEC_I_CONTEXT_EXPIRED)
                {
                //шифруємо пусті буфери й посилаємо їх в відповідності зі
специфікацією
                EncryptSend(new Byte[0], 0, state);
                Dispose();
                free(pReadBuff);
                throw new Common::Exceptions::PKIException("Помилка
дешифрування. Контекст закінчений.");
                }

            // Локальні дані й зовнішній (опціонально) буфер.
                SecBuffer* pDataBuffer = NULL;
                SecBuffer* pExtraBuffer=NULL;
            for(int i = 1; i < 4; i++)
            {
                if(pDataBuffer == NULL && Buffers[i].BufferType ==
SECBUFFER_DATA)
                {
                    pDataBuffer = &Buffers[i];
                }
                if(pExtraBuffer == NULL && Buffers[i].BufferType ==
SECBUFFER_EXTRA)
                {
                    pExtraBuffer = &Buffers[i];
                }
```

```
            }

            // Декодування даних.
            if(pDataBuffer && pDataBuffer->cbBuffer > 0)
            {
                DispatchPlainData(pDataBuffer->pvBuffer, pDataBuffer->cbBuffer,
state);
            }

            // Переміщуємо додаткові дані на вхідний буфер, коректуємо довжину
та оброблюємо знову
            if(pExtraBuffer != NULL)
            {
                MoveMemory(pReadBuff, pExtraBuffer->pvBuffer, pExtraBuffer-
>cbBuffer);
                ActualLen = pExtraBuffer->cbBuffer;
            }
            else if(scRet == S_OK)
                    break;
            if(scRet == SEC_I_RENEGOTIATE)
            {
                // Сервер вільний для виконання іншого рукопотискння
                DoRenegotiate(this);
                m_bInHandShake=true;
                int dummy =0;
                if(pExtraBuffer != NULL)
                    ClientHandshakeLoop(pReadBuff, ActualLen, &ExtraBuffer,
state);
                else
                    ClientHandshakeLoop(NULL, dummy, &ExtraBuffer, state);
                // Переміщуємо інші зовнішні данні до вхідного буферу.
                if(ExtraBuffer.pvBuffer != NULL)
                {
                    MoveMemory(pReadBuff, ExtraBuffer.pvBuffer,
ExtraBuffer.cbBuffer);
                    ActualLen = ExtraBuffer.cbBuffer;
                    free(ExtraBuffer.pvBuffer);
                    ExtraBuffer.pvBuffer = NULL;
                    ExtraBuffer.cbBuffer =0;
                }
                else
                    break;
            }
        }
          free(pReadBuff);
    }

    bool PKIConnection::DispatchSend(const char* pbMessage, DWORD cbMessage,
Object* state)
    {
        Byte data[] = new Byte[cbMessage];
        Marshal::Copy(IntPtr((void*)pbMessage), data, 0, cbMessage);
        return DoWrite(data, state);
    }
    void PKIConnection::DispatchPlainData(void* pData, long Len, Object* state)
    {
        Byte data[] = new Byte[Len];
        Marshal::Copy(IntPtr(pData), data, 0, Len);
        DoPlainData(data, state);
    }
    void PKIConnection::Init()
    {
        if(m_pSecurityFunc != NULL)
            return;
        m_SecExtraBuffer.BufferType = -1;
        m_SecExtraBuffer.cbBuffer = 0;
        m_SecExtraBuffer.pvBuffer = NULL;
        m_bInHandShake = false;
        try
```

```
        {
            m_pSChannelCred = __nogc new SCHANNEL_CRED();
            m_phClientCreds = __nogc new CredHandle();
            m_phContext     = __nogc new CtxtHandle();
        }
        catch(const std::bad_alloc&)
        {
            throw new OutOfMemoryException();
        }
        SecInvalidateHandle(m_phClientCreds);
        SecInvalidateHandle(m_phContext);
        memset(m_pSChannelCred, 0, sizeof(SCHANNEL_CRED));
        m_pSecurityFunc = NULL;
        m_hSecurity = NULL;
        SecurityFunctionTable* pSecurityFunc = m_pSecurityFunc;
        if(!LoadSecurityLibrary(m_hSecurity, pSecurityFunc))
            throw new Common::Exceptions::PKIException(S"Failed to load security
dll.");
        m_pSecurityFunc = pSecurityFunc;
        m_ServerIP = NULL;
    }

    void PKIConnection::SetupCredentials(Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot)
    {
        TimeStamp       tsExpiry;
        SECURITY_STATUS Status;
        HCERTSTORE      hCertStore;
        PCCERT_CONTEXT  pCertContext = NULL;

        // Відкриваємо "MY" записи сертифікатів, в Internet Explorer  записах
сертифікатів цього Інтернет клієнту.
        hCertStore = CertOpenSystemStore(0, _T("MY"));
        if(hCertStore == NULL)
        {
           throw new Common::Exceptions::PKIException(String::Concat(S"Помика
відкриття запису MY Certificate. Помилка: ", Convert::ToString((unsigned
int)GetLastError())));
        }

        if(thumbPrint != NULL && thumbPrint->Length > 0)
        {
            int HashLen = thumbPrint->Length;
            BYTE* pbData = (BYTE*)malloc(HashLen);
            Marshal::Copy(thumbPrint, 0, pbData, HashLen);
            CRYPT_HASH_BLOB hash={HashLen, pbData};
            SetLastError(0);
            pCertContext = CertFindCertificateInStore(hCertStore,
                                                      X509_ASN_ENCODING,
                                                      0,
                                                      CERT_FIND_HASH,
                                                      &hash,
                                                      NULL);
            free(pbData);
            Status = GetLastError();
            if(pCertContext == NULL)
            {
                CertCloseStore(hCertStore, 0);
                hCertStore = NULL;
                throw new
Common::Exceptions::PKIException(String::Concat(S"Помилка відповідності
інформації сертифікату. Помилка: ", Convert::ToString((unsigned int)Status)));
            }
        }
        m_pSChannelCred->dwVersion  = SCHANNEL_CRED_VERSION;
        if(pCertContext != NULL)
        {
            m_pSChannelCred->cCreds     = 1;
            m_pSChannelCred->paCred     = &pCertContext;
```

```
        }
        m_pSChannelCred->grbitEnabledProtocols = prot;
        m_pSChannelCred->dwFlags |=
SCH_CRED_NO_DEFAULT_CREDS|SCH_CRED_MANUAL_CRED_VALIDATION;

        Status = m_pSecurityFunc->AcquireCredentialsHandleA( NULL,
// Найменуваня автентифікатору
                                                             UNISP_NAME_A,
// Найменування пакету

SECPKG_CRED_OUTBOUND,   // Прапор використання
                                                             NULL,
// Крапка підключення ID
                                                             m_pSChannelCred,
// Пакет спеціальних даних
                                                             NULL,
// Вказник на GetKey()
                                                             NULL,
// Зашифровані значення в GetKey()
                                                m_phClientCreds,        // (out)
Рукопотискання
                                                             &tsExpiry);
// (out) Час життя (опція)
        if(Status != SEC_E_OK)
        {
            if(pCertContext != NULL)
            {
                CertCloseStore(hCertStore, 0);
                hCertStore = NULL;
                CertFreeCertificateContext(pCertContext);
            }
            throw new Common::Exceptions::PKIException(String::Concat(S"Помилка
створення автентифікатору. Помилка: ", Convert::ToString((int)Status)));
        }
        //
        // Звільнення контексту сертифікату. Копія була створенна у каналі.
        //
        if(pCertContext != NULL)
        {
            CertCloseStore(hCertStore, 0);
            CertFreeCertificateContext(pCertContext);
            pCertContext = NULL;
        }
    }
    void PKIConnection::Dispose(bool disposing)
    {
        m_ServerIP = NULL;
        DoWrite=NULL;
        DoPlainData=NULL;
        DoRenegotiate=NULL;
        DoServerCertVerify=NULL;
        DoHandShakeSuccess=NULL;
        m_bInHandShake = false;
        if(m_SecExtraBuffer.cbBuffer > 0)
            free(m_SecExtraBuffer.pvBuffer);
        m_SecExtraBuffer.cbBuffer = 0;
        m_SecExtraBuffer.pvBuffer = NULL;

        if(SecIsValidHandle(m_phContext))
        {
            m_pSecurityFunc->DeleteSecurityContext(m_phContext);
            SecInvalidateHandle(m_phContext);
        }
        if(SecIsValidHandle(m_phClientCreds))
        {
            m_pSecurityFunc->FreeCredentialsHandle(m_phClientCreds);
            SecInvalidateHandle(m_phClientCreds);
        }
```

```
            if(!disposing)
            {
                delete m_pSChannelCred;
                delete m_phClientCreds;
                delete m_phContext;
                m_pSChannelCred = NULL;
                m_phClientCreds = NULL;
                m_phContext = NULL;
                if(m_hSecurity != NULL)
                {
                    FreeLibrary(m_hSecurity);
                    m_hSecurity = NULL;
                    m_pSecurityFunc = NULL;
                }
            }
        }
    }
    }
```

Файл PKI.h – бібліотека для файлу PKI.cpp

```
// PKI.h
/**************************************************************************

#pragma once
#include "PKIcommon.h"

namespace PKI
{
    public __value struct ServerCeriticateInfo;
    namespace Client
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WritePKI(Byte data[], Object* state);
        //Процес дешифрування даних
        public __delegate void PlainData(Byte data[], Object* state);
        //Інформація сертифікату серверу (опціонально)
        public __delegate void VerifyServCert(Common::Misc::CeriticateInfo
ServCertInfo);
        //Рукопотискання с сервером відбулося
        public __delegate void HandShakeSuccess();

        __sealed public __gc class PKIConnection;
        //Сервер потребує узгодження, новий сертифікат завантаження, PKIConn-
>LoadNewClientCredentials
        public __delegate void NewCertificate(PKIConnection* PKIConn);

        __sealed public __gc class PKIConnection : public IDisposable
          {
        public:

            PKIConnection();
            ~PKIConnection()
            {
                Dispose(false);
            }
        public:
            //ініціалізація з'єднання, береться ip-адреса сервера і клієнтський
хеш сертифікату,
            //данні які потрібно відіслати були  повернені з WritePKI
            void InitiateHandShake(String* ipAddress, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state);
            //шифруємо дані, шифруємі дані повертаються в WritePKI
            void EncryptSend(Byte data[], int ActualLen, Object* state);
            // дешифруємо дані, дешифруємі дані повертаються в PlainData
            void DecryptData(Byte data[], Int32 ActualSize, Object* state);
            //роз'єднання з сервером
            bool Disconnect(Object* state);
            //очищення
            void Dispose() { Dispose(true); GC::SuppressFinalize(this);}
            //завантажуємо нові автентифікатори клєнта з NewCertificate
            void LoadNewClientCredentials(Byte sha1hash[]);
        public:
            //максимальний розмір даних для посилки/отримання за один раз
            __property int get_MaxDataChunkSize()
            {
                SecPkgContext_StreamSizes notused; return
GetMaxChunkSize(notused);
            }
            //рекомендований початковий розмір, коли починається рукопотискання.
Це максимальний розмір токену автентифікації
            __property int get_MaxInitialChunkSize()
            {
                PSecPkgInfo psecInfo;
                SECURITY_STATUS scRet = QuerySecurityPackageInfo(UNISP_NAME,
&psecInfo);
                if (scRet != SEC_E_OK)
```

```
                    throw new Common::Exceptions::PKIException(S"Взятий
максимальний розмір токену PKI помилковий. Помилка: ", scRet);
                return psecInfo->cbMaxToken;
            }
        public:
            //Повертаємо зашифровані дані
            WritePKI*         DoWrite;
            //Повертаємо розшифровані дані
            PlainData*        DoPlainData;
            //Опціонально
            NewCertificate*   DoRenegotiate;
            //Опціонально
            VerifyServCert*   DoServerCertVerify;
            //Опціонально
            HandShakeSuccess* DoHandShakeSuccess;

////////////////////////////////////////////////////////////////////////////
        private:
            void Init();
            void SetupCredentials(Byte[],
Common::Misc::SecurityProviderProtocol);
            void PerformHandShake(Object* state);
            bool ClientHandshakeLoop(void*, int&, SecBuffer*, Object* state);
            bool DispatchSend(const char*, DWORD, Object* state);
            void DispatchPlainData(void*, long, Object* state);
            DWORD GetMaxChunkSize(SecPkgContext_StreamSizes&);
            void Dispose(bool);
        private:
            bool            m_bInHandShake;
            String*         m_ServerIP;
            int             m_Port;
            SCHANNEL_CRED __nogc*   m_pSChannelCred;
            CredHandle __nogc*           m_phClientCreds;
            CtxtHandle __nogc*           m_phContext;
            SecurityFunctionTable __nogc* m_pSecurityFunc;
            HMODULE   m_hSecurity;
            SecBuffer m_SecExtraBuffer;
          };
    }
}
```

Файл PKIcommon.cpp – організація шифрування та формування ланцюжка сертифікатів

```
#include "stdafx.h"
#include "PKIcommon.h"


bool LoadSecurityLibrary(HMODULE hSecurity, SecurityFunctionTable __nogc*&
pSecurityFunc)
{
    if(hSecurity != NULL)
        return true;
    INIT_SECURITY_INTERFACE pInitSecurityInterface;
    OSVERSIONINFO VerInfo;
    char lpszDLL[MAX_PATH];

    //
    //  пошук бібліотек захисту DLL які використовують
    //  включення до Win2k, NT або Win9x
    //

    VerInfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
    if (!GetVersionEx (&VerInfo))
    {
        return false;
    }

    if (VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT
        && VerInfo.dwMajorVersion == 4)
    {
        strcpy (lpszDLL, "Security.dll" );
    }
    else if (VerInfo.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS ||
        VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT )
    {
        strcpy (lpszDLL, "Secur32.dll" );
    }
    else
    {
        return false;
    }

    //
    //  Завантажуємо  DLL зашифрування
    //

    hSecurity = LoadLibrary(lpszDLL);
    if(hSecurity == NULL)
    {
        return false;
    }

    pInitSecurityInterface = (INIT_SECURITY_INTERFACE)GetProcAddress(
                                    hSecurity,
                                    "InitSecurityInterfaceA");

    if(pInitSecurityInterface == NULL)
    {
        FreeLibrary(hSecurity);
        hSecurity = NULL;
        return false;
    }

    pSecurityFunc = pInitSecurityInterface();

    if(pSecurityFunc == NULL)
    {
        FreeLibrary(hSecurity);
```

```
        hSecurity = NULL;
        return false;
    }
    return true;
}

bool VerifyCertificate(bool fTargetServer, SecurityFunctionTable __nogc*
pSecurityFunc, CtxtHandle __nogc* phContext, LPWSTR pwszServerName, DWORD
dwCertFlags, PKI::Common::Misc::CeriticateInfo* CertInfo)
{
    HTTPSPolicyCallbackData  polHttps;
    CERT_CHAIN_POLICY_PARA   PolicyPara;
    CERT_CHAIN_POLICY_STATUS PolicyStatus;
    CERT_CHAIN_PARA          ChainPara;
    PCCERT_CHAIN_CONTEXT     pChainContext = NULL;
    PCCERT_CONTEXT                  pServerCert = NULL;

    SECURITY_STATUS Status = pSecurityFunc->QueryContextAttributesA(phContext,

      SECPKG_ATTR_REMOTE_CERT_CONTEXT,
                                                    (PVOID)&pServerCert);
    if(Status != SEC_E_OK || pServerCert == NULL)
    {
        return false;
    }
    //
    // Будуємо ланцюжок сертифікатів .
    //
    ZeroMemory(&ChainPara, sizeof(ChainPara));
    ChainPara.cbSize = sizeof(ChainPara);
    ChainPara.RequestedUsage.dwType = USAGE_MATCH_TYPE_OR;

    LPSTR ServerUsages[] = {  szOID_PKIX_KP_SERVER_AUTH,
                              szOID_SERVER_GATED_CRYPTO,
                              szOID_SGC_NETSCAPE };
    LPSTR ClientUsage = szOID_PKIX_KP_CLIENT_AUTH;

    ChainPara.RequestedUsage.Usage.cUsageIdentifier     = fTargetServer?3:1;
    ChainPara.RequestedUsage.Usage.rgpszUsageIdentifier =
fTargetServer?ServerUsages:&ClientUsage;

    if(!CertGetCertificateChain(NULL,pServerCert,NULL,
                                pServerCert->hCertStore,&ChainPara,
                                0,NULL,&pChainContext))
    {
        if(pChainContext)
        {
            CertFreeCertificateChain(pChainContext);
        }
    }
    //
    // Перевіряємо ланцюжок сертифікатів .
    //
    ZeroMemory(&polHttps, sizeof(HTTPSPolicyCallbackData));
    polHttps.cbStruct           = sizeof(HTTPSPolicyCallbackData);
    polHttps.dwAuthType         = fTargetServer?AUTHTYPE_SERVER:AUTHTYPE_CLIENT;
    polHttps.fdwChecks          = dwCertFlags;
    polHttps.pwszServerName     = pwszServerName;
    memset(&PolicyPara, 0, sizeof(PolicyPara));
    PolicyPara.cbSize            = sizeof(PolicyPara);
    PolicyPara.pvExtraPolicyPara = &polHttps;
    memset(&PolicyStatus, 0, sizeof(PolicyStatus));
    PolicyStatus.cbSize = sizeof(PolicyStatus);
    if(!CertVerifyCertificateChainPolicy(
                            CERT_CHAIN_POLICY_PKI,
                            pChainContext,
                            &PolicyPara,
                            &PolicyStatus))
    {
```

```
        if(pChainContext)
        {
            CertFreeCertificateChain(pChainContext);
            pChainContext = NULL;
        }
    }
    bool    bRet  =     true;
    if(CertInfo != NULL && pChainContext != NULL)
    {
        Byte CertData[] = new Byte[pServerCert->cbCertEncoded];
        Marshal::Copy(IntPtr(pServerCert->pbCertEncoded), CertData, 0,
pServerCert->cbCertEncoded);
        CertInfo->PolStatus =
PKI::Common::Misc::ServerCertChainPolicyStatus(PolicyStatus.dwError);
        CertInfo->CertEncodingType = pServerCert->dwCertEncodingType;
        CertInfo->CertData = CertData;
          bRet    =     false;
    }
    if(pChainContext)
    {
        CertFreeCertificateChain(pChainContext);
    }
    return bRet;
}
```

Файл PKIcommon.h – бібліотека для файлу PKIcommon.cpp

```
#pragma once
#using <mscorlib.dll>
#include <windows.h>
#include <wincrypt.h>
#include <schannel.h>
#define SECURITY_WIN32
#include <security.h>
#include <sspi.h>
#include "tchar.h"

using namespace System;
using namespace System::Net;
using namespace System::Runtime::InteropServices;

#pragma comment(lib, "Crypt32")
#pragma comment(lib, "Secur32")

namespace PKI
{
namespace Common
{
    namespace Exceptions
    {
    [Serializable]
      public __gc class PKIException : public ApplicationException
    {
    public:
        PKIException(){}
        PKIException(String* message):ApplicationException(message){}
        PKIException(String* message, Exception* innerException)
            :ApplicationException(message,innerException){}
        PKIException(String* message, UInt32 error)
            :ApplicationException(String::Concat(message,
Convert::ToString(error))){}

    };
    [Serializable]
    __sealed public __gc class PKIServerFailedToFindExistingClientID : public
PKIException
    {
    public:
        PKIServerFailedToFindExistingClientID(){}
        PKIServerFailedToFindExistingClientID(String*
message):PKIException(message){}
        PKIServerFailedToFindExistingClientID(String* message, Exception*
innerException)
            :PKIException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class PKIServerDisconnectedException : public
PKIException
    {
    public:
        PKIServerDisconnectedException(){}
        PKIServerDisconnectedException(String* message):PKIException(message){}
        PKIServerDisconnectedException(String* message, Exception*
innerException)
            :PKIException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class PKIReadException : public PKIException
    {
    public:
        PKIReadException(){}
```

```
    PKIReadException(String* message):PKIException(message){}
    PKIReadException(String* message, Exception* innerException)
        :PKIException(message,innerException){}
};

[Serializable]
__sealed public __gc class PKISendException : public PKIException
{
public:
    PKISendException(){}
    PKISendException(String* message):PKIException(message){}
    PKISendException(String* message, Exception* innerException)
        :PKIException(message,innerException){}
    PKISendException(String* message, UInt32 error)
        :PKIException(String::Concat(message, Convert::ToString(error))){}
};
} //простір імен Exeptions
namespace Misc
{
public __value enum ServerCertChainPolicyStatus
{
    CERT_OK                    =S_OK,
    TRUST_NOSIGNATURE          =TRUST_E_NOSIGNATURE,
    CERT_EXPIRED               =CERT_E_EXPIRED,
    CERT_VALIDITYPERIODNESTING=CERT_E_VALIDITYPERIODNESTING,
    CERT_ROLE                  =CERT_E_ROLE,
    CERT_PATHLENCONST          =CERT_E_PATHLENCONST,
    CERT_CRITICAL              =CERT_E_CRITICAL,
    CERT_PURPOSE               =CERT_E_PURPOSE,
    CERT_ISSUERCHAINING        =CERT_E_ISSUERCHAINING,
    CERT_MALFORMED             =CERT_E_MALFORMED,
    CERT_UNTRUSTEDROOT         =CERT_E_UNTRUSTEDROOT,
    CERT_CHAINING              =CERT_E_CHAINING,
    TRUST_FAIL                 =TRUST_E_FAIL,
    CERT_REVOKED               =CERT_E_REVOKED,
    CERT_UNTRUSTEDTESTROOT     =CERT_E_UNTRUSTEDTESTROOT,
    CERT_REVOCATION_FAILURE    =CERT_E_REVOCATION_FAILURE,
    CERT_CN_NO_MATCH           =CERT_E_CN_NO_MATCH,
    CERT_WRONG_USAGE           =CERT_E_WRONG_USAGE,
    TRUST_EXPLICIT_DISTRUST    =TRUST_E_EXPLICIT_DISTRUST,
    CERT_UNTRUSTEDCA           =CERT_E_UNTRUSTEDCA,
    CERT_INVALID_POLICY        =CERT_E_INVALID_POLICY,
    CERT_INVALID_NAME          =CERT_E_INVALID_NAME
};
public __value struct CeriticateInfo
{
    ServerCertChainPolicyStatus PolStatus;
    long                        CertEncodingType;
    Byte                        CertData[];
};
public __value enum SecurityProviderProtocol
{
    PROT_PKI3 = SP_PROT_PKI3,
    PROT_TLS1 = SP_PROT_TLS1,
    PROT_NONE = SP_PROT_NONE
};
public __value enum InitializeSecurityContextRequirements
{
    ISCREQ_DELEGATE              =ISC_REQ_DELEGATE,
    ISCREQ_MUTUAL_AUTH           =ISC_REQ_MUTUAL_AUTH,
    ISCREQ_REPLAY_DETECT         =ISC_REQ_REPLAY_DETECT,
    ISCREQ_SEQUENCE_DETECT       =ISC_REQ_SEQUENCE_DETECT,
    ISCREQ_CONFIDENTIALITY       =ISC_REQ_CONFIDENTIALITY,
    ISCREQ_USE_SESSION_KEY       =ISC_REQ_USE_SESSION_KEY,
    ISCREQ_PROMPT_FOR_CREDS      =ISC_REQ_PROMPT_FOR_CREDS,
    ISCREQ_USE_SUPPLIED_CREDS    =ISC_REQ_USE_SUPPLIED_CREDS,
    ISCREQ_ALLOCATE_MEMORY       =ISC_REQ_ALLOCATE_MEMORY,
    ISCREQ_USE_DCE_STYLE         =ISC_REQ_USE_DCE_STYLE,
    ISCREQ_DATAGRAM              =ISC_REQ_DATAGRAM,
```

```
        ISCREQ_CONNECTION            =ISC_REQ_CONNECTION,
        ISCREQ_CALL_LEVEL            =ISC_REQ_CALL_LEVEL,
        ISCREQ_FRAGMENT_SUPPLIED     =ISC_REQ_FRAGMENT_SUPPLIED,
        ISCREQ_EXTENDED_ERROR        =ISC_REQ_EXTENDED_ERROR,
        ISCREQ_STREAM                =ISC_REQ_STREAM,
        ISCREQ_INTEGRITY             =ISC_REQ_INTEGRITY,
        ISCREQ_IDENTIFY              =ISC_REQ_IDENTIFY,
        ISCREQ_NULL_SESSION          =ISC_REQ_NULL_SESSION,
        ISCREQ_MANUAL_CRED_VALIDATION=ISC_REQ_MANUAL_CRED_VALIDATION,
        ISCREQ_RESERVED1             =ISC_REQ_RESERVED1,
        ISCREQ_FRAGMENT_TO_FIT       =ISC_REQ_FRAGMENT_TO_FIT
    };
    } //namespace Misc
} //namespace Common
} // namespace PKI
#pragma unmanaged
template<int nBuffs>
class CAutoSecBuffer : public SecBufferDesc
{
    SecurityFunctionTable* m_pSecurityFunc;
    SecBuffer   m_SecBuffer[nBuffs];
    bool        m_bRelease;

public:
    CAutoSecBuffer(SecurityFunctionTable* pSecurityFunc, bool bRelease)
        :m_pSecurityFunc(pSecurityFunc), m_bRelease(bRelease)
    {
        cBuffers = nBuffs;
        pBuffers = m_SecBuffer;
        ulVersion = SECBUFFER_VERSION;
    }
    ~CAutoSecBuffer()
    {
        if(m_bRelease)
        {
            for(int i=0; i<nBuffs; ++i)
            {
                m_pSecurityFunc->FreeContextBuffer(m_SecBuffer[i].pvBuffer);
                m_SecBuffer[i].pvBuffer = NULL;
            }
        }

    }
    void FreeBuffer(int nBuff)
    {
        m_pSecurityFunc->FreeContextBuffer(m_SecBuffer[nBuff].pvBuffer);
        m_SecBuffer[nBuff].pvBuffer = NULL;
    }
    void SetSecurityBufferToken(int nBuff, void* pvBuffer, int nLen)
    {
        m_SecBuffer[nBuff].BufferType = SECBUFFER_TOKEN;
        m_SecBuffer[nBuff].cbBuffer = nLen;
        m_SecBuffer[nBuff].pvBuffer = pvBuffer;
    }
    void SetSecurityBufferData(int nBuff, void* pvBuffer, int nLen)
    {
        m_SecBuffer[nBuff].BufferType = SECBUFFER_DATA;
        m_SecBuffer[nBuff].cbBuffer = nLen;
        m_SecBuffer[nBuff].pvBuffer = pvBuffer;
    }
    void SetSecurityBufferStreamHeader(int nBuff, void* pvBuffer, int nLen)
    {
        m_SecBuffer[nBuff].BufferType = SECBUFFER_STREAM_HEADER;
        m_SecBuffer[nBuff].cbBuffer = nLen;
        m_SecBuffer[nBuff].pvBuffer = pvBuffer;
    }
    void SetSecurityBufferStreamTrailer(int nBuff, void* pvBuffer, int nLen)
    {
        m_SecBuffer[nBuff].BufferType = SECBUFFER_STREAM_TRAILER;
```

```
        m_SecBuffer[nBuff].cbBuffer = nLen;
        m_SecBuffer[nBuff].pvBuffer = pvBuffer;
    }
    void SetSecurityBufferEmpty(int nBuff)
    {
        m_SecBuffer[nBuff].BufferType = SECBUFFER_EMPTY;
        m_SecBuffer[nBuff].cbBuffer   = 0;
        m_SecBuffer[nBuff].pvBuffer   = NULL;
    }
    SecBuffer& operator[](unsigned int i)
    {
        return m_SecBuffer[i];
    }
private:
};

struct WStringComp
{   // Визначення хеш-функції для строк
    enum { // Параметри для хеш-таблиці
    bucket_size = 4, // 0 < bucket_size
    min_buckets = 8}; // min_buckets = 2 ^^ N, 0 < N
    size_t operator()(const std::wstring& s1) const
    {
        const wchar_t *p = s1.c_str();
        size_t nHash = 0;
        while (*p != '\0')
            nHash = (nHash<<5) + nHash + (*p++);
        return nHash;
    }
    bool operator()(const std::wstring &s1, const std::wstring &s2) const
    { // тест якщо s1 завантажено поперед s2
        return (s1 < s2);
    }
};
#pragma managed
bool LoadSecurityLibrary(HMODULE, SecurityFunctionTable __nogc*&);
bool VerifyCertificate(bool fTargetServer, SecurityFunctionTable __nogc*,
CtxtHandle __nogc*, LPWSTR pwszServerName, DWORD dwCertFlags,
PKI::Common::Misc::CeriticateInfo* CertInfo);
```