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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI”

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

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2022 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

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

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

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

Шевченку Вадиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI*

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

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

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

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

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

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

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

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

6. Консультанти розділів роботи

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 6 » вересня 2022 р.

Підпис керівника
_____ Петренюк В.І.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2022 р.

Підпис здобувача
_____ Шевченко В.О.
(прізвище та ініціали)

АНОТАЦІЯ

Шевченко В.О. Дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

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

Метою розробки є дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

Об'єктом дослідження є процес доступу до хмарних сервісів з використанням технології PKI.

Предметом дослідження є методи доступу до хмарних сервісів з використанням технології PKI.

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

Результат роботи – програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

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

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

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

Програму розроблено в середовищі Visual C++.

Ключові слова: комп'ютерні науки, хмарні сервіси, PKI

ABSTRACT

Shevchenko V.O. Research and software implementation of the system of access to cloud services using PKI technology. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.

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

The purpose of the development is the research and software implementation of the system of access to cloud services using PKI technology.

The object of the study is the process of accessing cloud services using PKI technology.

The subject of research is methods of accessing cloud services using PKI technology.

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

The result of the work is the software implementation of the system of access to cloud services using PKI technology.

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

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

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

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

Keywords: computer science, cloud services, PKI

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. Сьогодні більшість компаній так чи інакше використовують Інтернет, і із цим зв'язані проблеми захисту віддалених і мобільних користувачів інформаційних систем компанії, захисту корпоративних хмарних сервісів (інтранет-сайту й будь-якого додатка компанії, що працює по http протоколу). Інтернет – це зона підвищеного ризику, відповідно, потрібні спеціальні засоби захисту при роботі віддалених користувачів з WEB-додатками по SSL-протоколу. Таким чином, підсистема захисту WEB-ресурсів вирішує наступні задачі:

– забезпечення єдиного інтерфейсу до додатків;

– інтегрований контроль доступу до корпоративних хмарних сервісів;

– захист клієнтських браузерів;

– захист хмарних сервісів.

Існує багато технологій захисту хмарних сервісів. У магістерському проекті пропонується система захисту основана на використанні протоколів SSL/TLS, які побудовані з використання інфраструктури відкритих ключів (PKI).

У протоколі SSL/TLS використовується ряд симметричних алгоритмів, асиметричних алгоритмів та геш-функцій. Тому одним із завдань, які потрібно вирішити у даному магістерському проекті є вибір того, або іншого алгоритму, які використовуються на різних етапах побудови системи захисту доступу до хмарних сервісів.

Технологія PKI дозволяє перевіряти й засвідчувати дійсність користувача. PKI забезпечує єдину ідентифікацію, автентифікацію й авторизацію користувачів системи, додатків і процесів і разом із цим гарантує доступність, цілісність і конфіденційність інформації.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

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

– Огляд існуючих систем доступу до хмарних сервісів з використанням технології PKI.

– Дослідження системи доступу до хмарних сервісів з використанням технології PKI.

– Програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

Об'єктом дослідження є процес доступу до хмарних сервісів з використанням технології PKI.

Предметом дослідження є методи доступу до хмарних сервісів з використанням технології PKI.

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

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

– Удосконалено метод доступу до хмарних сервісів з використанням технології PKI.

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

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі доступу до хмарних сервісів з використанням технології PKI.

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

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

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

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Процес глобалізації інформаційно-телекомунікаційних комплексів, впровадження на Інтернет-простір України телекомунікаційних технологій, реалізованих переважно на апаратно-програмних засобах закордонного виробництва, істотно загострили проблему залежності якості процесів транспортування інформації, від можливих навмисних і ненавмисних впливів порушника на передані дані користувача, інформацію керування й апаратно-програмні засоби, що забезпечують ці процеси.

Збільшення об'ємів збереженої й переданої інформації, територіальна розподіленість мереж зв'язку приводять до нарощування потенційних можливостей порушника по несанкціонованому доступу до інформаційної сфери Інтернету і впливу на процеси її функціонування.

Ускладнення застосовуваних технологій і процесів функціонування Інтернет приводить до того, що апаратно-програмні засоби, використовувані в Інтернет, об'єктивно можуть містити ряд помилок і недекларованих можливостей, які можуть бути використані порушниками.

Відсутність в Інтернет необхідних засобів захисту в умовах інформаційного протиборства робить WEB-ресурси в цілому уразливими від можливих ворожих акцій, несумлінної конкуренції операторів зв'язку, а також кримінальних і інших протиправних дій.

У зв'язку із цим з'явилися принципово нові задачі, пов'язані з необхідністю:

– захисту мереж (вузлів мережі й центра керування мережею) від несанкціонованого доступу користувачів до надаваних мережею послугам зв'язку;

– захисту системи керування хмарними сервісами (вузлів мережі й центра керування мережею) у тому числі підсистеми керування інформаційної безпеки від можливості несанкціонованого доступу до процесів керування по різних каналах доступу, із забезпеченням конфіденційності й цілісності інформації керування (команд, повідомлень і інформації маршрутизації), що циркулює в різних каналах зв'язку мережі, в умовах можливих навмисних впливів порушника;

– забезпечення заданого (гарантуємого мережею) якості процесу передачі даних користувача (наприклад, вірогідності переданих через мережу даних користувача) в умовах можливих навмисних впливів порушника (впливів порушника) на інформаційну сферу Інтернет;

– забезпечення своєчасного виявлення спроби блокування порушником процесу передачі даних користувача з локалізацією місця впливів порушника й ліквідацією в заданий час наслідку впливів порушника;

– захисту інформаційної сфери Інтернет від можливості активізації порушником шкідливих програм («закладок», «вірусів» і т.д.) за допомогою спеціальних команд, що посилаються їм по різним «легальним» і «нелегальним» каналах доступу (у тому числі й по каналах зв'язку при сполученні з іншими мережами зв'язку, зокрема, з мережею Інтернет).

– забезпечення можливості створення в Інтернет надійного шляху (тракту) транспортування інформації (даних) для певної групи користувачів, у тому числі в умовах навмисних впливів порушника на інформаційну сферу Інтернет.

Забезпечення транспортних функцій Інтернет в умовах можливих впливів порушника на її інформаційну сферу зв'язано зі специфічними проблемами, що ставляться, в основному, до організації ефективного керування (на базі моніторингу стану мереж зв'язку) і взаємодією окремих апаратно-програмних засобів зв'язку Інтернет, розосереджених на більших відстанях друг від друга. Транспортування повідомлень по каналах зв'язку за допомогою засобів зв'язку

Інтернет (маршрутизаторів, комутаторів, центрів комутації пакетів і т.д.) пов'язана з необхідністю забезпечення основних характеристик якості обслуговування Інтернет:

– надійності доведення отриманого від відправника повідомлення до одержувача;

– продуктивності;

– інформаційної безпеки.

Забезпечення гарантованих якісних характеристик процесу передачі даних в Інтернет або якості обслуговування Інтернет в умовах можливих впливів порушника на інформаційну сферу Інтернет і становить основу проблеми забезпечення інформаційної безпеки хмарних сервісів.

Рішення проблеми забезпечення інформаційної безпеки Інтернет пов'язане з необхідністю створення в Інтернет системи забезпечення інформаційної безпеки й повинне бути спрямоване на:

– формування єдиної політики в області забезпечення інформаційної безпеки при створенні, розвитку, модернізації й експлуатації хмарних сервісів;

– вироблення підходів до забезпечення інформаційної безпеки в умовах навмисних і ненавмисних впливів порушника;

– виявлення уразливостей в Інтернет і здійснення комплексу адекватних і економічно обґрунтованих мер по їхньому зниженню, запобіганню впливів порушника й ліквідації наслідків цього впливу на інформаційну сферу хмарних сервісів;

– створення системи державного регулювання в області забезпечення інформаційної безпеки хмарних сервісів (ліцензування, сертифікації й атестації);

– обґрунтування економічної доцільності забезпечення інформаційної безпеки;

– знаходження балансу між безпекою мережі й безпекою споживачів;

– впровадження керування ризиками й страхування відповідальності операторів Інтернет;

– створення системи підготовки й перепідготовки кадрів в області забезпечення інформаційної безпеки Інтернет;

– розробку, удосконалювання й стандартизацію методів, способів, алгоритмів і вітчизняних засобів (механізмів) забезпечення інформаційної безпеки хмарних сервісів;

– розробку механізмів страхування інформаційних ризиків.

Основними цілями забезпечення інформаційної безпеки хмарних сервісів є підтримка й збереження в умовах впливів порушника на інформаційну сферу хмарних сервісів наступних основних характеристик інформаційної безпеки Інтернет:

– конфіденційності інформаційної сфери хмарних сервісів;

– цілісності інформаційної сфери хмарних сервісів;

– доступності інформаційної сфери хмарних сервісів;

– підзвітності інформаційної сфери хмарних сервісів.

Забезпечення інформаційної безпеки хмарних сервісів повинне досягатися комплексним використанням організаційних, технічних, апаратно-програмних і криптографічних засобів захисту інформаційної сфери Інтернет, а також здійсненням безперервного контролю за ефективністю реалізованих заходів щодо забезпечення інформаційної безпеки Інтернет.

Забезпечення інформаційної безпеки Інтернет припускає створення перешкод для можливого несанкціонованого втручання в процес її функціонування.

У цьому змісті проблема забезпечення інформаційної безпеки хмарних сервісів містить у собі не тільки задачу захисту інформації від несанкціонованого доступу, але й ряд інших задач, пов'язаних із забезпеченням процесів функціонування хмарних сервісів, зокрема, із забезпеченням погодженого між оператором і клієнтом якості обслуговування в умовах впливів порушника на інформаційну сферу Інтернет.

1.2 Область застосування

Областю застосування розробляємого, у результаті виконання магістерського проекту, програмного забезпечення є захист доступу до хмарних сервісів. Це досягається за рахунок введення технології інфраструктури відкритих ключів (Public Key Infrastructure, PKI). Технологія PKI дозволяє перевіряти й засвідчувати дійсність користувача. PKI забезпечує єдину ідентифікацію, автентифікацію й авторизацію користувачів системи, додатків і процесів і разом із цим гарантує доступність, цілісність і конфіденційність інформації. Інфраструктура PKI являє собою систему цифрових сертифікатів. При використанні індивідуального секретного пароля й засобів криптографічного захисту, цифрові сертифікати одержують роль електронних паспортів.

Використання в корпоративній мережі технології PKI значно підвищує безпеку всієї мережі в цілому, тому що дозволяє відмовитися від використання парольної автентифікації користувачів усередині, а також забезпечує безпечний доступ віддалених користувачів у систему. Інфраструктура відкритих ключів забезпечує:

- автентифікацію користувачів хмарних сервісів, у тому числі при віддаленому доступі;
- захист електронної пошти (шифрування й електронно-цифровий підпис);
- захист корпоративних хмарних сервісів і порталів;
- захист внутрішніх і зовнішніх інформаційних потоків;
- захист бездротових мереж;
- захист інформації на ПК;
- контроль цілісності автентифікаційної інформації.

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

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Безпека хмарних сервісів уже не перший рік є важливим елементом захисту інформаційних систем. З огляду на тенденцію до переносу стандартних клієнт-серверних додатків в хмарні сервіси, що росте популярність технологій AJAX і інших елементів Web 2.0 можна констатувати, що із часом актуальність захисту онлайн-додатків тільки росте.

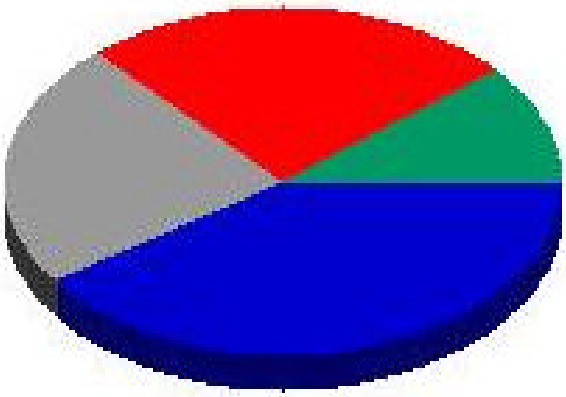
Наявність статистики дає можливість прогнозувати ризики пов'язані з використанням інформаційних систем і обґрунтовувати вибір контрзаходів. На жаль, одержати достовірну кількісну оцінку об'єму помилок, а тим більше – ймовірності їхньої експлуатації досить важко. У даному розділі наведений огляд деяких джерел, що дозволяють одержати статистичні дані про вразливості і атаки на Web-додатки.

Бази даних вразливостей

Відповістити на запитання про ймовірність виявлення тієї або іншої проблеми можна за допомогою інформації з довідників (баз даних) вразливостей. На сьогоднішній день визнаним галузевим стандартом у цій області є список Common Vulnerabilities and Exposures (CVE) [1]. Однак безпосередньо сам список слабко впорядкований і вимагає серйозної аналітичної роботи для одержання корисних статистично результатів.

Щорічно аналітиками Mitre проводиться аналіз інформації в базі даних і публікується звіт [2] про розподіл вразливостей за різними критеріями. Відповідно до звіту більше чверті проблем, виявлених в 2022 році, доводиться на недоліки безпеки хмарних сервісів.

Аналогічну інформацію публікують і інші бази даних вразливостей. За інформацією порталу SecurityLab.ru, близько 40% всіх виявлених в 2022 вразливостей доводиться на Web-додатки (рисунок 2.1).



Web-додатки (40,38%)
Серверні додатки(23,21%)
Клієнтські додатки (25,47%)
Вразливості в ОС (10,94%)

Рисунок 2.1 – Розподіл вразливостей по типу додатків в 2022 році

Незважаючи на те, що інформація з баз даних вразливостей досить цікава з теоретичної точки зору, вона мало підходить для практичного використання в розглянутому контексті. Це пов'язано з тим, що в бази даних попадає інформація про широко розповсюджені додатки, чиє впровадження носить масових характер. Що стосується хмарних сервісів, то найчастіше вони створюються під конкретну задачу й можуть бути розгорнуті тільки в одній мережі або в єдиному екземплярі. Останнім часом намітилася тенденція публікації в базах дані інформації не тільки про популярні Web-додатки, але й у часто використовуваних on-line сервісах. Наприклад – у загальнодоступних системах електронної пошти, пошукових системах, соціальних мережах і т.д. Однак поки це більш виключення, ніж правило. Таким чином, відповістити на запитання «Наскільки ймовірне виявлення вразливості в Web-додатку» за допомогою баз даних вразливостей неможливо.

Аналіз захищеності систем

Існують і інші підходи, наприклад, використання як вихідну інформацію результатів робіт з аналізу й оцінки захищеності хмарних сервісів. Як правило,

цей метод використовується консалтинговими компаніями, що мають великий досвід в області безпеки додатків.

Як приклад подібних звітів можна привести щорічний звіт «Статистика вразливості хмарних сервісів» компанії Positive Technologies (PT) [4] і щоквартальний огляд «Website Security Statistics Report» компанії WhiteHat Security (WH) [5]. Обидва звіти мають подібну структуру й містять статистику вразливостей хмарних сервісів, отриману в ході робіт з тестування на проникнення, аудита безпеки й ін. Для одержання даних використовувалися різні підходи, від сканування хмарних сервісів за допомогою сканерів з наступною перевіркою результатів до тестування методом "білого ящика", що включає також частковий аналіз вихідного коду.

Обидва звіти використовують класифікацію вразливостей Web Application Security Consortium Web Security Threat Classification [6]. У цьому документі зібрані воєдино й організовані різні погрози безпеки хмарних сервісів. Проект є спробою розробки й популяризації стандартної термінології опису проблем безпеки в Web-додатках. Розповсюджені вразливості хмарних сервісів організовані в структурований список, що складається із шести класів, кожний з яких містить кілька типів вразливостей і атак. У цей час готується до публікації друга редакція класифікації, що містить дев'ять класів погроз.

На даний момент виділені основні наступні класи погроз:

1. Cross-Site Request Forgery – Підробка HTTP-запиту;
2. Cross-Site Scripting, XSS) – Міжсайтове виконання сценаріїв;
3. SQL Injection – Впровадження операторів SQL;
4. Information Leakage – Витік інформації;
5. HTTP Response Splitting – Розщеплення HTTP-відповіді.

Обидва звіти використовують елементи Web Security Threat Classification version 2 для опису таких проблем як «Розщеплення HTTP-відповіді» (HTTP Response Splitting) і «Підробка HTTP-запиту» (Cross-Site Request Forgery).

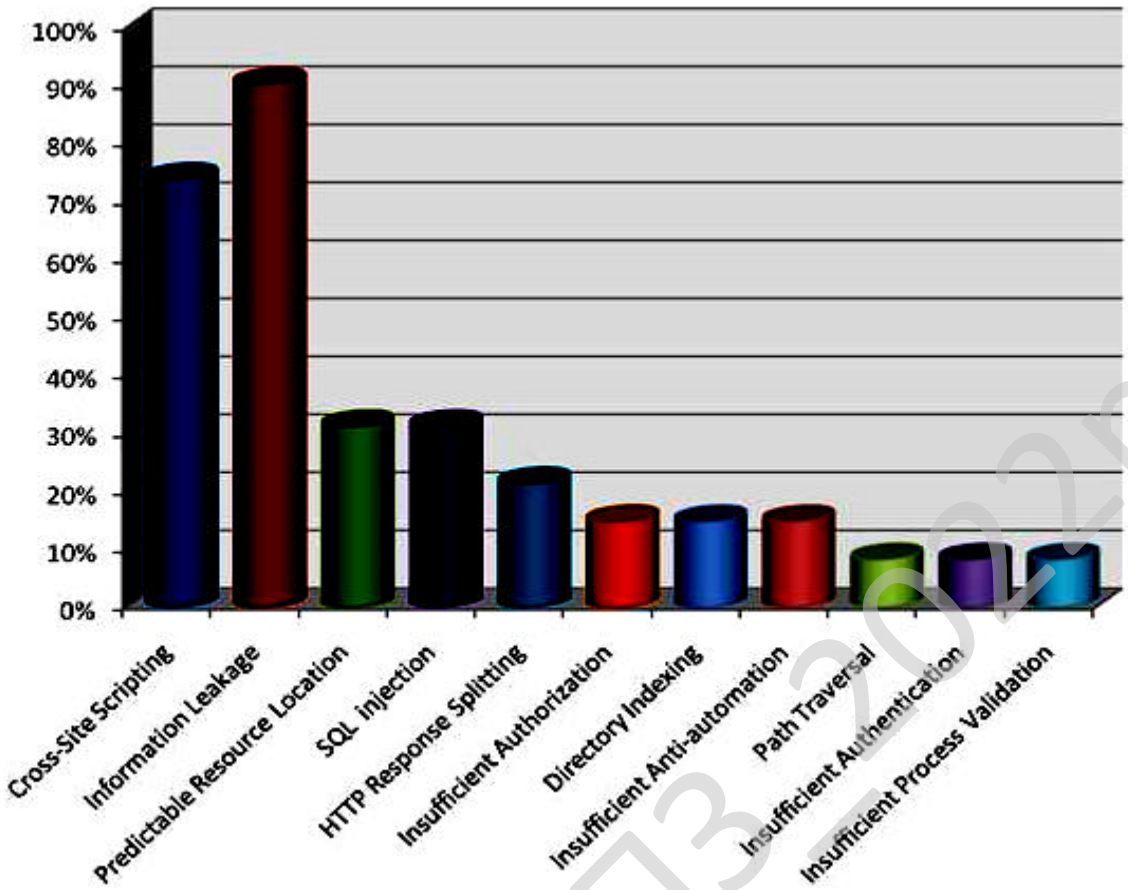


Рисунок 2.2 – Ймовірність виявлення вразливостей різного типу (Positive Technologies)

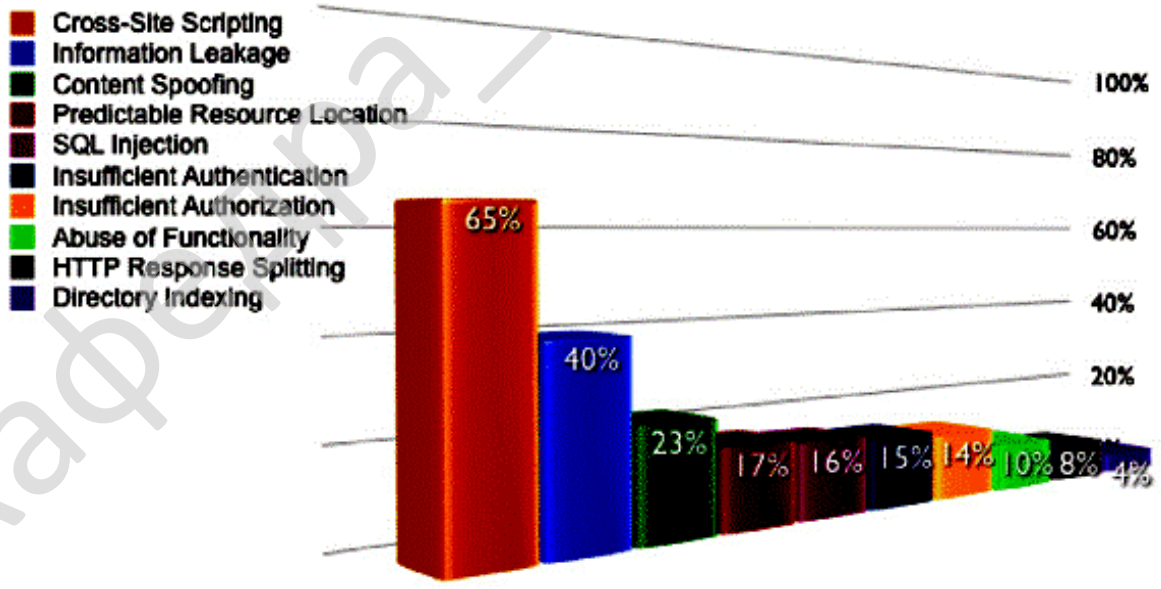


Рисунок 2.3 – Ймовірність виявлення вразливостей різного типу (WhiteHat Security)

Цікаво, що дані звіти близькі не тільки за структурою, але й за результатами. Так, найпоширенішою уразливістю обидва джерела визнають «Міжсайтове виконання сценаріїв» (Cross-Site Scripting, XSS). Ця уразливість була виявлена в 74% всіх сайтів за версією Positive Technologies (рисунок 2.2) і в 65 випадках з 100 (рисунок 2.3) у клієнтів WhiteHat Security.

Найпоширенішою уразливістю високим ступенем ризику обидві компанії визнають «Впровадження операторів SQL» (SQL Injection) ймовірність виявлення якої становить 31% – PT і 16% – WH. Менша ймовірність виявлення проблем різних типів у трактуванні WhiteHat Security цілком з'ясовна більшою зрілістю західного ринку й використанням результатів повторних перевірок додатків того самого клієнта. На жаль, більшість українських власників хмарних сервісів перебуває на етапі становлення процесу керування інформаційною безпекою.

У деяких пунктах, наприклад по кількості вразливостей типу «Витік інформації» (Information Leakage) звіти досить серйозно розходяться (90% – PT і 40% – WH). Це пов'язано з тим, що WhiteHat Security включає у звіт тільки вразливості, що мають «критичний», «невідкладний» і «високий» рівень ризику, у той час як Positive Technologies задіє всі знайдені недоліки.

Оцінка ступеня ризику

Питання класифікації ступеня ризику, пов'язаного з вразливостями додатків є важливою темою. У даний момент існує множина методик оцінки небезпеки вразливості, але найпоширеніші наступні підходи:

– класична «світлофорна» оцінка, що виділяє вразливості «високого», «середнього» і «низкого» ступеня ризику;

– пятирівнева модель, прийнята в стандарті PSI DSS [7] і визначальний рівні «критичний», «невідкладний», «високий», «середній» і «низький» (Urgent, Critical, High, Medium, Low);

– метод Common Vulnerability Scoring System (CVSS) [8], що оцінює ступінь ризику як число від 0 до 10.

Не стосуючись достоїнств або недоліків кожного з методів можна виділити наступні особливості, які можуть впливати на вірогідність оцінки:

– залежність від контексту;

– залежність від конфігурації системи;

– залежність від методу визначення.

У різних додатках вразливості одного типу можуть мати різний ступінь ризику. Так, уразливість «Підробка HTTP-запиту» може не представляти погрози для типового репрезентативного сайту або пошукової машини, і навпаки – класифікуватися як проблема високого ступеня ризику в Web-інтерфейсі електронної пошти або платіжної системи. У результаті витоку інформації зловмисник може одержати доступ до журналів роботи додатка (низький або середній ступінь ризику), а може завантажити резервну копію вихідних текстів сайту (високий ступінь ризику).

Конфігурація конкретної системи також може впливати на ступінь ризику. Так, уразливість «Впровадження операторів SQL» звичайно класифікують високу небезпеку, що як має. Однак у випадку якщо Web-додаток працює із сервером СУБД із обмеженими привілеями, вона може бути віднесена до проблем середнього або низького ступеня ризику. В іншій інсталяції або реалізації додатка ця ж уразливість може бути використана для одержання доступу до операційної системи із правами суперкористувача, що природно робить її найбільш критичної.

Залежно від методу в глибини аналізу ступінь та сама уразливість може бути оцінена по-різному. Якщо взяти наведений вище приклад «Впровадження операторів SQL», використання мережного сканера дозволить тільки констатувати наявність проблеми. Для визначення привілеїв, доступних потенційному зловмисникові потрібно або спробувати використовувати помилку, або уточнити порядок взаємодії між Web-додатком і СУБД методом «білого ящика».

Кожний з наведених методів оцінки використовує свої підходи для обліку позначених обставин. Це може бути абстрактне «експертна думка» в «світлофорній» оцінці або вагарні коефіцієнти в CVSS, але в кожному разі – метод і глибина аналізу впливають на оцінку ризиків. Це обставина необхідно приймати увагу при роботі зі звітами й статистичними даними.

Міжнародна статистика

Проблема об'єднання даних з декількох джерел, що використовують різні методи аналізу й підходи до оцінки ступеня ризику, позначилася на актуальності проекту Web Application Security Statistics Project міжнародної групи Web Application Security Consortium [9]. Споконвічно дані для цього проекту надавалися компаніями Cenzic, Positive Technologies, SPI Dynamics і WhiteHat Security використовуючими схожі підходи до аналізу захищеності систем. Однак при спробі залучення більшої кількості учасників для формування звіту за 2022 рік, ініціативна група проекту зштовхнулася з неможливістю об'єднання деяких результатів. Це привело до того, що дані за 2022 рік поки не були опубліковані, і ймовірно не будуть доступні раніше другого півріччя 2023 року.

У кожному разі Web Application Security Statistics Project є одним з найбільш масштабних проектів у галузі, і містить у собі результати аналізу 31373 додатків, що становить близько 0,03% всіх сайтів у мережі Інтернет в 2021 році. Залишається сподіватися, що роботи над проектом буде продовжені.

Ймовірність атаки

Ймовірність виявлення вразливості не є ймовірністю реалізації погрози (атаки), що потрібно в класичній моделі оцінки ризиків. Одержати реальні дані про інциденти досить складно. В основному використовуються два підходи: реєстрація інформації про інциденти й протоколювання спроб за допомогою технології honeypot або антивірусного сканування.

Як цікаві приклади першого типу можна привести портал Zone-H [10] і проект Web Hacking Incidents Database [11]. Портал Zone-H реєструє наслідку компрометації хмарних сервісів. Як правило, інформацію в архів заносять самі

зловмисники. Цим визначається й наповнення архіву, що містить в основному інформацію про атаки «зламати хоч що-небудь».

База даних Web Hacking Incidents Database поповнюється на основі інформації, доступної в засобах масової інформації. У зв'язку із цим технічні деталі, як правило, відсутні або недостатньо конкретні, однак об'єкти успішних атак досить і досить значні. Найчастіше в інформації про інцидент є інформація, що дозволяє оцінити наслідку атаки.

Прикладами використання технології honeypot для визначення найбільш ймовірних методів атак представлені в публікаціях Internet Security Threat Report компанії Symantec і звітах проекту «Distributed Open Proxy Honyepot».

Існуючі на справжній момент відкриті джерела дають достатню базу для проведення оцінки ризику, пов'язаного з вразливостями в Web-додатках. Різні джерела використовують різні дані й методи їхньої інтерпретації. У зв'язку із цим важливо вибирати найбільш підходящий для рішення конкретних задач інструмент. Благо, на справжній момент у них немає недоліку.

Природно, не варто забувати, що вразливості хмарних сервісів, це не тільки технічні проблеми прикладної частини. Голосні інциденти найчастіше пов'язані із крадіжкою паролів на адміністрування сайту або інших проблем, пов'язаними з персоналом.

2.2 Обґрунтування вибору засобів для побудови системи 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 другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи доступу до хмарних сервісів з використанням технології PKI.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Опис SSL/ TLS

SSL – криптографічний протокол, що забезпечує безпечну передачу даних по мережі Інтернет. При його використанні створюється захищене з'єднання між клієнтом і сервером. SSL споконвічно розроблений компанією Netscape Communications. Згодом на підставі протоколу SSL 3.0 був розроблений і прийнятий стандарт RFC, що одержав ім'я TLS.

Використовує шифрування з відкритим ключем для підтвердження дійсності передавача й одержувача. Підтримує надійність передачі даних за рахунок використання коригувальних кодів і безпечних геш-функцій.

SSL складається із двох рівнів. На нижньому рівні багаторівневого транспортного протоколу (наприклад, TCP) він є протоколом запису й використовується для інкапсуляції (тобто формування пакета) різних протоколів (SSL працює разом з таким протоколами як POP3, IMAP, XMPP, SMTP і HTTP). Для кожного інкапсульованого протоколу він забезпечує умови, при яких сервер і клієнт можуть підтверджувати один одному свою дійсність, виконувати алгоритми шифрування й робити обмін криптографічними ключами, перш ніж протокол прикладної програми почне передавати й одержувати дані.

Для доступу до веб-сторінок, захищеним протоколом SSL, в URL замість звичайного префікса http, як правило, застосовується префікс https, що вказує на те, що буде використовуватися SSL-з'єднання. Стандартний TCP-порт для з'єднання по протоколу https – 443.

Для роботи SSL потрібно, щоб на сервері був SSL-сертифікат.

TLS – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет. TL*S*-протокол заснований на Netscape SSL-

протоколі версії 3.0 і складається із двох частин – TLS Record Protocol і TLS Handshake Protocol. Розходження між SSL 3.0 і TLS 1.0 незначні, тому далі в тексті термін «SSL» буде відноситися до них обох. TLS Working Group, заснована в 1996 році, продовжує працювати над протоколом.

Опис

TLS надає можливості автентифікації й безпечної передачі даних через Інтернет з використанням криптографічних засобів. Часто відбувається лише автентифікація сервера, у той час як клієнт залишається неавтентифікованим. Для взаємної автентифікації кожна зі сторін повинна підтримувати інфраструктуру відкритого ключа (PKI), що дозволяє захистити клієнт-серверні додатки від перехоплення повідомлень, редагування існуючих повідомлень і створення підроблених.

SSL містить у собі три основних фази:

– Діалог між сторонами, метою якого є вибір алгоритму шифрування.

– Обмін ключами на основі криптосистем з відкритим ключем або автентифікація на основі сертифікатів.

– Передача даних, шифруємих за допомогою симетричних алгоритмів шифрування.

Алгоритм процедури встановлення з'єднання по протоколу TLS handshake

Клієнт і сервер, що працюють по TLS, установлюють з'єднання, використовуючи процедуру handshake ("рукостискання"). Протягом цього handshake, клієнт і сервер приймають угоду щодо параметрів, використовуваних для встановлення захищеного з'єднання.

Послідовність дій при встановленні TLS з'єднання:

– клієнт підключається до TLS – підтримуваного сервера й запитує захищене з'єднання;

– клієнт надає список підтримуваних алгоритмів шифрування й геш-функцій;

– сервер вибирає зі списку, наданого клієнтом, найбільш стійкі алгоритми, які також підтримуються сервером, і повідомляє про свій вибір клієнтові;

– сервер відправляє клієнтові цифровий сертифікат для власної ідентифікації. Звичайно цифровий сертифікат містить ім'я сервера, ім'я довіреного центра сертифікації й відкритий ключ сервера;

– клієнт може зв'язатися із сервером довіреного центра сертифікації й підтвердити автентичність переданого сертифіката до початку передачі даних;

– для того щоб згенерувати ключ сесії для захищеного з'єднання, клієнт шифрує випадково згенерировану цифрову послідовність відкритим ключем сервера й посилає результат на сервер. З огляду на специфіку алгоритму асиметричного шифрування, використовуваного для встановлення з'єднання, тільки сервер може розшифрувати отриману послідовність, використовуючи свій закритий ключ;

Handshake у деталях

Відповідно до протоколу TLS додатки обмінюються записами, інкапсулюючими (що зберігають усередині себе) інформацію, яка повинна бути передана. Кожен із записів може бути стисла, доповнена, зашифрована або ідентифікована MAC залежно від поточного стану з'єднання (стану протоколу). Кожний запис в TLS містить наступні поля: content type (визначає тип умісту запису), поле, що вказує довжину пакета, і поле, що вказує версію протоколу TLS.

Коли з'єднання тільки встановлюється, взаємодія йде по протоколу TLS handshake, content type якого 22.

Нижче описаний простий приклад установлення з'єднання:

1. Клієнт посилає повідомлення **ClientHello**, указуючи найбільш останню версію підтримуваного TLS протоколу, випадкове число й список підтримуваних методів шифрування й стиски, що підходять для роботи з TLS.

2. Сервер відповідає повідомленням ServerHello, що містить: обрану сервером версію протоколу, випадкове число, послане клієнтом, що підходить алгоритм шифрування й стиски зі списку наданого клієнтом.

3. Сервер посилає повідомлення **Certificate**, що містить цифровий сертифікат сервера (залежно від алгоритму шифрування цей етап може бути пропущений)

4. Сервер може запросити сертифікат у клієнта, у такому випадку з'єднання буде взаємно автентифіковано.

5. Сервер відсилає повідомлення **ServerHelloDone**, що ідентифікує закінчення handshake.

6. Клієнт відповідає повідомленням **ClientKeyExchange**, що містить PreMasterSecret відкритий ключ, або нічого (знову ж залежить від алгоритму шифрування).

7. Клієнт і сервер, використовуючи PreMasterSecret ключ і випадково згенеровані числа, обчислюють загальний секретний ключ. Вся інша інформація про ключ буде отримана із загального секретного ключа (і згенерованих клієнтом і сервером випадкових значень).

8. Клієнт посилає **ChangeCipherSpec** повідомлення, що вказує на те, що вся наступна інформація буде зашифрована встановленим у процесі handshake алгоритмом, використовуючи загальний секретний ключ. Це повідомлення рівня записів і тому має тип 20, а не 22.

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

10. Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити геш і MAC. Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване.

11. Сервер посилає **ChangeCipherSpec** і зашифроване **Finished** повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

Із цього моменту handshake уважається завершеним, протокол установленим. Весь наступний уміст пакетів іде з типом 23, а всі дані будуть зашифровані.

Алгоритми, що використовуються в TLS

У даній поточній версії протоколу доступні наступні алгоритми:

– Для обміну ключами й перевірки їхньої дійсності застосовуються комбінації алгоритмів: RSA (асиметричний шифр), Diffie-Hellman (DH) (безпечний обмін ключами), DSA (алгоритм цифрового підпису) і алгоритми технології Fortezza.

– Для симетричного шифрування: RC2, RC4, IDEA, DES, Triple DES або AES;

– Для геш-функцій: MD5 або SHA.

Алгоритми можуть доповняться залежно від версії протоколу.

На рисунку 3.1 наведено архітектуру SSL/TLS.

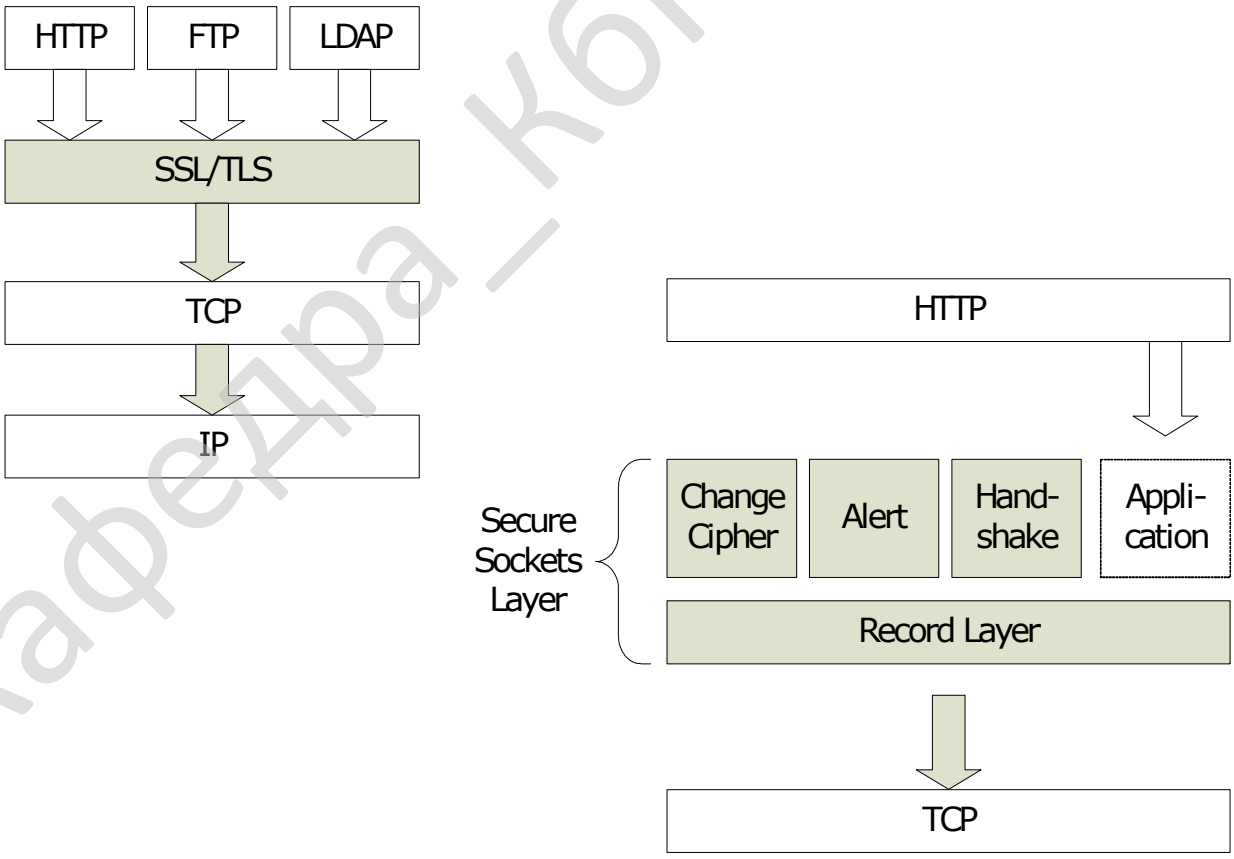


Рисунок 3.1 – Архітектура SSL/TLS

Як ми бачимо в архітектурі задіяні наступні протоколи:

– HTTP – протокол прикладного рівня передачі даних у першу чергу у вигляді текстових повідомлень. Основою HTTP є технологія « клієнт-сервер», тобто передбачається існування споживачів (клієнтів), які ініціюють з'єднання й надсилають запит, і постачальників (серверів), які очікують з'єднання для одержання запиту, роблять необхідні дії й повертають обернено повідомлення з результатом. HTTP використовується також у якості «транспорту» для інших протоколів прикладного рівня, таких як SOAP. Основним об'єктом маніпуляції в HTTP є ресурс, на який вказує URI (Uniform Resource Identifier) у запиті клієнта. Звичайно такими ресурсами є файли, що зберігаються на сервері, але ними можуть бути логічні об'єкти або щось абстрактне. Особливістю протоколу HTTP є можливість указати в запиті й відповіді спосіб подання того самого ресурсу по різних параметрах: формату, кодуванню, мові й т.д. Саме завдяки можливості вказівки способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча даний протокол є текстовим. HTTP – протокол прикладного рівня, аналогічними йому є FTP і SMTP. Обмін повідомленнями йде за звичайною схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує глобальні URI. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаної з останніми запитами й відповідями. Браузер, що посилає запити, може відслідковувати затримки відповідей. Сервер може зберігати IP-адреси й заголовки запитів останніх клієнтів. Однак сам протокол не обізнаний про попередні запити й відповіді, у ньому не передбачена внутрішня підтримка стану, до нього не пред'являються такі вимоги.

– FTP – протокол, призначений для передачі файлів у комп'ютерних мережах. FTP дозволяє підключатися до серверів FTP, переглядати вміст каталогів і завантажувати файли із сервера або на сервер; крім того, можливий

режим передачі файлів між серверами. Протокол FTP відноситься до протоколів прикладного рівня й для передачі даних використовує транспортний протокол TCP. Команди й дані, на відміну від більшості інших протоколів передаються по різних портах. Порт 20 використовується для передачі даних, порт 21 для передачі команд. Протокол не шифрується, при автентифікації передає логін і пароль відкритим текстом. Якщо зловмисник перебуває в одному сегменті мережі з користувачем FTP, те, використовуючи сніффер, він може перехопити логин і пароль користувача, або, при наявності спеціального ПЗ, одержувати передані по FTP файли без авторизації. Щоб запобігти перехопленню трафіку, необхідно використовувати протокол шифрування даних SSL, що підтримується багатьма сучасними FTP-серверами й деякими FTP-Клієнтами.

– LDAP – це мережний протокол для доступу до служби каталогів X.500, розроблений IETF як полегшений варіант розробленого ITU-T протоколу DAP. LDAP – відносно простий протокол, що використовує TCP/IP і дозволяє робити операції автентифікації (bind), пошуку (search) і порівняння (compare), а також операції додавання, зміни або видалення записів. Звичайно LDAP-сервер приймає вхідні з'єднання на порт 389 по протоколах TCP або UDP. Для LDAP-сеансів, інкапсульованих в SSL, звичайно використовується порт 636. Усякий запис у каталозі LDAP складається з одного або декількох атрибутів і має унікальне ім'я (DN). Унікальне ім'я складається з одного або декількох відносних унікальних імен (RDN), розділених комою. На одному рівні каталогу не може існувати двох записів з однаковими відносними унікальними іменами. У силу такої структури унікального ім'я запису в каталозі LDAP можна легко представити у вигляді дерева. Запис може складатися тільки з тих атрибутів, які визначені в описі класу запису (object class), які, у свою чергу, об'єднані в схеми (schema). У схемі визначено, які атрибути є для даного класу обов'язковими, а які – необов'язковими. Також схема визначає тип і правила порівняння атрибутів. Кожний атрибут запису може зберігати кілька значень.

– TCP – один з основних мережних протоколів Internet, призначений для керування передачею даних у мережах і під мережах TCP/IP. Виконує функції протоколу транспортного рівня спрощеної моделі OSI. IP-ідентифікатор – 6. TCP – це транспортний механізм, що надає потік даних, з попередньою установкою з'єднання, за рахунок цього даючий впевненість у вірогідності одержуваних даних, здійснює повторний запит даних у випадку втрати даних і усуває дублювання при одержанні двох копій одного пакета. На відміну від UDP, гарантує, що додаток одержить дані точно в такій же послідовності, у якій вони були відправлені, і без втрат. Реалізація TCP як правило, убудована в ядро системи, хоча є й реалізації TCP у контексті додатка. TCP часто позначають «TCP/IP». Коли здійснюється передача від комп'ютера до комп'ютера через Internet, TCP працює на верхньому рівні між двома кінцевими системами, наприклад, інтернет-браузер і інтернет-сервер. Також TCP здійснює надійну передачу потоку байт від однієї програми на деякому комп'ютері в іншу програму на іншому комп'ютері. Програми для електронної пошти й обміну файлами використовують TCP. TCP контролює довжину повідомлення, швидкість обміну повідомленням, мережний трафік.

– IP – маршрутизуємий мережний протокол, основа стека протоколів TCP/IP. Протокол IP використовується для негарантованої доставки даних (поділюваних на так звані пакети) від одного вузла мережі до іншого. Це означає, що на рівні цього протоколу (третій рівень мережної моделі OSI) не дається гарантій надійної доставки пакета до адресата. Зокрема, пакети можуть прийти не в тому порядку, у якому були відправлені, продублюватися (коли приходять дві копії одного пакета; у реальності це буває вкрай рідко), виявитися ушкодженими (звичайно ушкоджені пакети знищуються) або не прибути зовсім. Гарантії безпомилкової доставки пакетів дають протоколи більш високого (транспортного) рівня мережної моделі OSI – наприклад, TCP – які IP використовують як транспорт. У сучасній мережі Інтернет використовується IP четвертої версії, також відомий як IPv4. У протоколі IP цієї версії кожному вузлу

мережі відноситься у відповідність IP-адреса довжиною 4 октети (іноді говорять «байта», маючи на увазі розповсюджений восьмибітовий мінімальний адресуємий фрагмент пам'яті ЕОМ). При цьому комп'ютери в підмережах поєднуються загальними початковими бітами адреси. Кількість цих біт, загальна для даної підмережі, називається маскою підмережі (раніше використовувалося ділення простору адрес по класах – A, B, C; клас мережі визначався діапазоном значень старшого октету й визначав число адресуємих вузлів у даній мережі, зараз використовується безкласова адресація). У поточний час вводиться в експлуатацію шоста версія протоколу – IPv6, що дозволяє адресувати значно більшу кількість вузлів, чим IPv4. Ця версія відрізняється підвищеною розрядністю адреси, убудованою можливістю шифрування й деяких інших особливостей. Перехід з IPv4 на IPv6 пов'язаний із трудомісткою роботою операторів зв'язку й виробників програмного забезпечення й не може бути виконаний одномоментно.

Опис алгоритму шифрування AES

Визначення й допоміжні процедури AES

Визначення

Block – послідовність бітів, з яких складається input, output, State і Round Key. Також під Block можна розуміти послідовність байтів.

Cipher Key – секретний, криптографічний ключ, що використовується Key Expansion процедурою, щоб зробити набір ключів для раундів(Round Keys); може бути представлений як прямокутний масив байтів, що має чотири рядки й *Nk* колонок.

Ciphertext – вихідні дані алгоритму шифрування.

Key Expansion – процедура, що використовуєтьчя для генерації Round Keys з Cipher Key.

Round Key – Round Keys виходять із Cipher Key використовуючи процедуру Key Expansion. Вони застосовуються до State при шифруванні і дешифруванні.

State – проміжний результат шифрування, що може бути представлений як прямокутний масив байтів що має 4 рядки й *Nb* колонок.

S-box – нелінійна таблиця замін використовується в декількох трансформаціях заміни байт і в процедурі Key Expansion для взаємооднозначної заміни значення байта.

Nb – число стовпців(32– бітних слів), що становлять State. Для, AES *Nb* = 4.

Nk – число 32-бітних слів, що становлять шифроключ. Для AES, *Nk* = 4,6, або 8.

Nr – число раундів, що є функцією *Nk* і *Nb*. Для AES, *Nr* = 10, 12, 14.

Rcon[] – масив, що складається з бітів 32-х розрядного слова і є постійним для даного раунду.

Допоміжні процедури

AddRoundKey() – трансформація при шифруванні й зворотному шифруванні, при якій Round Key виконує XOR з State. Довжина RoundKey дорівнює розміру State (якщо *Nb* = 4, то довжина RoundKey дорівнює 128 бітів або 16 байтів).

InvMixColumns() – трансформація при розшифруванні яка є зворотною стосовно MixColumns().

InvShiftRows() – трансформація при розшифруванні яка є зворотною стосовно ShiftRows().

InvSubBytes() – трансформація при розшифруванні яка є зворотною стосовно SubBytes().

MixColumns() – трансформація при шифруванні яка бере всі стовпці State і змішує їх дані (незалежно друг від друга), щоб одержати нові стовпці.

RotWord() – функція, яка використовується в процедурі Key Expansion, що бере чотирьохбайтне слово й робить над ним циклічну перестановку.

ShiftRows() – трансформації при шифруванні, які обробляють State, циклічно зміщаючи останні три рядки State на різні величини.

SubBytes() – трансформації при шифруванні які обробляють State використовуючи нелінійну таблицю заміщення байтів(*S*-box), застосовуючи її незалежно до кожного байта State.

SubWord() – функція, яка використовується в процедурі Key Expansion, що бере на вході чотирьохбайтне слово й застосовуючи *S*-box до кожного із чотирьох байтів видає вихідне слово.

Шифрування

Для AES довжина input (блоку вхідних даних) і State (стану) постійна й дорівнює 128 бітів, а довжина шифроключа **K** становить 128, 192, або 256 бітів. Для позначення обраних довжин input, State і Cipher Key у байтах використовується нотація $Nb = 4$ для input і State, $Nk = 4, 6, 8$ для Cipher Key відповідно для різних довжин ключів.

На початку шифрування input копіюється в масив State за правилом $s[r,c] = \text{in}[r + 4c]$, для $0 \le r \le 4$ і $0 \le c < Nb$. Після цього до State застосовується процедура AddRoundKey() і потім State проходить через процедуру трансформації(раунд) 10, 12, або 14 разів (залежно від довжини ключа), при цьому треба врахувати, що останній раунд трохи відрізняється від попередніх. У підсумку, після завершення останнього раунду трансформації, State копіюється в output за правилом $\text{out}[r + 4c] = s[r,c]$, для $0 \le r \le 4$ і $0 \le c < Nb$.

Шифр описаний у псевдокоді Cipher. Окремі трансформації SubBytes(), ShiftRows(), MixColumns(), і AddRoundKey() – обробляють State. Масив *w*[] – містить key schedule.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
        state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
```

```
        AddRoundKey(state, W[round*Nb, (round+1)* Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, W[Nr*Nb, (Nr+1)* Nb-1])
    out = state
end
```

SubBytes()

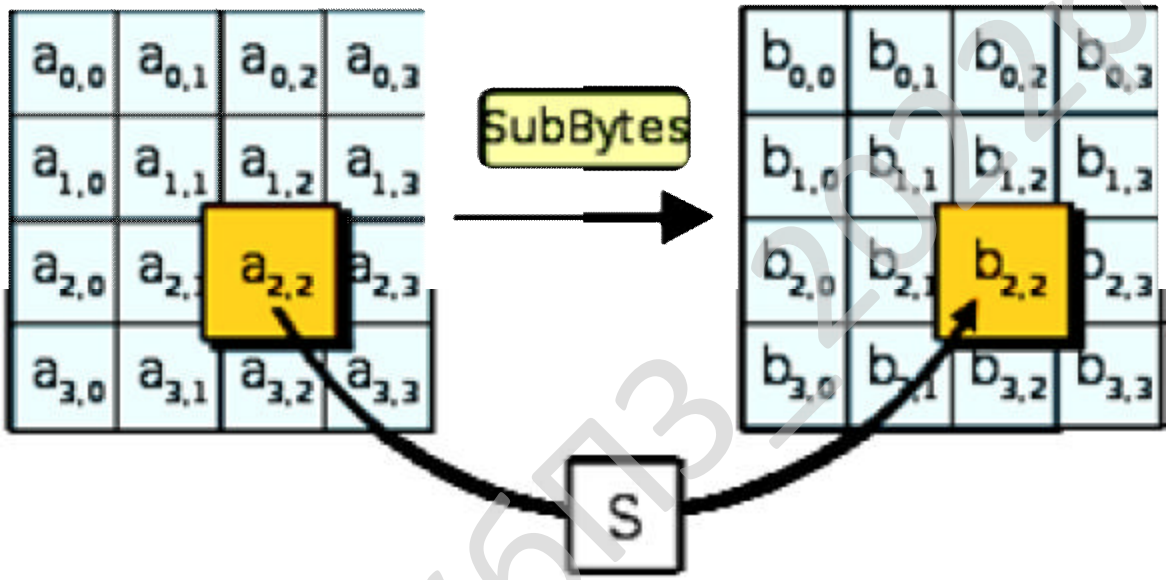


Рисунок 3.2 – Процедура SubBytes

У процедурі SubBytes, кожний байт в state заміняється відповідним елементом у фіксованої 8-бітній таблиці пошуку, S; $b_{ij} = S(a_{ij})$.

Процедура SubBytes() обробляє кожний байт стану, незалежно роблячи нелінійну заміну байтів використовуючи таблицю замін (S-box). Така операція забезпечує нелінійність алгоритму шифрування. Побудова S-box складається із двох кроків. По-перше, виробляється взяття оберненого числа в $GF(2^8)$. По-друге, до кожного байта b з яких складається S-box застосовується наступна операція:

$$b_s^{'} = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i ,$$

де $0 \le i \le 8$, і де b_i є i-ий біт b, а c_i – i-ий байт c = {63} або {01100011}. У такий спосіб забезпечується захист від атак заснованих на простих алгебраїчних властивостях.

$$\begin{Vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{Vmatrix} * \begin{Vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix}$$

ShiftRows()

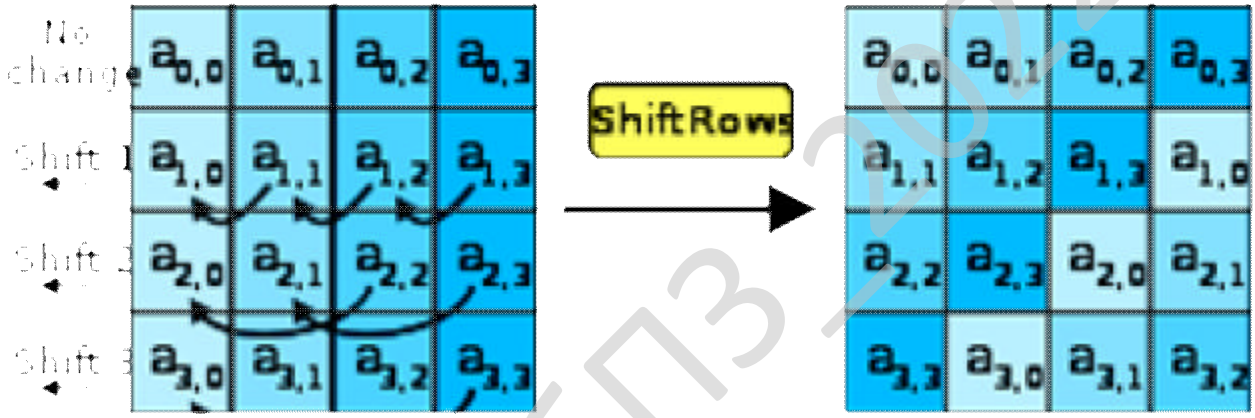


Рисунок 3.3 – Процедура ShiftRows

У процедурі ShiftRows, байти в кожному рядку state циклічно зсувається вліво. Розмір зсуву байтів кожного рядка залежить від його номера.

ShiftRows працює з рядками State. При цій трансформації рядка стани циклічно зсувається на *r* байт по горизонталі, залежно від номера рядка. Для нульового рядка *r* = 0, для першого рядка *r* = 1, і т.д. У такий спосіб кожний стовпчик вихідного стану після застосування процедури ShiftRows складається з байтів з кожного стовпчика початкового стану. Для алгоритму Rijndael паттерн зсуву рядків для 128 і 192-бітних рядків однаковий. Однак для блоку розміром 256 бітів відрізняється від попередніх тим, що 2, 3, і 4-ий рядки зміщаються на 1, 3, і 4 байти відповідно.

MixColumns()

У процедурі MixColumns, кожний стовпчик стану перемножується з фіксованим багаточленом $c(x)$.

У процедурі MixColumns, чотири байти кожного стовпчика State змішуються використовуючи для цього оборотну лінійну трансформацію. MixColumns обробляє стан по стовпчиках, трактуючи кожен з них як поліном четвертого ступеня. Над цими поліномами виробляється множення в $GF(2^8)$ по модулю $x^4 + 1$ на фіксований багаточлен $c(x) = 3x^3 + x^2 + x + 2$. Разом з ShiftRows, MixColumns вносить diffusion у шифр.

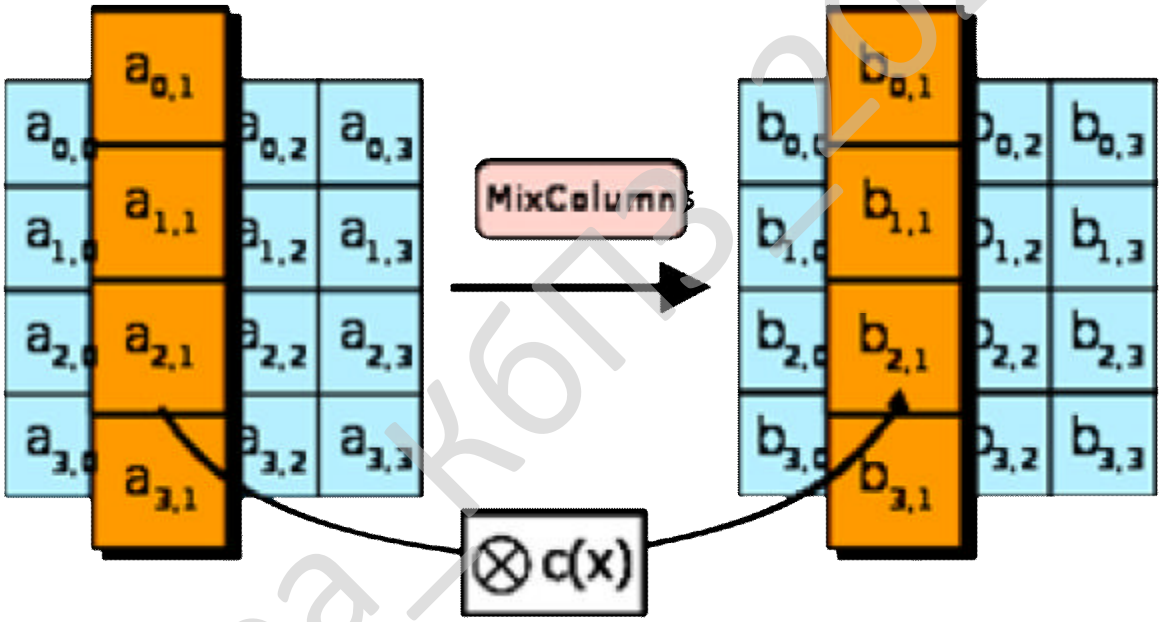


Рисунок 3.4 – Процедура MixColumns

AddRoundKey()

У процедурі AddRoundKey, кожний байт стану поєднується з RoundKey використовуючи XORoperation(). У процедурі AddRoundKey, RoundKey кожного раунду поєднується з State. Для кожного раунду Roundkey виходить із CipherKey використовуючи процедуру KeyExpansion; кожний RoundKey такого ж розміру, що й State. Процедура робить побітовий XOR кожного байта State з кожним байтом RoundKey.

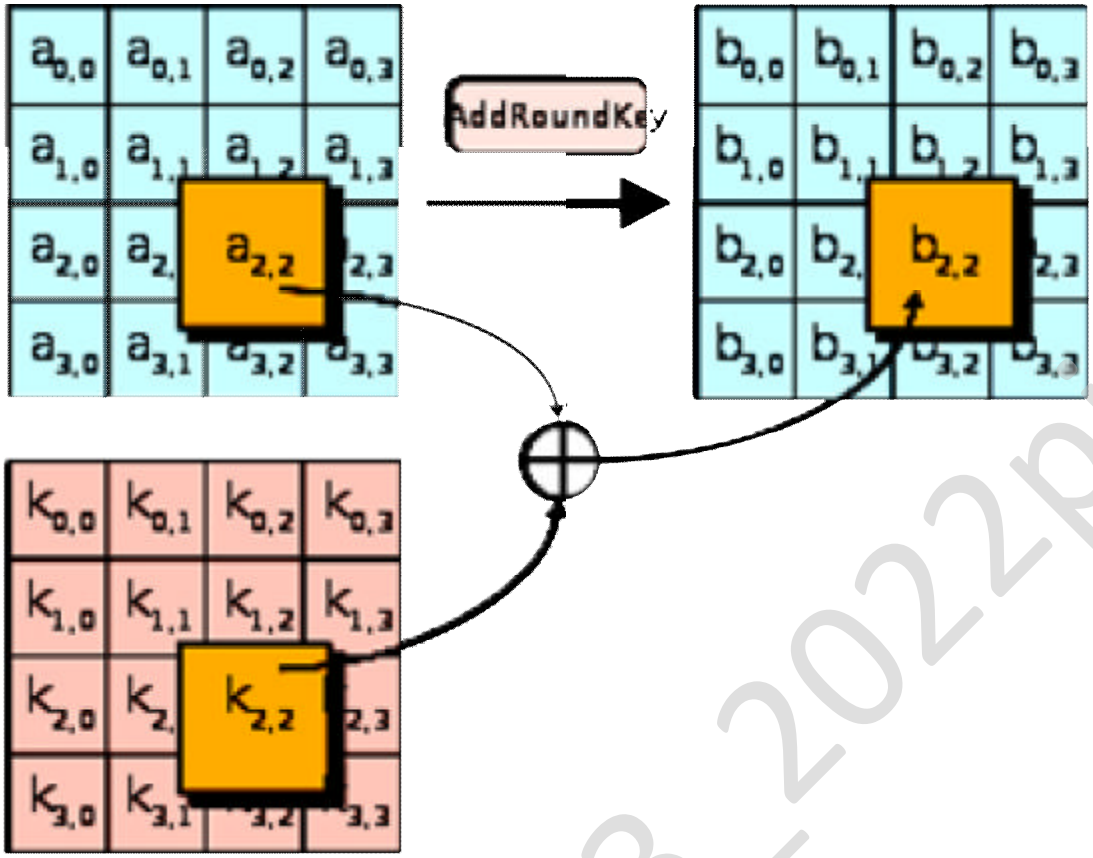


Рисунок 3.5 – Процедура AddRoundKey

KeyExpansion()

AES алгоритм використовуючи процедуру KeyExpansion() і подаючи в неї Cipher Key, K, одержує ключі для всіх раундів. Усього вона одержує *Nb**(*Nr* + 1) слів: с початку для алгоритму потрібен набір з *Nb* слів, і кожному з *Nr* раундів потрібно *Nb* ключових наборів даних. Отриманий масив ключів для раундів позначається як *w*[*i*], $0 \le i < Nb*(Nr + 1)$. Алгоритм KeyExpansion() показаний у псевдокоді нижче

Функція SubWord() бере чотирьохбайтне вхідне слово й застосовує *S*-box до кожного із чотирьох байтів, те, що вийшло подається на вихід. На вхід процедури RotWord() подається слово $[a_0,a_1,a_2,a_3]$ яке вона циклічно переставляє й повертає $[a_1,a_2,a_3,a_0]$. Масив слів, слів постійний для даного раунду, *Rcon*[*i*], містить значення $[x^{i-1},00,00,00]$, де $x = \{02\}$, а x^{i-1} є ступенем *x* в $GF(2^8)$ (*i* починається з 1).

Перші *Nk* слів розширеного ключа заповнені Cipher Key. У кожне наступне слово, *w*[*i*], кладе значення отримане при операції XOR *w*[*i*-1] і *w*[*i* – *Nk*]. Для слів, позиція яких кратна *Nk*, перед XOR'ом до *w*[*i*-1] застосовується трасформація, за якою слідує XOR з константою раунду *Rcon*[*i*]. Зазначена вище трансформація складається із циклічного зсуву байтів у слові(RotWord()), за якою слідує процедура SubWord() – теж саме, що й SubBytes(), тільки вхідні й вхідні дані будуть розміром у слово.

Важливо помітити, що процедура KeyExpansion() для 256 бітного Cipher Key не набагато відрізняється від тих, які застосовуються для 128 і 192 бітних шифроключів. Якщо *Nk* = 8 й *i* – 4 кратно *Nk*, то SubWord() застосовується до *w*[*i*-1] до XOR'а.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0;
    while ( i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while ( i < Nb * (Nr+1))
        temp = w[ i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[ i-Nk] xor temp
        i = i + 1
    end while
end
```

Розшифрування

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[Nr*Nb, (Nr+1)* Nb-1])
    for round = Nr-1 step -1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvAddRoundKey(state, w[round*Nb, (round+1)* Nb-1])
        InvMixColumns(state)
    end for
    InvShiftRows(state)
    InvSubBytes(state)
    InvAddRoundKey(state, w[Nr*Nb, (Nr+1)* Nb-1])
    out = state
end
```

Опис протоколу обміну ключами Діффі-Хеллмана

Ціль алгоритму полягає в тому, щоб два учасники могли безпечно обмінятися ключем, що надалі може використовуватися в якому-небудь алгоритмі симетричного шифрування. Сам алгоритм Діффі-Хеллмана може застосовуватися тільки для обміну ключами.

Алгоритм заснований на труднощі обчислень дискретних логарифмів. Дискретний логарифм визначається в такий спосіб. Уводиться поняття примітивного кореня простого числа Q як числа, чиї ступені створюють всі цілі від 1 до $Q - 1$. Це означає, що якщо A є примітивним коренем простого числа Q, тоді числа: $A \bmod Q$, $A^2 \bmod Q$, . . . , $A^{Q-1} \bmod Q$, є різними й складаються із цілих від 1 до $Q - 1$ з деякими перестановками. У цьому випадку для будь-якого цілого $Y < Q$ і примітивного кореня A простого числа Q можна знайти єдину експоненту X, таку, що:

$$Y = A^X \bmod Q, \text{ де } 0 \le X \le (Q - 1).$$

Експонента X називається дискретним логарифмом, або індексом Y, по підставі $A \bmod Q$. Це позначається як:

$$\text{ind}_{A,Q}(Y).$$

Тепер опишемо алгоритм обміну ключів Діффі-Хеллмана.

Загальновідомі елементи

Q – просте число

A – $A < Q$ і A є примітивним коренем Q

Створення пари ключів клієнтом

Вибір випадкового числа X_i (*закритий ключ*): $X_i < Q$

Обчислення числа Y_i (*відкритий ключ*): $Y_i = A^{Xi} \bmod Q$

Створення відкритого ключа сервером

Вибір випадкового числа X_j (*закритий ключ*): $X_j < Q$

Обчислення випадкового числа Y_j (*відкритий ключ*): $Y_j = A^{Xj} \bmod Q$

Створення спільного секретного ключа клієнтом

$$K = (Y_j)^{Xi} \bmod Q.$$

Створення спільного секретного ключа сервером

$$K = (Y_i)^{Xj} \bmod Q.$$

Розпишемо алгоритм більш докладно. Передбачається, що існують два відомих усім числа: просте число Q і ціле A, що є примітивним коренем Q. Тепер припустимо, що клієнт та сервер хочуть обмінятися ключем для алгоритму симетричного шифрування. Клієнт вибирає випадкове число $X_i < Q$ і обчислює $Y_i = A^{Xi} \bmod Q$. Аналогічно сервер незалежно вибирає випадкове ціле число $X_j < Q$ і обчислює $Y_j = A^{Xj} \bmod Q$. Кожна сторона тримає значення X у секреті й робить значення Y доступним для іншої сторони. Тепер клієнт обчислює ключ як $K = (Y_j)^{Xi} \bmod Q$, і сервер обчислює ключ як $K = (Y_i)^{Xj} \bmod Q$. У результаті обоє одержать те саме значення:

$$K = (Y_j)^{Xi} \bmod Q = (A^{Xj} \bmod Q)^{Xi} \bmod Q = (A^{Xj})^{Xi} \bmod Q =$$

(за правилами модульної арифметики)

$$= A^{Xj\,Xi} \bmod Q = (A^{Xj})^{Xj} \bmod Q = (A^{Xi} \bmod Q)^{Xj} \bmod Q = (Y_i)^{Xj} \bmod Q$$

Таким чином, дві сторони обмінялися секретним ключем. Так як X_i і X_j є закритими, зловмисник може одержати тільки наступні значення: Q, A, Y_i і Y_j. Тобто дві взаємодіючі сторони по відкритому каналу змогли поширити секретний ключ не розкриваючи його третій стороні.

Для обчислення ключа атакуючий повинен зламати дискретний логарифм, тобто обчислити: $X_j = \text{ind}_{a,\,q}(Y_j)$.

Безпека обміну ключа в алгоритмі Діффі-Хеллмана випливає з того факту, що, хоча відносно легко обчислити експоненти за модулем простого числа, дуже важко обчислити дискретні логарифми. Для великих простих чисел задача вважається нерозв'язною.

Варто помітити, що даний алгоритм уразливий для атак типу "in-the-middle". Якщо зловмисник може здійснити активну атаку, тобто має можливість не тільки перехоплювати повідомлення, але й заміняти їх іншими, він може перехопити відкриті ключі учасників Y_i і Y_j , створити свою пару відкритого й закритого ключа (X_{on}, Y_{on}) і послати кожному з учасників свій відкритий ключ. Після цього кожний учасник обчислить ключ, що буде спільним із зловмисником, а не з іншим учасником. Якщо немає контролю цілісності, то учасники не зможуть виявити подібну підміну.

3.2 Розробка структурної схеми

Розроблене програмне забезпечення представляє із себе набір компонентів призначених для забезпечення політики безпеки як у вже існуючих, так і в створюваних мережних інформаційних системах.

Розроблене програмне забезпечення дозволяє забезпечити захист переданої по мережі інформації, строгу взаємну автентифікацію користувачів і серверів, гнучке розмежування доступу. Для реалізації цих функцій у системі використовуються SSL/TLS протоколи й X.509 цифрові сертифікати, тобто

універсальні, що стали стандартом де-факто, механізми, підтримувані практично всіма розповсюдженими Веб-агентами.

За допомогою розробленого програмного забезпечення легко забезпечуються вимоги по інформаційній безпеці, запропоновані різними Інтернет додатками, такими як сервера платіжних систем, інтернет-магазини, багатопрофільні корпоративні Веб-сервера, що містять інформацію з різним рівнем конфіденційності, B2B системи, системи захищеного документообігу, обміну електронною поштою й багато які інші.

На рисунку 3.2 представлена структурна схема розробленої системи. На цій схемі введені наступні позначення:

– ЕЦП – електроний цифровий підпис;

– ЦС – цифровий сертифікат;

– PKI – інфраструктура відкритих ключів;

– DVCS – Data Validation and Certification Server Protocols – протокол підтвердження даних та сертифікації серверу;

– OCSP – Online Certificate Status Protocol – онлайн протокол статусу сертфікату;

– TSP – Time-Stamp Protocol – протокол часових міток;

– TLS – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет;

– RFC – документ, у якому описується той або інший стандарт.

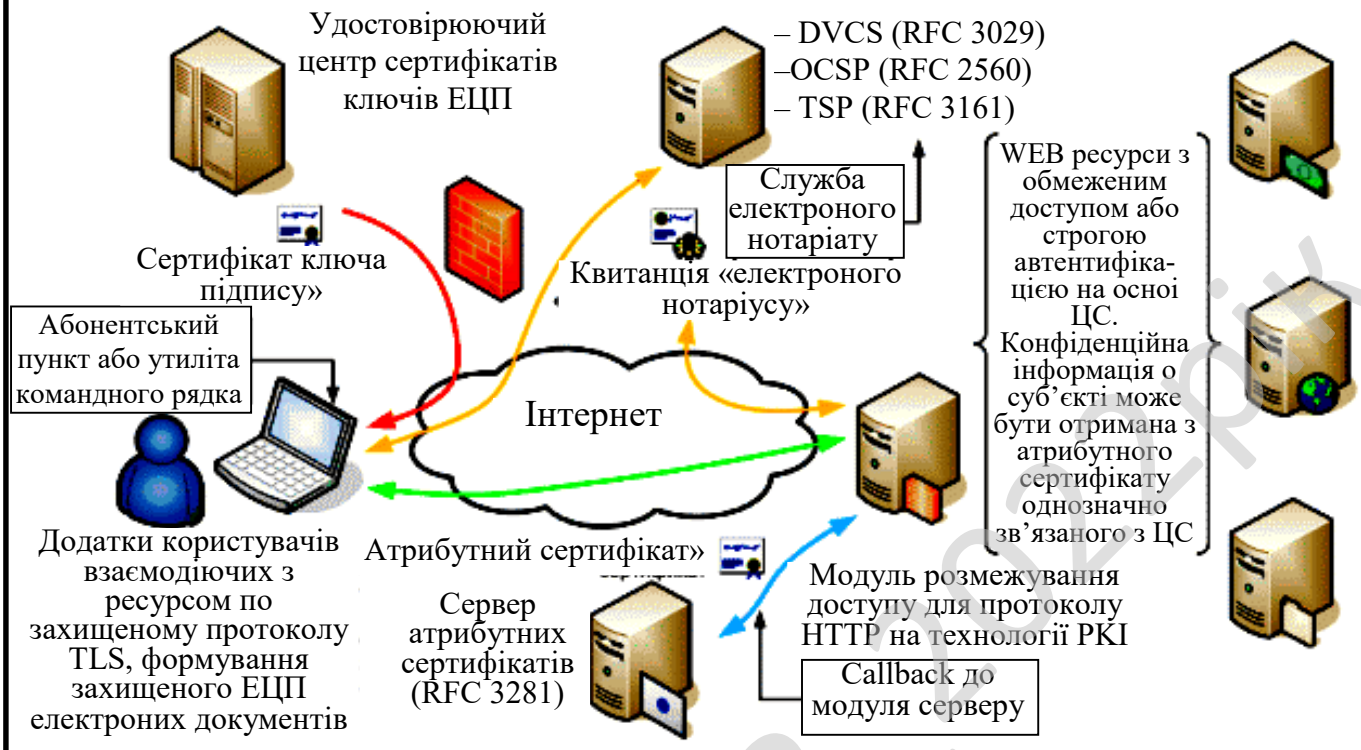


Рисунок 3.2 – Структурна схема

Розглянемо більш детально складові структурної схеми, та принцип роботи системи.

Центр, що засвідчує, сертифікатів ключів підпису

Центр, що засвідчує, сертифікатів ключів підпису (удостовірюючий центр – УЦ) є повністю вітчизняною розробкою:

– відповідає вимогам Доктрини інформаційної безпеки в плані заміщення імпортних технічних і програмних засобів у українських інформаційних системах;

– завдяки відомості всіх криптографічних процедур в ізольований криптографічний PKSC#11 токен, в УЦ можуть використовуватися як вітчизняні криптографічні алгоритми, так і закордонні;

– використовувані вітчизняні криптографічні механізми й протоколи опираються на інтернет драфти, розроблені вітчизняними компаніями, а отже, рішення сумісні з рішеннями інших вітчизняних виробників;

– технічна реалізація заснована на сучасних керівних документах, стандартах і міжнародних рекомендаціях, що дозволяє:

1) в одному технічному рішенні підтримувати невиразно велике число зовсім ізольованих, у тому числі з різними криптографічними алгоритмами, видавців;

2) підтримуються механізми кросування видавців (аж до рівня мостового УЦ), у тому числі й зовнішніх, для утворення єдиних зон обігу захищених документів;

3) для систем наближених до On-line передбачене поширення відновлень списків відкликаних сертифікатів (delta CRL);

4) компоненти самого УЦ для побудови ланцюжків сертифікації використовують внутрішній сервіс OCSP, що може бути оформлений як корпоративний зі складу служби "електронного нотаріату";

5) до складу системи входить власна служба роздачі міток часу, з механізмами підстроювання під зовнішні еталони;

6) відповідно до RFC 3039 Internet X.509 Public Key Infrastructure. Qualified Certificate Profile у сертифікат може бути введений серійний номер імені, тим самим вирішена "колізія імен" – проблема «однофамільців»;

7) реєстр крім самого сертифіката користувача може містити додаткову інформацію про суб'єкта, включаючи графічні елементи (фотографії, відбитки пальців і т.п.);

8) всі інформаційні блоки при транспортуванні й зберіганні захищені електронними цифровими підписами, що забезпечує цілісність, авторство й невідрікаємість і спрощує процедури розбору конфліктних ситуацій.

Центр, що засвідчує, сертифікатів ключів підпису (УЦ) є основою комп’ютерних систем захищеного документообігу на технології відкритого розподілу ключів (Public Key Infrastructure (PKI)). Технічна реалізація Центра, що засвідчує, відповідає вимогам Закону України "Про електронний цифровий підпис" за умови використання в ЦУ сертифікованих ДССЗЗІ засобів

електронного цифрового підпису. УЦ, може виступати як ключовий компонентом для різного типу прикладних захищених систем корпоративного рівня (захищений документообіг, Інтернет-банкінг, білінгові системи, електронна комерція (B2C, B2B), Інтернет процесінг і.т.п.).

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,
    ---і якщо є присутнім, версія повинна
    ---і бути v2 або v3
    subjectUniqueID [2] IMPLICIT
        UniqueIdentifier OPTIONAL,
    ---і якщо є присутнім, версія повинна бути
    ---і v2 або v3
    extensions [3] EXPLICIT Extensions OPTIONAL
    ---і 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.

Служба "Електронного нотаріату"

"Електронний нотаріус" (ЕН) може бути як додатковим сервісом в комп’ютерній системі, що виконує функції Центра, що засвідчує (УЦ) сертифікатів ключів підпису в якості стандартизованогоRFC 3029 Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols (DVCS). RFC 2560 Online Certificate Status Protocol – OCSP. RFC 3161 Time-Stamp Protocol (TSP)) технічного рішення «Про Електронний цифровий підпис», так і як самостійний програмний комплекс, і реалізовувати разову або абонентську

послугу з перевірки й сертифікації інформації, перевірки сертифікатів і виробленню квитанції, що містять «штамп» часу. В ЕН зведені служби, технічна реалізація яких стандартизована міжнародними рекомендаціями, по фактах усіляких перевірок, підтверджень і вироблення «штампа часу» для зовнішніх компонентів інфраструктури відкритих ключів.

"Електронний нотаріус" виконує наступні функції:

– Посвідчення факту володіння інформацією з або без її подання сервісу.

– Перевірка дійсності ЕЦП.

– Перевірка дійсності сертифіката відкритого ключа (для компонентів DVCS або OCSP).

– Вироблення квитанції, що містить «штамп» часу (TSP).

Задачі, у рішенні яких, може бути використана Служба "Електронного нотаріату":

1. Створення єдиного домена захищеного електронного документообігу, у тому числі побудованому на несумісних між собою засобах криптографічного захисту інформації, але маючих сертифікат ДССЗЗІ на засоби криптографічного захисту інформації для гетерогенних програмно-апаратних платформ.

2. Одержання штампа «дійсного часу для даної PKI системи» на завіреному електронному документі. Досить важливо (для попередження шахрайських дій або колізій) при завіренні електронного документа коректно вказувати дату підписання, однак проставляння дійсної дати цілком є відповідальністю підписується стороны, що. ЕН у цьому випадку є «третьою» стороною – довіреному арбітром, що фіксує факт наявності дійсної ЕЦП на конкретний момент часу. Даний сервіс іноді називають Time Stamping. Наявність «третьої» незалежної сторони може виявитися корисною щоб зафіксувати певний етап у технологічному ланцюжку документообігу (якийсь документ пройшов яку те стадію свого формування), наприклад, на конкретний момент часу податкова декларація завірена й доставлена від платника податків в інспекцію. У більше

широкому змісті ЕН може бути використаний абстрактною прикладною системою як джерело TSP міток «еталонного часу».

3. Тривале архівне зберігання електронних документів. ЕЦП на електронному документі має «строк життя», що, зокрема визначається «строком життя» персонального сертифіката, який бере участь у формуванні ЕЦП. Через багато причин цей час досить обмежений, що не дозволяє будувати повноцінну систему документообігу, включаючи такий важливий компонент як архівне зберігання. Наявність DVC квитанцій по перевірці ЕЦП дозволяє робити висновки про дійсність ЕЦП уже після витікання часу дійсності сертифіката, що брав участь у виробленні даної ЕЦП. Дана властивість пояснюється тим, що сертифікат ЕН (DVCS), яким завірена квитанція, більш тривала й існують механізми пролонгації квитанцій (випуск квитанції на квитанцію).

4. Організація перевірки ЕЦП «третьою» стороною для користувачів дозволяє перевести сам факт перевірки із площини криптографічних обчислень на сертифікованих серверах захисту інформації в площину організації довіреної доставки квитанцій із сервера ЕН, що в багатьох випадках значно технологичніше. Ступінь «доручення» доставки квитанцій не регламентується законом «Про ЕЦП» і цілком визначається специфікою комп'ютерної системи, у якій використовуються завірені документи. Способів доставки досить багато: від доставки квитанції кур'єрською службою, підтвердженням по телефоні, порівнянням самих файлів – квитанцій отриманих по мережі й з репозиторія ЕН (DVCS), до організації захищених сегментів мережі, на підтвердження що мають, наприклад, атестат ДССЗЗІ.

5. Покладання на сервіс ЕН функцію перевірки дійсності якогось цифрового сертифіката істотно спрощує комп'ютерну систему, у якій циркулюють завірені електронні документи. Сама по собі процедура перевірки сертифіката досить трудомістка, необхідно побудувати ланцюжок перевірки кінцевого сертифіката з перевіркою всіх проміжних кореневих сертифікатів,

визначити місце розповсюдження, одержати й обробити списки відкликаних сертифікатів і т.п.

6. Даний сервіс може бути досить корисний для комп'ютерних систем (КС), у яких використовується факт володіння користувачем якоюсь інформацією без її опублікування. Наприклад, КС проведення різних тендерів, регламент яких визначає, що до певного строку ніхто не повинен мати доступ до конкурсного матеріалу (за винятком коротких анотацій) і тільки по настанню часу початку конкурсу «конверти» повинні бути розкриті. Для таких систем учасники представляють квитанції на істинність ЕЦП конкурсного матеріалу без фактичної передачі самого матеріалу до моменту настання конкурсу. Істинність представленого в наслідку матеріалу підтверджено ЕЦП зі складу DVC квитанції. У цьому випадку захист конкурсного матеріалу покладає на самих конкурсантів – самих зацікавлених у захисті даних осіб і повністю знімає ризик шахрайства в системі.

Служба атрибутування (реалізація заснована на RFC 3281) вирішує дві задачі:

– Дозволяє здійснити криптографічний зв'язок сертифіката ключа підпису з додатковою інформацією, захищеної ЕЦП, що визначає роль власника сертифіката в КС, наприклад, для цілей розмежування доступу, розміщення персональної інформації, розміщення інформації уточнюючого повноваження й т.п.

– Дозволяє здійснити криптографічний зв'язок між абстрактним блоком даних і додатковою інформацією (метаданими), наприклад, такий атрибутний контейнер можна асоціювати з електронним документом разом з метаданими при міжсистемному (міжвідомчому) інформаційному обміні. Додатково, використовувана технологія дозволяє (ЕЦП у вигляді CMS або PKCS#7, або «підпис із розширеними даними для перевірки» по ETSI TS 101 733 не може це забезпечити) ввести поняття строку дійсності документа, включаючи механізми екстреного визнання розміщеної в контейнері інформації недійсною. Дана

унікальна властивість може бути використана при випуску електронних дозволів, наприклад, імпортно-карантинні дозволи, ліцензії (у тому числі й на програмне забезпечення) і т.п.

3.3 Розробка функціональної схеми

У якості функціональних схем, так як вони є складовими частинами структурних схем наведемо функціональну схему виконання дій по протоколу OCSP та функціональну схему роботи протоколу SSL/TLS. Ці протоколи є складовими системи доступу до хмарних сервісів з використанням технології PKI.

На рисунку 3.3 наведена функціональна схема OCSP.

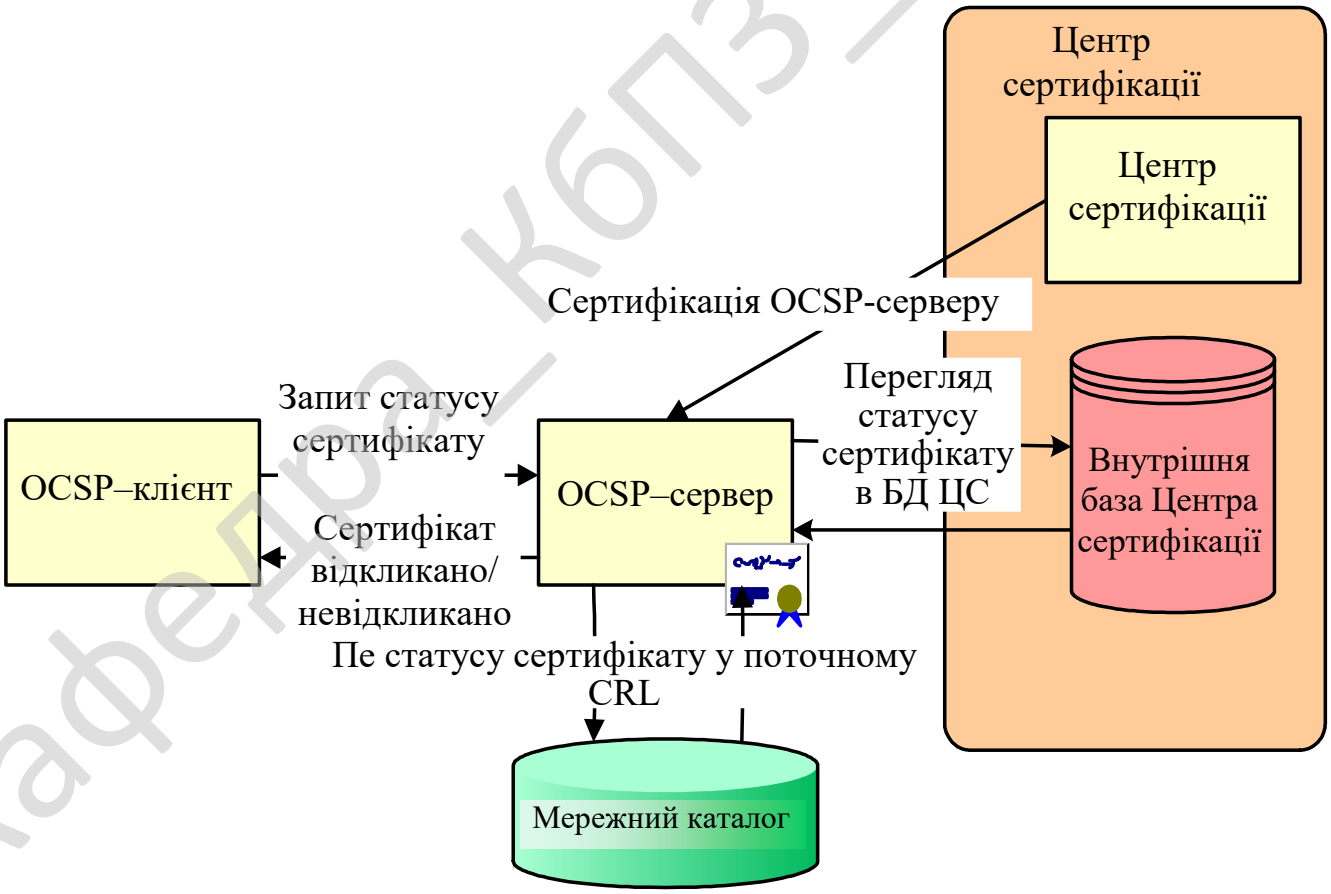


Рисунок 3.3 – Функціональна схема OCSP

На рисунку 3.4 зображена функціональна схема роботи протоколу SSL/TLS.

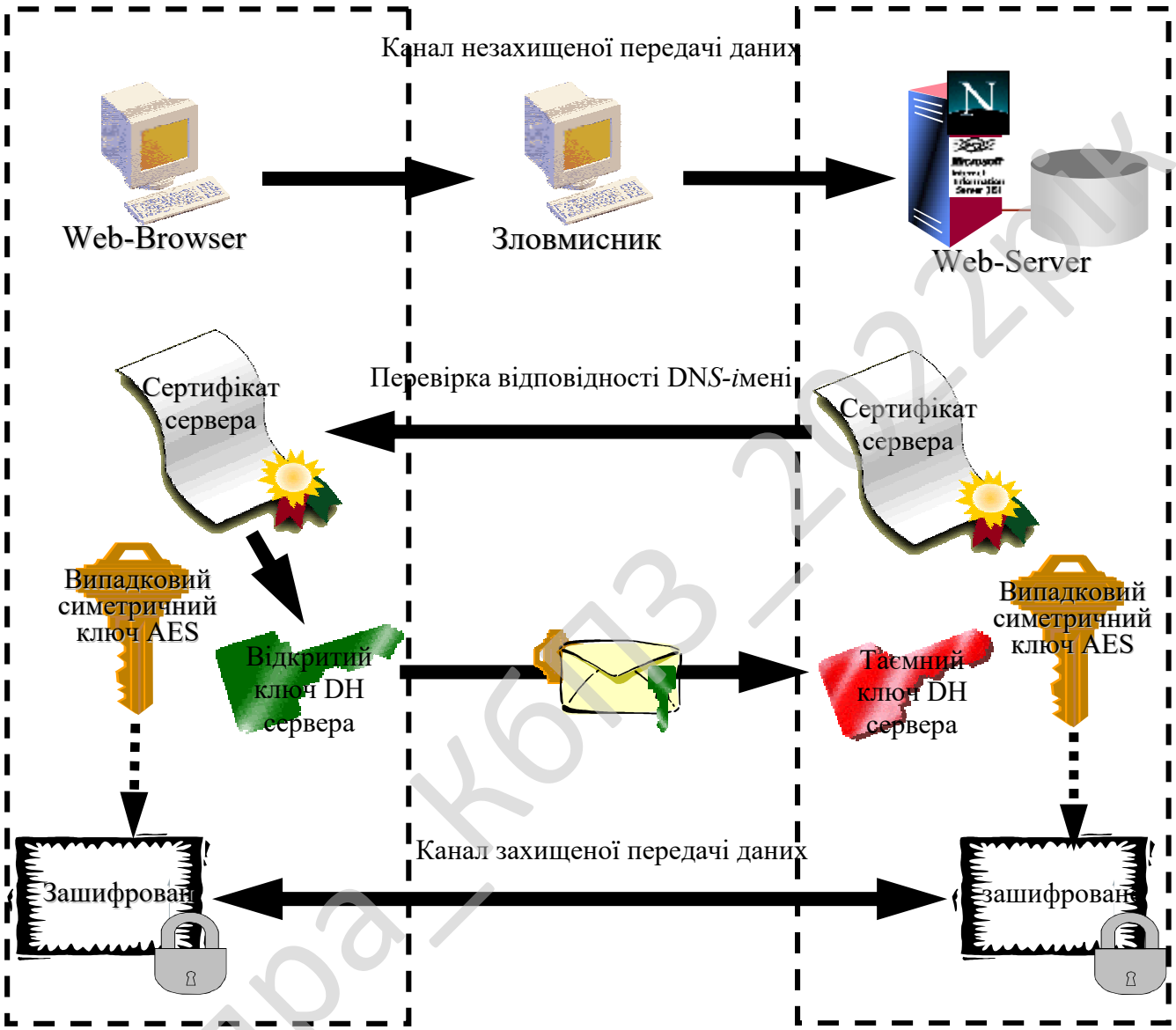


Рисунок 3.4 – Функціональна схема роботи протоколу SSL/TLS

Робота протоколу відбувається наступним чином:

1. Клієнт посилає повідомлення ClientHello, указуючи найбільш останню версію підтримуваного TLS протоколу, випадкове число й список підтримуваних методів шифрування й стиски, що підходять для роботи з TLS.

2. Сервер відповідає повідомленням ServerHello, що містить: обрану

сервером версію протоколу, випадкове число, послане клієнтом, що підходить алгоритм шифрування AES й стиски зі списку наданого клієнтом.

3. Сервер посилає повідомлення Certificate, що містить цифровий сертифікат сервера.

4. Сервер може запросити сертифікат у клієнта, у такому випадку з'єднання буде взаємно автентифіковано.

5. Сервер відсилає повідомлення ServerHelloDone, що ідентифікує закінчення handshake.

6. Клієнт відповідає повідомленням ClientKeyExchange, що містить PreMasterSecret відкритий ключ Діффі-Хеллмана.

7. Клієнт і сервер, використовуючи PreMasterSecret ключ Діффі-Хеллмана і випадково згенеровані числа, обчислюють загальний секретний ключ Діффі-Хеллмана. Вся інша інформація про ключ Діффі-Хеллмана буде отримана із загального секретного ключа Діффі-Хеллмана (і згенерованих клієнтом і сервером випадкових значень).

8. Клієнт посилає ChangeCipherSpec повідомлення, що вказує на те, що вся наступна інформація буде зашифрована встановленим у процесі handshake алгоритмом AES, використовуючи загальний секретний ключ Діффі-Хеллмана. Це повідомлення рівня записів і тому має тип 20, а не 22.

9. Клієнт посилає повідомлення Finished, що містить геш MD-5 і MAC, згенеровані на основі попередніх повідомлень handshake.

10. Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити геш MD-5 і MAC. Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване.

11. Сервер посилає ChangeCipherSpec і зашифроване Finished повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

Із цього моменту handshake уважається завершеним, протокол установленим. Весь наступний вміст пакетів іде з типом 23, а всі дані будуть зашифровані AES.

Геш-функція MD-5 працює наступним чином. Вона отримує на вході повідомлення довільної довжини і створює на виході дайджест повідомлення довжиною 128 біт. Алгоритм складається з наступних кроків:

1. Додавання недостаючих біт. Повідомлення доповнюється так, щоб його довжина стала рівна 448 по модулю 512 (довжина 448 mod 512). Це означає, що довжина доданого повідомлення на 64 біта менше, ніж число, кратне 512. Додавання проводиться завжди, навіть якщо повідомлення має потрібну довжину. Наприклад, якщо довжина повідомлення 448 біт, воно доповнюється 512 бітами до 960 біт. Таким чином, число біт, що додаються, знаходиться в діапазоні від 1 до 512. Додавання складається з одиниці, за якою слідує необхідна кількість нулів.

2. Додавання довжини. 64-бітове представлення довжини початкового (до додавання) повідомлення в бітах приєднується до результату першого кроку. Якщо первинна довжина більша, ніж 264, то використовуються тільки останні 64 біта. Таким чином, поле містить довжину початкового повідомлення по модулю 264. В результаті перших двох кроків створюється повідомлення, довжина якого кратна 512 бітам. Це розширене повідомлення представляється як послідовність 512-бітових блоків Y_0, Y_1,..., Y_{l-1}, при цьому загальна довжина розширеного повідомлення рівна L*512 бітам. Таким чином, довжина отриманого розширеного повідомлення кратна шістнадцяти 32-бітовим словам.

3. Ініціалізація MD-буфера. У алгоритмі Md5 використовується 128-бітовий буфер для зберігання проміжних і остаточних результатів геш-функції. Буфер може бути представлений як чотири 32-бітові регістри (*A*, *B*, *C*, *D*). Ці регістри ініціалізувалися наступними шістнадцатерічними числами:

A = 01234567

B = 89abcdef

C = Fedcba98

D = 76543210

4. Обробка послідовності 512-бітових (16-словних) блоків. Основою алгоритму Md5 є модуль, що складається з чотирьох циклічних обробок, позначений як Hmd5. Чотири цикли мають схожу структуру, але кожен цикл використовує свою елементарну логічну функцію, *ff*, що позначається, *fg*, *fh* і *fi* відповідно.

Кожен цикл приймає на вхід поточний 512-бітовий блок Y_q, що обробляється в даний момент, і 128-бітове значення буфера ABCD, яке є проміжним значенням дайджесту, і змінює вміст цього буфера. Кожен цикл також використовує четвертую частина 64-елементної таблиці *T*[1 . 64], побудованої на основі функції sin. *i*-ий елемент *T*, *T*[i], що позначається, має значення, рівне цілій частині від 232 * abs (sin (*i*)), і задане в радіанах. Оскільки abs (sin (*i*)) є числом між 0 і 1, кожен елемент *T* є цілим, яке може бути представлене 32 бітами. Таблиця забезпечує "випадковий" набір 32-бітових значень, які повинні ліквідовувати будь-яку регулярність у вхідних даних.

Для отримання *Mdq*+1 вихід чотирьох циклів складається по модулю 232 з *Mdq*. Складання виконується незалежно для кожного з чотирьох слів в буфері.

5) Вихід Md5. Після обробки всіх *L* 512-бітових блоків виходом *L*-ої стадії є 128-бітовий дайджест повідомлення.

Детальніше логіку кожного з чотирьох циклів виконання одного 512-бітового блоку розглянуто нижче. Кожен цикл складається з 16 кроків, що оперують з буфером *ABCD*. *A* *B* + Clss (*A* + *f* (*B*, *C*, *D*) + *X* [*k*] + *T* [*i*]), де *A*, *B*, *C*, *D* – чотири слова буфера; після виконання кожного окремого кроку відбувається циклічне зрушення вліво на одне слово; *f* – одна з елементарних функцій *ff*, *fg*, *fh*, *fi*; CLSs – циклічне зрушення вліво на *s* біт 32-бітового аргументу; *X* [*k*] – *M* [*q* * 16 + *k*] – кодує 32-бітове слово в *q*-ому 512 блоці повідомлення; *T* [*i*] – *i*-е 32-бітове слово в матриці T; + – складання по модулю 232.

На кожному з чотирьох циклів алгоритму використовується одна з чотирьох елементарних логічних функцій. Кожна елементарна функція отримує три 32-бітові слова на вході і на виході створює одне 32-бітове слово. Кожна

функція є безліччю побітових логічних операцій, тобто *n*-ий біт виходу є функцією від n-ого біта трьох входів. Елементарні функції наступні:

ff = (*B* & *C*) (not *B* & *D*)

fg = (*B* & *D*) V (*C* & not *D*)

fh = *B* *C* *D*

fi = *C* (*B* & not *D*)

Масив з 32-бітових слів *X* [0..15] містить значення поточного 512-бітового вхідного блоку, який обробляється зараз. Кожен цикл виконується 16 разів, а оскільки кожен блок вхідного повідомлення обробляється в чотирьох циклах, то кожен блок вхідного повідомлення обробляється по схемі 64 рази. Якщо представити вхідний 512-бітовий блок у вигляді шістнадцяти 32-бітових слів, то кожне вхідне 32-бітове слово використовується чотири рази, по одному разу в кожному циклі, і кожен елемент таблиці T, що складається з 64 32-бітових слів, використовується тільки один раз. Після кожного кроку циклу відбувається циклічне зрушення вліво чотирьох слів *A*, *B*, *C* і *D*. На кожному кроці змінюється тільки одне з чотирьох слів буфера *ABCD*. Отже, кожне слово буфера змінюється 16 разів, і потім 17-й раз в кінці для отримання остаточного виходу даного блоку.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.5.

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 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.

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.

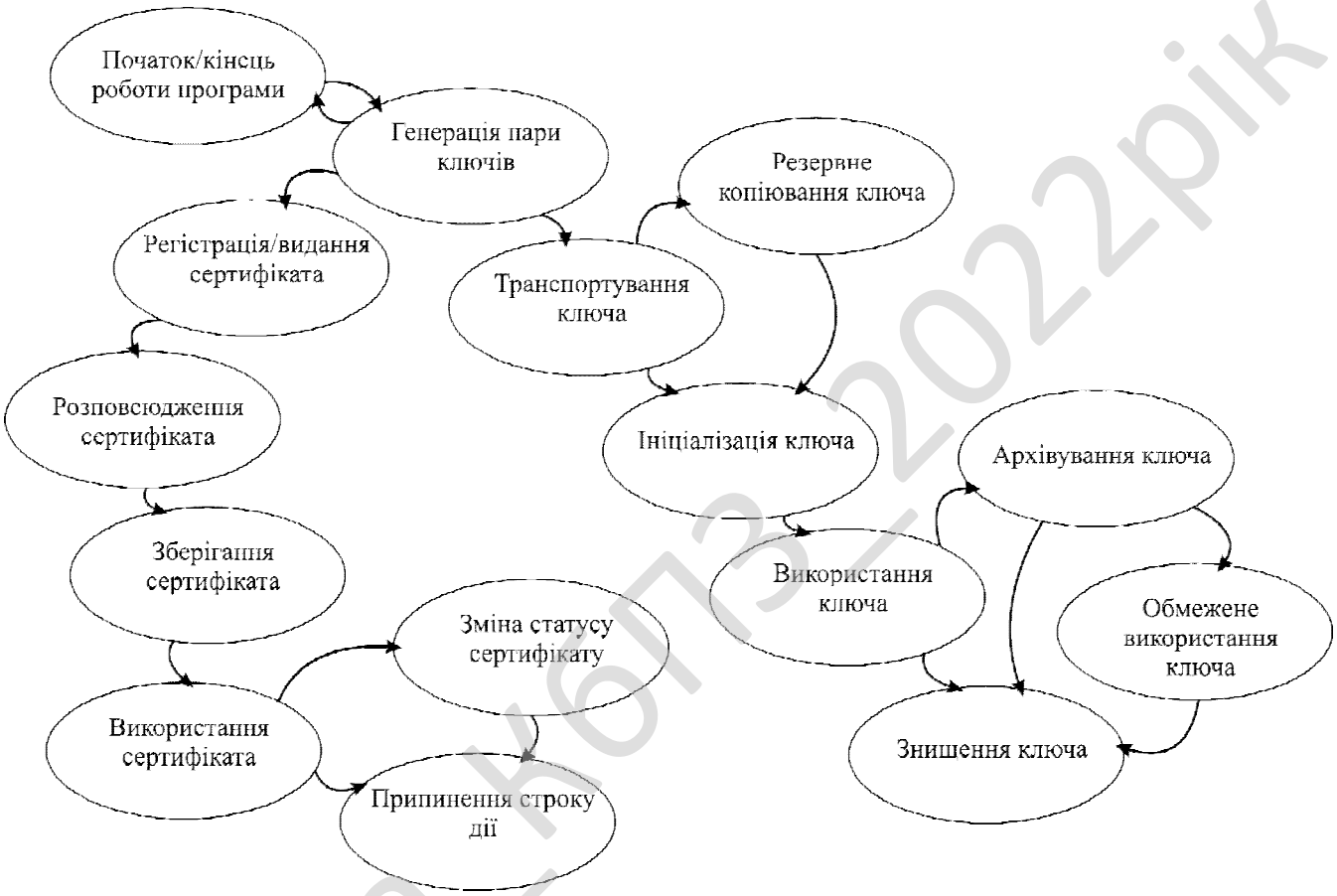


Рисунок 3.5 – Діаграма процесів системи

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 зображена блок-схема роботи основної програми.

Після запуску розробленої системи на екран виводиться основне вікно програми. Перед з'єднанням з сервером користувач повинен ввести параметри з'єднання. Після того як користувач вперше підключиться до сервера, він повинен згенерувати та надіслати запит на отримання сертифіката та закритого ключа. Маючи сертифікат та закритий ключ користувач може запустити підпрограму обміну ключами з сервером та встановлення захищеного з'єднання. У разі успішного обміну ключами встановлюється захищене з'єднання і користувач може працювати з хмарними сервісами по захищеному каналу. У разі невдалої спроби автентифікації користувач повинен спробувати поновити свій сертифікат та ключ.

На рисунку 4.2 зображена блок-схема роботи підпрограми обміну ключами та встановлення захищеного з'єднання.

Як видно з рисунку спочатку клієнт посилає повідомлення ClientHello, що включає:

- останню версію підтримуваного TLS протоколу;
- випадкове число;
- список підтримуваних методів шифрування;
- стиснення, які підходять для роботи з TLS.

Потім сервер відповідає повідомленням ServerHello, що містить:

- обрану сервером версію протоколу;
- випадкове число, надіслане клієнтом;
- підходящий алгоритм шифрування;
- стиснення зі списку наданого клієнтом.

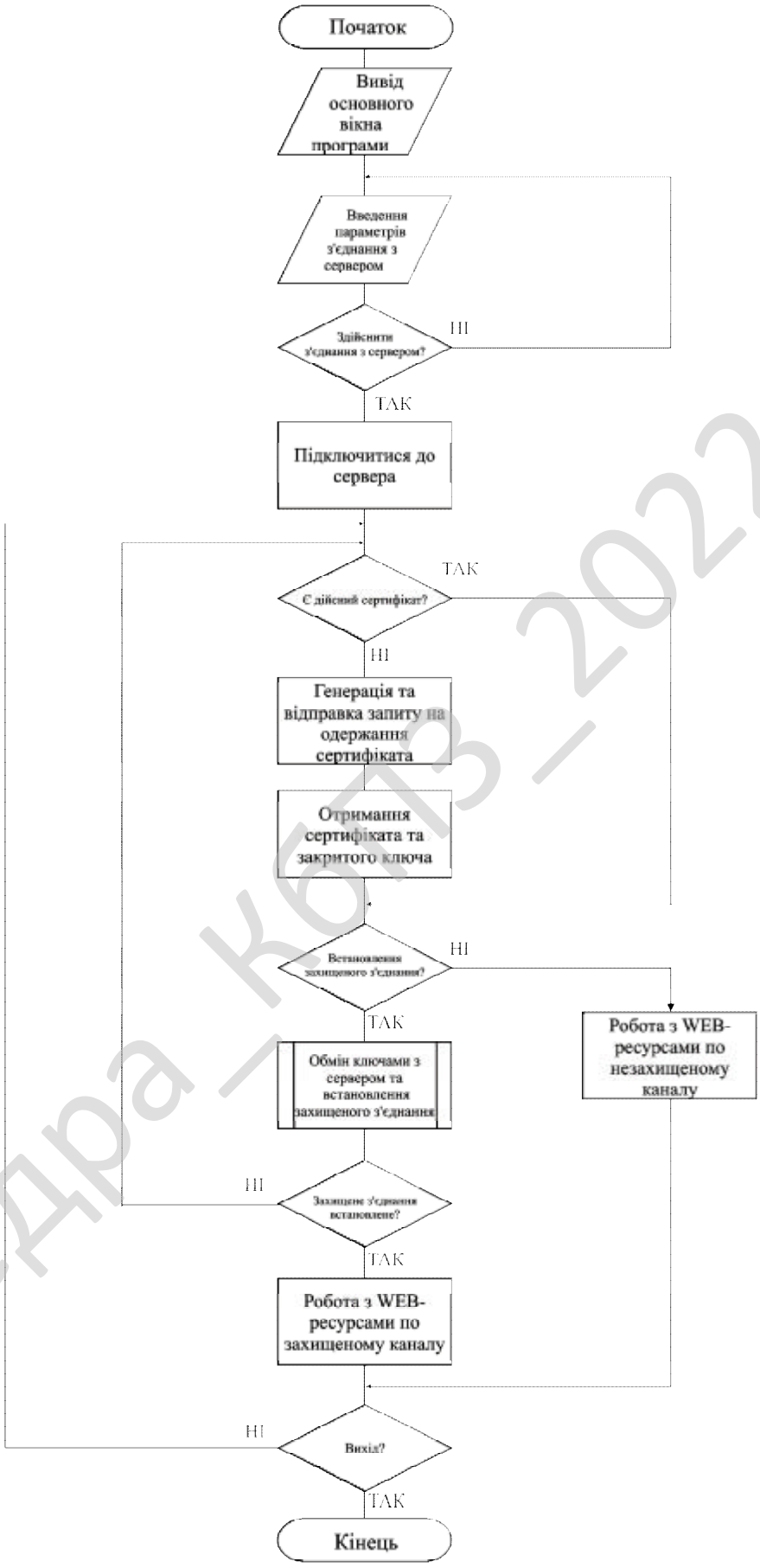


Рисунок 4.1 – Блок-схема роботи основної програми

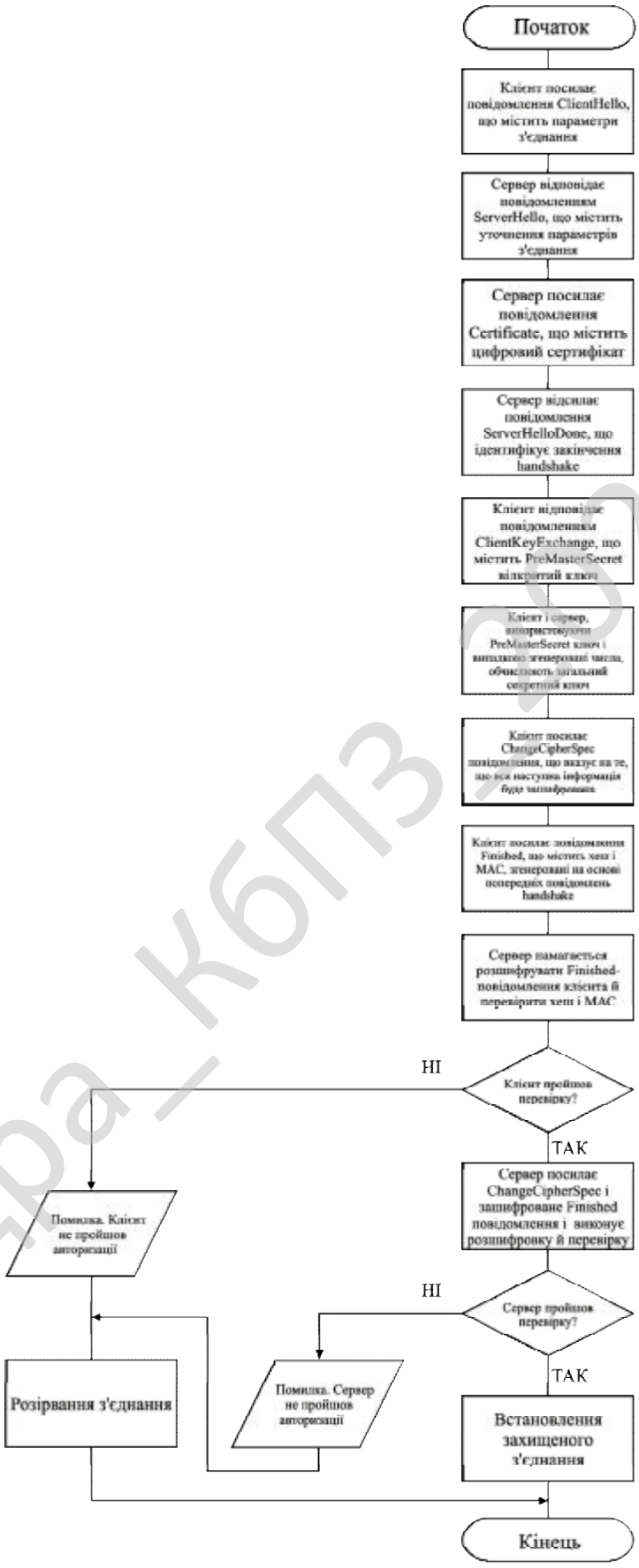


Рисунок 4.2 – Блок-схема роботи підпрограми обміну ключами та встановлення захищеного з'єднання

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 посилання клієнтом повідомлення ClientHello 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 геш і MAC, згенеровані на основі попередніх повідомлень handshake.

Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити геш 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 перегляду руко потискань 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
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::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

Захист розробленого програмного забезпечення буде відбуватися за допомогою IDEA – симетричний блоковий алгоритм шифрування даних, запатентований швейцарською фірмою Ascom. Відомий тим, що застосовувався в пакеті програм шифрування PGP. У листопаді 2000 року IDEA був представлений як кандидат у проекті NESSIE в рамках програми Європейської комісії IST (англ. Information Societes Technology, інформаційні громадські технології).

Першу версію алгоритму розробили в 1990 році Лай Сюецзя (Xuejia Lai) і Джеймс Мессі (James Massey) зі Швейцарського інституту ETH Zürich (за контрактом з Hasler Foundation, яка пізніше влилася в Ascom-Tech AG) як заміна DES (англ. Data Encryption Standard, стандарт шифрування даних) і назвали її PES (англ. Proposed Encryption Standard, запропонований стандарт шифрування). Потім, після публікації робіт Біхамом і Шаміра по диференціальному криптоаналізу PES, алгоритм був поліпшений з метою посилення криптостійкості і названий IPES (англ. Improved Proposed Encryption Standard, покращений запропонований стандарт шифрування). Через рік його перейменували в IDEA (англ. International Data Encryption Algorythm).

Так як IDEA використовує 128-бітний ключ і 64-бітний розмір блоку, відкритий текст розбивається на блоки по 64 біт. Якщо таке розбиття неможливо, останній блок доповнюється різними способами певною послідовністю біт. Для уникнення витоку інформації про кожному окремому блоці використовуються різні режими шифрування. Кожен вихідний незашифрований 64 – біт ний блок ділиться на чотири підблока по 16 біт кожен, так як всі алгебраїчні операції, що використовуються в процесі шифрування, відбуваються над 16-бітними числами. Для шифрування і розшифрування IDEA використовує один і той же алгоритм.

Позначення операцій:

- $\boxplus$ Додавання за модулем 2^{16}.
- $\odot$ Множення за модулем $2^{16}+1$.
- $\oplus$ Побітова виключна диз'юнкція.

Фундаментальним нововведенням в алгоритмі є використання операцій з різних алгебраїчних груп, а саме:

Додавання за модулем 2^{16}.

Множення за модулем $2^{16}+1$.

Побітова виключна диз'юнкція (XOR).

Ці три операції несумісні в тому сенсі, що ніякі дві з них не задовольняють дистрибутивному закону, тобто:

$$a \odot (b \oplus c) \neq (a \odot b) \oplus (a \odot c).$$

Застосування цих трьох операцій ускладнює криптоаналіз IDEA в порівнянні з DES, який базується виключно на операції виключає АБО, а також дозволяє відмовитися від використання S-блоків і таблиць заміни. IDEA є модифікацією мережі Фейстеля.

Генерація ключів

З 128-бітного ключа для кожного з восьми раундів шифрування генерується по шість 16-бітних подключів, а для вихідного перетворення генерується чотири 16-бітних підключа. Всього буде потрібно 52 = 8 x 6 + 4

різних подключів по 16 біт кожен. Процес генерації п'ятдесяти двох 16-бітних ключів полягає в наступному:

Насамперед, 128-бітний ключ розбивається на вісім 16-бітних блоків. Це будуть перші вісім подключів по 16 біт кожен – $(K_1^{(1)}K_2^{(1)}K_3^{(1)}K_4^{(1)}K_5^{(1)}K_6^{(1)}K_1^{(2)}K_2^{(2)})$

Потім цей 128-бітний ключ циклічно зсувається вліво на 25 позицій, після чого новий 128-бітний блок знову розбивається на вісім 16-бітних блоків. Це вже наступні вісім подключів по 16 біт кожен – $(K_3^{(2)}K_4^{(2)}K_5^{(2)}K_6^{(2)}K_1^{(3)}K_2^{(3)}K_3^{(3)}K_4^{(3)})$

Процедура циклічного зсуву і розбивки на блоки триває до тих пір, поки не будуть згенеровані всі 52 16-бітних підключа.

Шифрування

Структура алгоритму IDEA показана на рисунку 4.3.

Процес шифрування складається з восьми однакових раундів шифрування і одного вихідного перетворення. Вихідний незашифрований текст ділиться на блоки по 64 біта. Кожен такий блок ділиться на чотири підблока по 16 біт кожен. На рисунку ці підблоки позначені D_1, D_2, D_3, D_4. У кожному раунді використовуються свої підключі згідно з таблицею підключів. Над 16-бітними підключами і підблока незашифрованого тексту проводяться наступні операції:

– Множення за модулем $2^{16}+1 = 65537$, причому замість нуля використовується 2^{16}.

– Додавання за модулем 2^{16}.

– Побітове виключне АБО.

В кінці кожного раунду шифрування є чотири 16-бітних підблоки, які потім використовуються як вхідні підблоки для наступного раунду шифрування. Вихідна перетворення являє собою скорочений раунд, а саме, чотири 16-бітних підблоки на виході восьмого раунду і чотири відповідних підключа піддаються операціям:

– Множення за модулем $2^{16}+1$.

– Додавання за модулем 2^{16}.

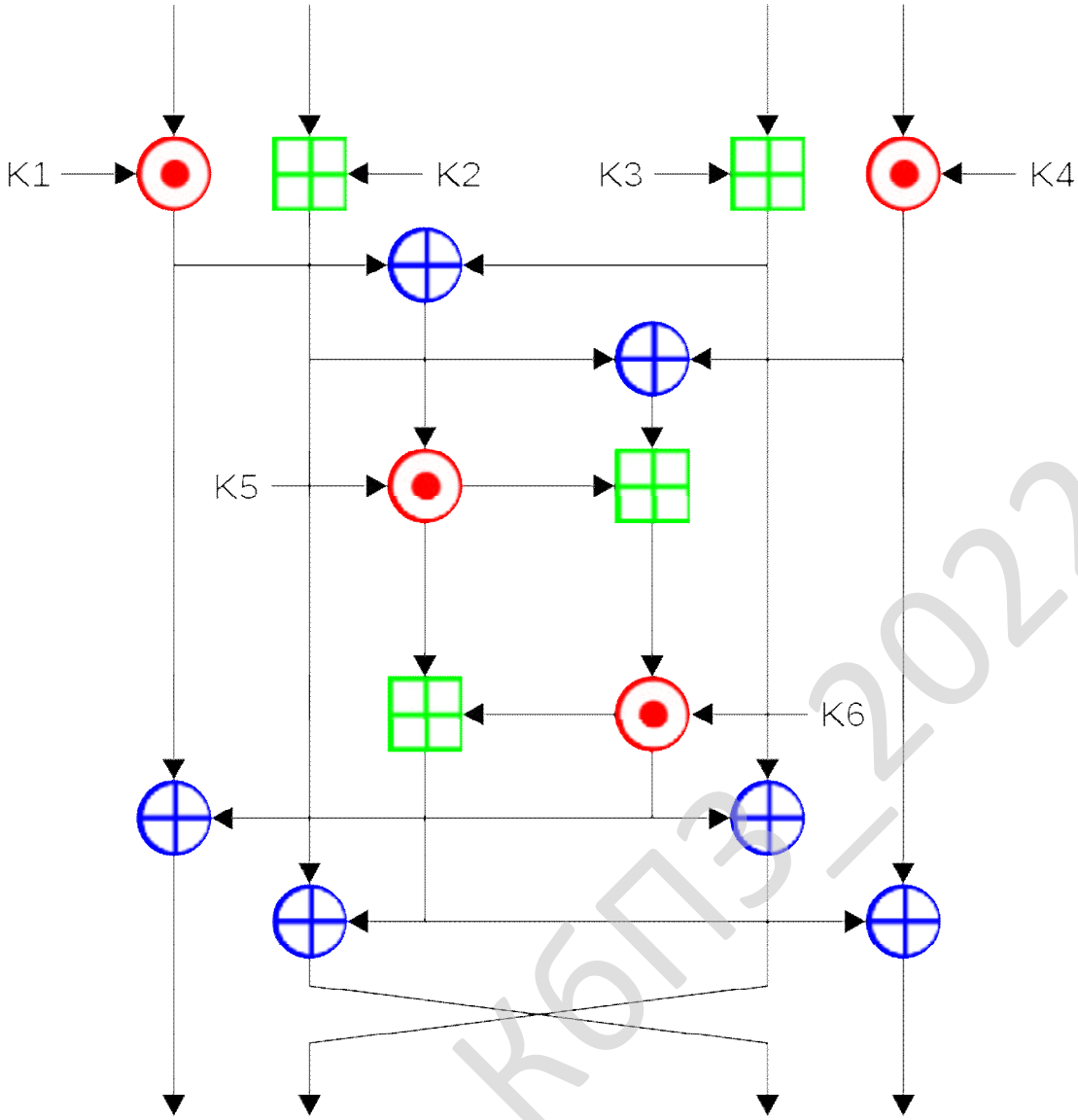


Рисунок 4.3 – Структура алгоритму IDEA

Після виконання вихідного перетворення конкатенація підблоків D_1', D_2', D_3' і D_4' являє собою зашифрований текст. Потім береться наступний 64-бітний блок незашифрованого тексту і алгоритм шифрування повторюється. Так продовжується до тих пір, поки не зашифрують всі 64-бітові блоки вихідного тексту.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма проста у використанні й не вимагає установки на комп'ютері користувача іншого програмного забезпечення або баз даних. Використовуючи комбінацію тунелювання, шифрування, аутентификации й контролю доступу, програма надає користувачам захищений спосіб доступу через Интернет. Головне вікно програми зображене на рисунку 5.1.

Після запуску програми треба ввести IP-адресу серверу та порт, по якому буде відбуватися передача даних. Слід вибрати алгоритм захисту, який буде використовуватися під час роботи з хмарними сервісами. Користувач має можливість використати алгоритм махисту інформації SSL/TLS або AES.

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

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

Головне вікно програми містить наступні кнопки:

– "З'єднання" – для з'єднання з сервером.

– "Роз'єднання" – для розірвання зв'язку з сервером.

– "Сертифікати" – для роботи з сертифікатами (генерації запиту на сертифікат та перегляд наявних сертифікатів).

– "Про програму…" – відкриває довідку.

– "Відкрити Інтернет-браузер у захищеному режимі " – відкриває Інтернет-браузер у захищеному режимі.

В нижній частині вікна відображається стан відкритого з'єднання.

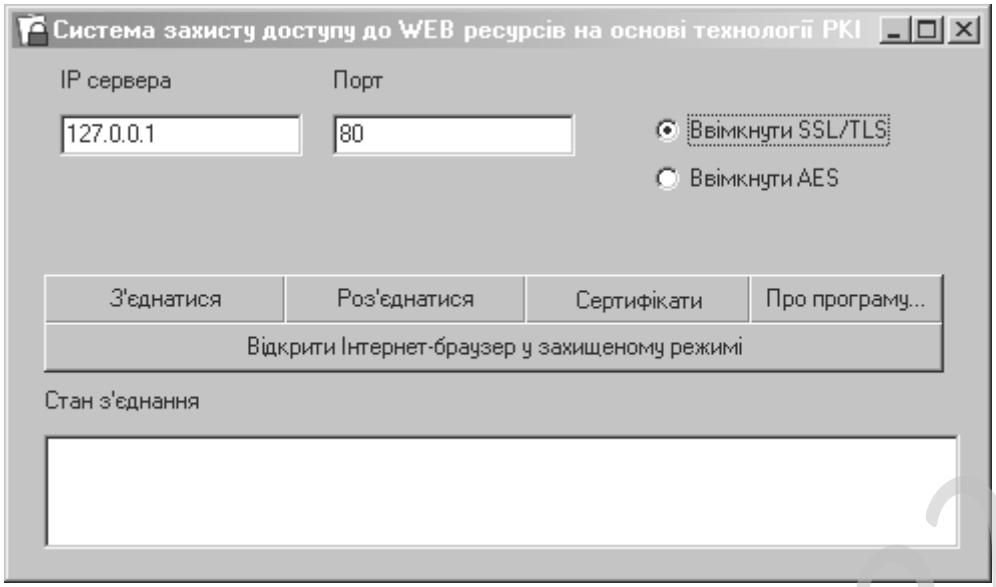


Рисунок 5.1 – Головне вікно програми

Коротку довідку про розроблену програму можна переглянути натиснувши кнопку "Про програму…", після чого з'явиться вікно зображене на рисунку 5.2.

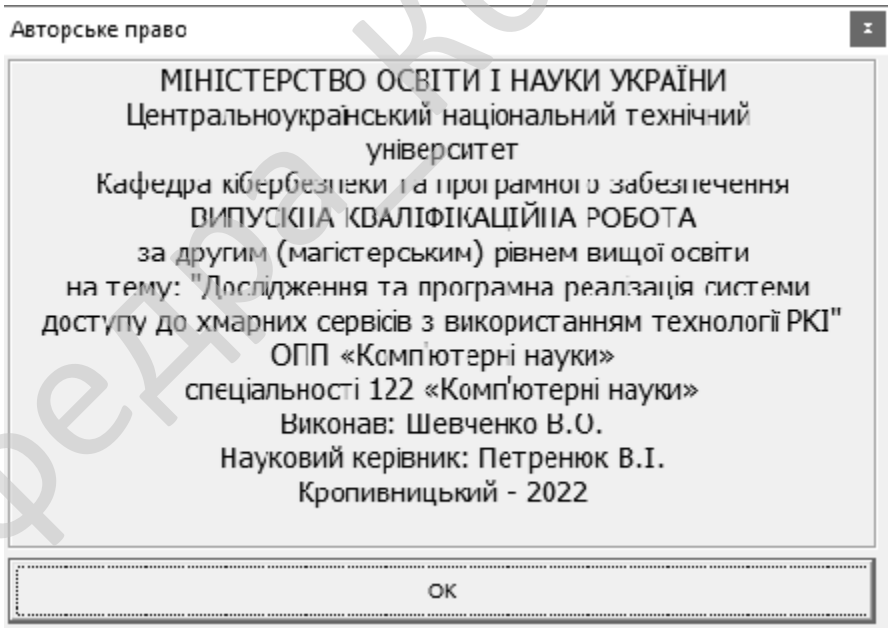


Рисунок 5.2 – Довідка

6 НАУКОВА НОВИЗНА

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

Метою розробки є дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

Об'єктом дослідження є процес доступу до хмарних сервісів з використанням технології PKI.

Предметом дослідження є методи доступу до хмарних сервісів з використанням технології PKI.

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

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

– Удосконалено метод доступу до хмарних сервісів з використанням технології PKI.

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

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

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

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці). В магістерській роботі було проведене дослідження та виконана програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	99
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

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

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	100000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

де: A – коефіцієнт Боема, A = 2,45;

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

B – показник ступеня, що визначається співвідношенням:

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

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

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

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

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

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

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

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

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

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

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

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

$$T_{РП} = 0{,}3 \cdot 2{,}66 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 70 = 118 \text{ люд/день.}$$

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

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	118	Ф 7.1-7.4
Впровадження	13	Д13
Всього	159	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

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

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

$$Ч = \frac{159 \cdot 1}{60 - 5} = 2,9 \text{ ставки.}$$

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

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

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

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор- маршрутизатор	30	1	30	0,5
Кабельні господарства ЛОМ на 1 м. п.	2,5	160	400	6,67
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	37,33

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{37 \cdot 3}{1,2} = 92,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 92,5/(60 \cdot 8) = 0,2 \text{ ставки.}$$

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

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

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

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

Продовження таблиці 7.4

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	7870	23610
Продакт-менеджер	0,25	7000	5250
Інженер-програміст	2,9	8500	73950
Інженер - електронщик	0,2	7000	4200
Інженер-системотехнік	0,25	7000	5250
Адміністратор мережі	0,5	7000	10500
Системний програміст	0,1	7000	2100
Дизайнер WEB	0,25	7000	5250
Інженер-верстальник	0,25	7000	5250
Бухгалтер-економіст	0,1	7000	2100
Всього за період розробки	$R_{сп} = 5,8$	-	$Ф_{роб} = 137460$

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{137460}{5,8 \cdot 60} = 395 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

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

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

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

$$Б_{уд} = 8 \cdot 8 \cdot 29000 = 1858000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 185800 грн.

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

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

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

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Портал за 07.11.22 – джерело http://www.portal.ks.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	AMD FX-6300 3,5-4,1 GHz 6C 8 MB 9 Вт BOX	1750
Системна плата	GIGABYTE GA-78LMT-S2 R AMD760G/FSB HT5200/DDR3 1333/D Sub+PCI-E16x/6 SATA/7.1/mATX	1200
Відеокарта	GIGABYTE GV-N710D3-1GL GeForc GT710 1 GB DDR3 954/1800 MHz 64-bit D Sub+HDMI+DVI	750
Жорсткий диск	3,5" SEAGATE 1 TB 7200 rpm 64 MI ST1000DM003 / ST1000DM010 SATA-3	1200
Оперативна пам’ять	DIMM DDR-3 4 GB 1333 MHz PC 10600 Samsung	900
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bul 22x, SecurDisc, black	416
Корпус	ATX Middle Tower FOXCONN Pro, 3GTLA 489, PSU 350W(FSP Brand: ATX-350PNR 12cm), black, (front bezel – black+light silver body material – 0.6mm), 80mm fan (rear) 2xUSB2.0/AUDIO/MIC, Air Duct, Tool-les chassis design,Thermally Advantaged Chassis	911

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black	220
інше	Клавіатура, мишка	Подару-нок
Монітор	22" LG 22MP58VQ-P 5 мс IPS 1920x108 250/1000M:1 178/178 D-Sub+HDMI+DVI	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	15	10947	16420,5	180625,5
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	199177

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

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1858000	-	-
2. Передавальні пристрої	185800	-	-
Всього по групі	2043800	5	102190
Група 4			
3. Обчислювальна техніка	199177	-	-
Всього по групі	199177	50	99588,5
4. Нематеріальні активи	100000	10	10000
Група 5, 6			
5. Вимірювальні пристрої	9031	25	2257,75
6. Транспортні засоби	143000	20	28600
7. Господарський інвентар	28000	25	7000
Всього по групі	180031	-	37857,75
Разом	$К_р$ = 2523008		$А_р$ = 249636,25

Примітка: вартість автомобіля Sens (Standard+) взята по даним з автосалону «Кіровоград-Авто», джерело http://kirovograd-avto.ukravto.ua /catalog/tm-9/model-80/description, складає 143000 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

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

$$З_о = 395 \cdot 159/99 = 635 \text{ грн.}$$

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

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

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

$$З_д = 635 \cdot 10 \cdot 0{,}01 = 64 \text{ грн.}$$

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

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

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

$$С_{оц} = 0{,}01 \cdot 22(635+64) = 154 \text{ грн.}$$

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

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

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 635 \cdot 15 \cdot 0{,}01 = 95 \text{ грн.}$$

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

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

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

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=206 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 206 \cdot 1,5 = 309 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 51):

$$З_{М2} = \sum Ц_д \,, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 23,6 грн./шт., DVD-R box – 35 грн./шт.

$$З_{М2} = 23,6 \cdot 50 + 35 = 1215 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

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

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

$$З_М = (309 + 1215 + 1702)/99 = 33 \text{ грн.}$$

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

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

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

$$О_п = 635 \cdot 15 \cdot 0,01 = 95 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 99 прим.):

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

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

$$A_м = 249636 \cdot 3/(99 \cdot 12) = 631 \text{ грн.}$$

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	635
2. Додаткова зарплата виконавців	$З_д$	64
3. Відрахування на соціальні потреби	$С_{оц}$	154
4. Загальногосподарські витрати	$Г_{осп}$	95
5. Витрати на матеріали	$З_М$	33
6. Освоєння нових операційних систем, мов програмування	$О_п$	95
7. Амортизація основних фондів	$А_м$	631
8. Повна собівартість програмного забезпечення	$С_п$	1707
9. Плановий прибуток	$П_р$	940
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	2647
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	529,4
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	3176,4

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

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

$$C_п = 635+64+154+95+33+95+631 = 1707 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 50%.

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

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 55 \cdot 1707 = 940 \text{ грн.}$$

7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

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

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

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	3176
Всього капітальних витрат	–	3176

7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	21257	4429
2. Витрати на електроенергію	$З_{ел}$	205	43
3. Витрати на амортизацію	$З_{ам}$	0	794
Всього витрат за рік	$І$	21462	5266

Витрати на обслуговування системи:

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

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

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 240 годин на рік до 50 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 240 \cdot 66 \cdot 1{,}1 \cdot 1{,}22 = 21257\ грн,$$

до:

$$З_{р\ нов} = 50 \cdot 66 \cdot 1{,}1 \cdot 1{,}22 = 4429\ грн.$$

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

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

$$З_{ел\ баз} = 0{,}45 \cdot 240 \cdot 1{,}9 = 205 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}45 \cdot 50 \cdot 1{,}9 = 43 \text{ грн.}$$

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

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	3176	–	794
Всього відрахувань	-	–	3176	–	794

7.8 Визначення економічної ефективності програмної продукції

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

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

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

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

$$Ев = (2647\text{-}1707) \cdot 99\text{-}(0{,}05 \cdot 2043800 + 0{,}5 \cdot 199177 + 0{,}25 \cdot 37031 + $$
$$+ 0{,}1 \cdot 100000 + 0{,}2 \cdot 143000) \cdot 3/12 = 30651 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

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

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

$$T_в = \frac{479208}{(2647\text{-}1707)\cdot 99\cdot 12/3} = 1{,}3 \text{ років}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	99
2. Повна собівартість розробленої програми	Грн.	1707
3. Ціна розробленої програми	Грн.	2647
4. Плановий прибуток від реалізації розробленої програми	Грн.	940
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2523008
7. Загальний прибуток від реалізації програмної продукції	Грн.	93060
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	30651
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	1,3
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3176
11.Величина економічного ефекту у користувача програмної продукції	Грн.	15402
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,2

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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

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

$$E_{сп} = (21462\text{-}5266)\text{-}0{,}25 \cdot 3176 = 15402 \text{ грн.}$$

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

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

$$T_{сп} = \frac{3176}{21462 - 5266} = 0{,}2 \text{ року}.$$

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Електронно-обчислювальна машина (ЕОМ) відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують ЕОМ для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин, роботи тощо. Багато людей користуються ЕОМ у професійних цілях, оскільки завдяки ЕОМ зявилося багато нових професій. Тому для розробника хмарних сервісів так важливо розробити зручний інтерфейс для зручного сприйняття інформації, та необхідний функціонал, який буде відповідати необхідним вимогам та навантаженням. Все це вимагає багато багато часу та великого навантаження з боку розробників.

Тому так важливо слідкувати за умовами праці, в яких відбувається робочий процес. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машина (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

– ризик виникнення надзвичайних ситуацій природного або штучного характеру на об’єкті або території.

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– монотонність праці;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шум;

– статичні навантаження на кістково-м’язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

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

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	6
Довжина	7
Висота	2,9

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

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	7
Об'єм, V	$м^3$	не менше 20.0	20,3

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

У зазначеному приміщенні працюють 6 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [5], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [5] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таким чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість

атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,1

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

В літню пору застосовується кондиціонер.

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

У приміщенні знаходяться наступні джерела шуму: принтер *Prinics PicKit M1 Smartphone Photo Printer White*, електродвигуни вентиляторів ЕОМ.

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

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

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

8.4. Розробка заходів з умов поліпшення охорони праці

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

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

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значеннь;

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

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

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

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

8.5 Розрахункова частина

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 6 м, довжина – 7 м, висота – 2,9 м.

У зазначеному приміщенні працює 4 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F=ESKZ/n,$$

де:

F – світловий потік, що розраховується, Лм;

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

S – площа освітлюваного приміщення (у нашому випадку *S*=6 х 7 = 42 м^2);

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

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

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

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

де:

S – площа приміщення, *S* = 42 м^2;

h – розрахункова висота підвісу, *h* = 2,9 м (співпадає з висотою стелі, т.я. лампи освітлення закріплюються на стелі);

A – ширина приміщення, *A* = 6 м;

B – довжина приміщення, *B* = 7 м.

Підставимо всі значення у формулу та визначемо індекса приміщення:

i=1,4.

Знаючи індекс приміщення, за знаходимо n = 0,29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [8]. Підставимо всі значення у формулу, визначемо світловий потік: *F=71689 Лм.*

Для розрахунку дудемо використовувати світлодіодні стельові панелі *Delux LED Panel 41 44Bm.*, світловий потік яких $F_л$ = *3600 Лм.*

Число ламп визначається за формулою:

$$N=F/F_л$$

де:

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення:

N= 71689 / 3600=19,9 шт.

Приймаємо необхідну кількість світлодіодних світильників 20 шт.

8.6 Висновки до розділу

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

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

9 ОСНОВНІ ВИСНОВКИ

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

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

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

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем доступу до хмарних сервісів з використанням технології PKI.

– Досліджена система доступу до хмарних сервісів з використанням технології PKI.

– На основі отриманих результатів досліджень створена програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI.

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

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм IDEA.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 15402 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,2 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шевченко В.О. Дослідження та програмна реалізація системи доступу до хмарних сервісів з використанням технології PKI // Збірник праць молодих науковців ЦНТУ. – Вип. 13. – Кропивницький: ЦНТУ, 2022.

2. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022, pp. 1-12. **(Scopus).**

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

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

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

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

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

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

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

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

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

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

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

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

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

15. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions».

Lecture Notes in Networks and Systems, vol 152. **Springer**, Cham. 2021, pp 66-84. **(Scopus)**.

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

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

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

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

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

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

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

23. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during

Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019. **(Scopus)**.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи доступу до хмарних сервісів з використанням технології PKI.

2 Підстава для розробки

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

3 Мета та призначення розробки

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

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

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

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи доступу до хмарних сервісів з використанням технології PKI;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

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

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

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті шкідливі і небезпечні фактори при роботі з комп’ютером.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 116 аркушів.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

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

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 20.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Петренюк В.І.

Дослідження та програмна реалізація

системи доступу до хмарних сервісів з використанням технології PKI

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 38

Літера: РП

Кропивницький – 2022 року

Основна програма

SSL.vcproj – Файл конфігурації проекту

```
<?xml version="1.0" encoding = "Windows-1252"?>
<VisualStudioProject
      ProjectType="Visual C++"
      Version="7.00"
      Name="SSL"
      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)/SSL.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)/SSL.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="SSL.cpp">
                  </File>
                  <File
                        RelativePath="SSLServer.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="sslcommon.cpp">
                  </File>
            </Filter>
            <Filter
                  Name="Header Files"
                  Filter="h;hpp;hxx;hm;inl;inc">
                  <File
                        RelativePath="SSL.h">
                  </File>
                  <File
                        RelativePath="SSLServer.h">
```

```
                </File>
                <File
                      RelativePath="Stdafx.h">
                </File>
                <File
                      RelativePath="sslcommon.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 SSL.cpp основної програми

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

    void SSLConnection::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::SSLException*)
        {
            Dispose();
            throw;
        }
    }

    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.");
            }
        }
    }
    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
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::SSLException(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 SSLConnection::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::SSLException(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::SSLException(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 SSLConnection::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::SSLException(S"Максимальни SSL розмір
помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    bool SSLConnection::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::SSLException(S"Помлка з’єднання.
Помилка: ", Status);
        }
        //
        // Будуємо повідомлення про закриття SSL
        //
        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::SSLException("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::SSLSendException(S"Відправлення на
сервер не відбулося.");
            }

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

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

    void SSLConnection::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::SSLException("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::SSLException(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::SSLSendException(S"Відправлення
помилкове. Помилка: ", scRet);
        }

        free(pbIoBuffer);
    }

    void SSLConnection::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::SSLException("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::SSLException("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 SSLConnection::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 SSLConnection::DispatchPlainData(void* pData, long Len, Object* state)
    {
        Byte data[] = new Byte[Len];
        Marshal::Copy(IntPtr(pData), data, 0, Len);
        DoPlainData(data, state);
    }
    void SSLConnection::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::SSLException(S"Failed to load security
dll.");
        m_pSecurityFunc = pSecurityFunc;
        m_ServerIP = NULL;
    }

    void SSLConnection::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::SSLException(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::SSLException(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::SSLException(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 SSLConnection::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 SSL.h – бібліотека для файлу SSL.cpp

```
// SSL.h
/**************************************************************************

#pragma once
#include "sslcommon.h"

namespace SSL
{
    public __value struct ServerCeriticateInfo;
    namespace Client
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WriteSSL(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 SSLConnection;
        //Сервер потребує узгодження, новий сертифікат завантаження, SSLConn-
>LoadNewClientCredentials
        public __delegate void NewCertificate(SSLConnection* SSLConn);

        __sealed public __gc class SSLConnection : public IDisposable
          {
        public:

            SSLConnection();
            ~SSLConnection()
            {
                Dispose(false);
            }
        public:
            //ініціалізація з'єднання, береться ip-адреса сервера і клієнтський
хеш сертифікату,
            //данні які потрібно відіслати були  повернені з WriteSSL
            void InitiateHandShake(String* ipAddress, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state);
            //шифруємо дані, шифруємі дані повертаються в WriteSSL
            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::SSLException(S"Взятий
максимальний розмір токену SSL помилковий. Помилка: ", scRet);
                return psecInfo->cbMaxToken;
            }
        public:
            //Повертаємо зашифровані дані
            WriteSSL*         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 sslcommon.cpp – організація шифрування та формування ланцюжка сертифікатів

```
#include "stdafx.h"
#include "sslcommon.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, SSL::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_SSL,
                              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 =
SSL::Common::Misc::ServerCertChainPolicyStatus(PolicyStatus.dwError);
        CertInfo->CertEncodingType = pServerCert->dwCertEncodingType;
        CertInfo->CertData = CertData;
          bRet    =     false;
    }
    if(pChainContext)
    {
        CertFreeCertificateChain(pChainContext);
    }
    return bRet;
}
```

Файл sslcommon.h – бібліотека для файлу sslcommon.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 SSL
{
namespace Common
{
    namespace Exceptions
    {
    [Serializable]
      public __gc class SSLException : public ApplicationException
    {
    public:
        SSLException(){}
        SSLException(String* message):ApplicationException(message){}
        SSLException(String* message, Exception* innerException)
            :ApplicationException(message,innerException){}
        SSLException(String* message, UInt32 error)
            :ApplicationException(String::Concat(message,
Convert::ToString(error))){}

    };
    [Serializable]
    __sealed public __gc class SSLServerFailedToFindExistingClientID : public
SSLException
    {
    public:
        SSLServerFailedToFindExistingClientID(){}
        SSLServerFailedToFindExistingClientID(String*
message):SSLException(message){}
        SSLServerFailedToFindExistingClientID(String* message, Exception*
innerException)
            :SSLException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class SSLServerDisconnectedException : public
SSLException
    {
    public:
        SSLServerDisconnectedException(){}
        SSLServerDisconnectedException(String* message):SSLException(message){}
        SSLServerDisconnectedException(String* message, Exception*
innerException)
            :SSLException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class SSLReadException : public SSLException
    {
    public:
        SSLReadException(){}
```

```
    SSLReadException(String* message):SSLException(message){}
    SSLReadException(String* message, Exception* innerException)
        :SSLException(message,innerException){}
};

[Serializable]
__sealed public __gc class SSLSendException : public SSLException
{
public:
    SSLSendException(){}
    SSLSendException(String* message):SSLException(message){}
    SSLSendException(String* message, Exception* innerException)
        :SSLException(message,innerException){}
    SSLSendException(String* message, UInt32 error)
        :SSLException(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_SSL3 = SP_PROT_SSL3,
    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 SSL
#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,
SSL::Common::Misc::CeriticateInfo* CertInfo);
```

Файл SSLServer.cpp – Серверна частина

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

namespace SSL
{
namespace Server
{
    SSLServer::SSLServer(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::SSLException(S"Failed to load security
dll.");
        m_pSecurityFunc = pSecurityFunc;
    }
    bool SSLServer::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::SSLSendException(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::SSLException(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::SSLException(S"Неправильна
умова. Помилка: ", scRet);
            }
        }
        return true;
    }

    void SSLServer::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::SSLException(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::SSLException("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::SSLSendException(S"Відправлення на
сервер не відбулося.");
            }
            m_pSecurityFunc->FreeContextBuffer(pbMessage);
        }
        RemoveClient(ClientID);
    }
    void SSLServer::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::SSLException("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::SSLException(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::SSLSendException(S"Помилка
відправлення. Помилка: ", scRet);
        }
        free(pbIoBuffer);
    }

    void SSLServer::DecryptData(Byte data[], Int32 ActualLen, Guid ClientID,
Object* state)
    {
        CLIENTDATA* pClientData=NULL;
        try
        {
                GetClientIDAssoc(ClientID, pClientData);
        }
        catch(Common::Exceptions::SSLServerFailedToFindExistingClientID*)
        {
            //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::SSLException("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::SSLException("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 SSLServer::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 SSLServer::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 SSLServer::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::SSLException(S"Взятий максимальний
розмір блоку SSL помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    void SSLServer::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::SSLException(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::SSLException(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::SSLException(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 SSLServer::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::SSLServerFailedToFindExistingClientID("SSL сервер не знайшов
id клієнта");
        }
            LeaveCriticalSection(m_pCS);
    }
    void SSLServer::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 SSLServer::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::SSLException(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::SSLSendException(S"Відправлення на
сервер не відбулося.");
            }
       }
    }
    void SSLServer::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 SSLServer.h – бібліотека для файлу SSLServer.cpp

```
#pragma once

#include "sslcommon.h"

namespace SSL
{
    namespace Server
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WriteSSL(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 SSLServer : public IDisposable
        {
        public:
            SSLServer();
            ~SSLServer(){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 WriteSSL
            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::SSLException(S"Взятий
максимальний розмір точену SSL 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
            WriteSSL* 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;
        };
    }
}
```