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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки при використані хмарних сервісів”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Ланецька І.С.
«____» _____ 2025 р.

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

м. Кропивницький

Центральноукраїнський національний технічний університет

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

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

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

Ланецькій Ірині Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки при використані хмарних сервісів*

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

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

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки при використані хмарних сервісів*

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

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

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

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

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
« 17 » січня 2025 р.

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

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Ланецька І.С.
(прізвище та ініціали)

АНОТАЦІЯ

Ланецька І.С. Програмне забезпечення системи кібербезпеки при використані хмарних сервісів. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки при використані хмарних сервісів.

Метою розробки є програмне забезпечення системи кібербезпеки при використані хмарних сервісів.

Результат роботи – програмна реалізація системи кібербезпеки при використані хмарних сервісів.

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

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

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

Програму розроблено в середовищі Visual C++.

Ключові слова: кібербезпека, хмарні сервіси

ABSTRACT

Lanetska I.S. Software for a cybersecurity system using cloud services. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a cybersecurity system using cloud services.

The purpose of the development is software for a cybersecurity system using cloud services.

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

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

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

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

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

Keywords: cybersecurity, cloud services

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЕН	–	електронний нотаріус
ЕЦП	–	електронний 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. У публічних хмарах для створення нових клієнтських облікових записів і хмарних серверів буде потрібно лише кілька хвилин часу й діюча кредитна картка. Досвід показує, що підприємствам не вдається запобігати ситуації, коли їхні співробітники, приміром, здобувають ресурси для цілей моделювання, включаючи їхню вартість у відрядні витрати або оплачуючи за допомогою корпоративних кредитних карт. На жаль, у результаті реальне число хмарних серверів, на яких зберігаються конфіденційні корпоративні дані, може значно перевершувати ту кількість, що перебуває під контролем відповідальних фахівців.

Пару років тому побоювання щодо захищеності хмарних обчислень уважалися однією з основних причин недостатньо динамічного розвитку ринку хмарних обчислень. В українських умовах об'єктивні складності із забезпеченням ІБ у хмарі були помножені на консерватизм вітчизняних ІТ-директорів. Основний контраргумент із їх боку полягав у тому, що використання будь-яких хмарних сервісів веде до розмивання периметра безпеки корпоративних ІТ. Інакше кажучи, навіть невеликий крок убік хмарних сервісів (наприклад, перенос системи електронної пошти в хмару) негативно впливає на інформаційну безпеку.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки при використані хмарних сервісів.

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

– Огляд існуючих систем при використані хмарних сервісів.

– Дослідження системи кібербезпеки при використані хмарних сервісів.

– Програмна реалізація системи кібербезпеки при використані хмарних сервісів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі при використані хмарних сервісів.

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

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

ІБ перестає бути внутрішнім завданням окремо взятої компанії й у числі іншого виявляється в залежності й від дій провайдеру, до якого, швидше за все, немає повної довіри. Втім, у цьому контексті доречно згадати системних інтеграторів, що чомусь не викликають в CIO почуття небезпеки. Тим часом ступінь відкритості всіх основних бізнес-процесів компанії для сторонніх очей при установці, скажемо, ERP-системи незрівнянно вище. Давайте спробуємо розібратися в ситуації.

У забезпеченні інформаційної безпеки хмарних послуг беруть участь три сторони: розроблювач, провайдер і споживач. Кожний з них вносить свій внесок у захист сервісу й даних клієнта. Основне навантаження при цьому лягає на провайдера хмарних послуг. Самі сервіси реалізуються на базі власного або орендованого (повністю або частково) центру обробки даних. На даному етапі заходу щодо забезпечення ІБ хмарного сервісу мало відрізняються від реалізованих хостинг-провайдерами, а тому практично цілком залежать від прийнятих у центрі обробки даних. Тому й вибір хмарного провайдеру багато в чому схожий на вибір хостера й прив'язаний до якості ЦОД.

Таким чином, при оцінці безпеки сервісу того або іншого хмарного провайдеру буде незайвим запитати, у якому ЦОД перебуває його встаткування. ЦОД повинен бути захищений від падінь каналу зв'язку, від перебоїв з електроживленням, від якихось імовірних для даної місцевості катастроф (наприклад, центри японського провайдеру Tsukaeru.net у Нагано побудовані з таким розрахунком, щоб не зупинятися навіть під час землетрусу середньої сили), а також від фізичного ушкодження встаткування в результаті аварій, необережних дій персоналу або диверсії зловмисників.

Необхідні міри фізичного захисту (кілька периметрів безпеки на території центра обробки даних, розмежування доступу персоналу залежно від наданих повноважень, поділ гермозон спеціальними клітками, щоб у секцію не проникнув сторонній системний адміністратор, і т.д.) добре відомі, тому зупинятися на них сенсу немає. Незважаючи на їхню очевидність, у Україні не так багато ЦОД, де ці міри реалізовані на належному рівні. З'ясувавши в хмарного провайдеру, у якому ЦОД розташовується його сервіс, можна зробити досить достовірний вивід про надійність хмарної послуги.

1.2 Область застосування

Товстий канал зв'язку для хмарного провайдеру – нагальна потреба: хмарні сервіси приймають і відправляють дані через Мережу, тому передбачити захист від мережних атак так само важливо, як забезпечити безпека фізичних границь. У ЦОД постачальників хмарних послуг для протидії атакам ефективно використовується досвід усе тих же хостерів. У цьому зв'язку варто згадати одну дуже характерну українську, та й не тільки, проблему в області мережної безпеки. Це атаки DDoS, які, судячи з регулярних і гучних історій з LiveJournal, стали надзвичайно популярні в зловмисників. Наскільки можна судити за доступною інформацією, атака тривалістю два-три дні коштує кілька тисяч доларів. Наслідки такої диверсії для хмарного провайдеру можуть виявитися жалюгідними – аж до повного руйнування в результаті масового обігу клієнтів за грошовою компенсацією за умовами SLA. Тому хмарний ЦОД потрібно убезпечити від атак DDoS. Робиться це апаратними засобами, зокрема з використанням Juniper NetScreen і Cisco Guard DDoS. З їхньою допомогою в потоці трафіку розпізнаються й блокуються запити, схожі на DDoS. Таке встаткування повинне бути встановлене й у ЦОД, і в провайдеру, якому належить канал зв'язку від центра обробки даних. Інакше користі буде мало. Захист від вторгнення через порти TCP/UDP здійснюється за допомогою брандмауерів – як програмних, так і

апаратних. Ця стандартна й добре відома технологія застосовується навіть у домашніх маршрутизаторах. Тут же варто згадати й IDS/IPS – поведінкові системи, які здатні виявляти аномалії трафіку, характерні для спроб вторгнення, і відтинати їх ще на далеких підступах. По-справжньому ефективний захист досягається тільки шляхом застосування комплексних мір. Відповідний арсенал коштує дуже дорого – покупка потрібного встаткування й ПЗ може зажадати сотень тисяч, а те й мільйонів доларів (без обліку витрат на інтеграцію). Із цієї причини публічні хмари безпечніше для малих і середніх (до 500 чоловік) компаній, ніж інфраструктура й Web-сервіси, розгорнуті й підтримувані силами внутрішнього департаменту IT. Невеликі підприємства просто не зможуть добрати способу на забезпечення гідного рівня ІБ. У результаті настільки важливе завдання буде вирішена як прийдеться – а свідомо гірше, ніж у хмарному центрі обробки даних.

У комерційних (або в публічних) хмарах полягає неявний внутрішній ризик – клієнти хмари становлять потенційну небезпеку друг для друга. До речі, саме тому послуги SaaS з публічної хмари є набагато більше безпечними для використання, ніж PaaS і IaaS. (Звичайно, за умови, що ПЗ як послуга не містить уразливостей, постачено базовим захистом від злому, архітектурно й на рівні коду підтримує ізоляцію облікових записів.) Справа в тому, що платформні й інфраструктурні сервіси дозволяють привносити в хмару практично будь-який програмний уміст, аж до додатків з достатком уразливостей і шкідливого коду. Чистий SaaS таких вільностей не припускає, оскільки робота йде тільки з користувальницькими даними. Таким чином, користувачам коштувало б більше остерігатися Amazon EC2 і йому подібних провайдерів (де через проломи в системі безпеки одного Web додатку зловмисники можуть порушити стабільність усього хмари й працюючих у ньому сервісів клієнтів), а не рішень на зразок SalesForce або, наприклад, Асофт CRM, де модифікація коду виключена, а всі операції зводяться тільки до обробки даних, що завантажуються користувачем, або (у самому демократичному випадку) до використання клієнтами сервісу API.

Вільності, припустимі в рамках IaaS і PaaS, множать ризики хмарних провайдерів і їхніх клієнтів. Щоб уникнути цього, на наш погляд, хмарної індустрії варто якомога швидше прийняти загальний стандарт на хмарні додатки. Такий стандарт повинен гармонізувати контент, що привноситься клієнтами в хмару, але при цьому не повинен ставити розроблювачів додатків у тверді рамки й змушувати застосовувати для створення ПЗ які-небудь конкретні інструменти.

Роботу в цьому напрямку Parallels початку ще на початку 2000-х. У результаті в 2004 році з'явився Application Packaging Standard (APS) і запропоновані специфікації, при використанні яких на розроблювачів хмарного ПЗ не накладало б ніяких обмежень, крім ряду нескладних операцій при здійсненні так званого впакування додатків для їхній наступного поширення через хмари хостерів і телеком-провайдерів, у яких програмне забезпечення для автоматизації підтримує APS.

На даний момент специфікації APS прийняті сотнями розроблювачів ПЗ в усьому світі. У каталозі представлено більше тисячі додатків. Загальна кількість завантажень APS-пакетів перевищило 1 млн. Процедура сертифікації передбачає перевірку готового пакета разом із включеним у нього додатком на якість коду й сховану функціональність. Звичайно, сертифікація не гарантує абсолютної захищеності хмарного ПЗ, але при такому підході хмарне APS-сумісне програмне забезпечення виявляється набагато безпечніше для провайдеру (а виходить, і для його клієнтів), оскільки при роботі в рамках стандарту об'єднані зусилля вендора й контриб’ютора APS.

Провайдери, що пропонують інфраструктуру або платформу в оренду із хмари, аж ніяк не випадково приділяють підвищену увагу ізоляції різних віртуальних середовищ. Це робиться для того, щоб катастрофа сервісу в одного клієнта ніяк не позначилося на працездатності Web-додатка в інших. Звичайно, найкращий спосіб – виділення окремого фізичного сервера для кожного клієнта. Але на практиці хмарні провайдери ніколи так не надходять, тим більше що в настільки радикальних мірах особливої необхідності немає. Сучасні підходи до

віртуалізації – контейнерної й гіпервізорної – забезпечують практично повну ізоляцію. Інша справа, що до ряду додатків не завжди можна застосувати віртуалізацію й ізоляцію. Мова йде про багатокористувальницькі (multi-tenant) сервіси, які використовуються головним чином у внутрішніх хмарах. Такі додатки являють собою, образно говорячи, один великий басейн, звідки користувачі черпають стільки води (читай – ресурсів), скільки їм потрібно в конкретний момент.

При цьому басейн не поділений нарівно між всіма споживачами. Яскравий приклад такого додатка – СУБД, що працює у великому кластері. Більші розміри й архітектурні особливості не дозволяють помістити її у віртуальну машину. Тому клієнти (вони ж – користувачі цієї СУБД) можуть розділятися тільки по ім'ю й паролю – саме в цей момент, при уведенні цієї інформації, клієнт хмарного сервісу зіштовхується з першою ланкою ланцюжка забезпечення ІБ, завдання якого – зберегти автентифікаційні дані.

Якщо сам додаток і критично важливі дані перебувають у хмарі, а інформація для авторизації зберігається на офісному або домашньому комп'ютері, не оснащеному антивірусом, то навіть самий потужний арсенал хмарного захисту буде неспроможний. Дані можуть бути украдені або загублені – практично так само, як банківська карта. Звичайно, банк ніякої відповідальності за інцидент не несе. Максимум, ніж він зможе допомогти, – заблокувати рахунок.

У ланцюзі хмарної ІБ самою слабкою ланкою є користувач. Ця теза підтверджується дослідженням, проведеним організацією Software Engineering Institute: 46% респондентів визнають втрату від дій інсайдерів набагато більше істотним, ніж зовнішні погрози.

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

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Кібербезпека або інформаційна безпека – це практика захисту комп'ютерних систем, мереж і даних від несанкціонованого доступу, атак, пошкодження або крадіжки. Його основна мета – забезпечити конфіденційність, цілісність і доступність ресурсів інформаційних технологій.

У епоху цифрових технологій, коли компанії, уряди та окремі люди сильно залежать від технологій, кібербезпека має вирішальне значення для захисту конфіденційної інформації та підтримки функціональності структур.

Хмарні обчислення стосуються надання обчислювальних послуг, включаючи зберігання, обчислювальну потужність і різноманітні програми, через Інтернет. Замість того, щоб покладатися на локальний сервер або персональний комп'ютер для вирішення цих завдань, хмарні обчислення дозволяють клієнтам отримувати доступ і використовувати ресурси, розміщені на віддалених серверах.

Поширені приклади служб хмарних обчислень включають Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) і різноманітні SaaS-додатки, як-от Google Workspace і Microsoft 365. Хмарні обчислення стали основоположною технологією для бізнесу, забезпечуючи економічно ефективні рішення, масштабованість і легкість доступу до широкого спектру обчислювальних ресурсів.

Важливість кібербезпеки в хмарних обчисленнях

Захист даних

Захист конфіденційної інформації: у хмарі дані зберігаються та обробляються віддалено, що робить надійні заходи кібербезпеки вирішальними для захисту конфіденційної інформації. Шифрування, контроль доступу та регулярні аудити є ключовими компонентами для запобігання несанкціонованому доступу та захисту цілісності даних.

Зменшення ризику взлому: кібербезпека в хмарі включає впровадження заходів для зменшення ризику взлому даних. Це включає в себе безперервний моніторинг, виявлення загроз і механізми реагування для оперативного вирішення та нейтралізації потенційних загроз безпеці.

Збереження довіри клієнтів

Гарантія конфіденційності та приватності: клієнти довіряють свої дані постачальникам хмарних послуг, очікуючи високого рівня конфіденційності та приватності. Практики кібербезпеки вселяють впевненість, гарантуючи, що дані клієнтів обробляються відповідально, знижуючи ризик розкриття даних або неправомірного використання.

Прозорі комунікації: демонстрація відданості кібербезпеці через прозорі комунікації створює та підтримує довіру клієнтів. Регулярне інформування клієнтів про заходи безпеки, інциденти та способи їх вирішення створює відчуття безпеки та партнерства.

Відповідність і юридичні зобов'язання

Відповідність нормативним стандартам. Хмарні обчислення часто передбачають обробку даних, які підпадають під дію різних нормативних актів, зокрема GDPR, HIPAA або галузевих стандартів відповідності. Дотримання цих правил є не лише юридичним обов’язком, але й важливим аспектом кібербезпеки в хмарі, щоб уникнути тюремних покарань і шкоди репутації.

Забезпечення відповідності постачальників послуг. Організації, які використовують хмарні служби, повинні переконатися, що їхні компанії з

хмарних послуг дотримуються галузевих і регіональних інструкцій. Це включає в себе оцінку протоколів безпеки постачальника та підтвердження їх відповідності вимогам законодавства.

Безперервність і стійкість бізнесу

Планування аварійного відновлення: кібербезпека в хмарі поширюється на планування аварійного відновлення. Хмарні рішення пропонують підприємствам можливість копіювання та резервного копіювання даних, забезпечуючи швидке відновлення в разі кібератак, стихійних лих або системних збоїв.

Зменшення ризиків простою: безперервна доступність є ключовою перевагою хмарних обчислень, але заходи кібербезпеки є життєво важливими для пом'якшення небезпек, пов'язаних із простоєм. Захист від DDoS-атак, уразливості системи та проактивний моніторинг сприяють безперебійній роботі бізнесу.

Виклики кібербезпеки в хмарних обчисленнях

Питання безпеки та конфіденційності даних

Витоки даних. Спільний характер хмарних середовищ викликає занепокоєння щодо несанкціонованого доступу та потенційного витоку даних. Організації повинні запровадити надійне шифрування, засоби контролю доступу та надійні механізми автентифікації, щоб захистити конфіденційну інформацію від злому.

Відповідність нормативним вимогам: дотримання правил захисту даних, таких як GDPR або HIPAA, стає складним у хмарі через транскордонний потік даних. Забезпечення відповідності вимагає ретельного розгляду постійності даних, обробки та контролю доступу.

Керування ідентифікацією та доступом

Неавторизований доступ: керування ідентифікацією користувачів і дозволами доступу в динамічних хмарних середовищах може бути складним. Неправильні конфігурації або слабкі засоби контролю доступу можуть призвести

до несанкціонованого доступу, що робить критично важливим для організацій впровадження надійних методів управління ідентифікацією та доступом (IAM), щоб зменшити ці ризики.

Вразливості спільної інфраструктури

Ризики, пов'язані з використанням кількох орендарів. Хмарні сервіси часто використовують модель із кількома клієнтами, коли кілька користувачів спільно використовують одну інфраструктуру. Уразливі місця в середовищі одного користувача потенційно можуть бути використані для порушення безпеки інших. Механізми ізоляції та постійний моніторинг є важливими для пом'якшення цих ризиків.

Кіберзагрози та атаки

Розширені стійкі загрози (APT): хмарні середовища вразливі до складних і постійних кіберзагроз. Зокрема, APT становлять значну проблему, оскільки вони можуть залишатися непоміченими протягом тривалого часу. Регулярне оновлення інформації про загрози, постійний моніторинг і проактивне реагування на інциденти є важливими для протидії таким загрозам.

Розподілена відмова в обслуговуванні (DDoS): Хмарні служби є привабливими цілями для DDoS-атак через їх масштабованість. Забезпечення надійних стратегій пом'якшення DDoS має вирішальне значення для запобігання збоям у роботі та простою.

Відповідність і юридичні проблеми

Відсутність видимості та контролю: організації можуть зіткнутися з проблемами підтримки видимості та контролю над своїми даними та заходами безпеки, покладаючись на постачальників хмарних послуг. Це може ускладнити спроби продемонструвати відповідність галузевим і нормативним стандартам.

Можливість перевірки. Аудит та оцінка стану безпеки хмарних середовищ може бути складним завданням, особливо коли ви маєте справу з кількома постачальниками. Встановлення чітких журналів аудиту та забезпечення прозорості постачальників є важливими для виконання вимог

відповідності.

Управління та управління безпекою

Управління конфігурацією хмари: неправильне налаштування параметрів хмари є поширеним джерелом уразливості безпеки. Ефективне управління безпекою та управління необхідне для створення та впровадження політик, проведення регулярних аудитів і забезпечення відповідності конфігурацій найкращим практикам безпеки.

Відсутність навичок безпеки: розвиток хмарної безпеки вимагає кваліфікованих фахівців, які добре знаються як на кібербезпеці, так і на хмарних технологіях. Дефіцит експертів у цій галузі створює проблему для організацій, які прагнуть підтримувати надійну безпеку.

Найкращі практики кібербезпеки в хмарних обчисленнях

Оскільки організації все частіше переходять на хмарні обчислення для підвищення масштабованості, гнучкості та ефективності, потреба в надійних заходах кібербезпеки в хмарі стала більш критичною, ніж будь-коли. Захист конфіденційних даних, забезпечення відповідності та подолання кіберзагроз є першочерговими. Ось кілька найкращих практик для підтримки кібербезпеки в середовищах хмарних обчислень:

1. Шифрування даних

Застосуйте надійні протоколи шифрування для даних як під час передачі, так і в стані спокою. Шифрування захищає інформацію від несанкціонованого доступу, гарантуючи, що навіть якщо дані перехоплені, вони залишаються нечитабельними без відповідних ключів дешифрування.

2. Керування ідентифікацією та доступом (IAM)

Застосовуйте суворі практики IAM для керування та контролю доступу користувачів. Реалізуйте принцип найменших привілеїв, надаючи користувачам лише дозволи, необхідні для їхніх конкретних ролей. Регулярно переглядайте та оновлюйте привілеї доступу, щоб мінімізувати ризик несанкціонованого доступу.

3. Багатофакторна автентифікація (MFA)

Покращте автентифікацію користувачів, запровадивши MFA. Це додає додатковий рівень безпеки, вимагаючи від користувачів підтверджувати свою особу кількома засобами, такими як паролі, біометричні дані або одноразові коди.

4. Регулярні аудити та моніторинг безпеки

Проводьте регулярні перевірки безпеки, щоб виявити вразливі місця та оцінити відповідність політикам кібербезпеки. Впроваджуйте постійний моніторинг, щоб виявляти будь-які підозрілі дії та оперативно реагувати на них, забезпечуючи загальну цілісність хмарного середовища.

5. Резервне копіювання даних і аварійне відновлення

Створіть надійний план резервного копіювання даних і аварійного відновлення. Регулярно створюйте резервні копії важливих даних і переконайтеся, що механізми відновлення працюють, щоб мінімізувати час простою та втрату даних у разі інциденту безпеки або збою системи.

6. Заходи безпеки хмарного постачальника

Зрозумійте та використовуйте функції безпеки, які надає постачальник хмарних послуг. Це може включати брандмауери, системи виявлення вторгнень і можливості журналювання. Будьте в курсі оновлень і нових функцій безпеки, які пропонує провайдер.

7. Регулярне навчання з безпеки

Навчіть співробітників передовим практикам кібербезпеки, характерним для хмарних середовищ. Переконайтеся, що вони обізнані про потенційні загрози, важливість надійного керування паролями та належне поводження з конфіденційною інформацією в хмарі.

8. Управління відповідністю

Будьте в курсі та дотримуйтеся галузевих вимог і правил відповідності. Користувачі хмарних послуг повинні переконатися, що їхній постачальник хмарних послуг відповідає відповідним стандартам, щоб уникнути юридичних і нормативних проблем.

9. План реагування на інциденти

Розробіть і регулярно оновлюйте план реагування на інциденти, у якому описано кроки, яких необхідно вжити у випадку інциденту безпеки. Цей план має включати стратегії зв'язку, ролі та обов'язки, а також процедури розслідування та пом'якшення порушень безпеки.

10. Регулярні оновлення програмного забезпечення

Підтримуйте все програмне забезпечення, включаючи операційні системи та програми, в актуальному стані за допомогою останніх патчів безпеки. Регулярно оновлюйте та виправляйте компоненти хмарної інфраструктури, щоб усунути відомі вразливості та підвищити загальну безпеку.

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 додатків 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, ніж звичайні виклики функцій. За рахунок цього значно спрощується розробка додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються

генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні додатки, а також додатки, 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 додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків 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 додатки 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-додатки 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-додатків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

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

2.3 Розгорнута постановка завдання

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

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Контрольоване використання ресурсів

Набагато безпечніше забезпечити співробітникам підприємства контрольований доступ до цих ресурсів – зрозуміло, тільки в дійсно необхідних межах. Це дозволить не тільки впорядкувати доступ, але й управляти ім. Багато брокерів зовнішніх і внутрішніх хмар пропонують широкі можливості адміністрування доступу для окремих осіб відповідно до користувальницької, рольової або мандантної моделі в рамках загального доступу. З одного боку, у цьому випадку надаються тільки ті права, які дійсно необхідні користувачам, а з іншого боку – підвищується безпека на підприємстві, оскільки права розподіляються централізовано й диференційовано.

Такі функції надають усе найбільш популярні хмарні брокери, такі як vCloud Director, Amazon/ IAM, OpenStack і CloudStack. У багатьох випадках можлива навіть комбінація локальних і зовнішніх ресурсів. Відмова від цих функцій рівносильний наявності одного загального пароля для всіх співробітників компанії, що працюють за загальним комп'ютером, – очевидно, не самий сприятливий сценарій. Тому створення й використання ролей, індивідуальних облікових записів або мандантів варто вважати обов'язковим.

Незалежно від того, під керуванням якої операційної системи функціонує хмарний сервер, доступ до нього необхідно захистити. Звичайно, можна покластися на надані брокером хмарних сервісів випадкові паролі, але звичайно користувачі ці паролі або де-небудь записують, або заміняють їх простими стандартними комбінаціями. Обидва варіанти не самі вдалі.

Доступ до хмарних серверів

Один з підходів полягає в поділі внутрішнього й зовнішнього підтвердження прав доступу (Credentials). Це можна реалізувати на базі власних серверів доступу/SSH/ RDP Proxy або за допомогою спеціалізованих комерційних рішень. При цьому співробітники вказують свій внутрішній пароль для реєстрації на сервері доступу, а після перевірки наданих їм прав одержують доступ до хмарного сервера. Таким чином, при звертанні до зовнішніх ресурсів вони зможуть використовувати свої власні (доменні) паролі без погрози для безпеки.

Цей варіант привабливий особливо в середовищах Windows, оскільки, з погляду користувачів, доменні паролі придатні для авторизації на хмарних серверах, причому зовнішні хмарні сервери не одержують доступу до елементів Active Directory і їх не потрібно включати в домен. Періодична зміна паролів теж значно полегшується (а в деяких випадках стає нарешті-те можливою), оскільки внутрішні й зовнішні процеси авторизації розділені.

Журнальні записи в хмарі

Як і будь-яка інша система, хмарні сервери теж створюють журнальні записи (Log Data). Однак фахівці часто забувають про те, що журнальні записи можуть служити важливим джерелом інформації при проведенні інженерно-технічних експертиз (особливо для запускаємих динамічно й хмарних серверів, що зупиняються,). При необхідності такої експертизи, приміром, у випадку хакерської атаки, ці дані (при їхній наявності) найчастіше виявляються єдиною зачіпкою, адже відповідні хмарні сервери вже можуть бути виведені з експлуатації.

Тому збереження важливих журнальних записів (вхід/вихід користувачів, цілісність системи, відновлення) за весь термін служби хмарного сервера є важливою й необхідною мірою. Тільки збір даних, їхнє об'єднання й вивід на зовнішні інструменти, наприклад на сервер Syslog, платформу безпеки з функцією перевірки журналів або на систему керування подіями інформаційної безпеки (Security Information and Event Management, SIEM), дозволять здійснити

їхній наступний аналіз. Атака на групу хмарних серверів може залишитися непоміченою, якщо атаки зловмисника стануть направлятися щораз на новий сервер, але після виводу журнальних записів за межі хмарних серверів і виконання їхньої систематизації таке поводження можна буде швидко виявити.

Керування безпекою

З погляду забезпечення безпеки хмари являють собою лише одну з різновидів інфраструктурних платформ, нехай навіть із високим ступенем автоматизації. Зрозуміло, і в хмарі не обійтися без процесів, які добре зарекомендували себе в традиційних фізичних системах. Такі основні функції, як міжмережний екран, IDS/IPS, віртуальне закриття уразливостей (Virtual Patching) і антивіруси, є обов'язковими елементами будь-якої концепції безпеки, будь те фізичні, віртуальні або хмарні системи. А от завдання, що виникають при керуванні цими функціями, у різних інфраструктурах відрізняються.

Необхідно, щоб всі сервери з локальних, внутрішніх або зовнішніх хмар були ідентифіковані й інтегровані в концепцію керування безпекою. При використанні хмарних сервісів цього можна домогтися за допомогою прямої інтеграції із брокером хмарних сервісів, що у будь-який момент може надати інформацію про наявність різних хмарних серверів. Зіставляти вручну параметри експлуатованих серверів з вимогами систем безпеки в динамічних середовищах неможливо, а при спробах такого порівняння виникають численні помилки.

Фільтрація даних

На жаль, у цьому випадку діє принцип самої слабкої ланки: якщо який-небудь хмарний сервер не інтегрований у систему керування безпекою, тобто не захищений у достатній мірі, то для зловмисників саме він є найбільш привабливою мішенню. А коли інформація про всі хмарні сервери надається автоматично, незалежно від їхнього розташування, за допомогою профілів безпеки можна оперативно вносити зміни для будь-якої кількості хмарних серверів.

Разом з тим, це дозволяє централізовано контролювати зміни, вироблені на хмарних серверах (приміром, у файлах або реєстрі). Щоб знизити трудомісткість відстеження таких подій, необхідно забезпечити автоматичну фільтрацію цих даних за допомогою спеціалізованого ПЗ – сортувати вручну всі події, наприклад при відновленні стандартної системи Linux або Windows, не вийде. Завдяки автоматичній фільтрації «позитивних» подій з'являється можливість докладніше вивчити що залишилися, виявити ймовірні несанкціоновані дії й відстежити їх.

Незалежно від того, зв'язаний ваш бізнес із чи Інтернетом ні, у вашій компанії однаково є інформаційна мережа.

Комп'ютерні мережі спрощують логістику й фінансові операції, сприяють продуктивній роботі співробітників, де б вони не перебували, і забезпечують збереження даних. Зростаючі темпи інтеграції фізичного й цифрового миру привели до появи середовища, повної нових можливостей для навчання, ведення спільної діяльності й підвищення її ефективності. У найближчому майбутньому майже всі пристрої, як і люди, будуть зв'язані один з одним електронними комунікаціями.

Все це веде до зростаючої популярності хмарних обчислень, віртуалізації й мобільності, поширенню інтелектуальних датчиків і інтересу до використання персональних пристроїв для рішення бізнес-завдань (BYOD).

Однак не всі потенційні цифрові можливості свідомо приводять до успіху. Кіберзлочинці, хакери й вороже настроєні стосовно інших країн держави прагнуть скористатися нестабільністю й пробілами в безпеці, що виникають при розвитку цих нових тенденцій. З ростом взаємозалежності між фізичним і електронним мирами вкрай важливого значення набувають збалансовані міри кібербезпеки, які дозволили б одержати максимальний ефект, використовувати переваги нових цифрових моделей і при цьому тримати під контролем їхні негативні сторони.

Цифровізація бізнесу – запорука успіху, але, на жаль, у більшості організацій відсутній яка-небудь кіберстратегія. Не розуміючи супутніх небезпек, яким вони себе піддають, багато організацій не мають стратегічне розуміння того, як можна звести до мінімуму погрози безпеки й скористатися можливостями, що відкриваються.

Щоб повністю скористатися достоїнствами збалансованої кіберстратегії, необхідно розробити два інтегрованих плани – реалізації цифрових переваг і забезпечення безпеки. Для їхнього успішного перетворення в життя компаніям потрібно використовувати переваги стратегічного керівництва, продемонструвати прагнення до мети й забезпечити постійний контроль при чіткому розумінні завдань і стимулів. Стратегія повинна виходити з того факту, що кіберфункції здобувають ключову роль у досягненні бізнесу-успіху.

Реалізація цифрових переваг

Першим кроком до розробки збалансованої кіберстратегії є створення плану реалізації переваг цифрового світу, для чого необхідно досліджувати, які можливості й вигоди відкривають перед організацією нові технології. Для цього буде потрібно вивчення й поглиблений аналіз як уже існуючих цифрових рішень, так і тих, які будуть розроблені в найближчому майбутньому.

У першу чергу варто вивчити наступне:

– технології, що забезпечують більше простий обмін інформацією, а також дають виграш в ефективності за рахунок масштабу, віртуалізації й міграції в хмару;

– інновації в області комунікацій і спільної роботи; можливість максимального використання соціальних технологій за допомогою краудсорсинга й залучення замовників у процес досліджень і розробок, розвитку продуктів і маркетингу.

Цифрова безпека

Крім того, щоб звести до мінімуму ризики, створювані інноваційними рішеннями, потрібно розробити план забезпечення безпеки. І насамперед варто

подбати про забезпечення стійкості за рахунок поінформованості про ситуацію, ефективного й діючого керування безпекою й операціями, а також про створення й підтримку динамічної системи безпеки й протидії.

Останнім каменем у даній стратегії є здатність до гнучкості. З розвитком цифрових можливостей повинна еволюціонувати й система інформаційної безпеки. Вибудовування твердих бар'єрів і використання статичних апаратно-залежних технологій дозволять лише йти в ногу зі змінами й інноваціями, у той час як хакери діють швидше, випереджаючи здатність бізнесу організувати захист.

Для успішного захисту рівень організації систем безпеки повинен бути вище за середнє, а умови для проникнення в інформаційні системи настільки складними, щоб витрати часу й зусиль, що направляються на злом, виявлялися для хакерів невигідними. Для компанії «краща практика» полягає у тому, щоб порівнювати себе з конкурентами й безупинно розвивати не тільки свою здатність до оборони, але й готовність до нападу.

Інтернет завжди був і буде чудовим джерелом і каталізатором інновацій. Для успішної конкуренції компаніям варто знаходити підходящі для себе кіберможливості.

У той же час оволодіння ними може піддати компанію додатковому ризику, адже безперервність бізнес-процесів починає більшою мірою залежати від надійності мережі, партнери одержують доступ до корпоративним даних, а електронна взаємодія й співробітництво перетворюються в найважливіші елементи взаємодії із замовниками. Навіть якщо організація не користується Інтернетом, ігнорувати ці ризики не можна.

Дуже важливо знайти баланс між ризиком і виграшем. Складові такого балансу специфічні для кожної організації й залежать від ступеня небезпеки, що вона вважає прийнятної для досягнення конкретних показників повернення інвестицій. Однак керівники підприємств повинні бути обачні, знати про можливі погрози й не упускати їх з виду, захоплюючись технологічними інноваціями, що

потенційно може внести негативні корективи у вже затверджені стратегії розвитку, а в остаточному підсумку – ушкодити здатності бізнесу планувати й реалізовувати поставлені завдання.

Захист даних

Надання надійно захищеного хмарного сервера для обробки даних – це лише перший крок при використанні хмарних сервісів.

На другому й третьому етапах потрібно перемістити дані на захищений хмарний сервер і подбати про те, щоб вони й у хмарі (тобто за межами сфери фізичного доступу клієнта) залишалися в безпеці.

Останнє можна забезпечити за допомогою повного шифрування віртуальних жорстких дисків: якщо буде потреба одержання доступу дані розшифровуються при зчитуванні, а потім знову зашифровуються при записі на диск. Завдяки цьому можна уникнути ситуацій, коли незашифрована інформація попадає в системи тривалого зберігання провайдерів послуг або зберігається у вигляді резервних копій.

Шифрування даних

Сервер KMS перевіряє ідентифікаційні дані (приміром, по хмарному сервері, IP-адресам, шаблонам, ЦОД або місцезнаходженню) і цілісність (брандмауер, установлені латки, наявність активного антивірусного сканера) хмарного сервера, що направив запит, – точно так само, як це робить людина при традиційному шифруванні жорстких дисків. У випадку успіху ключ надається або автоматично, або після підтвердження вручну уповноваженою особою. Після цього хмарний сервер одержує доступ до даних – доти, поки не виявиться порушені його цілісність або ідентичність.

Роздільне адміністрування ключів

Крім вищезгаданого варіанта шифрування жорстких дисків з роздільним керуванням ключами при наданні інфраструктури як сервіс (Infrastructure as a Service, IaaS), ця концепція зустрічається й при реалізації моделі надання платформи як послуги (Platform as a Service, PaaS). Інформація зашифровується

таким чином, що в бази даних, надавані провайдером хмарних сервісів, у вигляді відкритого тексту вона не попадає. У цьому випадку необхідні для роботи із зашифрованими даними ключі теж контролюються користувачем.

Обоє зазначених методи не слід плутати із шифруванням даних на стороні провайдеру, коли дані шифруються провайдером перед їхнім переміщенням на системи зберігання або при резервному копіюванні (такий підхід застосовується, приміром, у сервісі Amazon S3 Server-Side Encryption). Ця технологія дозволяє захистити інформацію від зловмисників, навіть якщо ті одержать фізичний доступ до жорстких дисків (украдені дані будуть непридатні для використання), але не вбереже від зловживання посадовими повноваженнями з боку співробітників провайдеру хмарних сервісів.

Захист залежно від хмарної моделі

Ринок хмарних сервісів росте швидкими темпами, тому й питання безпеки їхнього використання коштують дуже гостро. На території України робота з такими сервісами має свою специфіку в силу законодавчих вимог обов'язкового захисту інформації, наприклад персональних даних, лікарської таємниці й т.д. У першу чергу необхідно визначитися з підходами до рішення питань ІБ при різних реалізаціях хмарних сервісів.

Модель «ПЗ як сервіс» (Software as a Service, SaaS) має на увазі обробку й зберігання всіх даних на стороні провайдеру. У цьому випадку доводиться повністю довіряти прийнятим їм мірам і технологіям захисту інформації. Використання SaaS для обробки, скажемо, персональних даних – причому як у технічному плані, так і в організаційно-правовому – можливо тільки в довірених інфраструктурах. Тому єдиним застосуванням цієї моделі представляється реалізація приватних хмар, коли довірений провайдер є підрозділом або підвідомчою організацією вищого органа.

При цьому особлива увага варто приділити питанням доступу до сервісу SaaS. Для його захисту необхідні використовувати двофакторну автентифікацію користувачів за допомогою відчужуваних носіїв (USB-токенів або смарт-карт),

наприклад eToken, і передача даних у зашифрованому виді. Для цієї мети підходять HTTPS або VPN з підтримкою сертифікованої «української» криптографії, шифрування вибіркових даних на прикладному рівні за допомогою модуля браузера, що підключається, або ж рішення із застосуванням власних proxy-серверів, які захищають комунікацію із провайдером.

Модель «платформа як сервіс» (Platform as a Service, PaaS), як і SaaS, при обробці персональних даних застосовна тільки при роботі з довіреним провайдером. І на неї теж поширюються всі перераховані вище підходи (з деяким розширенням).

Кардинальну протилежність SaaS представляє модель «інфраструктура як сервіс» (Infrastructure as a Service, IaaS). У цьому випадку споживачеві надається готова інфраструктура. Це може бути як віртуальна машина, так і ціла мережа. При звертанні до послуг недовіреного провайдеру варто контролювати доступ на рівні гіпервізора, інакше всі спроби побудувати ефективний захист будуть те саме що побудова міцних воріт без забору. При цьому завдання забезпечення інформаційної безпеки зводиться до комплексного захисту віддаленого підрозділу із застосуванням традиційних методів, що зарекомендували себе, застосовуваних у фізичних системах.

У випадку недовіреного провайдеру при використанні будь-якої моделі актуальне зберігання даних у хмарі в зашифрованому виді. Особливу гостроту цьому питанню надає популярність загальнодоступних сервісів Data as a Service і Database as a Service, які не передбачають захищеної взаємодії між провайдером і споживачем, порушуючи вимоги регуляторів в області ІБ. Коли HTTPS або VPN із сертифікованою «українською» криптографією не використовується, єдино можливим рішенням представляється передача даних по відкритих каналах зв'язку в зашифрованому виді. І такі пропозиції є на ринку – наприклад, «Крипто БД» у режимі роботи з посередником, що шифрує (proxy). Відмінна риса цих рішень полягає у тому, що керування ключами здійснюється адміністратором ІБ у рамках локальної інфраструктури, а не в інфраструктурі провайдеру.

Зашифрована передача

Але що дасть зашифроване зберігання даних навіть у сполученні з надійним контейнером, якщо передача даних здійснюється в незахищеному виді й без перевірки цілісності? Для безпечної обробки даних обов'язковою умовою є їх шифруєма передача.

Цікавий варіант являє собою комбінація із зашифрованих даних і технологій захищеної передачі.

Тема безпеки в хмарі дуже велика. Особливу увагу варто звернути на контекст, у якому працюють хмарні сервери: адже в зовнішніх хмарах найближчий сусід може виявитися основним конкурентом, тому варто споконвічно виходити з того, що ви перебуваєте на «ворожій території», і забезпечити відповідні міри безпеки. На жаль, до обіцянок, які дають провайдери хмарних сервісів, варто ставитися з обережністю, адже правова відповідальність за схоронність даних і їхню втрату покладає на самих клієнтів.

Технічні міри

Крім загальних правових умов, існують міри технічного характеру, дотримання яких дозволить підвищити захищеність хмарних ЦОД. Насамперед всі дії, спрямовані на поліпшення їхнього захисту й керування ними, не повинні ставати перешкодою для роботи користувачів. Це зажадає, з одного боку, застосування вже перевірених інструментів і методів – частково в більше розширеному форматі (як у випадку шифрування даних і роздільного керування ключами). З іншого боку, обов'язковою умовою для підвищення рівня автоматизації є більше тісна інтеграція різних функцій безпеки із брокером хмарних сервісів. Облік цих двох аспектів дозволить повною мірою скористатися перевагами хмар і не допускати компромісів у сфері безпеки.

3.2 Розробка структурної схеми

У хмарі безпека – це не розкіш, а необхідність. Подумайте про хмарну безпеку як про тихого охоронця вашої цифрової екосистеми. Несанкціонований доступ, витік даних і тіньові IT – все це потенційні загрози, що ховаються у величезному просторі хмари. Але подібно до того, як досвідчений охоронець передбачає та нейтралізує загрози, не привертаючи уваги, хмарна безпека постійно захищає ваші активи, виявляючи вразливі місця та безперебійно запроваджуючи засоби контролю, щоб ви могли зосередитися на інноваціях.

У сучасному середовищі загроз, що швидко розвивається, хмарна безпека полягає не лише в блокуванні даних; йдеться про забезпечення стійкості, довіри та гнучкості під час масштабування вашої організації в хмарі.

Хмарна безпека стосується широкого набору стратегій і технологій, призначених для захисту даних, програм та інфраструктури, розміщених у хмарі. Оскільки все більше організацій покладаються на хмарні обчислення для зберігання та керування критично важливими бізнес-даними, забезпечення безпеки цих середовищ стає головним пріоритетом. Хмарна безпека включає різноманітні інструменти, політики та елементи керування, які захищають хмарні системи від несанкціонованого доступу, витоку даних і нових кіберзагроз.

Хмарні обчислення (широко відомі як «хмара») – це надання обчислювальних послуг на вимогу – таких як сервери, сховище, бази даних і програмне забезпечення – через Інтернет. Це дозволяє підприємствам швидко масштабуватися, зменшувати витрати на IT і підвищувати гнучкість. Однак зростання поширення хмарних служб створює нові виклики безпеці, такі як керування контролем доступу та збереження конфіденційності даних у складних багатохмарних або гібридних середовищах.

Основні цілі хмарної безпеки включають:

– Захист конфіденційності даних: забезпечення безпеки конфіденційної інформації під час передачі та зберігання

– Керування багатохмарною безпекою: вирішення унікальних проблем безпеки, пов’язаних із використанням кількох постачальників хмарних послуг (CSP)

– Контроль доступу: обмеження доступу до хмарних середовищ і забезпечення того, що лише авторизовані користувачі, пристрої та програми можуть взаємодіяти з хмарою

Впроваджуючи надійні заходи безпеки в хмарі, організації можуть впевнено використовувати переваги хмарних обчислень, мінімізуючи ризики та підтримуючи відповідність галузевим стандартам і нормам.

Хмарна безпека для різних моделей розгортання

Хмарні обчислення можна розгорнути декількома способами, кожен з яких має унікальні проблеми безпеки та найкращі практики. Розуміння цих моделей розгортання – загальнодоступних, приватних, гібридних і багатохмарних – має важливе значення для створення надійної стратегії безпеки хмари.

Публічна хмара

Загальнодоступна хмара належить і управляється сторонніми CSP, такими як Amazon Web Services (AWS), Microsoft Azure і Google Cloud. У цій моделі послуги та ресурси розподіляються між кількома організаціями через Інтернет. Загальнодоступні хмарні середовища популярні завдяки своїй масштабованості та економічній ефективності, але вони мають унікальні проблеми безпеки:

– Проблеми з безпекою: спільний характер загальнодоступної хмари може збільшити ризики, зокрема порушення даних і неправильні конфігурації безпеки. Оскільки хмарний провайдер керує значною частиною інфраструктури, організації повинні зосередитися на захисті своїх даних і програм у загальнодоступній хмарі.

– Рекомендації: використовуйте шифрування, керування ідентифікацією та доступом (IAM) і багатофакторну автентифікацію (MFA), щоб захистити конфіденційні дані та забезпечити доступ лише авторизованим користувачам.

Приватна хмара

Приватна хмара призначена для однієї організації, пропонуючи більший контроль над даними, безпекою та відповідністю. Він може бути розміщений локально або стороннім постачальником, але залишається ізольованим від інших користувачів. Ця модель ідеально підходить для організацій із суворими нормативними вимогами, наприклад, у сфері охорони здоров'я чи фінансовому секторі:

– Питання безпеки: хоча приватні хмари забезпечують підвищену безпеку та контроль, вони пов'язані з вищими витратами та потребують детального керування для забезпечення захисту від внутрішніх загроз.

– Найкращі методи: запроваджуйте надійні засоби контролю доступу, регулярні перевірки безпеки та технології запобігання втраті даних (DLP), щоб забезпечити цілісність даних і відповідність галузевим стандартам, таким як HIPAA або PCI DSS.

Гібридна хмара

Гібридна хмара поєднує в собі переваги загальнодоступних і приватних хмарних середовищ, дозволяючи організаціям масштабувати свої операції, зберігаючи при цьому безпеку для конфіденційних робочих навантажень. Наприклад, організація може запускати клієнтські програми в загальнодоступній хмарі, зберігаючи фінансові дані в приватній хмарі.

– Проблеми з безпекою: складність керування безпекою як у публічному, так і в приватному середовищах збільшує ризик уразливостей. Передача даних між цими середовищами також має бути захищеною.

– Найкращі методи: використовуйте надійне шифрування для переміщення даних між хмарами, інтегруйте моніторинг безпеки в обох середовищах і застосовуйте узгоджені політики керування доступом на всіх платформах.

Багатохмарність

Багатохмарна стратегія передбачає використання кількох публічних

хмарних служб від різних постачальників. Такий підхід надає організаціям гнучкість і знижує ризик прив'язки до постачальника. Однак керування безпекою на різних платформах створює додаткові проблеми:

– Проблеми безпеки: Послідовне керування політиками безпеки на кількох хмарних платформах може призвести до прогалин, якими можуть скористатися зловмисники.

– Найкращі методи: запровадьте уніфіковану платформу керування безпекою, яка інтегрується з усіма хмарними провайдерами та постійно відстежує загрози. Посередники безпеки доступу до хмари (CASB) та інструменти керування правами доступу до хмарної інфраструктури (CIEM) також корисні для підтримки узгоджених політик і видимості.

Вибравши правильну модель розгортання та застосувавши ці найкращі практики, організації можуть адаптувати свої стратегії хмарної безпеки відповідно до своїх конкретних потреб, незалежно від того, надають вони перевагу гнучкості, контролю чи відповідності нормативним вимогам.

Основні інструменти захисту хмари

Щоб захистити хмарне середовище, організації покладаються на ряд інструментів, призначених для захисту даних, керування доступом і реагування на загрози в реальному часі. Хмарна платформа захисту додатків (CNAPP) об'єднує багато з цих хмарних рішень безпеки в одну платформу. Він допомагає захистити хмарні додатки шляхом сканування на наявність вразливостей, моніторингу робочих навантажень у хмарі та захисту даних із коду в хмарі.

Завдяки інтеграції цих важливих інструментів хмарної безпеки організації можуть захистити свої хмарні середовища від широкого спектру загроз безпеці, забезпечуючи відповідність і зберігаючи контроль над конфіденційними даними.

Чому нульова довіра має вирішальне значення для безпеки хмари

Нульова довіра – це модель безпеки, яка передбачає, що жоден користувач або пристрій не є довіреним автоматично, незалежно від того, чи знаходяться

вони в мережі чи поза нею. У хмарних середовищах, де дані розподіляються між кількома платформами, ця структура є важливою для захисту конфіденційної інформації.

Ключові принципи нульової довіри:

– Безперервна перевірка: кожна спроба доступу – з боку користувача, пристрою чи програми – постійно перевіряється, щоб запобігти несанкціонованому доступу.

– Доступ з найменшими привілеями: користувачам і пристроям надається лише мінімально необхідний доступ, що зменшує ризики безпеки.

– Мікросегментація: Хмара розділена на менші сегменти, що обмежує бічний рух, якщо зловмисник отримує доступ.

У хмарних середовищах загрози можуть надходити з будь-якого місця. Zero Trust захищає як від внутрішніх загроз, так і від зовнішніх атак, забезпечуючи перевірку та контроль кожної дії. Ця модель також запобігає вільному переміщенню хакерів, якщо вони порушують одну частину системи.

Нульова довіра є ключем до захисту сучасної хмарної інфраструктури, забезпечуючи постійний захист даних, програм і користувачів у хмарі.

Розуміння моделі спільної відповідальності

У хмарній безпеці модель спільної відповідальності визначає, як відповідальність за безпеку розподіляється між CSP і клієнтом. Ця модель є важливою, оскільки обидві сторони відіграють певну роль у забезпеченні належного захисту даних і систем.

Як модель спільної відповідальності працює в хмарній безпеці

Постачальник хмарних технологій несе відповідальність за безпеку самої хмарної інфраструктури, включаючи апаратне забезпечення, програмне забезпечення та мережу, яка запускає хмарні служби. Це передбачає захист фізичних центрів обробки даних і основної хмарної інфраструктури від кібератак, забезпечення безвідмовної роботи та підтримку безпеки платформи.

З іншого боку, клієнт несе відповідальність за безпеку своїх даних,

програм і будь-яких конфігурацій у хмарі. Це включає такі завдання, як керування ідентифікацією та контролем доступу, належне налаштування параметрів безпеки та забезпечення відповідності галузевим стандартам.

Наприклад, у той час як хмарний провайдер гарантує безпеку базових систем, клієнт повинен переконатися, що конфіденційні дані зашифровані, дозволи доступу належним чином налаштовано, а вразливі місця в програмах усунено.

Розуміючи та впроваджуючи модель спільної відповідальності, організації можуть краще захистити свої дані та забезпечити більш безпечне хмарне середовище.

Хмарна безпека для регульованих галузей

Такі галузі, як охорона здоров'я, фінанси та роздрібна торгівля, стикаються із суворими правилами, які вимагають посиленої безпеки в хмарі для захисту конфіденційних даних і забезпечення відповідності.

Хмарна безпека для охорони здоров'я (відповідність HIPAA)

Організації охорони здоров'я повинні дотримуватися HIPAA, забезпечуючи захист захищеної медичної інформації (PHI). Це передбачає шифрування даних, використання надійної автентифікації та регулярний аудит хмарних середовищ. Приватні або гібридні хмарні моделі часто використовуються для збереження контролю над PHI та відповідності вимогам HIPAA.

Хмарна безпека для фінансів (відповідність PCI DSS)

Фінансові установи повинні відповідати стандартам PCI DSS для обробки платіжних даних, що вимагає шифрування, контролю доступу та моніторингу. Фінансові компанії часто використовують гібридні хмари, щоб збалансувати масштабованість із суворим захистом даних.

Хмарна безпека для роздрібної торгівлі (захист електронної комерції)

Роздрібні продавці повинні захищати платіжні дані клієнтів і дотримуватися PCI DSS, щоб запобігти порушенням під час транзакцій.

Загальнодоступні хмарні служби керують трафіком, а шифрування та CASB забезпечують безпеку даних клієнтів.

Створення структури управління безпекою в хмарі

Структура управління безпекою в хмарі гарантує, що політики безпеки, ролі та обов'язки чітко визначені та впроваджені в хмарних середовищах. Це надзвичайно важливо для збереження контролю над даними, зниження ризиків і забезпечення дотримання галузевих норм.

Що таке хмарне управління?

Управління хмарою передбачає створення структурованого набору політик і елементів керування для ефективного керування безпекою в хмарі. Він охоплює такі ключові сфери, як керування даними, управління ризиками та відповідність. Без надійної системи управління організації ризикують втратити видимість своєї хмарної інфраструктури, що призведе до неправильних налаштувань і прогалин у безпеці.

Ключові компоненти структури хмарного управління

Структура управління безпекою в хмарі встановлює політики для керування безпекою даних, контролю доступу, відповідності та моніторингу в хмарних середовищах. Він забезпечує захист конфіденційних даних за допомогою шифрування та належного керування доступом, використовуючи такі інструменти, як DSPM і CIEM, для забезпечення конфіденційності та доступу з найменшими привілеями.

Крім того, регулярні аудити необхідні для підтримки відповідності галузевим стандартам, таким як PCI DSS і ISO 27001. Безперервний моніторинг і плани реагування на інциденти допомагають виявляти та пом'якшувати загрози в режимі реального часу за допомогою інструментів на основі ШІ. Такий єдиний підхід має вирішальне значення, особливо в складних багатохмарних або гібридних середовищах.

Багато визнаних експертів в області інформаційної безпеки й віртуалізації вважають, що хмарні технології забезпечують більше високий рівень безпеки,

ніж при використанні традиційних методів. Це твердження підтверджує й той факт, що за даними компанії GigaOm Pro, що веде дослідження в області інформаційної безпеки, дотепер не було ні одного серйозного злому систем безпеки в хмарі.

Розроблене програмне забезпечення представляє із себе набір компонентів призначених для забезпечення системи безпеки як у вже існуючих, так і в створюваних хмарних сервісах.

Розроблене програмне забезпечення дозволяє забезпечити захист переданої мережею у хмарні сервіси інформації, строгу взаємну автентифікацію користувачів і серверів хмарних сервісів, 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 3.1 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 хмарного сервісу.

– 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.

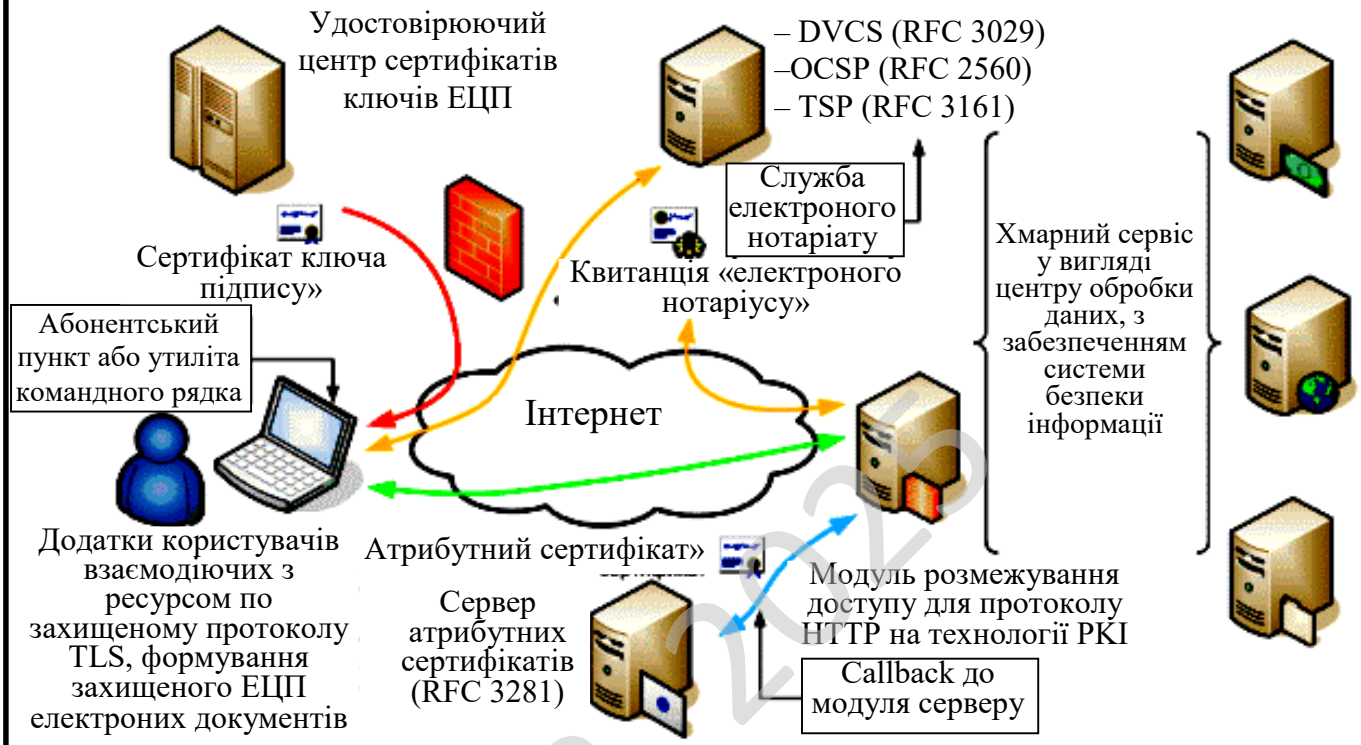

Рисунок 3.1 – Структурна схема

3.2 Розробка функціональної схеми

У якості функціональної схеми, так як вона є складовою частиною структурної схеми наведемо функціональну схему роботи протоколу SSL/TLS. Цей протоколи є складовою системи безпеки при використані хмарних сервісів.

На рисунку 3.2 зображена 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 хмарного сервісу 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 хмарного сервісу посилає повідомлення Certificate, що містить цифровий сертифікат сервера.

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

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

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

7. Клієнт і сервер хмарного сервісу, 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 хеш MD-5 і MAC, згенеровані на основі попередніх повідомлень handshake.

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

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

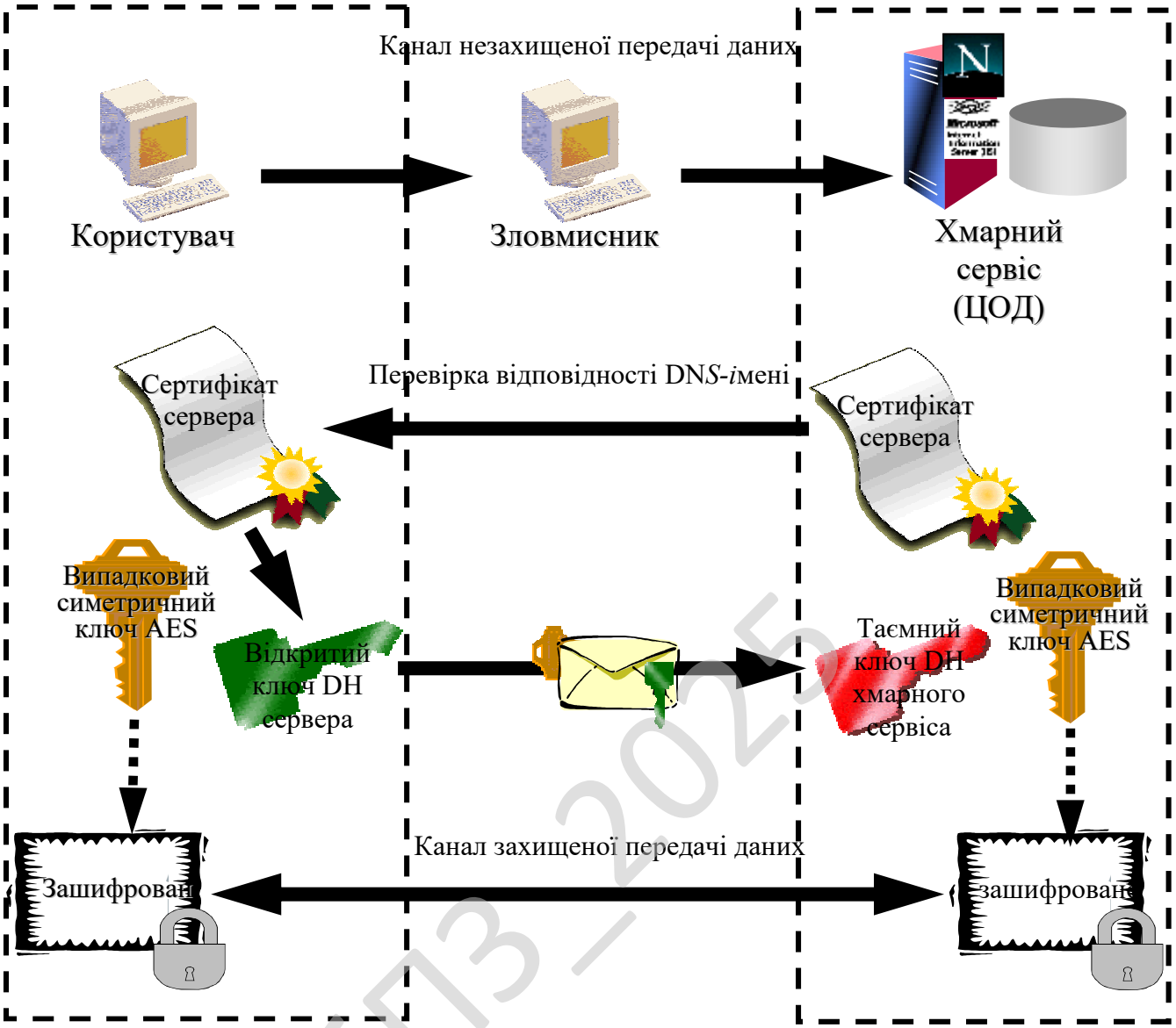

Рисунок 3.2 – Функціональна схема роботи протоколу SSL/TLS

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

Хеш-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 хеш-функції. Буфер може бути представлений як чотири 32-бітові регістри (*A*, *B*, *C*, *D*). Ці регістри ініціалізувалися наступними шістнадцятковими 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 $ABCD$:

$$A \quad B + \text{Clss}\,(A + f\,(B, C, D) + X\,[k] + T\,[i]),$$

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:

$$ff = (B \ \& \ C) \quad (\text{not } B \ \& \ D)$$

$$fg = (B \ \& \ D) \ \text{V} \ (C \ \& \ \text{not } D)$$

$$fh = B \quad C \quad D$$

$$fi = C \quad (B \ \& \ \text{not } D)$$

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.

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

3.4 Розробка діаграми процесів

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

– Інтерфейс ПЗ.

– Налаштування ПЗ.

– Моніторинг генерації ключів.

Рисунок 3.3 – Діаграма взаємодії процесів

– Транспортування ключа.

– Ініціалізація ключа.

– Резервне копіювання ключа.

– Використання ключа.

– Архівування ключа.

– Обмежене використання ключа.

– Знищення ключа.

– Реєстрація/видання сертифіката.

– Розповсюдження сертифіката.

– Зберігання сертифіката.

– Використання сертифіката.

– Зміна статусу сертифікату.

– Припинення строку дії.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

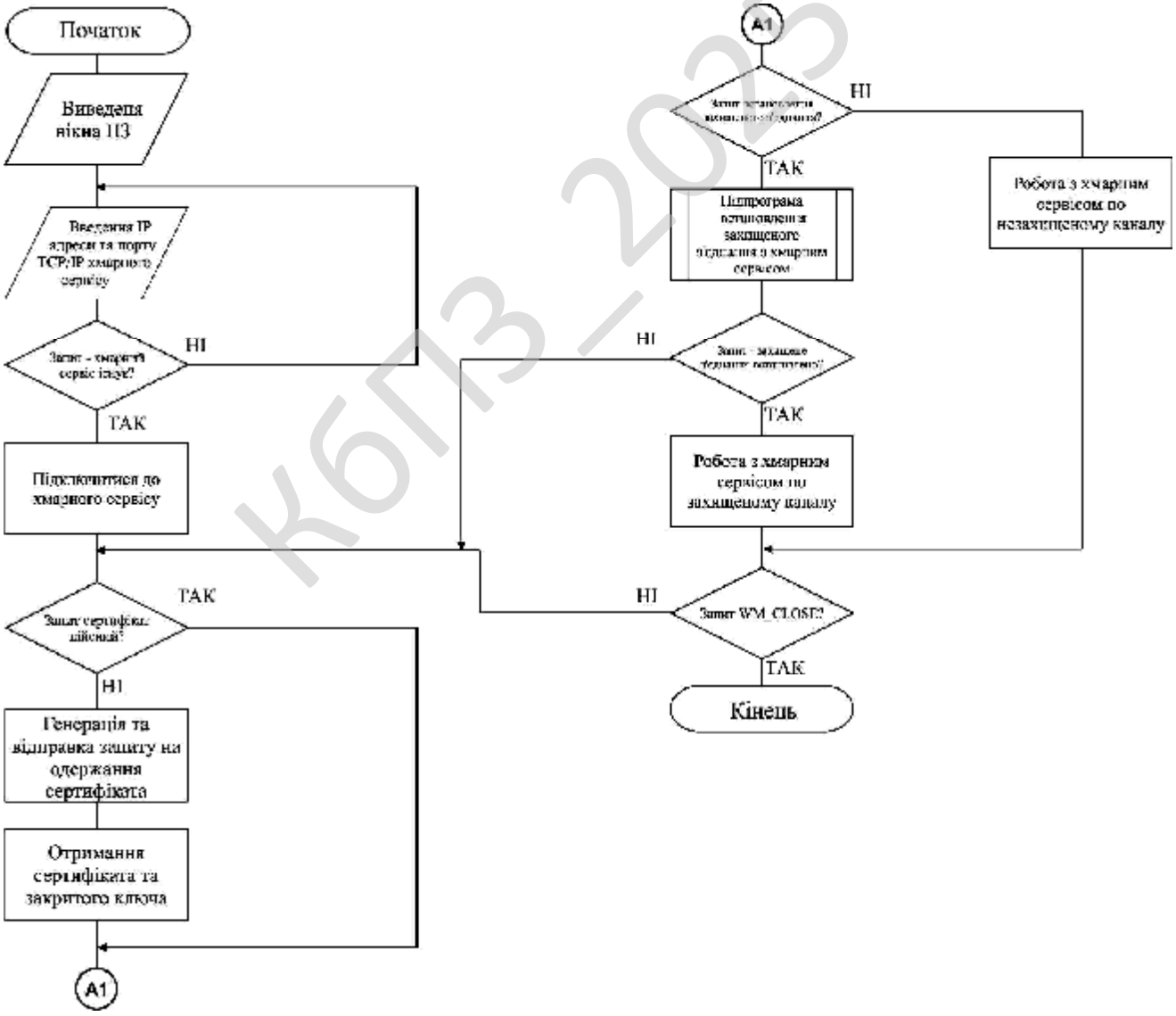

Рисунок 4.1 – Блок схема основної програми

З якої видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

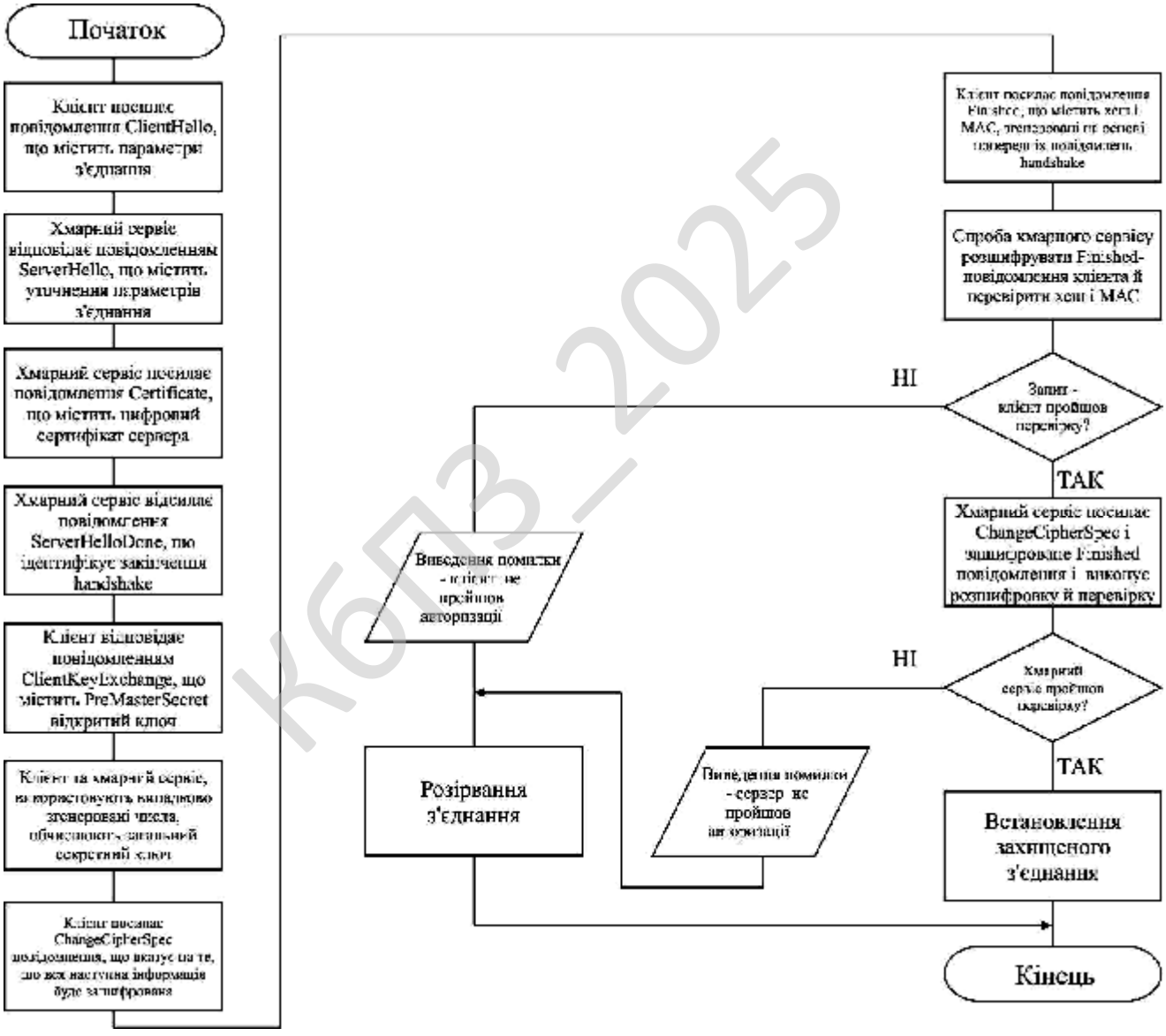

Рисунок 4.2 – Блок схема підпрограми

Опис алгоритмів функціонування системи

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

Це фундаментальна одиниця визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Опис обробки повідомлень

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 хеш і MAC, згенеровані на основі попередніх повідомлень handshake.

Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити хеш і MAC. Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване. У разі вдалої перевірки клієнта, сервер посилає ChangeCipherSpec і зашифроване Finished повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

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

Процедура перегляду рукопотискань клієнтом описується наступним чином:

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

        TimeStamp tsExpiry;
        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT |
                            ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR |
                            ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;
                            SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
        while(scRet == SEC_I_CONTINUE_NEEDED || scRet ==
SEC_I_INCOMPLETE_CREDENTIALS)
         {
            //
```

Встановлення вхідних буферів. Буфер 0 використовує шифрування в даних отриманих з серверу. Schannel читає усі дані або тільки признаки.

Відложені дані будуть накопичуватися у буфері 1 та надавати буферу тип SECBUFFER_EXTRA.

```
InBuffer.SetSecurityBufferToken(0, IoBuffer, ActualLen);
InBuffer.SetSecurityBufferEmpty(1);
OutBuffer.SetSecurityBufferToken(0, NULL, 0);
scRet = m_pSecurityFunc->InitializeSecurityContextA(m_phClientCreds,
                                    m_phContext,
                                    NULL,
                                    dwSSPIFlags,
                                    0,
                                    SECURITY_NATIVE_DREP,
                                &InBuffer,
                                0,
                                NULL,
                                &OutBuffer,
                                &dwSSPIOutFlags,
                                &tsExpiry);
```

Якщо InitializeSecurityContext працює правильно (або якщо помилка припустима), відправляємо зміст вихідного буфера на сервер.

```
if(scRet == SEC_E_OK || scRet ==
            SEC_I_CONTINUE_NEEDED || (FAILED(scRet)
   && (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR)))
{
    if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
    {
        bool bSent = DispatchSend(static_cast<char*>(OutBuffer[0].
                    pvBuffer), OutBuffer[0].cbBuffer, state);
        if(!bSent)
        {
throw new Common::Exceptions::SSLSendException(S"Відправлення на сервер не
відбулося.");
        }
    }
    OutBuffer.FreeBuffer(0);
}
```

Якщо InitializeSecurityContext повертає SEC_E_INCOMPLETE_MESSAGE, тоді ми читаємо наступні данні з сервера.

```
//
if(scRet == SEC_E_INCOMPLETE_MESSAGE)
```

```
            {
// Викликаємо повідомлення для збереження
// змісту буферу у разі помилки при подальшому вводі;
            }
            //
// Якщо InitializeSecurityContext повертає SEC_E_OK,
// тоді рукопотискання завершено успішно.
            //
            if(scRet == SEC_E_OK)
            {
                //
// Якщо "додатковий" буфер містить дані – то це закодована інформація для
// прикладного протоколу OSI. Це повинно бути збережено. Данні для прикладного
// рівня будуть декодовані з  DecryptMessage.
                //
                if(InBuffer[1].BufferType == SECBUFFER_EXTRA)
                {
                    pExtraData->pvBuffer = malloc(InBuffer[1].cbBuffer);
                    if(pExtraData->pvBuffer == NULL)
                    {
                        throw new OutOfMemoryException();
                    }
                    MoveMemory(pExtraData->pvBuffer,(BYTE*)IoBuffer + (ActualLen -
InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                    pExtraData->cbBuffer   = InBuffer[1].cbBuffer;
                    pExtraData->BufferType = SECBUFFER_TOKEN;
                }
                else
                {
                    pExtraData->pvBuffer   = NULL;
                    pExtraData->cbBuffer   = 0;
                    pExtraData->BufferType = SECBUFFER_EMPTY;
                }
// Для виходу зберігається у БД
                m_bInHandShake = false;
                if(DoServerCertVerify != NULL)
                {
                    IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemUni(m_ServerIP);
                    Common::Misc::CeriticateInfo ServCertInfo;
                    VerifyCertificate(true, m_pSecurityFunc, m_phContext,
static_cast<wchar_t*>(ptrServerName.ToPointer()), 0, &ServCertInfo);
```

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

```
            {
                ActualLen = 0;
                break;
            }
        }
        return true;
    }
```

Опис ініціалізації рукопотискання

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;
        }
    }
```

4.2 Захист розробленого програмного забезпечення

Tiny Encryption Algorithm (TEA) [1] – блочний алгоритм шифрування типу «Мережі Фейстеля». Алгоритм був розроблений на факультеті комп'ютерних наук Кембриджського університету Девідом Вілером (David Wheeler) і Роджером Нідгемом (Roger Needham) та вперше представлений в 1994 році [2] на симпозіумі зі швидкими алгоритмами шифрування в Льовені (Бельгія).

Шифр не патентований, широко використовується в ряді криптографічних додатків і широкому спектрі апаратного забезпечення, завдяки вкрай низькими вимогами до пам'яті й простоті реалізації. Алгоритм має як програмну реалізацію на різних мовах програмування, так і апаратну реалізацію на інтегральних схемах типу FPGA.

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму TEA, який заснований на бітових операцій з 64-бітним блоком, має 128-бітний ключ шифрування. Стандартна кількість раундів мережі Фейстеля біля 64 (32 циклу), однак, для досягнення найкращої продуктивності або шифрування, число циклів можна варіювати від 8 (16 раундів) до 64 (128 раундів). Мережа Фейстеля несиметрична через використання в якості операції накладення додавання за модулем 2^{32}.

Перевагами шифру є його простота в реалізації, невеликий розмір коду й досить висока швидкість виконання, а також можливість оптимізації виконання на стандартних 32-бітних процесорах, так як в якості основних операцій використовуються операції виключна «АБО» (XOR), побітового зсуву й додавання за модулем 2^{32}. Оскільки алгоритм не використовує таблиць підстановки і раундова функція досить проста, алгоритму потрібно не менше 16 циклів (32 раундів) для досягнення ефективної дифузії, хоча повна дифузія досягається вже через 6 циклів (12 раундів).

Алгоритм має відмінну стійкість до лінійного криптоаналізу і досить гарну до диференціального криптоаналізу. Головним недоліком цього алгоритму

шифрування є його вразливість до атак «на пов'язаних ключах» (англ. Related-key attack). Через простий розклад ключів кожен ключ має 3 еквівалентних ключа. Це означає, що ефективна довжина ключа складає всього 126 біт [3] [4], тому даний алгоритм не слід використовувати в якості геш-функції.

Опис алгоритму

Вихідний текст розбивається на блоки по 64 біта кожен. 128-бітний ключ K ділиться на чотири 32-бітних підключа K [0], K [1], K [2] і K [3]. На цьому підготовчий процес закінчується, після чого кожен 64-бітний блок шифрується протягом 32 циклів (64 раундів) за нижченаведеним алгоритмом. [5]

Припустимо, що на вхід n-го раунду ($1 \leq n \leq 64$) надходять права й ліва частини (L_n, R_n), тоді на виході n-го раунду будуть ліва й права частини (L_{n+1}, R_{n+1}), які обчислюються за такими правилами:

$$L_{n+1} = R_n.$$

Якщо $n = 2 * i - 1$ для $1 \leq i \leq 32$ (непарні раунди), то:

$$R_{n+1} = L_n (\{ [R_n\ 4] K [0] \} \{ R_n\ i * \delta \} \{ [R_n\ 5] K [1] \}).$$

Якщо $n = 2 * i$ для $1 \leq i \leq 32$ (парні раунди), то:

$$R_{n+1} = L_n (\{ [R_n\ 4] K [2] \} \{ R_n\ i * \delta \} \{ [R_n\ 5] K [3] \}).$$

Де

X Y – операція додавання чисел X і Y за модулем 232.

X Y – побітове виключне АБО» (XOR) чисел X і Y, яке в мові програмування Сі позначається як X ^ Y

X Y і X Y – операції побітового зсуву числа X на Y біт вліво й вправо відповідно.

Константа δ була виведена з Золотого перерізу:

$$\delta = (-1) * 2^{31} = 2654435769 = 9E3779B9_h.$$

У кожному раунді константа множиться на номер циклу i. Це було зроблено для запобігання простих атак, заснованих на симетрії раундів.

Також очевидно, що в алгоритмі шифрування TEA немає як такого алгоритму розкладу ключів. Замість цього в непарних раундах використовуються підключі K [0] та [1], у парних – K [2] і [3].

Так як це блочний шифроалгоритм, де довжина блоку 64-біт, а довжина даних може бути не кратна 64-біт, значення всіх байтів, які доповнюють блок до кратності в 64-біт, встановлюється в 0x01.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні розділи: "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. Якщо користувач перший раз використовує програму та не має сертифікату, або строк дії його сертифіката закінчився, то повинен згенерувати запит на одержання сертифікату. Після того як користувач вказав IP-адресу сервера та порт передачі даних, вибрав алгоритм захисту інформації та отримав сертификат та закритий ключ, він може здійснити з'єднання з сервером і відкрити Інтернет-браузер у захищеному режимі.

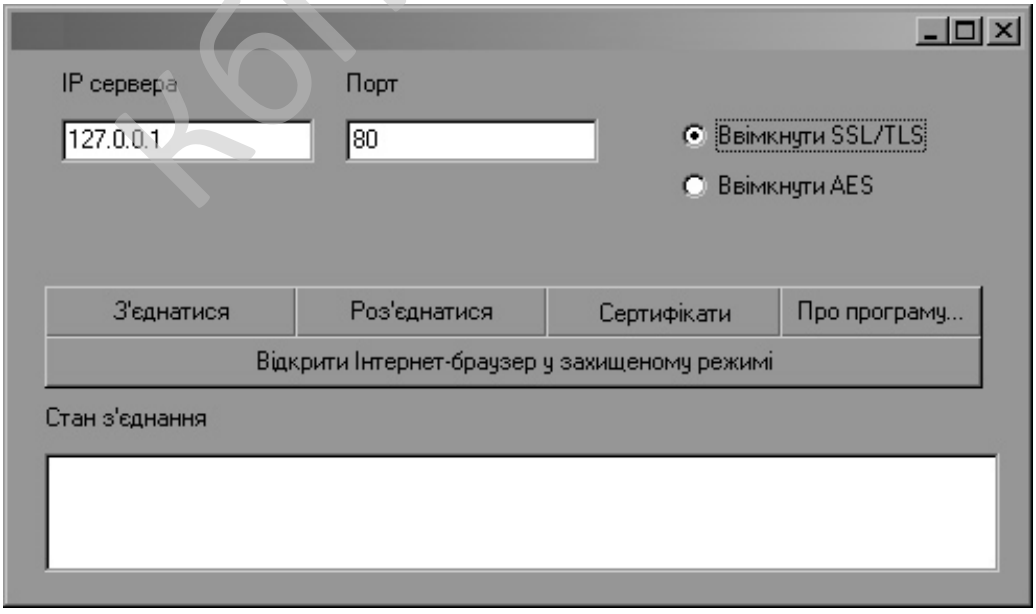

Рисунок 5.1 – Головне вікно ПЗ

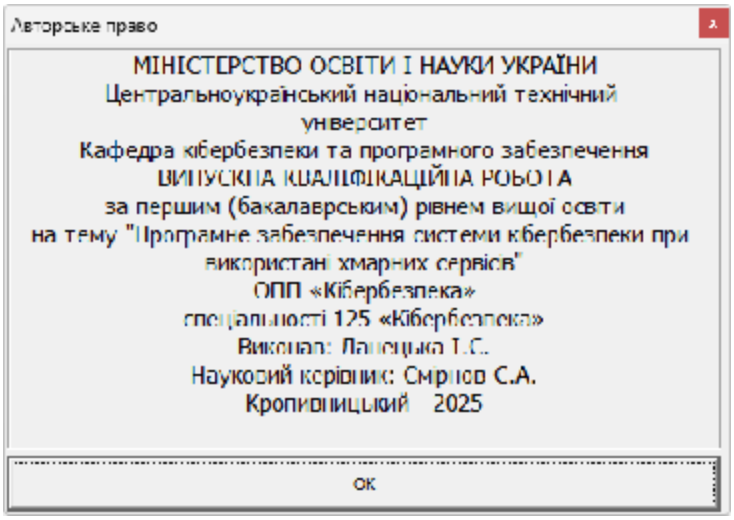

Рисунок 5.2 – Авторське право

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки при використані хмарних сервісів.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем при використані хмарних сервісів.

– Досліджена система при використані хмарних сервісів.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки при використані хмарних сервісів.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання при використані хмарних сервісів.

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

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

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки при використані хмарних сервісів. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної

техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм TEA.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

2. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

3. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

4. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

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

6. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

7. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

8. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

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

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

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

12. 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.

13. 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.

14. 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.

15. 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.

16. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties».

International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

17. 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.

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

19. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

20. 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.

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

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

23. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising

Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

24. 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.

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

26. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

27. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

28. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

29. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

30. 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.

31. 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.

32. 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.

33. 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

34. 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.

35. 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.

36. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

37. 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.

38. 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.

39. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

40. 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).

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

42. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

43. Смірнова Т.В., Солових Є.К., Смірнов О.А., 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.

44. 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.

45. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

46. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

47. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

48. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

49. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.

50. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. - Випуск 2 (46) - Х.: ХУПС - 2016. - С. 146-149.

51. Смірнов О.А., Кавун С.В., Доренський О.П., Вялкова В.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 151 с.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки при використані хмарних сервісів.

2 Підстава для розробки

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

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки при використані хмарних сервісів.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки при використані хмарних сервісів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 67 аркушів.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

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

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

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

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

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки при використані хмарних сервісів

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2025 року

Основна програма

CloudSSL.vcproj – Файл конфігурації проекту

```
<?xml version="1.0" encoding = "Windows-1252"?>
<VisualStudioProject
      ProjectType="Visual C++"
      Version="7.00"
      Name="CloudSSL"
      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)/CloudSSL.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)/CloudSSL.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="CloudSSL.cpp">
            </File>
            <File
                  RelativePath="CloudSSLServer.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="CloudSSL.h">
            </File>
            <File
                  RelativePath="CloudSSLServer.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 CloudSSL.cpp основної програми

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

    void CloudSSLConnection::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::CloudSSLException*)
        {
            Dispose();
            throw;
        }
    }

    void CloudSSLConnection::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::CloudSSLException(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::CloudSSLSendException(S"Відправлення помилки Сервера.");
            }
        }
    }
    bool CloudSSLConnection::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::CloudSSLSendException(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::CloudSSLException(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 CloudSSLConnection::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::CloudSSLException(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::CloudSSLException(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 CloudSSLConnection::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::CloudSSLException(S"Максимальни
CloudSSL розмір помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    bool CloudSSLConnection::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::CloudSSLException(S"Помлка з’єднання.
Помилка: ", Status);
        }
        //
        // Будуємо повідомлення про закриття CloudSSL
        //
        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::CloudSSLException("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::CloudSSLSendException(S"Відправлення на сервер не
відбулося.");
            }

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

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

    void CloudSSLConnection::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::CloudSSLException("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::CloudSSLException(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::CloudSSLSendException(S"Відправлення
помилкове. Помилка: ", scRet);
        }

        free(pbIoBuffer);
    }

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

    void CloudSSLConnection::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::CloudSSLException(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::CloudSSLException(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::CloudSSLException(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 CloudSSLConnection::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 CloudSSL.h – бібліотека для файлу CloudSSL.cpp

```
// CloudSSL.h
/*************************************************************************

#pragma once
#include "sslcommon.h"

namespace CloudSSL
{
    public __value struct ServerCeriticateInfo;
    namespace Client
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WriteCloudSSL(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 CloudSSLConnection;
        //Сервер потребує узгодження, новий сертифікат завантаження,
CloudSSLConn->LoadNewClientCredentials
        public __delegate void NewCertificate(CloudSSLConnection* CloudSSLConn);

        __sealed public __gc class CloudSSLConnection : public IDisposable
          {
        public:

            CloudSSLConnection();
            ~CloudSSLConnection()
            {
                Dispose(false);
            }
        public:
            //ініціалізація з'єднання, береться ip-адреса сервера і клієнтський
хеш сертифікату,
            //данні які потрібно відіслати були  повернені з WriteCloudSSL
            void InitiateHandShake(String* ipAddress, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state);
            //шифруємо дані, шифруємі дані повертаються в WriteCloudSSL
            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::CloudSSLException(S"Взятий
максимальний розмір токену CloudSSL помилковий. Помилка: ", scRet);
                return psecInfo->cbMaxToken;
            }
        public:
            //Повертаємо зашифровані дані
            WriteCloudSSL*         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, CloudSSL::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_CloudSSL,
                            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 =
CloudSSL::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 CloudSSL
{
namespace Common
{
    namespace Exceptions
    {
    [Serializable]
      public __gc class CloudSSLException : public ApplicationException
    {
    public:
        CloudSSLException(){}
        CloudSSLException(String* message):ApplicationException(message){}
        CloudSSLException(String* message, Exception* innerException)
            :ApplicationException(message,innerException){}
        CloudSSLException(String* message, UInt32 error)
            :ApplicationException(String::Concat(message,
Convert::ToString(error))){}

    };
    [Serializable]
    __sealed public __gc class CloudSSLServerFailedToFindExistingClientID :
public CloudSSLException
    {
    public:
        CloudSSLServerFailedToFindExistingClientID(){}
        CloudSSLServerFailedToFindExistingClientID(String*
message):CloudSSLException(message){}
        CloudSSLServerFailedToFindExistingClientID(String* message, Exception*
innerException)
            :CloudSSLException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class CloudSSLServerDisconnectedException : public
CloudSSLException
    {
    public:
        CloudSSLServerDisconnectedException(){}
        CloudSSLServerDisconnectedException(String*
message):CloudSSLException(message){}
        CloudSSLServerDisconnectedException(String* message, Exception*
innerException)
            :CloudSSLException(message,innerException){}
    };
    [Serializable]
    __sealed public __gc class CloudSSLReadException : public CloudSSLException
    {
    public:
```

```
        CloudSSLReadException(){}
        CloudSSLReadException(String* message):CloudSSLException(message){}
        CloudSSLReadException(String* message, Exception* innerException)
            :CloudSSLException(message,innerException){}
    };

    [Serializable]
    __sealed public __gc class CloudSSLSendException : public CloudSSLException
    {
    public:
        CloudSSLSendException(){}
        CloudSSLSendException(String* message):CloudSSLException(message){}
        CloudSSLSendException(String* message, Exception* innerException)
            :CloudSSLException(message,innerException){}
        CloudSSLSendException(String* message, UInt32 error)
            :CloudSSLException(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_CloudSSL3 = SP_PROT_CloudSSL3,
        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 CloudSSL
#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,
CloudSSL::Common::Misc::CeriticateInfo* CertInfo);
```

Файл CloudSSLServer.cpp – Серверна частина

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

namespace CloudSSL
{
namespace Server
{
    CloudSSLServer::CloudSSLServer(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::CloudSSLException(S"Failed to load
security dll.");
        m_pSecurityFunc = pSecurityFunc;
    }
    bool CloudSSLServer::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 дя 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::CloudSSLSendException(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::CloudSSLException(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::CloudSSLException(S"Неправильна умова. Помилка: ", scRet);
            }
        }
        return true;
    }

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

    void CloudSSLServer::DecryptData(Byte data[], Int32 ActualLen, Guid
ClientID, Object* state)
    {
        CLIENTDATA* pClientData=NULL;
        try
        {
                GetClientIDAssoc(ClientID, pClientData);
        }
        catch(Common::Exceptions::CloudSSLServerFailedToFindExistingClientID*)
        {
            //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::CloudSSLException("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::CloudSSLException("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 CloudSSLServer::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 CloudSSLServer::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 CloudSSLServer::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::CloudSSLException(S"Взятий
максимальний розмір блоку CloudSSL помилковий. Помилка: ", scRet);
        }
        return Sizes.cbMaximumMessage;
    }
    void CloudSSLServer::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::CloudSSLException(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::CloudSSLException(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::CloudSSLException(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 CloudSSLServer::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::CloudSSLServerFailedToFindExistingClientID("CloudSSL сервер
не знайшов id клієнта");
        }
            LeaveCriticalSection(m_pCS);
    }
    void CloudSSLServer::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 CloudSSLServer::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::CloudSSLException(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::CloudSSLSendException(S"Відправлення на сервер не
відбулося.");
            }
       }
    }

    void CloudSSLServer::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 CloudSSLServer.h – бібліотека для файлу CloudSSLServer.cpp

```
#pragma once

#include "sslcommon.h"

namespace CloudSSL
{
    namespace Server
    {
        //Запис данних, які надаються ззовні
        public __delegate bool WriteCloudSSL(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 CloudSSLServer : public IDisposable
        {
        public:
            CloudSSLServer();
            ~CloudSSLServer(){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 WriteCloudSSL
            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::CloudSSLException(S"Взятий
максимальний розмір точену CloudSSL 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
            WriteCloudSSL* 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;
        };
    }
}
```