Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Клієнт-серверна система безпечного ліцензування

Студента 2 курсу, 2мз групи,
спеціальності 121, Інженерія
програмного забезпечення.
Освітньої програми «Інженерія
програмного забезпечення» _____ Ігнатова Микити
Сергійовича
підпис студента

Науковий керівник
кандидат економічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки _____ Палагута Катерина
Олексіївна
підпис керівника

Гарант освітньої програми
кандидат педагогічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки _____ Котенко Наталія
Олексіївна
підпис керівника

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Ігнатову Микиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Клієнт-серверна система безпечного ліцензування»

Затверджена наказом ректора від «09» грудня 2022 р. № 3339

2. Строк здачі студентом закінченої роботи 27 листопада 2023

3. Цільова установка та вихідні дані до роботи

Мета роботи розробка власної клієнт-серверної системи для безпечного ліцензування ПЗ.

Об'єкт дослідження система безпечного ліцензування ПЗ.

Предмет дослідження різні методи та технології ліцензування, які застосовуються для забезпечення контролю та безпеки використання ПЗ.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ЗАГАЛЬНА ЧАСТИНА

1.1. Аналіз попередніх досліджень

1.2. Огляд технічних методів захисту та ліцензування ПЗ

1.3. Огляд існуючого програмного забезпечення

1.4. Висновки до розділу 1

РОЗДІЛ 2 ВИБІР СПОСОБІВ ТА ЗАСОБІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

2.1. Вибір способу рішення задачі

2.2. Обґрунтуванню вибору способів рішення задачі

2.3. Обґрунтуванню вибору засобів розробки

2.4. Опис концепції системи

2.5. Висновки до розділу 2

РОЗДІЛ 3 РЕАЛІЗАЦІЯ

3.1. Склад інформаційного забезпечення

3.2. Структура інформаційного забезпечення

3.3. Вибір технологій доступу до баз даних

3.3. Методи контролю інформації

3.3. Опис функцій системи

3.3. Опис функціональних блоків системи

3.3. Опис клієнтського додатку

3.4. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ТЕСТУВАННЯ ДОДАТКА

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. ЗАГАЛЬНА ЧАСТИНА*	*10.04.2023*	*08.04.2023*
6.	*Розділ 2. ВИБІР СПОСОБІВ ТА ЗАСОБІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ*	*24.05.2023*	*18.05.2023*
7.	*Розділ 3. РЕАЛІЗАЦІЯ*	*06.09.2023*	*05.09.2023*
8.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
9.	*Написання наукової статті*	*17.05.2023*	*18.04.2023*
10.	*Висновки та пропозиції*	*01.11.2023*	*29.10.2023*
11.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
12.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
13.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
14.	*Здача зброшурованої випускної кваліфікаційної роботи*	*27.11.2023*	*27.11.2023*
15.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*29.11.2023*	*29.11.2023*
16.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*.12.2023-*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Палагута К.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Ігнатов М.С.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Ігнатова М.С._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Котенко Н.О._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці власного рішення для безпечного ліцензування програмного забезпечення на базі клієнт-серверної архітектури. Випускна кваліфікаційна робота на тему «Клієнт-серверна система безпечного ліцензування» містить 45 сторінок, 7 рисунків. Перелік використаних джерел налічує 17 найменувань.

В роботі було зроблено огляд попередніх досліджень та програмного забезпечення в сфері захисту та ліцензування ПЗ, розглянуто та зроблено порівняльну характеристику наявних методів захисту та ліцензування програмного забезпечення.

На основі цього аналізу було обрано оптимальну стратегію ліцензування відібравши та модернізувавши наявні методи таким чином, що недоліки одного методу могли б бути компенсовані використанням іншого методу.

На основі цієї стратегії було розроблено клієнт-серверну систему з наступними підходами:

- ліцензія зберігається в ліцензійному файлі
- ліцензійний файл зашифрований симетричним ключем
- ліцензійний файл підписаний приватним ключем
- перевірка підпису ліцензійного файлу здіснюється за допомогою публічного ключа що отримується з сертифікату
- ліцензійний файли (зашифрований файл ліцензії та підпис) поширюються через зашифрований канал мережею Інтернет.
- в якості ідентифікатора користувача використовуються дані про апаратне забезпечення

Засоби реалізації було обрано таким чином щоб забезпечити надійне та ефективне функціонування системи, що має здатність розгортання на багатьох сучасних платформах (Windows, Linux, macOS) та є загальнодоступним (відкрите та безоплатне програмне забезпечення).

Ключові слова: безпека та ліцензування програмного забезпечення, клієнт-серверна система, ліцензійний файл, шифрування, цифровий підпис.

ABSTRACT

According to the purpose of the study, the work is devoted to the development of a proprietary solution for secure software licensing based on client-server architecture. Graduation qualification work on the topic "Client-server system of secure licensing" contains 45 pages, 7 drawings. The list of used sources includes 17 items.

The work reviewed previous research and software in the field of software protection and licensing, considered and made a comparative description of the existing methods of software protection and licensing.

Based on this analysis, the optimal licensing strategy was chosen by selecting and modernizing the existing methods in such a way that the shortcomings of one method could be compensated by the use of another method.

Based on this strategy, a client-server system was developed with the following approaches:

- the license is stored in the license file
- the license file is encrypted with a symmetric key
- the license file is signed with a private key
- verification of the signature of the license file is performed using the public key obtained from the certificate
- license files (encrypted license file and signature) are distributed through an encrypted channel over the Internet.
- hardware data is used as a user ID

The means of implementation were chosen in such a way as to ensure reliable and efficient functioning of the system, which has the ability to be deployed on many modern platforms (Windows, Linux, macOS) and is publicly available (open and free software).

Keywords: software security and licensing, client-server system, license file, encryption, digital signature.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

API – application programming interface (прикладниий програмний інтерфейс);

LDK – licensing development kit, в тексті «*Thales Sentinel Licensing Development Kit»* – програмний засіб для управління ліцензіями;

СКБД – система керування базою даних;

UML – unified modeling language (уніфікована мова моделювання).

Хеш-функція – функція яка перетворює вхідні дані довільного розміру в дані фіксованого розміру.

Хеш-код – значення що є результатом перетворення вхідних даних хеш-функцією.

Апаратні токени/донгли – фізичні пристрої або апаратні ключі які використовуються для забезпечення додаткового рівня безпеки.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де комп'ютерне програмне забезпечення (ПЗ) відіграє ключову роль у різних сферах діяльності, безпека та захист ПЗ стають надзвичайно важливими аспектами.

Захист вашого програмного забезпечення стає дедалі важливішим, оскільки його вразливі місця можуть призвести до серйозних наслідків, таких як витік конфіденційної інформації, викрадення особистих даних або контроль програм зловмисниками. Усі ці вразливості відомі давно, але в той час, коли цифрова інформаційна діяльність поширюється на нові сфери та досягає нових масштабів, ці загрози постають не лише перед звичайним користувачем, а й стають серйозною загрозою для економічної та державної безпеки.

Одним з основних завдань захисту програмного забезпечення є контроль за його використанням та ліцензуванням. Ліцензування визначає правила та умови використання ПЗ, забезпечуючи власнику ПЗ контроль над його поширенням та використанням, а також забезпечуючи прибуток розробникам. Ось кілька факторів, які підкреслюють актуальність питань ліцензуванні ПЗ:

1. Захист інтелектуальної власності: програмне забезпечення є одним із ключових елементів інтелектуальної власності, і його захист важливий для розробників і компаній, які інвестують ресурси в його створення. Зловживання програмним забезпеченням, наприклад незаконне копіювання, розповсюдження або неліцензійне використання, може призвести до втрати прибутку та порушення прав розробника. Захист інтелектуальної власності на програмне забезпечення є важливим економічним чинником, який сприяє інноваціям та розвитку високотехнологічного сектору, яким є ІТ-галузь в Україні.

2. Контроль використання програмного забезпечення: це може бути важливим фактором національної безпеки, оскільки використання неліцензійного або піратського програмного забезпечення може поставити під загрозу безпеку державних систем та інфраструктури. Незаконне використання програмного забезпечення також може створити ризики порушення правил використання та витоку даних, що може мати серйозні наслідки для національної безпеки.
3. Гарантія якості та безпеки програмного забезпечення: Ліцензування може також включати вимоги щодо якості та безпеки програмного забезпечення, забезпечуючи дотримання відповідних стандартів і правил розробки. Це може бути важливою проблемою для національної безпеки, оскільки ненадійне або незахищене програмне забезпечення може становити загрозу для захисту даних, конфіденційності та цілісності інформації, зокрема державної. Отже, захист та ліцензування програмного забезпечення в Україні є актуальними факторами економічної та національної безпеки держави з ряду причин. Вони сприяють захисту інтелектуальної власності, розкриттю економічного потенціалу, забезпеченню конкурентоспроможності національної IT-галузі, забезпеченню якості та безпеки програмного забезпечення, а також виконанню міжнародних зобов'язань.

Мета дослідження: розробка власної клієнт-серверної системи для безпечного ліцензування ПЗ. Ця система буде здатна забезпечити ефективний контроль за ліцензійними угодами, перевірку та автентифікацію користувачів, а також захист від несанкціонованого використання ПЗ.

Об'єктом дослідження є даного проекту є система безпечного ліцензування ПЗ. Це означає, що головною метою проекту є розробка ефективної клієнт-серверної системи, яка забезпечуватиме безпеку

ліцензування ПЗ, зменшуючи ризик несанкціонованого використання та поширення програмного продукту.

Предметом дослідження є різні методи та технології ліцензування, які застосовуються для забезпечення контролю та безпеки використання ПЗ. Це включає технології апаратного та програмного забезпечення, криптографічні методи, системи шифрування, алгоритми автентифікації та інші підходи, які дозволяють створити надійну систему ліцензування.

Предметна область захист та ліцензування програмного забезпечення.

Наукова новизна даного дослідження полягає у розробці нової клієнт-серверної системи для безпечного ліцензування ПЗ, яка буде відповідати сучасним вимогам безпеки та ефективності.

Для досягнення поставленої мети проекту будуть використовуватись різні методи дослідження:

1. Аналіз літературних джерел та наукових статей, що стосуються ліцензування ПЗ, систем безпеки та контролю за використанням ПЗ. Цей аналіз дозволить отримати обґрунтовану базу знань та визначити ключові аспекти безпечного ліцензування.
2. Вивчення існуючих систем ліцензування ПЗ та їх аналіз з метою визначення їх переваг та недоліків. Далі, буде проведено дослідження різних методів та технологій ліцензування, таких як апаратне та програмне забезпечення, криптографічні алгоритми, системи шифрування та автентифікації. На основі отриманих результатів буде розроблена власна клієнт-серверна система для безпечного ліцензування ПЗ.
3. Розробка клієнт-серверної системи безпечного ліцензування ПЗ. Цей етап передбачає проектування архітектури системи, розробку протоколів комунікації між клієнтом та сервером, інтеграцію

криптографічних методів та механізмів автентифікації для забезпечення безпеки.

4. Проведення експериментів та тестування розробленої системи. Цей етап дозволить оцінити функціональність та ефективність системи, а також перевірити її безпеку та стійкість до атак.
5. Аналіз отриманих результатів та формулювання висновків.

РОЗДІЛ 1
ЗАГАЛЬНА ЧАСТИНА

1.1. Аналіз попередніх досліджень

Захист та ліцензування програмного забезпечення є важливим фактором захисту інтелектуальної власності, економічної та національної безпеки. Дослідники з різних галузей, такі як право, бізнес, техніка та етика, досліджували цю проблематику з різних перспектив. Деякі з відомих дослідників, які присвятили свої роботи цій проблемі, включають:

- Річард Столлман (Richard Stallman) є відомим американським програмістом і активістом, засновником Free Software Foundation. У своїх наукових працях, зокрема у статті "Чому програмне забезпечення повинно бути безкоштовним" ("Why Software Should Be Free"), Столлман розглядає значущість захисту вільної ліцензії для прав користувачів і розробників.
- Брюс Шнайер (Bruce Schneier), також відомий американський криптограф і експерт з кібербезпеки, у своїх працях, зокрема у книзі "Секрети та брехня: цифрова безпека в мережевому світі" ("Secrets and Lies: Digital Security in a Networked World"), розглядає різні аспекти захисту програмного забезпечення та ліцензування з криптографічної та технічної безпеки.
- Лоуренс Лессіг (Lawrence Lessig), відомий американський юрист і активіст, є автором теорії "кодексу і закону" (code is law), яка стосується взаємодії правового регулювання і технологій, зокрема програмного забезпечення. У своїй роботі "Код: версія 2.0" ("Code: Version 2.0"),

Лессіг проводить аналіз впливу ліцензування та правових аспектів на поширення та використання програмного забезпечення.

У своїх наукових працях "Захист прав інтелектуальної власності на програмне забезпечення" ("Protection of intellectual property rights to software") та "Управління інноваціями в новій економіці: стратегії інтелектуальної власності та практика ліцензування програмного забезпечення" ("Innovation Management in the New Economy: Intellectual Property Strategies and Software Licensing Practices"), дослідники John F. Sherman та Thomas J. Malnight дослідили різні аспекти захисту та ліцензування програмного забезпечення з погляду бізнесу. Вони наголосили на важливості правового захисту програмного забезпечення, включаючи авторські права, патенти та умови ліцензії, для забезпечення прав власності та контролю над розповсюдженням і використанням програмного забезпечення. Вони також вказали на складнощі досягнення балансу між захистом прав розробників і користувачів, стимулюванням інновацій та сприянням відкритій співпраці, а також забезпеченням безпеки і конфіденційності під час ліцензування програмного забезпечення.

1.2. Огляд технічних методів захисту та ліцензування ПЗ

Огляд різних методів захисту програмного забезпечення є суттєвим компонентом у процесі розробки програм, оскільки це дозволяє забезпечити захист від несанкціонованого доступу, копіювання, підробки та зміни програмного коду. Для досягнення цієї мети використовуються різноманітні методи, такі як криптографічні методи, методи обфускації та віртуалізації, а також методи, що базуються на аналізі поведінки програм, разом з різноманітними методами ліцензування.

Один з ефективних методів захисту програмного забезпечення - це криптографічні методи, які забезпечують захист даних та коду шляхом шифрування і підписування [1]. Криптографічні методи можуть

використовуватись для захисту ліцензійного ключа, валідації програмного забезпечення та забезпечення цілісності та конфіденційності даних, які використовуються програмою.

Методи обфускації та віртуалізації також використовуються для захисту програмного забезпечення [2]. Обфускація полягає в затрудненні розуміння коду програми шляхом зміни його структури та логіки, тоді як віртуалізація передбачає виконання програми в ізольованому середовищі, що ускладнює аналіз та розуміння її роботи.

Методи, що базуються на аналізі поведінки програм, також використовуються для виявлення та захисту від потенційно шкідливої активності. Ці методи передбачають відстеження дій програми під час виконання та виявлення ненормальної активності, такої як відправка незвичних мережевих запитів або зміна внутрішнього стану програми, що може свідчити про можливість атаки або незаконної діяльності [3].

Огляд наявних технічних методів ліцензування програмного забезпечення може включати різні технологічні рішення, які допомагають контролювати використання програмного забезпечення та захищати інтелектуальну власність розробника. Деякі з таких технічних рішень включають:

1. Активаційні ключі: Це один з методів ліцензування, коли розробник надає спеціальні ключі активації, які користувач повинен ввести під час встановлення або активації програмного забезпечення. Ключ активації може бути пов'язаний з певним обладнанням або залежати від кількості користувачів, які мають доступ до програми.
2. Хешування апаратного забезпечення: Це ще один метод ліцензування, при якому програмне забезпечення перевіряє хеш-код апаратного забезпечення, на якому воно встановлене, і

порівнює його зі збереженим значенням. Це дозволяє розробнику обмежувати використання програмного забезпечення лише на певних комп'ютерах або пристроях.

3. Контроль доступу: Це також метод ліцензування, коли розробник встановлює механізми контролю доступу до програмного забезпечення, такі як паролі, рівні доступу або автентифікація, що дозволяють обмежувати доступ до програми лише певним користувачам або групам користувачів.
4. Шифрування: Це метод ліцензування, коли програмне забезпечення шифрується, що робить його незрозумілим або недоступним без відповідного розшифрування або ключа. Це дозволяє розробникам захистити своє програмне забезпечення від несанкціонованого використання або копіювання.
5. Віддалене управління: Це також метод ліцензування, коли розробник може віддалено керувати використанням свого програмного забезпечення. Наприклад, розробник може використовувати хмарні сервіси для моніторингу та контролю використання своєї програми на різних пристроях, включаючи вимкнення доступу до програми в разі порушення ліцензійних умов.
6. Цифрові підписи: Це ще один метод ліцензування, коли програмне забезпечення підписується цифровим ключем розробника, що дозволяє перевіряти автентичність та цілісність програми. Це допомагає відслідковувати та запобігати змінам або модифікаціям програми без дозволу розробника.
7. Ліцензійні сервери: Це також метод ліцензування, коли розробник використовує централізований сервер для контролю доступу до свого програмного забезпечення. Користувачі повинні

зв'язуватися з цим сервером для отримання ліцензії або активації програми.

8. Використання апаратних токенів або донглів: Це ще один метод ліцензування, коли розробник використовує спеціальні апаратні токени або донгли для фізичного контролю доступу до програми. Ці токени можуть містити ключі, сертифікати або іншу інформацію, необхідну для ліцензування програмного забезпечення [4].

Ці технічні підходи до ліцензування можуть бути використані окремо або в поєднанні залежно від вимог розробника та ринкових потреб. Забезпечення захисту та ліцензування програмного забезпечення може здійснюватися шляхом застосування різноманітних технічних методів, кожен з яких має свої переваги та обмеження. Ось кілька загальних прикладів:

1. Апаратний ключ (dongle):

   Переваги:

   - Високий рівень захисту: апаратний ключ має фізичну форму, що ускладнює його копіювання або злам.
   - Зручність у керуванні: інформація про ліцензію зберігається на апаратному ключі, що дозволяє зручно керувати ліцензіями та контролювати використання програмного забезпечення.
   - Можливість фізично відключити доступ до програмного забезпечення: у разі порушення ліцензійних умов, апаратний ключ можна відключити, що дозволяє заборонити використання програмного забезпечення.

   Недоліки:

   - Вартість: придбання апаратних ключів та їх обслуговування можуть бути витратними, особливо при великому масштабі використання програмного забезпечення.

- Можливість втрати або пошкодження: якщо апаратний ключ загублений або пошкоджений, це може призвести до втрати доступу до програмного забезпечення.
- Обмеження використання на різних платформах: апаратний ключ зазвичай прив'язаний до конкретної платформи, що може обмежити його використання на різних пристроях або платформах.

2. Ліцензійний файл:

Переваги:

- Зручність у поширенні: ліцензійний файл може бути надісланий електронно або включений безпосередньо в програму, що сприяє зручному розповсюдженню програмного забезпечення та ліцензій.
- Гнучкість у керуванні: ліцензійний файл може включати різні параметри ліцензування, такі як термін дії, кількість користувачів, обмеження функціональності та інші, що дозволяє гнучко налаштовувати ліцензії для різних клієнтів.

Недоліки:

- Можливість копіювання або передачі ліцензійного файлу: ліцензійний файл може бути скопійований або переданий іншим користувачам, що може призвести до незаконного використання програмного забезпечення.
- Вразливість до копіювання або редагування: ліцензійний файл може бути скопійовано або відредаговано, що може призвести до нелегального використання програмного забезпечення.
- Складність контролю використання: в залежності від реалізації, контроль за використанням ліцензійного файлу може бути складним, особливо при розповсюдженні програмного забезпечення на багатьох пристроях або платформах.

3. Мережевий ліцензування:

Переваги:

- Контроль за використанням через мережу: використання мережевого ліцензійного сервера надає можливість забезпечити ефективний контроль над використанням програмного забезпечення на різних комп'ютерах у мережі, що дає змогу забезпечити високий рівень захисту ліцензій та централізоване управління ліцензіями.
- Гнучкість у керуванні: мережевий ліцензійний сервер може надавати різні параметри ліцензування для різних користувачів, такі як кількість одночасних використань, обмеження функціональності та інші, що дозволяє гнучко налаштовувати ліцензії відповідно до потреб клієнтів.
- Проста активація та деактивація: користувачі можуть легко активувати та деактивувати ліцензії на різних комп'ютерах через мережевий ліцензійний сервер, що дозволяє ефективно управляти ліцензіями в разі зміни обладнання або потреби змінити кількість використань.

Недоліки:

- Вимагає налаштування мережевого ліцензійного сервера: використання мережевого ліцензійного сервера потребує налаштування та управління, що може бути викликом для деяких організацій.
- Залежність від мережі: використання мережевого ліцензування передбачає наявність працюючої мережі, що може бути обмеженням в деяких випадках, наприклад, при використанні програмного забезпечення в автономному режимі або на віддалених робочих місцях.

4. Хмарне ліцензування:

Переваги:

- Зручність в використанні: користувачі можуть з легкістю активувати ліцензії через хмарний сервіс, уникаючи необхідності установки та налаштування ліцензійних серверів.
- Можливість віддаленого доступу: хмарне ліцензування надає користувачам можливість отримати доступ до програмного забезпечення з різних пристроїв та місць, що забезпечує високий рівень мобільності.
- Оновлення та підтримка: хмарні сервіси можуть забезпечувати автоматичні оновлення та підтримку програмного забезпечення, що дозволяє користувачам завжди мати останні версії та запобігти виникненню помилок під час ручного оновлення.

Недоліки:

- Залежність від Інтернет-з'єднання: для використання хмарного ліцензування необхідне стабільне Інтернет-з'єднання, що може бути обмеженням у випадках зі слабким Інтернетом або відсутністю доступу до Інтернету.
- Конфіденційність даних: використання хмарного ліцензування передбачає зберігання даних ліцензій та використання програмного забезпечення на хмарних серверах, що може викликати обмеження з точки зору конфіденційності даних.
- Вартість: хмарне ліцензування може мати вищу вартість у порівнянні з іншими методами, особливо при великій кількості користувачів або великому обсязі використання [5].

Порівняльна характеристика локального, мережевого та хмарного методів:

- Активація та деактивація: Усі три методи дозволяють активувати та деактивувати ліцензії залежно від потреб користувачів. Однак мережеве та хмарне ліцензування можуть бути більш зручними, оскільки вони не потребують налаштування ліцензійних серверів.
- Управління ліцензіями: Мережеве та хмарне ліцензування забезпечують більшу гнучкість та контроль над управлінням ліцензіями, оскільки вони дозволяють централізовано керувати ліцензіями на різних комп'ютерах або пристроях.
- Залежність від мережі: Мережеве та хмарне ліцензування потребують функціонуючої мережі, що може створювати обмеження у деяких випадках. У той же час локальне ліцензування не має таких обмежень і може використовуватись в офлайн-режимі.
- Конфіденційність даних: Локальне ліцензування зазвичай забезпечує вищий рівень конфіденційності даних, оскільки ліцензійні ключі та інформація про ліцензії зберігаються на локальних комп'ютерах або пристроях. У той же час, мережеве та хмарне ліцензування вимагають передачі цих даних через мережу та їх збереження на серверах, що може бути менш безпечним.
- Вартість: Локальне ліцензування може бути економічнішим, оскільки не потребує додаткових витрат на налаштування ліцензійних серверів або підписку на хмарні послуги. Мережеве та хмарне ліцензування можуть бути дорожчими, особливо при великому обсязі використання або багатокористувацькому середовищі.
- Доступність: Локальне ліцензування може бути більш доступним в умовах з обмеженим або нестабільним Інтернет-з'єднанням, оскільки не вимагає постійного з'єднання з Інтернетом. Мережеве та хмарне ліцензування можуть бути менш доступними в таких умовах, оскільки вони потребують стабільного Інтернет-з'єднання.

Узагальнюючи, можна стверджувати, що локальне ліцензування може бути більш придатним для офлайн-роботи, забезпечувати більш високий рівень конфіденційності даних та бути економічно вигідним. Мережеве та хмарне ліцензування можуть бути більш зручними для управління ліцензіями та забезпечення централізованого контролю, але можуть мати певні обмеження в доступності та безпеці даних. Остаточний вибір між різними методами ліцензування повинен враховувати конкретні потреби організації, включаючи розмір компанії, тип діяльності, наявність Інтернету, фінансові можливості та безпекові вимоги [6].

1.3. Огляд існуючого програмного забезпечення

Ось кілька прикладів програм, які дозволяють розробникам забезпечувати управління ліцензіями для свого програмного забезпечення:

- License4J
- Thales Sentinel Licensing Development Kit (LDK)

Ці програми та рішення сприяють розробникам у захисті їх програмного забезпечення від несанкціонованого копіювання, поширення та використання, а також у керуванні ліцензіями та відстеженні використання їх продуктів. Вони забезпечують розробникам широкі можливості для налаштування ліцензійних моделей, активації, деактивації та моніторингу ліцензій, а також захищають програмне забезпечення від зловживань та несанкціонованого використання.

License4J є однією з популярних програм, яка дозволяє розробникам забезпечувати захист та ліцензування їх програмного забезпечення. Цей програмний застосунок надає розробникам зручні та потужні інструменти для створення, керування та відстеження ліцензій для їх додатків.

Основні можливості License4J включають:

1. Створення ліцензій: License4J дозволяє розробникам створювати різні типи ліцензій з різними обмеженнями, включаючи часові обмеження,

обмеження кількості користувачів, функціональні обмеження та інші параметри ліцензування.

2. Керування ліцензіями: Програма надає засоби для керування ліцензіями, такі як генерація ліцензійних ключів, активація, деактивація, відновлення, скасування та перевірка статусу ліцензій.
3. Захист від копіювання: License4J використовує різні методи захисту від несанкціонованого копіювання, такі як шифрування, підписи, хеш-коди та інші техніки, щоб запобігти несанкціонованому копіюванню та поширенню програмного забезпечення.
4. Відстеження використання: Програма дозволяє розробникам відстежувати використання своїх програм, включаючи інформацію про кількість активацій, користувачів, дати та інші дані, що дозволяє відслідковувати ліцензії та контролювати використання продуктів.
5. Кастомізація: License4J дозволяє налаштовувати зовнішній вигляд та поведінку ліцензійних вікон, повідомлень про помилки, інтерфейсу користувача та інших аспектів програми.
6. Інтеграція: Програма надає API для інтеграції з програмним забезпеченням розробників, що дозволяє автоматизувати процеси створення, активації та відстеження ліцензій безпосередньо з програмного забезпечення.
7. Підтримка різних платформ: License4J підтримує різні платформи, включаючи Java, Android, .NET, C/C++ та інші, що робить його відповідним для розробників різних типів програмного забезпечення.
8. Локальна та серверна : License4J дозволяє використовувати як локальну, так і серверну ліцензію для забезпечення захисту та управління ліцензіями на різних рівнях.
9. Підтримка різних видів ліцензування: License4J дозволяє використовувати різні види ліцензування, включаючи одноразову

ліцензію, періодичну ліцензію, ліцензію з обмеженням функціональності та інші варіанти ліцензування.

License4J, в цілому, представляє собою потужний програмний інструмент для захисту та ліцензування програмного забезпечення, що надає розробникам широкі можливості для створення та управління ліцензіями своїх додатків. Він пропонує значну кількість функціональності, гнучкі налаштування та підтримку різних платформ, що робить його популярним вибором для розробників, які прагнуть ефективно захистити своє програмне забезпечення від несанкціонованого використання [7].

Незважаючи на переваги License4J, такі як багатий функціонал та підтримка різних платформ, у нього також є деякі недоліки. До них відносяться:

1. Вартість: License4J є комерційним програмним забезпеченням, тому вартість його придбання може бути високою, особливо для невеликих розробників або стартапів з обмеженим бюджетом.
2. Складність налаштування: Налаштування License4J може бути складним процесом, особливо для новачків, які не мають попереднього досвіду роботи з ліцензуванням програмного забезпечення.
3. Залежність від стороннього рішення: License4J може вимагати інтеграції з іншими сторонніми рішеннями або бібліотеками для досягнення певного функціоналу, що може бути викликом або обмеженням для деяких розробників.

Thales Sentinel Licensing Development Kit (LDK) є програмним засобом, який дозволяє розробникам управляти ліцензіями для свого програмного забезпечення. Оглядаючи цю програму, розглянемо її можливості, переваги та недоліки.

Огляд програми та її можливостей:

- Управління ліцензіями: Thales Sentinel LDK надає розробникам широкі можливості для створення та керування ліцензіями свого програмного забезпечення. Вона дозволяє налаштовувати різні типи ліцензій, включаючи часові обмеження, обмеження функціональності та інші параметри.
- Захист від несанкціонованого використання: програма забезпечує захист від несанкціонованого використання ПЗ. Шляхом використання різних методів безпеки, такі як шифрування та цифрові підписи, щоб запобігти несанкціонованому доступу та розповсюдженню програми.
- Гнучкість налаштувань: Thales Sentinel LDK дозволяє розробникам налаштовувати різні аспекти ліцензування, включаючи обмеження функціональності, періоди пробних версій та механізми активації.
- Відстеження використання: Програма дозволяє відстежувати використання програмного забезпечення, збираючи дані про активації, користувачів та інші відомості. Це допомагає розробникам контролювати ліцензії та використання своїх продуктів [8].

Переваги Thales Sentinel LDK:

- Гнучкість: Thales Sentinel LDK надає розробникам широкі можливості для налаштування ліцензійних параметрів, що дозволяє їм враховувати особливості свого програмного забезпечення та бізнес-моделі.
- Аналітика використання: Програма забезпечує можливість збирати дані про використання програмного забезпечення, що дозволяє розробникам отримувати цінну інформацію для подальшого розвитку та вдосконалення своїх продуктів.

Недоліки Thales Sentinel LDK:

- Вартість: Якщо програма є комерційним продуктом, може виникнути висока вартість для розробників з обмеженим бюджетом.

- Складність налаштування: Налаштування параметрів ліцензування може бути складним процесом, особливо для розробників без попереднього досвіду з цією програмою.

Загалом, Thales Sentinel LDK є потужним інструментом для управління ліцензіями програмного забезпечення, забезпечуючи міцний захист та гнучкість налаштувань. Однак, розробникам слід враховувати вартість програми та можливу складність налаштування перед її використанням.

Загалом, готові програмні застосунки для ліцензування, такі як License4J, Thales Sentinel LDK чи інші, можуть бути корисними рішеннями для деяких розробників, які шукають швидкий та простий спосіб реалізації ліцензійного управління. Однак, вони також мають свої недоліки, такі як обмежена гнучкість, обмежені можливості налаштування, відсутність повного контролю ПЗ, можливість взлому та залежність від підтримки постачальника програмного забезпечення.

1.4. Висновки до розділу 1

Захист і ліцензування програмного забезпечення визнаються важливими аспектами для забезпечення прав розробників і користувачів, їх інтелектуальної власності, економічної стабільності бізнесу та національної безпеки. Вчені з різних галузей, таких як право, бізнес, інженерія та етика, досліджували ці проблеми з різних точок зору, наголошуючи на важливості ефективного захисту програмного забезпечення та ліцензування.

У контексті безпеки програмного забезпечення та ліцензування важливу роль відіграють криптографічні методи. Вони забезпечують захист даних і коду за допомогою шифрування та підпису. Криптографічні методи використовуються для захисту ліцензійних ключів, перевірки програм і забезпечення цілісності та конфіденційності даних, які використовує програма. Вони є ефективним інструментом у боротьбі з несанкціонованим використанням програмного забезпечення.

Отже, існують технічні підходи до ліцензування програмного забезпечення, такі як ключі активації, апаратне хешування, контроль доступу, шифрування, віддалене керування, цифрові підписи та використання апаратних маркерів або ключів. Кожен із цих методів має свої переваги та обмеження, і вибір залежить від конкретних потреб розробника та характеристик програмного забезпечення.

Локальне, мережеве та хмарне ліцензування – це різні методи розповсюдження ліцензій на програмне забезпечення та керування ними. Кожен з них має свої переваги та обмеження. Мережеве та хмарне ліцензування забезпечує більше контролю та гнучкості в управлінні ліцензіями та може бути зручнішим для користувачів у великих організаціях. Однак для них потрібне стабільне підключення до Інтернету, що може бути обмеженням у деяких сценаріях. Локальне ліцензування може бути більш доступним і незалежним від підключення до Інтернету, але може бути менш гнучким в управлінні ліцензіями у великих мережах.

Вибір методу ліцензування має ґрунтуватися на конкретних потребах розробника та вимогах ринку. Крім того, важливо забезпечити високий рівень захисту програмного забезпечення від несанкціонованого використання та зберегти конфіденційність даних користувача.

Існують програми та рішення, такі як License4J і Thales Sentinel Licensing Development Kit (LDK), які сприяють розробникам у вирішенні завдань ліцензування програмного забезпечення.

License4J є потужним інструментом з багатим функціоналом, гнучкими налаштуваннями та підтримкою різних платформ. Він дозволяє створювати, керувати, захищати та відстежувати ліцензії, а також підтримує різні види ліцензування.

Thales Sentinel LDK також надає гнучкість у керуванні ліцензіями та захист від несанкціонованого використання. Вона дозволяє налаштовувати різні параметри ліцензій та збирати аналітику використання.

Обидва розглянуті інструменти мають свої переваги, але також мають недоліки, такі як висока вартість та складність налаштування.

Вибір між License4J, Thales Sentinel LDK чи іншим ПЗ може залежати від потреб розробника, бюджету та рівня досвіду у роботі з ліцензуванням програмного забезпечення.

РОЗДІЛ 2
ВИБІР СПОСОБІВ ТА ЗАСОБІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

2.1. Вибір способу рішення задачі

Для розроблюваної системи було обрано наступні технічні рішення:

Мережеве ліцензування з використанням ліцензійного файлу. Передача даних здійснюється в зашифрованому каналі (за допомогою використання асинхронних ключів). Файл ліцензії додатково шифрується симетричним ключем та підписується приватним ключем з подальшою перевіркою підпису за допомогою публічного сертифіката.

2.2. Обґрунтуванню вибору способів рішення задачі

Ліцензійний файл – використовується для можливості роботи в режимі офлайн та більшої гнучкості в порівнянні з апаратним ключем.

Мережеве ліцензування – використовується для централізованого управління ліцензіями, це дозволяє зробити простим процес активації та деактивації ліцензій та значно збільшує гнучкість в їх керуванні. Вибір використання мережевого ліцензування з власним сервером також має переваги над хмарним ліцензуванням такі як, більший рівень конфіденційності даних та менша собівартість. Є також недоліки проте вони мінімізовані використанням комбінації з іншими методами (ліцензійний файл, шифрування, цифрові підписи), та такі недоліки як складність розгортання та використання залишається відносними та залежать від реалізації системи та навичок персоналу.

Використання зашифрованого каналу – забезпечує захист від перехоплення сторонніми особами ліцензії та інших даних під час комунікації клієнта системи та серверу.

Шифрування файлу ліцензії – забезпечує захист файлу ліцензії під час зберігання інформації на стороні клієнта та подвійний захист в разі дискредитацію зашифрованого каналу.

Підписування файлу ліцензії та подальша перевірка – забезпечує цілісність ліцензії (захист від її модифікування) та аутентифікацію ліцензійного серверу.

2.3. Обґрунтуванню вибору засобів розробки

- Ubuntu 18.1 як основна платформа розробки – вибір сімейства linux для забезпечення стабільної роботи та дистрибутива Ubuntu для забезпечення простоти налаштування системи [9];
- C++ з компілятором g++ 7.3.0 або пізніше – вибір мови програмування C++ для забезпечення швидкодії шифрування та системи в цілому, простоти розгортування скомпільованих програм в операційній системі;
- CMake-3.14 або пізніше – для забезпечення кросплатформенного рішення зборки програм. Використання CMake допомагає керувати проектами будь-якої складності. Він підтримує створення багатомодульних проектів з багатьма бібліотеками та виконуваними файлами та полегшує процес збирання, підтримку та розповсюдження програмного забезпечення [10];
- Boost-1.66 або пізніше – для розширення функціоналу стандартної бібліотеки STL та більш швидкої та простої кросплатформної розробки. Дана бібліотека є досить популярною та має велику спільноту розробників. Це означає, що ви можете знайти багато документації, прикладів, порад та підтримки [11];
- Protobuf v3.15.0 – для забезпечення структури повідомлень між клієнтом і сервером, як додаткова перевага, може бути використаний для взаємодії між додатками написаними різними мовами програмування [12];

- gRPC framework v1.35.0 – для забезпечення каналу взаємодії між клієнтом і сервером, кросплатформне рішення що дозволяє просто та швидко імплементувати клієнт-серверну взаємодію. В поєднанні з бібліотекою OpenSSl надає можливості для створення зашифрованого каналу передачі даних [13];
- OpenSSl бібліотека – використовується для шифрування файлів ліцензії, підписування ліцензій та забезпечення зашифрованого каналу передачі даних. Дана бібліотека є кросплатформною та має багато імплементованих алгоритмів шифрування [14].
- Qt QML – використовується для реалізації графічного інтерфейсу користувача. Даний фреймворк є кросплатформним та надає багато різноманітних можливостей для побудови графічного інтерфейсу [15].
- Nginx – для розміщення сайту з сертифікатом. Загальнодоступне рішення яке надає можливості для легкого розміщення веб сторінок в мережі [16].
- SQLite – для зберігання ліцензій. Це полегшена версія традиційної реляційної системи керування базами даних. Це вбудована база даних що не потребує додаткових ресурсів для обміну повідомленнями, бо використовує API виклики мови програмування, замість сокет технології [17].

2.4. Опис концепції системи

Проект складається з 3 компонентів:

- Серверний додаток – сервер ліцензій на стороні замовника;
- Клієнтський додаток – ліцензована програма на стороні клієнта;
- Веб-сторінка - сторінка, з якої можна завантажити сертифікат x.509 для перевірки цілісності файлу ліцензії;

Сервер ліцензування – інтернет-сервіс, який відповідає на запити ліцензування та зберігає ліцензії.

Клієнт системи – конкретне розгортання програми в середовищі клієнта. Примірник програми перевіряє встановлену ліцензію та її атрибути та застосовує політику ліцензії.

Процес перевірки підпису:

- Клієнтський додаток отримує зашифрований файл ліцензії та її підпис
- Клієнтський додаток завантажує сертифікат x.509 з веб-сайту розробника https://alfa.org і використовує його для перевірки цифрового підпису файлу ліцензії.

Процес синхронізації це періодичний механізм, який забезпечує відповідність даних ліцензування як в екземплярі програми, так і в сервері ліцензування.

Формат файлу ліцензії

{
"address_ipv4": "145.47.139.40",
"total_memory_size": 15,
"logical_cpus_number": 12,
"cpu_model": "ЦП Intel(R) Core(TM) i7-5820K @ 3,30 ГГц",
"expiration_date": "2024-12-31",
"licenced_bandwidth": 10240
}

Концептуальна діаграма зображена на Рис. 2.1.

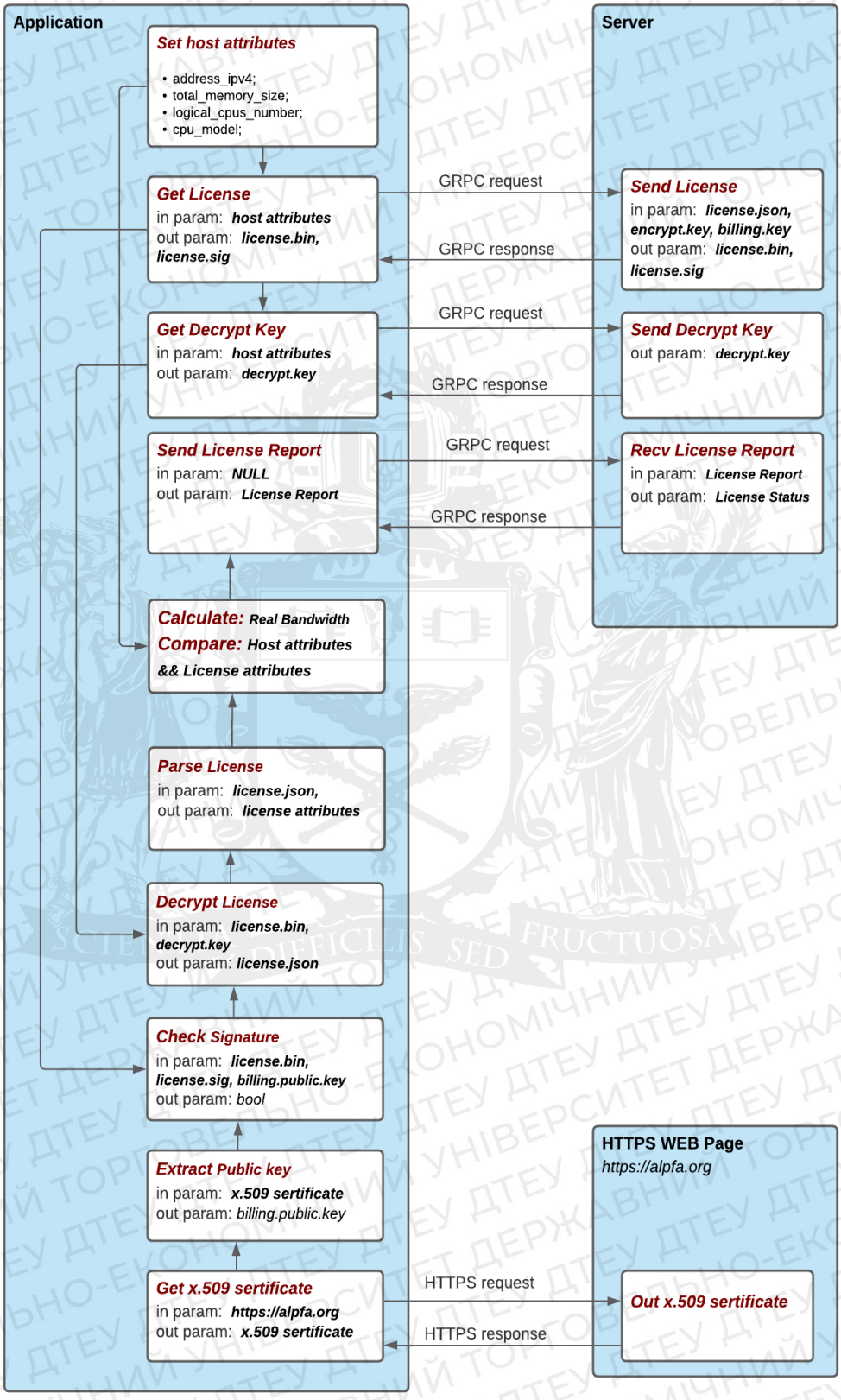

Рис. 2.1. License server Data Flow

Алгоритм формування та перевірки ліцензії

Опис:

•Для отримання ліцензії клієнт звертається до сервера.

•Сервер видає клієнту зашифрований і підписаний файл ліцензії.

•Клієнт перевіряє справжність ліцензії за допомогою сертифіката.

Обмеження:

1) Ключі генеруються один раз і використовуються лише для шифрування ліцензії на стороні сервера та розшифрування на стороні клієнта.
2) Для передачі відкритого ключа клієнту є два варіанти:
   – вбудовується в код програми;
   – передається із сервера за запитом.

Діаграми процесів шифрування підпису та перевірки підпису представленні на Рис. 2.2-2.3

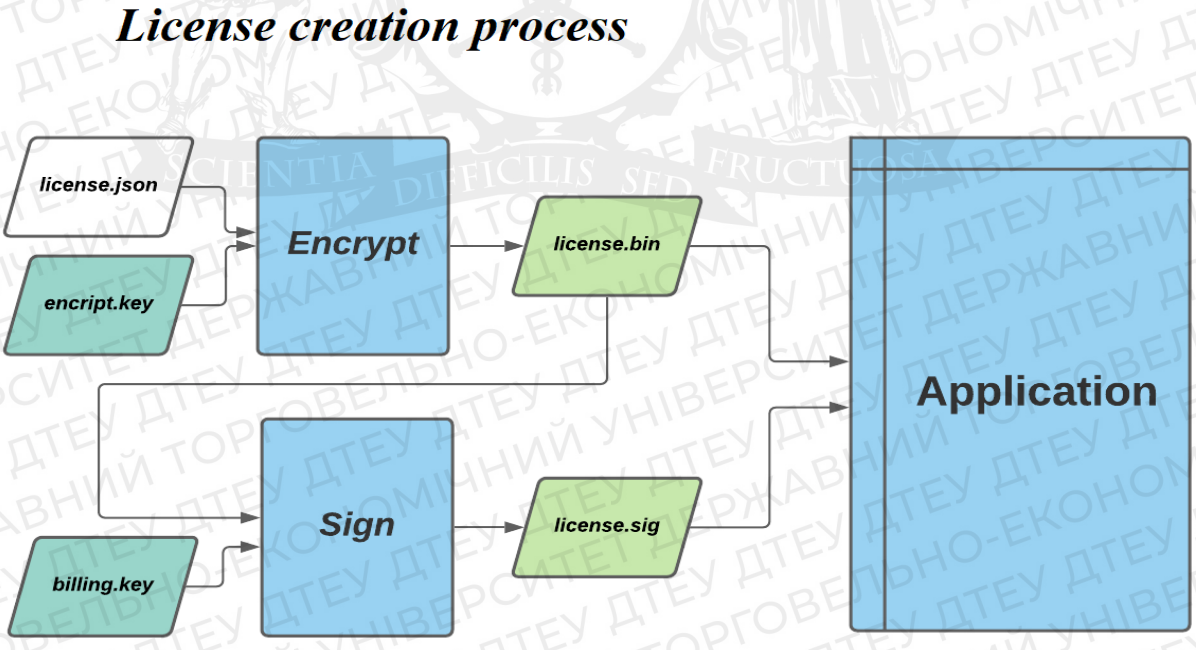

Рис. 2.2. Шифрування та підписання ліцензії за допомогою приватних ключів

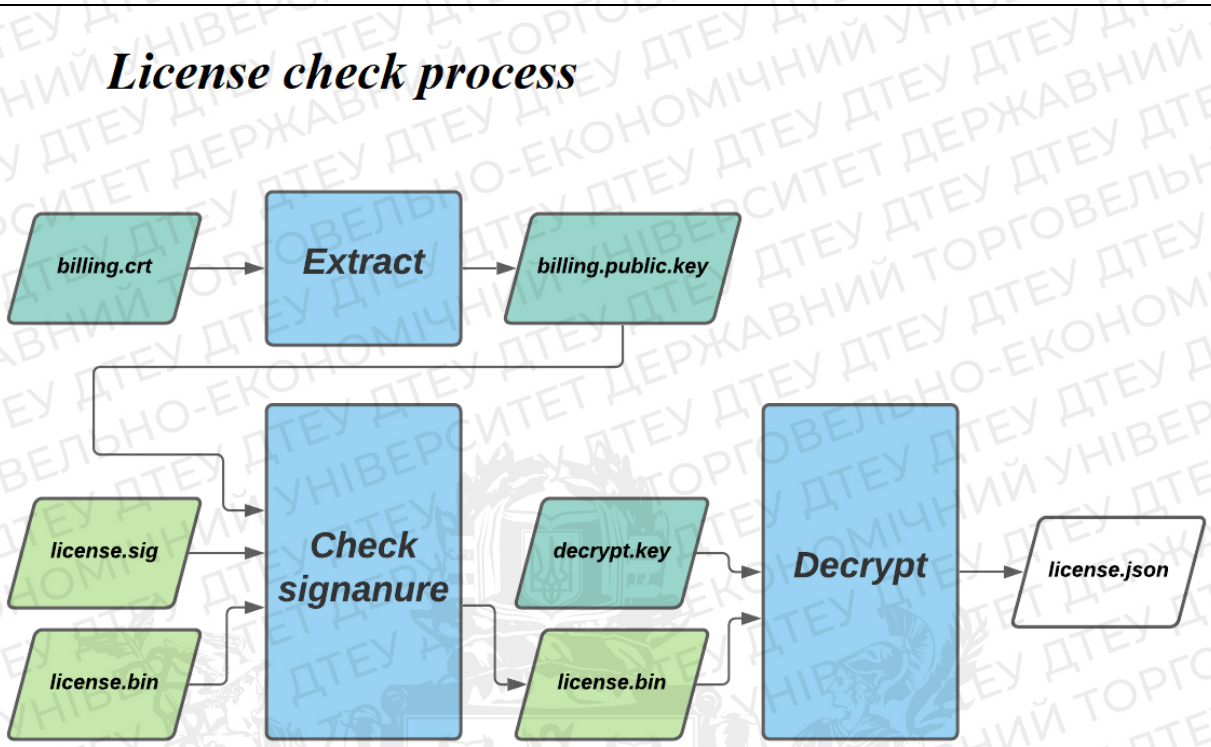

Рис. 2.3. Для перевірки цілісності ліцензії використовується сертифікат

2.5. Висновки до розділу 2

Цей розділ був присвячений вибору рішення для поставленої задачі та обґрунтуванню засобів для її реалізації.

Отже серед інших методів було обрано наступні: мережеве ліцензування з використанням ліцензійного файлу, де передача даних здійснюється в зашифрованому каналі (за допомогою використання асинхронних ключів), в якості даних виступають, зашифрований симетричним ключем файл ліцензії, його підпис та симетричний ключ для розшифрування файлу ліцензії. Перевірка цілісності ліцензії здійснюється за допомогою публічного сертифіката та підпису.

Підхід до вибору засобів реалізацію був спрямований на забезпеченні надійного та ефективного рішення, що має здатність розгортання на багатьох

сучасних платформах (Windows, Linux, macOS) та є загальнодоступним (відкрите програмне забезпечення, безоплатне програмне забезпечення).

Система складається з трьох компонентів (сервер, додаток, веб сторінка), що розміщені на різних комп'ютерах та взаємодіють між собою через мережу Інтернет.

В якості ідентифікатора клієнта використовується IP адреса та дані про його апаратне забезпечення.

Мережева комунікація між клієнтом та сервером відбувається за допомогою gRPC викликів. Комунікація між клієнтом та веб сторінкою відбувається з використанням HTTPS протоколу.

Клієнтський додаток має можливість відправити серверу запити для отримання ліцензії та отримання ключу для розшифрування файлу ліцензії; відправити серверу звіт про використання ліцензії; відправити запит на отримання сертифікату з веб сторінки; перевірити цілісність файлу ліцензії; розшифрувати файл ліцензії; порівнювати показники параметрів системи що вказані в ліцензії з поточними показниками.

Сервер має можливість відповідати клієнту на запити отримання ліцензії та отримання ключу для розшифрування файлу ліцензії; приймати звіти про використання ліцензії від клієнтів.

Веб сторінка розміщує сертифікат та надає можливості будь якому клієнту його отримати.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ

3.1. Склад інформаційного забезпечення

Включає:

- Інформаційні ресурси – це дані та інформація, які зберігаються і обробляються в системі ліцензування. Це дані користувачів, дані ліцензії (такі як, обсяги використання ліцензійних параметрів, дата закінчення ліцензії тощо) ліцензійні ключі, логи подій, сертифікати тощо.
- Апаратне та програмне забезпечення – включає в себе обладнання, на якому працює система, а також програми, які виконують різні функції, включаючи обробку та зберігання інформації, аутентифікацію тощо. Серед прикладів апаратного обладнання є: користувацькі комп'ютери, сервер ліцензування, сервер що розміщує веб сторінку з сертифікатом. Серед прикладів програмного забезпечення є: клієнтський додаток, який потребує ліцензування; серверний додаток, що приймає запити на ліцензування та видає ліцензії; база даних, що зберігає ліцензії; nginx, що розміщує веб сторінку з сертифікатом.
- Інфраструктура мережі – оскільки система ліцензування використовує клієнт-серверну архітектуру, мережева інфраструктура здійснює незаперечно важливу роль у забезпеченні комунікації між клієнтами та сервером. Для забезпечення мережевої інфраструктури було обрано механізм виклику віддалених процедур, реалізований за допомогою qRPC фреймворка.

3.2. Структура інформаційного забезпечення

Базується на принципах:

- Конфіденційності – забезпечення обмеженого доступу до інформації лише авторизованим користувачам. Це досягається за допомогою механізмів автентифікації. В даному випадку на основі IP адреси клієнта та даних про апаратне забезпечення.
- Цілісності – гарантування недопущення несанкціонованих змін інформації. Це досягається за допомогою механізмів контролю цілісності даних та перевірки їх при зберіганні та передачі. Цей принцип досягається за допомогою підписування ліцензії та її подальшої перевірки за допомогою сертифікату.
- Доступності – забезпечення доступу до інформації та ресурсів системи в будь-який час, коли це необхідно. Для цього використовуються механізми резервного копіювання, балансування навантаження та інші підходи. Цей принцип досягається завдяки розділенню повноважень між сервером видачі ліцензій, клієнтським додатком та сервером з сертифікатом.

3.3. Вибір технологій доступу до баз даних

Всі ліцензійні дані, інформація про користувачів та ПЗ зберігаються в централізованій базі даних, що дозволяє забезпечити єдиний джерело інформації та спрощує її керування. Завдяки використанню реляційній моделі даних забезпечується простота та структурованість даних. У якості системи керування базою даних (СКБД) виступає SQLite, завдяки тому що ця СКБД є вбудованою, вона не використовує багато ресурсів та може використовуватися в вбудованих системах.

Забезпечується регулярне резервне копіювання бази даних для запобігання втраті даних у випадку аварій.

3.4. Методи контролю інформації

Включають:

- Шифрування – застосування криптографічних методів для захисту конфіденційності даних під час їх передачі та зберігання.
- Аутентифікація і авторизація – встановлення і перевірка ідентичності користувачів та надання їм відповідних прав доступу.

3.5. Створення та налагодження програми

Демонстраційні програми реалізована у межах дипломної роботи, створені на C++ з використанням наступних технологій:

- Cmake – для забезпечення кросплатформенного зборки рішення
- OpenSSL – для шифрування та підписування ліцензій, створення симетричних та асиметричних ключів
- gRPC – для забезпечення мережевої комунікації
- Protobuf – для забезпечення структури повідомлень між клієнтом і сервером
- SQLite – для зберігання та доступу до ліцензій
- Qt QML– для реалізації графічного інтерфейсу користувача
- Boost – для роботи з пам’яттю
- Nginx – для розміщення сайту з сертифікатом.

Система складається з 3 компонентів: клієнт, сервер, веб-сторінка.

Операційна система – Ubuntu, як основна платформа розробки та впровадження

3.6. Опис функцій системи

Розглянемо докладніше основні функції програми, які представлені у вигляді діаграм прецедентів на рисунку 3.1.

На першій діаграмі виділено одного актора, який є користувачем системи, котрому потрібно отримати ліцензію.

На діаграмі актор-користувач має назву «Оператор» і може виконувати такі дії:

1. Підготувати дані для запиту ліцензії

1.1. Отримання даних про апаратне забезпечення оператора

2. Завантажити ліцензію
3. Перевірити підпис ліцензії

3.1. Завантаження публічного сертифікату

4. Завантажити ключ для розшифрування ліцензії
5. Розшифрувати ліцензію
6. Отримати дані з ліцензії
7. Порівняти ліцензійні параметри з поточними
8. Відправити звіт про ліцензійні та поточні параметри
9. Отримати час серверу

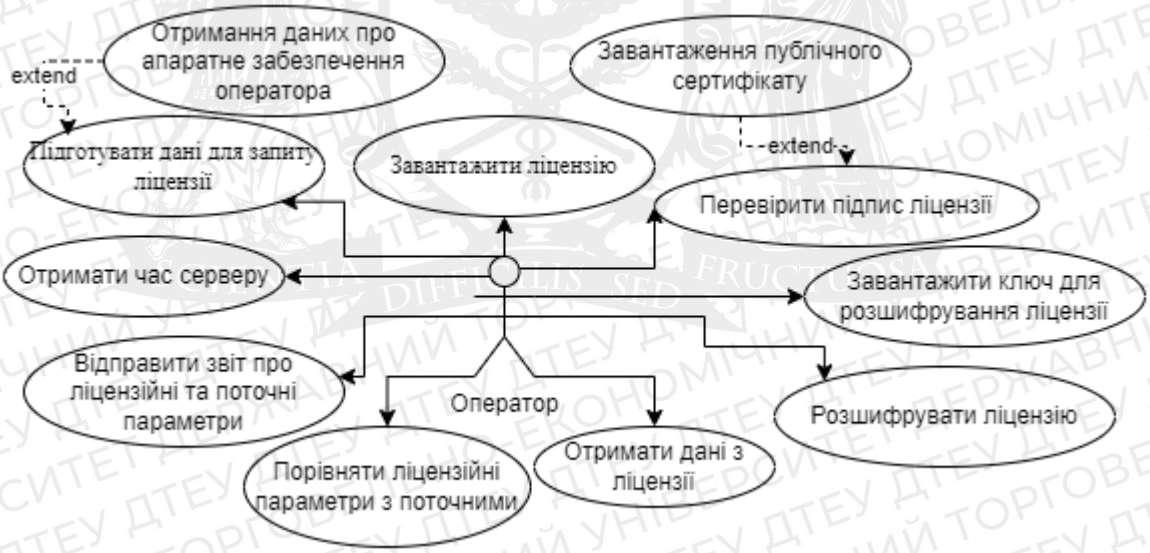

Рис. 3.1. UML- діаграма прецедентів: Оператор

На рисунку 3.2 зображено алгоритм функції отримання ліцензії на серверній частині програми.

Згідно алгоритму функція отримання ліцензії на серверній частині починає роботу з перевірки на коректність даних запиту що надіслав клієнт. В

разі хибності виразу, метод в результаті свого виконання виводить повідомлення та повертає помилку.

Якщо перевірка пройдена, то викликається метод отримання існуючої ліцензії для даного клієнту. Якщо існуюча ліцензія отримана виводиться повідомлення про статус ліцензії та повертається результат клієнту (у вигляді зашифрованого ліцензійного файлу та його підпису).

Якщо отримати існуючу ліцензію не вдалося тоді викликається метод для отримання нової ліцензії. Якщо отримати нову ліцензію не вдалося, тоді виводиться повідомлення та повертається помилка.

Якщо отримати нову ліцензію вдалося тоді виводиться повідомлення про статус ліцензії та повертається результат клієнту (у вигляді зашифрованого ліцензійного файлу та його підпису).

Вихід з функції.

Функція отримання існуючої ліцензії починає роботу зі спроби отримання ліцензії зі сховища. Якщо це зробити не вдалось тоді виводиться повідомлення про помилку.

Якщо ліцензія отримана зі сховища успішно тоді викликається метод отримання ліцензійних файлів (зашифрований файл ліцензії та його підпис) зі сховища.

Якщо отримати ліцензійні файли зі сховища не вдалося тоді викликається метод створення нових ліцензійних файлів (на основі отриманої ліцензії зі сховища).

Якщо створити нові ліцензійні файли не вдалося тоді виводиться повідомлення та повертається помилка. Якщо нові ліцензійні файли було створено успішно тоді вони повертаються з функції.

Вихід із функції.

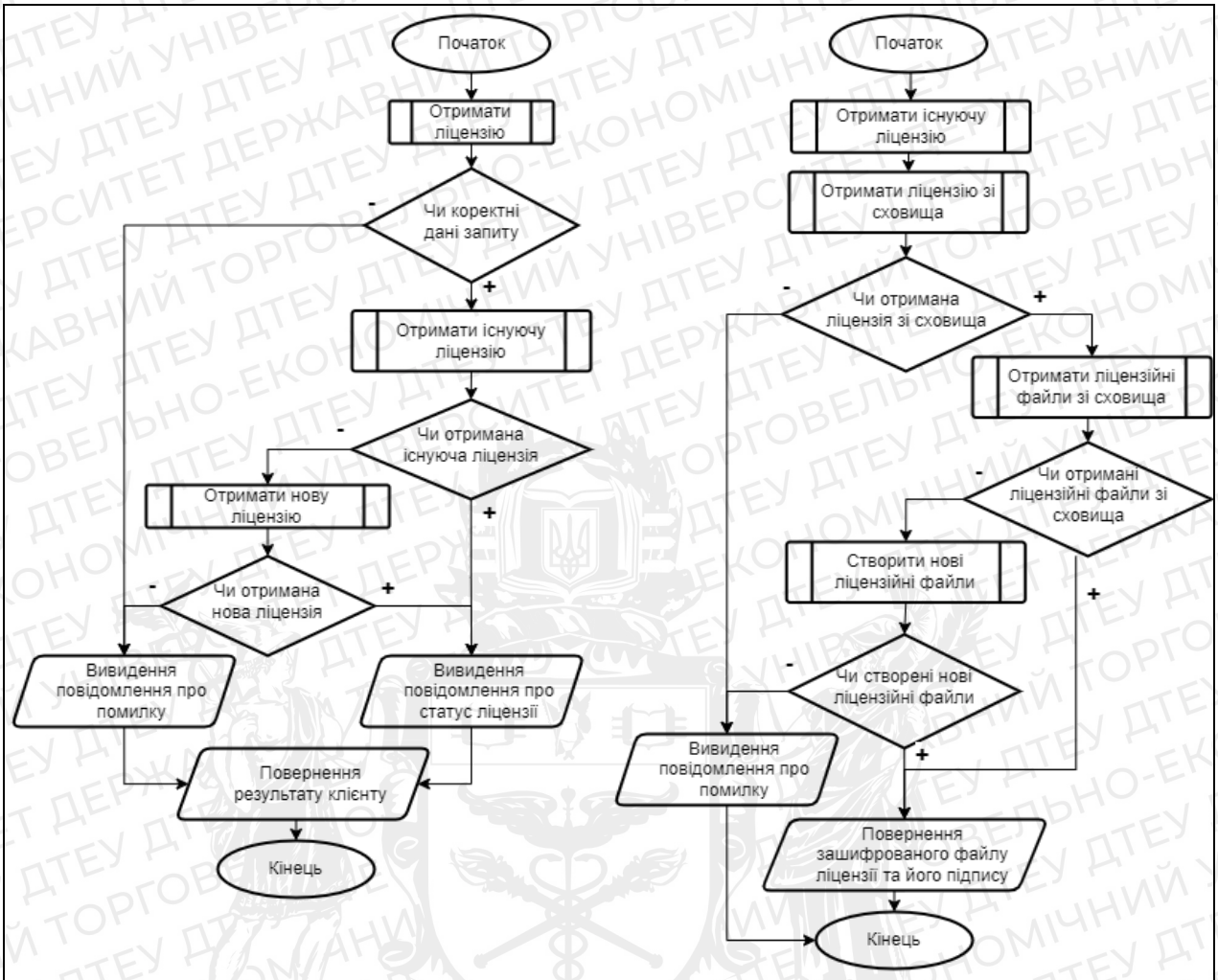

Рис. 3.2. Блок-схема розробленого алгоритму

3.7. Опис функціональних блоків системи

Клієнт-серверна система складатися з таких функціональних блоків, як зображено на рисунку 3.3.

Користувацький додаток.

Блок взаємодії з користувачем – обробляє дані введені користувачем, а також дії, що виконує користувач (див. рис. 3.1).

Блок управління – керує процесом запиту та перевірки ліцензії використовуючи блок роботи з криптографією та блок мережевої взаємодії.

Блок мережевої взаємодії – дозволяє відправляти запити на ліцензію та приймати файли ліцензії та службову інформацію в мережі.

Серверний додаток.

Блок мережевої взаємодії – дозволяє приймати запити на ліцензію та відправляти файли ліцензії та службову інформацію в мережі.

Блок управління – керує процесом запиту та видачі ліцензії та додаткової інформації використовуючи блок роботи з криптографією та блок мережевої взаємодії.

Блок роботи з базою даних – дозволяє зберігати дані ліцензії в базі даних.

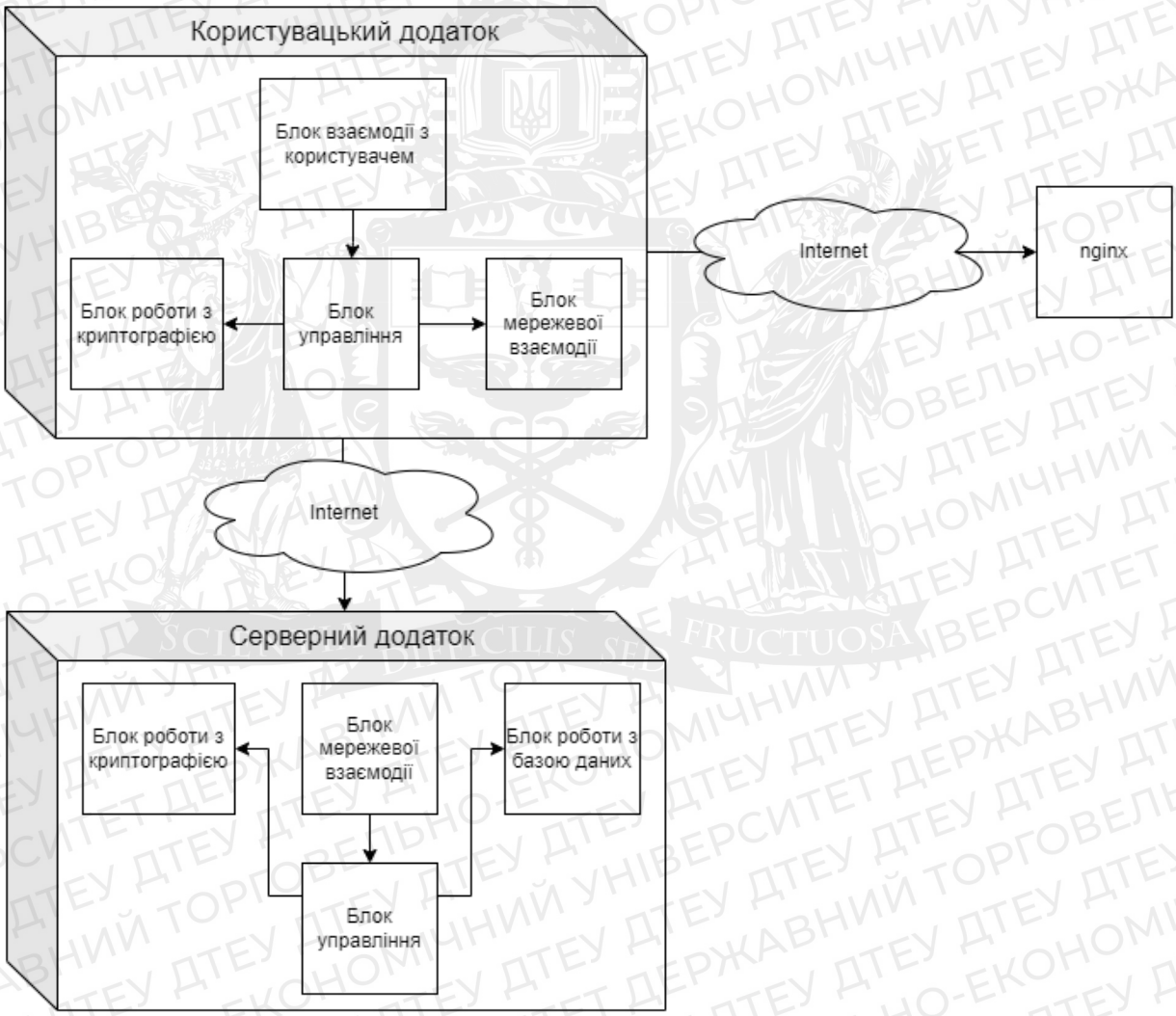

Рис. 3.3. Функціональна схема системи

3.8. Опис клієнтського додатку

В якості клієнтського додатку було розроблено демонстраційний стенд, який відображає функції що виконує система. Демонстраційний стенд

представлений у вигляді графічного застосунку та має головне вікно (див. рис. 3.4) з наступними функціоналом:

1. Користувач може викликати дії на правій бічній панелі головної форми програми.
2. Натискання кнопки «Init» провокує заповнення параметрів про апаратне забезпечення, ці параметри необхідні для надсилання запиту на отримання ліцензії.
3. Натискання кнопки «Get License» відправляє запит серверу на отримання ліцензії, в результаті повертається ліцензійні файли.
4. Натискання кнопки «Check Signature» завантажує з веб-сторінки сертифікат та перевіряю підпис отриманих ліцензійних файлів.
5. Натискання кнопки «Get Decrypt Key» відправляє запит серверу на отримання ключу для розшифрування файлу ліцензії.
6. Натискання кнопки «Decrypt License» розшифровує файл ліцензії.
7. Натискання кнопки «Parse License» отримує дані ліцензії з розшифрованого файлу.
8. Натискання кнопки «Compare License Attributes» порівнює рівень поточного показники з рівнем ліцензійного показника.
9. Натискання кнопки «Send Report» відправляє звіт про використання ліцензійного показника серверу.
10. Натискання кнопки «Get Server Time» відправляє запит на отримання часу серверу, в результаті повертається локальний час серверу.
11. По черзі кроків ліцензування загоряються індикатори ліцензії.
12. Деталі ліцензії можна побачити в текстовій області «License Details».
13. Параметр що ліцензується це пропускна здатність на мережевому інтерфейсі комп’ютера користувача, він може мати динамічну поведінку та його використання відображається в колонці правої частини вікна.

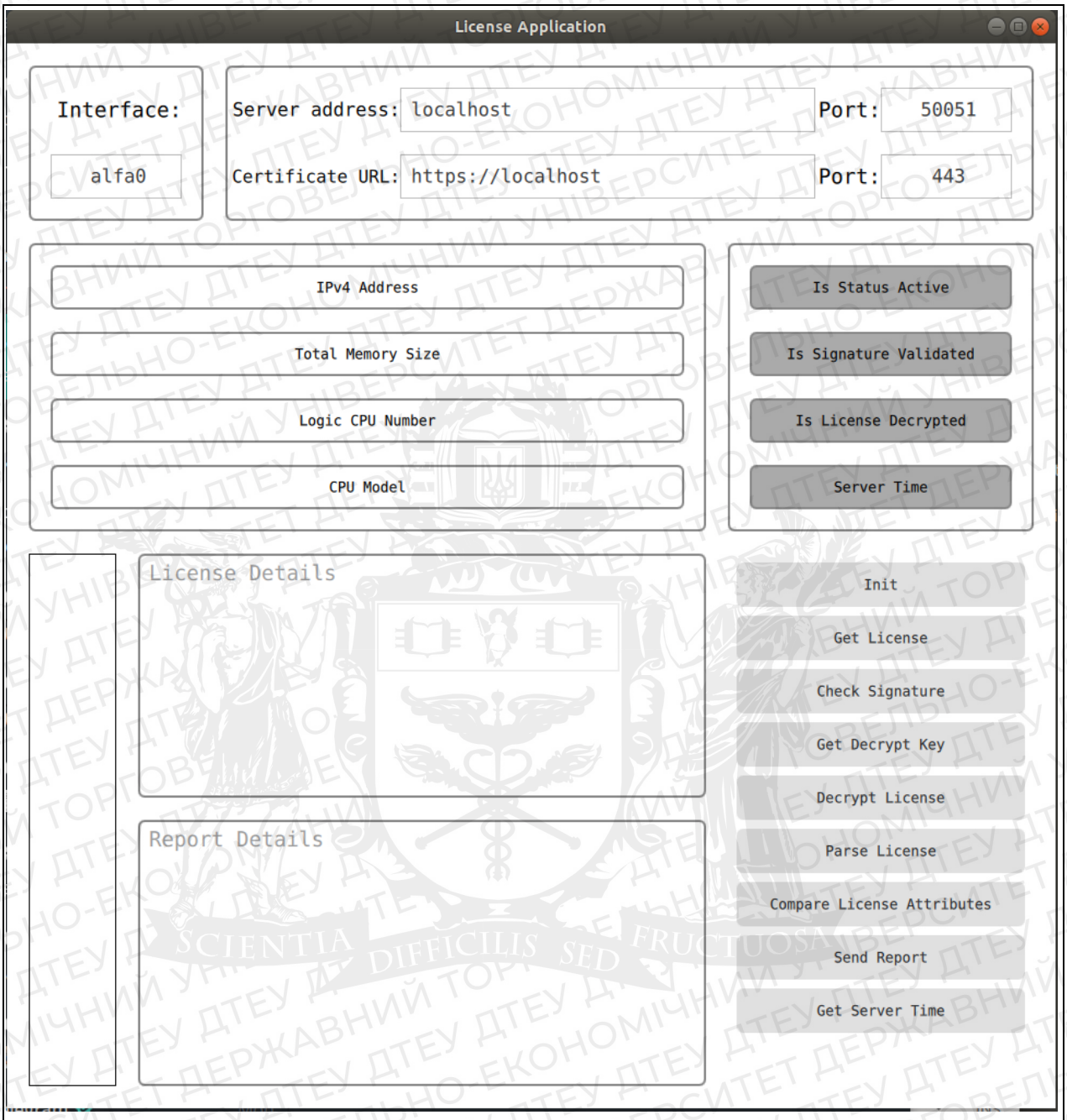

Рис. 3.4. Головне вікно клієнтського додатку

3.9. Висновки до розділу 3

Визначивши склад, структуру програмного забезпечення, вибір технології доступу до БД та методи контролю інформації було описано функції системи та представлено наступні діаграми: UML- діаграма прецедентів оператора, блок-схема розробленого алгоритму дій серверного додатку. Також було описано функціональні блоки системи та надана діаграма

функціональної схема системи. Описано головне вікно клієнтського додатку та надано відповідний рисунок.

Реалізація системи здійснена з використанням мови програмування C++ та Qt QML фреймворка для кросплатформної реалізації графічного інтерфейсу клієнтського додатку.

За зберігання та доступ до даних ліцензій на серверній частині відповідає СКБД SQLite, забезпечуючи надійне зберігання та швидкий доступ до даних.

За мережеву комунікацію відповідає технологія виклику віддалених процедур qRPC.

За реалізацію криптографічних методів таких як шифрування, підписування даних, створення симетричних та асиметричних ключів та сертифікатів відповідає бібліотека OpenSSL.

За розміщення веб сторінки в мережі Інтернет відповідає веб сервер Nginx.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У цьому проекті була детально розглянута тема захисту та ліцензування програмного забезпечення. Дослідження охопило аналіз попередніх досліджень з цієї тематики, огляд технічних методів захисту та тестування програмного забезпечення, існуючих програмних рішень License4J та Thales Sentinel Licensing Development Kit (LDK).

Загальні висновки:

- Важливість захисту та ліцензування ПЗ. Захист та ліцензування ПЗ є критичними аспектами для розробників, користувачів, бізнесу та держави загалом, оскільки порушення цілісності програмного продукту чи неліцензований доступ може призвести до значних збитків, порушень в працездатності ПЗ, збоїв в роботі критичних систем.
- Важливість відповідного підходу. Ефективне ліцензування програмного забезпечення вимагає комплексного підходу, який враховує правильну стратегію ліцензування, технічні, юридичні та безпекові аспекти.
- Готові рішення. Програми, які спеціалізуються на ліцензуванні, такі як License4J і Thales Sentinel LDK, надають розробникам зручні інструменти для створення та керування ліцензіями. Однак, перш ніж користуватися цими або іншим готовими програмними рішеннями для ліцензування, розробник повинен ретельно обдумати переваги і недоліки такого вибору, врахувати свої потреби та вимоги, а також оцінити витрати та ризики, пов'язані з використанням готового програмного забезпечення.
- Немає універсального рішення. Немає єдиного універсального рішення для всіх ситуацій. Вибір програми для ліцензування повинен базуватися

на конкретних потребах та обставинах розробника. У деяких випадках розробникам варто розглянути альтернативні опції, такі як власна розробка рішення для ліцензування або використання рішень з відкритим вихідним кодом, які можуть забезпечити більший рівень гнучкості та контролю.

Розроблена система

Серед інших методів були обрані такі підходи: мережеве ліцензування, при якому передача даних відбувається по зашифрованому каналу з використанням асинхронних ключів. Даними в цьому процесі є файл ліцензії, зашифрований симетричним ключем, його підпис і симетричний ключ для розшифровки файлу ліцензії. Цілісність ліцензії перевіряється за допомогою публічного сертифіката та підпису.

Підхід до вибору цих методів полягав у створенні надійного та ефективного рішення, яке має можливість працювати на різних платформах, таких як Windows, Linux і macOS. Було важливо забезпечити доступність цього рішення, поширюючи його як відкрите та безкоштовне програмне забезпечення. Саме тому було обрано використання популярних та загальнодоступних інструментів та бібліотек.

Подальші перспективи для розробки полягають в безпосередній інтерграції системи з клієнтським програмним забезпеченням, яке потребує ліцензування та розширення ліцензованих параметрів, відповідно до потреб клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC press, pp. 2-6.
2. Collberg, C., & Nagra, J. (2009). Surreptitious software: Obfuscation, watermarking, and tamperproofing for software protection. Addison-Wesley, pp. 13-20.
3. Chandra, P., & Arora, R. (2018). A Comprehensive Survey on Software Protection and Licensing Techniques. International Journal of Computer Applications, 181(47), pp. 8-15.
4. Zhou, L., Deng, H., & Varadharajan, V. (2015). Software protection and licensing: A survey. Journal of Systems and Software, 107, pp. 166-185.
5. Balasubramanian, S., & Bhowmik, S. (2019). A survey on software protection techniques: from traditional to modern approaches. International Journal of Information Technology, 11(1), pp. 87-96.
6. Chandra, R., Wondracek, G., & Holz, T. (2011). "The Battle against Phishing: Dynamic Security Skins." In 2011 20th International Conference on Computer Communications and Networks (ICCCN), pp. 1-6.
7. License4J \\ Режим доступу: https://www.license4j.com/ (останнє звернення 29.03.2023р.)
8. Thales Sentinel Licensing Development Kit (LDK) \\ Режим доступу: https://cpl.thalesgroup.com/software-monetization/license-development-kit-ldk (останнє звернення 30.03.2023р.)
9. Офіційний сайт Ubuntu \\ Режим доступу: https://ubuntu.com (останнє звернення 05.04.2023р.)

10. Офіційний сайт CMake \\ Режим доступу: https://cmake.org/about/ (останнє звернення 07.04.2023р.)
11. Офіційний сайт Boost \\ Режим доступу: https://www.boost.org (останнє звернення 10.04.2023р.)
12. Офіційний сайт Protocol Buffers (protobuf) \\ Режим доступу: https://protobuf.dev/overview/ (останнє звернення 13.04.2023р.)
13. Офіційний сайт gRPC \\ Режим доступу: https://grpc.io/about/ (останнє звернення 13.04.2023р.)
14. Офіційний сайт OpenSSL \\ Режим доступу: https://www.openssl.org (останнє звернення 15.04.2023р.)
15. Документація Qt QML \\ Режим доступу: https://doc.qt.io/qt-6/qtqml-index.html (останнє звернення 16.04.2023р.)
16. Офіційний сайт Nginx \\ Режим доступу: https://www.nginx.com/resources/faq/ (останнє звернення 17.04.2023р.)
17. Офіційний сайт SQLite \\ Режим доступу: https://www.sqlite.org/about.html (останнє звернення 17.04.2023р.)

ТЕХНІЧНЕ ЗАВДАННЯ

1. Загальні відомості

1.1. Найменування системи

«Клієнт-серверна система безпечного ліцензування»

1.2. Планові терміни початку та закінчення робіт

Початок робіт 01.06.2023

Закінчення робіт 15.10.2023

1.3. Головний бенефіціар та потенційні користувачі системи

Виробники програмного забезпечення

2. Мета та призначення створення системи

2.1. Призначення системи

Забезпечити контроль над ліцензіями та захистити програмне забезпечення від несанкціонованого використання.

2.2. Мета створення системи

Полягає у досягненні наступних цілей:

2.2.1. Забезпечити технічні засобів контролю для поширення легального використання програмного забезпечення.

2.2.2. Гарантувати, що користувачі використовують програмне забезпечення відповідно до ліцензійних умов та не порушають авторські права розробника.

2.2.3. Захисту від піратства. Головною метою є запобігання несанкціонованому копіюванню, розповсюдженню та використанню програмного забезпечення, що може завдати збитків розробнику, та спричинити шкоду системі користувача в наслідок роботи вірусного програмного забезпечення.

3. Вимоги до системи

3.1. Вимоги до системи в цілому

3.1.1. Вимоги до структури та функціонування системи, перелік підсистем

3.1.1.1. Система має клієнт-серверну архітектуру, де сервер відповідає за зберігання та обробку ліцензій, а клієнтські додатки взаємодіють з сервером для отримання та активації ліцензій.

3.1.1.2. Для забезпечення безпеки використовуються шифрування та цифрові підписи для перевірки цілісності ліцензій.

3.1.1.3. Підсистеми: серверна частина (серверний додаток та база даних), клієнтська частина, веб-сервер де розміщена веб-сторінка.

3.1.2. Вимоги до способів і засобів інформаційного обміну між компонентами системи

3.1.2.1. Обмін повідомленнями повинен відбуватися в зашифрованому каналі передачі даних.

3.1.2.2. Обмін між клієнтом та сервером повинен бути реалізований за допомогою викликів віддалених процедур gRPC.

3.1.3. Вимоги до режимів функціонування системи

3.1.3.1. Режим створення ліцензій. Цей режим дозволяє розробникам створювати ліцензійні ключі для свого програмного забезпечення.

Вимоги:

3.1.3.1.1. Можливість вказувати обмеження, такі як часові обмеження, кількість користувачів та обмеження функціональності.

3.1.3.1.2. Підтримка різних видів ліцензій.

3.1.3.1.3. Захист від несанкціонованого доступу та змін ліцензійних даних.

3.1.3.2. Режим відстеження використання. Цей режим дозволяє розробникам відстежувати, як користувачі використовують їхнє програмне забезпечення.

Вимоги:

3.1.3.2.1. Збір та збереження даних про активації, користувачів, дати та інші відомості.

3.1.3.2.2. Забезпечення конфіденційності та цілісності зібраних даних.

3.1.3.3. Режим керування ліцензіями. У цьому режимі розробники можуть проводити операції з ліцензіями, такі як деактивація, відновлення, скасування та перевірка статусу.

Вимоги:

3.1.3.3.1. Можливість виконувати операції з ліцензіями через інтерфейс системи.

3.1.3.3.2. Захист від несанкціонованих змін ліцензій та їх статусу.

3.1.3.4. Режим відповіді на запити клієнтів. Система повинна бути готовою відповідати на запити клієнтів щодо активації, деактивації та інших операцій з ліцензіями.

Вимоги:

3.1.3.4.1. Висока доступність системи для обробки запитів.

3.1.3.4.2. Швидка реакція на запити користувачів.

3.1.4. Показники призначення

3.1.4.1. Параметри, що характеризують ступінь відповідності системи призначенням:

3.1.4.1.1. Надійність: Система має бути надійною та забезпечувати коректну роботу призначення навіть в умовах внутрішніх і зовнішніх помилок.

3.1.4.1.2. Швидкодія: Система повинна оптимально виконувати всі функції, що стосуються обробки запитів користувачів, забезпечуючи високу продуктивність.

3.1.4.1.3. Безпека: Система має гарантувати захист ліцензійних даних інформації від несанкціонованого доступу та маніпуляцій.

3.1.4.1.4. Гнучкість: Система повинна бути гнучкою та дозволяти змінювати параметри ліцензій, відстежувати користувачів та контролювати використання продуктів.

3.1.4.2. Вимоги до пристосовності системи до змін:

3.1.4.2.1. Масштабованість: Система повинна бути масштабованою для забезпечення росту числа користувачів та обсягів роботи.

3.1.4.2.2. Інтеграція: Система повинна легко інтегруватися з іншим програмним забезпеченням, що використовується розробниками.

3.1.4.2.3. Змінність: Система повинна допускати зміни в правилах та параметрах ліцензування без значного впливу на її роботу.

3.1.4.2.4. Підтримка різних платформ: Система повинна бути адаптованою для різних операційних систем і платформ (Windows, Linux, macOS тощо).

3.1.4.3. Вимоги до збереження працездатності системи в різних ймовірних умовах:

3.1.4.3.1. Запас потужності: Система повинна мати достатній запас потужності для подолання навантаження в періоди підвищеної активності.

3.1.4.3.2. Резервне збереження даних: Повинно бути організовано резервне зберігання ліцензійних даних для відновлення в разі втрати або пошкодження.

3.1.4.3.3. Захист від критичних помилок: Система повинна мати заходи безпеки для запобігання серйозним помилкам, які можуть призвести до втрати ліцензійних даних чи злому системи.

3.1.5. Вимоги до надійності

3.1.5.1. Склад показників надійності до системи в цілому:

3.1.5.1.1. Доступність системи: Система повинна бути доступною для користувачів протягом визначеного часу без великих перерв у роботі.

3.1.5.1.2. Відмовостійкість: Система повинна продовжувати функціонувати, навіть якщо сталася помилка або збій в роботі окремих компонентів.

3.1.5.1.3. Стійкість до втрати даних: Система повинна забезпечувати захист даних від втрати або пошкодження.

3.1.5.2. Вимоги до надійності технічних засобів і програмного забезпечення:

3.1.5.2.1. Надійність апаратного забезпечення: Технічні засоби, на яких працює система, повинні бути надійними і стійкими до відмов.

3.1.5.2.2. Надійність програмного забезпечення: Програмне забезпечення повинно бути стійким до помилок та вразливостей, які можуть вплинути на надійність системи.

3.1.5.2.3. Засоби резервного збереження: Потрібно передбачити резервне зберігання даних та можливість відновлення системи в разі втрати даних або виходу з ладу основної системи.

3.1.5.3. Вимоги до методів оцінки і контролю показників надійності на різних стадіях створення системи:

3.1.5.3.1. Аналіз і проектування:

3.1.5.3.1.1. Моделювання надійності: Використовувати моделі для прогнозування надійності системи на етапі проектування.

3.1.5.3.2. Розробка:

3.1.5.3.2.1. Кодування обробки помилок та відновлення.

3.1.5.3.2.2. Автоматизоване тестування: Використовувати автоматизовані тести для виявлення помилок та вразливостей на ранніх етапах розробки.

3.1.5.3.3. Тестування:

3.1.5.3.3.1. Функціональні тести надійності: Виконувати функціональні тести для перевірки, чи працює система відповідно до очікувань.

3.1.5.3.3.2. Тести на відмовостійкість: Провести тести на відмовостійкість для визначення, як система реагує на збої та відмови.

3.1.5.3.4. Експлуатація та підтримка:

3.1.5.3.4.1. Постійне оновлення: Регулярно випускати оновлення та виправлення для забезпечення надійності та безпеки.

3.1.5.3.5. Аудит безпеки:

3.1.5.3.5.1. Регулярний аудит безпеки: Проводити регулярний аудит безпеки для виявлення та виправлення можливих вразливостей, які можуть вплинути на надійність системи.

3.1.5.3.6. Резервне збереження та відновлення:

3.1.5.3.6.1. Тестування процедур відновлення: Перевіряти процедури відновлення системи з резервного копіювання для забезпечення швидкого та надійного відновлення в разі втрати даних або збою.

3.1.6. Вимоги до захисту інформації від несанкціонованого доступу

3.1.6.1. Автентифікація та авторизація:

3.1.6.1.1. Система повинна мати механізм автентифікації, що дозволяє ідентифікувати користувачів перед наданням доступу.

3.1.6.2. Шифрування даних:

3.1.6.2.1. Всі дані, які передаються між клієнтом і сервером, повинні бути зашифровані для унеможливлення перехоплення чи читання несанкціонованими особами.

3.1.6.2.2. Застосовувати сучасні криптографічні протоколи та алгоритми для забезпечення безпеки даних.

3.1.6.3. Захист від SQL-ін'єкцій та інших атак:

3.1.6.3.1. Перевірка та фільтрація всіх введених користувачем даних перед їх використанням в запитах до бази даних.

3.1.6.3.2. Захист від вразливостей типу "кросс-сайтовий скриптінг" та інших типів атак на веб-застосунки.

3.1.6.4. Моніторинг та реагування на інциденти:

3.1.6.4.1. Система повинна мати механізми моніторингу та журналювання подій для виявлення незвичайної активності.

3.1.6.4.2. Розробити план реагування на інциденти, включаючи відключення від системи та інші дії для забезпечення безпеки в разі атаки.

3.1.6.5. Захист від несанкціонованого доступу до системи:

3.1.6.5.1. Забезпечити фізичну безпеку серверів та обладнання, де розміщена система.

3.1.6.5.2. Застосовувати заходи захисту мережі, такі як брандмауери та віртуальні приватні мережі (VPN).

3.1.6.6. Юридичні заходи:

3.1.6.6.1. Розглянути можливість внесення до угоди з користувачем положень про відповідальність за порушення безпеки та незаконний доступ.

3.1.7. Вимоги до захисту від впливу зовнішніх факторів

3.1.7.1. Захист від природних катастроф:

3.1.7.1.1. Забезпечити фізичний захист серверів і обладнання від природних катастроф, таких як пожежі, повені, землетруси і т. д.

3.1.7.1.2. Регулярно робити резервне копіювання даних та зберігати їх в інших географічних регіонах або хмарних системах.

3.1.7.2. Захист від кібератак:

3.1.7.2.1. Захистити систему від кібератак і хакерських атак за допомогою застосування брандмауерів, виявлення вторгнень та інших засобів.

3.1.7.2.2. Вчасно оновлювати програмне забезпечення та застосовувати патчі для усунення вразливостей.

3.1.7.3. Захист від технічних відмов:

3.1.7.3.1. Забезпечити наявність резервних каналів і серверів для уникнення відмови у роботі системи внаслідок технічних проблем.

3.1.7.4. Захист від втрати даних:

3.1.7.4.1. Здійснювати регулярне резервне копіювання даних та забезпечувати можливість відновлення системи в разі втрати даних.

3.1.7.4.2. Захищати дані від випадкового видалення чи зміни.

3.1.8. Вимоги безпеки

3.1.8.1. Аутентифікація і авторизація:

3.1.8.1.1. Система повинна мати механізми аутентифікації користувачів, які гарантують, що лише авторизовані особи мають доступ до системи.

3.1.8.2. Шифрування:

3.1.8.2.1. Для захисту конфіденційної інформації під час передачі повинен використовуватися шифрування.

3.1.8.2.2. Захищати дані на сервері та в базі даних шляхом шифрування відповідно до встановлених стандартів.

3.1.8.3. Захист від вірусів і зловмисних програм:

3.1.8.3.1. Регулярно оновлювати антивірусне програмне забезпечення для захисту від загроз.

3.1.8.3.2. Забезпечити обмеження можливості виконання зловмисних програм на серверах і комп'ютерах користувачів.

3.1.8.4. Фізична безпека:

3.1.8.4.1. Захищати сервери та мережеве обладнання фізично від несанкціонованого доступу.

3.1.8.4.2. Забезпечити захист обладнання від природних катастроф та випадкових пошкоджень.

3.1.8.5. Захист даних:

3.1.8.5.1. Визначити процедури для регулярного резервного копіювання даних і забезпечити їх захист від втрати і пошкодження.

3.1.8.5.2. Встановити правила для доступу до і зберігання конфіденційної інформації.

3.1.8.6. Безпека мережі:

3.1.8.6.1. Забезпечити захист мережі системи від атак і небажаних доступів, використовуючи мережеві брандмауери та інші технічні засоби.

3.2. Перелік підсистем системи

3.2.1. Серверна частина

Повинна мати такі основні функції:

3.2.1.1. Зберігання ліцензій та відомостей про клієнтів.

3.2.1.2. Створення ліцензій та оброблення запитів на ліцензування клієнтів.

3.2.1.3. Отримання звітів від клієнтів системи про використання ліцензійних показників.

3.2.1.4. Активація ліцензій та відкликання вже активованих ліцензій.

3.2.2. Клієнтська частина

Повинна мати такі основні функції:

3.2.2.1. Отримання даних про поточне апаратне забезпечення.

3.2.2.2. Надсилання запитів серверу для отримання ліцензії.

3.2.2.3. Завантаження ліцензійних файлів з сервера.

3.2.2.4. Завантаження сертифікату розробника з веб-сторінки.

3.2.2.5. Перевірка підпису ліцензії та збереження ліцензії на клієнтському пристрої.

3.2.2.6. Розшифрування зашифрованої ліцензії.

3.2.2.7. Генерація та передача звітів на сервер щодо використання ліцензій.

3.2.3. Веб-сервер

Повинен виконувати такі основні функції:

3.2.3.1. Розміщення веб-сторінки з сертифікатом

3.3. Вимоги до видів забезпечення

3.3.1. Вимоги до інформаційного забезпечення

3.3.1.1. Доступність інформації: Забезпечити безперервний доступ до системи.

3.3.1.2. Конфіденційність: Забезпечити захист конфіденційної інформації від несанкціонованого доступу та розголошення.

3.3.1.3. Цілісність інформації: Забезпечити збереження інформації в незмінному стані та виявлення будь-яких змін чи пошкоджень.

3.3.1.4. Аутентифікація і авторизація: Реалізувати механізми аутентифікації користувачів та авторизації доступу до різних частин системи.

3.3.1.5. Резервне копіювання: Забезпечити регулярне резервне копіювання інформації для захисту від втрати даних.

3.3.2. Вимоги до програмного забезпечення

3.3.2.1. Програмне забезпечення повинно бути сумісним з усіма операційними системами, на яких буде використовуватися система.

3.3.2.2. Програмне забезпечення повинно забезпечувати вимоги безпеки, зокрема, щодо аутентифікації, шифрування та захисту даних.

3.3.2.3. Регулярно оновлювати програмне забезпечення для виправлення виявлених вразливостей.

3.3.3. Вимоги до технічного забезпечення:

3.3.3.1. Система повинна бути розгорнута на технічному обладнанні, яке відповідає вимогам щодо продуктивності та надійності.

Вимоги до середовища виконання:

3.3.3.1.1. Мінімальні системні вимоги для клієнту системи:

3.3.3.1.1.1. Оперативна пам’ять: 2 Гб;

3.3.3.1.1.2. 8 Гб вільного дискового простору

3.3.3.1.1.3. Процесор: Intel Pentium 4 і вище, частота від 1.3 МГц;

3.3.3.1.1.4. Операційна система Microsoft Windows 10 / Ubuntu 18.0;

3.3.3.1.2. Мінімальні системні вимоги для фізичного серверу системи:

3.3.3.1.2.1. Оперативна пам’ять: 8 Гб;

3.3.3.1.2.2. 64 Гб вільного дискового простору

3.3.3.1.2.3. Процесор: Intel Core i5 і вище, частота від 2.3 ГГц;

3.3.3.1.2.4. Операційна система Ubuntu 18.0;

3.3.3.2. Технічне обладнання має бути забезпечене резервним живленням та заходами безпеки.

3.3.3.3. Забезпечити регулярне обслуговування та моніторинг технічного обладнання.

4. Технології

4.1. Операційні платформи

4.1.1. Основна платформа розробки

Ubuntu 18.1

4.1.2. Підтримка платформ

4.1.2.1. Windows 10

4.1.2.2. macOS 11

4.2. Мова програмування

4.2.1. C++ з компілятором g++ 7.3.0 або пізніше

4.3. Система збірки

CMake версії 3.14 або пізніше

4.4. База даних

SQLite

4.5. Веб-сервер

Nginx

4.6. Бібліотека для графічного інтерфейсу

Qt QML

4.7. Бібліотеки для мережевої взаємодії

4.7.1. gRPC framework v1.35.0

4.7.2. Protobuf v3.15.0

4.8. Бібліотеки для роботи з криптографією

4.8.1. OpenSSL

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

1. Програма тестування

Тестування системи безпечного ліцензування буде проводитися з метою перевірки відповідності функціоналу системи вимогам та забезпечення її стабільності та безпеки.

Програма тестування включатиме наступні етапи:

1.1. Функціональне тестування

Цей етап буде спрямований на перевірку всіх функцій системи відповідно до вимог. Включатиме тестування створення, активації та відстеження ліцензій, а також інші функції, пов'язані із забезпеченням ліцензування.

1.2. Тестування на безпеку

На цьому етапі буде перевірятися стійкість системи до можливих загроз та атак. Включатиме тестування на несанкціонований доступ, шифрування даних та відновлення після можливих витоків у системі безпеки.

1.3. Тестування витривалості

Важливий етап, спрямований на перевірку роботи системи протягом тривалого періоду. В ході цього тестування будуть проведені довготривалі тести з метою перевірки стабільності та продуктивності системи та перевірку відновлення системи після можливих витоків ресурсів.

1.4. Інтеграційне тестування

На етапі інтеграційного тестування буде перевірятися взаємодія різних компонентів системи та їхній сумісний функціонал.

2. Методика тестування

2.1. Створення тестових сценаріїв

Розробка повних тестових сценаріїв, що охоплюють всі аспекти функціоналу та можливі варіанти використання системи.

2.2. Симуляція атак

Використання спеціальних інструментів для симуляції різних видів атак з метою перевірки реакції системи на них.

2.3. Визначення метрик тестування

Установлення конкретних метрик для оцінки результатів тестування, таких як швидкість відновлення, виявлення помилок та ступінь покриття функціоналу тестами.

2.4. Стрес тестування

Використання тестових навантажень для оцінки роботи системи в умовах максимального навантаження.

2.5. Ручне тестування

Паралельно з автоматизованим тестуванням, проведення ручних тестів для перевірки специфічних сценаріїв та виявлення можливих помилок.

2.6. Перевірка документації

Перевірка відповідності документації та інструкцій реальним можливостям та функціоналу системи.

2.7. Регулярне оновлення тестових сценаріїв

В процесі розробки оновлення тестових сценаріїв для врахування змін у функціоналі та вимогах.

3. Процедури тестування:

3.1. Тестування повинно включати проведення тестових сценаріїв, які охоплюють всі можливі варіанти використання системи.

3.2. Тестування на безпеку повинно включати витяги, зокрема, намагання несанкціонованого доступу до ліцензійних даних та спроби злому системи.

3.3. Тестування на витривалість повинно проводитися протягом тривалого часу та з великими обсягами даних.

4. Критерії успішності тестування

Успішне тестування передбачає виконання наступних критеріїв:

4.1. Функціональність системи відповідає задокументованим вимогам.

4.2. Система витримує тривале тестування та великі обсяги даних.

5. План тестування:

5.1. Розробити докладний план тестування, включаючи розподіл завдань та терміни виконання.

5.2. Визначити метрики для вимірювання результатів тестування.

6. Звітність тестування

Після завершення тестування скласти звіт, що містить результати, виявлені помилки та їхні описи, а також заходи, які вжиті для усунення недоліків.

7. Виправлення помилок

По закінченні тестування, виявлені помилки та уразливості повинні бути виправлені перед впровадженням системи чи нової версії системи в експлуатацію.

8. Перевірка та підтвердження відповідності:

8.1. Технічні засоби системи повинні бути перевірені та підтверджені, що вони відповідають стандартам та специфікаціям.

ДОДАТКИ

Додаток А

Лістинг програмного коду

Файл grpc_service.hpp

```
#ifndef GRPC_SERVER_HPP
#define GRPC_SERVER_HPP
#include <boost/filesystem.hpp>
#include <boost/json.hpp>
#include <fstream>
#include <grpc++/grpc++.h>
#include <string>

#include "alfa.h"
#include "data_layer.hpp"
#include "license.grpc.pb.h"
#include "sqlite_database.hpp"

namespace grpcService
{
    using grpc::Server;
    using grpc::ServerBuilder;
    using grpc::ServerContext;
    using grpc::Status;
    // NOLINTNEXTLINE
    namespace bf = boost::filesystem;
    // NOLINTNEXTLINE
    namespace dl = data_layer;

    using namespace basecamp;

    class GrpcService final : public LicenseAPI::Service
    {
      public:
        Status GetDecryptKey(ServerContext *context, const Hardware
*harwareDetails,
                             DecryptKey *key) override;
        Status GetLicense(ServerContext *context, const Hardware
*harwareDetails,
                          License *license) override;
        Status GetServerTime(ServerContext *context, const Blank *blank,
                             Time *time) override;
        Status LicenseReport(ServerContext *context, const Report *report,
                             Blank *blank) override;

        static void SetPathToKey(std::string pathToKey);

      private:
        bool GetExistLicense(const Hardware *harwareDetails, License
*license);
        bool GetNewLicense(const Hardware *harwareDetails, License *license);

        bool CreateLicense(const Hardware *harwareDetails, dl::License
*license);
        bool CreateLicenseFilesContent(const dl::License &license_info,
                                       std::string *license_bin,
                                       std::string *license_sig);
```

```
        bool GetLicenseFromStorage(const Hardware *harwareDetails,
dl::License *license);
        bool SaveLicenseToStorage(const dl::License &license);
        bool GetLicenseFilesContentFromStorage(const dl::License &license,
                                               std::string *license_bin,
                                               std::string *license_sig);
        bool SaveLicenseFilesContentToStorage(const dl::License &license,
                                              const std::string &license_bin,
                                              const std::string
&license_sig);
        inline void ActivateLicense(License *license);

        data_layer::License FormationLicense(const Hardware *harwareDetails);
        bool FormationLicenseFilesContent(const dl::License &license_info,
                                          std::string *license_bin,
                                          std::string *license_sig);

        inline bool IsValidHardwareDetails(const Hardware *harwareDetails);
        std::string GetExpirationData();
        int GetLicensedBandwidth();

        bool Encrypt(const std::string &key_fname, const std::string
&license_json,
                     std::string *license_bin);
        bool Sign(const std::string &key_fname, const std::string
&license_bin,
                  std::string *license_sig);

        std::string GetKeyPath(const char *fileName);
        std::string GetFileContentAsString(const char *fileName);

        static std::string PathToKey;
        Sqlite::SqliteDatabase database;
    };
} // namespace grpcService

#endif
```

Файл grpc_service.cpp

```
#include "grpc_service.hpp"
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iterator>
#include <sstream>

namespace grpcService
{
    constexpr const char *const kEnvironmentVariable = "LICENSE_SERVER";
    constexpr const char *const kEncryptFileName = "encrypt.key";
    constexpr const char *const kDecryptFileName = "decrypt.key";
    constexpr const char *const kBillingFileName = "billing.key";
    std::string GrpcService::PathToKey = "";

    Status GrpcService::GetDecryptKey(ServerContext *context,
                                      const Hardware *harwareDetails,
DecryptKey *key)
    {
        std::cout << "\n\n============================\n";
        std::cout << "============================\n";
```

```
        std::cout << "\nGetDecryptKey request with following info:\n"
                  << "\tCore Count: " << harwareDetails->cpu_number() << '\n'
                  << "\tCPU String model: " << harwareDetails->cpu_model() <<
'\n'
                  << "\tMemory size: " << harwareDetails->memory_size() <<
'\n'
                  << "\tIPv4 Address: " << harwareDetails->ipv4_address() <<
'\n';

        std::string decryptFileContent =
GetFileContentAsString(kDecryptFileName);
        key->set_size(decryptFileContent.size());
        key->add_key(decryptFileContent);

        std::cout << "\nResponse sent\n";
        std::cout << "\n============================\n";
        std::cout << "============================\n";

        return ::grpc::Status::OK;
    }

    Status GrpcService::GetLicense(ServerContext *context, const Hardware
*harwareDetails,
                                   License *license)
    {
        /// TODO: change log
        std::cout << "\n\n============================\n";
        std::cout << "============================\n";
        std::cout << "\nGetLicense request with following info:\n"
                  << "\tCore Count      : " << harwareDetails->cpu_number()
<< '\n'
                  << "\tCPU String model: " << harwareDetails->cpu_model() <<
'\n'
                  << "\tMemory size     : " << harwareDetails->memory_size()
<< '\n'
                  << "\tIPv4 Address    : " << harwareDetails->ipv4_address()
<< '\n';

        if (false == IsValidHardwareDetails(harwareDetails))
        {
            /// TODO: change log
            std::cerr << "Error GetLicense, HarwareDetails is invalid\n";
            return {grpc::StatusCode::ABORTED, "HarwareDetails is invalid"};
        }

        if (false == GetExistLicense(harwareDetails, license))
        {
            std::cout << "\nTrying to create a new license\n";
            if (false == GetNewLicense(harwareDetails, license))
            {
                /// TODO: change log
                std::cerr << "Server-side error, GetLicense\n";
                return {grpc::StatusCode::INTERNAL,
                        "Server-side error while creating license"};
            }
        }

        /// TODO: change log
        std::cout << "\nFinal: The license was granted, state license: "
                  << license->status() << '\n';
        std::cout << "\n============================\n";
        std::cout << "============================\n";
```

```
        return Status::OK;
    }

    Status GrpcService::GetServerTime(ServerContext *context, const Blank
*blank,
                                      Time *time)
    {
        std::cout << "\n\n============================\n";
        std::cout << "============================\n";

        time->set_value(0);
        std::time_t result = std::time(nullptr);
        time->set_value(result);

        std::cout << "\nServer time: " << result << "sec\n";
        std::cout << "\n============================\n";
        std::cout << "============================\n";

        return Status::OK;
    }

    Status GrpcService::LicenseReport(ServerContext *context, const Report
*report,
                                      Blank *blank)
    {
        std::cout << "\n\n============================\n";
        std::cout << "============================\n";

        std::cout << "\nGet report with following info:\n"
                  << "\tStatus: " << report->status() << '\n'
                  << "\tType: " << report->type() << '\n'
                  << "\tMemory size: " << report->memory_size() << '\n'
                  << "\tCPU number: " << report->cpu_number() << '\n'
                  << "\tCPU molde: " << report->cpu_model() << '\n'
                  << "\tIPv4 address: " << report->ipv4_address() << '\n'
                  << "\tHardware bandwidth: " << report->hardware_bandwidth()
<< '\n'
                  << "\tLicense bandwidth: " << report->license_bandwidth()
<< '\n'
                  << "\tValidate time: " << report->validate_time() << '\n'
                  << "\tVerify status: " << report->verify_status() << '\n'
                  << "\tDescription: " << report->description() << '\n';

        std::cout << "\n============================\n";
        std::cout << "============================\n";

        return Status::OK;
    }

    bool GrpcService::GetExistLicense(const Hardware *harwareDetails, License
*license)
    {
        dl::License license_info;
        if (false == GetLicenseFromStorage(harwareDetails, &license_info))
        {
            /// TODO: change log
            std::cerr << "GetLicenseFromStorage, license not found.\n";
            return false;
        }

        std::string license_bin, license_sig;
        if (false ==
```

```
            GetLicenseFilesContentFromStorage(license_info, &license_bin,
&license_sig))
        {
            std::cerr << "GetLicenseFilesContentFromStorage, license files
not found.\n";
            std::cout << "\nTrying to create license files.\n";
            if (false ==
                CreateLicenseFilesContent(license_info, &license_bin,
&license_sig))
            {
                /// TODO: change log
                std::cerr << "Error GetExistLicense\n";
                return false;
            }
        }
        license->set_status(license_info.status());
        license->add_license_bin(std::move(license_bin));
        license->add_license_sig(std::move(license_sig));

        return true;
    }

    bool GrpcService::GetNewLicense(const Hardware *harwareDetails, License
*license)
    {
        dl::License license_info;
        if (false == CreateLicense(harwareDetails, &license_info))
        {
            /// TODO: change log
            std::cerr << "Error GetNewLicense\n";
            return false;
        }

        std::string license_bin, license_sig;
        if (false == CreateLicenseFilesContent(license_info, &license_bin,
&license_sig))
        {
            /// TODO: change log
            std::cerr << "Error GetNewLicense\n";
            return false;
        }
        license->set_status(license_info.status());
        license->add_license_bin(std::move(license_bin));
        license->add_license_sig(std::move(license_sig));

        return true;
    }

    bool GrpcService::CreateLicense(const Hardware *harwareDetails,
dl::License *license)
    {
        *license = FormationLicense(harwareDetails);
        license->status(LicenseStatus::ACTIVE);

        if (false == SaveLicenseToStorage(*license))
        {
            license->status(LicenseStatus::NOTEXIST);

            /// TODO: change log
            std::cerr << "Error CreateLicense\n";
            return false;
        }
        return true;
```

```
    }

    bool GrpcService::CreateLicenseFilesContent(const dl::License
&license_info,
                                                std::string *license_bin,
                                                std::string *license_sig)
    {
        if (false == FormationLicenseFilesContent(license_info, license_bin,
license_sig))
        {
            /// TODO: change log
            std::cerr << "Error CreateLicenseFilesContent\n";
            return false;
        }
        if (false ==
            SaveLicenseFilesContentToStorage(license_info, *license_bin,
*license_sig))
        {
            /// TODO: change log
            std::cerr << "Error CreateLicenseFilesContent\n";
            return false;
        }
        return true;
    }

    dl::License GrpcService::FormationLicense(const Hardware *harwareDetails)
    {
        dl::License license;
        license.hardware_info(*harwareDetails);
        license.expiration_date(GetExpirationData());
        license.licensed_bandwidth(GetLicensedBandwidth());
        license.license_name(GetHash(license));

        /// TODO: change log
        std::cout << "\n\nFormation of a license:\n";
        std::cout << "----------------------------\n";
        std::cout << "Licensed bandwidth: " << license.licensed_bandwidth()
<< "\n";
        std::cout << "Expiration date: " << license.expiration_date() <<
"\n";
        std::cout << "----------------------------\n";

        return license;
    }

    bool GrpcService::FormationLicenseFilesContent(const dl::License
&license_info,
                                                   std::string *license_bin,
                                                   std::string *license_sig)
    {
        std::cout << "\nFormation of license files.\n";
        std::cout << "\nStep 1: Create license.json" << std::endl;

        std::string license_json;
        try
        {
            license_json = dl::SerializeToJson(license_info);
        }
        catch (const std::exception &e)
        {
            /// TODO: change log
            std::cerr << "Error FormationLicenseFilesContent,
SerializeToJson: "
```

```
                    << e.what() << '\n';
            return false;
        }

        // license_json.assign(dl::ToPrettyJson(license_json));
        std::cout << "----------------------------" << std::endl;
        std::cout << license_json;
        std::cout << "----------------------------" << std::endl;

        auto const encrypt_key_path = GetKeyPath(kEncryptFileName);

        std::cout << "\nStep 2: Encrypt license.json to license.bin" <<
std::endl;
        std::cout << "----------------------------" << std::endl;
        std::cout << "Uses key: " << encrypt_key_path << std::endl;
        std::cout << "----------------------------" << std::endl;

        if (false == Encrypt(encrypt_key_path, license_json, license_bin))
        {
            /// TODO: change log
            std::cerr << "Error FormationLicenseFilesContent\n";
            return false;
        }

        auto const billing_key_path = GetKeyPath(kBillingFileName);

        std::cout << "\nStep 3: Sign license.bin to license.sig" <<
std::endl;
        std::cout << "----------------------------" << std::endl;
        std::cout << "Uses key: " << billing_key_path << std::endl;
        std::cout << "----------------------------" << std::endl;

        if (false == Sign(billing_key_path, *license_bin, license_sig))
        {
            /// TODO: change log
            std::cerr << "Error, Sign\n";
            return false;
        }

        return true;
    }

    inline void GrpcService::ActivateLicense(License *license)
    {
        license->set_status(LicenseStatus::ACTIVE);
    }

    bool GrpcService::GetLicenseFromStorage(const Hardware *harwareDetails,
                                            dl::License *license)
    {
        try
        {
            /// TODO: change log
            std::cout << "\n----------------------------\n";
            std::cout << "Trying to get license from db" << std::endl;
            std::cout << "----------------------------\n";

            auto dto_hardware =
DTO::HardwareDTO::GetDtoFromHardware(*harwareDetails);

            dto_hardware = database.SelectFromTable(dto_hardware);
            if (dto_hardware.Id == -1)
            {
```

```
            /// TODO: change log
            std::cerr << "No hardware information found\n";
            return false;
        }

        auto dto_license =
database.SelectLicenseByHardwareId(dto_hardware.Id);
        if (dto_license.Id == -1)
        {
            /// TODO: change log
            std::cerr << "License not found\n";
            return false;
        }

        *license = DTO::LicenseDTO::GetLicenseFromDto(
            dto_license,
DTO::HardwareDTO::GetHardwareFromDto(dto_hardware));
    }
    catch (const std::exception &e)
    {
        /// TODO: change log
        std::cerr << "Error GetLicenseFromStorage: " << e.what() << '\n';
        return false;
    }
    return true;
}

bool GrpcService::SaveLicenseToStorage(const dl::License &license)
{
    try
    {
        /// TODO: change log
        std::cout << "\n----------------------------\n";
        std::cout << "Save license to db" << std::endl;
        std::cout << "----------------------------\n";

        auto dto_hardware =

DTO::HardwareDTO::GetDtoFromHardware(license.hardware_info());
        auto dto_license = DTO::LicenseDTO::GetDtoFromLicense(license);

        int res = database.InsertToTable(dto_hardware);
        if (res != -1)
        {
            dto_license.HardwareId = res;
        }
        else
        {
            dto_hardware = database.SelectFromTable(dto_hardware);
            if (dto_hardware.Id == -1)
            {
                /// TODO: change log
                std::cerr << "Error SaveLicenseToStorage";
                return false;
            }
            dto_license.HardwareId = dto_hardware.Id;
        }

        res = database.InsertToTable(dto_license);
        if (res == -1)
        {
            /// TODO: change log
            std::cerr << "Error SaveLicenseToStorage";
```

```
            return false;
        }

        bf::save_string_file(bf::path("license.json"),
dl::SerializeToJson(license));
    }
    catch (const std::exception &e)
    {
        /// TODO: change log
        std::cerr << "Error, SaveLicenseToStorage: " << e.what() << '\n';
        return false;
    }
    return true;
}

bool GrpcService::GetLicenseFilesContentFromStorage(const dl::License
&license,
                                                    std::string
*license_bin,
                                                    std::string
*license_sig)
{
    try
    {
        /// TODO: Change path with specifying dir
        auto bin =
bf::path(license.license_name()).replace_extension(".bin");
        auto sig =
bf::path(license.license_name()).replace_extension(".sig");

        /// TODO: change log
        std::cout << "\n----------------------------\n";
        std::cout << "Trying to load license files, names: \n\t" <<
bin.string()
                  << "\n\t" << sig.string() << std::endl;
        std::cout << "----------------------------\n";

        bf::load_string_file(bin, *license_bin);
        bf::load_string_file(sig, *license_sig);
    }
    catch (const std::ios::failure &e)
    {
        /// TODO: change log
        std::cerr << "No license files found: " << e.what() << '\n';
        return false;
    }
    return true;
}

bool GrpcService::SaveLicenseFilesContentToStorage(const dl::License
&license,
                                                   const std::string
&license_bin,
                                                   const std::string
&license_sig)
{
    try
    {
        /// TODO: Change path with specifying dir
        auto bin =
bf::path(license.license_name()).replace_extension(".bin");
        auto sig =
bf::path(license.license_name()).replace_extension(".sig");
```

```
            /// TODO: change log
            std::cout << "\n-----------------------------\n";
            std::cout << "Save license files, names: \n\t" << bin.string() <<
"\n\t"
                      << sig.string() << std::endl;
            std::cout << "-----------------------------\n";

            bf::save_string_file(bin, license_bin);
            bf::save_string_file(sig, license_sig);
        }
        catch (const std::exception &e)
        {
            /// TODO: change log
            std::cerr << "Error SaveLicenseFilesContentToStorage: " <<
e.what() << '\n';
            return false;
        }
        return true;
    }

    std::string GrpcService::GetExpirationData()
    {
        return "2024-12-31";
    }

    int GrpcService::GetLicensedBandwidth()
    {
        return 10240;
    }

    inline bool GrpcService::IsValidHardwareDetails(const Hardware
*harwareDetails)
    {
        if (harwareDetails->cpu_number() == 0 || harwareDetails-
>cpu_model().empty() ||
            harwareDetails->memory_size() == 0 || harwareDetails-
>ipv4_address().empty())
            return false;
        else
            return true;
    }

    bool GrpcService::Encrypt(const std::string &key_fname,
                              const std::string &license_json, std::string
*license_bin)
    {
        std::string key;
        try
        {
            bf::load_string_file(bf::path(key_fname), key);
        }
        catch (const std::exception &e)
        {
            /// TODO: change log
            std::cerr << "Error Encrypt: " << e.what() << '\n';
            return false;
        }
        unsigned char buff[DEF_ENCRYPT_DATA_SIZE] = {0};

        alfa::openssl_init();

        int length = alfa::crypto::rsa_private_encrypt(
```

```
            reinterpret_cast<const unsigned char *>(license_json.c_str()),
            license_json.length(), key.c_str(), buff);

        alfa::openssl_done();

        if (length != DEF_ENCRYPT_DATA_SIZE)
        {
            /// TODO: change log
            std::cerr << "Error Encrypt: incorect encrypt size" << '\n';
            return false;
        }

        license_bin->assign(reinterpret_cast<char *>(buff),
DEF_ENCRYPT_DATA_SIZE);
        return true;
    }

    bool GrpcService::Sign(const std::string &key_fname, const std::string
&license_bin,
                           std::string *license_sig)
    {
        alfa::openssl_init();

        EVP_PKEY *pkey =
alfa::crypto::rsa_load_private_key(key_fname.c_str());
        if (NULL == pkey)
        {
            std::cerr << "Error Sign, could not load private key " <<
key_fname.c_str()
                      << std::endl;
            return false;
        }

        size_t buff_len = DEF_SIGNATURE_SIZE;
        unsigned char buff[DEF_SIGNATURE_SIZE] = {0};

        // Create digital signature
        int exit_code = alfa::crypto::rsa_private_sign(
            pkey, reinterpret_cast<const unsigned char
*>(license_bin.c_str()),
            license_bin.length(), buff, &buff_len);

        // Free the EVP_PKEY structure we don't need anymore
        EVP_PKEY_free(pkey);

        alfa::openssl_done();

        if (exit_code != 1)
        {
            std::cerr << "Error Sign, could not sign" << std::endl;
            return false;
        }
        if (DEF_SIGNATURE_SIZE != buff_len)
        {
            std::cerr << "Error Sign, incorect signature size" << std::endl;
            return false;
        }

        license_sig->assign(reinterpret_cast<char *>(buff), buff_len);

        return true;
    }
```

```
    std::string GetEnv(const char *var)
    {
        const char *val = std::getenv(var);
        if (val == nullptr)
        {
            return "";
        }
        else
        {
            return val;
        }
    }

    std::string GrpcService::GetKeyPath(const char *fileName)
    {
        std::string env_var = GetEnv(kEnvironmentVariable);
        std::stringstream keyPath;
        if (PathToKey.length() > 0)
        {
            keyPath << PathToKey;
            if (PathToKey.at(PathToKey.length() - 1) != '/')
                keyPath << "/";
        }
        else if (env_var == "")
        {
            keyPath << "../../keys/";
        }
        else
        {
            keyPath << env_var << "keys/";
        }
        keyPath << fileName;
        return keyPath.str();
    }

    std::string GrpcService::GetFileContentAsString(const char *fileName)
    {
        std::string decryptPath = GetKeyPath(fileName);
        std::ifstream input(decryptPath, std::ios::binary);

        std::vector<char> bytes((std::istreambuf_iterator<char>(input)),
                                (std::istreambuf_iterator<char>()));

        input.close();
        return std::string(bytes.begin(), bytes.end());
    }

    void GrpcService::SetPathToKey(std::string pathToKey)
    {
        PathToKey = pathToKey;
    }
} // namespace grpcService
```

Файл data_layer.hpp

```
#ifndef DATA_LAYER_HPP
#define DATA_LAYER_HPP

#include "license.pb.h"
#include <boost/property_tree/json_parser.hpp>
#include <boost/property_tree/ptree.hpp>
```

```
namespace data_layer
{
    class License
    {
      public:
        const basecamp::Hardware &hardware_info() const;
        void hardware_info(const basecamp::Hardware &hardware_info);

        basecamp::LicenseStatus status() const;
        void status(basecamp::LicenseStatus status);

        int licensed_bandwidth() const;
        void licensed_bandwidth(int licensed_bandwidth);

        std::string description() const;
        void description(const std::string &description);

        std::string expiration_date() const;
        void expiration_date(const std::string &expiration_date);

        std::string license_name() const;
        void license_name(const std::string &license_name);

      private:
        std::string m_license_name;
        basecamp::Hardware m_hardware_info;
        basecamp::LicenseStatus m_status;
        int m_licensed_bandwidth;
        std::string m_description;
        std::string m_expiration_date;
    };

    std::string SerializeToJson(const data_layer::License &license, bool
pretty = true);

    std::string GetHash(const data_layer::License &license);
    std::string GetHash(const basecamp::Hardware &hardware_info);
};     // namespace data_layer
#endif // DATA_LAYER_HPP
```

Файл data_layer.cpp

```
#include "data_layer.hpp"
#include "alfa.h"
#include <sstream>

namespace data_layer
{
    const basecamp::Hardware &License::hardware_info() const
    {
        return m_hardware_info;
    }

    void License::hardware_info(const basecamp::Hardware &hardware_info)
    {
        m_hardware_info = hardware_info;
    }

    basecamp::LicenseStatus License::status() const
    {
        return m_status;
    }
```

```
void License::status(basecamp::LicenseStatus status)
{
    m_status = status;
}

int License::licensed_bandwidth() const
{
    return m_licensed_bandwidth;
}

void License::licensed_bandwidth(int licensed_bandwidth)
{
    m_licensed_bandwidth = licensed_bandwidth;
}

std::string License::description() const
{
    return m_description;
}

void License::description(const std::string &description)
{
    m_description = description;
}

std::string License::expiration_date() const
{
    return m_expiration_date;
}

void License::expiration_date(const std::string &expiration_date)
{
    m_expiration_date = expiration_date;
}

void License::license_name(const std::string &license_name)
{
    m_license_name = license_name;
}

std::string License::license_name() const
{
    return m_license_name;
}

std::string SerializeToJson(const License &license, bool pretty)
{
    using boost::property_tree::basic_ptree;
    using boost::property_tree::ptree;

    ptree root;

    auto hw = license.hardware_info();

    root.put("address_ipv4", hw.ipv4_address());
    root.put("total_memory_size", hw.memory_size());
    root.put("logical_cpus_number", hw.cpu_number());
    root.put("cpu_model", hw.cpu_model());
    root.put("expiration_date", license.expiration_date());
    root.put("licensed_bandwidth", license.licensed_bandwidth());

    std::ostringstream buf;
```

```
        write_json(buf, root, true);

        return buf.str();
    }

    std::string GetHash(const data_layer::License &license)
    {
        using namespace alfa::hash;

        auto hw = license.hardware_info();
        EVP_MD_CTX *context;
        int exit_code = hash_init_sha256(context);
        if (exit_code != 1)
        {
            std::__throw_ios_failure("GetHash, failed hash init");
        }

        exit_code = hash_update(context, hw.ipv4_address());
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
ipv4_address");
        }

        exit_code = hash_update(context,
static_cast<size_t>(hw.memory_size()));
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
memory_size");
        }

        exit_code = hash_update(context,
static_cast<size_t>(hw.cpu_number()));
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
cpu_number");
        }

        exit_code = hash_update(context, hw.cpu_model());
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
cpu_model");
        }

        exit_code = hash_update(context,
static_cast<size_t>(license.licensed_bandwidth()));
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error(
                "GetHash, failed hash update with licensed_bandwidth");
        }

        /// TODO: maybe replace string with date
        exit_code = hash_update(context, license.expiration_date());
        if (exit_code != 1)
```

```
        {
            hash_done(context);
            std::__throw_runtime_error(
                "GetHash, failed hash update with expiration_date");
        }

        std::string hash;
        exit_code = hash_final(context, &hash);
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash final");
        }

        hash_done(context);
        return hash;
    }

    std::string GetHash(const basecamp::Hardware &hardware_info)
    {
        using namespace alfa::hash;
        EVP_MD_CTX *context;
        int exit_code = hash_init_sha256(context);
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_ios_failure("GetHash, failed hash init");
        }

        exit_code = hash_update(context, hardware_info.ipv4_address());
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
ipv4_address");
        }

        exit_code = hash_update(context,
static_cast<size_t>(hardware_info.memory_size()));
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
memory_size");
        }

        exit_code = hash_update(context,
static_cast<size_t>(hardware_info.cpu_number()));
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
cpu_number");
        }

        exit_code = hash_update(context, hardware_info.cpu_model());
        if (exit_code != 1)
        {
            hash_done(context);
            std::__throw_runtime_error("GetHash, failed hash update with
cpu_model");
        }
```

```
            std::string hash;
            exit_code = hash_final(context, &hash);
            if (exit_code != 1)
            {
                hash_done(context);
                std::__throw_runtime_error("GetHash, failed hash final");
            }

            hash_done(context);
            return hash;
        }
    } // namespace data_layer
```