Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи забезпечення інформаційного захисту
IoT-системи на базі технології Intel

(тема)

Виконав:

студент II курсу, групи КСМм-21-1

Скорик В.А.
(прізвище, ініціали)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Піскарьов О.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Скорику Вадиму Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи забезпечення інформаційного захисту IoT-системи на базі технології Intel

затверджена наказом по університету від "07" листопада 2022 р. № 1453Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2022 р.

3. Вхідні дані до роботи

1) аналіз методів захисту IoT-систем;
2) розробка моделі захисту системи;
3) створення демонстраційної IoT-системи.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз сучасних практик захисту IoT-систем;
2) дослідження методів забезпечення всебічного захисту системи;
3) побудова моделі комплексної безпеки IoT-системи;
4) розробка демонстраційної системи на основі спроектованої системи безпеки;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасних практик захисту IoT-систем	08.11.22-13.11.22	
2	Огляд аспектів безпеки в IoT-системах	14.11.22-18.11.22	
3	Побудова моделі комплексного захисту	19.11.22-22.11.22	
4	Розробка демонстраційної системи	23.11.22-25.11.22	
5	Оформлення матеріалів кваліфікаційної роботи	26.11.22-04.12.22	
6	Подання роботи керівникові та її попередній	05.12.22-08.12.22	
	захист		
7	Подання роботи на рецензування	09.12.22-11.12.22	

Дата видачі завдання 07 листопада 2022 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Піскарьов О.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 82 с., 39 рис., 2 дод., 14 джерел.

IOT, ІНТЕРНЕТ РЕЧЕЙ, IOT-СИСТЕМА, ІНФОРМАЦІЙНА БЕЗПЕКА, МОДЕЛЬ БЕЗПЕКИ, ПЕРИФЕРІЙНИЙ ПРИСТРІЙ, ХМАРНИЙ СЕРВІС, ШИФРУВАННЯ.

Метою кваліфікаційної роботи є розробка методів та засобів забезпечення інформаційного захисту IoT-систем.

У ході виконання кваліфікаційної роботи для досягнення мети був проведений всебічний аналіз актуальних методів та кращих практик забезпечення захисту IoT-систем на кожному рівні, проведені дослідження щодо побудови комплексної моделі захисту системи на базі розроблених методів протидії різноманітним загрозам та атакам на усіх рівнях. Була розроблена демонстраційна IoT-система на базі побудованої моделі безпеки, яка підтвердила її ефективність та надійність на практиці.

ABSTRACT

Master’s thesis: 82 pages, 39 figures, 2 appendices, 14 sources.

IOT, INTERNET OF THINGS, IOT-SYSTEM, INFORMATION SECURITY, SECURITY MODEL, PERIPHERAL DEVICE, CLOUD SERVICE, ENCRYPTION.

The major goal of this thesis is the development of methods and tools of ensuring the information protection for IoT-systems.

In order to achieve the goal, a comprehensive analysis of current methods and best practices for ensuring protection of IoT-systems at each layer was done, research of building an integrated system protection model based on the developed methods to counter various threats and attacks at all layers was completed. A demonstration IoT-system was developed based on the constructed security model, which confirmed its effectiveness and reliability in practice.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

Фаєрвол – брандмауер

IoT – интернет речей (англ., Internet of Things)

API – програмний інтерфейс застосунку (англ., Application Programming Interface)

Backend – серверна частина

HTML – мова гіпертекстової розмітки (англ., HyperText Markup Language)

IaaS – инфраструктура як послуга (англ., Infrastructure-as-a-Service)

IDE – інтегроване середовище розробки (англ., Integrated development environment)

PaaS – платформа як послуга (англ., Platform as a Service)

SaaS – сервіс як послуга (англ., Software as a Service)

ВСТУП

Необхідність забезпечення якісної інформаційної безпеки будь якої мережі або системи важно переоцінити у наш час, вона виступає однією з найголовніших вимог до будь-якої інформаційної системи. Причиною цього є наявність нерозривного зв'язку між інформаційними технологіями та основними бізнес-процесами в усіх організаціях, державній службі, промислових підприємствах, фінансових структурах, операторах телекомунікацій тощо.

Забезпечення внутрішньої інформаційної безпеки – актуальна тема і одночасно величезна проблема. Ніхто поки не вирішив її в загальному випадку, не існує універсальних алгоритмів рішення, які б підійшли до усіх можливих випадків, але існує велика кількість різноманітних методів та рішень, які здатні вирішити окремі випадки цього завдання.

У цій роботі буде розглянуто застосування різних актуальних методів та алгоритмів забезпечення інформаційної безпеки на різних рівнях для вирішення задачі забезпечення захисту комп’ютерної IoT-системи на базі архітектури Intel, яку буде розроблено як демонстраційний приклад.

1 СЬОГОДНІШНІЙ СТАН ІОТ ТА ЙОГО ПРОБЛЕМИ

1.1 Дослідження концепції IoT

Концепція IoT (англ. Internet of Things – Інтернет “Речей”) полягає у створенні обчислювальної мережі фізичних об’єктів (тих самих “речей”, з назви концепції), побудованих на використанні технології взаємодії один з одним або із зовнішнім середовищем. Ця концепція розглядає організацію таких мереж як явище, здатне у майбутньому повністю перебудувати більшість суспільних та економічних процесів та звести до мінімуму необхідність у них людської участі.

Підбиваючи підсумки вищесказаного, можемо виділити три пукти філософії, на яких базується вся концепція IoT (рисунок 1.1):

- автоматизація, або забезпечення автономного та надійного виконання тих самих процесів, але вже без участі людини;
- віртуально-фізичний інтерфейс, за допомогою якого відбувається взаємодія між зовнішнім середовищем та програмно-апаратною частиною системи. Отримання необроблених даних з фізичних сенсорів, їх оцифрування за допомогою АЦП (англ. ADC, Analog to Digital Converter - Аналогово-Цифровий перетворювач), та передача оцифрованих далі по ланцюгу;
- аналіз даних та прийняття рішень: зібрані з сенсорів та оцифровані дані потрапляють до хмари, де проводиться їхній аналіз у реальному часі, та в залежності від конкретної IoT-системи, приймаються ті чи інші рішення, здійснюється керування пристроями тощо.

Рішення IoT дають змогу як кінцевим користувачам, так і постачальникам продукції отримувати інформацію в реальному часі та автоматизувати виконання завдань.

Рисунок 1.1 – Загальна архітектурна модель IoT

Перспективними сферами застосування IoT [1] є:

- домашня автоматизація;
- транспортна автоматизація;
- сфера охорони здоров'я;
- оптимізація виробництва;
- економія ресурсів;
- штучний інтелект;
- реагування на надзвичайні події.

1.2 Аналіз сучасних IoT-рішень та кращих практик

У наш час існує досить багато різноманітних платформ для розгортання хмарних сервісів IoT-систем, що вже мають добре-спроектовані та оптимізовані моделі комплексного захисту [2].

Різні платформи можуть бути більш чи менш універсальними, пропонувати функції для розробників (IoT AEP платформи) або, безпосередньо, для кінцевого користувача (IoTM платформи), більш або менш підходити для окремих прикладних сфер застосування тощо [3].

На рисунку 1.2 наведені результати аналізу ринку хмарних платформ за 2016-2018 роки. Проаналізуємо найбільш популярні та актуальні на даний момент хмарні платформи, з цього списку.

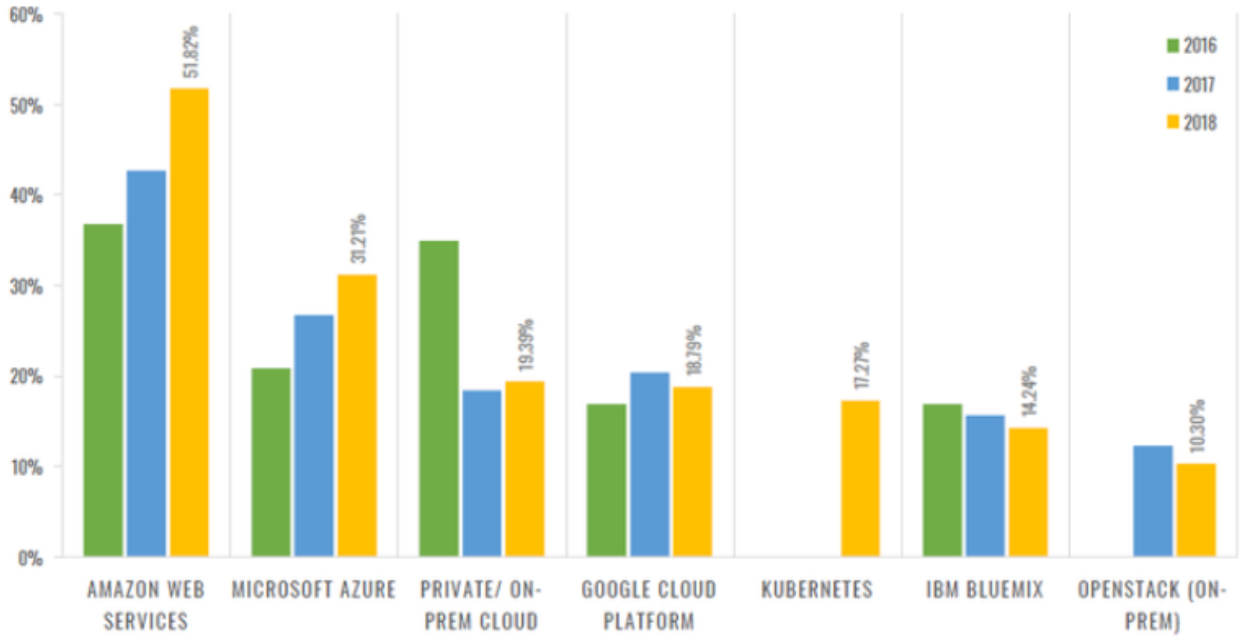

Рисунок 1.2 – Тренди IoT-платформ

1.2.1 Amazon Web Services

Amazon Web Services – найбільший та найпопулярніший постачальник хмарних послуг від компанії Amazon [4]. Платформа AWS IoT Core пропонує гнучкі, надійні, масштабовані та прості у використанні рішення для IoT. AWS є комплексною, простою у використанні обчислювальною платформою розробленою на основі поєднання інфраструктури як послуги (IaaS), платформи як послуги (PaaS) та пакетного програмного забезпечення як послуги (SaaS).

AWS IoT Core дозволяє просто та безпечно підключати фізичні пристрої до хмари, надійно їх масштабувати та забезпечує повну взаємодію між пристроями IoT з їх периферійним програмним забезпеченням та хмарними сервісами AWS (рисунок 1.3).

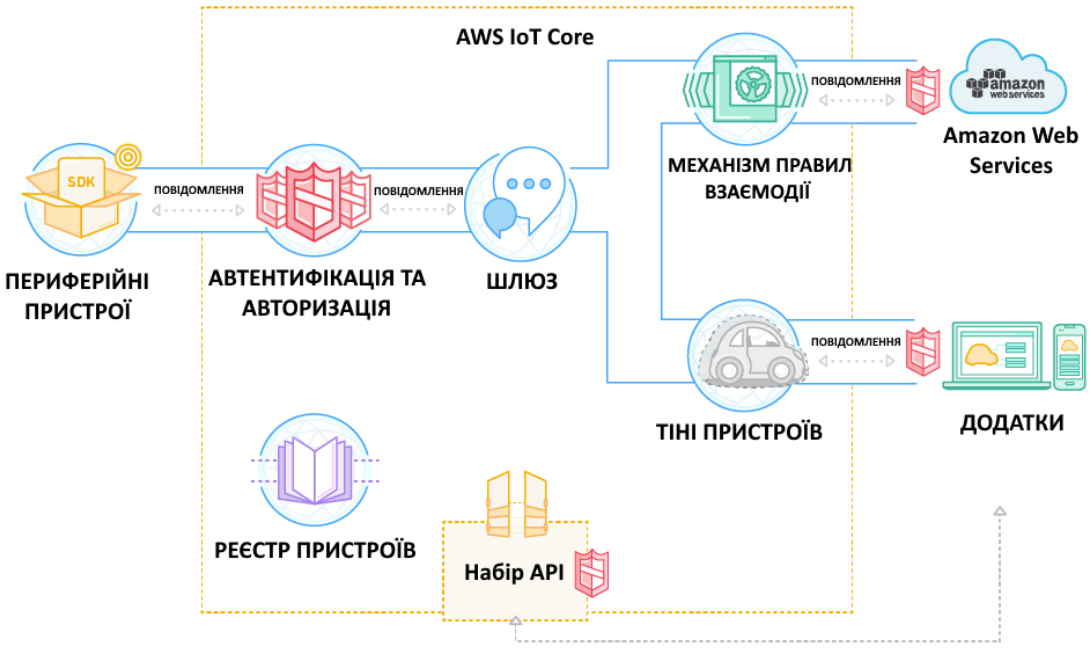

Рисунок 1.3 – Схема взаємодії AWS з периферійними пристроями за допомогою AWS IoT Core

Центральне місце AWS IoT Core – це шлюз, який забезпечує взаємодію пристроїв із платформою хмари. Фактично це брокер повідомлень, працюючий на базі протоколу MQTT, який, з одного боку, забезпечує безпечне підключення пристроїв з використанням механізму автентифікації та авторизації, а з іншого – дозволяє використовувати весь потенціал AWS-рішень та сервісів. Шлюз також підтримує протоколи WebSocket, HTTP 1.1. В разі втрати з'єднання з віддаленим пристроєм є незвичайне рішення – це тіні пристроїв, які являють собою деяку абстракцію або віртуальне уявлення останнього стану пристрою, який став недоступним. Також для “тіні” можна задати бажаний майбутній стан. Завдяки такому підходу можна гнучко проектувати свої програми для роботи з віддаленими периферійними пристроями в умовах нестійкого зв'язку (рисунок 1.4).

Рисунок 1.4 – Модель організації безпеки AWS IoT Core

Розглянемо платформу з точки зору безпеки. Платформа AWS IoT Core забезпечує взаємну автентифікацію та шифрування у всіх точках підключення. Таким чином, будь-який обмін даними між пристроями та платформою відбувається лише після підтвердження ідентифікації. Сервіс підтримує метод аутентифікації AWS (званий SigV4), автентифікацію на основі сертифіката

X.509 і спеціальну автентифікацію на основі токенів (через модулі авторизації). Підключення за протоколом HTTP можуть використовувати будь-який із цих методів. Підключення протоколу MQTT використовують автентифікацію на основі сертифікатів. Підключення по протоколу WebSocket можуть використовувати метод SigV4 і модулі авторизації. AWS IoT Core дозволяє використовувати сертифікати, сформовані самим сервісом, а також сертифікати, підписані вибраним центром сертифікації. Можна прив'язати до кожного сертифіката вибрані політики для надання авторизованого доступу пристроям або програмам з можливістю відкликання дозволу без необхідності безпосередньо працювати з пристроєм.

Сертифікати та політики пристроїв створюються та керуються з консолі або за допомогою API. Ці сертифікати пристроїв можна надавати, активувати та зв'язувати з відповідними політиками IoT, налаштованими за допомогою AWS IoT Core. Така схема дозволяє при необхідності відразу відкликати дозвіл доступу для конкретного пристрою.

Платформа також підтримує підключення з боку мобільних програм користувачів за допомогою сервісу Amazon Cognito, який повністю забезпечує створення унікальних ідентифікаторів для користувачів ваших програм та отримання тимчасових даних доступу AWS з обмеженими правами. Сервіс також може надавати тимчасові дані для доступу AWS після того, як пристрій автентифікувався за допомогою сертифіката X.509.

Після детального розглядання AWS IoT Core можна зробити висновок, що саме ця платформа займає перше місце по популярності за свою зручну та надійну систему організації безпеки та дуже гнучкі та глибокі можливості розміщення IoT-рішень.

1.2.2 Microsoft Azure

Microsoft Azure – ще один постачальник хмарних послуг, що на даний момент займає друге місце на ринку. Для IoT-рішень компанія Microsoft

пропонує платформу Azure IoT, яка складається з двох основних сервісів – Azure IoT Central та Azure IoT Edge для підтримки та масштабування IoT-проектів [5].

Azure IoT Central – платформа додатків як послуг (PaaS), яка надає хмарне середовище та обчислювальні потужності для створення, управління та розміщення IoT-рішень (рисунок 1.5), ключова перевага якої порівняно з іншими платформами – мінімізація витрат та зусиль з боку клієнта. Мета PaaS-пропозицій полягає у прискоренні розробки додатків, надаючи повне рішення для конкретних потреб бізнесу. Як підкатегорія типу хмарної служби "платформа як послуга" (PaaS), PaaS-пропозиції забезпечують масштабування та налаштування за клієнта, але все ж таки вимагають участі розробника для завершення рішення. Використовуючи платформу Azure IoT Central, клієнт може почати створювати рішення IoT Central прямо у своєму інтернет-браузері.

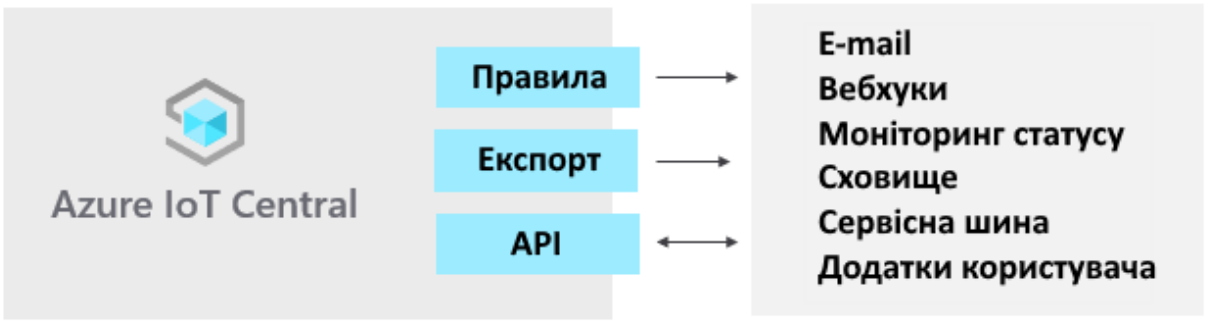

Рисунок 1.5 – Представлення роботи сервісу Azure IoT Central

Перейдемо до розглядання платформи з точки зору безпеки. Комплексну модель захисту платформи Azure IoT (рисунок 1.6) можна розділити на три області безпеки:

- безпека пристрою під час його розгортання на практиці;
- безпека підключення – забезпечення конфіденційності та захисту від несанкціонованої зміни всіх даних, що передаються між периферійним пристроєм IoT та Azure IoT Central;

- безпека у хмарі – надання засобів для захисту даних, що переміщуються та зберігаються у хмарі.

Рисунок 1.6 – Модель багатошарового захисту Azure IoT

Рішення Azure IoT захищають пристрої за допомогою наступних двох методів:

- надаючи унікальний ключ посвідчення (маркери безпеки) для кожного пристрою, який може використовуватися для взаємодії з Центром Інтернету речей;

- використовуючи сертифікат X.509 та закритий ключ на пристрої для автентифікації пристрою в Azure IoT Central. Цей метод автентифікації гарантує, що закритий ключ на пристрої ніколи не буде відомий поза цим пристроєм, що дозволяє підвищити рівень безпеки.

Метод маркерів безпеки забезпечує автентифікацію кожного виклику Azure IoT Central, надісланого пристроєм, за рахунок прив'язування симетричного ключа до кожного виклику. Застосування сертифікатів X.509 дозволяє автентифікувати пристрої IoT фізично при встановленні підключення TLS. Метод на основі маркерів безпеки можна використовувати без автентифікації на основі сертифікатів X.509, що є менш захищеною

схемою. Вибір між цими двома методами в основному визначається вимогами щодо безпеки автентифікації пристроїв та наявністю захищеного сховища на пристрої (для безпечного зберігання закритого ключа). Сервіс використовує маркери безпеки для автентифікації пристроїв та служб, щоб уникнути надсилання ключів через мережу. Крім того, маркери безпеки обмежені за часом та сферою дії. Пакети SDK для платформи автоматично створюють маркери без спеціального налаштування. Однак у деяких випадках потрібно, щоб користувач безпосередньо створив та використовував маркери. До цих сценаріїв входить безпосереднє використання протоколів MQTT, AMQP або HTTP або реалізація схеми служби маркерів.

Підбиваючи підсумки аналізу платформи Microsoft Azure, можна виділити такі її сильні сторон и порівняно з іншими рішеннями:

- відносно невисока ціна на оренду обчислювальних ресурсів та розгортання рішення;
- простіша організація розгортання та масштабування рішень з боку клієнта;
- зручна організація безпеки рішення IoT.

Таким чином, ця платформа підходить для не дуже складних та масштабних проектів, а також у тих випадках, коли клієнт бажає мінімізувати витрати за рахунок функціоналу або багатого вибору доступних сервісів платформи.

1.2.3 Google Cloud IoT

Хмарна платформа від компанії Google, яка на даний момент займає третє місце у рейтингу IoT-платформ (рисунок 1.7). Google Cloud пропонує платформу-сервіс Cloud IoT, яка складається з двох частин – Cloud IoT Core та Cloud IoT Edge [6].

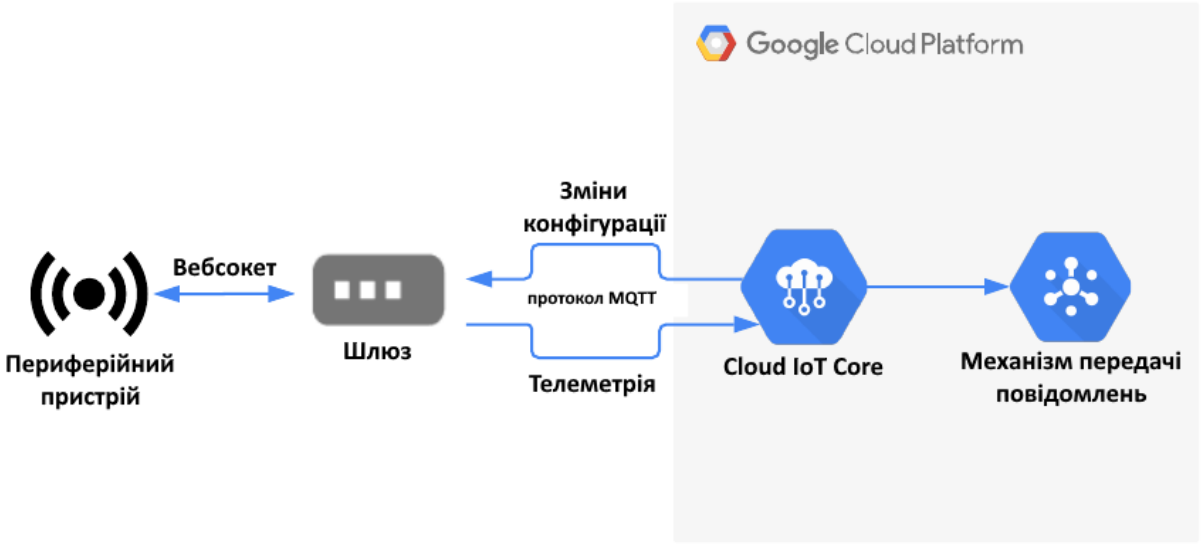

Рисунок 1.7 – Загальна архітектура Cloud IoT Core

Cloud IoT Core (рисунок 1.7) – це керована служба-сервіс, яка дозволяє швидко та безпечно підключатися, налаштовувати та отримувати дані з великої кількості периферійни пристроїв. Використовуючи спеціальний механізм передачі повідомлень Pub/sub (англ. Publish/subscribe – публікація/підписка), ядро може об’єднувати дані з децентралізованих пристроїв в єдину глобальну систему. У поєднанні з іншими сервісами Google, Cloud IoT Core пропонує комплексне рішення для збору, аналізу та візуалізації даних IoT у реальному часі. Це, у свою чергу, дозволяє створювати багатофункціональні моделі.

Google Cloud IoT Core може працювати зі звичними для IoT протоколами HTTP та MQTT, що дозволяє використовувати багато існуючих пристроїв для створення своїх систем IoT. Цей сервіс працює в інфраструктурі Google, яка автоматично масштабується в режимі реального часу.

Cloud IoT Core складається з диспетчера пристроїв, що налаштовує окремі пристрої та здійснює керування ними та протокольного мосту, що з’єднує кінцеві точки компонентів та забезпечує балансування навантаження. Його основні функції – вбудована підтримка HTTP та MQTT та збереження телеметрії у механізм передачі повідомлень.

Друга частина платформи – Cloud IoT Edge використовується для обробки зібраних даних та використання технологій і алгоритмів машинного навчання на периферійних пристроях, завдяки чому пристрої можуть працювати в режимі реального часу, використовуючи дані своїх датчиків, і локально прогнозувати результати. Сервіс Edge сумісний з ОС на базі Linux та складається з двох компонентів середовища виконання (Connect та ML), апаратного прискорювача TPU та мікросхеми ASIC.

На рисунку 1.8 наведена діаграма процесу надання облікових даних пристрою. Автентифікований постачальник, який найчастіше є користувачем, створює проект, налаштовує пристрої, ролі та дозволи – все це робиться за допомогою команд API або використовуючи консоль Cloud Platform Console. Передача інформації на усіх етапах захищена протоколом TLS 1.2.

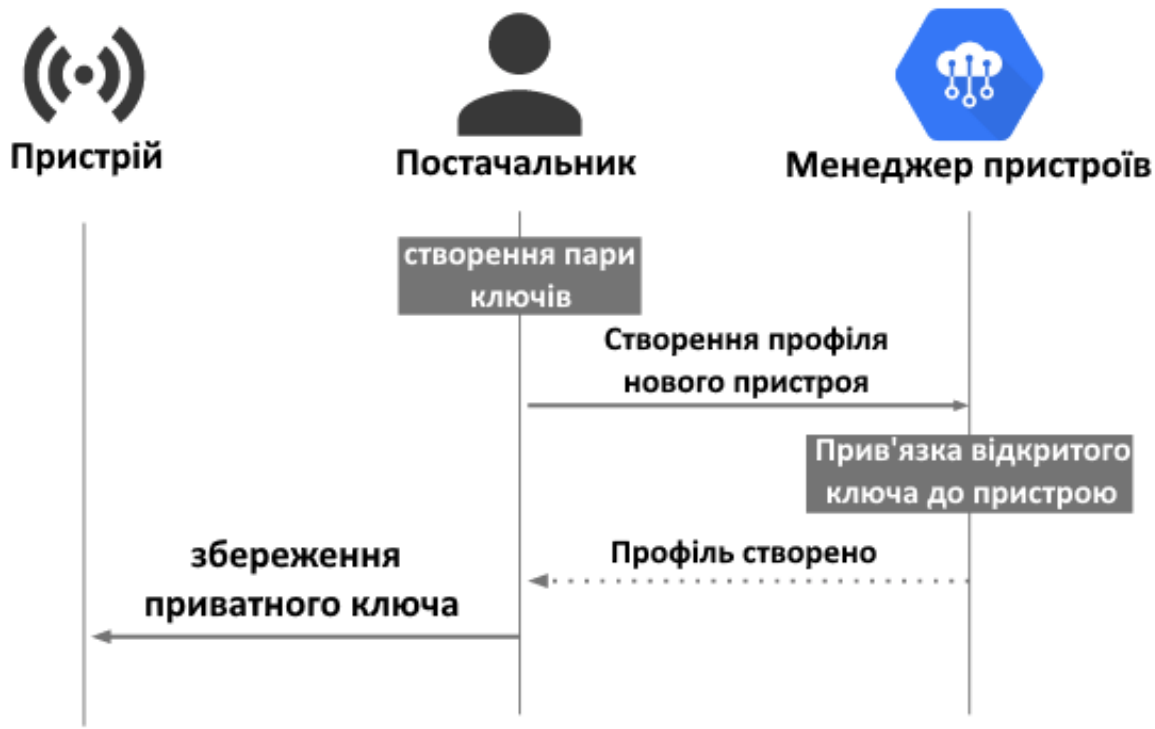

Рисунок 1.8 – Модель захисту пристроїв у Cloud IoT Core

Доступ до API контролюється ролями та дозволами IAM (англ. Identity and Access Management – керування ідентифікацією та доступом).

З точки зору забезпечення безпеки, платформа Google Cloud IoT пропонує цілу низку шарів безпеки на різних рівнях. Захист периферійних пристроїв реалізований через автентифікацію за допомогою пар ключів “приватний-відкритий”, реалізовану на базі JSON Web Tokens, які є валідними лише деякий час, після чого вони заново генеруються та обновлюються, що попереджує подальше використання ключів у разі їхнього перехвату.

1.2.4 Підсумки аналізу сучасних практик IoT

У процесі огляду існуючих хмарних платформ, що пропонують сервіси для розгортання та побудови IoT-рішень та аналізу їхніх моделей організації безпеки та забезпечення захисту систем, було встановлено три основні моделі надання послуг: інфраструктура як послуга (IaaS); платформа як послуга (PaaS); програмне забезпечення як послуга (SaaS).

Найбільш актуальним та популярним рішенням для більшості проектів на даний момент є модель PaaS, бо вона пропонує найбільший функціонал та дуже гнучкі можливості розгортання та масштабування.

З точки зору аналізу систем безпеки розглянутих платформ, були виявлені наступні спільні практики:

- віддання переваги протоколам MQTT та HTTP для обміну даними між елементами IoT-системи;
- захист каналів зв'язку за допомогою так званого протоколу “рукостискання” на базі протоколу TSL 1.2 з подальшим шифруванням інформації;
- реєстрація периферійних пристроїв у системі з подальшою автентифікацією за допомогою пар ключів та сертифікатів доступу стандарту X.509;
- забезпечення надійного захисту шлюзу від потенційних атак різного характеру з використанням різноманітних модулів аналізу трафіку, у тому числі модулів на базі штучного інтелекту.

Серед розглянутих платформ були помічені наступні загальні недоліки:

- порівняно висока ціна на оренду ресурсів;
- занадто комплексний підхід до розгортання, який може бути неоптимальним для проектів та рішень малого масштабу;
- обмеження швидкості обміну даними між пристроями та хмарою, що в деяких випадках може бути критичним.

1.3 Постановка задачі

Ціллю роботи є розробка методів та засобів забезпечення інформаційного захисту IoT-систем. Для демонстрації роботи спроектованої моделі безпеки потрібно розробити демонстраційну IoT-систему, що складається з периферійного пристрою на базі Intel, хмарного сервісу для обчислень, зберігання інформації та управління пристроями, та клієнтського додатку для моніторингу стану системи, забезпечити належний захист цієї системи на усіх рівнях: захист пристрою; захист інформаційних каналів системи; захист хмарного сервісу та бази даних.

Також розроблена IoT-система повинна відповідати переліку вимог до розробки, сформованому після аналізу недоліків існуючих рішень: безоплатне використання; просте та швидке розгортання; висока швидкість обміну інформацією між елементами системи, стабільний зв’язок.

Було висунуто наступні кроки реалізації:

- розробка програмного забезпечення периферійного пристрою;
- розробка хмарного сервісу та його основного функціоналу;
- створення каналів обміну даними між елементами системи;
- розробка клієнтського додатку для моніторингу стану системи, його інтеграція у систему.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ IOT-СИСТЕМИ

Для побудови IoT-системи та забезпечення її безпеки потрібно послідовно вирішити наступні задачі:

- провести аналіз актуальних архітектур IoT-систем;
- побудувати модель IoT-системи;
- проаналізувати сучасні методи забезпечення безпеки IoT-рішення на різних рівнях та побудувати модель захисту системи.
- обрати засоби програмної та апаратної реалізацій;

Перейдемо до вирішення кожної із задач.

2.1 Аналіз топології типової IoT-системи

Топологія IoT відрізняється від звичайної рівневої моделі, як OSI (англ. Open Systems Interconnection Basic Reference Model - базова еталонна модель взаємодії відкритих систем). Це не лінійний та набагато складніший граф потоків. Деякі компоненти є необов'язковими і можуть бути відсутніми у конкретному класі рішень. Можуть бути два типи логіки - M2M (від машини до машини) і M2P (від машини до людини), а також більш окремі випадки такі як C2C (від автомобіля до автомобіля) тощо [7].

IoT рішення має два фізичні розташування - перше це кінцеві (периферійні) пристрої, а друге - в центрі обробки даних Backend (англ. Back-end – задня частина, серверний кінець) на серверах або в хмарі. У той же час, це не класична архітектура клієнт-серверної програми, як можна буде побачити далі.

На рисунку 2.1 можна побачити усі рівні IoT системи та їх взаємодію між собою. Розглянемо ці рівні детальніше у наступному пункті.

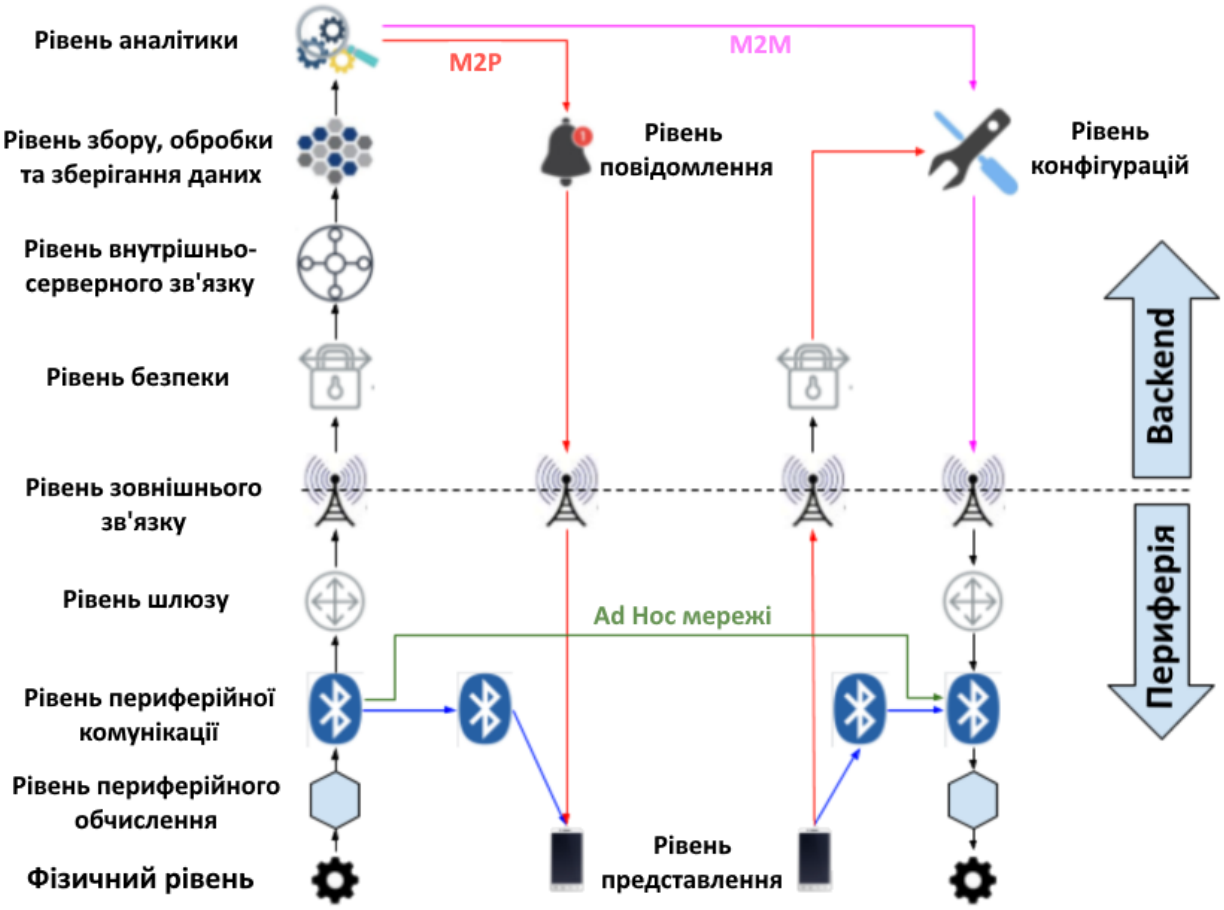

Рисунок 2.1 – Топологічний граф потоків IoT-системи

2.1.1 Аналіз периферійної частини IoT-системи

Почнемо з першого рівня – фізичного. Цей рівень представляє два типу операцій – сбір інформації з датчиків та здійснення механічної роботи (виконавчі механізми). Усі датчики можна розділити на дві категорії:

а) сенсори:

1) світлові: фото-діоди, транзистори, резистори, PIR (інфрачервоні) детектори;

2) звукові: мікрофони, ультразвукові сенсори;

3) вимикачі, зокрема кінцеві вимикачі, що реєструють крайні точки механічного руху. Вимірники куту повороту чи швидкості обертання.

Електромагнітні сенсори, що вимірюють зміну фізичних характеристик, таких як електрична ємність, індуктивність, опір.

б) Складні чи складові сенсори. До них відносяться спеціалізовані датчики, наприклад, газу, спектру та інші, а також окремий вид пристроїв збору інформації, що отримує все більше застосування – відеокамери.

У рішеннях IoT фізичні елементи мають певні загальні вимоги:

- якомога нижчу ціну через велику кількість сенсорів у рішенні IoT;
- живлення від батарейки, що в свою чергу потребує низького енергоспоживання. Сьогоднішній запит ринку – робота периферійних пристроїв без технічного обслуговування від 1 до 10 років, залежно від конкретного типу пристрою;
- можливість розташування у важкодоступних та віддалених місцях з мінімальними витратами на встановлення та обслуговування;
- у разі використання відеокамер, первинна обробка зображення з ухваленням рішення на основі штучного інтелекту.

Сумуючи вищесказане, можна виділити дві проблеми до вимог фізичного рівня:

- низьке енергоспоживання. Потрібний високий рівень інтеграції з верхніми шарами;
- використання відеокамер. Це також вимагає високого ступеня інтеграції з верхніми рівнями та вбудованими функціями штучного інтелекту та машинного навчання, реалізованими у периферійному пристрої.

Перейдемо до другого рівня – рівня периферійного обчислення. Цей рівень зазвичай підключається до одного датчика, або виконавчого механізму. Він забезпечує мінімальну функціональність для перетворення аналогової інформації в цифрову та/або навпаки. Для підключення датчиків існують ті ж самі вимоги щодо ціни та споживаної потужності. Багато виробників, що випускають ці типи пристроїв, не мають єдиного стандарту моделі даних, конфігурації та експлуатації, що створює окремі проблеми інтеграції.

Для зниження енергоспоживання периферійні пристрої зазвичай мають чотири режими роботи:

- режим сну;
- режим вимірювання та збору інформації з датчиків;
- режим зв'язку, передачі та отримання інформації;
- режим встановлення та підключення.

На рисунку 2.2 представлена блок-схема периферійного пристрою.

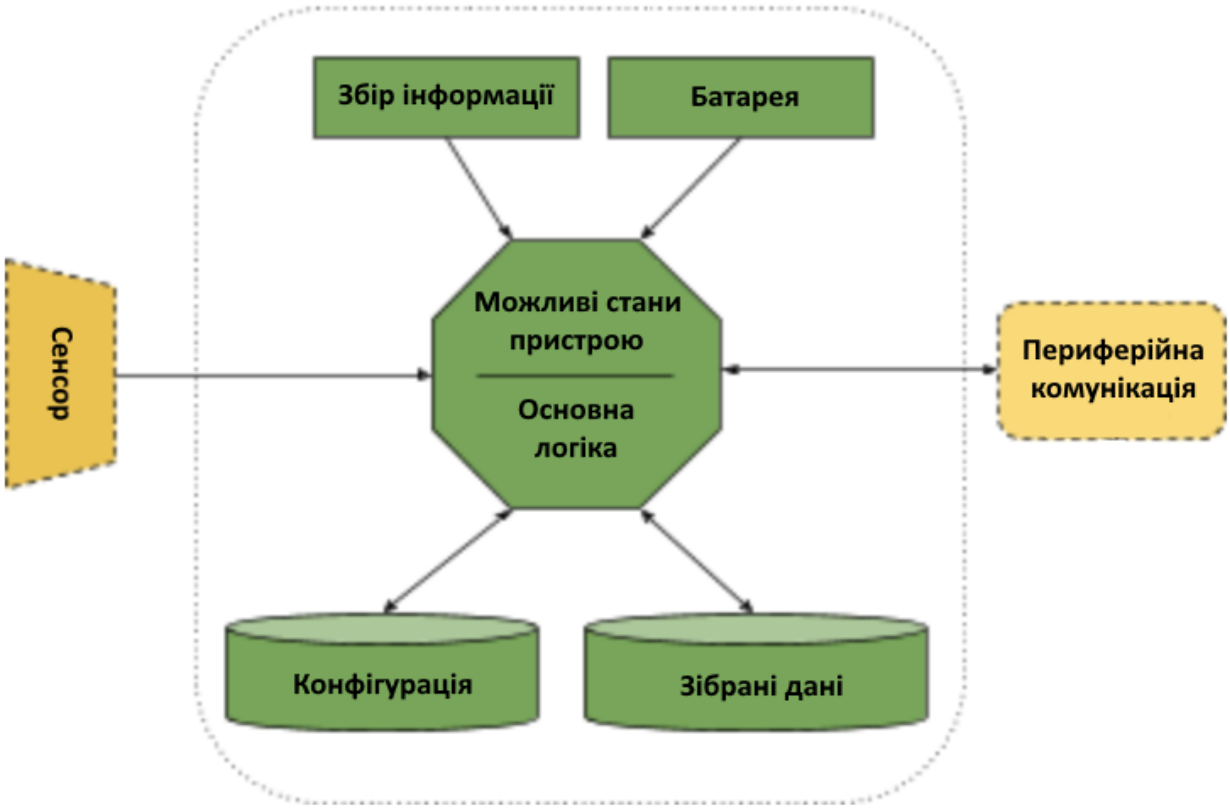

Рисунок 2.2 – Блок-схема периферійного пристрою

Воно зазвичай поєднує три рівня: фізичний, рівень периферійного обчислення та комунікаційний. Основна функціональність рівня периферійного обчислення – локальний процес ETL (англ. Extract, Transform, Load – витяг, перетворення та завантаження) – інформації з датчиків. Цей рівень відповідальний не лише за збирання інформації з датчика, але також і

за приведення її до стандартного вигляду, фільтрацію перешкод, попередній аналіз та локальне збереження.

Отже, основною вимогою рівня периферійного обчислення є низьке енергоспоживання. Це може бути досягнуто за допомогою апаратного забезпечення з низьким енергоспоживанням та алгоритмів енергозбереження Sleep / WakeUp.

Переходимо до наступного рівня – рівня периферійної комунікації. Передача даних є енергоємною частиною периферійного пристрою, так як більшість периферійних пристроїв не підключені до електромережі та дротових засобів зв'язку. Крім того, периферійні пристрої можуть бути розташовані досить далеко від шлюзу (не більше кількох кілометрів). З іншого боку, кількість інформації, що передається, зазвичай досить мала. Наступні протоколи використовуються лише на рівні периферійної комунікації:

- ZigBee / Zwave;
- BLE;
- LoRa;
- Proprietary low band;
- Ad Hoc;
- Wi-Fi;
- Bluetooth;
- LTE;
- Ethernet.

Для цілей конфігурації також можна використовувати протокол NFC. У процесі першої установки та/або технічного обслуговування сервісний інженер з мобільним додатком може підключатися до периферійного пристрою через рівень периферійної комунікації. Іноді Q-код, надрукований на периферійному пристрої, також використовується для автентифікації.

Наступним рівнем, який ми розглянемо, стане рівень шлюзу. У IoT-рішеннях існує кілька причин наявності цього рівня [8]:

- якщо Backend отримуватиме необроблену інформацію, це збільшить його вимоги до потужності і витрати будуть дуже великі;

- робота Backend не може гарантувати реакцію у реальному часі для великої кількісті периферійних пристроїв;

- через обмеження безпеки деяка інформація не може бути надіслана до Backend і не може постійно контролюватися людиною. До такої інформації відносяться дані камер вуличного спостереження, медична інформація та інше.

Шлюз повинен забезпечувати наступний основний функціонал:

- здійснювати другий рівень ETL від своїх периферійних пристроїв;

- фіксувати критичну ситуацію та видавати локальну реакцію, навіть без зв'язку з Backend. Це можна порівняти з сигналами серцебиття людини або дихання легень без участі головного мозку;

- підтримувати комунікацію з Backend. Відправляти на сервер оброблену інформацію з периферійних пристроїв та отримувати конфігураційні дані для периферійних пристроїв;

- зберігати інформацію про статус периферійних пристроїв та зібрані ними дані.

У деяких випадках функціональність AI/ML повинна бути присутня на рівні шлюзу. Шлюзовий пристрій в основному живиться від електромережі або має велику вбудовану батарею, але деякі рішення також потребують низького енергоспоживання. У такій ситуації виникає додаткова проблема - протокол синхронізації для зв'язку з периферійним пристроєм. Один із них (шлюз або периферійний пристрій) повинен передавати повідомлення «Готовий до спілкування» частіше, ніж інший пристрій готовий вийти на зв'язок. Вибір залежатиме від загального споживання енергії кожного пристрою та потрібного часу без обслуговування.

Сьогодні зростає кількість програм із джерелом інформації у вигляді відеокамери. У цих конкретних рішеннях рівень шлюзу та рівень периферійного обчислення можуть бути інтегровані разом. Функціональність

AI/ML у таких додатках стає дуже затребуваною. З новими прискорювачами машинного інтелекту для вбудованих систем це рішення стало реальністю.

2.1.2 Аналіз серверної частини IoT-системи

Розглянемо рівень зовнішнього зв’язку. Він поділяє периферійну і Backend частину архітектури IoT-системи. Шлюз найчастіше підключений до Backend з використанням мобільного бездротового зв'язку, такий як 4G / 5G, але іноді використовується дротовий доступ до Інтернету. Логічний рівень зовнішнього зв'язку має стандартизований протокол рішень IoT, який називається LvM2M. Протокол LvM2M був розроблений для доступу до кожного периферійного пристрою, але оскільки багато постачальників периферійних пристроїв не підтримують інтерфейси LvM2M, шлюзовий пристрій може вирішити цю проблему та створити обгортку для зв'язку з периферійними пристроями.

Рівень зовнішнього зв'язку також містить у собі комунікаційні послуги та моделі ISO. Він включає служби балансування та позиціонування, засновані на DNS сервісі, транспортний протокол COAP, шифрування DTLS, тощо.

Протокол DTLS розглянемо детальніше – це протокол з ключами безпеки та сесією встановлення з'єднання – handshake (англ. Handshake — “рукостискання”). Він працює за схемою точка-точка. Для розшифровування пакетів DTLS ми повинні використовувати той самий екземпляр Backend, який був у нас під час сесії з'єднання. Це створює проблему для балансувальника навантаження який є частиною безпеки на нашій схемі. Балансувальник навантаження, у свою чергу, необхідний для автоматичного масштабування за високого завантаження системи. Щоб уникнути цього обмеження, DNS використовується як балансувальник навантаження. Через деякі проміжки часу DNS-запит отримує нову IP-адресу екземпляра рівня безпеки.

Перейдемо до розглядання рівня безпеки. Цей рівень є одним з найважливіших та забезпечує функції AAA (англ. Authentication, Authorization

and Accounting – автентифікація, авторизація та облік) та шифрування/дешифрування разом з іншими послугами, пов'язаними з Інтернетом. Всі хмарні платформи мають свої власні реалізації безпеки, але функціонально всі вони побудовані на принципі ролей і дозволів. Більш детально це буде розглянуто у наступному пункті. Цей рівень також виконує функції термінатора шифрованого з'єднання DTLS. Підключення кінцевого користувача до Backend також має компонент безпеки.

Розглянемо рівень внутрішньосерверного зв’язку. Цей рівень забезпечує внутрішню Cloud-функціональність балансування навантаження, черги повідомлень та передачі потокової інформації. Компоненти цього рівня мають бути дубльовані та автоматично масштабуватися.

Рівень реалізується переважно на основі мікросервісів або PaaS від Cloud провайдерів. Така вимога випливає з парадигми стрибків та провалів обсягу даних. Автоматичне масштабування знижує вартість Backend реалізації. Фактична реалізація сервісу може бути різною, але загальний принцип залишається одним – забезпечити асинхронну передачу повідомлень із буферизацією та перерозподілом навантаження. Таким чином, різні компоненти Backend можуть виконувати свою роботу незалежно і горизонтально масштабуватися в залежності від навантаження.

На рисунку 2.3 представлена схематична діаграма патернів внутрішньосерверного зв'язку. Балансувальники навантаження призначені для розподілу навантаження між різними сервісами. Черги забезпечують проміжну буферизацію для реалізації асинхронної роботи послідовних сервісів. Отримувачі підписуються на відповідні їх логіці черги, щоб отримувати послідовно повідомлення після обробки попередніх повідомлень.

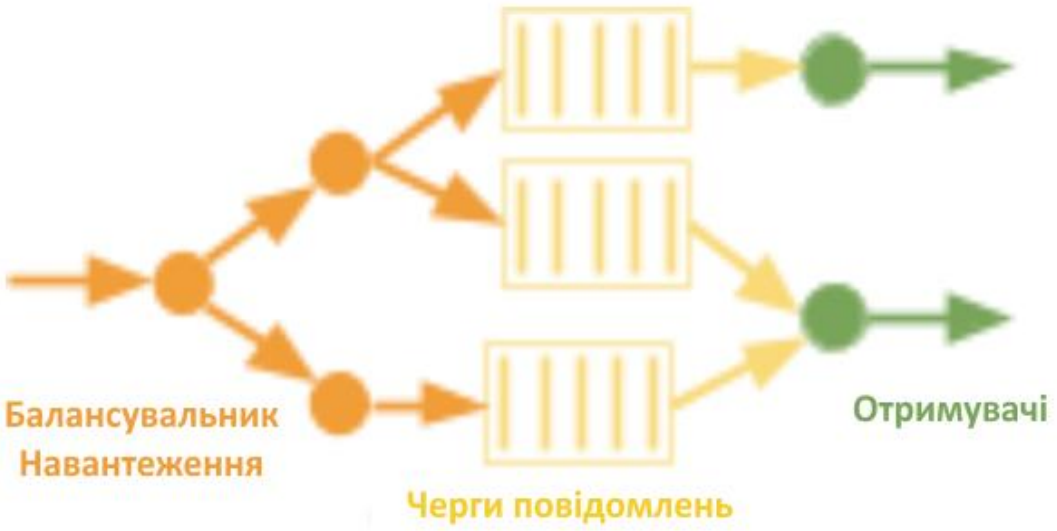

Рисунок 2.3 – Діаграма патернів внутрішньосерверного зв’язку

Перейдемо до рівня збору, обробки та зберігання даних. Цей внутрішній рівень, який також називається ETL (витяг, перетворення та завантаження) є третьою операцією ETL. Перший був у периферійному пристрої, другий – у шлюзі.

Backend ETL накопичує дані з усіх периферійних пристроїв та шлюзів та відповідає за наступні операції:

- збір інформації;
- приведення інформації до стандартного вигляду;
- збереження інформації для подальшого використання;
- управління життєвим циклом інформації, включаючи архівування та знищення;
- повідомлення інших сервісів про надходження нових даних.

Загальна схема реалізації цього рівня представлена на рисунку 2.4. Операція збору даних включає зчитування інформації з релевантних черг. Операція перетворення може виконуватись спеціалізованими сервісами Cloud, такими як Lambda, або обчислювальними засобами усередині контейнерів та просто віртуальними машинами. Кожен із вищеперелічених методів має свої позитивні та негативні властивості.

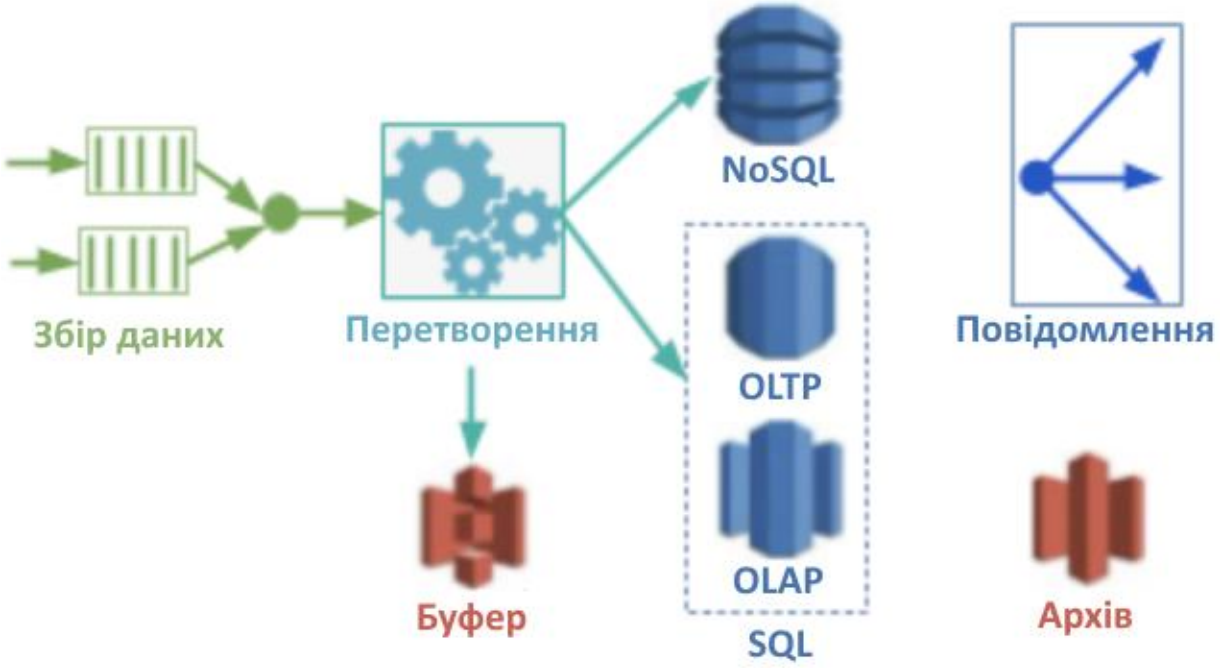

Рисунок 2.4 – Загальна схема рівня ETL

Так наприклад, Lambda сервіс зручний майже повною автоматизацією, але має суттєвий час створення і тому не є кращім вибором, якщо потрібна швидка реакція на якісь події. Також Lambda погано підходить для постійних обробок, оскільки тарифікується за часом використання. Найчастіше застосована служба – контейнеризовані обчислення. Вони зручно масштабуються та легко переносяться на різні Backend. Основне завдання цієї операції – зробити приведення даних до зручного для зберігання, сортування та пошуку виду. Для цього часто дані поєднуються з різних повідомлень і навіть черг.

Операції зберігання призначені для збереження, сортування та подальшого пошуку інформації. Залежно від типу інформації та варіантів її використання застосовуються різні інструменти. Якщо дані не мають чіткої схеми колонок таблиці, всі вони зберігаються в NoSQL базах. Однак, якщо дані можуть бути систематизовані фіксованою схемою, то використовуються SQL типи баз даних. Останні, у свою чергу, мають 2 типи — OLTP (англ. Online Transactional Processing – транзакційна обробка онлайн) і OLAP (англ.

Online Analytic Processing – аналітична обробка онлайн). Як випливає з назви, перший тип найбільш підходить для самого процесу ETL – запису в базу нових значень, тоді як другий зручніше для пошуку та аналізу даних. Тому часто після завантаження в базу OLTP, у фоновому режимі, дані копіюються в OLAP. Зустрічаються ситуації, коли дані не зручно чи неможливо зберігати у базах даних, наприклад вигляді записи, Ці дані записують у буфер, а метадані записів зберігають у базах даних. Для скорочення витрат на зберігання застарілі дані архівуються або видаляються. І останнім компонентом цього рівня є внутрішня нотифікація про наявність нових збережених даних для подання клієнтам та сервісів аналізу.

Рівень аналітики залежить від конкретної IoT-системи. Великі дані (Big Data) та аналітичний рівень отримуватимуть ситуативну інформацію з усього набору периферійних пристроїв. Ця частина менш стандартизована, тому що вона сильно відрізняється від одного додатка до іншого через різні завдання рішень. Алгоритми AI/ML також широко використовуються в цьому рівні. Окремою категорією є передбачення майбутніх подій, таких як необхідні частини на складі, споживання майбутніх ресурсів, погода та інше.

Рівень повідомлення - на цьому рівні може існувати кілька компонентів, але всі вони мають алгоритм сповіщення про підписку. Клієнтська програма підписується на необхідні події і, коли це відбувається, отримує інформаційний сигнал – повідомлення. В основному це програми електронної пошти та мобільні клієнти, рідше – телефонні дзвінки (використовується для екстреного оповіщення). Мобільний додаток змушений переходити в режим сну для енергоспоживання, але ОС iOS та Android мають механізм повідомлень, що вказує на прибуття нових даних.

Розглянемо рівень представлення. Додаток IoT може мати два потоки: M2M (від машини до машини) та M2P (від машини до людини). Рівень представлення, пов'язаний із потоком M2M, де Backend обробляє інформацію та надає її клієнту або інженеру служби підтримки. Сьогодні немає стандартизованого UI / UX подання для цього рівня, скоріше за все, що в

найближчому майбутньому він з'явиться. Рівень представлення також відповідає за обслуговування, конфігурацію та зміни стану системи, включаючи периферійні пристрої та шлюзи. У ньому представлені команди на керуючі виконавчими механізмами периферійних пристроїв.

Перейдемо до розглядання останнього рівня – рівня конфігурацій (рисунок 2.5). Цей рівень відноситься до обох потоків – M2M і M2P і працює як сховище для трьох типів статусів периферійних пристроїв:

- актуальний стан периферійного пристрою;
- новий стан периферійного пристрою, який буде завантажено;
- проміжний статус периферійного пристрою вказує на процес оновлення від старих станів до нових. Часто цей статус відсутній.

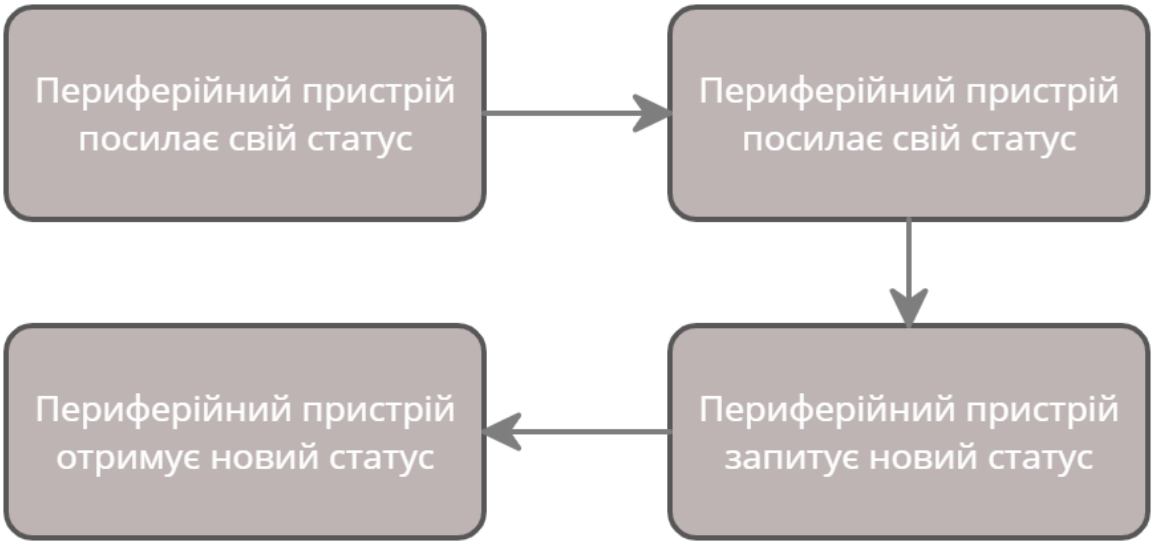

Рисунок 2.5 – Процес комунікації периферійних пристроїв з Backend

Периферійний пристрій і навіть шлюз можуть мати короткий час підключення до Backend. Будь-яка зміна статусу від клієнта або системи зберігається на цьому рівні, і протягом часу зв'язку відправляється на шлюз або периферійний пристрій.

Щоб така логіка працювала, зазвичай реалізується процес комунікації, зазначений на рисунку 2.5. Якщо шлюз присутній у схемі передачі, то більшість інформації від периферійних пристроїв відправляється на серверну частину у вигляді пакетів даних, зібраних з декількох периферійних пристроїв.

2.1.3 Підсумки аналізу архітектури IoT-систем

За результатами аналізу, можна виділити наступні тенденції побудови систем, які спостерігаються у IoT рішеннях:

а) датчики поділяються на 2 групи;

1) прості, дешеві з максимально низьким споживанням енергії, низькою швидкістю та високою дальністю передачі інформації. Це фактично разові пристрої, які не підлягають обслуговуванню;

2) засновані на відеокамері. Пристрій інтегрований з периферійним комп'ютером. Має вбудовані механізми розпізнавання образів та прийняття базових рішень;

б) ETL процеси відбуваються на декількох рівнях – периферійних пристроїв, шлюзів та Backend. Поки що немає єдиного підходу, що має робитися на кожному рівні, але ідеологія така – все, що можна обробити, має бути оброблено на якомога нижчому рівні;

в)основним, універсальним засобом передачі є бездротовий інтернет. Але фактичні протоколи відрізняються різних рівнях. Логічний рівень зв'язку – протокол LvM2M;

г) Backend – переважно Cloud. Більшість рішень на сьогоднішній день використовують Amazon Web Services;

д) як пристрій відображення фактичної інформації використовується мобільний додаток, а аналітична інформація представляється WEB додатками. Екстрене сповіщення через мобільний дзвінок із встановленим голосовим повідомленням. Електронною поштою виходять переважно звіти.

2.2 Аналіз сучасних алгоритмів та методів забезпечення захисту IoT-систем

Для забезпечення надійного захисту будь-якої IoT-системи, модель безпеки повинна бути комплексно спроектованою та охоплювати наступні три напрямки потенційної атаки:

- периферійний пристрій;
- хмарний сервіс;
- канали передачі інформації між елементами системи.

З точки зору побудови нової IoT-системи, задача найчастіше зводиться до створення захищеного хмарного сервісу-центра системи, розробки надійної та стійкої до атак системи реєстрації та автентифікації периферійних пристроїв у хмарі та забезпечення захисту каналів обміну інформацією між пристроєм та хмарою. Реалізації забезпечення безпеки першого та третього пунктів дещо перетинаються, бо для передачі будь-якої інформації щодо ідентифікації та автентифікації будь-якого пристроя системи потрібно мати захищений канал зв’язку [9].

Щоб будь-яку інформацію неможливо було просто перехопити, її треба спочатку зашифрувати. Звичні алгоритми шифрування не підходять під IoT-системи через чисельні обмеження наявних ресурсів та причин, таких як живлення, обмежена ємність акумуляторів, виконання у реальному часі та інші. Для вирішення проблеми шифрування інформації в умовах IoT-систем прийнято використовувати так звану “Легку криптографію” – така підкатегорія криптографії, метою якої є надання рішень для швидкозростаючих додатків, які широко використовують пристрої з низьким електроживленням, такі, як більшість периферійних пристроїв.

Існує два основних види способів, на які діляться легкі алгоритми шифрування (рисунок 2.6):

- симетричні;
- асиметричні.

Поговоримо про кожен з них детальніше.

Рисунок 2.6 – Алгоритми симетричного та асиметричного шифрування

У симетричних шифрах відправник та отримувач сумісно використовують спільний ключ, що передається по зашифрованому каналу як для шифрування, так і для дешифрування. Симетрична криптографія більш підходить для IoT-додатків через її швидкі операції, які здебільшого являють собою операцію XOR та перестановки. Швидкість обробки вища та вони не використовують багато ресурсів. Шляхом роботи над такими параметрами, як вибір блочного або потокового шифру, розміру ключа, розміру блока, вибір структури побудови блочного шифру та кількості раундів, отримуються легкі симетричні алгоритми.

Приклади легких симетричних шифрів можна побачити на рисунку 2.7.

Легкі симетричні алгоритми					
Назва алгоритму	**Довжина кода**	**Структура побудови шифру**	**Кількість раундів**	**Розмір ключа**	**Розмір блока**
AES	2606	SPN	10	128	128
HEIGHT	5672	GFS	32	128	64
TEA	1140	Feistel	32	128	64
PRESENT	936	SPN	32	80	64
RC5	Не фіксована	ARX	20	16	32

Рисунок 2.7 – Порівняльна таблиця деяких симетричних алгоритмів

Асиметричні шифрі менше підходять під опис легких алгоритмів, але їх все одно потрібно брати до уваги. Криптографія з асиметричним ключем відома як криптографія з публічним ключем, бо у цьому методі використовується пара публічного та приватного (або відкритого та закритого) ключів. Легкі асиметричні алгоритми складні з точки зору роботи та неефективні по часу, але у останній час спостерігається тенденція збільшення долі асиметричних алгоритмів у легкій криптографії, втім результати ще не дуже стабільні та плідні, як у криптографії з симетричним ключем. На рисунку 2.8 зображено порівняння двох основних легких асиметричних шифрів, що використовуються у наш час.

Легкі асиметричні алгоритми		
Назва алгоритму	**Розмір ключа**	**Довжина кода**
RSA	1024	900
ECC	160	8838

Рисунок 2.8 – Порівняльна таблиця деяких асиметричних алгоритмів

Отже, якщо підбивати підсумки порівняння симетричних та асиметричних алгоритмів шифрування, можна зробити наступні висновки:

- симетричні алгоритми мають один спільний секретний ключ, у той час як асиметричні мають пару публічний-приватний ключі;
- симетричні шифри мають значно більшу швидкість, ніж асиметричні, передача та обчислення яких вимагають значно більше часу та ресурсів;
- асиметричні шифри набагато складніше зламати, ніж симетричні.

Більшість протоколів передачі та захисту даних, таких як SSL та його наступник TLS використовують одразу обидва види шифрування для встановлення з’єднання та передачі інформації. Спочатку за допомогою асиметричного ключа здійснюється ідентифікація пристроя системою та встановлюється захищене з'єднання, а потім сервер генерує симетричний ключ і передає його пристрою для швидкого обміну даними без великих витрат обчислювальних потужностей та часу. Цей алгоритм часто називають “протоколом рукостискання”.

Отже, вище були розглянуті актуальні методи забезпечення захищеного підключення до серверу. Настав час перейти до розглядання організації безпеки, безпосередньо, самого хмарного сервісу.

Захист серверної частини системи можна сміливо назвати найголовнішою задачею у побудові IoT-системи. Першочергово завжди потрібно вирішити питання фізичного захисту сервера, але у даній роботі розглядається забезпечення саме інформаційної безпеки системи, тому перейдемо одразу до наступного пункту.

Виділимо основні дві причини загрози безпеки системи:

- вторгнення;
- вірусне та шпигунське ПЗ.

Щоб захистити хмару від вторгнення, вірусів та витоку даних, необхідно налаштувати контроль доступу. Це не просто логін і пароль, це глобальне поняття, яке стосується всіх засобів безпеки. Іншими словами, задача полягає в тому, щоб не просто налаштувати доступ для користувачів, а не допустити вторгнення ззовні. Розглянемо методи запобігання вторгнень [10].

Фільтрація даних та контроль використання ресурсів заради запобігання несанкціонованому доступу до хмарних послуг. Основний спосіб коректно відфільтрувати відомості та проконтролювати доступ до ресурсів – використовувати правильний фаєрвол (брандмауер), їх почали використовувати для захисту даних у хмарі нещодавно. Сучасні фаєрволи відрізняються інтегрованими функціями безпеки для хмарних сервісів:

- функції NAT/PAT;
- глибока перевірка пакетів – з поведінковим підписом або IDS/IPS/;
- спеціалізовані веб-елементи управління - правила WAF.

Інфраструктуру IoT-системи можна представити у вигляді мережей та кінцевих точок, якими є периферійні пристрої та хмарний сервіс. Фаєрвол повинен контролювати трафік на усіх ділянках мережі, такий принцип зветься безшовною інтеграцією. Головною її перевагою є можливість розгортання списків контролю доступу на будь-якому рівні – мережевому або на кінцевих точках. Тобто для ефективного захисту інформації у хмарі розгортається велика кількість списків контролю доступу на всіх можливих точках входу.

Управління безпекою в хмарних сервісах також може включати додаткові методи захисту, наприклад:

- модулі розпізнавання та запобігання DDoS-атак;
- системи розподілення трафіку;
- захист DNS;
- системи керування контролем доступу, ролей та дозволів.

Важливу роль відіграють також інструменти відстеження інцидентів, які найчастіше складаються з двох елементів – засобів моніторингу та засобів візуалізації. Вони дозволяють відстежувати трафік та його збої, слабкі місця захисту, атаки на всіх рівнях хмарної архітектури [11].

2.3 Використання досліджених методів захисту у побудові моделі комплексної системи безпеки

Підбиваючи підсумки вищесказаного, було висунуто наступні вимоги до побудови моделі захисту демонстраційної IoT-системи [12]:

- реєстрація та автентифікація периферійних пристроїв методом симетричного ключа та алгоритму шифрування AES-256;
- забезпечення захищеного з’єднання за допомогою використання запитів, побудованих на базі протоколу TLS 1.2;
- розробка хмарного сервісу з аналізом вхідного трафіку, мережевим журналом та брокером повідомлень, що використовує протокол AMQP для передачі повідомлень з черги до сервера;
- введення системи акаунтів з різними рівнями доступу для контролю користувачів та їхніх пристроїв;
- розширення функціоналу хмарного сервісу додатковими елементами інформаційного захисту, такими як фільтр IP-адресів, захист від підбору пароля, захист від SQL-ін’єкцій, та інші функції.

2.4 Моделювання захисту системи від потенційних атак

Розділимо усі потенційні атаки на основні три категорії:

- атаки на периферійний пристрій;
- атаки на канал обміну інформацією;
- атаки на хмарний сервер.

Ці три типи атак можна віднести до трьох рівнів – рівня представлення, мережевого та прикладного рівнів. Кожен з цих трьох рівнів має свої механізми захисту. Розглянемо їх по черзі.

2.4.1 Моделювання потенційної атаки на пристрій

Говорячи про загальні атаки на рівень представлення, можемо виділити наступні види атак [13]:

- фізична атака. Зловмистник у разі отримання фізичного доступу до периферійного пристрою може пошкодити або цілком знищити усю мережу шляхом фізичного пошкодження пристроїв або маніпулювання та експлуатації периферійних пристроїв з метою вторгнення у систему;
- атаки на шифрування. Цей клас атак поєднує у собі одразу декілька видів, усі з яких зводяться до злому шифру та отримання доступу до ключа шифрування, який дозволить зловмистнику перехопити інформацію. Серед менш критичних наслідків цих атак – витрати енергії та часу периферійних пристроїв або витік інформації.

Запропонована модель безпеки повинна бути здатна впоратися з кожною з цих атак. Поговоримо окремо про кожну з них.

Кращий вид захисту від фізичної атаки – ізоляція периферійних пристроїв поза відкритим доступом для уникнення ситуацій, у яких потенційний зловмистник може отримати фізичний доступ до пристроя. Прикладами цього може будет розміщення IP-камер на достатній висоті, щоб виключити можливість фізичної взаємодії з нею.

Для запобігання атак на шифрування запропонована система захисту використовує актуальні та надійні алгоритми шифрування, такі, як AES-256 та RSA. Ці алгоритми мають високий рівень криптостійкості та використовуються навіть для зберігання засекреченої інформації урядом США. Тому можно сміливо стверджувати, запропонована система є стійкою до атак на шифрування.

2.4.2 Моделювання потенційної атаки на канал обміну інформацією

Самою розповсюдженною атакою цього типу є Denial-of-Service атака (атака на відмову в обслуговуванні). Цей тип атак полягая у створенні таких умов, при яких сервер не може працювати зі звичайною швидкістю чи навіть взагалі перестає працювати. При цьому доступ користувачів до системних ресурсів буде обмежений або навіть закритий. Найчастіше цей тип атак проводится з метою переповнення пропускної смуги та подальшого вичерпання системних ресурсів.

Для захисту від DoS атак використовується фаєрвол з аналізатором трафіку на базі машинного навчання, який здатні виявляти підозрілі запити та блокувати їх. Щоразу, коли виявляється підвищення обсягу трафіку, що потрапляє на сервер як орієнтир можна брати максимально можливий обсяг трафіку, який може обробити хост без погіршення його доступності. Така концепція називається обмеженням швидкості.

Більш просунуті методи захисту відповідно мають додаткові можливості і можуть інтелектуально приймати тільки трафік, який дозволено, аналізуючи окремі пакети. Для використання подібних засобів необхідно визначити характеристики хорошого трафіку, який зазвичай отримує цільовий об'єкт, та мати можливість порівнювати кожен пакет із цим еталоном.

Крім того, через унікальність деяких атак ви необхідно бути здатним самостійно нейтралізувати заборонені запити, які можуть мати певні характеристики, наприклад, можуть визначатися як відмінні від хорошого

трафіку або виходити з підозрілих IP-адрес, з несподіваних географічних регіонів.

2.4.3 Моделювання потенційної атаки на сервер

Враховуючи архітектуру хмарного серверу, основними напрямками потенційних атак можуть стати атаки на БД, наприклад, атаки з ін’єкцією шкідливого коду шляхом введення до бази даних SQL-запитів, обгорнутих у звичайний поток даних з метою скомпрометувати цілісність БД та системи та атаки, що здійснюються з метою підбору паролів до акаунтів користувачів для отримання контролю над їхніми пристроями.

Для захисту від атак з ін’єкцією хмарний сервер спроектований таким чином, щоб не використовувати вводиму користувачем інформацію у формуванні SQL-запиту. Це досягається шляхом параметризованих запитів. Замість конкатенації введеної користувачем інформації треба використовувати параметри. У цьому випадку, який би текст користувач не ввів, сервер буде шукати цей текст як параметр. Якщо текст містить нерелевантний фрагмент (наприклад, з SQL-запитом), ніякої ін'єкції не відбудеться.

Для запобігання підбору паролів користувачів, сервер має такі захисні функції, як обмеження допустимої кількості невдалих спроб авторизації у системі з подальшим тайм-аутом, тривалість якого можна змінювати. Також, у випадку необхідності, можна повністю обмежити підключення до сервера з усіх IP-адресів, окрім вказаних, що дозволить закрити доступ до сервера потенційному зловмистнику.

2.5 Вибір програмних засобів реалізації побудованої системи безпеки

У ході аналізу актуальних архітектур IoT-систем, алгоритмів їхньої побудови та методів забезпечення всебічного захисту системи, а також у

процесі отримання інформації о актуальних засобах розробки, було вирішено використати наступні технології для створення демонстраційної IoT-системи та забезпечення її захисту, що відповідає поставленому завданню:

- мова програмування Python 3.9.1 для написання основної логіки ПЗ периферійного пристрою та хмарного сервісу;
- фреймворк Flask 2.2.1 у якості фундаменту хмарного веб-застосунку;
- брокер повідомлень RabbitMQ 3.10.13 для зручного та комфортного налаштування оброблення та маршрутизації потоку повідомлень від пристроїв до серверу;
- бібліотеки Crypto, base64, SSL, urllib та Fernet для шифрування та побудови комплексної системи безпеки IoT-системи;
- бібліотека imutils для формування та передачі відеопотоку з пристроя на сервер;
- бібліотека Open-CV та згорткова нейронна мережа архітектури опорних векторів для детекції та підрахунку людей у входному відеопотоку у реальному часі;
- СУБД MySQL для збереження даних з сенсорів пристроїв та ідентифікаційних даних користувачів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛІ БЕЗПЕКИ

3.1 Загальна структурна схема системи

Після проведення аналізу та проектування архітектури, була запропонована наступна демонстраційна система, що побудована по усім принципам та стандартам IoT-архітектури. Розроблена демонстраційна система є універсальною, бо вона може працювати з різними типами периферійних пристроїв, здатна обробляти різні види вхідної інформації з сенсорів, обробляти її тощо. Нижче приведено структурну схему розробленої системи (рисунок 3.1).

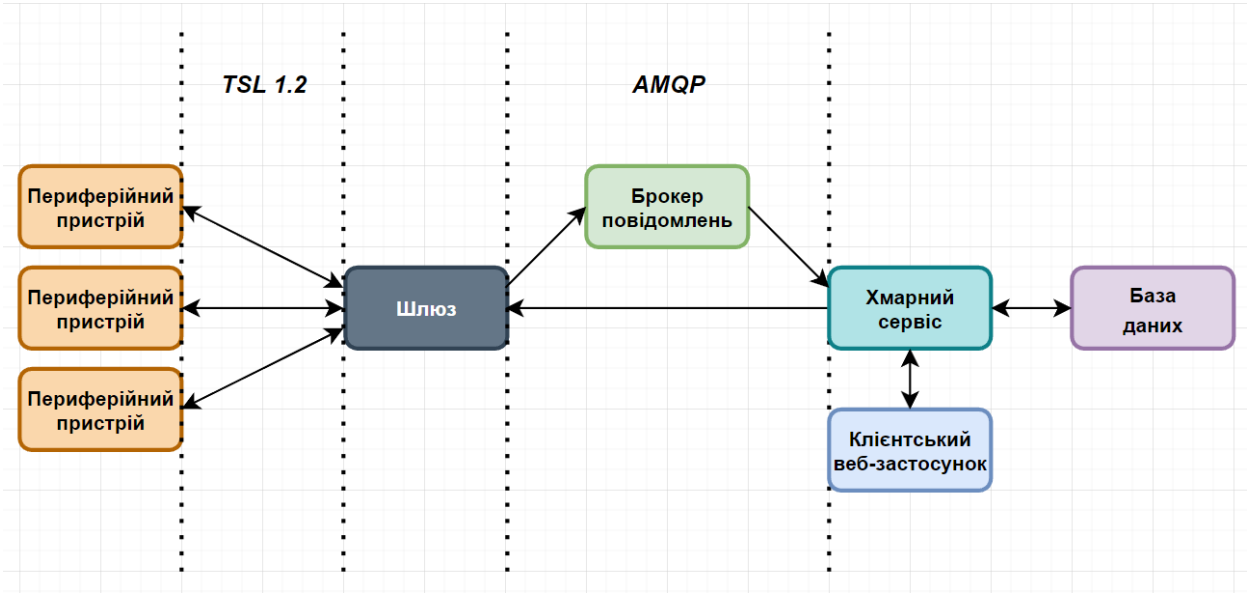

Рисунок 3.1 – Структурна схема демонстраційної IoT-системи

Спочатку відбувається реєстрація, автентифікація та підключення периферійних пристроїв до хмари за допомогою шлюзу, з'єднання організовано шляхом запитів на базі протоколу TSL 1.2, що забезпечує надійний захист від перехвату та дешифровки інформації. Далі авторизовані у системі пристрої починають передачу зібраної зі своїх сенсорів інформації, яка

поступає у брокер повідомлень, який формує чергу повідомлень та здійснює керування нею, забезпечуючи маршрутизацію та сортування повідомлень, отриманих від пристроїв. Усі канали обміну інформацією між брокером повідомлень та іншими елементами системи працюють згідно протоколу AMQP. Далі повідомлення від брокера попадають до хмарного сервісу, де виконуються усі необхідні обчислення та можливі подальші дії, такі як, наприклад, сигнал про зміну стану периферійного пристрою. У разі необхідності дані зберігаються у базі даних. Клієнтський застосунок представляє собою графічний веб-інтерфейс, за допомогою якого користувач може здійснювати керування своїми пристроями та отримувати інформація про них. Адміністратор системи також може отримувати інформацію про стан системи, підключені пристрої, акаунти клієнтів та змінювати налаштування серверу, у тому числі налаштування безпеки.

3.2 Програмування ПЗ периферійного пристрою

Для створення основної логіки ПЗ периферійного пристрою була використана мова Python, а також допоміжні бібліотеки imutils, зокрема її клас VideosStream, методи якого дозволять зручно оброблювати та перетворювати вхідну інформація з камери (яка є основним сенсором демонстраційного периферійного пристрою), base64, Crypto, ssl та urllib3 для шифрування інформації та створення захищеного з’єднання зі шлюзом.

3.2.1 Налаштування камери та формування відеопотоку

Поговоримо про використану бібліотеку imutils та її клас VideoStream. Цей клас об’єднує між собою два інших класи, які називаються WebcamVideoStream і PiVideoStream, але у нашому випадку (USB/Embed камери) нас цікавить лише перший с них.

Лістинг 3.1 – Код конструктора класу VideoStream з бібліотеки imutils

```
from .webcamvideostream import WebcamVideoStream

class VideoStream:
     def __init__(self, src=0, usePiCamera=False,
          resolution=(600, 600),framerate=30, **kwargs):
          if usePiCamera:
               from .pivideostream import PiVideoStream
               self.stream =PiVideoStream(resolution=resolution,
                    framerate=framerate, **kwargs)
          else:
               self.stream = WebcamVideoStream(src=src)
```

Лістинг 3.2 – Код конструктора класу WebcamVideoStream

```
from threading import Thread
import cv2

class WebcamVideoStream:
     def __init__(self, src=0,
          name=&amp;quot;WebcamVideoStream&amp;quot;):
     self.stream = cv2.VideoCapture(src)(self.grabbed,
          self.frame) = self.stream.read()
     self.name = name
     self.stopped = False
```

Обидва наведених вище класи мають спільні методи:

- .start(self) – метод запуску, що використовується для запуску потока метода оновлення;
- .update(self) – метод оновлення, опитує модуль камери та вихоплює останній кадр із відеопотоку, та тимчасово зберігає його у змінній кадру;
- .read(self) – метод, що повертає останній вихопленний кадр;
- .stop(self) – метод, зупиняющій потік;

Лістинг 3.3 – Створення об’єкту класа VideoStream та включення камери з двохсекундною паузою для активації сенсора

```
vs = VideoStream(src=0).start()
time.sleep(2)
```

Будемо передавати відеопотік у форматі окремих кадрів, тому розіб'ємо його (Лістинг 3.4).

Лістинг 3.4 – Конвертація вхідного відеопотока у набор окремих кадрів

```
while True:
     frame = vs.read()
     frame = imutils.resize(frame, width = 600, height = 600)
```

3.2.2 Створення захищеного підключення до хмари

Обмін інформацією між пристроєм та хмарою повинен здійснюватися за допомогою захищених запитів, для цього скористаємося бібліотеками ssl та requests.

Лістинг 3.5 – Підключення бібліотек ssl та requests

```
import requests
import ssl
```

Самі по собі запити бібліотеки requests не мають суттевого захисту, тому напишемо свій клас HTTPAdapter з примусовим вибором потрібного протоколу, у нашому випадку – TLS 1.2.

Лістинг 3.6 – Створення классу HTTPAdapter

```
class ForceTLSV2Adapter(adapters.HTTPAdapter):
    def init_poolmanager(self, connections, maxsize,
     block=False):
        self.poolmanager = poolmanager.PoolManager(
            num_pools=connections,
            maxsize=maxsize,
            block=block,
            ssl_version=ssl.PROTOCOL_TLSv2,
        )

    def proxy_manager_for(self, proxy, **proxy_kwargs):
```

```
        proxy_kwargs['ssl_version'] = ssl.PROTOCOL_TLSv1
        return super(ForceTLSV2Adapter,
self).proxy_manager_for(proxy, **proxy_kwargs)
```

Якщо спробувати здійснити будь-який тестовий захищений запит зараз, то ми отримаємо помилку, бо немає сертифікати, що підтверджує справжність та безпеку кінцевого хоста.

Лістинг 3.7 – Спроба здійснити тестовий запит та код помилки

```
response=requests.get(url)

print(response)

-------------------------------------------------------------------

requests.exceptions.SSLError: HTTPSConnectionPool: Max retries
exceeded with url : / (Caused by SSLError
(SSLCertVerificationError(1, '[SSL:CERTIFICATE_VERIFY_FAILED]
certificate verify failed: unable to get local issuer
certificate (_ssl.c:1131)')))
```

Процес самостійного створення сертифіката SSL для демонстраційної системи складається з трьох кроків:

- створення закритого ключа RSA;
- створення запиту на підписання сертифікату за допомогою закритого ключа;
- підписання запиту для створення сертифікату.

Після створення сертифікату, ми використовуватимемо його під час надсилання запитів. Перейдемо до реалізації.

Спочатку перевіримо наявність бібліотеки OpenSSL (рисунок 3.2) на пристрої (найчастіше вона вже включена у більшість версій Windows та Linux).

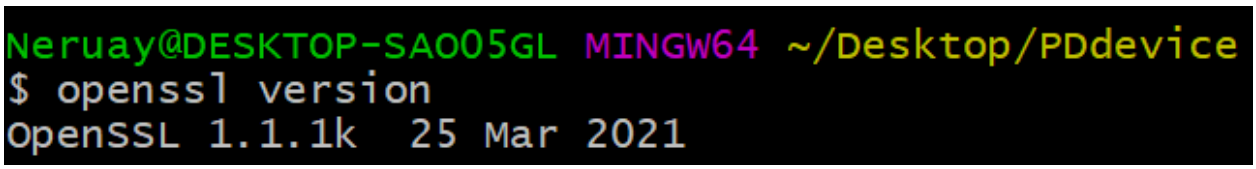

Рисунок 3.2 – Перевірка наявності модуля OpenSSL у системі

Ми можемо створити закритий ключ із паролем або без нього. Щоб додати пароль, ми використовуємо шифр AES-256. Сгенеруємо 2048-бітний закритий ключ RSA (рисунок 3.3).

Рисунок 3.3 – Генерація закритого RSA ключа

На цьому кроці ми використовуємо закритий ключ, згенерований на попередньому кроці, для створення запиту на підписання сертифікату. Введемо консольну команду і заповнимо необхідні поля з додатковою інформацією (рисунок 3.4).

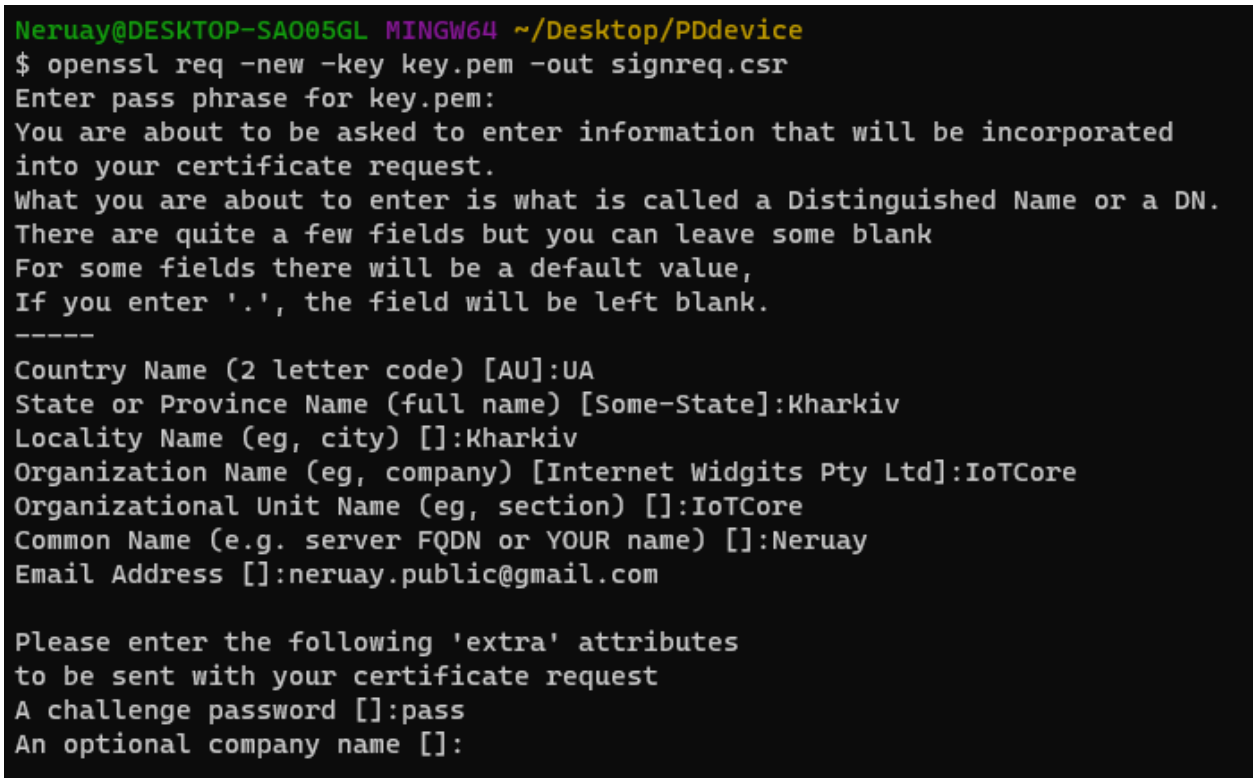

Рисунок 3.4 – Запит на підписання сертифікату

Останнім кроком ми підписуємо сертифікат на 1 рік командою у лістингу 3.8.

Лістинг 3.8 – Підписання сертифікату

```
openssl x509 -req -days 365 -in signreq.csr -signkey key.pem -
out certificate.pem
```

Після підписання сертифікату ми можемо перевірити його статус та деталі, як показано на рисунку 3.5.

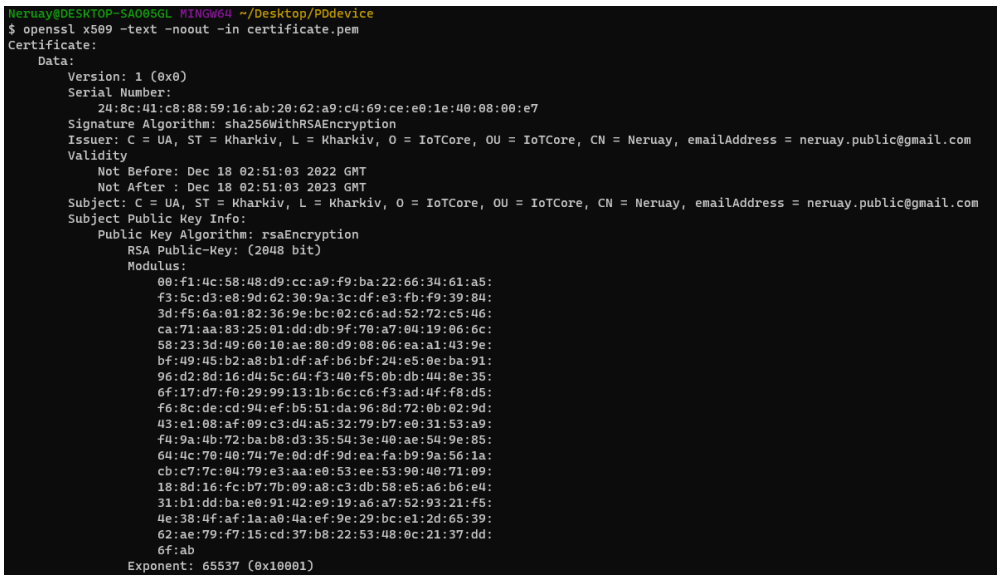

Рисунок 3.5 – Отримання інформації про підписаний сертифікат

Після успішного створення сертифікату, ми можемо його використати для створення тестового сервера HTTPS, який буде використовувати шифрування TLS 1.2 під час спілкування з клієнтом. Наведена нижче програма Python запускає HTTPS сервер на локальному хості на порту 443.

Лістинг 3.9 – Запуск локального HTTPS сервера

```
import http.server
import ssl

httpd = http.server.HTTPServer(('localhost', 443),
http.server.SimpleHTTPRequestHandler)
httpd.socket = ssl.wrap_socket (httpd.socket,
certfile='./certificate.pem', server_side=True,
ssl_version=ssl.PROTOCOL_TLS
httpd.serve_forever()
```

Викликаємо метод HTTPServer на http.server, передавши адресу та номер порту сервера. Далі обгортуємо сокет за допомогою SSL, вказавши шлях до створеного сертифіката в методі wrap_socket(). Нарешті, запускаємо сервер останньою строчкою. А зараз спробуємо здійснити запит з лістингу 3.7 ще раз та побачимо відгук з кодом 200, що означає успіх.

Рисунок 3.6 – Успішне виконання запиту

Після встановлення безпечного каналу зв’язку, можна перейти до імплементації процедури реєстрації та автентифікації пристроїв у системі. Для цього при першому підключенні пристрою до хмари, користувачу буде запропоновано обрати пароль, після чого на стороні сервера буде згенеровано симетричний ключ, який також буде передано на пристрій, а обраний пароль буде зашифровано. Функції генерації ключа, кодування та декодування паролю наведено у листінгу 3.10.

Листінг 3.10 – Основні криптографічні функції генерації ключа, кодування та декодування пароля за ключем.

```
//deriving the key from the passphrase
def derive_key(self, salt: bytes, iterations: int):
        k = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,
            iterations=iterations,
            backend=self.backend)
        return b64e(k.derive(self.passphrase.encode()))

//encrypting password with the key
def password_encrypt(self, message: bytes, iterations: int):
        salt = secrets.token_bytes(self.default_salt_bytes)
        key = self.derive_key(salt, iterations)
        return b'~'.join([
            b64e(salt),
            b64e(iterations.to_bytes(int(log(iterations, 256)) +
1, 'big')),
            Fernet(key).encrypt(message)
        ])
```

```
//decrypting password with the key
def password_decrypt(self, token: bytes):
    salt, iterations_b, token = token.split(b'~')
    salt = b64d(salt)
    iterations = int.from_bytes(b64d(iterations_b), 'big')

    key = self.derive_key(salt, iterations)
    try:
        return Fernet(key).decrypt(token)
    except (InvalidToken, TypeError):
        return None
```

Функція генерації приймає парольну фразу та використовує механізм HMAC для отримання більш безпечного ключа, використовуючи шифрування SHA256, криптографічну сіль принаймні 16 байтів та більше 10 ітерацій хешу HMAC.

PBKDF2 – це ключова функція виведення, яка використовується для зменшення вразливості до атак грубою силою. Цей KDF використовує сіль і парольну фразу, багато разів пропускаючи їх через HMAC для створення похідного ключа, який також відомий як розтягування ключа.

Отримавши ключ із парольної фрази, ми можемо зашифрувати наше повідомлення. Для шифрування ми викликаємо метод Fernet.encrypt(M) і пакуємо нашу сіль, ітерації та зашифрований текст у “конверт”. Зразок конверта наведено в листінгу 3.11. Дешифрування працює схожим чином.

Листінг 3.11 – Зразок запакованої інформації Fernet.encrypt

```
iMDwBtpp-mw0jWaDPzHQWg==~Cg==~gAAAAABeKevMZAtUo43RcEJZCz...
```

Тепер, коли усі функції генерації, шифровки та дешифровки створені, задача реєстрації та автентифікації зводиться до збереження інформації про пристрій та подальшого надсилання запиту до БД при кожному підключенні пристрою. Приклад реєстрації нового пристрою у БД наведено у листінгу 3.12.

Листінг 3.12 – Реєстрація пристроя у системі

```
sql = "INSERT INTO devices (name, pass, address, owner) VALUES
(%s, %s, %s, %s)"
val = (name, pass, address, owner)
mycursor.execute(sql, val)
mydb.commit()
```

3.3 Імплементація хмарного IoT-сервісу

Хмарний сервер та користувацький веб-інтерфейс були створені за допомогою фреймворка Flask на мові програмування Python. Фронтенд був зроблений на Vue.js. Враховуючи тематику роботи, будемо докладно розглядати лише окремі пункти розробки веб-сервісу, які стосуються, безпосередньо, суті роботи. Почнемо зі створення функції реєстрації та авторизації для користувачів. Створимо модель користувача, яка складається з 4 полей – ідентифікатор, ім’я, пароль та роль (лістинг 3.13).

Лістинг 3.13 – Створення моделі користувача

```
from . import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    roles = db.relationship('Role', secondary='user_roles')
```

Додамо дві базові ролі – адміністратор та звичайний користувач.

Лістинг 3.14 – Формування списку ролей

```
class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String(50), unique=True)
class UserRoles(db.Model):
    __tablename__ = 'user_roles'
```

```
    id = db.Column(db.Integer(), primary_key=True)
    user_id = db.Column(db.Integer(), db.ForeignKey('users.id',
          ondelete='CASCADE'))
    role_id = db.Column(db.Integer(), db.ForeignKey('roles.id',
          ondelete='CASCADE'))
user.roles.append(Role(name='admin'))
user.roles.append(Role(name='user'))
```

Налаштуемо функції реєстрації та автентифікації за стандартними шаблонами Flask. Для функції реєстрації візьмемо дані, які користувач надсилає у форму та додамо їх до БД. Нам потрібно буде переконатися, що користувача із таким ім'ям ще не існує в базі даних. Перед додаванням до БД пароль потрібно хешувати.

Фрагмент коду функції реєстрації можна побачити нижче (лістинг 3.15).

Лістинг 3.15 – Функція реєстрації у системи

```
def signup_post():
    name = request.form.get('name')
    password = request.form.get('password')
    roles = request.form.get('roles')

    user = User.query.filter_by(name).first()

    if user:
        return redirect(url_for('auth.signup'))

    new_user = User(name=name,
password=generate_password_hash(password, method='sha256'),
roles=roles)

    db.session.add(new_user)
    db.session.commit()

    return redirect(url_for('auth.login'))
```

Зовнішній вигляд функції реєстрації можна побачити на рисунку 3.7.

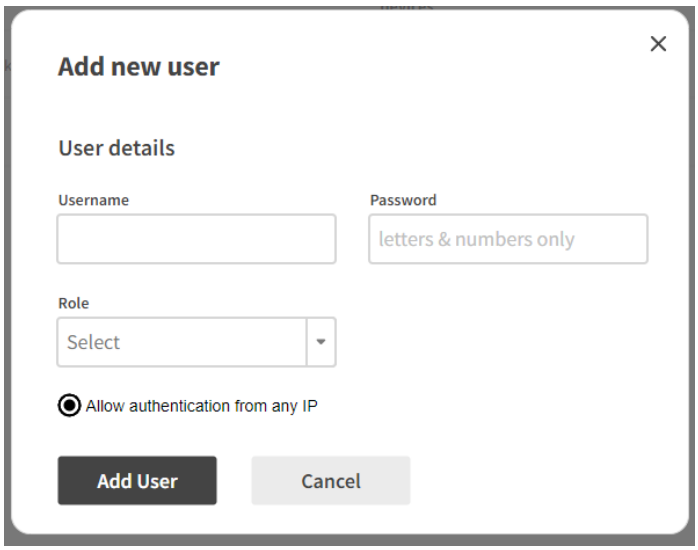

Рисунок 3.7 – Вікно реєстрації нового користувача

Перейдемо до імплементації функції входу у систему. Спосіб входу подібний до функції реєстрації. У цьому випадку ми порівнюємо введене ім’я, щоб побачити, чи є воно у БД. Якщо так, перевіряємо пароль, наданий користувачем, хешуючи його і порівнюємо з хешованим паролем БД. Якщо паролі збігаються – користувач ввів правильний пароль.

Викликаючи login_user, Flask-Login створить сеанс для цього користувача, який триватиме, поки користувач не ввійшов у систему, що дозволить користувачеві переглядати захищені сторінки.

Нижче наведено фрагмент функції входу у систему (лістинг 3.16).

Лістинг 3.16 – Функція автентифікації у систему

```
def login_post():
    name = request.form.get('username')
    password = request.form.get('password')

    user = User.query.filter_by(name=name).first()

    if not user or not check_password_hash(user.password,
          password):
        flash('Access denied: wrong password')
        return redirect(url_for('auth.login'))
    return redirect(url_for('main'))
```

Тепер ми маємо вікно автентифікації з функцією регістрації користувача при спробі зайти на головну сторінку веб-застосунку хмари (рисунок 3.8).

Рисунок 3.8 – Вікно авторизації у системі

Спробуємо створити тестового користувача (рисунок 3.9).

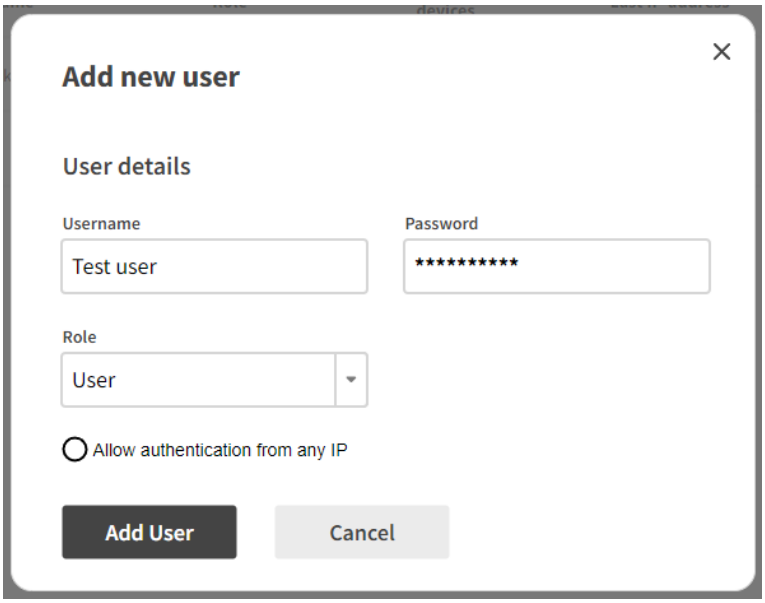

Рисунок 3.9 – Створення тестового користувача

Тепер спробуємо зайти у створений акаунт користувача (рисунок 3.10).

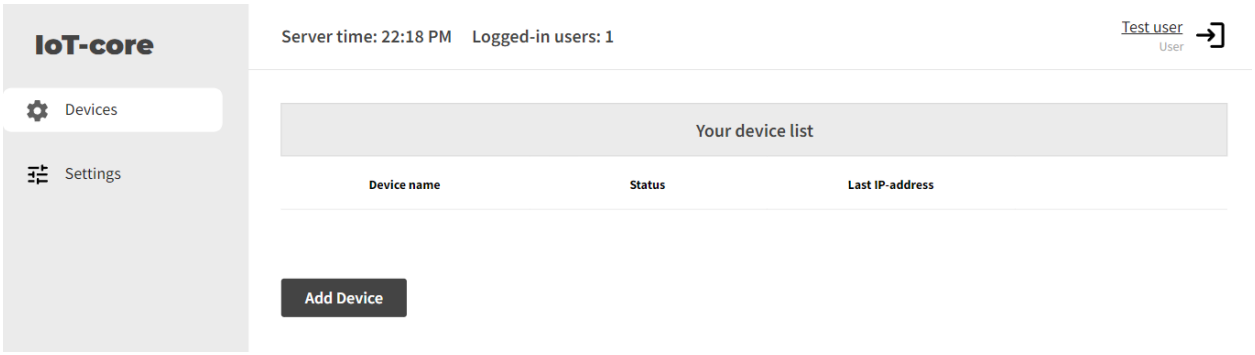

Рисунок 3.10 – Інтерфейс акаунту користувача

Підключимо наш пристрій (рисунок 3.11).

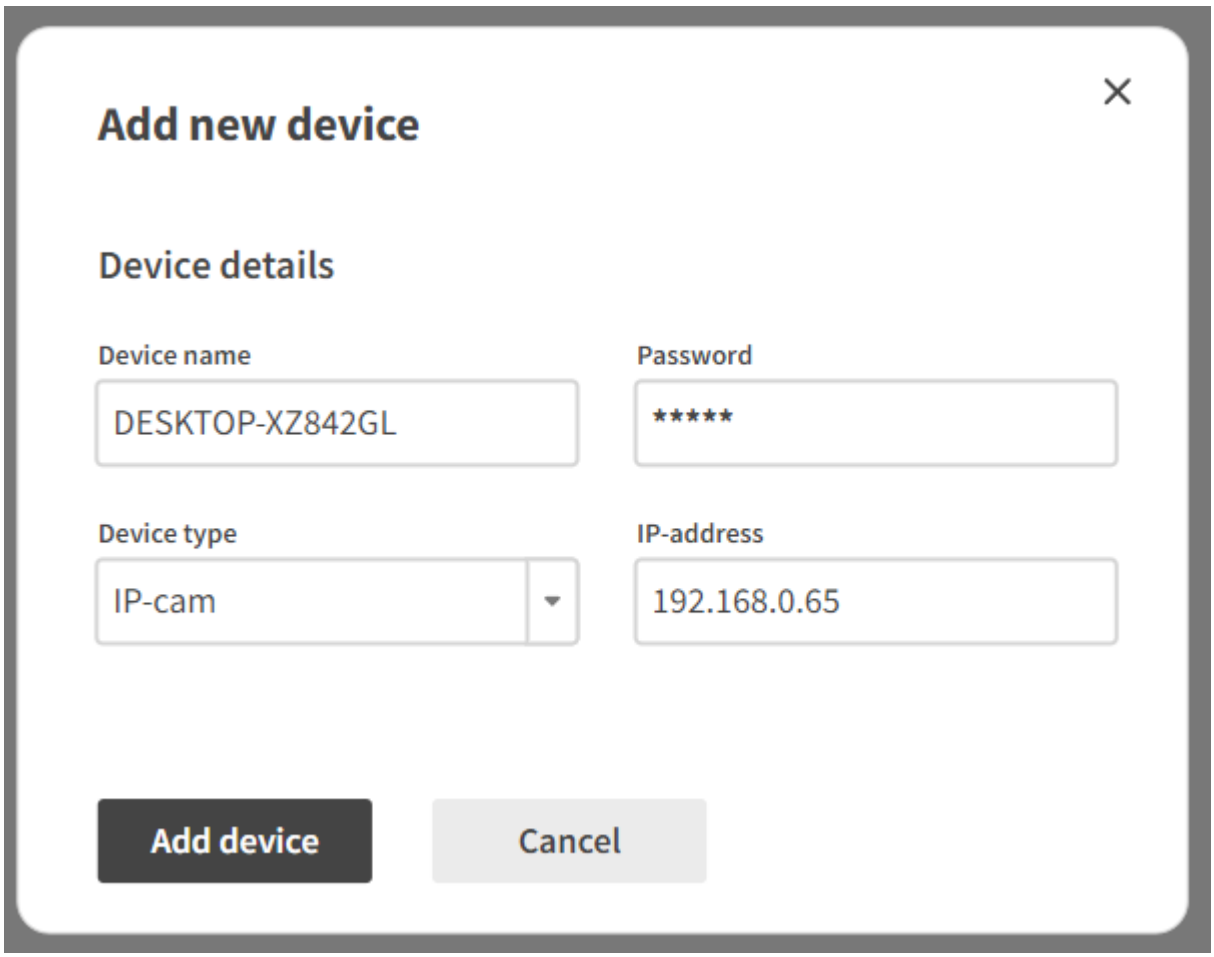

Рисунок 3.11 – Підключення нового пристрою

Тепер ми можемо його бачити у списку підключених пристроїв у акаунті (рисунок 3.12).

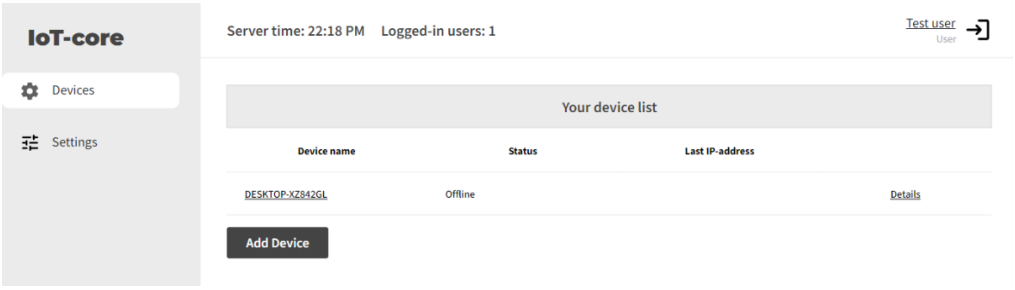

Рисунок 3.12 – Новий пристрій з’явився у списку

Підключимось з пристроя до хмари (лістинг 3.17).

Лістинг 3.17 – Під'єднання камери до хмари за портом RTSP 554

```
req =
“rtsp://username:password@ip_address:554/user=username_password=
'qwert'_channel=channel_number_stream=0.sdp'”

device = cv2.VideoCapture(req)

app.connect(host=flask.app(), port=554, param=(device, true,
true, fps=30), key=key)
```

Тепер ми можемо побачити, що пристрій підключено до системи (рисунок 3.13).

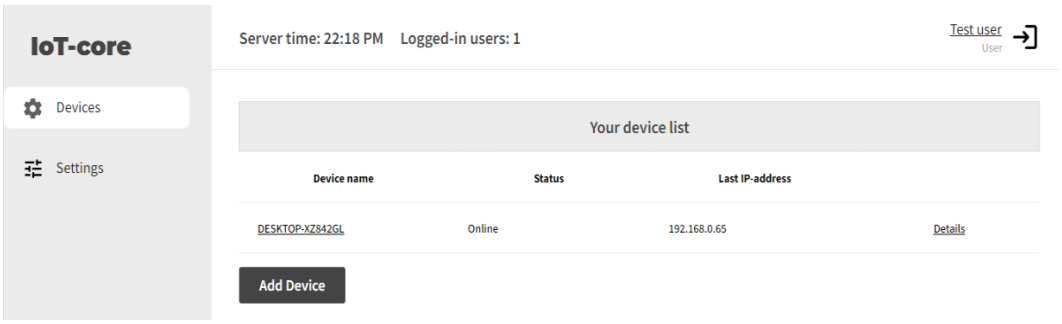

Рисунок 3.13 – Пристрій підключено до системи

Переглянемо інформацію про підключений пристрій детальніше. Як можна побачити, він ще не транслює відеопотік та не надсилає ніяких даних з сенсорів (рисунок 3.14).

Рисунок 3.14 – Первинна інформація про під’єднаний пристрій

Створимо функціонал передачі кадрів на сервер, та напишемо сервіс для розпізнавання людей у відеопотоці (лістинг 3.18).

Лістинг 3.18 – Функція генерації окремих кадрів на стороні сервера

```
def gen_frames():
    while True:
        success, frame = camera.read()
        if not success:
            break
        else:
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame +
                         b'\r\n')
```

Створимо класс детектора людей у кадрів (лістинг 3.19).

Лістинг 3.19 – Прототип класу детектору

```
class SingleMotionDetector:
     def __init__(self, accumWeight=0.5):
          self.accumWeight = accumWeight
          self.bg = None
```

Імплементуємо необіхдні для роботи з відео функції оновлення кадру та розпізнавання (лістинги 3.20 та 3.21).

Лістинг 3.20 – Функція оновлення кадру

```
def update(self, image):
     if self.bg is None:
          self.bg = image.copy().astype("float")
          return
     cv2.accumulateWeighted(image, self.bg,
               self.accumWeight)
```

Лістинг 3.21 – Функція розпізнавання людей у кадрі

```
def detect(self, image, tVal=25):
     delta = cv2.absdiff(self.bg.astype("uint8"), image)
          thresh = cv2.threshold(delta, tVal, 255,
             cv2.THRESH_BINARY)[1]
     thresh = cv2.erode(thresh, None, iterations=2)
     thresh = cv2.dilate(thresh, None, iterations=2)
     cnts = cv2.findContours(thresh.copy(),
             cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
     cnts = imutils.grab_contours(cnts)
     (minX, minY) = (np.inf, np.inf)
     (maxX, maxY) = (-np.inf, -np.inf)
     if len(cnts) == 0:
          return None
     for c in cnts:
          (x, y, w, h) = cv2.boundingRect(c)
          (minX, minY) = (min(minX, x), min(minY, y))
          (maxX, maxY) = (max(maxX, x + w), max(maxY, y +
     return (thresh, (minX, minY, maxX, maxY))
```

Створимо маршрут для передачі оброблених кадрів (лістинг 3.22).

Лістинг 3.22 – Налаштування маршрутів

```
@app.route("/video_feed")
def video_feed():
     return Response(generate(),
          mimetype = "multipart/x-mixed-replace;
boundary=frame")
```

Перевіримо роботи сервісу (рисунок 3.15).

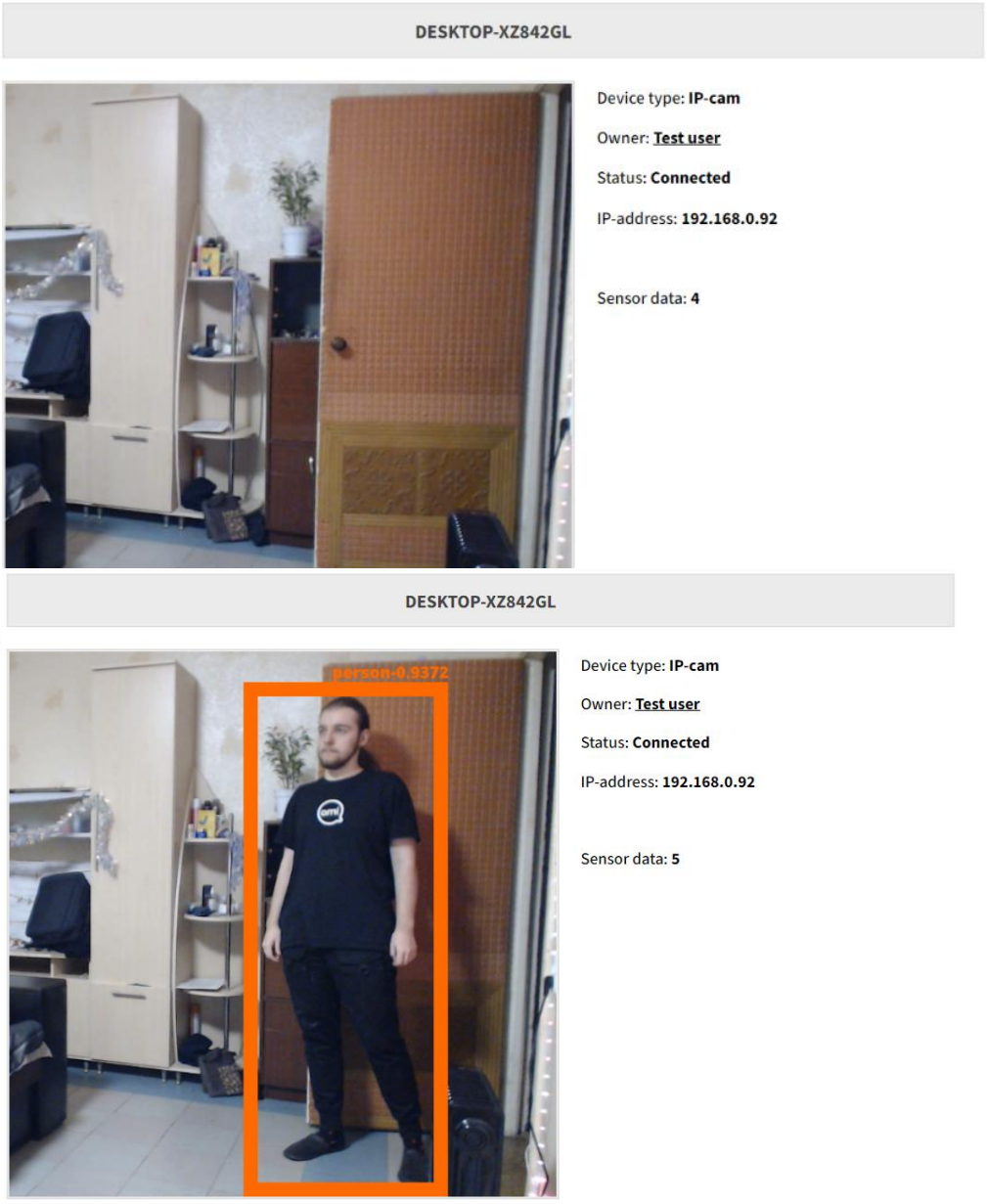

Рисунок 3.15 – Демонстрація роботи передачі зображення у хмару з периферійного пристрою

Як можна побачити, модуль працює у повному обсязі, зображення стабільно передається на сервер, де розроблена функція пошуку людей успішно знаходить людину у кадрі та оновляє лічильнік сенсора.

3.4 Тестування створеної IoT-системи

Сервер розробленої системи був розгорнутий на платформі з наступною конфігурацією:

- CPU – Intel Core i7-4770K @ 3.5 ГГц;
- RAM – 16 ГБ;
- GPU – NVIDIA GeForce GTX 1070;
- VRAM – 8 ГБ;
- OS – Windows 10.

У якості периферійного пристрою було використано платформу з наступною конфігурацією:

- CPU – Intel Core i7-1065G7 @ 2.5 ГГц;
- RAM – 16 ГБ;
- GPU – Intel Iris Integrated Graphics;
- OS – Windows 11.
- роздільна здатність – Full HD (1920x1080);
- фокусування – Авто;
- частота кадрів в секунду – 30;
- інтерфейс – USB 2.0.
- Для перевірки працездатності розробленої IoT-системи, у хмарі було створено два акаунти – адміністратора та звичайного користувача (Test user). З акаунту користувача було підключено периферійний пристрій – IP-камеру на базі ноутбуку Lenovo Yoga C940 зі встановленним ПЗ для роботи з хмарним сервісом. Пристрій вправно встановив захищенне з’єднання з сервером та почав трансляцію відеопотоку. На стороні хмари здійснювалися усі

обчислення, у тому числі знаходження і підрахунок людей, що з’являлися у полі зору камери (рисунок 3.16).

Рисунок 3.16 – Камера периферійного пристрою

Панель керування адміністратора вправно показує усю інформацію о системі, користувачах, пристроях. Журнал останніх дій працює належним чином. Налаштування безпеки на стороні сервера захищають користувачів від злому їх облікових записів, а фільтр IP-адресів не дозволяє підключатися до хмари з тих адресів, що не були додані у Whitelist. Загальні результати роботи системи наведені на нижчезазначених рисунках 3.17 – 3.23.

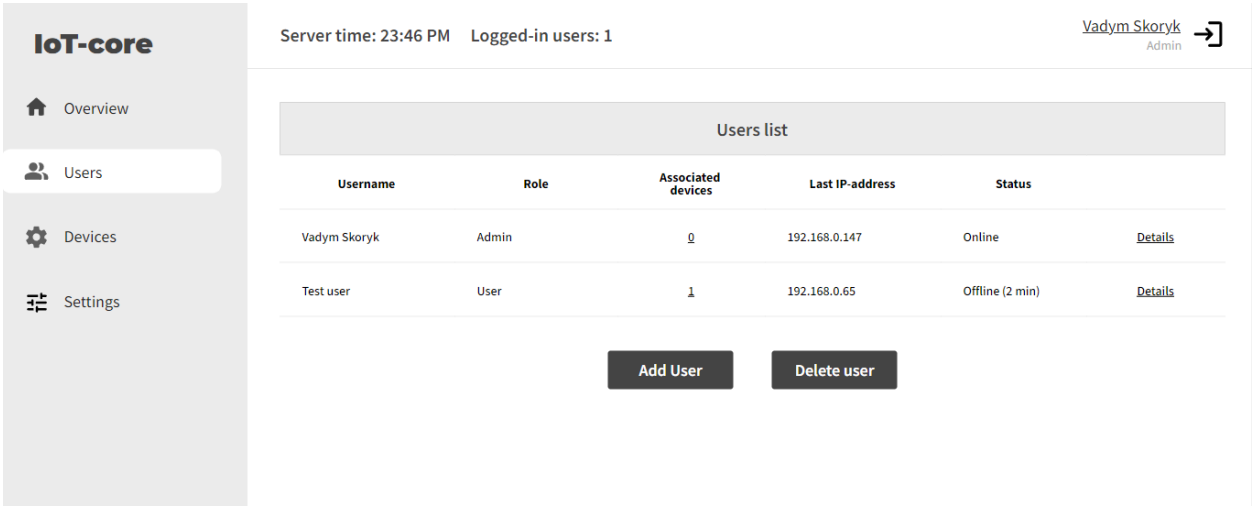

Рисунок 3.17 – Панель моніторингу адміністратора

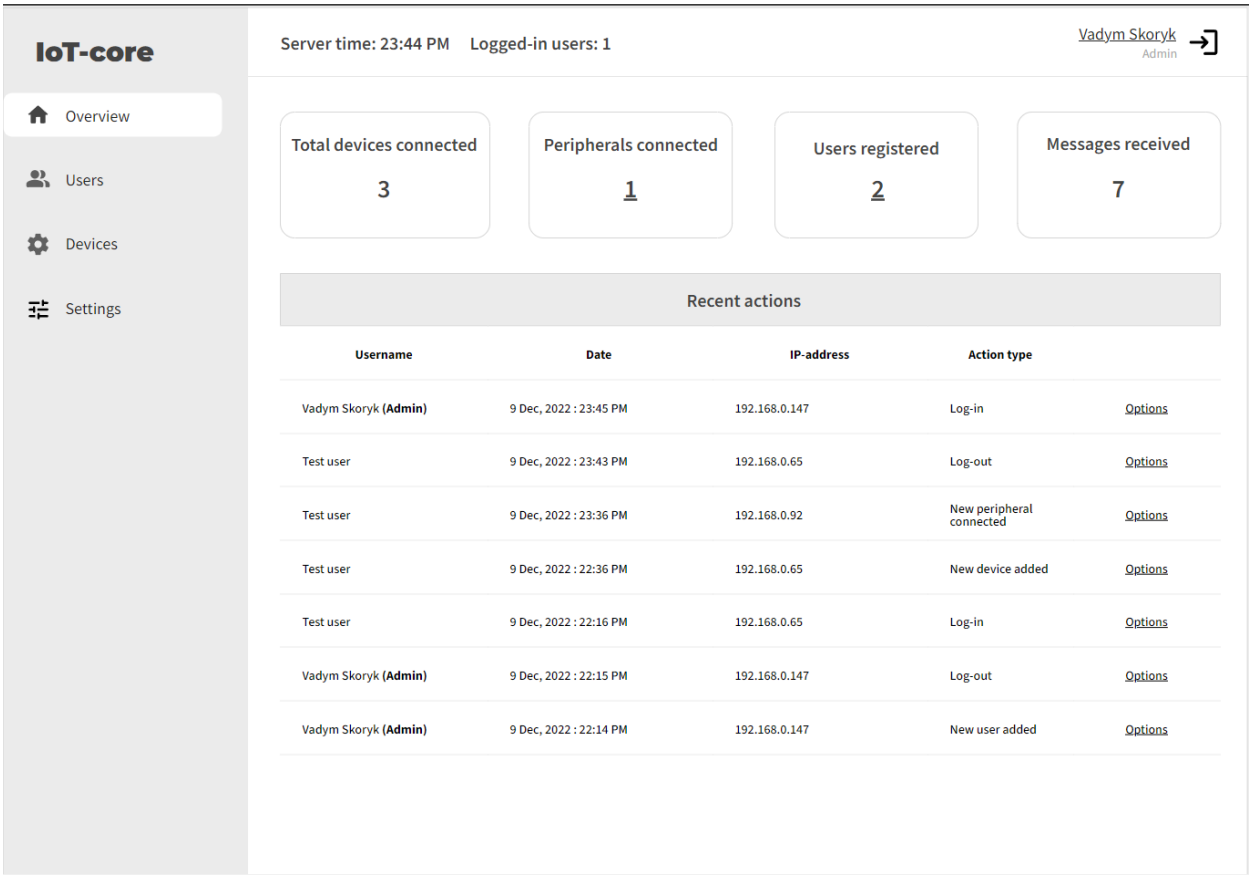

Рисунок 3.18 – Список зареєстрованих користувачів та інформація про них

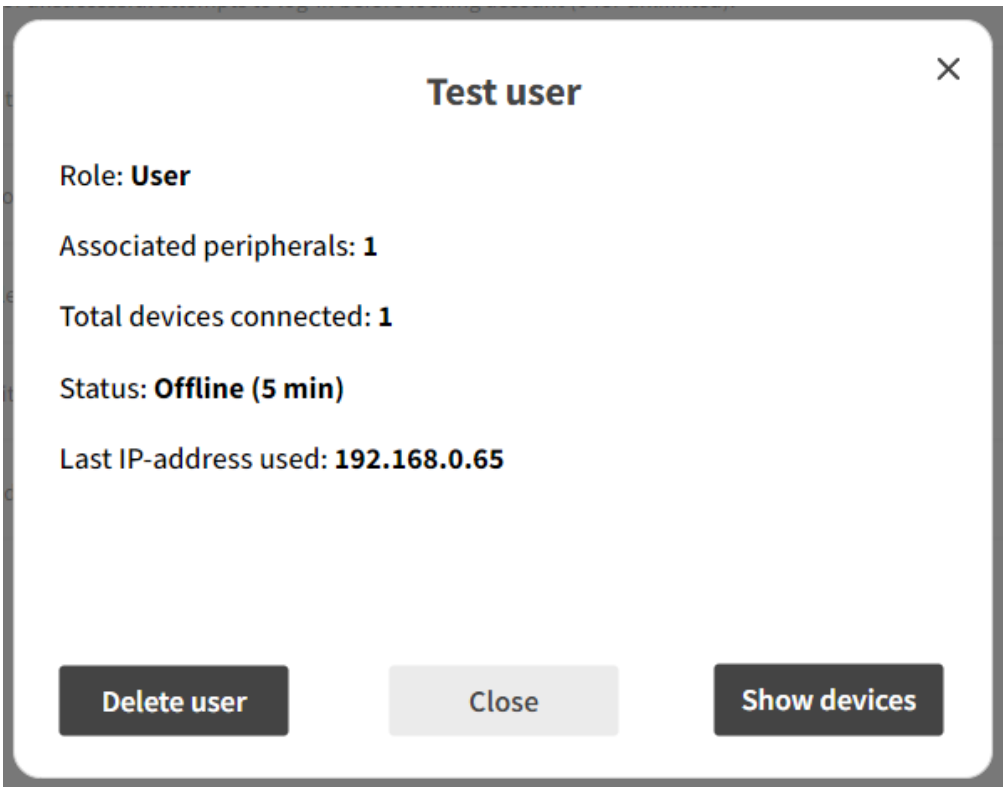

Рисунок 3.19 – Детальна інформація про користувача

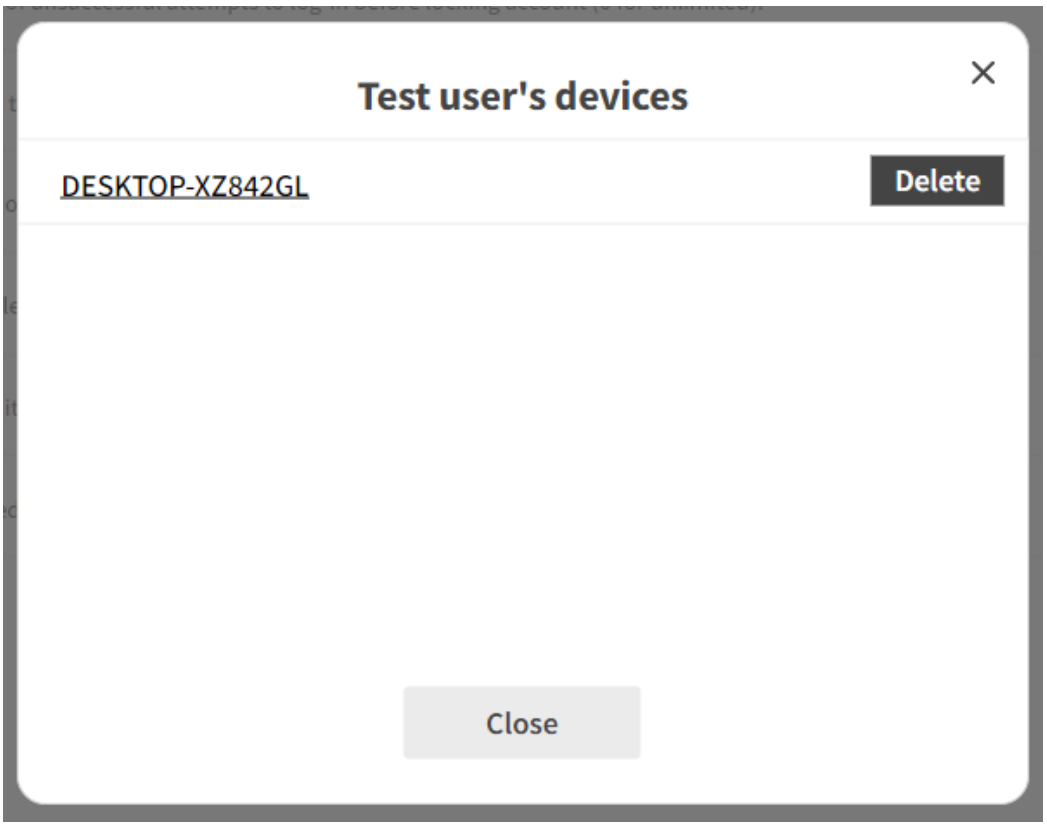

Рисунок 3.20 – Список підключених пристроїв тестового користувача

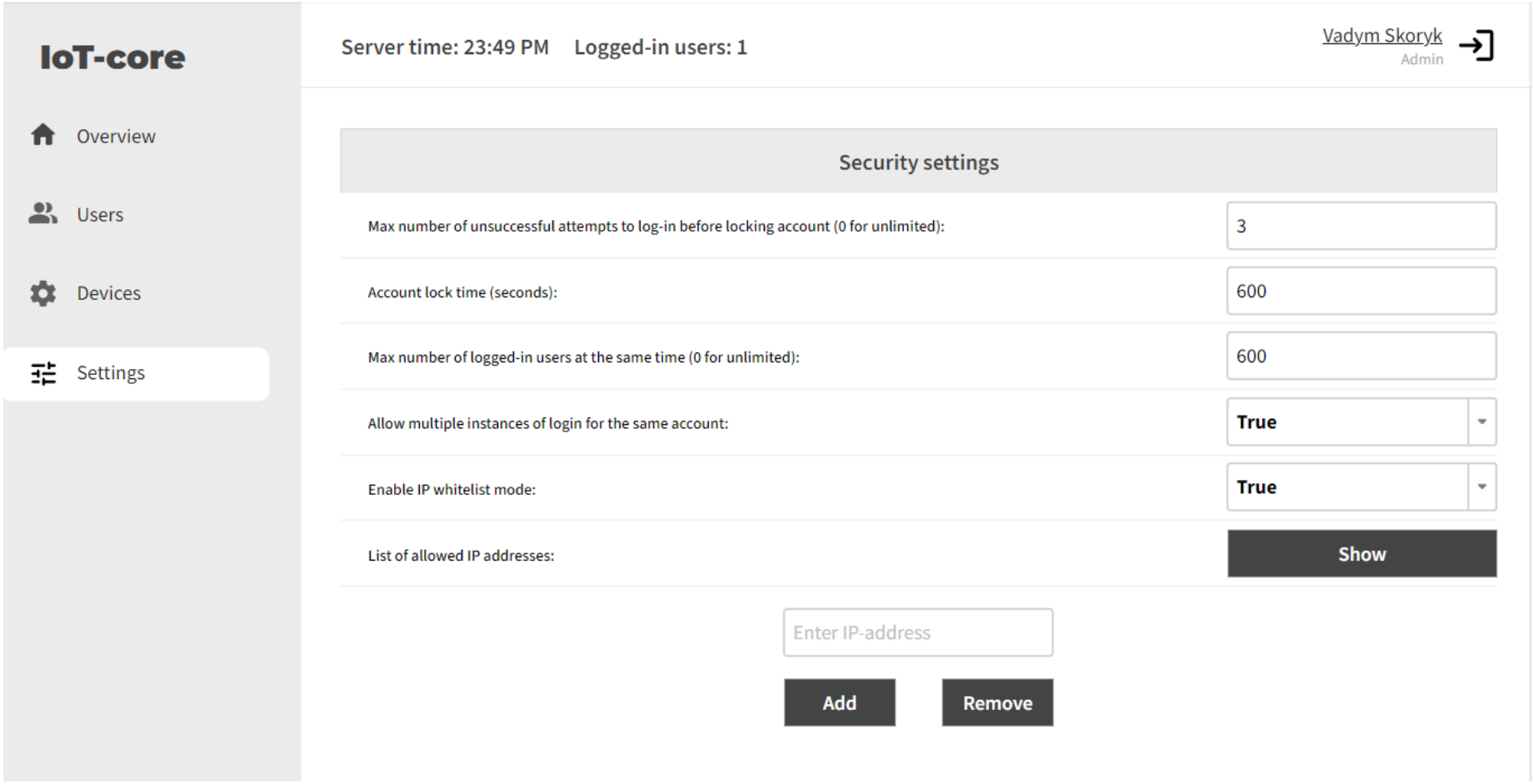

Рисунок 3.21 – Меню налаштування системи безпеки хмари

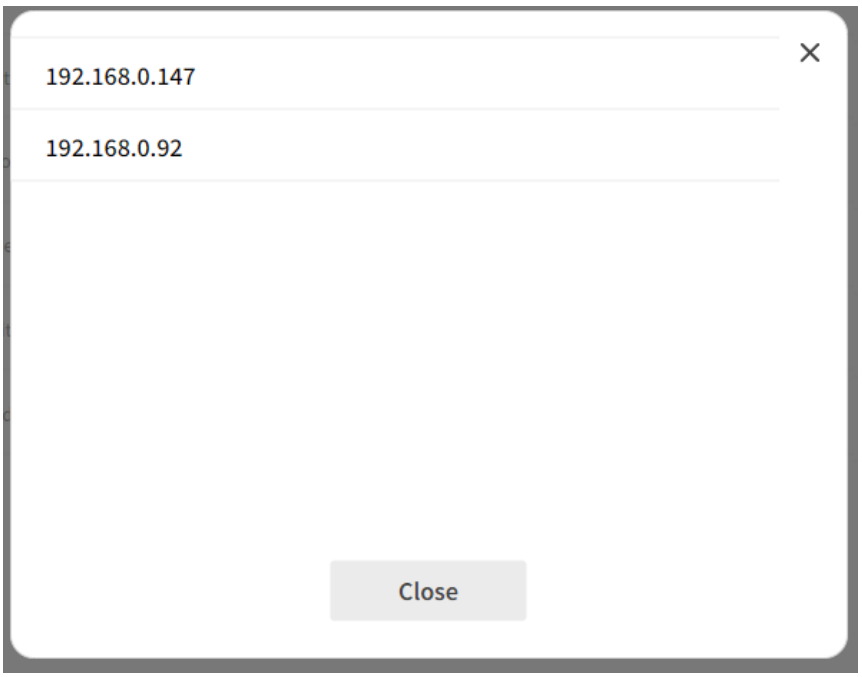

Рисунок 3.22 – Список дозволених IP-адресів

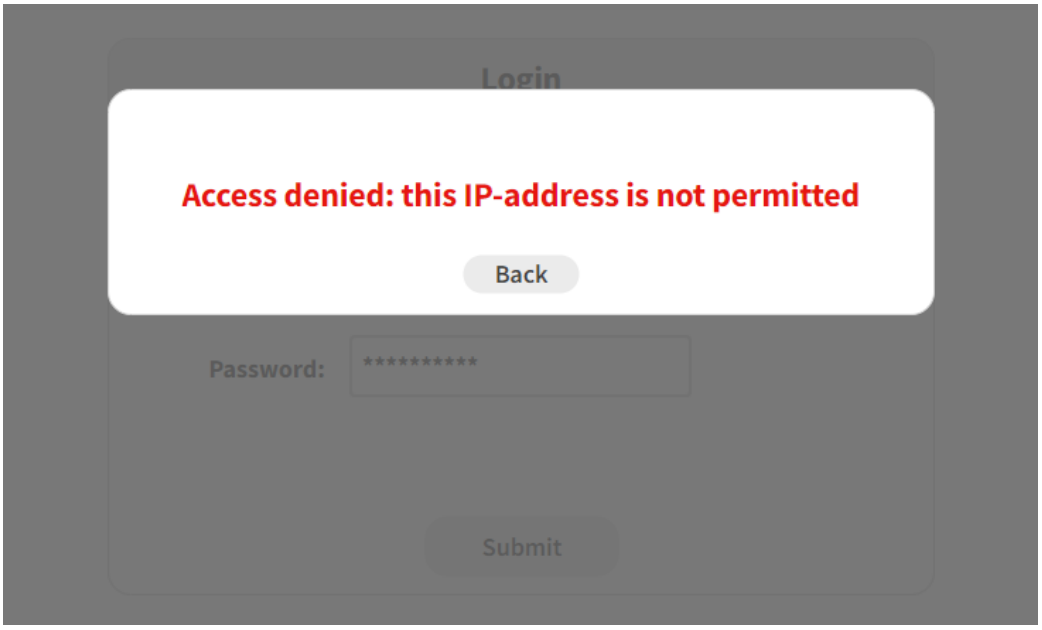

Рисунок 3.23 – Невдала спроба авторизуватися з забороненого IP-адреса

Після проведення тестувань та діагностичних експериментів можна сказати, що система працює справно, а технічне завдання було повністю виконане.

ВИСНОВКИ

Одною з найбільших проблем Інтернету речей у наш час є проблема забезпечення захисту та приватності, тому вкрай важливим є розуміння значення надійної системи безпеки IoT-рішень, а також видів загроз та атак, з якими їй доведеться боротися. В роботі були розглянуті усі сучасні та актуальні методи та підходи до побудови комплексної системи інформаційного захисту IoT-систем, досліджені типи загроз та алгоритми боротьби з ними.

За підсумками огляду та проведених досліджень була побудована модель комплексної системи забезпечення захисту, яка здатна впоратися с більшістю розповсюджених у наш час загроз. Ця модель забезпечує захист усіх елементів системи на усіх рівнях – від периферійних пристроїв до хмарного сервісу та каналі обміну інформацією. На базі запропонованої моделі була створена демонстраційна IoT-система, що складається з хмарного серверу, клієнтського веб-застосунку для контролю та моніторингу стану системи та периферійного пристрою – IP-камери.

Розроблена модель безпеки спочатку пройшла теоритечне моделювання з метою перевірки її надійності проти розповсюджених видів атак та загроз. Результати перевірки моделі на практиці співпали с результатами моделювання та підтвердили її ефективність та надійність у запобіганні шкідливого впливу на систему з боку атак та загроз.

Серед можливих подальших напрямів удосконалення та розвитку системи – додавання системи детекції втручань, розширення функціоналу додаткових налаштувань безпеки та розширення спектру типів периферійних пристроїв, що підтримуються.

За темою кваліфікаційної роботи опубліковано тези доповіді [14] в рамках десятої міжнародної науково-технічної конференції “Проблеми інформатизації” (додаток Б).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Shrikant Patel, Soumya Ranjan Jena, Mahindra University, “Internet of Things (IoT) - Theory and Applications”, International Journal of Computer Applications, vol 111, no. 7, pp. 53-64, 2015.

2. Kaiming He. IaaS, PaaS and SaaS as effective models [Текст] / Kaiming He, Georgia Gkioxari, Piotr Dollar – Tsinghua University, 2017 – 15p.

3. Boniface M., Nasser B., Papay J., Phillips S. C., Servin A., Yang X., Kousiouris G. “Platform-as-a-service architecture for real-time quality of service management in cloudsInternet and Web Applications and Services (ICIW)”, 2010 Fifth International Conference on. – IEEE, 2010. – С. 155-160.

4. Amazon Web Services - Securing Internet of Things (IoT) with AWS [Електронний ресурс] / Amazon – Режим доступа: https://aws.amazon.com/ - 21.10.2022 – Загл. з екрана.

5. Azure IoT - Quickly turn your vision into reality with secure, scalable and open edge-to-cloud solutions from the Microsoft Cloud [Електронний ресурс] / Microsoft – Режим доступа: https://azure.microsoft.com/en-gb/solutions/iot/#overview/ – Загл. з екрана.

6. Google Cloud IoT-core - Solutions to easily and securely connect, manage, and ingest data from globally dispersed devices [Електронний ресурс] / Google – Режим доступа: https://cloud.google.com/iot-core – 16.11.2022 - Загл. з екрана.

7. Pallavi Sethi, Smruti R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications", Journal of Electrical and Computer Engineering, vol. 2017, pp. 1-25, 2017.

8. Timea Bezdan. IoT Architectures and best practices [Текст] / Timea Bezdan, Nebojsa Bacanin Dzakula – Singidunum University, 2019 – 450p.

9. M.U. Farooq, Waseem M., Khairi A., Mazhar S., “A Critical Analysis onthe Security Concerns of Internet of Things (IoT)”, International Journal of

Computer Applications, vol. 111, no. 7, pp. 1-6, 2015.

10. Akram A. Moustafa. A New Approach for designing IoT system [Текст] / Akram A. Moustafa, Mohammed-Issa Riad Mousa Jaradat – Al-al-Bayt University, 2015 – 25p.

11. Muhammed Rijah, Jiffriya Mohamed Abdul Cader. A Review of IoT Architecture, Applications, Security Requirements and Mechanisms [Текст] / Muhammed Rijah, Jiffriya Mohamed Abdul Cader - Sri Lanka German Training Institute, 2021 – 27p.

12. Arif Ulah, Imane Laassar, Canan Batur Sahin, Ozlem Batur Dinler, “Cloud and internet-of-things secure integration along with security concerns”, International Journal of Informatics and Communication Technology, vol 12, no. 1, pp. 62-71, 2022.

13. Rajarshi Roy Chowdhury, Pg Emeroylariffion Abas, “Device identification using optimized digital footprints”, IAES International Journal of Artificial Intelligence, vol 50, no. 4, pp. 342-370, 2021.

14. Скорик В.А. Методи забезпечення інформаційного захисту IoT-систем [Текст] / Піскарьов О.М., Скорик В.А. // ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ. Тези доповідей десятої міжнародної науково-технічної конференції (24 – 25 листопада 2022 року), Том 2. – 2022. – С. 117